D)
*
(A
+
B)
M
[X
]
I T
(O
I S

CISC Characteristics

A computer with large number of instructions is called complex instruction set computer or CISC. Complex instruction set computer is mostly used in scientific computing applications requiring lots of floating point arithmetic.

1. A large number of instructions - typically from 100 to 250 instructions.

2.Some instructions that perform specialized tasks and are used infrequently.

3.A large variety of addressing modes - typically 5 to 20 different modes.

4.Variable-length instruction formats

5.Instructions that manipulate operands in memory.

RISC Characteristics

A computer with few instructions and simple construction is called reduced instruction set computer or RISC. RISC architecture is simple and efficient. The major characteristics of RISC architecture are,

1. *Relatively few instructions*
2. *Relatively few addressing modes*
3. *Memory access limited to load and store instructions*
4. *All operations are done within the registers of the CPU*
5. *Fixed-length and easily-decoded instruction format.*
6. *Single cycle instruction execution*
7. *Hardwired and micro programmed control*

3.2.5. Basic Machine Instructions IA-32 Pentium example.

Pentium Addressing Modes

Virtual or effective address is offset into segment

—Starting address plus offset gives linear address

—This goes through page translation if paging is enabled

9 addressing modes available

—Immediate

—Register operand

—Displacement : offset in a segment

—Base : same as register indirect addressing

—Base with displacement

—Scaled index with displacement : scale factor is used

– scale factor of 2 can be used to index an

array of 16-bit integers

—Base with index and displacement

—Base scaled index with displacement

—Relative

– used in transfer-of-control instructions

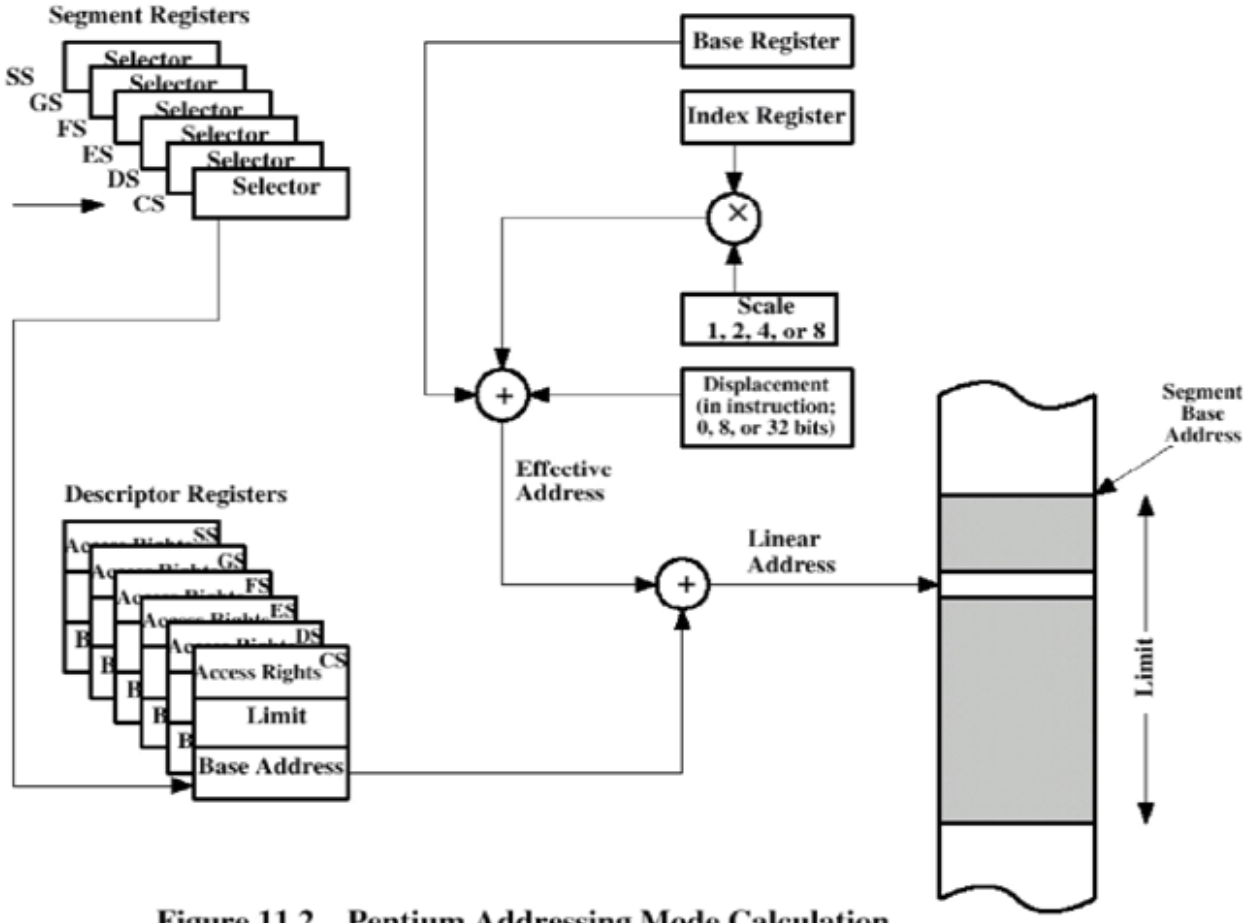


Figure 11.2 Pentium Addressing Mode Calculation

Mode	Algorithm
Immediate	Operand = A
Register Operand	LA = R
Displacement	LA = (SR) + A
Base	LA = (SR) + (B)
Base with Displacement	LA = (SR) + (B) + A
Scaled Index with Displacement	LA = (SR) + (I) × S + A
Base with Index and Displacement	LA = (SR) + (B) + (I) + A
Base with Scaled Index and Displacement	LA = (SR) + (I) × S + (B) + A
Relative	LA = (PC) + A

LA	=	linear address
(X)	=	contents of X
SR	=	segment register
PC	=	program counter
A	=	contents of an address field in the instruction
R	=	register
B	=	base register
I	=	index register
S	=	scaling factor

Pentium Instruction Formats

• Instruction consists of

0-4 optional prefixes

1-2 byte opcode

Optional address specifier

Consists of ModR/m byte

and Scale Index byte

Optional displacement

Optional immediate field

Pentium Instruction

Formats

Prefix bytes Instruction prefixes

LOCK or one of repeat prefixes

LOCK is used to ensure exclusive use of shared memory in multiprocessor environments

REP, REPE, REPZ, REPNE, and REPNZ Specify repeated operation on strings

Repeat until counter in CX goes zero or until the condition is met Segment override

