Digital Logic Design And Computer Organization

UNIT-I

Computer Types

Computer:-

- Computer is a fast electronic calculating machine that accepts digitized input information ,processes it according to a list of internally stored instruction ,and produces the result out put information.
- Here list of instructions called a computer program and the internal storage is called computer memory.

Computer Organization

- Computer organization is defined as the way the hardware components operate and the way they are connected together to form the computer system.

Computer Types

- There are many types of computers that differ many factors like size,cost ,performance and use of computer.

1)personal computer:

-having processing and storage units ,display ,audio and keyboard.

-used in homes ,schools and business offices.

2)Notebook computers:

-compact version of personal computer .

-all components are packed into a single unit with the size of briefcase.

3)Work sattions :

-same as desktop but its having high resolution graphics I/O capability.

-More computational power than personal computers.

-Used in Engineering applications and Interactive apllications.

Computer Types

4)Enterprise Systems or Main Frames :

- Used for business data processing in medium to large corporations that require much more computing power and storage capacity than workstations.

5)Servers:

-it contains sizable data units.

-servers capable of handling large volumes of requests to access the data.

- used in education ,business and user personal communities.

6)Super Computers:

-used for large scale numerical calculations.

-Examples weather forecasting ,and aircraft design and simulation.

Functional Units

- Computer Consists of Five functionally independent parts:

1) Input 2) Memory 3) ALU 4) Output 5)Control Unit

-input unit accepts coded information from human operators from electromechanical devices such as keyboard or from other computers over communication lines.

-The received information is either stored in memory or used by ALU to perform the desired operation.

-The processing steps are determined by a program stored in the memory.

-Finally results are sent back to the outside world through the output unit.

-All the above operations are coordinated by the control unit.

Functional Units

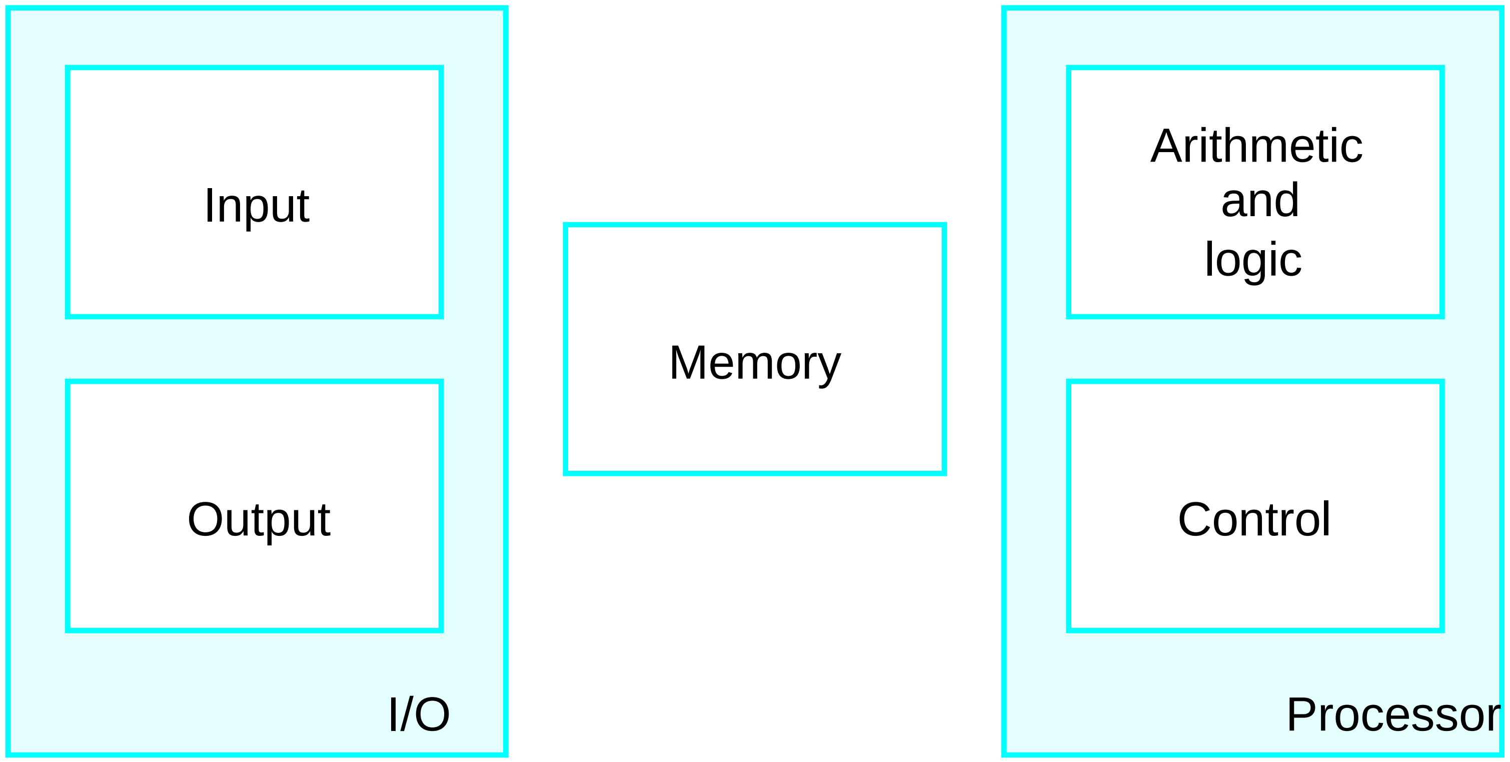


Figure 1.1. Basic functional units of a computer.

Functional Units

- Instructions or Machine instructions are explicit commands responsible for

  - The transfer of information with in a computer as well as between the computer and its I/O devices.

  -It specify the ALU operations to be performed.
- A list of instructions that perform a task is called a program.
- High level program called source program.
- Machine level program called object program.
- Information is encoded in the form of 0’s and 1’s.

Functional Units

- Input Unit:

-accepts coded information through input unit.

-when ever a key is pressed the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted to memory or processor.

-other input devices like joysticks ,mouse , microphone ,trackball etc.

- Memory Unit:

-Function of memory is to store programs and data.

-there are two types of memory

1.Primary memory

2.Secoundary memory

Primary Memory

-primary memory is fast memory operates at electronic speeds.

Functional Units

-memory is collection of semiconductor cells.

-Each cell is capable of storing one bit information.

-groups of fixed size cells called words.

-Each and every word is assigned with distinct addresses.

-the number of bits in each word is often referred as the word length of the computer.

-The memory in which any location can be reached in short and fixed amount of time after specifying its address is called random-access-memory .

-The time required to access one word is called memory access time.

-cache are small and fast accessing RAM units .these are tightly coupled with the processor to achieve high performance.

Secondary Storage:

-used to store large amount of data. Ex: tapes ,disk , optical disks.

Functional Units

ALU:

-Most of the computer operations are executed in the ALU of the processor.

Ex: add,sub,mul,div

-access time to registers are fast when compared to cache in memory hierarchy.

-The CU and ALU are many times faster than other devices that connect to a computer system . So that a single processor to control many external devices.

Output Unit:

-The output unit is counterpart of input unit.

-the main function is to send processed results to the outside world.

Ex: printers , monitors.

Functional Units

Control Unit:

-The control unit is effectively the nerve center that sends control signals to the other units.

-the memory ,ALU ,an I/O units store and process information and perform input and output operations. All these operations are carried out by CU.

Operations of computer Summarized as:

-the computer accepts information in the form of programs and data through an input unit and stores it in the memory.

-information stored in the memory is fetched under program control into an ALU where it is processed.

-processed information leaves the computer through an output unit.

-All activities inside the machine are directed by the control unit.

Basic Operational Concepts

- Activity in a computer is governed by instructions.
- To perform a task, an appropriate program consisting of a list of instructions is stored in the memory.
- Individual instructions are brought from the memory into the processor, which executes the specified operations.
- Data to be used as operands are also stored in the memory.
- An Instruction : Add LOCA, R0
- Add the operand at memory location LOCA to the operand in a register R0 in the processor.
- Place the sum into register R0.
- The original contents of LOCA are preserved.
- The original contents of R0 is overwritten.
- Instruction is fetched from the memory into the processor – the operand at LOCA is fetched and added to the contents of R0 – the resulting sum is stored in register R0.

Basic Operational Concepts

- Due to performance reasons the above instruction is implemented into two instructions:

```
Load   LocA , R1
Add     R1,R0
```

- Whose contents will be overwritten?

Basic Operational Concepts

Basic Operational Concepts

Registers

1) Instruction register (IR)

2) Program counter (PC)

3) General-purpose register (R_0 – R_{n-1})

4) Memory address register (MAR)

5) Memory data register (MDR)

Operating Steps

- Programs reside in the memory through input devices
- PC is set to point to the first instruction
- The contents of PC are transferred to MAR
- A Read signal is sent to the memory
- The first instruction is read out and loaded into MDR
- The contents of MDR are transferred to IR
- Decode and execute the instruction

Basic Operational Concepts

Operating Steps(Cont..)

- Get operands for ALU
  - ➢ General-purpose register
  - ➢ Memory (address to MAR – Read – MDR to ALU)
- Perform operation in ALU
- Store the result back
  - ➢ To general-purpose register
  - ➢ To memory (address to MAR, result to MDR – Write)
- During the execution, PC is incremented to the next instruction

Basic Operational Concepts

Interrupt

- Normal execution of programs may be preempted if some device requires urgent servicing.
- The normal execution of the current program must be interrupted – the device raises an *interrupt* signal.
- Interrupt-service routine
- Current system information backup and restore (PC, general-purpose registers, control information)

Bus Structures

- To achieve good performance of a computer system all the computer units can transfer one word of data at a time.
- All the bits of a word transfer in parallel, that is the bits are transferred simultaneously over many lines one bit per line.
- There are many ways to connect different parts inside a computer together.
- A group of lines that serves as a connecting path for several devices is called a *bus*.
- A bus that connects major components is called system bus.
- System bus is divided into three functional groups : Address bus , data bus , control bus.
- Only two units can use bus structure at any point of time.

Types of Buses

- There are different 9 type of buses

1) System bus
2) Single bus
3) Multiple Bus
4) Internal bus
5) External Bus
6) I/O Bus
7) Synchronous Bus
8) Asynchronous Bus
9) Back pane bus

Types of Buses

1)System Bus:

- System Bus Contains Data bus , Address Bus , Control Bus

Data Bus:

- Data bus consists of 8,16,32 or more parallel signal lines.
- These lines are used to send the data to memory and output ports and to receive data from memory and input ports.
- It is a bi-directional bus.

Address Bus:

- It is an Unidirectional bus.
- The address bus consists of 16, 20, 24 or more parallel signal lines.
- The cpu sends out the address of the memory location or I/O ports that is to be written to or read from.

Control Bus:

- The Control lines regulate the activity on the bus.

Ex: MEMW,MEMR,IOR,IOW,INTR   etc….

2) Single-bus

Speed Issue

•Different devices have different transfer/operate speed.

•If the speed of bus is bounded by the slowest device connected to it, the efficiency will be very low.

•How to solve this?

•A common approach – use buffers. to smooth timing differences between devices. Ex: processor and printer data transfer.

Advantages:

•The main advantage of single bus structure is its low cost and flexibility for attaching peripheral devices.

3) Multiple Bus:

- Multiple bus Connection uses more number of different bus to connect the components.
- Generally it uses local bus , system bus , expanded bus , high speed bus.
- Multiple buses are used to transfer video and graphics type of data.

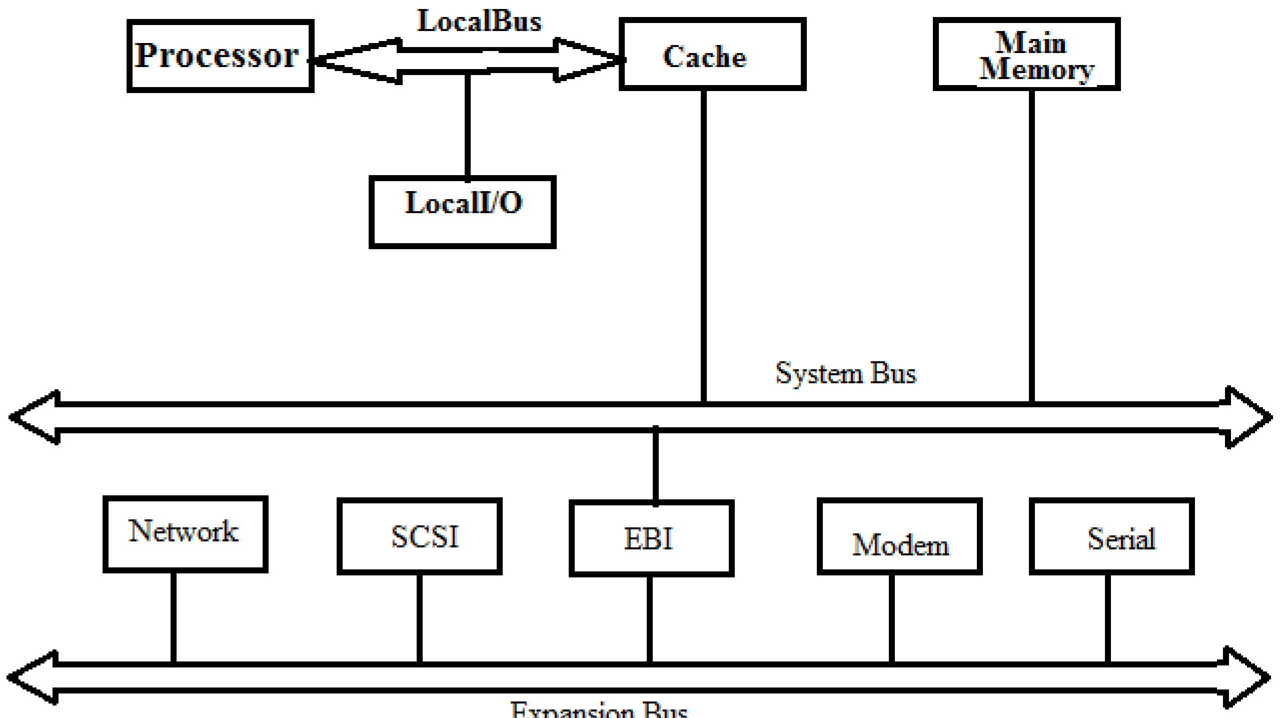


Fig: Multiple Bus Structure

Internal Bus:

- An internal bus connects all the internal components of a computer to the motherboard it is also called localbus.
- The internal bus of CPU connects the internal circuitry of the CPU.

External Bus:

- An External bus connects external peripherals to the mother boaed.
- Ex: USB.

I/O Bus:

- I/O bus is used to link between the processor and several peripherals .

Synchronous Bus:

- All devices derive timing information from a common clock signal then synchronous bus is used.

Asynchronous Bus:

- All devices derive timing information from a independent clock signal then Asynchronous bus is used.

Backplane Bus:

- A backplane or backplane system is a circuit board that connects several connectors in parallel to each other.

- It s used as a back bone to connect several system modules to make up a complete computer system.

Software

- To execute user application programs a computer contains a software called system software.
- System software is a collection of programs that are executed to perform functions like :

1) Receiving and interpreting user commands.

2) Entering and editing application programs and storing them as files in secondary storage device.

3) Managing the storage and retrieval of files in secondary storage devices.

4) Running Standard Application Programs such as Word processor , Spread Sheets or games with data supplied by the user.

5) Controlling I/O Units to receive input information and produce output results.

6) translating programs from source form prepared by the user into object form consisting of machine instructions.

Software

7) Linking and running user-written application programs with existing standard library routines such as numerical computation programs.

- Text editor ,Compiler , operating system are examples of System software.
- The operating system is used to manage the execution of more than one application program at a time.

Ex:- How the operating system manages the execution of application programs.

1) Transfer the file into the memory.
2) After Completion of transfer execution is started.
3) When execution of program reaches the point where the data file is needed , the program requests the operating system to transfer the data file from the disk to memory and passes control back to application program. Then proceed to perform the required computation.
4) when the computation is completed the application program sends a request to os .then os sends result to printer to print.

Software

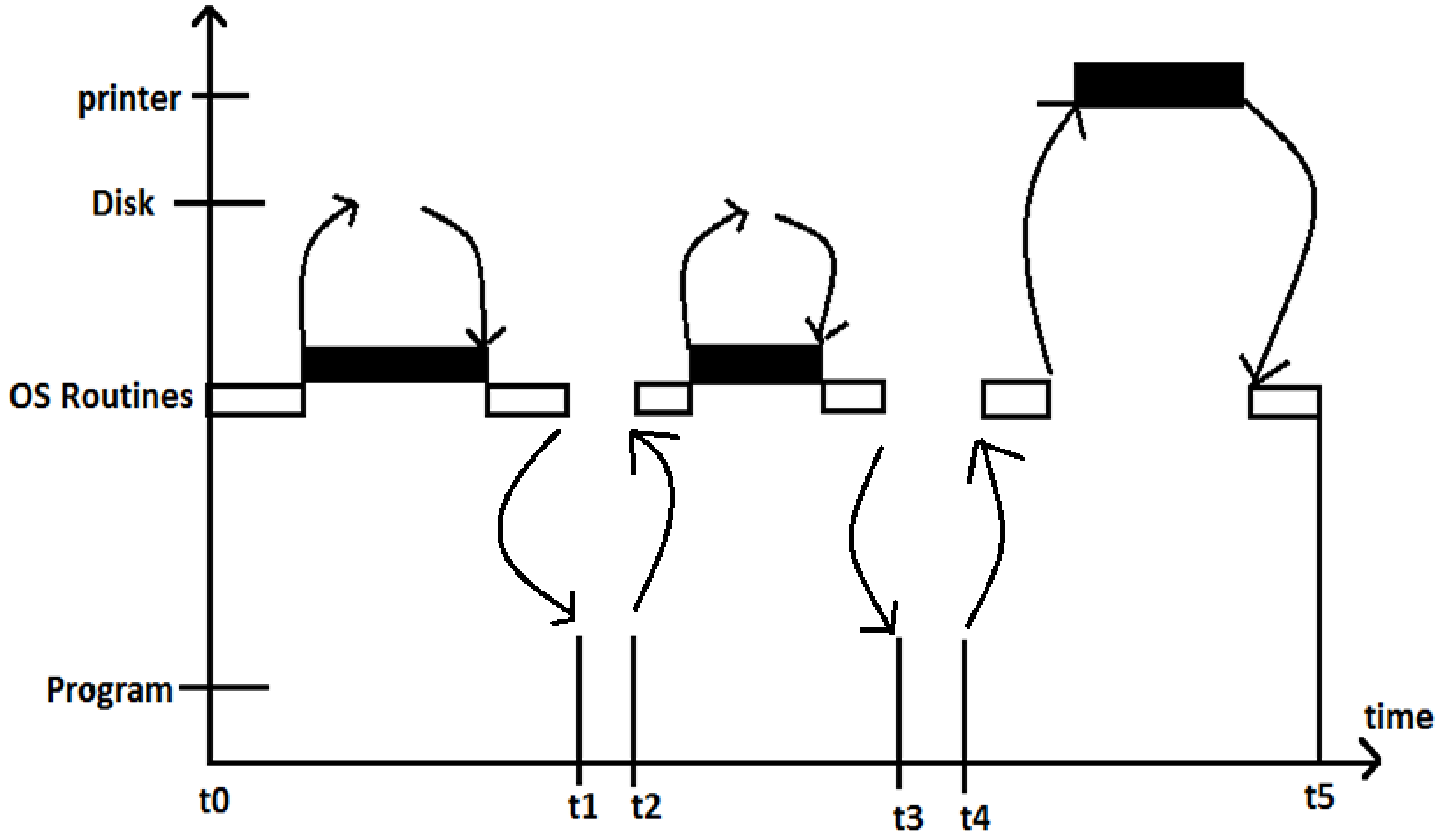


Fig: User program and OS routine Sharing of the processor

Performance

- The most important measure of a computer is how quickly it can execute programs.
- Three factors affect performance:
  - Hardware design
  - Instruction set
  - Compiler
- The time required to compute a total process is call elapsed time.
- The time that the processor execute the program is called processor time.
- Elapsed time for the execution of a program depends on all units in a computer system.

Performance

•The processor time depends on the hardware involved in the execution of individual machine instruction.

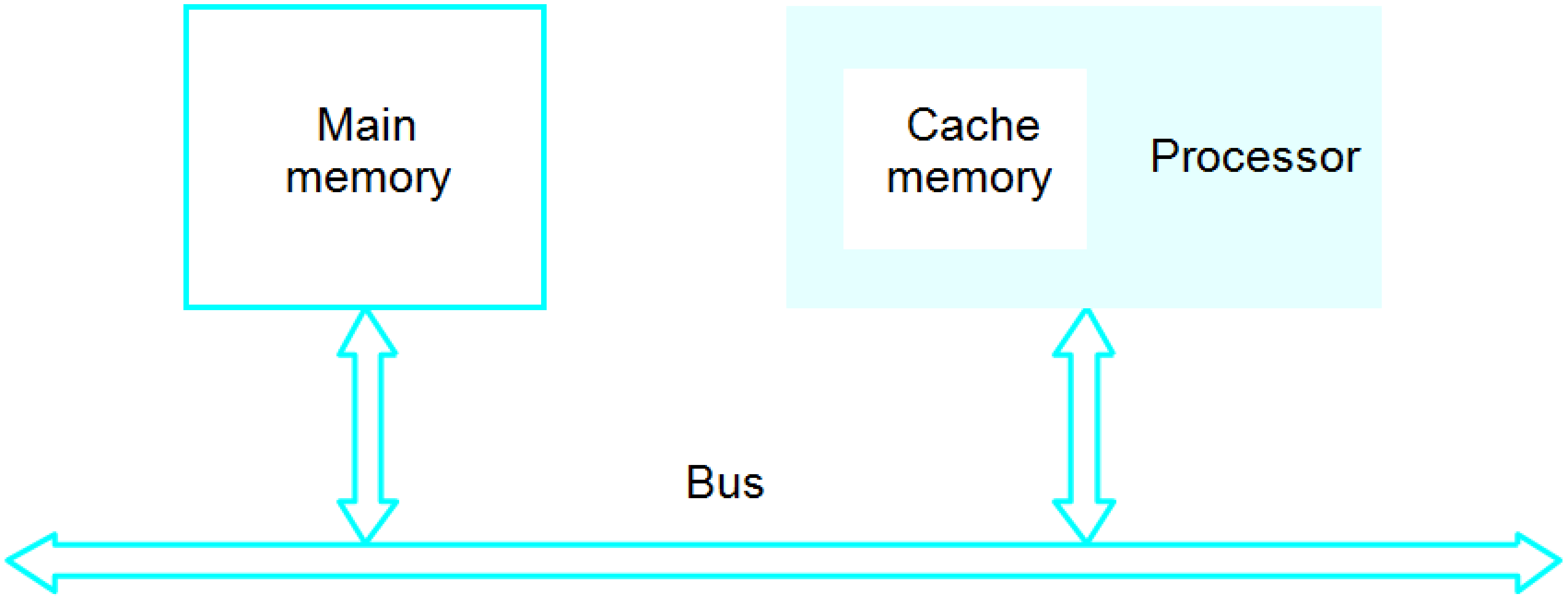


Figure 1.5. The processor cache.

Performance

Processor Clock

- Processor circuits are controlled by a timing signal called a clock.

- The clock defines regular timing intervals , called clock cycles.

- To execute any machine instruction the processor divide the instruction into a sequence of basic steps ,such that each step can be completed in one clock cycle.

- Clock rate R=1/p   here p is clock cycle length.

- Clock speed is measured in hertz(Hz)

Performance

Basic Performance Equation:

- T – processor time required to execute a program that has been prepared in high-level language
- N – number of actual machine language instructions needed to complete the execution (note: loop)
- S – average number of basic steps needed to execute one machine instruction. Each step completes in one clock cycle
- R – clock rate
- Note: these are not independent to each other

$$T = \frac{N \times S}{R}$$

How to improve T?

Performance Measurement

- T is difficult to compute.
- Measure computer performance using benchmark programs.

Ex: database ,compiler , game playing.

- System Performance Evaluation Corporation (SPEC) selects and publishes representative application programs for different application domains, together with test results for many commercially available computers.
- Reference computers like SPEC95 , SUN SPARC , SPEC2000.

$$\text{SPEC rating} = \frac{\text{Running Time on the reference Computer}}{\text{Running Time on the computer under test}}$$

$$\text{SPEC Rating} = \left(\prod_{i=1}^{n} \text{SPEC}_i\right)^{1/n}$$

- n is the number of programs in the suite.

Amdahl’s Law

- Amdahl’a law is used to calculate the performance gain that can be obtained by improving some portion of a computer.

$$\text{Speed up} = \frac{\text{Performance for entire task using improved machine}}{\text{Performance for entire task using old machine}}$$

(or)

$$\text{Speed up} = \frac{\text{Execution time for entire task using improved machine}}{\text{Execution time for entire task using original machine}}$$

Amdahl's Law

Fraction Enhanced(Fe):

- It is the fraction of the computation time in the original machine that can be converted to take advantage of the enhancement.
- Fe is always <=1.

Speedup Enhanced(Se):

- It tells how much faster the task would run if the enhancement mode was use for the entire program.
- Speed up enhancement is always >1.

$$\text{Speedup} = \frac{\text{Execution Time}_{\text{old}}}{\text{Execution Time}_{\text{new}}} = \frac{E_{TO}}{E_{TN}}$$

Amdahl's Law

$$E_{TN} = E_{TO} \times \left[\left(1\text{-}F_e\right)+\left(F_e/S_e\right)\right]$$

$$\text{Speed Up} = \frac{E_{TO}}{E_{TO} \times \left[\left(1\text{-}F_e\right)+\left(F_e/S_e\right)\right]}$$

$$= \frac{1}{(1\text{-}Fe)+(Fe/Se)}$$

Multiprocessors and Multicomputers

Multiprocessor computer

- Execute a number of different application tasks in parallel
- Execute subtasks of a single large task in parallel
- All processors have access to all of the memory – shared-memory multiprocessor
- Cost – processors, memory units, complex interconnection networks

Multicomputers

- Each computer only have access to its own memory
- Exchange message via a communication network – message-passing multicomputers

DATA REPRESENTATION

- Information that a Computer is dealing with

    * Data
        - Numeric Data
            Numbers(Integer, real)
        - Non-numeric Data
            Letters, Symbols

    * Relationship between data elements
        - Data Structures
            Linear Lists, Trees, Rings, etc

    * Program(Instruction)

NUMERIC DATA REPRESENTATION

- Number System
  - Nonpositional number system
    - Roman number system
  - Positional number system
    - Each digit position has a value called a *weight* associated with it
    - Decimal, Octal, Hexadecimal, Binary
- Base (or radix) R number
  - Uses R distinct symbols for each digit
  - Example $A_R = a_{n-1}\, a_{n-2} \ldots a_1\, a_0 . a_{-1} \ldots a_{-m}$
  - $V(A_R) = \sum_{i=-m}^{n-1} a_i R^i$

Radix point(.) separates the integer portion and the fractional portion

R = 10 Decimal number system, R = 2 Binary

R = 8 Octal, R = 16 Hexadecimal

NUMERIC DATA REPRESENTATION

Decimal Number System

- The decimal number system in every day use employs the radix 10 system.
- The 10 symbols are 0,1,2,3,4,5,6,7,8 and 9.
- The string of digits 834.5 is interpreted as:

  $8X10^2 + 3X10^1 + 4X10^0+5X10^{-1} \quad =834.5$

Binary Number System

- Binary number system uses the radix 2.
- The two digit symbols used are 0 and 1.
- The string of symbols 1001 is interpreted as:

  $1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \quad =8+0+0+1=9$

NUMERIC DATA REPRESENTATION

Octal Number System

- Octal Number System uses radix 8.
- The Symbols used to represent the octal number system is 0,1,2,3,4,5,6 and 7.
- The octal number is converted into decimal number system by forming the sum of the weighted digits.

