Computer Arithmetic

Arithmetic instructions in digital computers manipulate data to produce results necessary for the solution of computational problems. An arithmetic processor is the part of a processor unit that executes arithmetic operations. The data type assumed to reside in processor registers during the execution of an arithmetic instruction is specified in the definition of the instruction. The solution to any problem that is stated by a finite number of well-defined procedural steps is called an algorithm.

Addition and Subtraction with Signed-Magnitude Data: We designate the magnitude of the two numbers by A and B. When the signed numbers are added or subtracted, we find that there are eight different conditions to consider, depending on the sign of the numbers and the operation performed. These conditions are listed in the first column of Table 18. The other columns in the table show the actual operation to be performed with the magnitude of the numbers. The last column is needed to prevent a negative zero. In other words, when two equal numbers are subtracted, the result should be +0 not -0.

TABLE 18 Addition and Subtraction of Signed-Magnitude Numbers

Operation	Add Magnitudes	Subtract Magnitudes		
		When $A > B$	When $A < B$	When $A = B$
$(+A) + (+B)$	$+(A + B)$			
$(+A) + (-B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(-A) + (+B)$		$-(A - B)$	$+(B - A)$	$+(A - B)$
$(-A) + (-B)$	$-(A + B)$			
$(+A) - (+B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(+A) - (-B)$	$+(A + B)$			
$(-A) - (+B)$	$-(A + B)$			
$(-A) - (-B)$		$-(A - B)$	$+(B - A)$	$+(A - B)$

Hardware Implementation: Let A and B be two registers that hold the magnitudes of the numbers, and As and Bs be two flip-flops that hold the corresponding signs. Consider now the hardware implementation of the algorithms above:

1- First, a parallel-adder is needed to perform the microoperation A + B.
2- Second, a comparator circuit is needed to establish if A > B, A = B, or A < B.
3- Third, two parallel-subtractor circuits are needed to perform the microoperations (A-B) and (B-A).
4- The sign relationship can be determined from an exclusive- OR gate with A_s and B_s as inputs.

Careful investigation of the alternatives reveals that the use of 2's complement for subtraction and comparison is an efficient procedure that requires only an adder and a complementer. Figure 51 shows a block diagram of the hardware for implementing the addition and subtraction operations. It consists of registers A and B and sign flip-flops A_s and B_s. Subtraction is done by adding A to the 2's complement of B. The output carry is transferred to flip-flop E, where it can be checked to determine the relative magnitudes of the two numbers. The add-overflow flip-flop AVF holds the overflow bit when A and B are added.

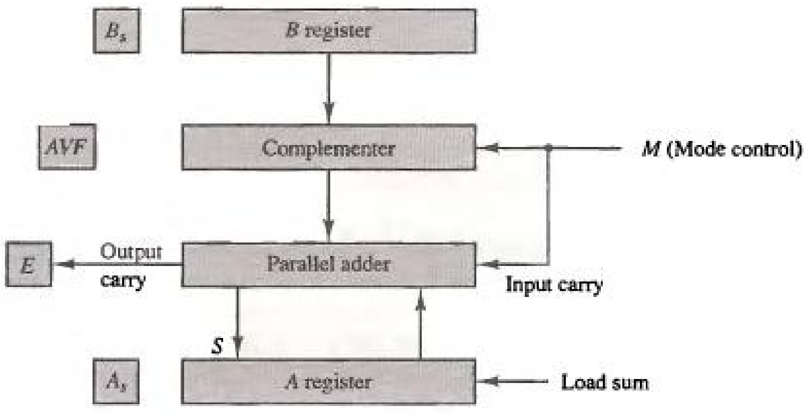


Figure 51 Hardware for signed-magnitude addition and subtraction.

The adder is equal to the sum A + B. When M = 1, the l's complement of B is applied to the adder, the input carry is 1, and output S = A + B +1. This is equal to A plus the 2's complement of B, which is equivalent to the subtraction A - B. The signed 2's complement representation of numbers together with arithmetic algorithms for addition and subtraction are introduced as: The leftmost bit of a binary number represents the sign bit: 0 for positive and 1 for negative. If the sign bit is 1, the entire number is represented in 2's complement form. Thus +33 is represented as 00100001 and -33 as 11011111. Note that 11011111 is the 2's complement of 00100001, and vice versa. *The addition of two numbers in signed 2's complement form consists of adding the numbers with the sign bits treated the same as the other bits of the number.* A carry-out of the sign-bit position is discarded. The subtraction consists of first taking the 2's complement of the subtrahend and then adding it to the minuend.