Address size

Switches between 32-bit and 16-bit address generation Operand size

Switches between 32-bit and 16-bit operands

Pentium Instruction Formats

Instruction

1. ***Opcode***
2. ModR/m

Specify whether an operand is in a register or in memory

3. SIB

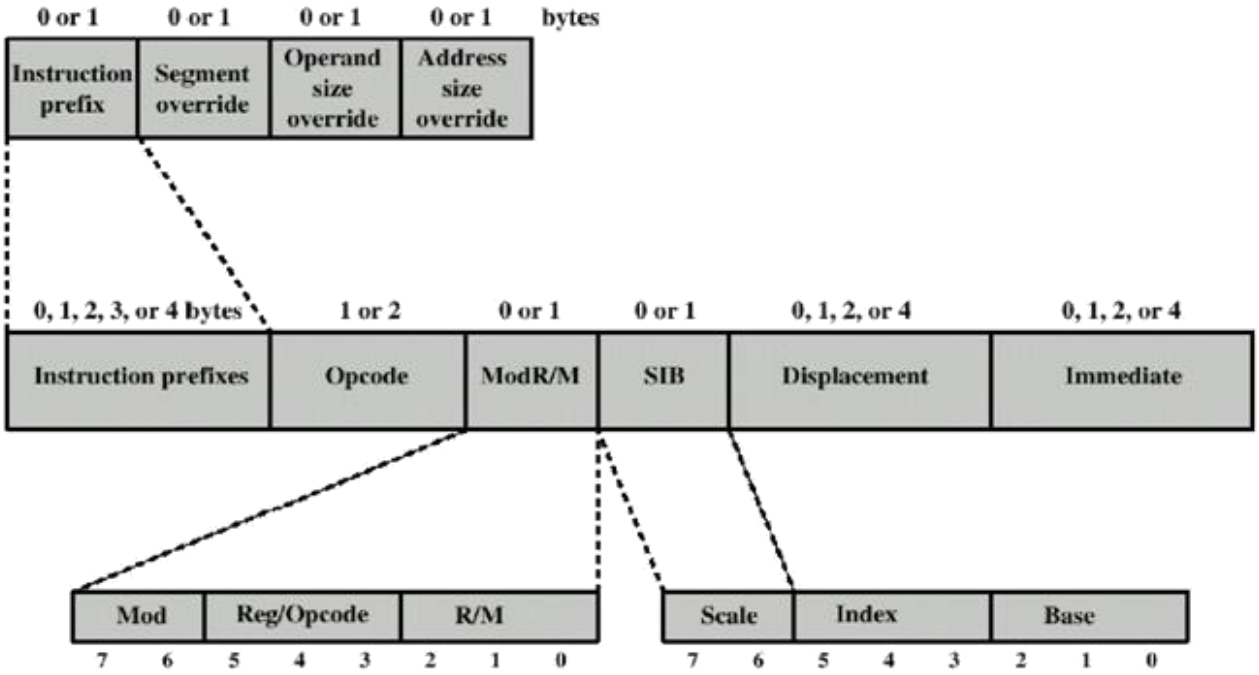
Specify fully the addressing mode

4.Displacement

– When used, 8-, 16-, or 32-bit displacement field is added 5.Immediate

When used, 8-, 16-, or 32-bit operand is provided

Pentium Instruction Formats


0 or 1
0 or 1
0 or 1
0 or 1
bytes
Instruction prefix
Segment override
Operand size override
Address size override
0, 1, 2, 3, or 4 bytes
1 or 2
0 or 1
0 or 1
0, 1, 2, or 4
0, 1, 2, or 4
Instruction prefixes
Opcode
ModR/M
SIB
Displacement
Immediate
Mod
Reg/Opcode
R/M
Scale
Index
Base
7 6 5 4 3 2 1 0
7 6 5 4 3 2 1 0

UNIT-4

Memory organization

4.1Concept Of Memory,

4.2RAM,ROM Memories

4.3Memory Hierarchy 4.4Cache

4.5Memories 4.6Virtual Memory,

4.7Secondary Storage,

Memory Management Requirements.

4.1 MEMORY HIERARCHY

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 of 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 called 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 tall 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** 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 in 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

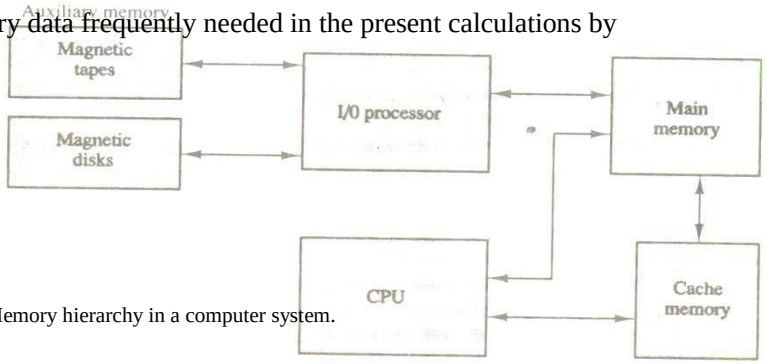


Figure - Memory hierarchy in a computer system.

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.

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 100ns, while main memory access time may be 700ns. Auxiliary memory average access time is usually 1000 times that of main memory. Block size in auxiliary memory typically ranges from256 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 indifferent 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.

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 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 charges on the capacitors tend 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 shorted 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 one 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 from 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

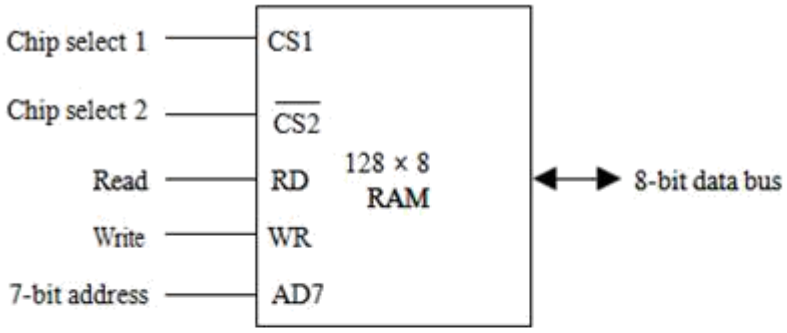
chip interconnection, we will show an example of a 1024 × 8 memory constructed with 128 × 8 RAM chips and 512 × 8 ROM chips.

4.2RAM 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. The capacity of the memory is 128 words of eight bits (one byte) per word. This requires a 7-bit.

Figure- Typical RAM Chip.



(a) Block diagram

CSI	$\overline{CS2}$	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

Address and an 8-bit bidirectional data bus. The read and write inputs specify 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 or read or write.

The function table listed in Fig. (b) Specifies the operation of the RAM chip. The unitis in operation only when CSI = 1 and CS2 = 0. The bar on top of the second select variable indicates that this input in enabled when it is equal to 0. If the chip select inputs are not enabled, or if they are enabled but the read but the read or write inputs are not enabled, the memory is inhibited and its data bus is in a high-impedance state. When SC1 = 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 below Fig. For the same-size chip, it is possible to have more bits of 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. Thetwo 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.