Ex:

$(736.4)_8 = ?$

$= 7 \text{ x } 8^2 + 3 \text{ x } 8^1 + 6 \text{ x } 8^0 + 4 \text{ x } 8^{-1}$

$= 7 \text{ x } 64 + 3 \text{ x } 8 + 6 \text{ x } 1 + 4/8 = (478.5)_{10}$

NUMERIC DATA REPRESENTATION

Hexadecimal Number System

- The hexadecimal number system uses radix 16.
- The symbols used to represent the hexadecimal number system is 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E and F.
- The hexadecimal number is converted into decimal number system by forming the sum of the weighted digits.

Ex:

$(F3)_{16} = ?$

$= F \times 16^{1} + 3 \times 16^{0}$

$= 15 \times 16 + 3 = (243)_{10}$

NUMERIC DATA REPRESENTATION

Decimal to Other Number Systems

- Conversion from decimal to its equivalent representation in the radix r system is carried our by separating the number into its integer part and fraction part and converting each part separately.

- The conversion of a decimal integer into a base r representation is done by successive divisions by r and accumulation of the reminders.

- The conversion of a decimal fraction to radix r representation is accomplished by successive multiplication by r and accumulation of the integer digits obtained.

Decimal to Binary Conversion:

Ex: $(41.6875)_{10} = (101001.1011)_2$

Integer = 41

41	
20	1
10	0
5	0
2	1
1	0
0	1

$(41)_{10} = (101001)_2$

Fraction = 0.6875

```
0.6875
x      2
1.3750
x      2
0.7500
x      2
1.5000
 x     2
1.0000
```

$(0.6875)_{10} = (0.1011)_2$

Binary to Octal and Hexadecimal Conversion

- Each octal digit corresponds to three binary numbers i.e $8=2^3$.
- Each hexadecimal digit corresponds to four binary numbers i.e $16=2^{4.}$

Binary, octal, and hexadecimal conversion

1 2 7 5 4 3	Binary
1 0 1 0 1 1 1 1 0 1 1 0 0 0 1 1	Octal
A F 6 3	Hexa

BCD:

– BCD is used to represent the decimal numbers system to binary number system

COMPLEMENT OF NUMBERS

Two types of complements for base R number system:

1) (R-1)'s complement

2) R's complement

1) The (R-1)'s Complement

- Number N in base r having n digits (r-1)’s complement is defined as $(r^n - 1) - N$.
- Subtract each digit of a number from (R-1)

Example

- 9's complement of 835_{10} is 164_{10}
- 1's complement of 1010_2 is 0101_2(bit by bit complement operation)

COMPLEMENT OF NUMBERS

2) The R's Complement

- The r’s complement of an n-digit number N in base r is defined as r^n-N for N is not 0.

- Add 1 to the low-order digit of its (R-1)'s complement

Example

- 10's complement of 835_{10} is $164_{10} + 1 = 165_{10}$

- 2's complement of 1010_2 is $0101_2 + 1 = 0110_2$

Subtraction of unsigned numbers

- The subtraction of two n-digit unsigned numbers M-N (N is not zero)in base r can be done as follows:

1) Add the minuend M to the r’s complement of the subtrahend N.this performs $M+(r^n-N)=M-N+r^n$.

2) If M>=N , the sum will produce an end carry r^n which is discarded , and what is left is the result of M-N.

3) If M<N , the sum does not produce an end carry and is equal to $r^n-(N-M)$, which is the r’s complement of (N-M).To obtain the answer in familiar form ,take the r’s complement of the sum and place a negative sign in front.

Subtraction of unsigned numbers

Example:

M=72532 N=13250

Here M>N

$$\begin{array}{lr} & 75532 \\ \text{10's Complement of N} & 86750 \\ \hline & 159282 \end{array}$$

End Carry Discard

Subtraction of unsigned numbers

Example2:

M<N

M=13250 N=72532

```
                    M=     13250
10’s complement of  N=     27468
                          ______
                  Sum =  40718
```

Answer is Negative 59282 =10’s complement of 40718

Subtraction of unsigned numbers

Example 3:

X=1010100 Y=1000011

```
                      X=    1010100
2’s Complement of     Y=    0111101
                          ---------
                    Sum=   10010001
      Answer of X-Y =       0010001
```

Subtraction of unsigned numbers

Example 4:

X= 1010100　　　　　　Y=1000011

```
                    Y=  1000011
2's Complement of   X=  0101100
                    ------------
             Sum    =   1101111
```

There is no End Carry.

Answer is negative 0010001 = 2' Complement of 1101111

Fixed Point Representation

- Positive integers , including zero ,can be represented as unsigned numbers.
- The positive sign is represented as 0 and 1 for negative.
- The binary point is used to represent fractions ,integers , and mixed integer and fractions.
- There are two ways of specifying the position of the binary point in a register

  - By giving it a fixed point representation

  - By employing a floating point representation
- The fixed point method assumes that the binary point is always fixed in one position.
- The two positions most widely used are:

  1) Binary point extreme left of the register

  2) Binary point extreme right of the register

Fixed Point Representation

Integer Representation

- When a binary number is positive the sign is represented by 0 and the magnitude by a binary positive number.
- When the number is negative the sign is represented by 1 and the rest of the number may be represented in one of the three ways:

1) Signed magnitude representation
2) Signed 1's complement representation
3) Signed 2's complement representation

Example: Represent +9 and -9 in 7 bit-binary number

Only one way to represent +9 ==> 0 001001

Three different ways to represent -9:

In signed-magnitude:	1 001001
In signed-1's complement:	1 110110
In signed-2's complement:	1 110111

Fixed Point Representation

- Complement
  - Signed magnitude: Complement only the sign bit
  - Signed 1's complement: Complement all the bits including sign bit
  - Signed 2's complement: Take the 2's complement of the number,including its sign bit.

Fixed Point Representation

Arithmetic Addition in Signed Magnitude

Rules

1 . Compare their signs

2 . If two signs are the *same* ,

ADD the two magnitudes - Look out for an *overflow*

3 . If *not the same* , compare the relative magnitudes of the numbers and then *SUBTRACT* the smaller from the larger .

4 . Determine the sign of the result

Note: 1) The operation performed always addition including sign bit.

2) any carry out of sign bit is discarded.

3) Negative results are always in 2s complement form.

Fixed Point Representation

Example:

+ 6	00000110	-6	11111010
+13	00001101	+13	00001101
+19	00010011	+7	00000111
+6	00000110	-6	11111010
-13	11110011	-13	11110011
-7	11111001	-19	11101101

Fixed Point Representation

Arithmetic Subtraction

Rules:

1) To perform subtraction of two signed binary numbers take the 2’s complement of the subtrahend (including sign bit) and add it to the minuend(including the sign bit) .

2) Discard the carry out of the sign bit position.

- The subtraction operation can be changed to addition if the sign of the subtrahend is changed.

$$(\pm A) - (- B) = (\pm A) + (+B)$$

$$(\pm A) - (+B) = (\pm A) + (- B)$$

Fixed Point Representation

Arithmetic Subtraction

Example:

(-6) – (-13)=+7

take the 2'complement of -13 i.e +13=00001101

```
   -6      11111010
   +13     00001101
---------------------
   +7     100000111
```

Remove the End carry   i.e 00000111=+7

Fixed Point Representation

Overflow

- When two numbers of n digits each are added and then the sum occupies n+1 digits then we call overflow occurred.
- An overflow is a problem in digital computers because the width of a register is finite.
- Because of this reason many computes detect the occurrence of overflow problem and set a overflow flip flop.
- When two numbers are added the overflow is detected by using an end carry.
- An overflow can not occur after an addition if one number is positive and the other is negative .
- An overflow may occur if the two numbers added are both positive or both negative.

Fixed Point Representation

Example :

carries :	0 1	carries:	1 0
+70	0 1000110	-70	1 0111010
+80	0 1010000	-80	1 0110000
+150	1 0010110	-150	0 1101010

Overflow Detection

- An overflow is detected by observing the carry into the sign bit position and the carry out of the sign bit position.
- If these two carries are not equal an overflow is occurred.
- If the two carries are applied to an exclusive-OR gate, an overflow will be detected when the output of the gate is equal to 1.

Floating-point Representation

- The floating-point representation of a number has two parts.
  - The first part represents a signed , fixed-point number called mantissa .
  - The second part designates the position of the decimal(or) binary point and is called the exponent.

$$m \text{ X } r^{e}$$

- The fixed point mantissa may be a fraction or an integer.

Example:

+6132.789 is represented in floating point as :

fraction=+0.613789 exponent= +04

Floating-point Representation

- A floating point binary number +10011.11 is represented as:

  Fraction=01001111   Exponent=000101

Normalization:

- A floating point number is said to be normalized if the most significant digit of the mantissa is nonzero.
- The number can be normalized by shifting it three positions to the left and discarding the leading 0's.

Example:

  - 00011010 is normalized as 11010000

- In the above example the three shifts multiply the number by 2^3 =8 to keep the same number the exponent must be subtracted by 3.

Floating-point Representation

- There are two standards to represent floating point numbers:1)ANSI 2)IEEE
- The ANSI 32 bit format is represented As:

Byte Format:

Byte1 Byte2 Byte3 Byte4

$$\underbrace{\text{SEEEE}}_{\text{Exponent}}\quad \underset{\text{Binary Point}}{\underset{\uparrow}{.}}\underbrace{\text{IMMMMMMM}\quad\text{MMMMMMMM}\quad\text{MMMMMMMM}}_{\text{Mantissa}}$$

S=Sign of Mantissa E=Exponent Bits in 2’s complement M=Mantissa

Ex: $13=1101=0.1101\text{X}2^{4}$

=00000100 11010000 00000000 00000000

$-17=-10001=-0.10001\text{X}2^{5}$ =10000101 10001000 00000000 00000000

$-0.125=-0.001=-1\text{X}2^{-2}$=11111110 10000000 00000000 00000000

Error Detection Codes

Parity System

- Simplest method for error detection
- One *parity* bit attached to the information
- *Even Parity* and *Odd Parity*

- Even Parity
  - One bit is attached to the information so that the total number of 1 bits is an even number

    1011001 0

    1010010 1

- Odd Parity
  - One bit is attached to the information so that the total number of 1 bits is an odd number

    1011001 1

    1010010 0

Error Detection Codes

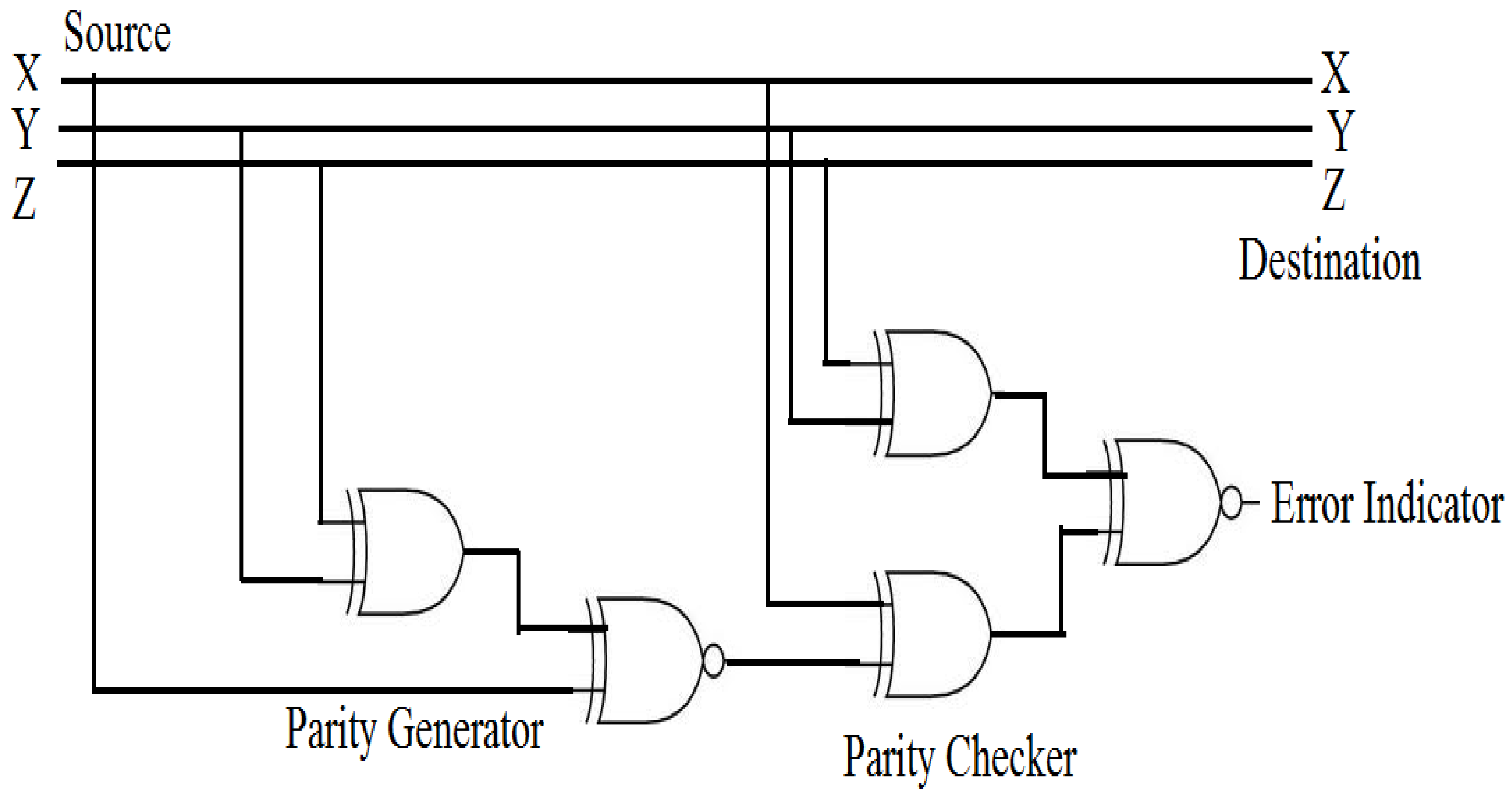


Fig: Error Detection With Odd Parity Bit

UNIT-II

Fundamentals of Boolean Algebra

- ***Basic Postulates***
- ***Postulate 1 (Definition)***: A Boolean algebra is a closed algebraic system containing a set K of two or more elements and the two operators $\bullet$ and +.
- ***Postulate 2 (Existence of 1 and 0 element)***:
  (a) $a + 0 = a$ (identity for +), (b) $a \bullet 1 = a$ (identity for $\bullet$)
- ***Postulate 3 (Commutativity)***:
  (a) $a + b = b + a$, (b) $a \bullet b = b \bullet a$
- ***Postulate 4 (Associativity)***:
  (a) $a + (b + c) = (a + b) + c$ (b) $a \bullet (b \bullet c) = (a \bullet b) \bullet c$
- ***Postulate 5 (Distributivity)***:
  (a) $a + (b \bullet c) = (a + b) \bullet (a + c)$ (b) $a \bullet (b + c) = a \bullet b + a \bullet c$
- ***Postulate 6 (Existence of complement)***:
  (a) $a + \overline{a} = 1$ (b) $a \bullet \overline{a} = 0$
- Normally $\bullet$ is omitted.

Fundamentals of Boolean Algebra

- ***Fundamental Theorems of Boolean Algebra***

- ***Theorem 1 (Idempotency)***:
  (a) a + a = a (b) *aa = a*
- ***Theorem 2 (Null element)***:
  (a) *a* + 1 = 1 (b) *a*0 = 0
- ***Theorem 3 (Involution)***
  $\overline{\overline{a}} = a$

- ***Properties of 0 and 1 elements*** (Table 2.1):

OR	AND	Complement
a + 0 = 0	*a*0 = 0	0' = 1
a + 1 = 1	*a*1 = *a*	1' = 0

Fundamentals of Boolean Algebra (3)

- ***Theorem 4 (Absorption)***
  (a) $a + ab = a$ (b) $a(a + b) = a$

- ***Examples***:
  - $(X + Y) + (X + Y)Z = X + Y$
  - $AB'(AB' + B'C) = AB'$
- ***Theorem 5***
  (a) $a + a'b = a + b$ (b) $a(a' + b) = ab$

- ***Examples***:
  - $B + AB'C'D = B + AC'D$
  - $(X + Y)((X + Y)' + Z) = (X + Y)Z$

Fundamentals of Boolean Algebra (4)

- ***Theorem 6***

  (a) $ab + ab' = a$ (b) $(a + b)(a + b') = a$

- ***Examples***:

– $ABC + AB'C = AC$

– $(W' + X' + Y' + Z')(W' + X' + Y' + Z)(W' + X' + Y + Z')(W' + X' + Y + Z)$

$= (W' + X' + Y')(W' + X' + Y + Z')(W' + X' + Y + Z)$

$= (W' + X' + Y')(W' + X' + Y)$

$= (W' + X')$

Fundamentals of Boolean Algebra (5)

- ***Theorem 7***

  (a) $ab + ab'c = ab + ac$ (b) $(a + b)(a + b'$
$+ c) = (a + b)(a + c)$

- ***Examples***:

– $wy' + wx'y + wxyz + wxz' = wy' + wx'y + wxy + wxz'$
$= wy' + wy + wxz'$

$= w + wxz'$

$= w$

– $(x'y' + z)(w + x'y' + z') = (x'y' + z)(w + x'y')$

Fundamentals of Boolean Algebra (6)

- ***Theorem 8 (DeMorgan's Theorem)***
  (a) $(a + b)' = a'b'$ (b) $(ab)' = a' + b'$

- Generalized DeMorgan's Theorem
  (a) $(a + b + \ldots z)' = a'b' \ldots z'$ (b) $(ab \ldots z)' = a' + b' + \ldots z'$

- ***Examples***:
  – $(a + bc)' = (a + (bc))'$
    $= a'(bc)'$
    $= a'(b' + c')$
    $= a'b' + a'c'$

  Note: $(a + bc)' \neq a'b' + c'$

Logic Gates

- ***Electrical Signals and Logic Values***

Electric Signal	Logic Value	
	Positive Logic	Negative Logic
High Voltage (H)	1	0
Low Voltage (L)	0	1

– A signal that is set to logic 1 is said to be *asserted*, *active*, or *true*.

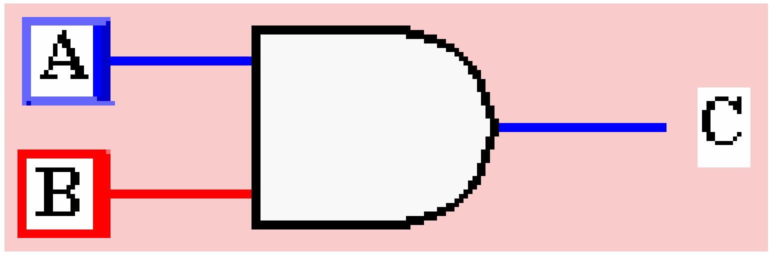
– An *active-high* signal is asserted when it is high (positive logic).

– An *active-low* signal is asserted when it is low (negative logic).

AND

– Logic notation A•B = C

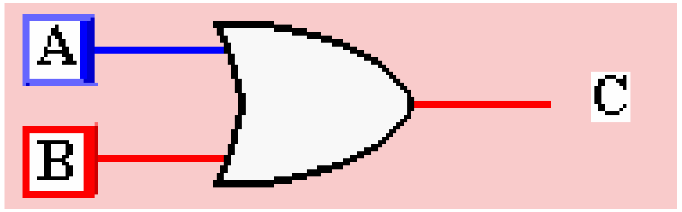
(Sometimes AB = C)



A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

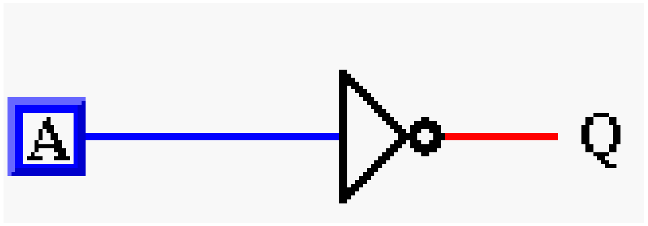
OR

– Logic notation A + B = C



A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

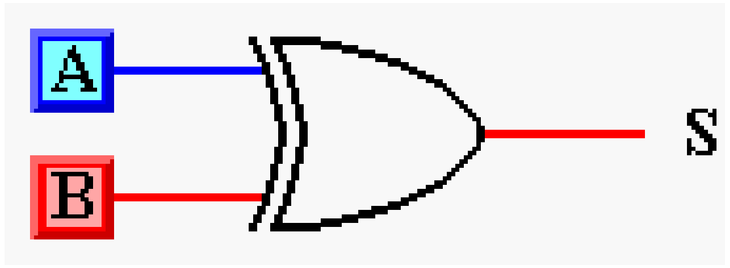
Inversion (NOT)



Logic: $Q = \overline{A}$

A	Q
0	1
1	0

Exclusive OR (XOR)



Either A or B, but not both

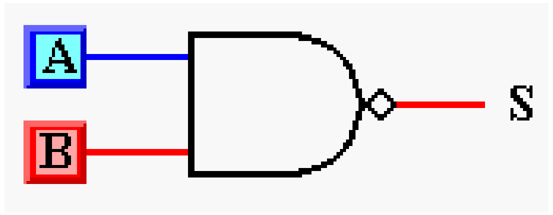
This is sometimes called the **inequality detector**, because the result will be 0 when the inputs are the same and 1 when they are different.

The truth table is the same as for S on Binary Addition. S = A ⊕ B

A	B	S
0	0	0
1	0	1
0	1	1
1	1	0

UNIVERSAL GATES

NAND (NOT AND)

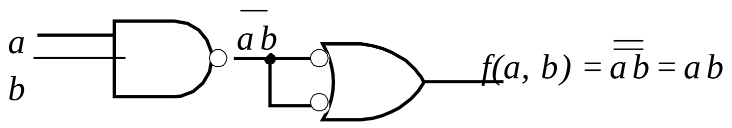


$$Q = \overline{A \cdot B}$$

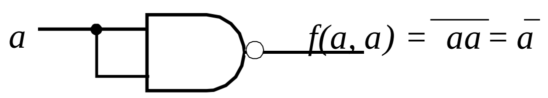
A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

Basic Functional Components

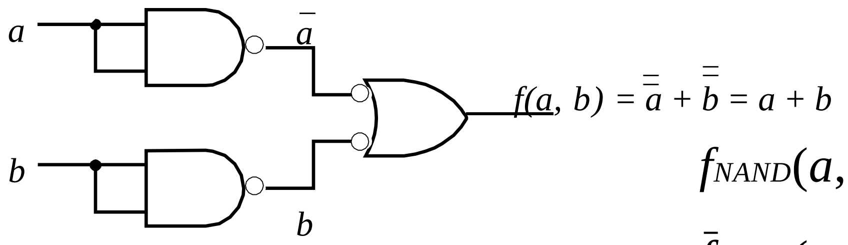
- AND, OR, and NOT gates constructed exclusively from NAND gates



AND gate



NOT gate


a
$\bar{a}$
b
$\bar{b}$
$f(a,\,b) = \overline{\bar{a}} + \overline{\bar{b}} = a + b$

OR gate

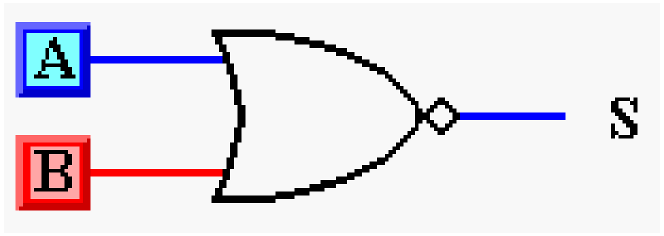
$$f_{NAND}(a,a) = \overline{a \cdot a} = \bar{a} = f_{NOT}(a)$$

$$\bar{f}_{NAND}(a,b) = \overline{\overline{a \cdot b}} = a \cdot b = f_{AND}(a,b)$$

$$f_{NAND}(\bar{a},\bar{b}) = \overline{\bar{a} \cdot \bar{b}} = a + b = f_{OR}(a,b)$$

Hence, NAND gate may be used to implement all three elementary operators.

NOR (NOT OR)

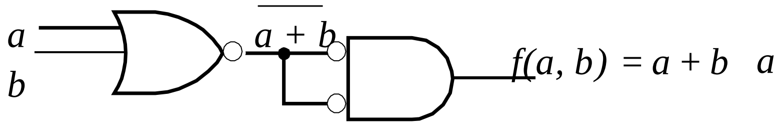


$$Q = \overline{A + B}$$

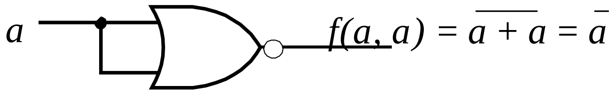
A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

Basic Functional Components

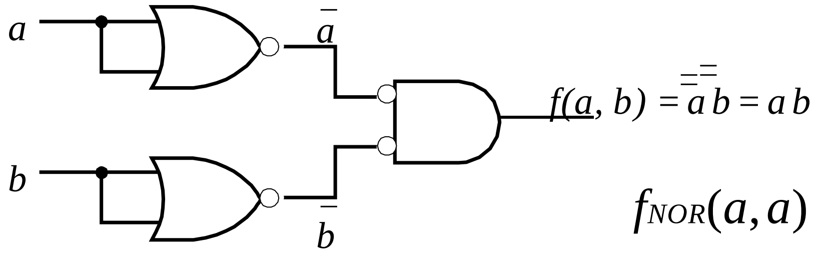
- AND, OR, and NOT gates constructed exclusively from NOR gates.



OR gate



NOT gate



AND gate

$$f_{NOR}(a,a) = \overline{a+a} = \bar{a} = f_{NOT}(a)$$

$$\bar{f}_{NOR}(a,b) = \overline{\overline{a+b}} = a+b = f_{OR}(a,b)$$

$$f_{NOR}(\bar{a},\bar{b}) = \overline{\bar{a}+\bar{b}} = a \cdot b = f_{AND}(a,b)$$

Hence, NOR gate may be used to implement all three elementary operators.