Multiplication Algorithms

Multiplication of two fixed-point binary numbers in signed-magnitude representation is done with paper and pencil by a process of successive shift and add operations. This process is best illustrated with a numerical example:

```
 23      10111   Multiplicand
 19    × 10011   Multiplier
         10111
        10111
       00000   +
      00000
     10111
437  110110101   Product
```

Figure 52 is a flowchart of the hardware multiply algorithm. Initially, the multiplicand is in B and the multiplier in Q. Their corresponding signs are in B_s and Q_s, respectively. ***The signs are compared, and both A and Q are set to correspond to the sign of the product*** since a double-length product will be stored in registers A and Q. Registers A and E are cleared and the sequence counter SC is set to a number equal to the number of bits of the multiplier.

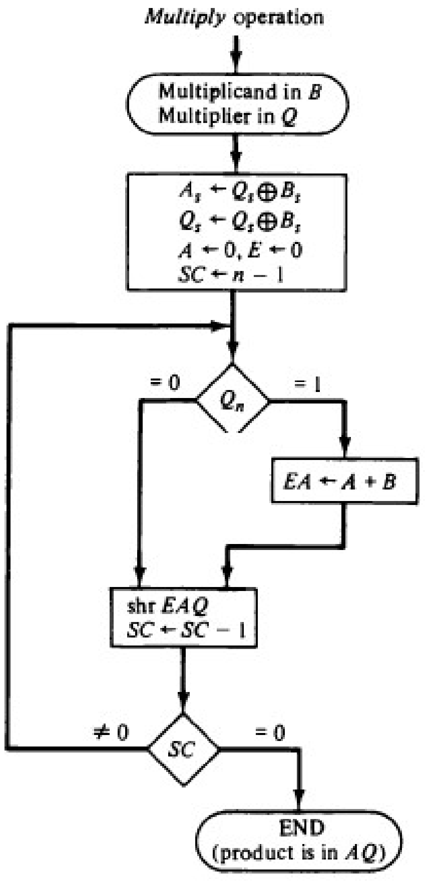


Figure 52 Flowchart for multiply operation.

The numerical example is repeated to clarify the hardware multiplication process. It operates on the fact that strings of 0's in the multiplier require no addition but just shifting, while string of 1's in the multiplier require addition with shifting. The table 19 illustrate numerical example for multiplier 23 (which in binary equal 10111) by 19 (which binary equal 10011) gives the result 437(in binary equal 0110110101).

TABLE 19 Numerical Example for Binary Multiplier

Multiplicand B = 10111	E	A	Q	SC
Multiplier in Q	0	00000	10011	101
Q_n = 1; add B		10111		
First partial product	0	10111		
Shift right EAQ	0	01011	11001	100
Q_n = 1; add B		10111		
Second partial product	1	00010		
Shift right EAQ	0	10001	01100	011
Q_n = 0; shift right EAQ	0	01000	10110	010
Q_n = 0; shift right EAQ	0	00100	01011	001
Q_n = 1; add B		10111		
Fifth partial product	0	11011		
Shift right EAQ	0	01101	10101	000
Final product in AQ = 0110110101				

Memory Hierarchy

The memory unit is an essential component in any digital computer since it is needed for storing programs and data. The memory unit that communicates directly with the CPU is called the main memory. Devices that provide backup storage are called auxiliary memory. 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.

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. Fig(29) shows the Memory Hierarchy:

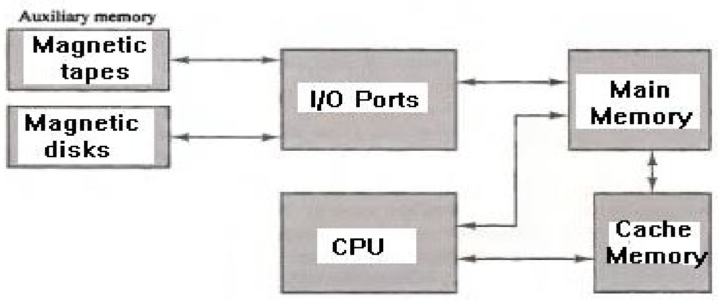


Figure 29 Memory hierarchy in a computer system.