4.2.1.1. 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 form 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

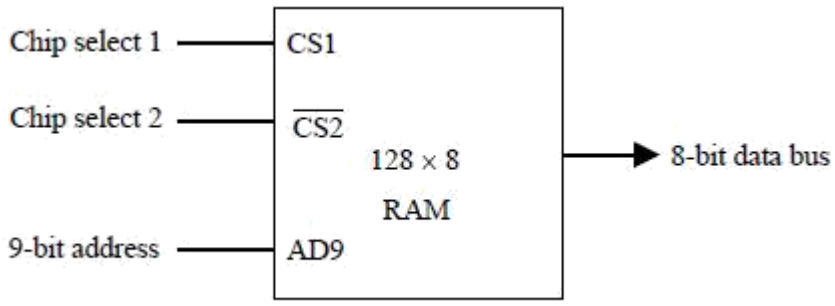


Figure-Typical ROM chip.

To be used are specified in Fig Typical RAM chip and Typical ROM chip. The memory address map for this configuration is shown in Table 7-1. 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 from 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 forms the information under the address bus assignment. The address bus lines are subdivided into groups of four bits each so

TABLE-Memory Address Map for Microprocomputer

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
RAM 2	0080—00FF	0 0	1 x x x	x x x x
RAM 3	0100—017F	0 1	0 x x x	x x x x
RAM 4	0180—01FF	0 1	1 x x x	x x x x
ROM	0200—03FF	1 x	x x x x	x x x x

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. This x's represent a binary number that can range from an all-0's to an all-1's value.

4.2 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 belowFig. This configuration gives a memory capacity of 512 bytes of RAM and 512 bytes of ROM. It implements the memory map of Table 7-1. 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 SCI input 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.