Summary

Summary for all 2-input gates

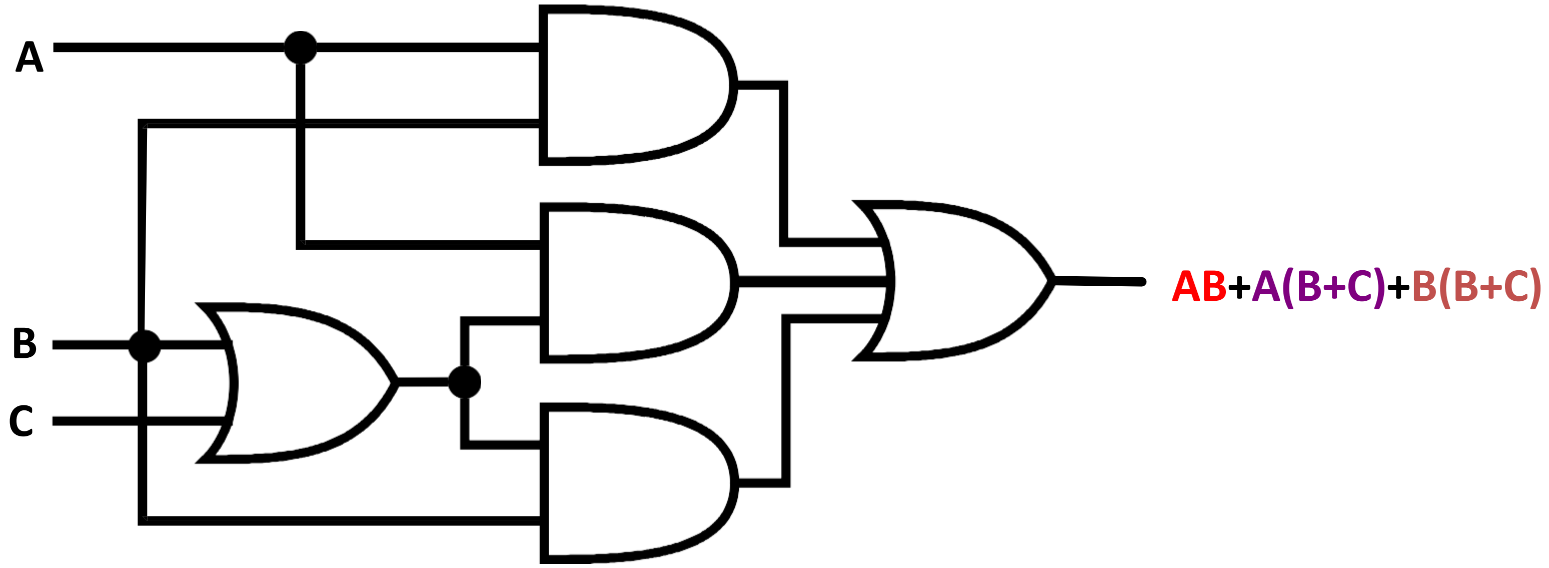
Inputs		Output of each gate					
A	B	AND	NAND	OR	NOR	XOR	XNOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

MINIMIZATON OF LOGIC EXPRESSION

- Goal -- minimize the cost of realizing a switching function
- Cost measures and other considerations
  - Number of gates
  - Number of levels
  - Gate fan in and/or fan out
  - Interconnection complexity
  - Preventing hazards
- Two-level realizations
  - Minimize the number of gates (terms in switching function)
  - Minimize the fan in (literals in switching function)
- Commonly used techniques
  - Boolean algebra postulates and theorems
  - Karnaugh maps

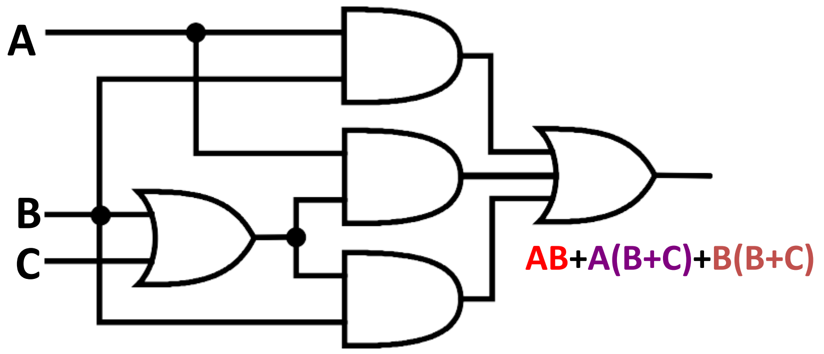
Simplification Using Boolean Algebra

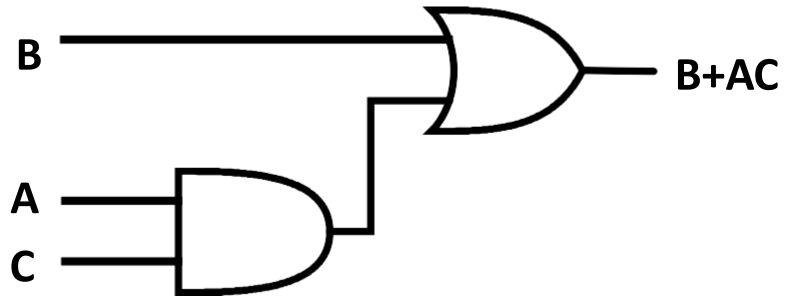
- A simplified Boolean expression uses the fewest gates possible to implement a given expression.



Simplification Using Boolean Algebra

- AB+A(B+C)+B(B+C)
  - (distributive law)
    - AB+AB+AC+BB+BC
  - (BB=B)
    - AB+AB+AC+**B**+BC
  - (AB+AB=AB)
    - **AB**+AC+B+BC
  - (B+BC=B)
    - AB+AC+**B**
  - (AB+B=B)
    - **B**+AC





Simplification Using Boolean Algebra

- Try these:

$$[\,AB\,(\bar{C} + BD) + A\bar{B}\,]\bar{C}$$

$$AB\bar{C} + ABC \overline{+} \bar{A}BC \overline{+} \bar{A}\bar{B}C + \bar{A}BC \;\; AB$$

$$+\, \overline{AC + AB}C \;\; \overline{}\,\overline{}$$

Standard Forms of Boolean Expressions

- All Boolean expressions, regardless of their form, can be converted into either of two standard forms:
  - The sum-of-products (SOP) form
  - The product-of-sums (POS) form
- Standardization makes the evaluation, simplification, and implementation of Boolean expressions much more systematic and easier.

The Sum-of-Products (SOP) Form

- An SOP expression → when two or more product terms are summed by Boolean addition.
  - Examples:

$AB + ABC$

$ABC + CDE + BCD \quad AB +$

$ABC + AC$

  - Also:

$A + ABC + BCD$

- In an SOP form, a single overbar cannot extend over more than one variable; however, more than one variable <u>in a term</u> can have an overbar:
  - example: $\bar{A}\bar{B}\bar{C}$ is OK!
  - **But not:** $\overline{ABC}$

Converting Product Terms to Standard SOP

- **Step 1:** Multiply each nonstandard product term by a term made up of the sum of a missing variable and its complement. This results in two product terms.

– As you know, you can multiply anything by 1 without changing its value.

- **Step 2:** Repeat step 1 until all resulting product term contains all variables in the domain in either complemented or uncomplemented form. In converting a product term to standard form, the number of product terms is doubled for each missing variable.

Converting Product Terms to Standard SOP (example)

- Convert the following Boolean expression into standard SOP form:

$$ABC + A\overline{B} + A\overline{B}\overline{C}D$$

$$ABC = ABC(D + D) = \boxed{ABCD + ABCD} \quad AB = AB(C + C) = ABC + ABC$$

$$\overline{A}\overline{B}C(D + \overline{D}) + \overline{A}\overline{B}\overline{C}(D + \overline{D}) = \boxed{\overline{A}\overline{B}CD + \overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}\overline{C}\overline{D}}$$

$$A\overline{B}C + \overline{A}\overline{B} + AB\overline{C}D = \boxed{A\overline{B}CD + A\overline{B}C\overline{D}} + \boxed{\overline{A}\overline{B}CD + \overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}\overline{C}\overline{D}} + AB\overline{C}D$$

The Product-of-Sums (POS) Form

- When two or more sum terms are multiplied, the result expression is a product- of-sums (POS):
  - Examples:

$(A + B)(A + B + C)$

$(A + B + C)(C + D + E)(B + C + D)$ $(A + B)(A + B + C)(A + C)$

- Also: $(A + B + C)(B + C + D)$

- In a POS form, a single overbar cannot extend over more than one variable; however, more than one variable in a term can have an overbar:

In a POS form, a single overbar cannot extend over more than one variable; however, more than one variable in a term can have an overbar:

example: $A + B + C$ is OK!

- example: is OK!

- **But not:** $\overline{A+B+C}$

Converting a Sum Term to Standard POS

- **Step 1:** Add to each nonstandard product term a term made up of the product of the missing variable and its complement. This results in two sum terms.
  - As you know, you can add 0 to anything without changing its value.
- **Step 2:** Apply rule → A+BC=(A+B)(A+C).
- **Step 3:** Repeat step 1 until all resulting sum terms contain all variable in the domain in either complemented or uncomplemented form.

Converting a Sum Term to Standard POS (example)

- Convert the following Boolean expression into standard POS form:

$$(A+\overline{B}+C)(\overline{B}+C+\overline{D})(A+\overline{B}+\overline{C}+D)$$

$$A+\overline{B}+C = A+\overline{B}+C+D\overline{D} = (A+\overline{B}+C+D)(A+\overline{B}+C+\overline{D})$$

$$\overline{B}+C+\overline{D} = \overline{B}+C+\overline{D}+A\overline{A} = (A+\overline{B}+C+\overline{D})(\overline{A}+\overline{B}+C+\overline{D})$$

$$(A+\overline{B}+C)(\overline{B}+C+\overline{D})(A+\overline{B}+\overline{C}+D) =$$

$$\boxed{(A+\overline{B}+C+D)(A+\overline{B}+C+\overline{D})}\boxed{(A+\overline{B}+C+\overline{D})(\overline{A}+\overline{B}+C+\overline{D})}\boxed{(A+\overline{B}+\overline{C}+D)}$$

Boolean Expressions & Truth Tables

- All <u>standard Boolean expression </u>can be easily converted into truth table format using binary values for each term in the expression.
- Also, <u>standard SOP or POS </u>expression can be determined from the truth table.

Converting SOP Expressions to Truth Table Format

- Recall the fact:
  - An SOP expression is equal to 1 only if at least one of the product term is equal to 1.
- Constructing a truth table:
  - **Step 1:** List all possible combinations of binary values of the variables in the expression.
  - **Step 2:** Convert the SOP expression to standard form if it is not already.
  - **Step 3:** Place a 1 in the output column (X) for each binary value that makes the <u>standard SOP</u> expression a 1 and place 0 for all the remaining binary values.

Converting SOP Expressions to Truth Table Format (example)

- Develop a truth table for the standard SOP expression

$$\bar{A}\bar{B}C + A\bar{B}\bar{C} + ABC$$

Inputs			Output	Product Term
A	B	C	X	
0	0	0	0	
0	0	1	1	$\bar{A}\bar{B}C$
0	1	0	0	
0	1	1	0	
1	0	0	1	$A\bar{B}\bar{C}$
1	0	1	0	
1	1	0	0	
1	1	1	1	ABC

Converting POS Expressions to Truth Table Format

- Recall the fact:
  - A POS expression is equal to 0 only if at least one of the product term is equal to 0.
- Constructing a truth table:
  - **Step 1:** List all possible combinations of binary values of the variables in the expression.
  - **Step 2:** Convert the POS expression to standard form if it is not already.
  - **Step 3:** Place a 0 in the output column (X) for each binary value that makes the standard POS expression a 0 and place 1 for all the remaining binary values.

Converting POS Expressions to Truth Table Format (example)

- Develop a truth table for the standard SOP expression

$$(A+B+C)(A+\overline{B}+C)(A+\overline{B}+\overline{C})(\overline{A}+B+\overline{C})(\overline{A}+\overline{B}+C)$$

Inputs			Output	Product Term
A	B	C	X	
0	0	0	0	$(A+B+C)$
0	0	1	1	
0	1	0	0	$(A+\overline{B}+C)$
0	1	1	0	$(A+\overline{B}+\overline{C})$
1	0	0	1	
1	0	1	0	$(\overline{A}+B+\overline{C})$
1	1	0	0	$(\overline{A}+\overline{B}+C)$
1	1	1	1	

Determining Standard Expression from a Truth Table

- To determine the standard **SOP expression** represented by a truth table.
- Instructions:
  - **Step 1:** List the binary values of the input variables for which the output is 1.
  - **Step 2:** Convert each binary value to the corresponding product term by replacing:
    - each 1 with the corresponding variable, and
    - each 0 with the corresponding variable complement.
- Example: 1010 → $A\overline{B}C\overline{D}$

Determining Standard Expression from a Truth Table

- To determine the standard **POS expression** represented by a truth table.
- Instructions:
  - **Step 1:** List the binary values of the input variables for which the output is 0.
  - **Step 2:** Convert each binary value to the corresponding product term by replacing:
    - each 1 with the corresponding variable complement, and
    - each 0 with the corresponding variable.
- Example: 1001 → $\overline{A}+B+C+\overline{D}$

The Karnaugh Map

- Feel a little difficult using Boolean algebra laws, rules, and theorems to simplify logic?
- A K-map provides a systematic method for simplifying Boolean expressions and, if properly used, will produce the simplest SOP or POS expression possible, known as the minimum expression.

What is K-Map

- It’s similar to truth table; instead of being organized (i/p and o/p) into columns and rows, the K-map is an array of cells in which each cell represents a binary value of the input variables.
- The cells are arranged in a way so that simplification of a given expression is simply a matter of properly grouping the cells.
- K-maps can be used for expressions with 2, 3, 4, and 5 variables.

The 3 Variable K-Map

- There are 8 cells as shown:

AB \ C	0	1
00	$\overline{A}\,\overline{B}\,\overline{C}$	$\overline{A}\,\overline{B}C$
01	$\overline{A}B\overline{C}$	$\overline{A}BC$
11	$AB\overline{C}$	ABC
10	$A\overline{B}\,\overline{C}$	$A\overline{B}C$

Or

A \ BC	00	01	11	10
0	$\overline{A}\,\overline{B}\,\overline{C}$	$\overline{A}\,\overline{B}C$	$\overline{A}BC$	$\overline{A}B\overline{C}$
1	$A\overline{B}\,\overline{C}$	$A\overline{B}C$	ABC	$AB\overline{C}$

The 4-Variable K-Map

AB \ CD	00	01	11	10
00	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}D$	$\bar{A}\bar{B}CD$	$\bar{A}\bar{B}C\bar{D}$
01	$\bar{A}B\bar{C}\bar{D}$	$\bar{A}B\bar{C}D$	$\bar{A}BCD$	$\bar{A}BC\bar{D}$
11	$AB\bar{C}\bar{D}$	$AB\bar{C}D$	$ABCD$	$ABC\bar{D}$
10	$A\bar{B}\bar{C}\bar{D}$	$A\bar{B}\bar{C}D$	$A\bar{B}CD$	$A\bar{B}C\bar{D}$

K-Map SOP Minimization

- The K-Map is used for simplifying Boolean expressions to their minimal form.
- A minimized SOP expression contains the fewest possible terms with fewest possible variables per term.
- Generally, a minimum SOP expression can be implemented with fewer logic gates than a standard expression.

Karnaugh Maps (K-maps)

- If m_i is a minterm of *f*, then place a 1 in cell *i* of

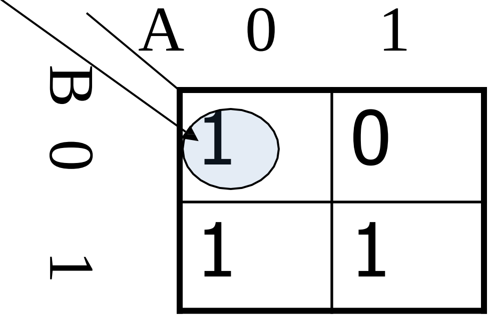
the K-map.

- If M_i is a maxterm of *f*, then place a 0 in cell *i.*

- If d_i is a don't care of *f*, then place a *d or x* in cell *i.*

Examples

- *Two variable K-map*
  f(A,B)=$\sum$*m*(0,1,3)=A`B`+A`B+AB



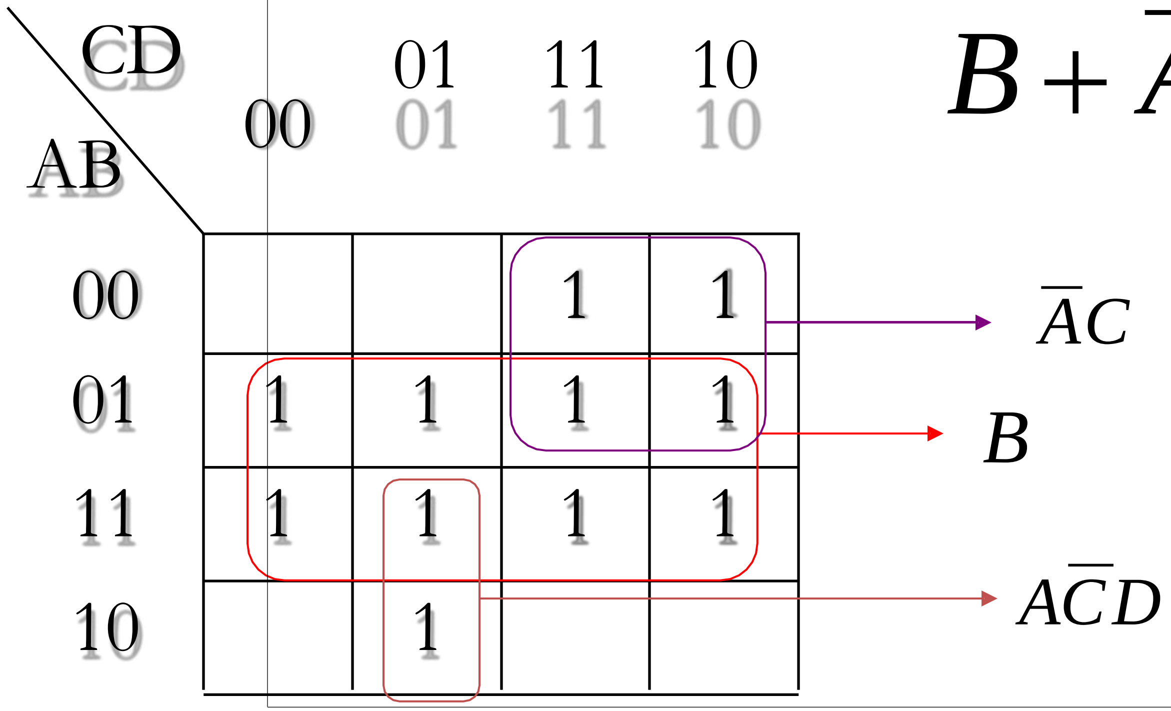
Grouping the 1s (rules)

1. A group must contain either 1,2,4,8,or 16 cells (depending on number of variables in the expression)
2. Each cell in a group must be adjacent to one or more cells in that same group, but all cells in the group do not have to be adjacent to each other.
3. Always include the largest possible number of 1s in a group in accordance with rule 1.
4. Each 1 on the map must be included in at least one group. The 1s already in a group can be included in another group as long as the overlapping groups include noncommon 1s.

Determining the Minimum SOP Expression from the Map

2. Determine the minimum product term for each group.
   - For a 3-variable map:
     1. A 1-cell group yields a 3-variable product term
     2. A 2-cell group yields a 2-variable product term
     3. A 4-cell group yields a 1-variable product term
     4. An 8-cell group yields a value of 1 for the expression.
   - For a 4-variable map:
     1. A 1-cell group yields a 4-variable product term
     2. A 2-cell group yields a 3-variable product term
     3. A 4-cell group yields a 2-variable product term
     4. An 8-cell group yields a a 1-variable product term
     5. A 16-cell group yields a value of 1 for the expression.

Determining the Minimum SOP Expression from the Map (example)



$$B+\overline{A}C+A\overline{C}D$$

Three-Variable K-Maps

$f=\sum(0,4)=\overline{B}\,\overline{C}$

A \ BC	00	01	11	10
0	**1**	**0**	**0**	**0**
1	**1**	**0**	**0**	**0**

$f=\sum(4,5)=A\overline{B}$

A \ BC	00	01	11	10
0	**0**	**0**	**0**	**0**
1	**1**	**1**	**0**	**0**

$f=\sum(0,1,4,5)=\overline{B}$

A \ BC	00	01	11	10
0	**1**	**1**	**0**	**0**
1	**1**	**1**	**0**	**0**

$f=\sum(0,1,2,3)=\overline{A}$

A \ BC	00	01	11	10
0	**1**	**1**	**1**	**1**
1	**0**	**0**	**0**	**0**

$f=\sum(0,4)=\overline{A}C$

A \ BC	00	01	11	10
0	**0**	**1**	**1**	**0**
1	**0**	**0**	**0**	**0**

$f=\sum(4,6)=A\overline{C}$

A \ BC	00	01	11	10
0	**0**	**0**	**0**	**0**
1	**1**	**0**	**0**	**1**

$f=\sum(0,2)=\overline{A}\,\overline{C}$

A \ BC	00	01	11	10
0	**1**	**0**	**0**	**1**
1	**0**	**0**	**0**	**0**

$f=\sum(0,2,4,6)=\overline{C}$

A \ BC	00	01	11	10
0	**1**	**0**	**0**	**1**
1	**1**	**0**	**0**	**1**

Three-Variable K-Map Examples

- We can write any way either AB and C or A BC

A \ BC	00	01	11	10
0		1		
1	1		1	1

A \ BC	00	01	11	10
0	1		1	1
1	1			1

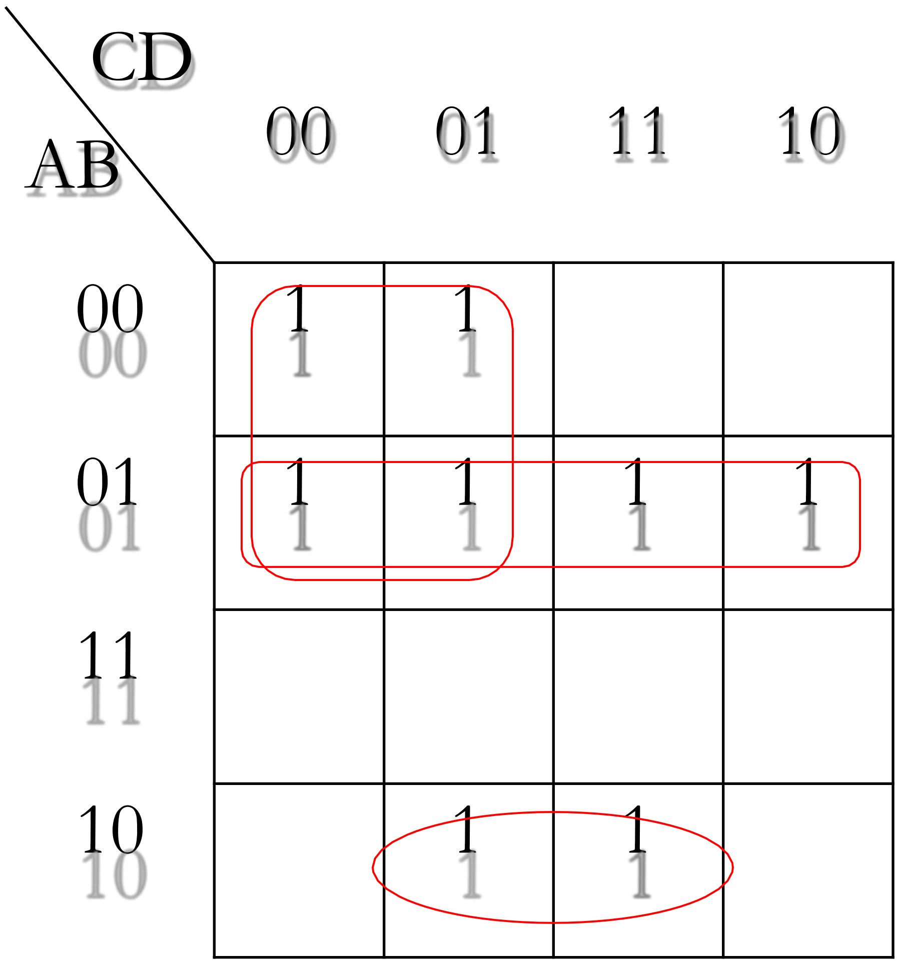
A \ BC	00	01	11	10
0			1	1
1	1	1		

A \ BC	00	01	11	10
0			1	
1	1		1	1

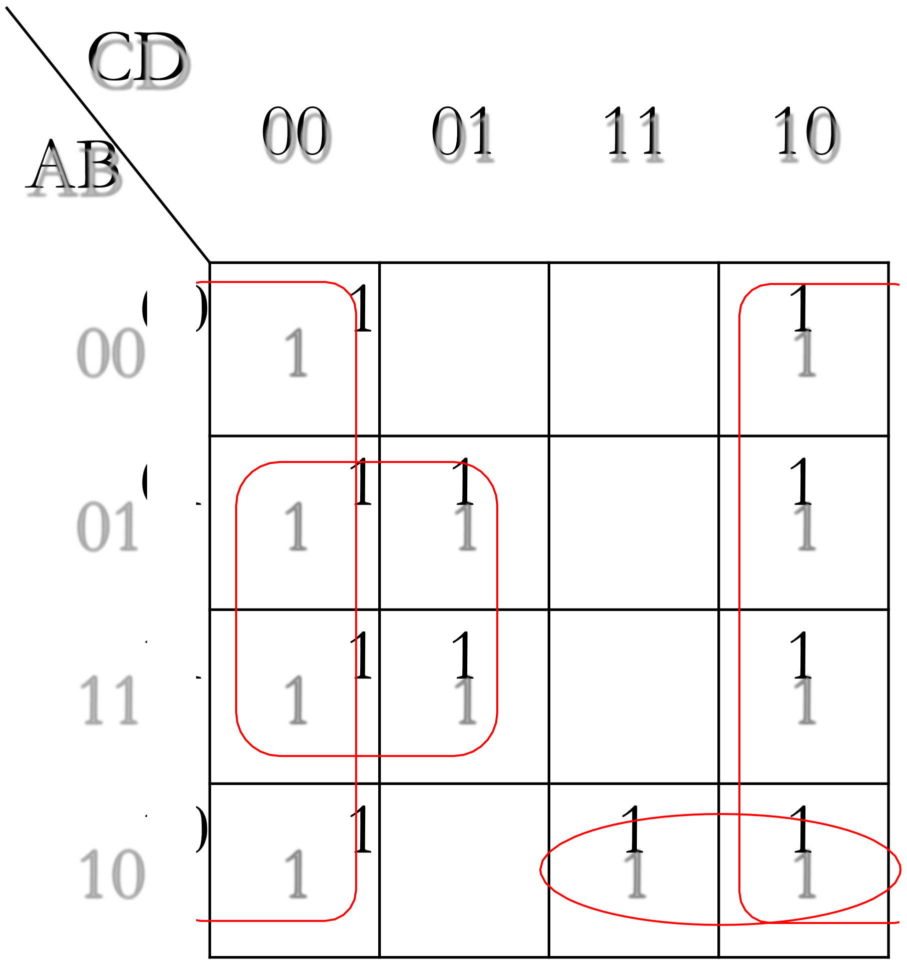
A \ BC	00	01	11	10
0		1	1	1
1		1	1	

A \ BC	00	01	11	10
0				
1				

Determining the Minimum SOP Expression from the Map (exercises)