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:

The static RAM consists essentially of internal flip-flops that store the binary information.

The dynamic RAM stores the binary information in the form of electric charges that are applied to capacitors.

<u>Associative Memory</u>

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.

A memory unit accessed by content is called an associative memory or content addressable memory (CAM). When a word is written in an associative 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.

The block diagram of an associative memory is shown in Fig(30):

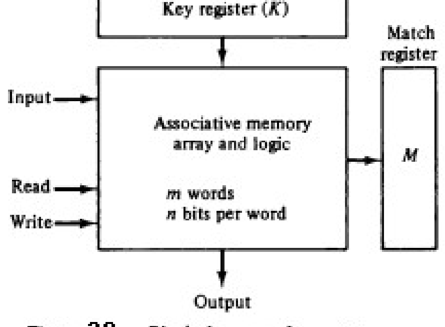
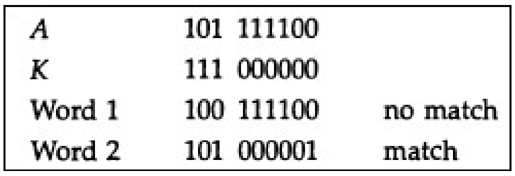


Figure 30 Block diagram of associative memory.

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 left most bits of A are compared with memory words because K has l'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.

Cache Memory

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.

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. 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*.

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

Virtual Memory

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. 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.

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.

The mapping table may be stored in a separate memory as shown in Fig(31) or in main memory. In the first case, an additional memory unit is required as well as one extra memory access time. In the second case, the table takes space from main memory and two accesses to memory are required with the program running at half speed.

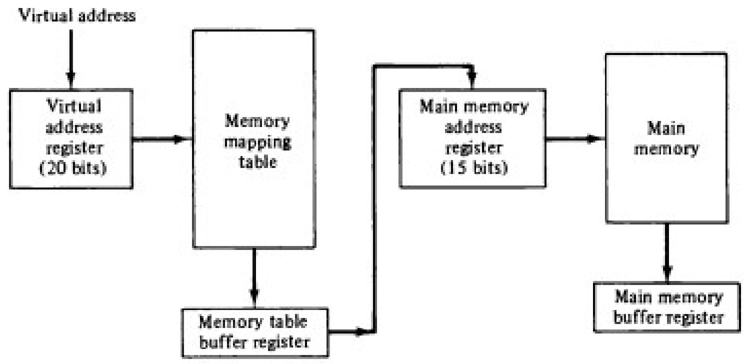

Virtual address
Virtual address register (20 bits)
Memory mapping table
Memory table buffer register
Main memory address register (15 bits)
Main memory
Main memory buffer register

Figure 31 Memory table for mapping a virtual address.

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 pages and blocks 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 IK words, then, using the previous example, address space is divided into 1024 pages and main memory is divided into 32 blocks.

The organization of the memory mapping table in a paged system is shown in Fig(32). 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 0, 1, 2, and 3, 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.

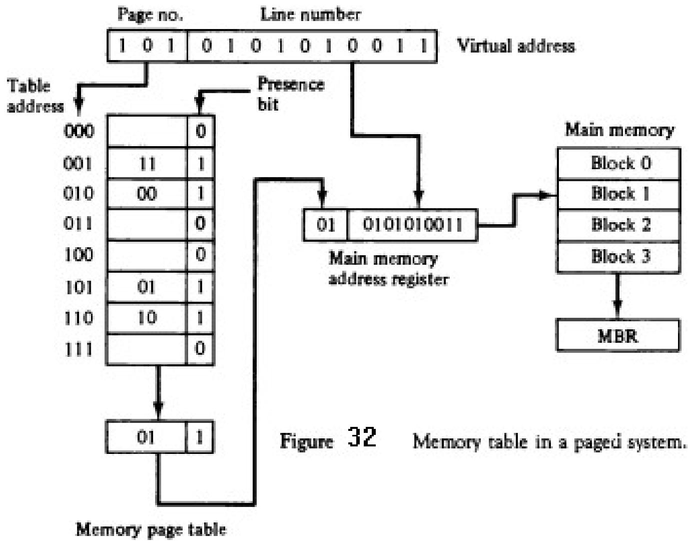


Figure 32 Memory table in a paged system.