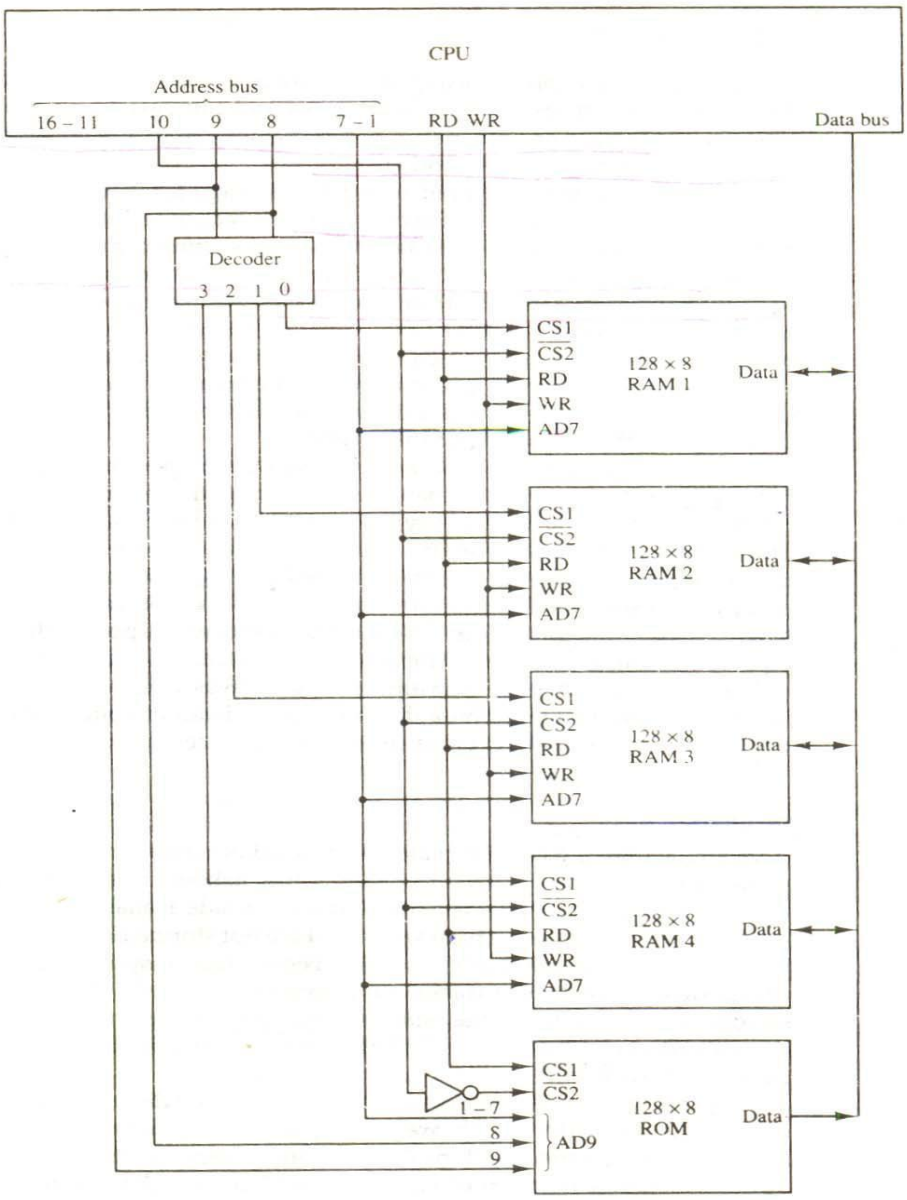


Figure -Memory connection to the CPU.

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 locaters 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 then 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.

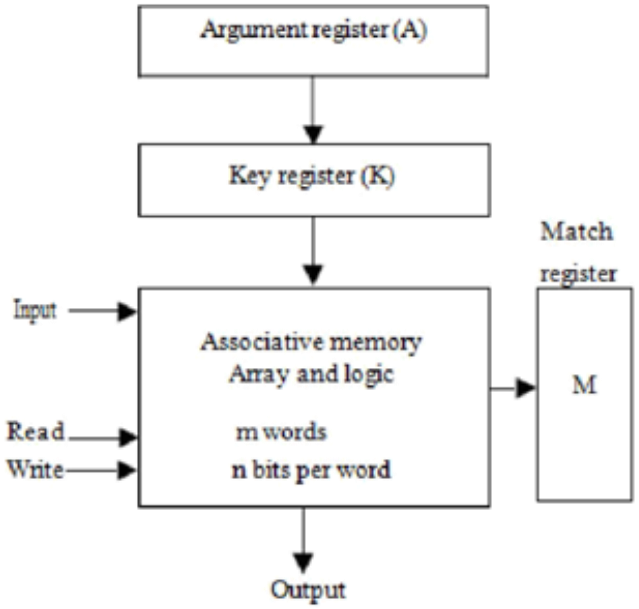
HARDWARE ORGANIZATION

The block diagram of an associative memory is shown in below Fig. It consists of a memory array and logic from 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. Thus the key provides a mask or identifying piece of information which specifies how the reference to memory is made.