$\overline{A}B + \overline{A}\overline{C} + A\overline{B}D$



$\overline{D} + A\overline{B}C + B\overline{C}$

Four-Variable K-Maps

AB \ CD	00	01	11	10
00	**0**	**0**	**0**	**0**
01	**1**	**1**	**1**	**1**
11	**0**	**0**	**0**	**0**
10	**0**	**0**	**0**	**0**

$f = \sum(4,5,6,7) = \overline{A} \bullet B$

AB \ CD	00	01	11	10
00	**0**	**0**	**1**	**0**
01	**0**	**0**	**1**	**0**
11	**0**	**0**	**1**	**0**
10	**0**	**0**	**1**	**0**

$f = \sum(3, 7,11,15) = C \bullet D$

AB \ CD	00	01	11	10
00	**1**	**0**	**1**	**0**
01	**0**	**1**	**0**	**1**
11	**1**	**0**	**1**	**0**
10	**0**	**1**	**0**	**1**

$f = \sum(0,3,5,6,9,10,12,15)$

$f = A \otimes B \otimes C \otimes D$

AB \ CD	00	01	11	10
00	**0**	**1**	**0**	**1**
01	**1**	**0**	**1**	**0**
11	**0**	**1**	**0**	**1**
10	**1**	**0**	**1**	**0**

$f = \sum(1,2,4,7,8,11,13,14)$

$f = A \oplus B \oplus C \oplus D$

AB \ CD	00	01	11	10
00	**0**	**1**	**1**	**0**
01	**0**	**1**	**1**	**0**
11	**0**	**1**	**1**	**0**
10	**0**	**1**	**1**	**0**

$f = \sum(1,3,5,7,9,11,13,15)$

$f = D$

AB \ CD	00	01	11	10
00	**1**	**0**	**0**	**1**
01	**1**	**0**	**0**	**1**
11	**1**	**0**	**0**	**1**
10	**1**	**0**	**0**	**1**

$f = \sum(0,2,4,6,8,10,12,14)$

$f = \overline{D}$

AB \ CD	00	01	11	10
00	**0**	**0**	**0**	**0**
01	**1**	**1**	**1**	**1**
11	**1**	**1**	**1**	**1**
10	**0**	**0**	**0**	**0**

$f = \sum(4,5,6,7,12,13,14,15)$

$f = B$

AB \ CD	00	01	11	10
00	**1**	**1**	**1**	**1**
01	**0**	**0**	**0**	**0**
11	**0**	**0**	**0**	**0**
10	**1**	**1**	**1**	**1**

$f = \sum(0,1,2,3,8,9,10,11)$

$f = \overline{B}$

Practicing K-Map (SOP)

$$A\overline{B}C + \overline{A}BC + \overline{A}\,\overline{B}C + \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}\,\overline{C}$$

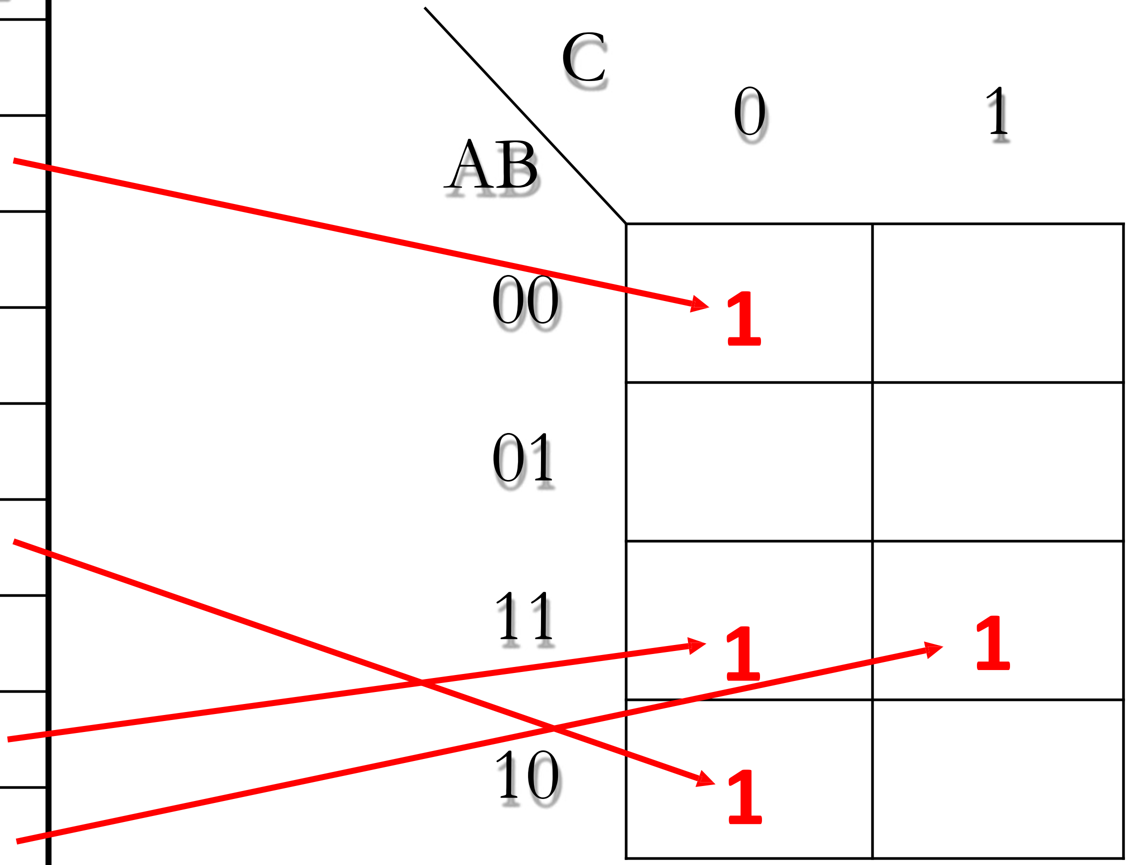
$$\overline{B} + \overline{A}C$$

$$\overline{B}\,\overline{C}\,\overline{D} + \overline{A}B\overline{C}\,\overline{D} + AB\overline{C}\,\overline{D} + \overline{A}\,\overline{B}CD + A\overline{B}CD +$$
$$\overline{A}\,\overline{B}C\overline{D} + \overline{A}BC\overline{D} + ABC\overline{D} + A\overline{B}C\overline{D}$$

$$\overline{D} + \overline{B}C$$

Mapping Directly from a Truth Table

I/P			O/P
A	B	C	X
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

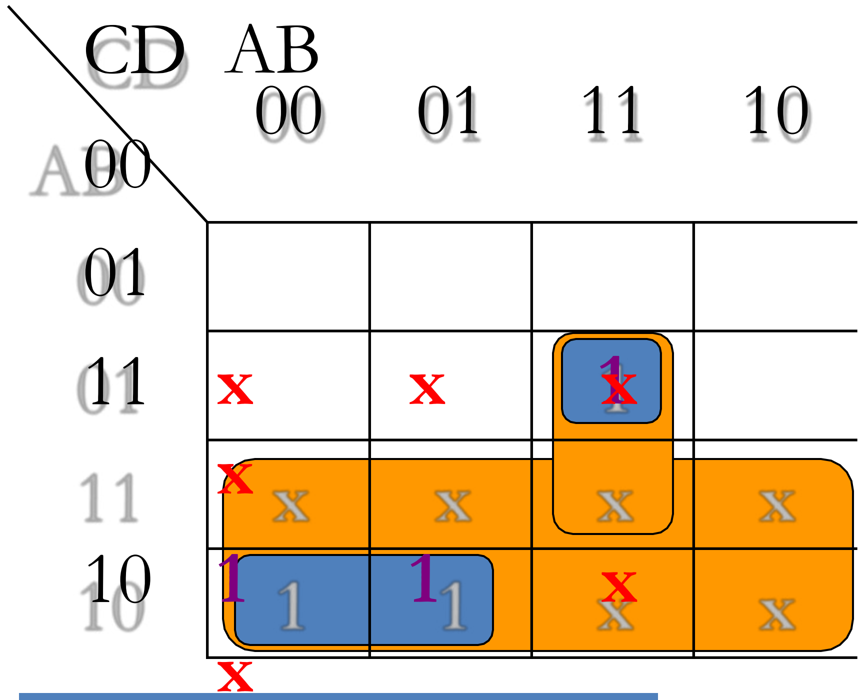


“Don’t Care” Conditions

- Sometimes a situation arises in which some input variable combinations are not allowed, i.e. BCD code:
  - There are six invalid combinations: 1010, 1011, 1100, 1101, 1110, and 1111.
- Since these unallowed states will never occur in an application involving the BCD code → they can be treated as “don’t care” terms with respect to their effect on the output.
- The “don’t care” terms can be used to advantage on the K-map (how? see the next slide).

“Don’t Care” Conditions

INPUTS				O/P
A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	x
1	0	1	1	x
1	1	0	0	x
1	1	0	1	x
1	1	1	0	x
1	1	1	1	x



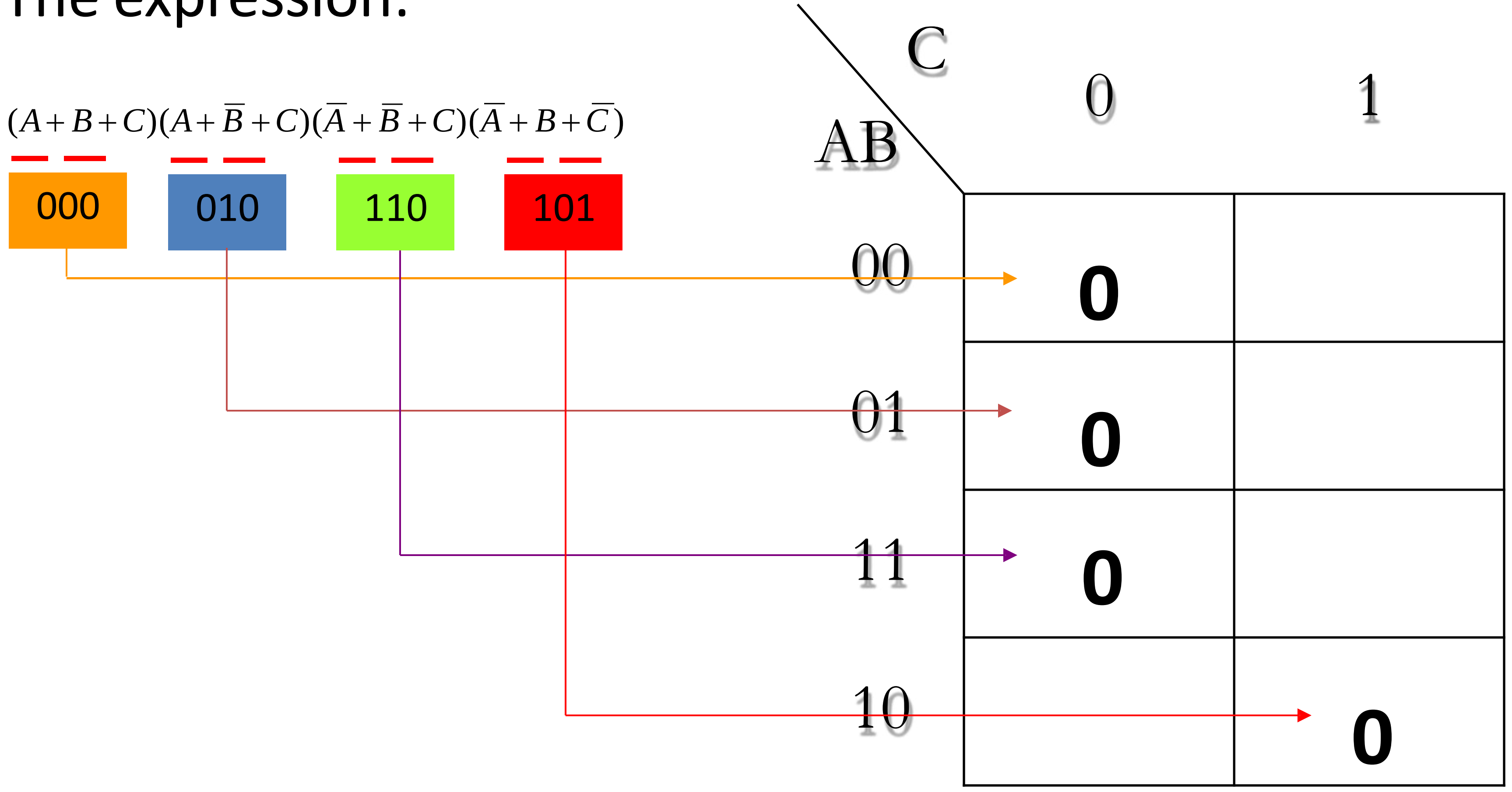
Without “don’t care”
$Y = ABC + ABCD$

With “don’t care”
$Y = A + BCD$

Mapping a Standard POS Expression (full example)

The expression:

$$(A+B+C)(A+\overline{B}+C)(\overline{A}+\overline{B}+C)(\overline{A}+B+\overline{C})$$



Combinational Circuits

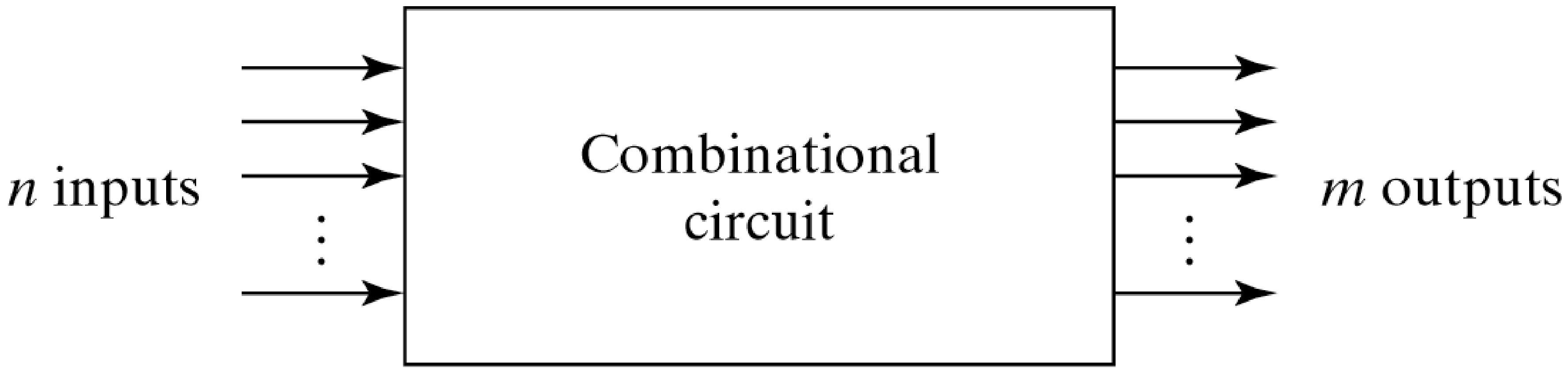


Fig. 4-1 Block Diagram of Combinational Circuit

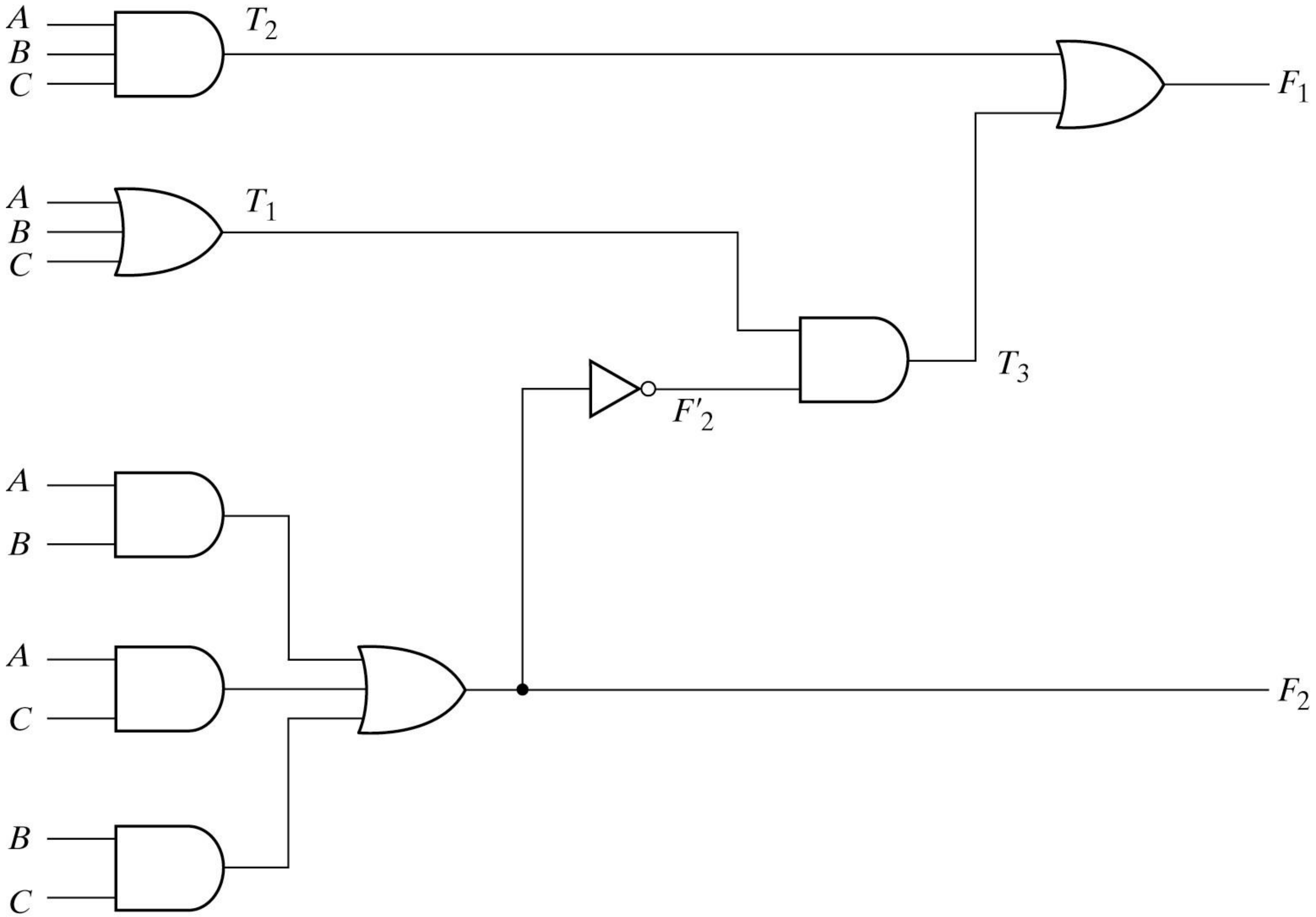


Fig. 4-2 Logic Diagram for Analysis Example

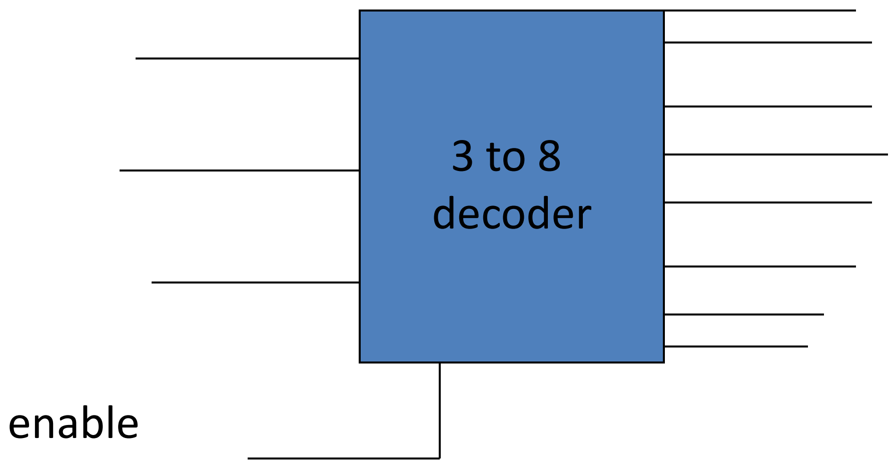
Designing Combinational Circuits

In general we have to do following steps:

1. Problem description
2. Input/output of the circuit
3. Define truth table
4. Simplification for each output
5. Draw the circuit

Decoder

- Is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines For example if the number of input is n=3 the number of output lines can be m=2^3 . It is also known as 1 of 8 because one output line is selected out of 8 available lines:



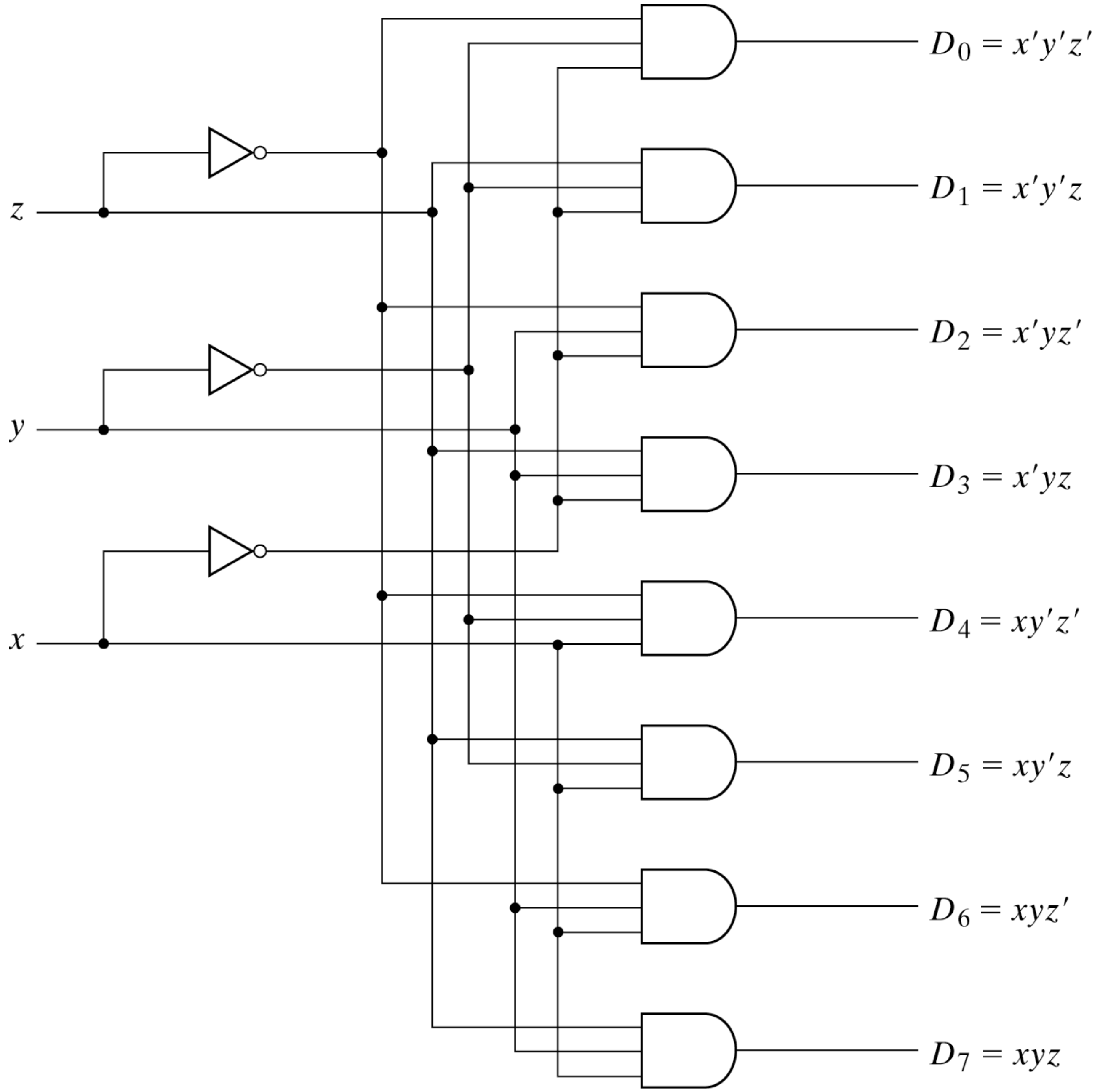


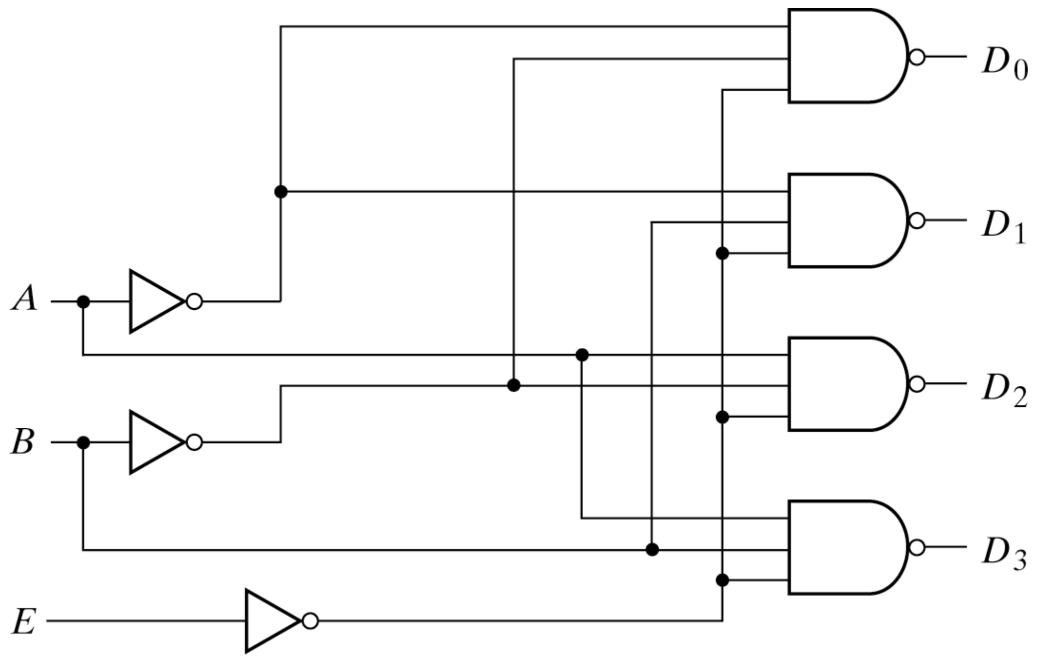
Fig. 4-18 3-to-8-Line Decoder

Decoder with Enable Line

- Decoders usually have an enable line,
- If enable=0 , decoder is off. It means all output lines are zero
- If enable=1, decoder is on and depending on input, the corresponding output line is 1, all other lines are 0
- See the truth table in next slide

Truth table for decoder

E	a2	a1	a0	D7	D6	D5	D4	D3	D2	D1	D0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1											
1	………………………………………										
1	……………………………………										
1											
1											
1	1	1	1	1	0	0	0	0	0	0	0



(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

(b) Truth table

Fig. 4-19 2-to-4-Line Decoder with Enable Input

Major application of Decoder

- Decoder is use to implement any combinational cicuits (f^n)

For example the truth table for full adder is s (x,y,z) = ∑ (1,2,4,7)

and C(x,y,z)= ∑ (3,5,6,7). The implementation with decoder is:

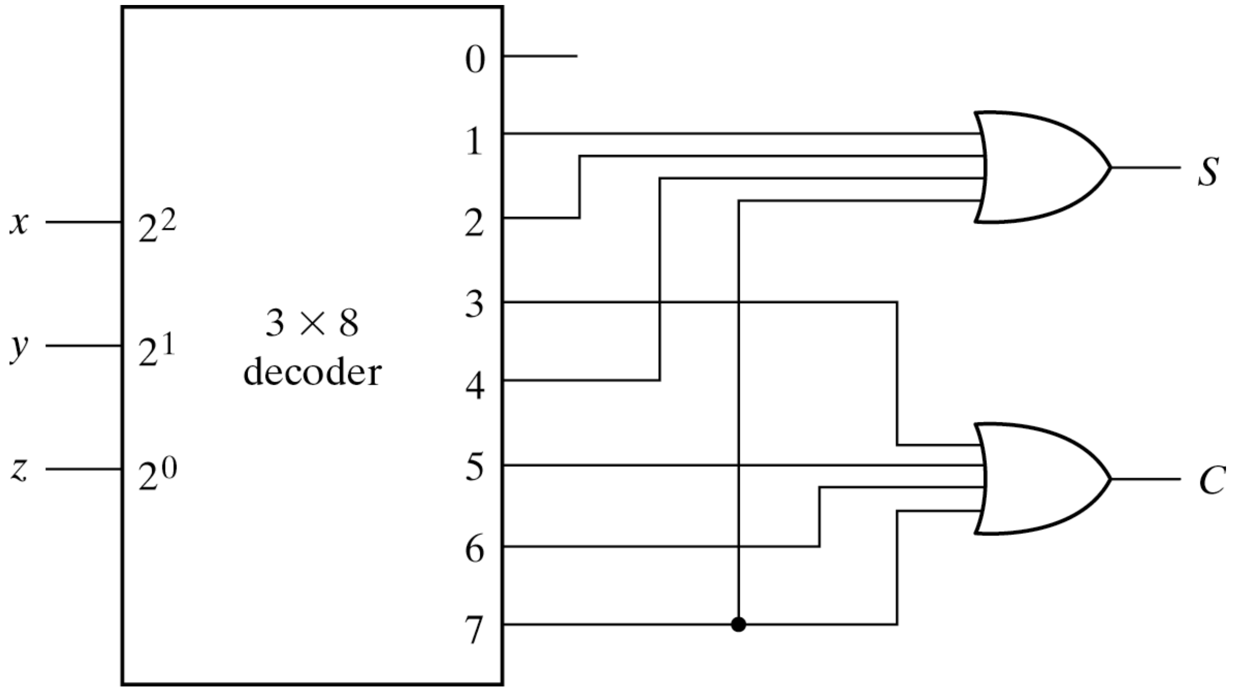
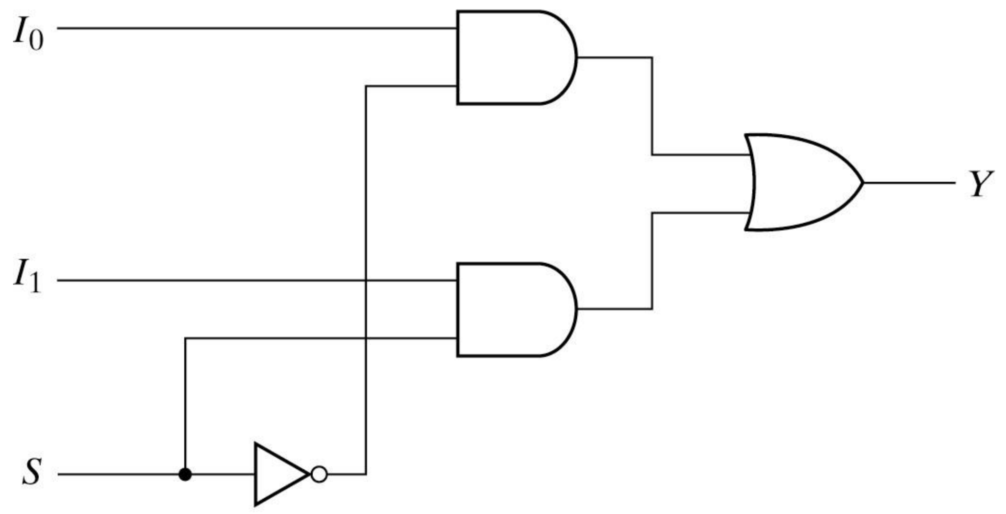


Fig. 4-21 Implementation of a Full Adder with a Decoder

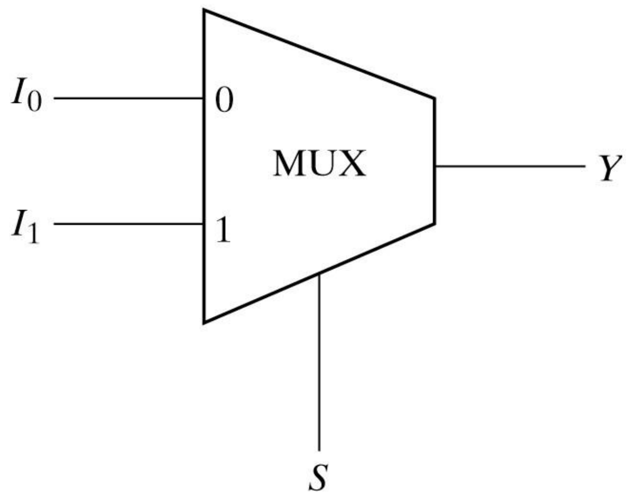
Multiplexer

- It is a combinational circuit that selects binary information from one of the input lines and directs it to a single output line
- Usually there are 2^n input lines and n selection lines whose bit combinations determine which input line is selected
- For example for 2-to-1 multiplexer if selection S is zero then I_0 has the path to output and if S is one I_1 has the path to output (see the next slide)

2-to-1 multiplexer

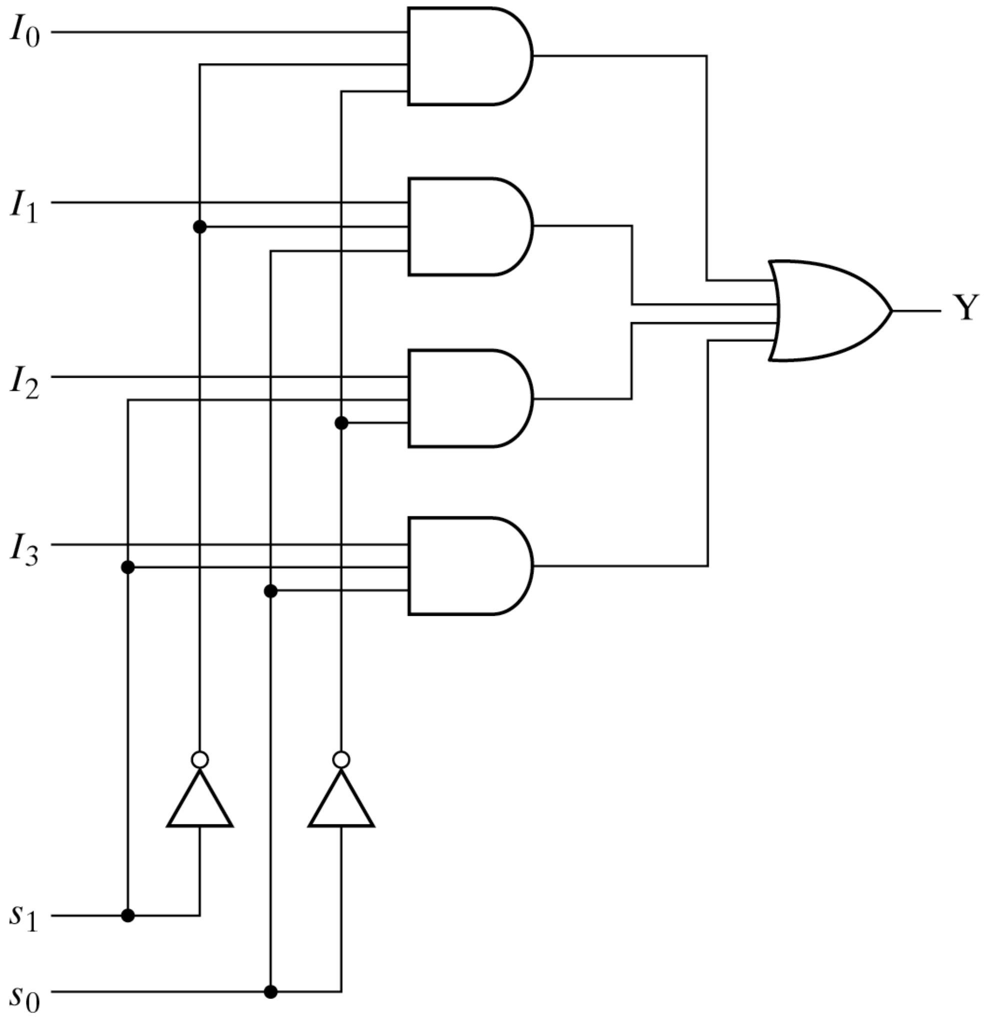


(a) Logic diagram



(b) Block diagram

Fig. 4-24 2-to-1-Line Multiplexer



(a) Logic diagram

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table

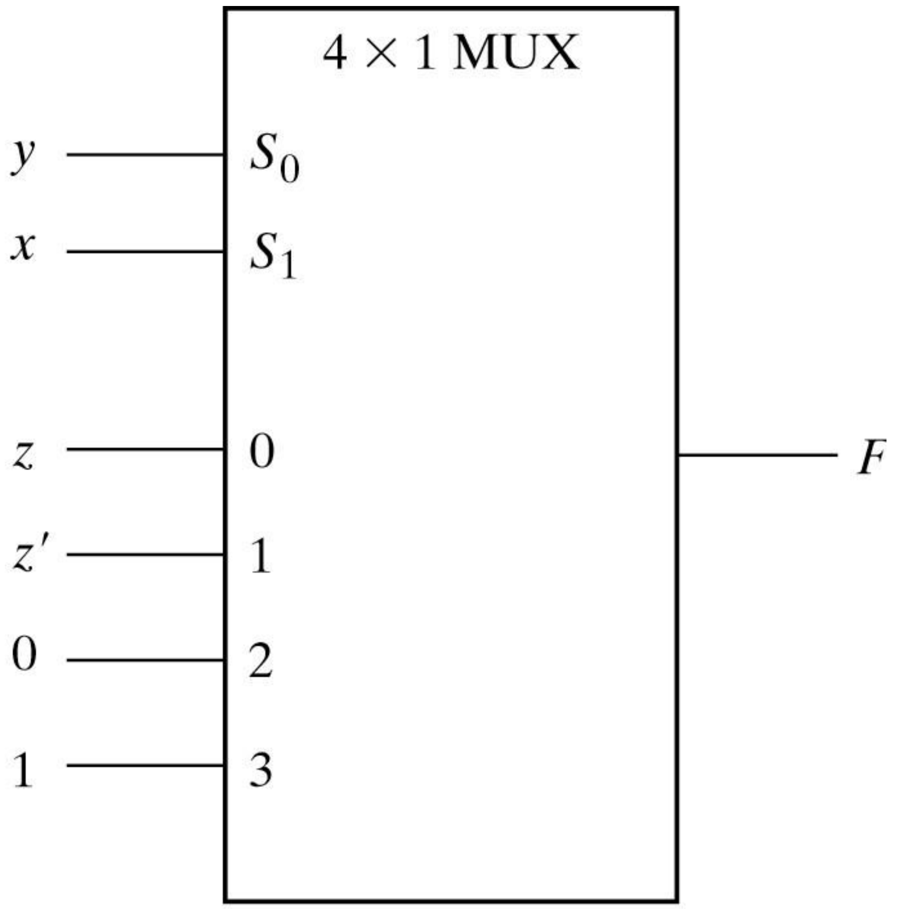
Fig. 4-25 4-to-1-Line Multiplexer

Boolean function Implementation

- Another method for implementing boolean function is using multiplexer
- For doing that assume boolean function has n variables. We have to use multiplexer with n-1 selection lines and
- 1- first n-1 variables of function is used for data input
- 2- the remaining single variable (named z)is used for data input. Each data input can be z, z’, 1 or 0. From truth table we have to find the relation of F and z to be able to design input lines. For example : $f(x,y,z) = \sum(1,2,6,7)$

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table



(b) Multiplexer implementation

Fig. 4-27 Implementing a Boolean Function with a Multiplexer

$F\ A,B,C,D = \sum(1,3,4,11,12,13,14,15)$

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

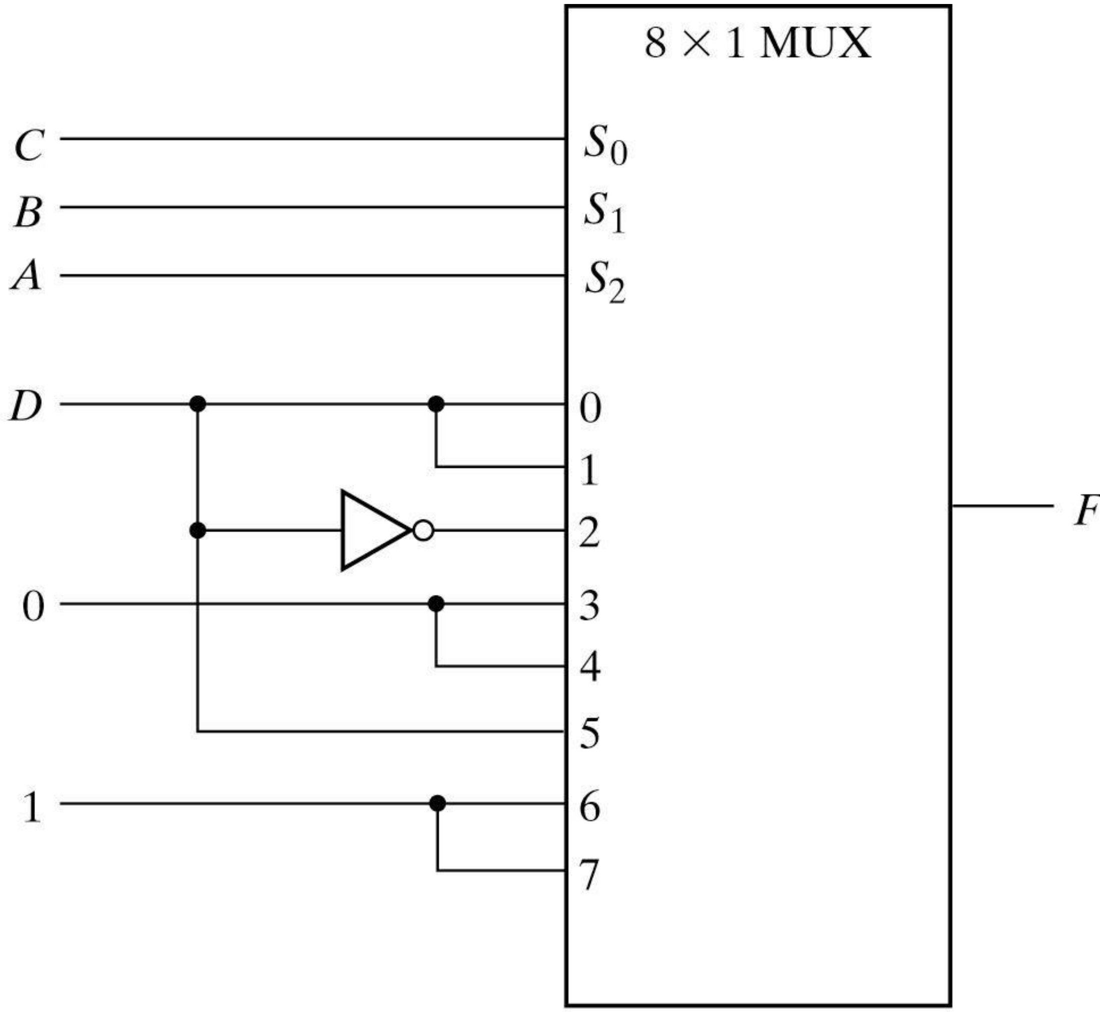
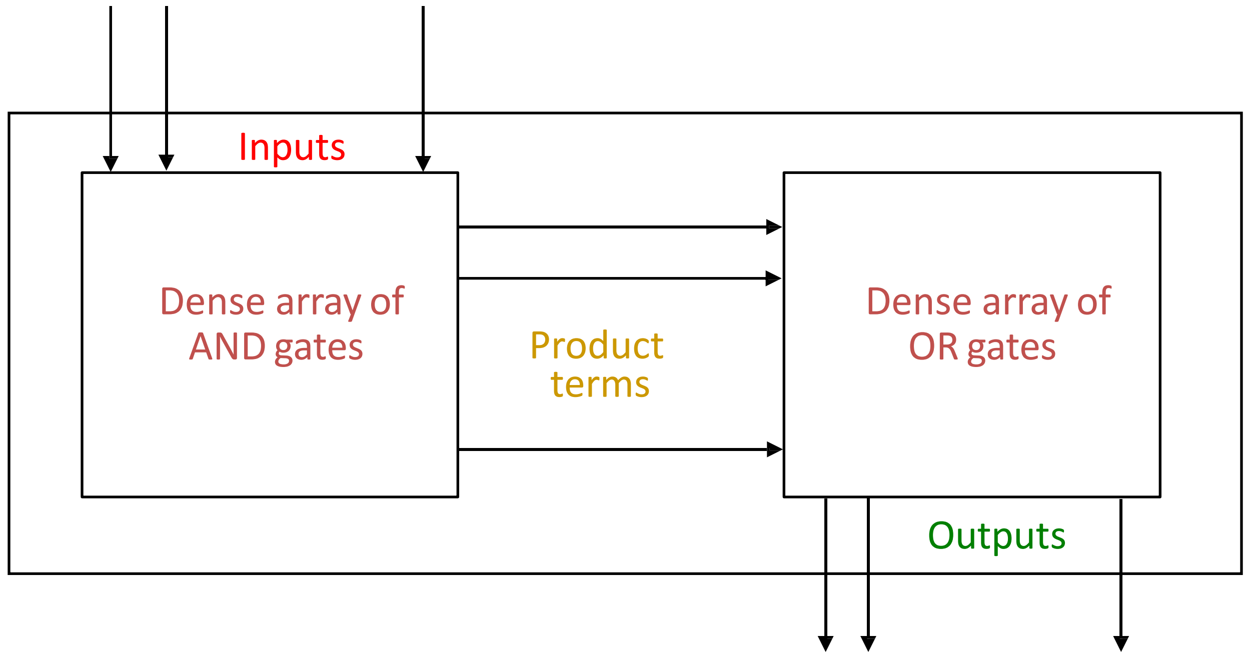


Fig. 4-28 Implementing a 4-Input Function with a Multiplexer

Prgrammable Logic Organization

- **Pre-fabricated building block of many AND/OR gates (or NOR, NAND)**
- **"Personalized" by making or breaking connections among the gates**



Programmable Array Block Diagram for Sum of ProductsForm

Basic Programmable Logic Organizations

- Depending on which of the AND/OR logic arrays is programmable, we have three basic organizations

ORGANIZATION	AND ARRAY	OR ARRAY
PAL	PROG.	FIXED
PROM	FIXED	PROG.
PLA	PROG.	PROG.

PLA Logic Implementation

Key to Success: Shared Product Terms

Equations

Example:

$F0 = A + \underline{B}\,\overline{C}\ \overline{\ }$
$F1 = A\ \underline{C + }A\ B$
$F2 = B\ \underline{C}+ A\ B$
$F3 = B\ C\ + A$

Personality Matrix

Product term	Inputs A	B	C	Outputs F0	F1	F2	F3
A B	1	1	-	0	1	1	0
B C	-	0	1	0	0	0	1
A C	1	-	0	0	1	0	0
B C	-	0	0	1	0	1	0
A	1	-	-	1	0	0	1

Reuse of terms

Input Side:

1 = asserted in term
0 = negated in term
- = does not participate

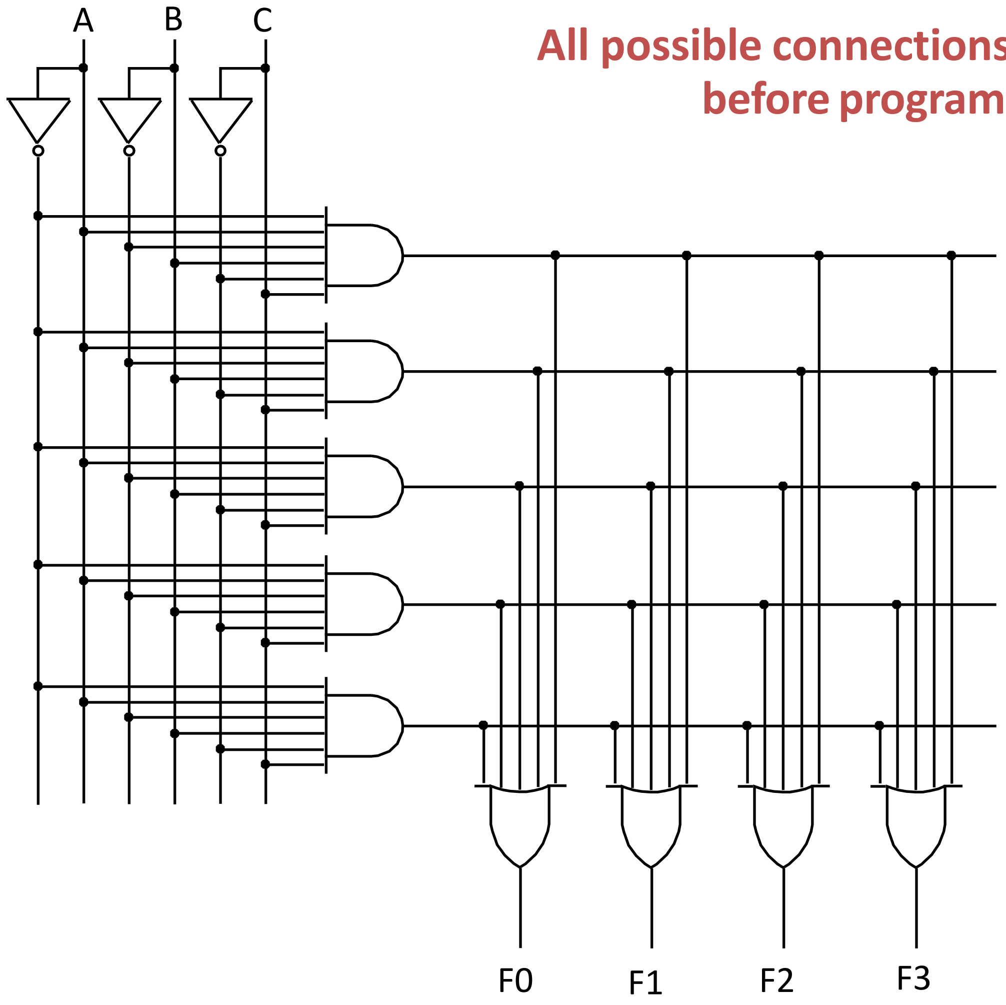
Output Side:

1 = term connected to output 0
= no connection to output

PLA Logic Implementation

Example Continued - Unprogrammed device

All possible connections are available before programming



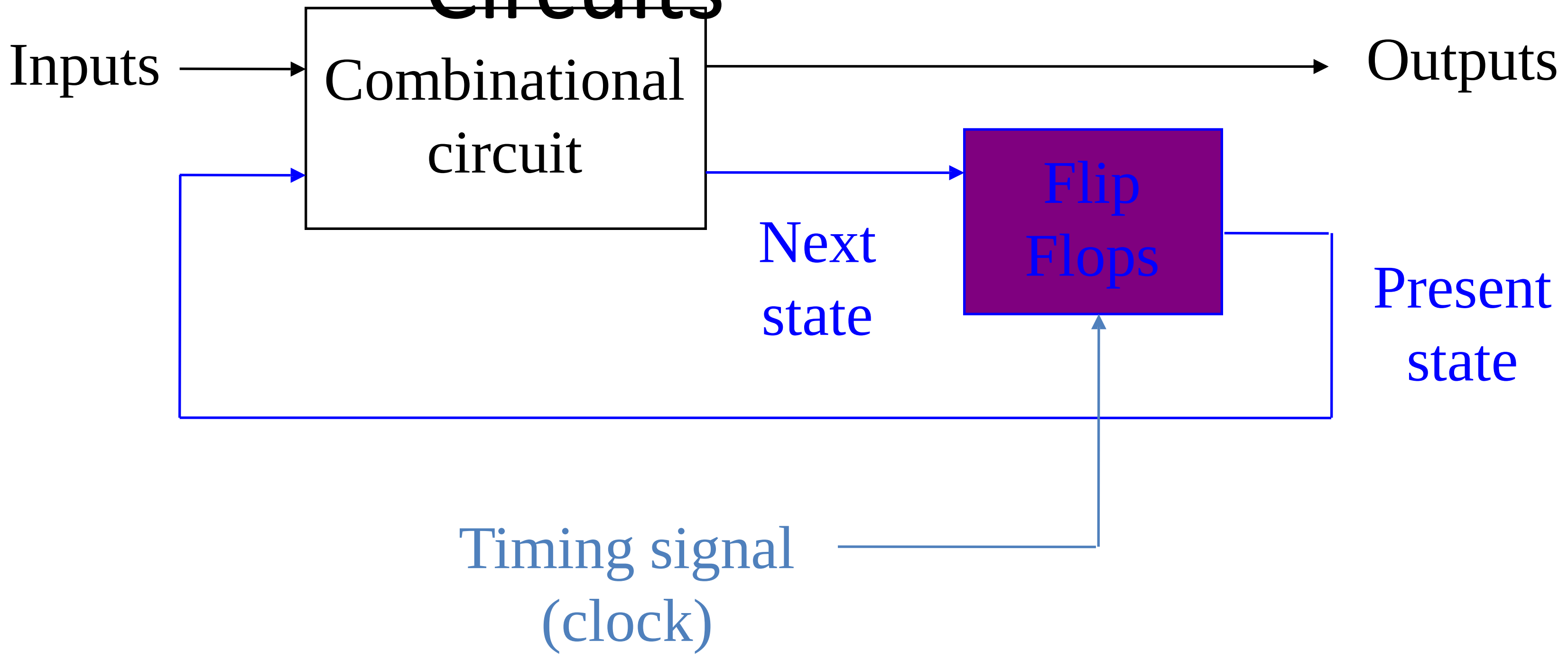
Sequential Circuits

- Circuits require memory to store intermediate data
- Sequential circuits use a periodic signal to determine when to store values.
  - A clock signal can determine storage times
  - Clock signals are periodic
- Single bit storage element is a flip flop
- A basic type of flip flop is a latch
- Latches are made from logic gates
  - NAND, NOR, AND, OR, Inverter

The story so far …

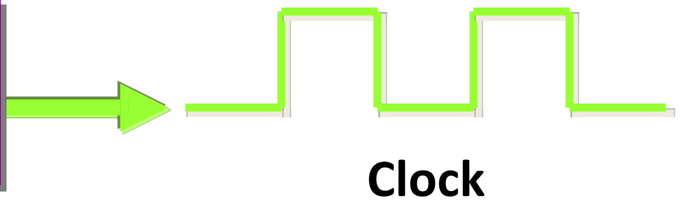
- Logical operations which respond to **combinations** of inputs to produce an output.
  - Call these **combinational logic** circuits.
- For example, can add two numbers. But:
  - No way of adding two numbers, then adding a third (a **sequential** operation);
  - No way of remembering or storing information after inputs have been removed.
- To handle this, we need **sequential logic** capable of storing intermediate (and final) results.