Memory Management Hardware

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.

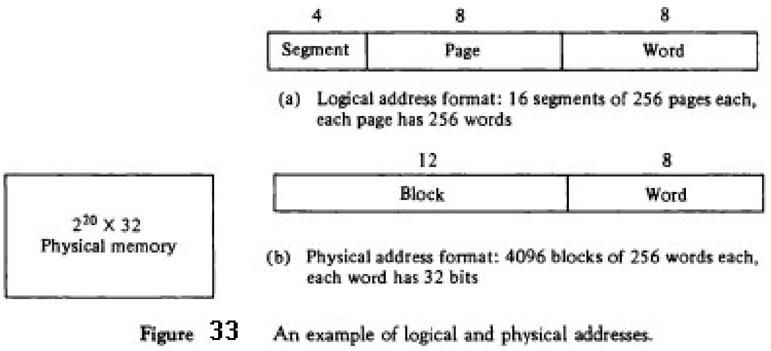
The basic components of a memory management unit are:

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 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 segment 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 address generated by a segmented program is called a logical address. 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.

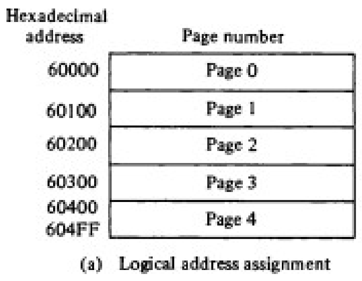
Numerical Example: A numerical example may clarify the operation of the memory management unit. Consider the 20-bit logical address specified in Fig(33-a).This configuration allows each segment to have any number of pages up to 256. The smallest possible segment will have one page or 256 words. The largest possible segment will have 256 pages, for a total of 256 x 256 = 64K words. The physical memory shown in Fig(33-b).



4
8
8
Segment
Page
Word
(a) Logical address format: 16 segments of 256 pages each, each page has 256 words
12
8
Block
Word
2^{20} x 32
Physical memory
(b) Physical address format: 4096 blocks of 256 words each, each word has 32 bits

Figure 33 An example of logical and physical addresses.

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(34-a). The total logical address range for the program is from hexadecimal 60000 to 604FF. The correspondence between each memory block and logical page number is then entered in a table as shown in Fig(34-b).



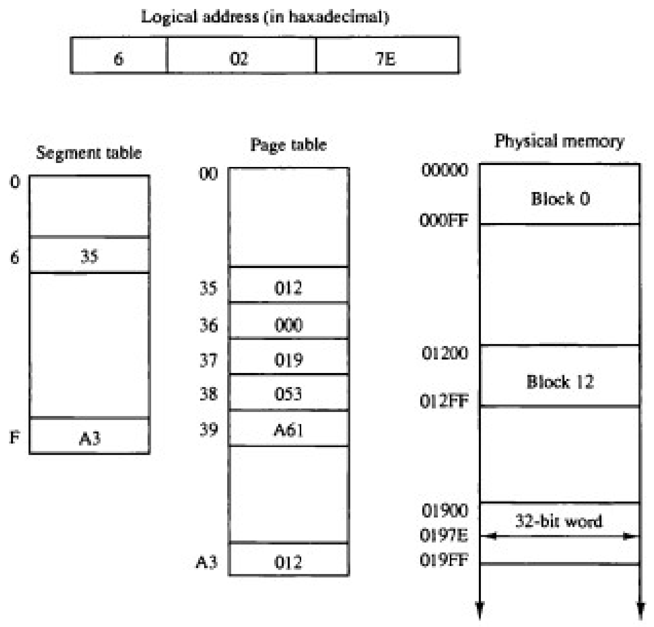
(a) Logical address assignment

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

(b) Segment-page versus memory block assignment

Figure 34 Example of logical and physical memory address assignment.

The information from this table is entered in the segment and page tables as shown in Fig(35- a). Now consider the specific logical address given in Fig(35). 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. The associative memory in Fig(35-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.



(a) Segment and page table mapping

Continue

Segment	Page	Block
6	02	019
6	04	A61

(b) Associative memory (TLB)

Figure 35 Logical to physical memory mapping example (all numbers are in hexadecimal).