Figure- Block diagram of associative memory



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.

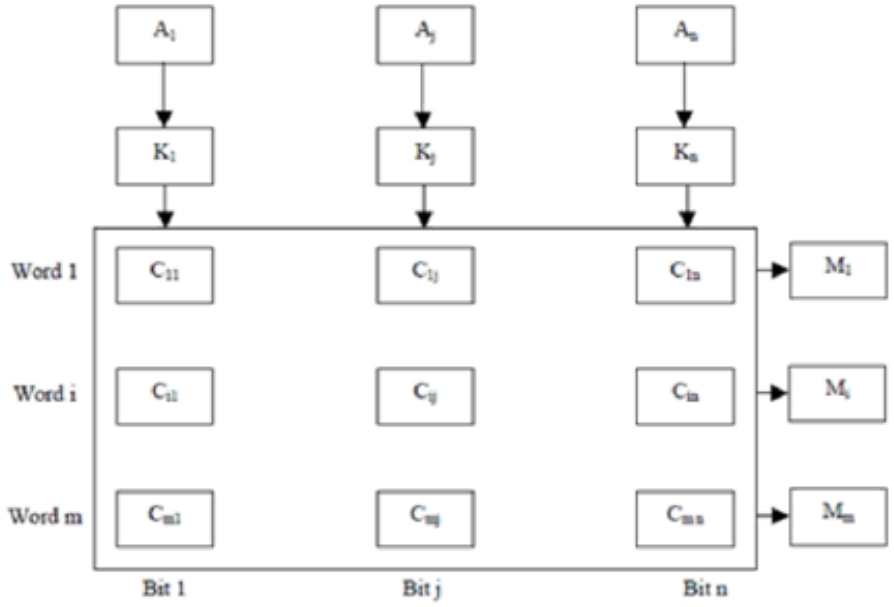
		1	
		1	
		1	
	1	1	
	0	0	
A	1	0	
		0	
		0	
		0	
		0	
	1	0	
	1	0	
K	1	0	
			no
Word 1		1	
		1	match
		1	
	1	1	
	0	0	
	0	0	
Word 2		0	
		0	match
		0	
	1	0	
	0	0	
	1	1	

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 below Fig. 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,…,n. If a matchoccurs 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.