Sequential Circuits



Clock

a periodic external event (input)

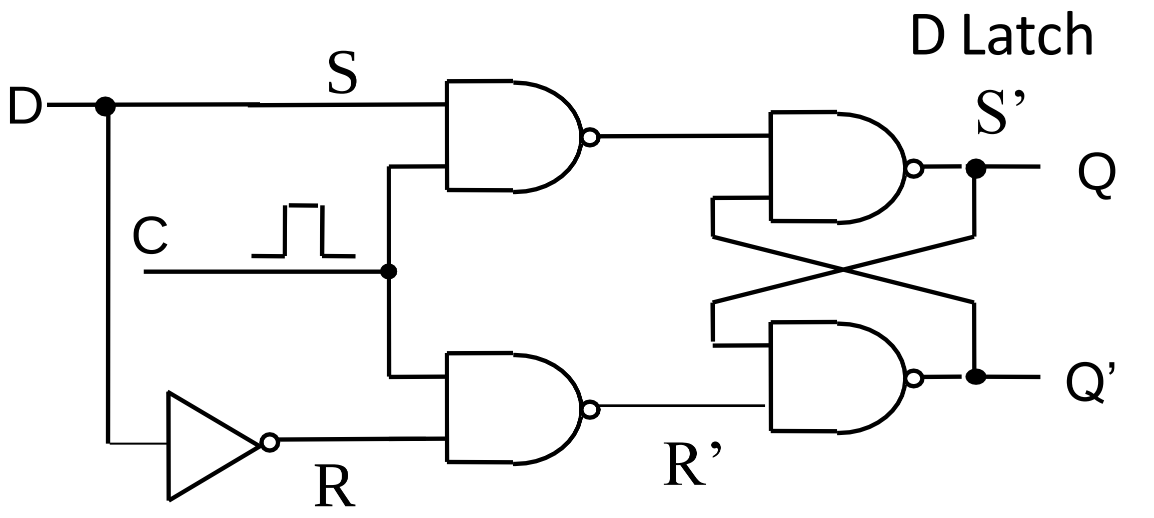


synchronizes when current state changes happen
keeps system well-behaved
makes it easier to design and build large systems

Sequential Circuits: Flip flops

Overview

- Latches respond to trigger levels on control inputs
  - Example: If G = 1, input reflected at output
- Difficult to precisely time when to store data with latches
- Flip flips store data on a rising or falling trigger edge.
  - Example: control input transitions from 0 -> 1, data input appears at output
  - Data remains stable in the flip flop until until next rising edge.
- Different types of flip flops serve different functions
- Flip flops can be defined with characteristic functions.



D	C	Q	Q'
0	1	0	1
1	1	1	0
X	0	Q_0	Q_0'

S	R	C	Q	Q'	
0	0	1	Q_0	Q_0'	Store
0	1	1	0	1	Reset
1	0	1	1	1	Set
1	1	1	1	2	Disallowed
X	X	0	Q_0	Q_0'	Store

- When C is high, D passes from input to output (Q)

Master-Slave D Flip Flop

- Consider two latches combined together
- Only one *C* value active at a time
- Output changes on falling edge of the clock

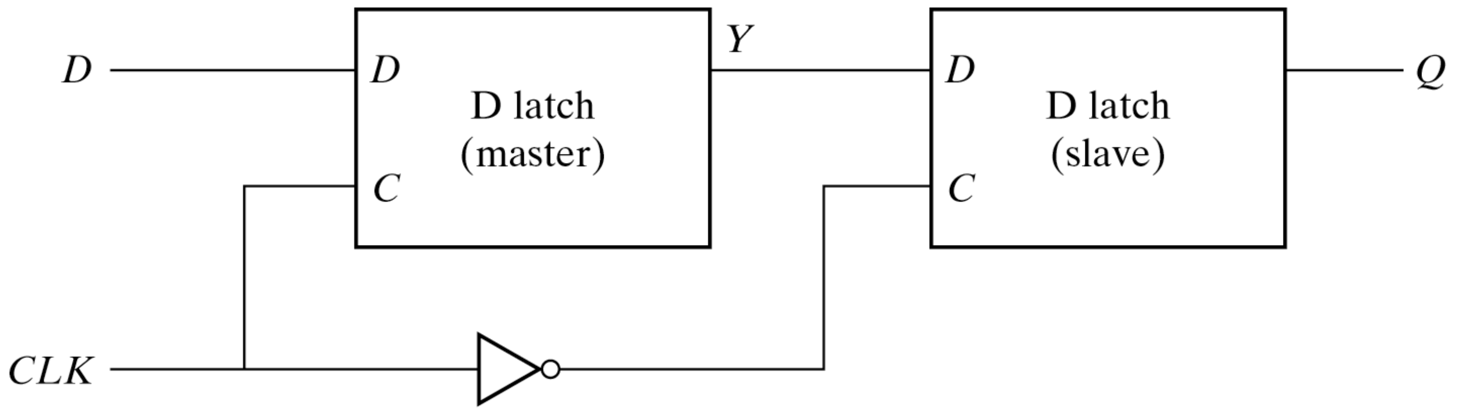
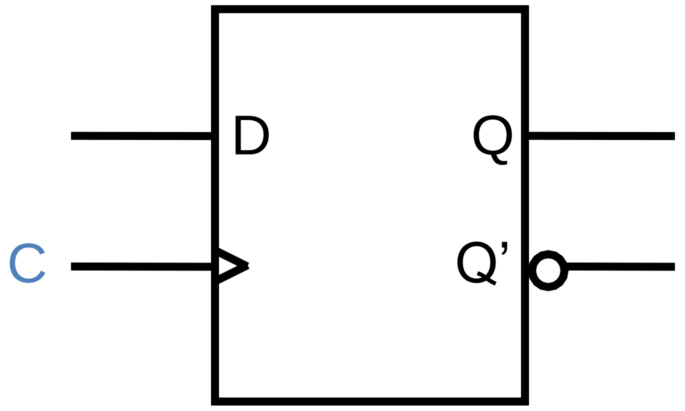


Fig. 5-9 Master-Slave *D* Flip-Flop

D Flip-Flop

- Stores a value on the positive edge of *C*
- Input changes at other times have no effect on output

Positive edge triggered

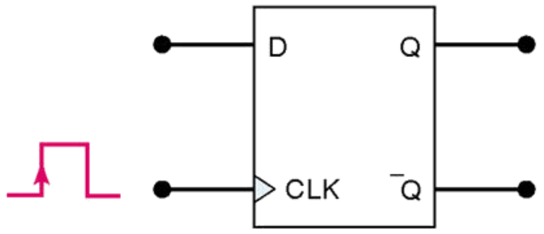


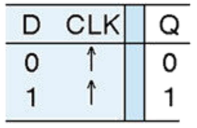
D	C	Q	Q’
0	↑	0	1
1	↑	1	0
X	0	Q_0	Q_0’

D gets latched to Q on the rising edge of the clock.

Clocked D Flip-Flop

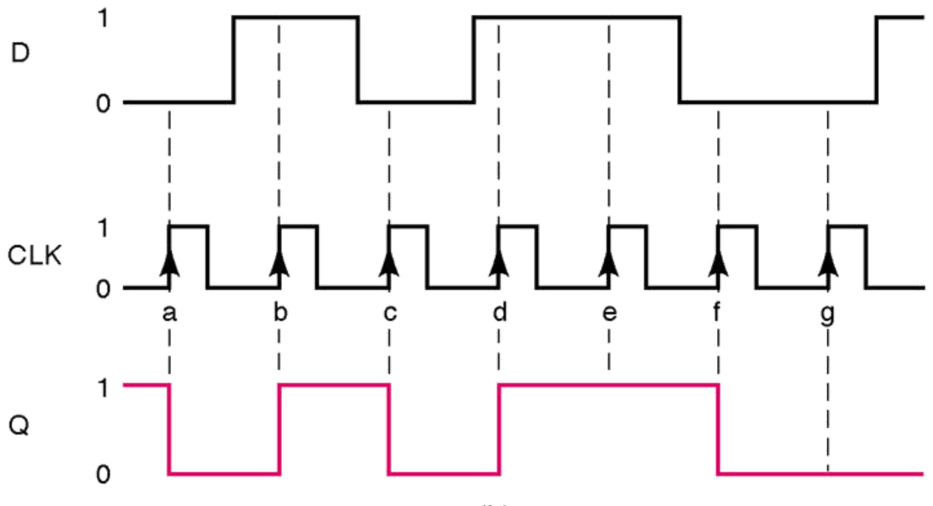
- Stores a value on the positive edge of *C*
- Input changes at other times have no effect on output





D	CLK	Q
0	↑	0
1	↑	1

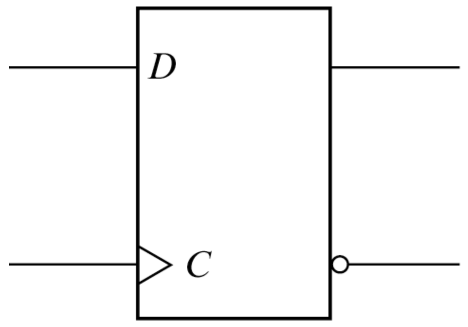
(a)



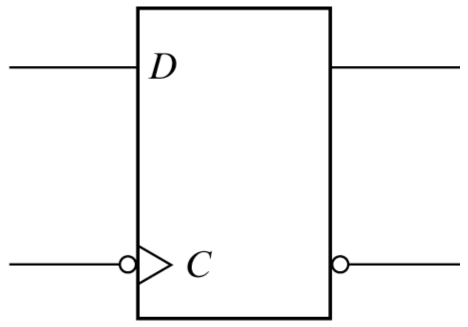
(b)

Positive and Negative Edge D Flip-Flop

- D flops can be triggered on positive or negative edge
- Bubble before *Clock (C)* input indicates negative edge trigger

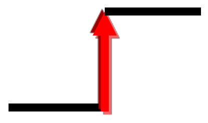


(a) Positive-edge

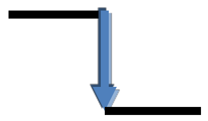


(a) Negative-edge

Fig. 5-11 Graphic Symbol for Edge-Triggered *D* Flip-Flop



Lo-Hi edge



Hi-Lo edge

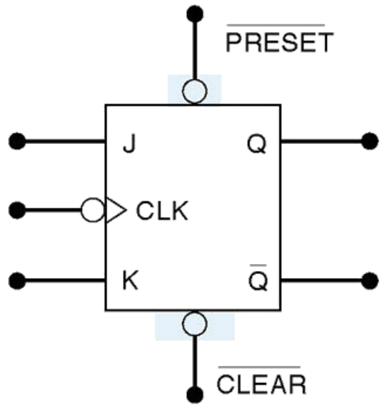
Asynchronous Inputs

- J, K are synchronous inputs

  o Effects on the output are synchronized with the CLK input.

- **Asynchronous inputs** operate independently of the synchronous inputs and clock

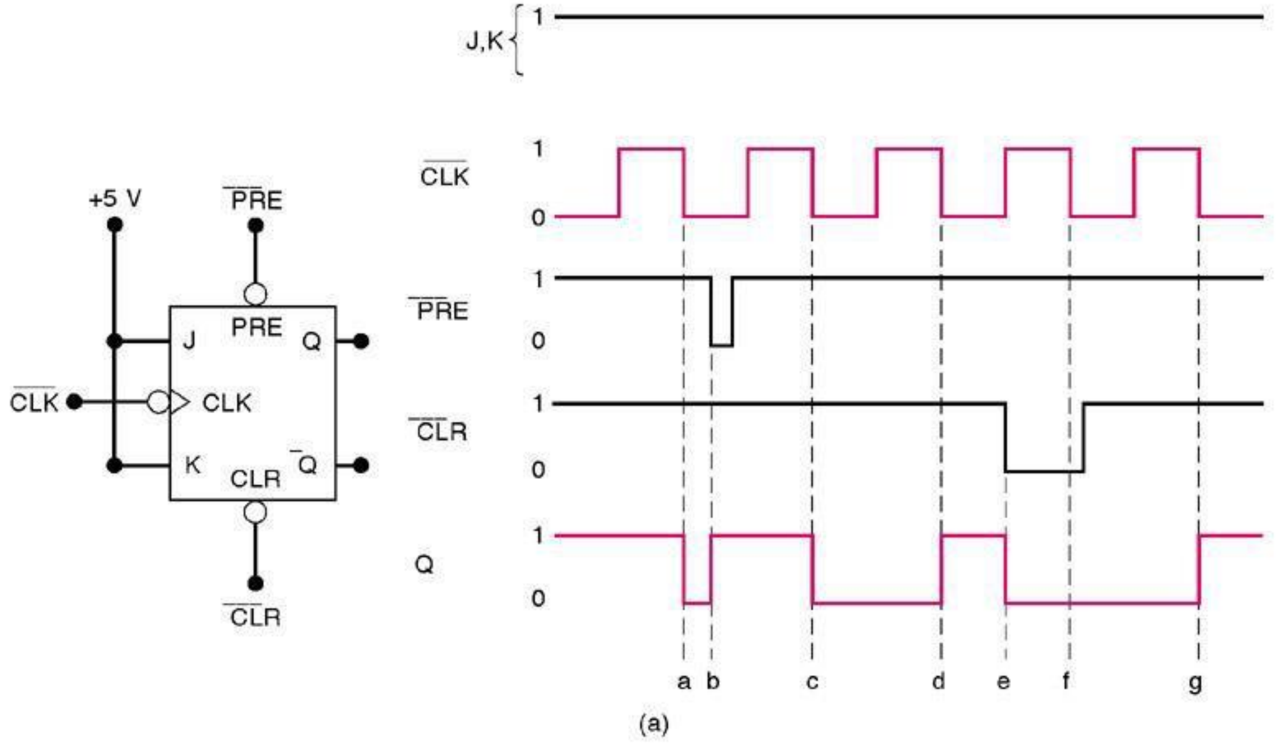
  o Set the FF to 1/0 states at any time.



$\overline{\text{PRESET}}$	$\overline{\text{CLEAR}}$	FF response
1	1	Clocked operation*
0	1	Q = 1 (regardless of CLK)
1	0	Q = 0 (regardless of CLK)
0	0	Not used

*Q will respond to J, K, and CLK

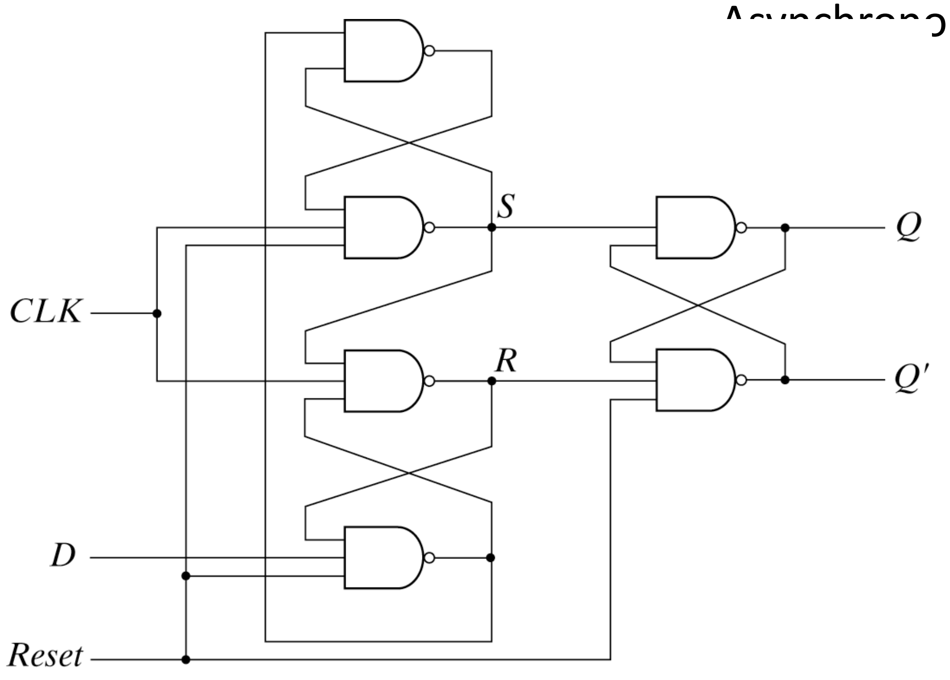
Asynchronous Inputs



Point	Operation
a	Synchronous toggle on NGT of $\overline{CLK}$
b	Asynchronous set on $\overline{PRE}$ = 0
c	Synchronous toggle
d	Synchronous toggle
e	Asynchronous clear on $\overline{CLR}$ = 0
f	$\overline{CLR}$ over-rides the NGT of $\overline{CLK}$
g	Synchronous toggle

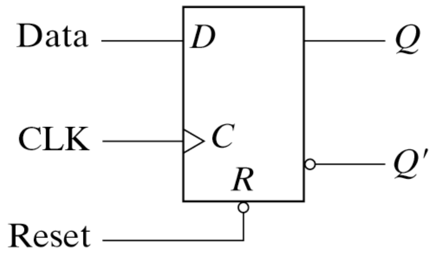
(b)

Asynchronous Inputs



(a) Circuit diagram

- Note reset signal (R) for D flip flop
- If R = 0, the output Q is cleared
- This event can occur at any time, regardless of the value of the CLK



(b) Graphic symbol

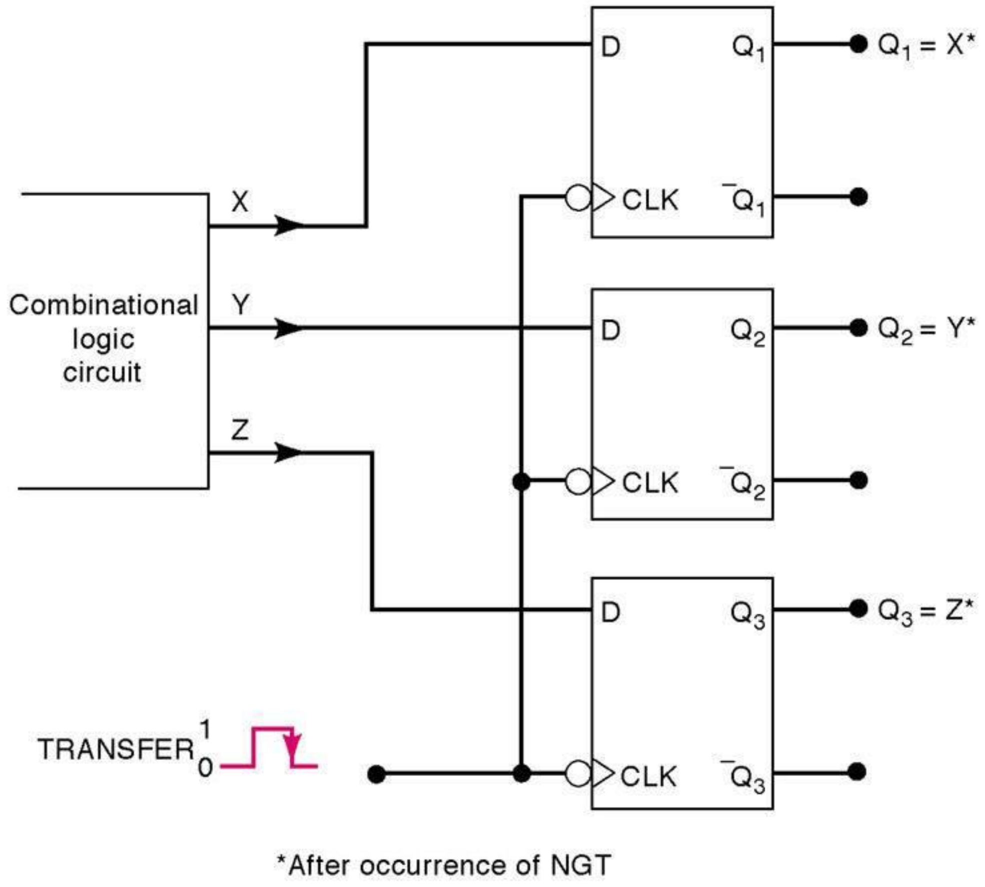
R	*C*	*D*	*Q*	*Q′*
0	X	X	0	1
1	↑	0	0	1
1	↑	1	1	0

(b) Function table

Fig. 5-14 *D* Flip-Flop with Asynchronous Reset

Parallel Data Transfer

- Flip flops store outputs from combinational logic
- Multiple flops can store a collection of data



Summary

- Flip flops are powerful storage elements
  - They can be constructed from gates and latches!
- D flip flop is simplest and most widely used
- Asynchronous inputs allow for clearing and presetting the flip flop output
- Multiple flops allow for data storage
  - The basis of computer memory!
- Combine storage and logic to make a computation circuit
- Next time: Analyzing sequential circuits.

Counters

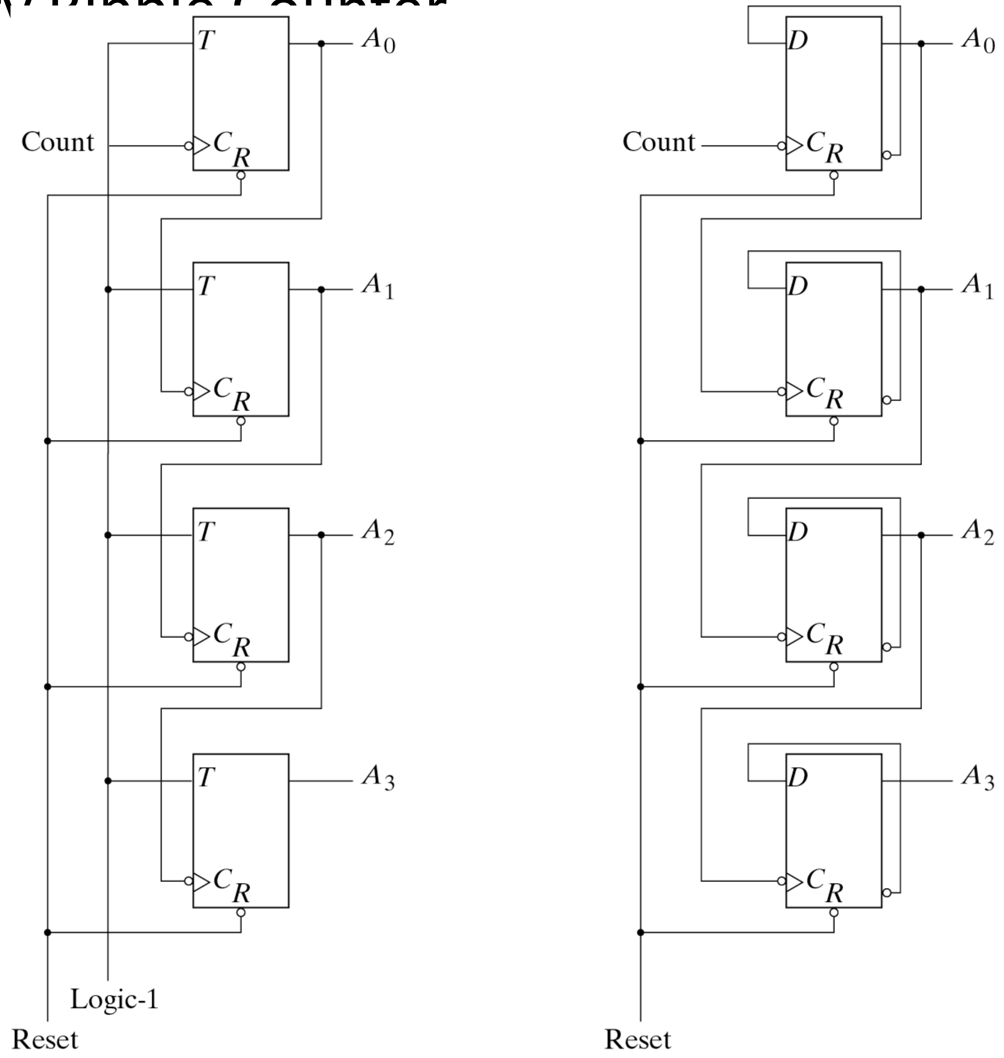
- Counters are important components in computers
  - The increment or decrement by one in response to input
- Two main types of counters
  - Ripple (asynchronous) counters
  - Synchronous counters
- Ripple counters
  - Flip flop output serves as a source for triggering other flip flops
- Synchronous counters
  - All flip flops triggered by a clock signal
- Synchronous counters are more widely used in industry.

Counters

- Counter: A register that goes through a prescribed series of states
- Binary counter
  - Counter that follows a binary sequence
  - N bit binary counter counts in binary from n to 2^{n-1}
- Ripple counters triggered by initial Count signal
- Applications:
  - Watches
  - Clocks
  - Alarms
  - Web browser refresh

Binary Ripple Counter

- Reset signal sets all outputs to 0
- Count signal toggles output of low-order flip flop
- Low-order flip flop provides trigger for adjacent flip flop
- Not all flops change value simultaneously
  - Lower-order flops change first
- Focus on D flip flop implementation



(a) With T flip-flops (b) With D flip-flops

Fig. 6-8 4-Bit Binary Ripple Counter

Asynchronous Counters

- Each FF output drives the CLK input of the next FF.
- FFs do not change states in exact synchronism with the applied clock pulses.
- *There is delay between the responses of successive FFs.*
- *Ripple counter* due to the way the FFs respond one after another in a kind of rippling effect.