Figure -Associative memory of m word, n cells per word



The internal organization of a typical cell C_{ij} is shown in Fig.. It consists of a flip-

Flop storage element F_{ij} and the circuits for reading, writing, and matching the cell. The input bit is transferred into the storage cell during a write operation. The bit stored is read out during a read operation. The match logic compares the content of the storage cell with the corresponding unmasked bit of the argument and provides an output for the decision logic that sets the bit in M_i.

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,…, n. Two bits are equal if they are both 1 or both 0. The equality of two bits can be expressed logically by the Boolean function $x_j = A_jF_{ij} + A'_jF'_{ij}$

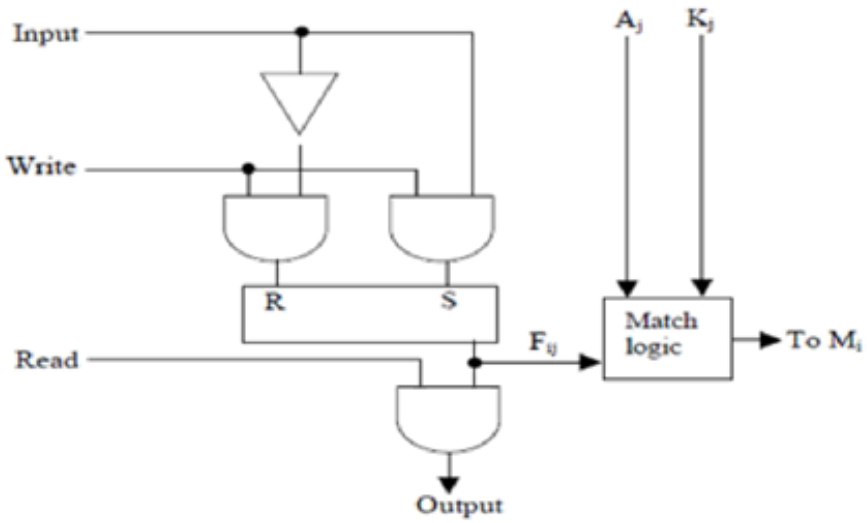
Where$x_j = 1$ if the pair of bits in position j are equal; otherwise, $x_j = 0$.

For a word i to be equal to the argument in a we must have all x_j variables equal to 1. This is the condition for setting the corresponding match bit M_i to 1. The Boolean function for this condition is

$M_i = x_1 x_2 x_3 \dots x_n$

And constitutes the AND operation of all pairs of matched bits in a word. **Figure**One cell of associative memory.

Figure- One cell of associative memory.



We now include the key bit K_j in the comparison logic. The requirement is that if $K_j = 0$, the corresponding bits of A_j and F_{ij} need no comparison. Only when $K_j = 1$ must they be compared. This requirement is achieved by ORing each term with K_j' , thus:

$$x_j + K_j' = 1 \quad \text{if } K_j = 1$$

When $K_j = 1$, we have $K_j' = 0$ and $x_j + 0 = x_j$. When $K_j = 0$, then $K_j' = 1$ $x_j + 1 = 1$. A term $(x_j + K_j')$ will be in the 1 state if its pair of bitsis not compared. This is necessary because each term isANDed with all other terms so that an output of 1 will have no effect. The comparison of the bits has an effect only when $K_j = 1$.

The match logic for word i in an associative memory can now be expressed by the following Boolean function:

$$M_i = (x_1 + K_j')(x_2 + K_j')(x_3 + K_j') \ldots (x_n + K_j')$$

Each term in the expression will be equal to 1 if its corresponding $K'_j = 0$. if$K_j = 1$, the term willbe either 0 or 1 depending on the value of x_j. A match will occur and M_i will be equal to 1 if all terms are equal to 1.

If we substitute the original definition of x_j. The Boolean function above can be expressed as follows:

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

Where ∏ 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, …., m.

The circuit for catching one word is shown in below Fig. 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 catch occurs and 0 if no match occurs.

Note that if the key register contains all 0's, output M_i will be a 1 irrespective of the value of A or the word. This occurrence must be avoided during normal operation.

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 catch register. It is then necessary to scan the bits of the match register on eat 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.

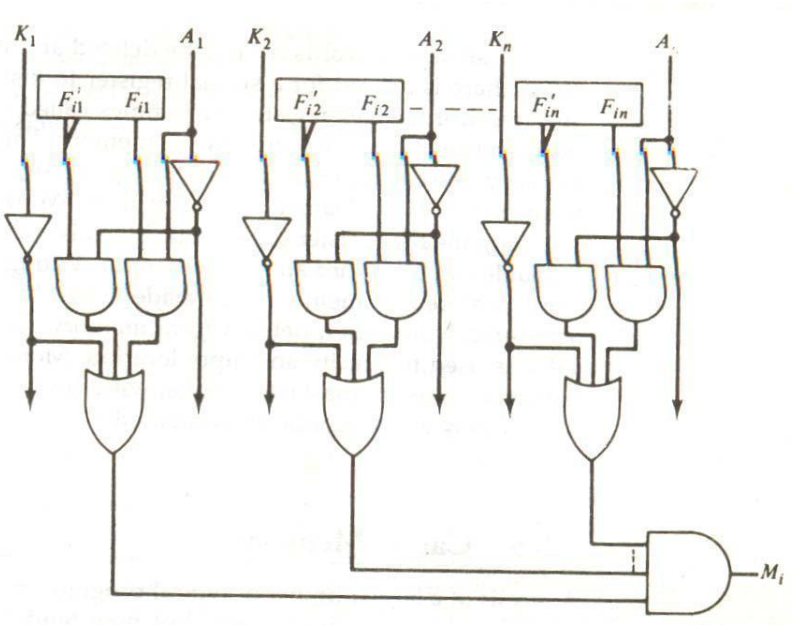


Figure -Match logic for one word of associative memory

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 zero content, an all-zero output will indicate that no match occurred and that the searched item is not available in memory.

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_j bits) with the argument word so that only active words are compared.

4.5 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. The phenomenon is known as the property of 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 frequently.

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 theCPU and main memory as illustrated in below Fig. 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 helping the discussion of these three mapping procedures we will use a specific example of a memory organization as shown in below Fig. 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.

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.

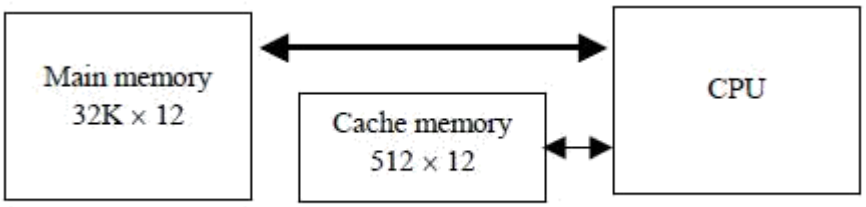
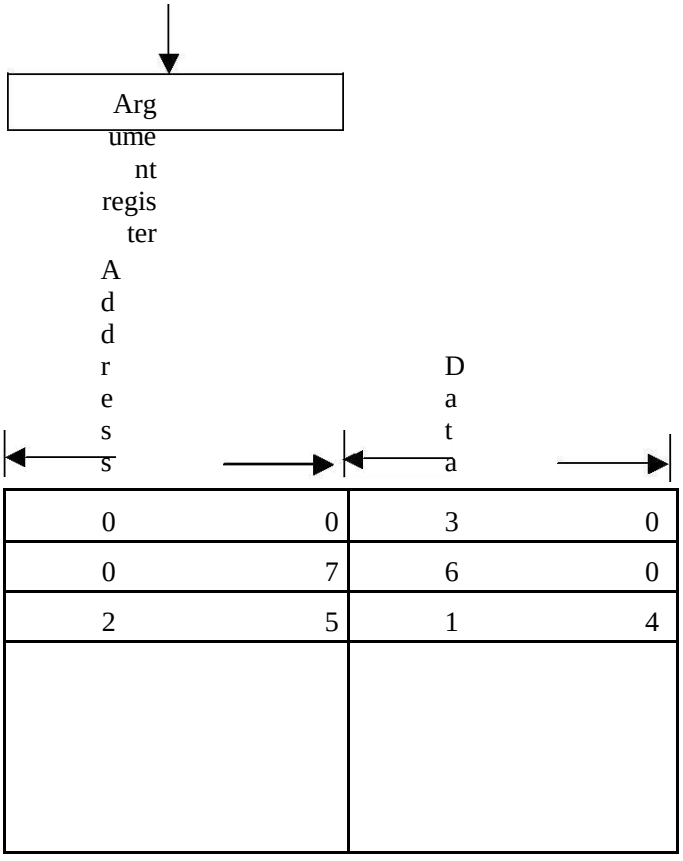


Figure - Example of cache memory

ASSOCIATIVE MAPPING

The fasters and most flexible cache organization use an associative memory. This organization is illustrated in below Fig. 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. If the address is found, the corresponding 12-bit data is read

Figure-Associative mapping cache (all numbers in octal) CPU address (15 bits)



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.