A_3	A_2	A_1	A_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
1	0	0	0
1	0	0	1

Synchrono

- Synchronous(parallel) counters
  - All of the FFs are triggered simultaneously by the clock input pulses.
  - All FFs change at same time
- Remember
  - If J=K=0, flop maintains value
  - If J=K=1, flop toggles
- Most counters are synchronous in computer systems.
- Can also be made from D flops
- Value increments on positive edge

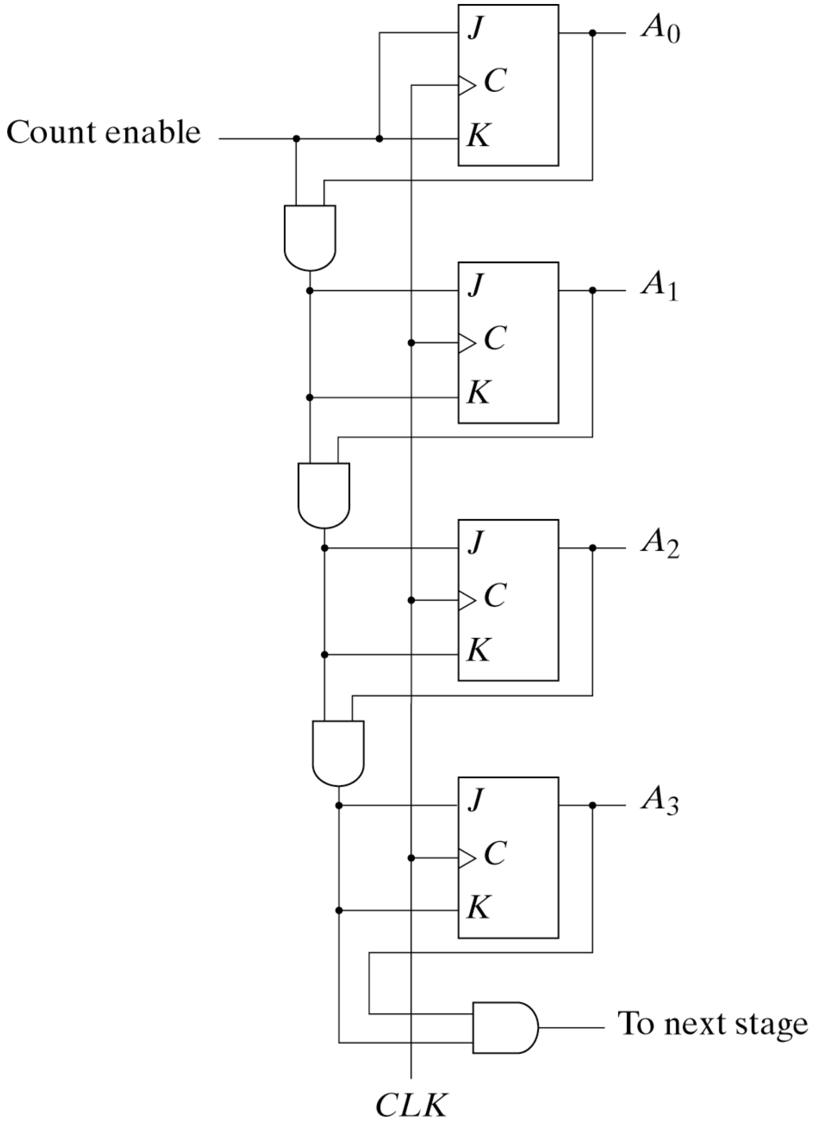
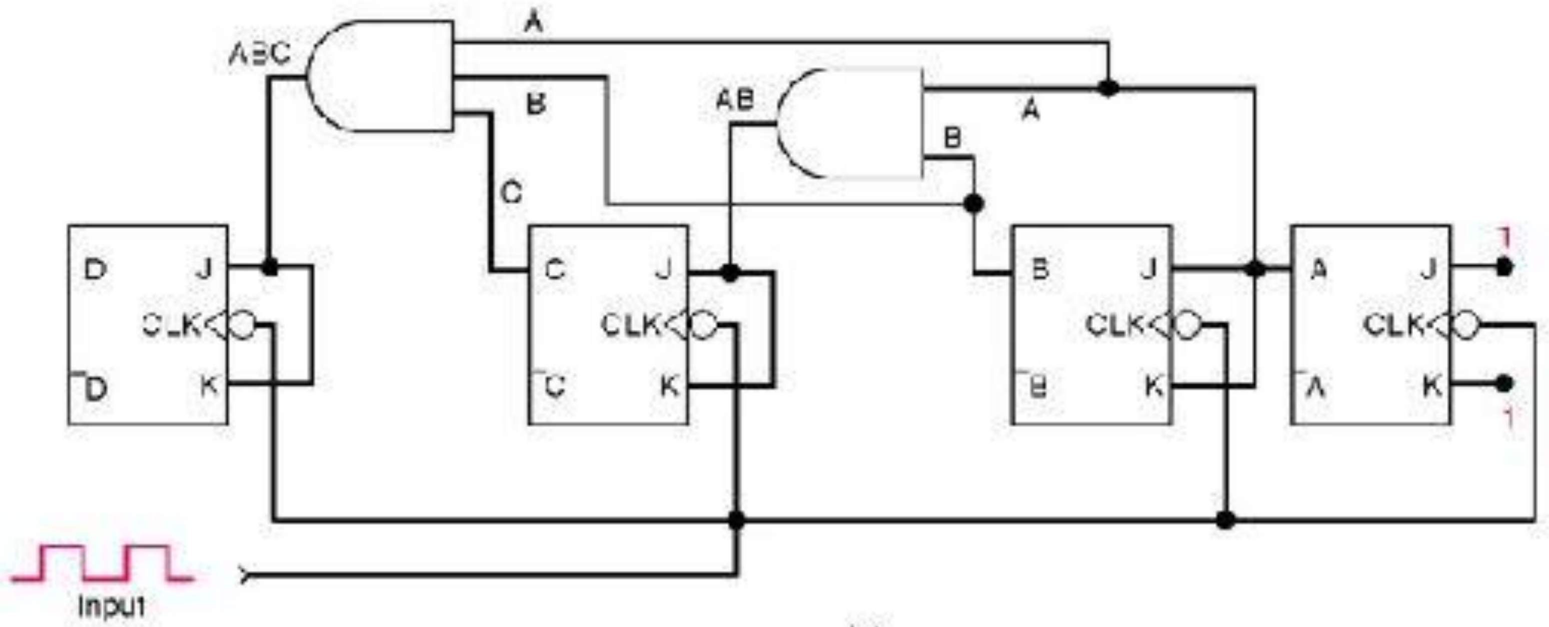


Fig. 6-12 4-Bit Synchronous Binary Counter

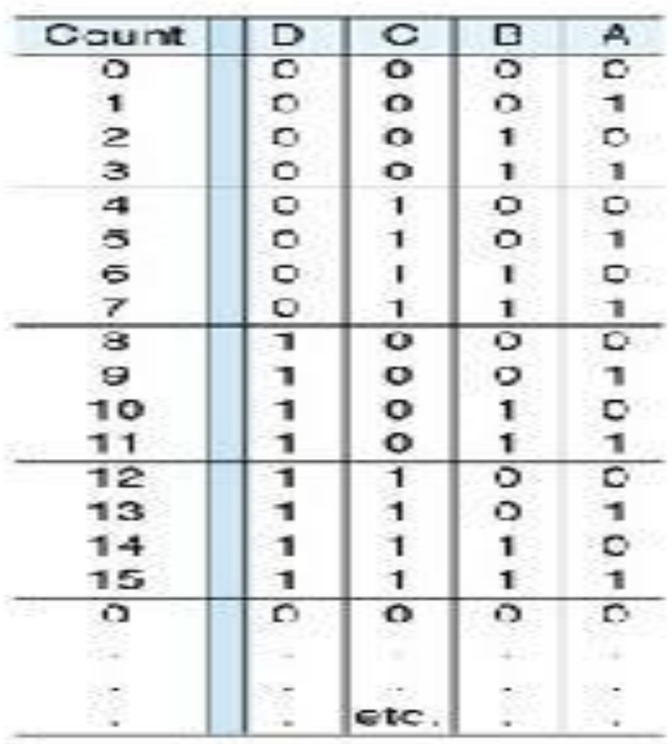
Synchronous counters

- Synchronous counters
  - Same counter as previous slide except Count enable replaced by J=K=1
  - Note that clock signal is a square wave
  - Clock fans out to all clock inputs



(a)

Circuit operation



Count	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
0	0	0	0	0
.	.	.	.	.
.	.	.	.	.
.	.	etc.	.	.

(b)

- Count value increments on each negative edge
- Note that low-order bit (A) toggles on each clock cycle

Registers

- Register
  - Consists of N Flip-Flops
  - Stores N bits
  - Common clock used for all Flip-Flops
- Shift Register
  - A register that provides the ability to shift its contents (either left or right).
  - Must use Flip-Flops
    - Either edge-triggered or master-slave
  - Cannot use Level-sensitive Gated Latches

Overview of Shift Registers

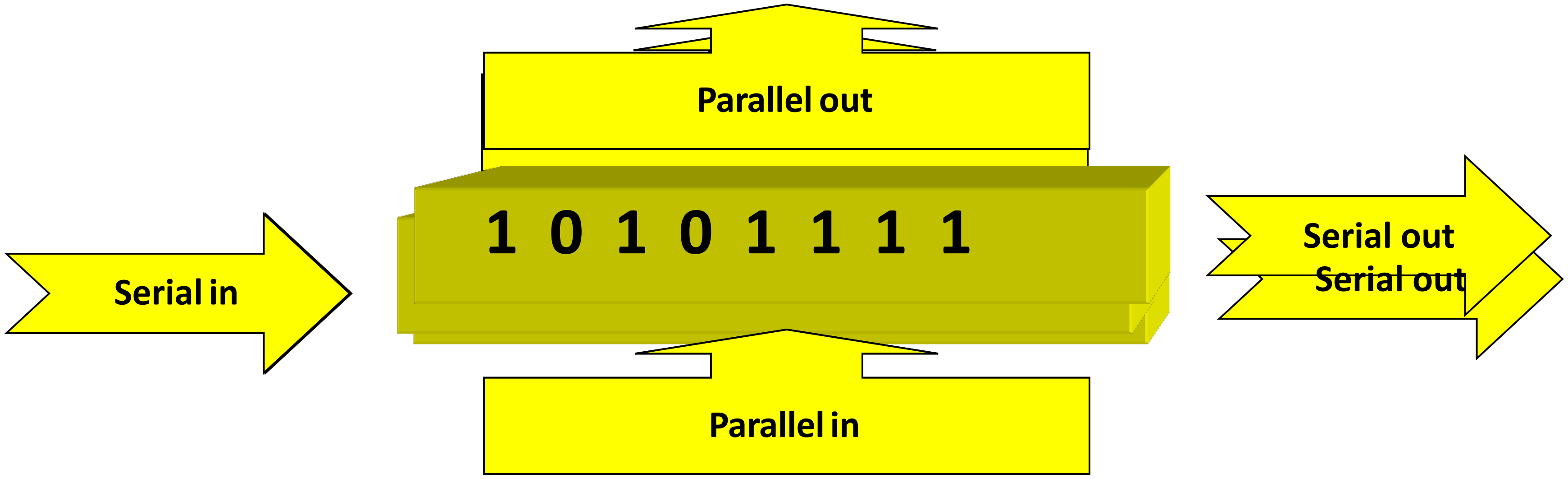
- **A shift register is a sequential logic device made up of flip-flops that allows parallel or serial loading and serial or parallel outputs as well as shifting bit by bit.**
- **Common tasks of shift registers:**
  - **Serial/parallel data conversion**
  - **Time delay**
  - **Ring counter**
  - **Twisted-ring counter or Johnson counter**
  - **Memory device**

Characteristics of Shift Registers

- **Number of bits (4-bit, 8-bit, etc.)**
- **Loading**
  - **Serial**
  - **Parallel (asynchronous or synchronous)**
- **Common modes of operation.**
  - **Parallel load**
  - **Shift right-serial load**
  - **Shift left-serial load**
  - **Hold**
  - **Clear**
- **Recirculating or non-recirculating**

Serial/Parallel Data Conversion

Shift registers can be used to convert from serial-to-parallel or the reverse from parallel-to-serial.



UNIT-III

Computer Arithmetic

- The Basic arithmetic operations are:

  1. Addition 2. subtraction 3. Multiplication 4. Division.

- An arithmetic instruction may specify binary or decimal data , and in each case the data may be in fixed point or floating point.
- The solution to any problem that is stated by a finite number of well defined procedural steps is called an Algorithm.
- Here the arithmetic operation are implemented for the following data types:

1) Fixed point binary data in SMR
2) Fixed point binary data in S2's CR
3) Floating point binary data
4) Binary-coded decimal(BCD)

Addition and Subtraction

- There are three ways to represent negative fixed point binary numbers.

  1)SMR 2)Signed 1'S Comp 3)Signed 2's Comp
- Floating point operations most of the computers use signed magnitude representation for the mantissa.

Addition Algorithm

- when the signs of A and B are identical add the two magnitudes and attach the sign of A to the result.
- When the signs of A and B are different compare the magnitudes and subtract the smaller number from the larger. Choose the sign of the result to be the same as A if A>B or the complement of the sign of A if A<B .if the two magnitudes are equal subtract b from a and make the sign of the result positive.

Addition and Subtraction

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)

Table : Addition and Subtraction of Signed Magnitude Numbers

Addition and Subtraction

Subtract Algorithm

- When the signs of A and B are different add the two magnitudes and attach the sign of A to the result.

- When the signs of A and B are identical compare the magnitudes and subtract the smaller number from the larger. Choose the sign of the result to be the same as A if A>B or the complement of the sign of A if A<B.if the two magnitudes are equal subtract b from a and make the sign of the result positive.

Addition and Subtraction

Hardware Implementation

- We need Two registers to store two numbers and two flip flops to store signs of numbers.
- It requires a parallel adder to perform the micro operation A+B.
- A comparator circuit is needed to compare A>B,A<B,A=B.
- The output carry flag.
- The add over flow flip flop AVF holds the overflow bit when A and B are added.

Addition and Subtraction

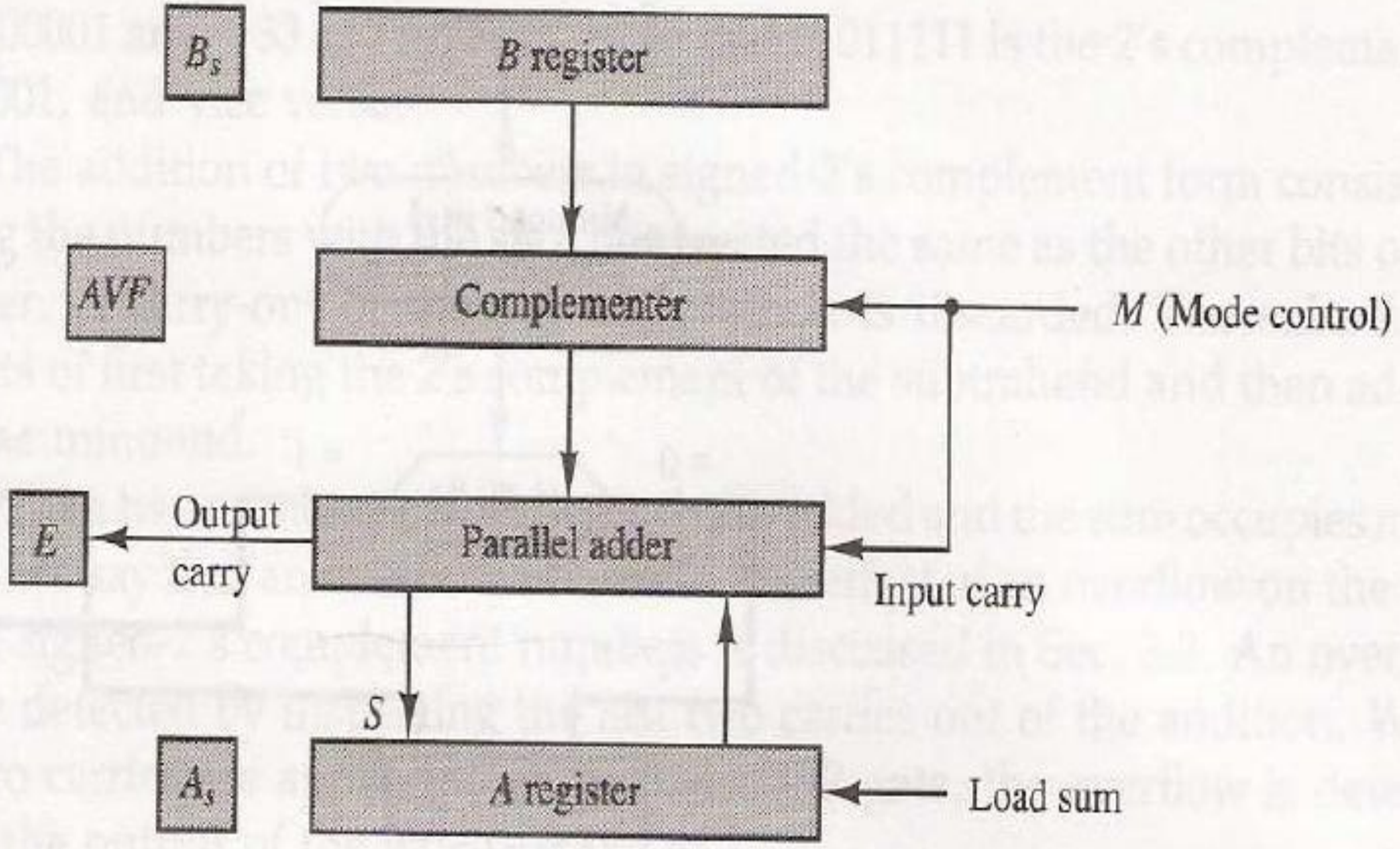


Figure 10-1 Hardware for signed-magnitude addition and subtraction.

Addition and Subtract Algorithm

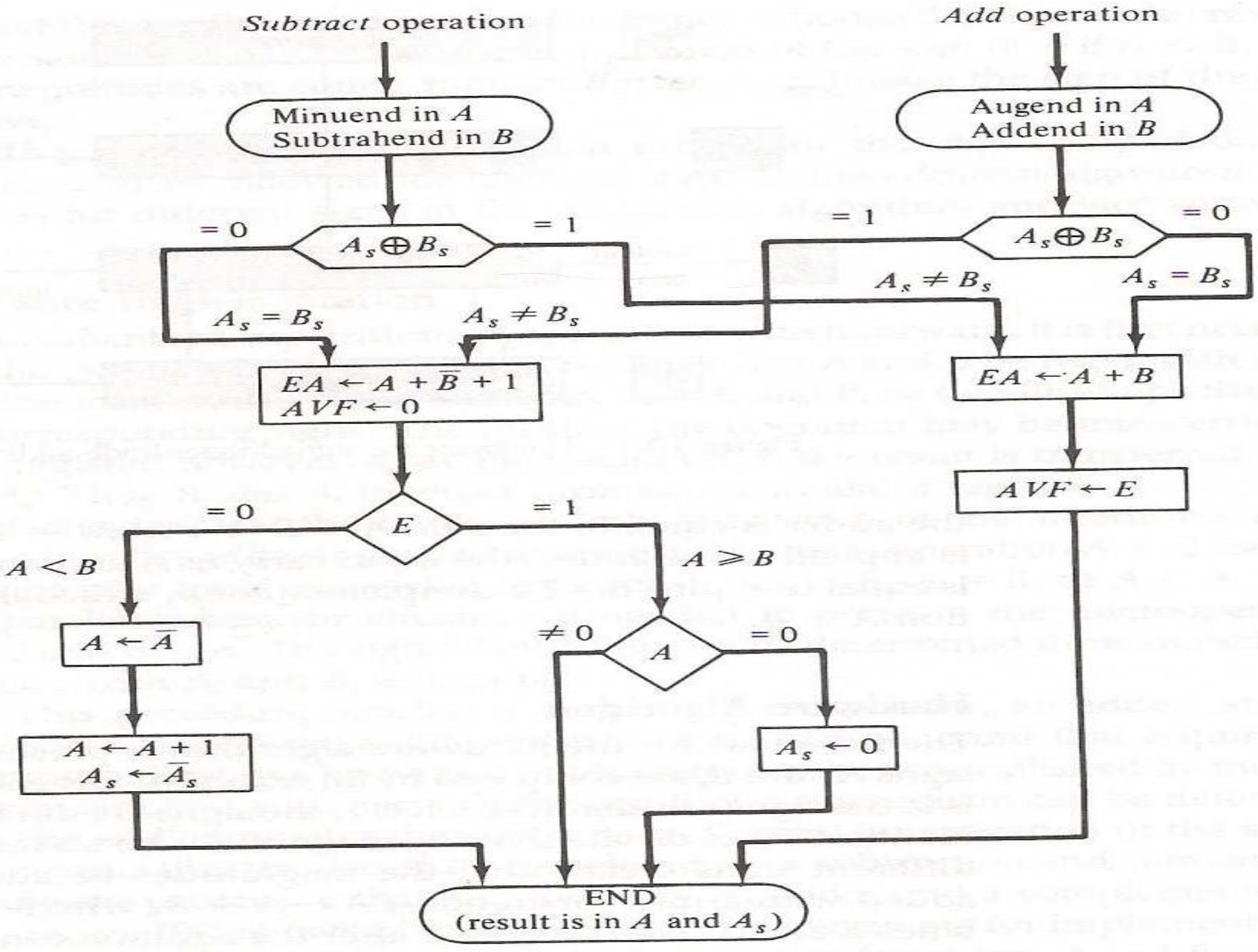
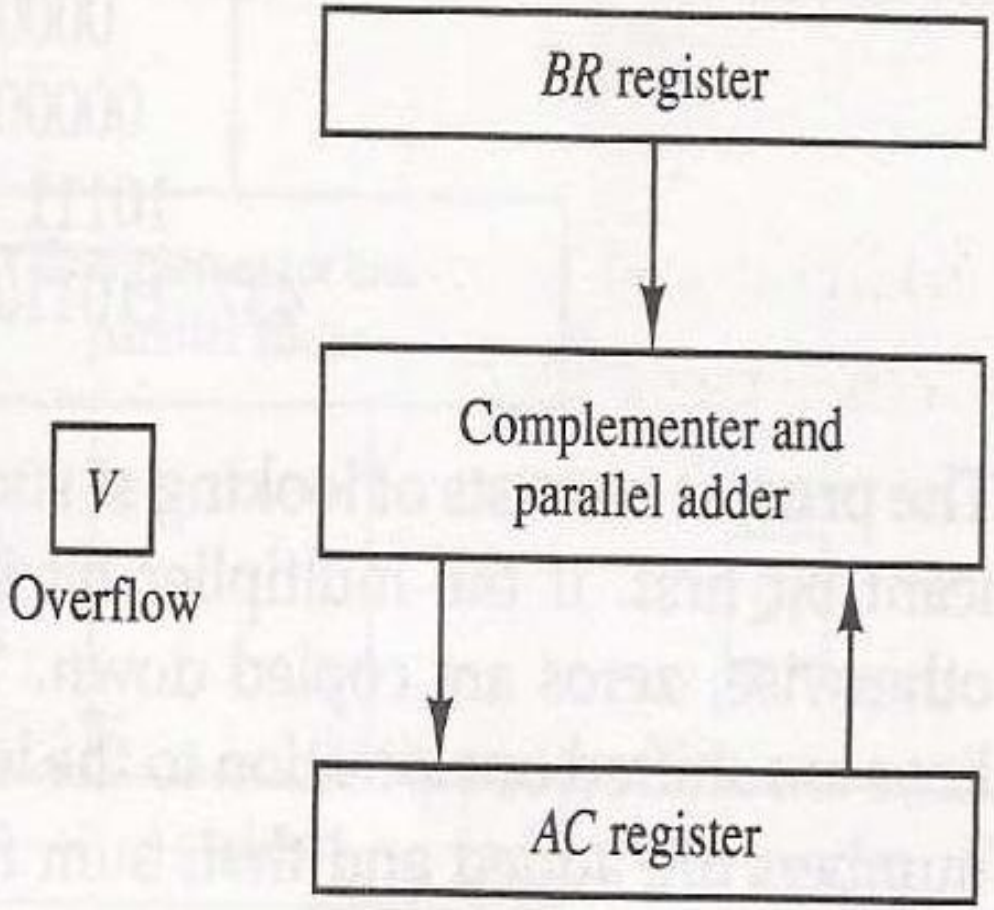


Figure 10-2 Flowchart for add and subtract operations.

Addition and Subtraction with Signed 2's Complement Data

Figure 10-3 Hardware for signed-2's complement addition and subtraction.



Addition and Subtraction with Signed 2's Complement Data

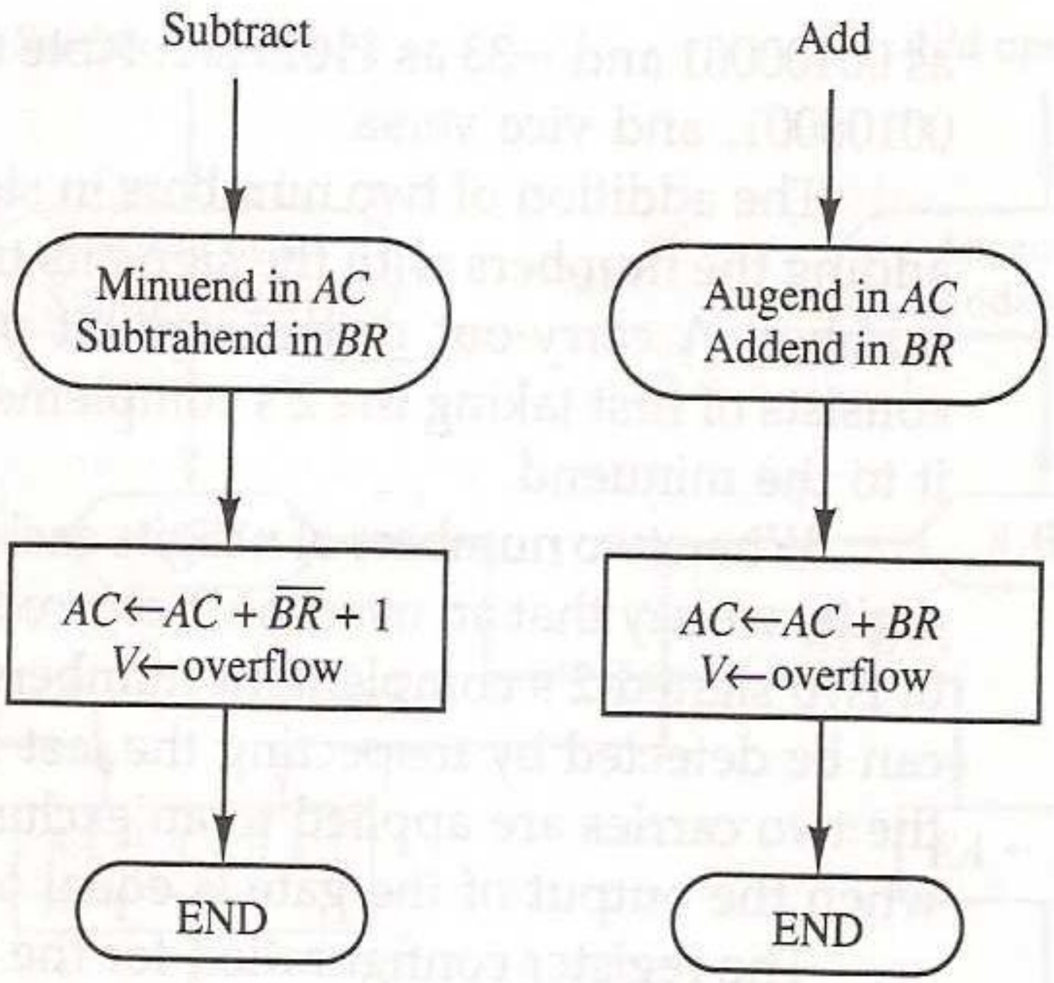
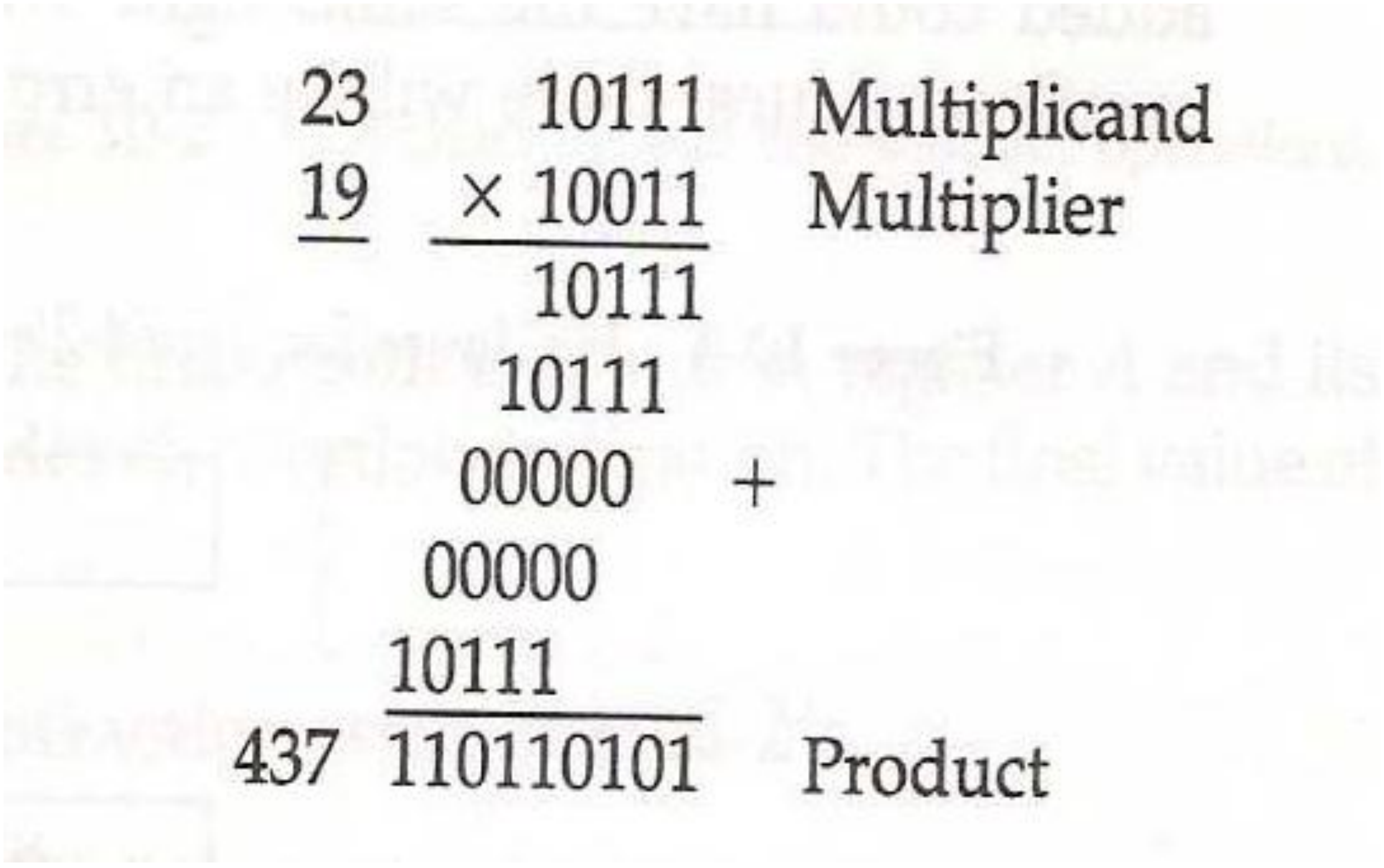


Figure 10-4 Algorithm for adding and subtracting numbers in signed-2's complement representation.