DIRECT MAPPING

Associative memories are expensive compared to random-access memories because of the added logic associated with each cell. The possibility of using a random-access memory for the cache is investigated in Fig. The CPU address of 15 bits is divided into two fields. Thenine least significant bits constitute the index field and the remaining six bits from 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. (b). 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 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 droop 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. 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 sued 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.

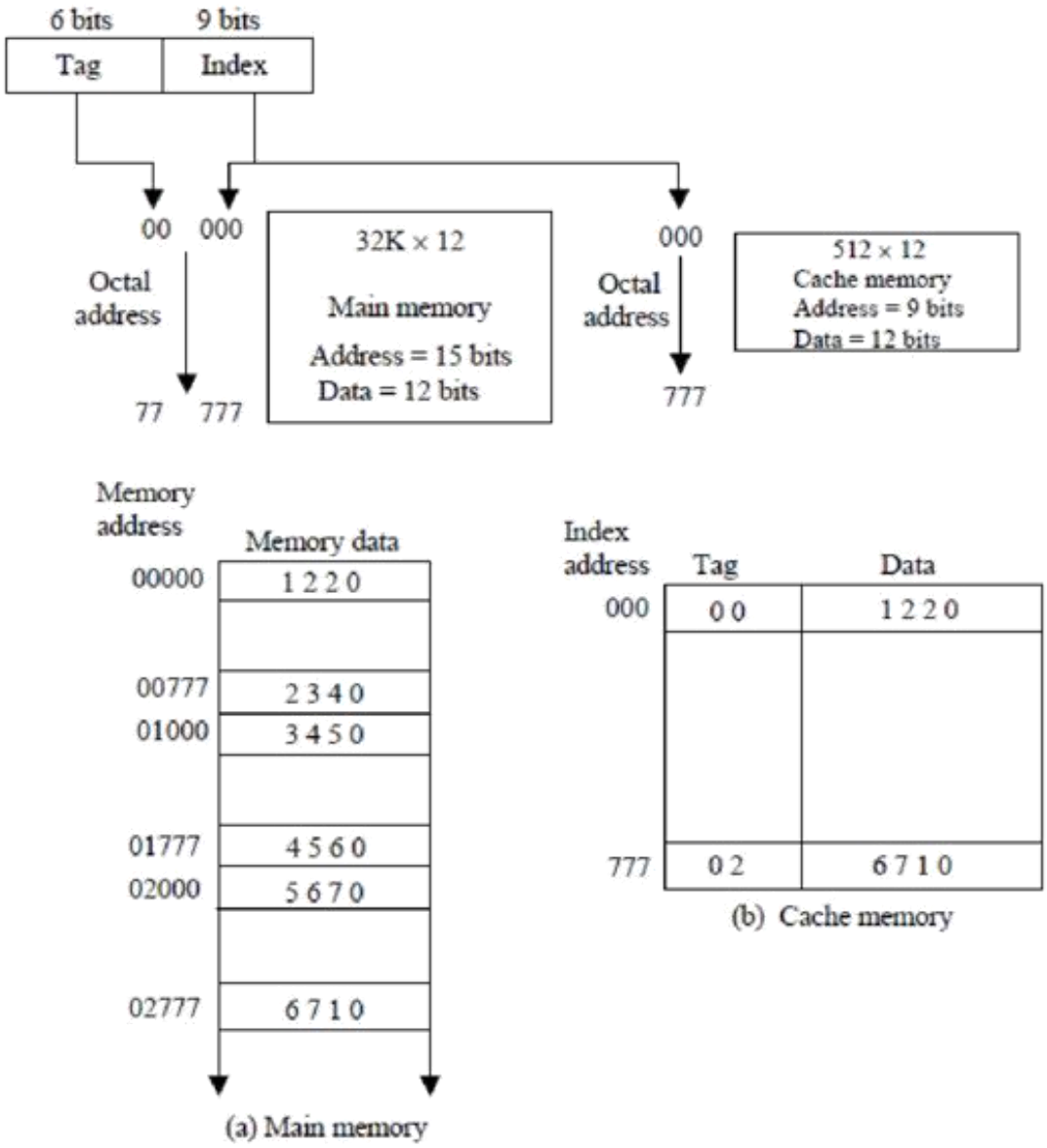


Fig-Direct mapping cache organization

The direct-mapping example just described uses a block size of one word. The same

organization but using a block size of 8 words is shown in below Fig. The index