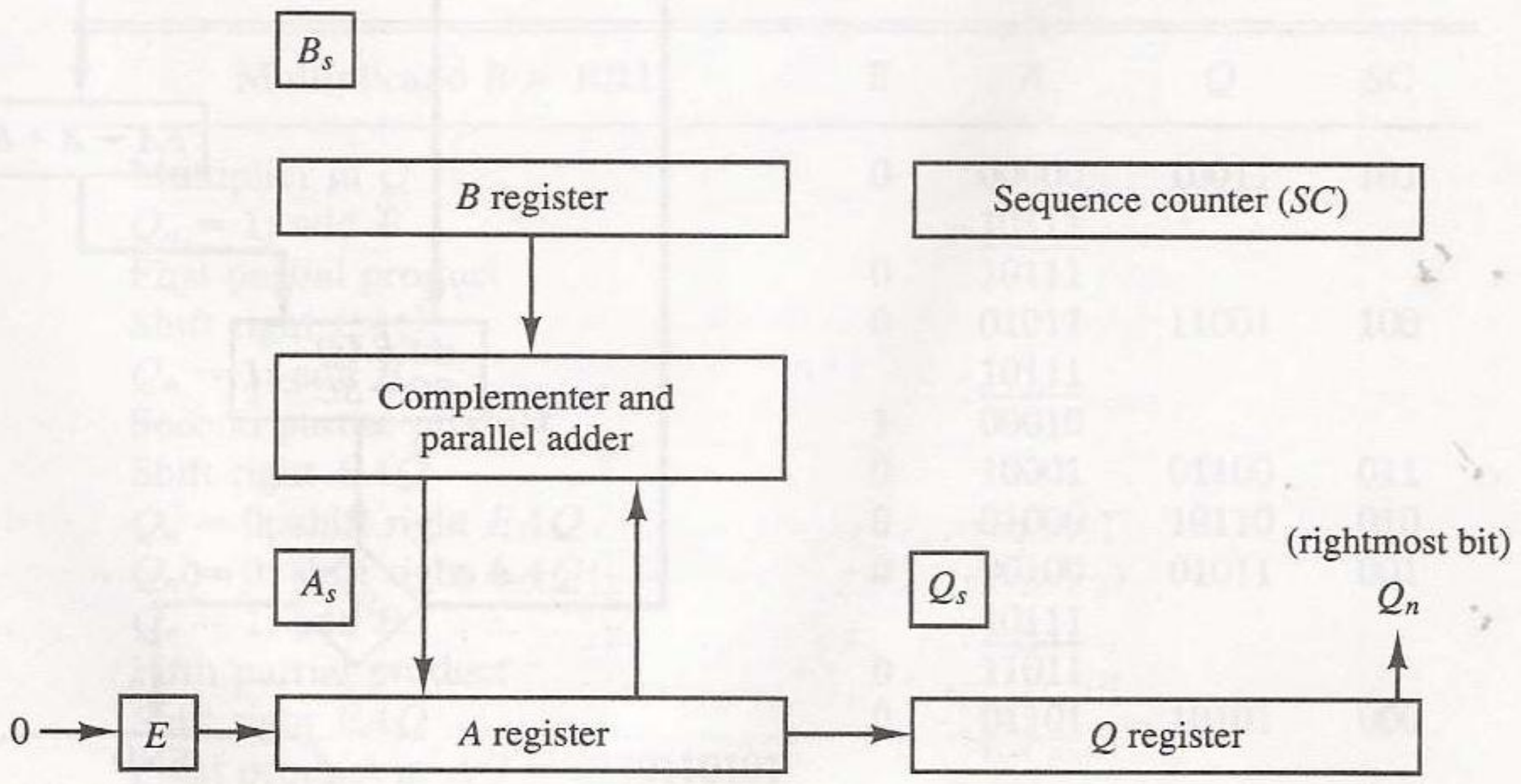
Multiplication Algorithm



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

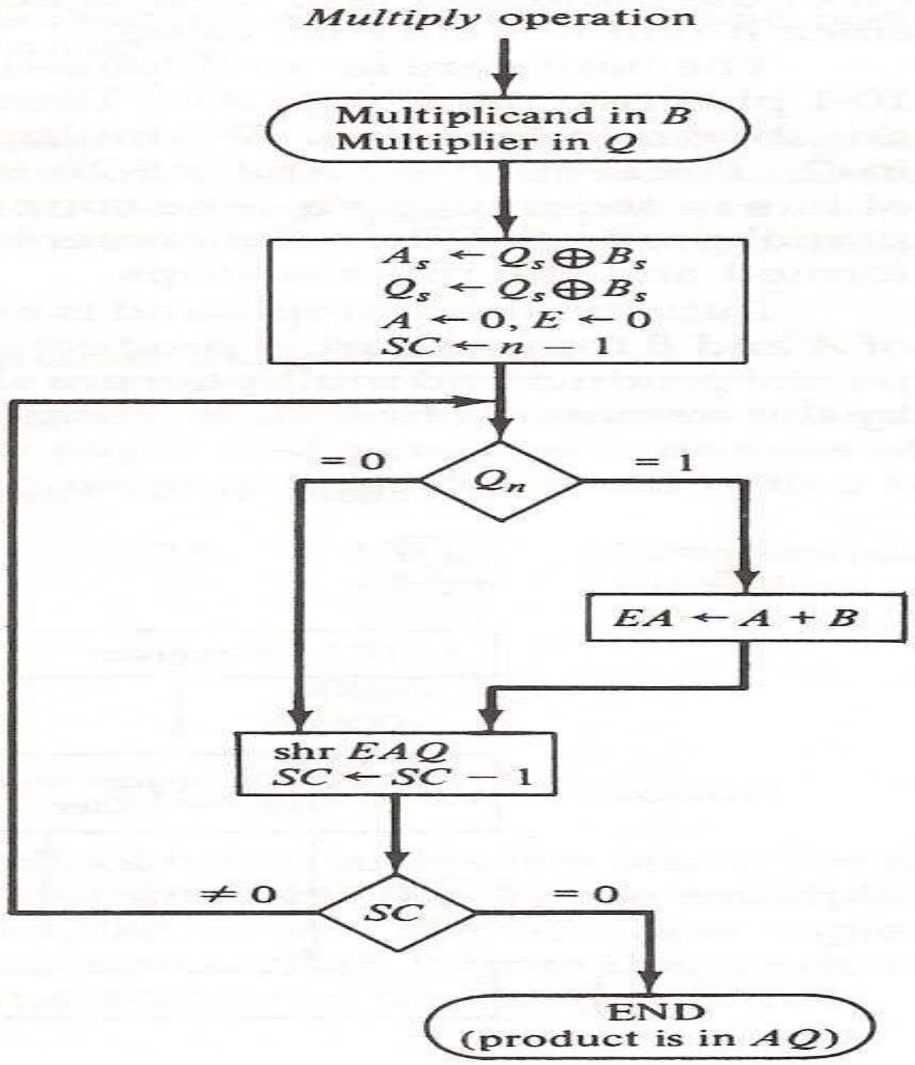
Multiplication Algorithm

Figure 10-5 Hardware for multiply operation.



Multiplication Algorithm

Figure 10-6 Flowchart for multiply operation.



Multiplication Algorithm

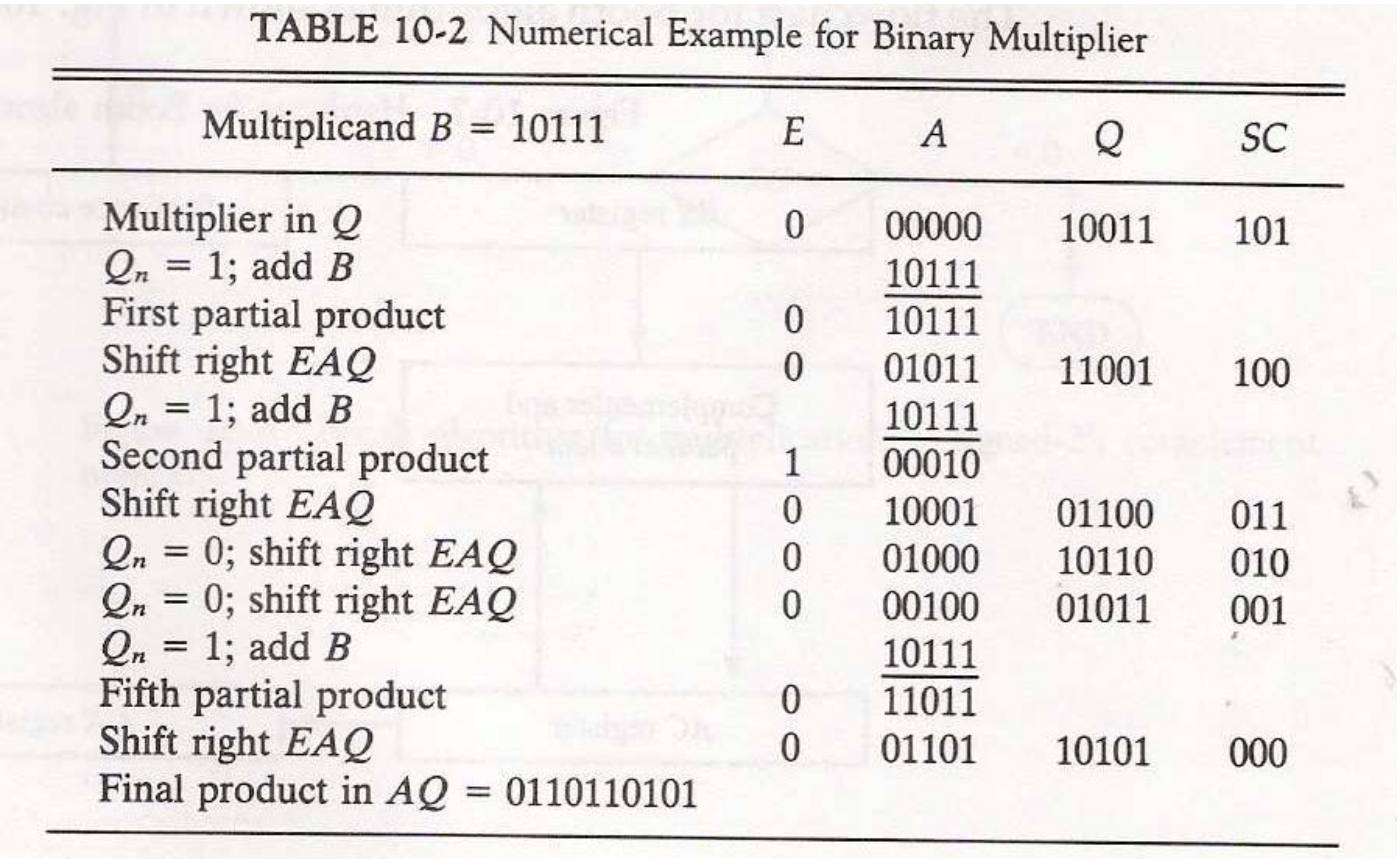


TABLE 10-2 Numerical Example for Binary Multiplier

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 $AQ = 0110110101$				

Booth Multiplication Algorithm

- Booth algorithm gives a procedure for multiplying binary integers in signed 2's complements representation.
- Here it it treats a string of 1's in the multiplier from bit weight 2^k to bit weight 2^m can be treated as $2^{k+1}-2^{m.}$
- The binary number 001110 has a string of 1's from 2^3 to 2^1(k=3,m=1).the number can be represented as $2^{k+1}-2^m=2^4-2^1=16-2=14$.
- So the product is obtained by M *14. where m is Multiplicand 14 is Multiplier can be done as $M*2^4-M*2^1$.
- I.E by shifting the binary multiplicand four times to the left and subtract M shifted left once.

Booth Multiplication Algorithm

- Booth algorithm requires examination of the multiplier bits and shifting of partial product.
- Prior to the shifting the multiplicand may be added to the partial product,subtracted from the partial product or left un changed based on the following rules.

1)The multiplicand is subtracted from the partial product upon encountering the first least significant bit is 1 in a string of 1's in the multiplier.

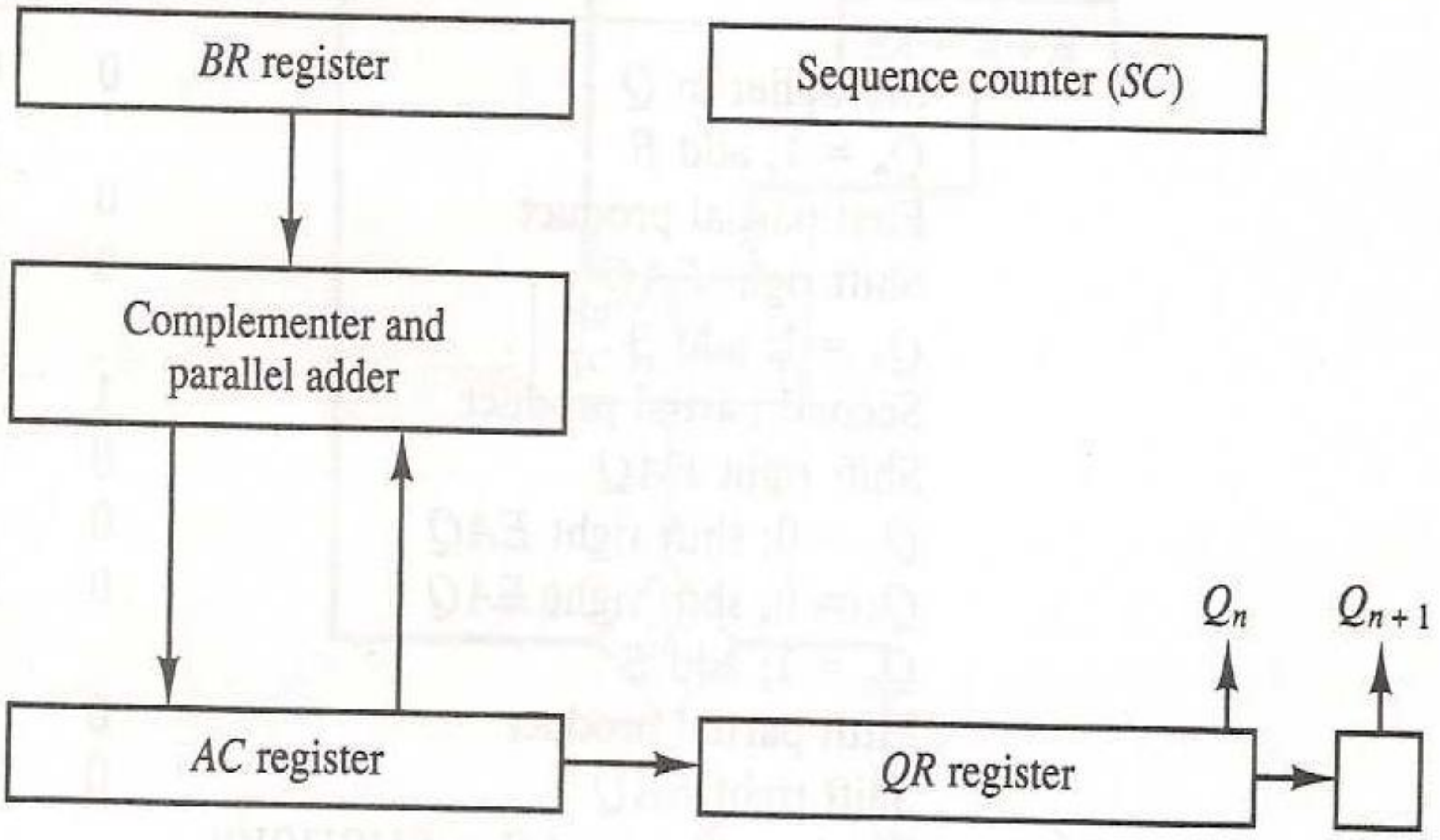
2)The multiplicand is added to the partial product upon encountering the first 0 in a string of 0's in the multiplier.

3) The partial product does not change when the multiplier bit is identical to the previous multiplier bit.

Negative:-14=110010 as $-2^4+2^2-2^{1.}$

Booth Multiplication Algorithm

Figure 10-7 Hardware for Booth algorithm.



Booth Multiplication Algorithm

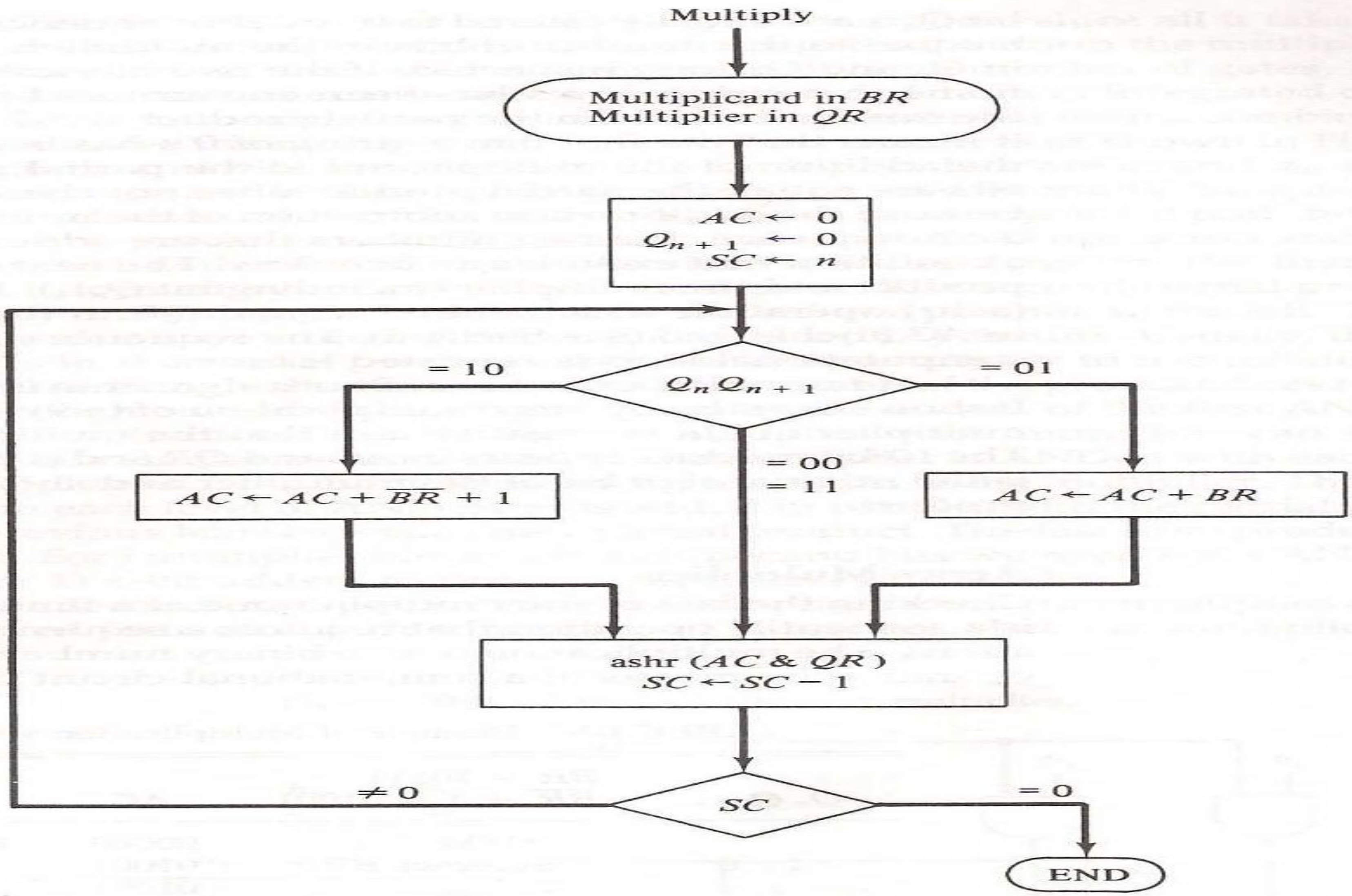


Figure 10-8 Booth algorithm for multiplication of signed-2's complement numbers.

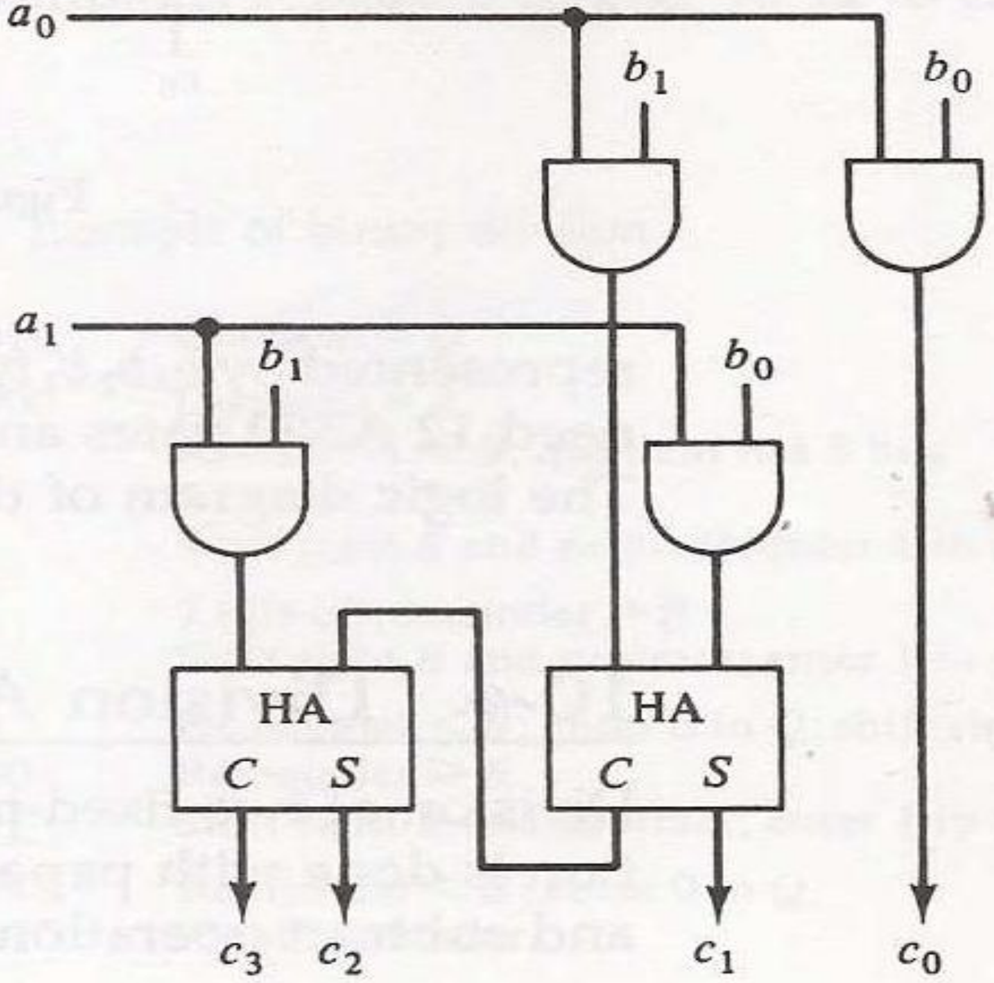
Example of Multiplication with Booth Algorithm

$Q_n Q_{n+1}$	$BR = 10111$ $\overline{BR} + 1 = 01001$	AC	QR	Q_{n+1}	SC
	Initial	00000	10011	0	101
1 0	Subtract *BR*	$\underline{01001}$			
		01001			
	ashr	00100	11001	1	100
1 1	ashr	00010	01100	1	011
0 1	Add *BR*	$\underline{10111}$			
		11001			
	ashr	11100	10110	0	010
0 0	ashr	11110	01011	0	001
1 0	Subtract *BR*	$\underline{01001}$			
		00111			
	ashr	00011	10101	1	000

Array Multiplier

Figure 10-9 2-bit by 2-bit array multiplier.

$$\begin{array}{cccc}
 & & b_1 & b_0 \\
 & & a_1 & a_0 \\
\hline
 & & a_0 b_1 & a_0 b_0 \\
 & a_1 b_1 & a_1 b_0 & \\
\hline
c_3 & c_2 & c_1 & c_0
\end{array}$$



Array Multiplier

- Example:

Multiplicand:3=11

Multiplier : 2=10

```
          1   1
          1   0
        ---------
          0   0
      1   1
    -------------
      1   1   0
```

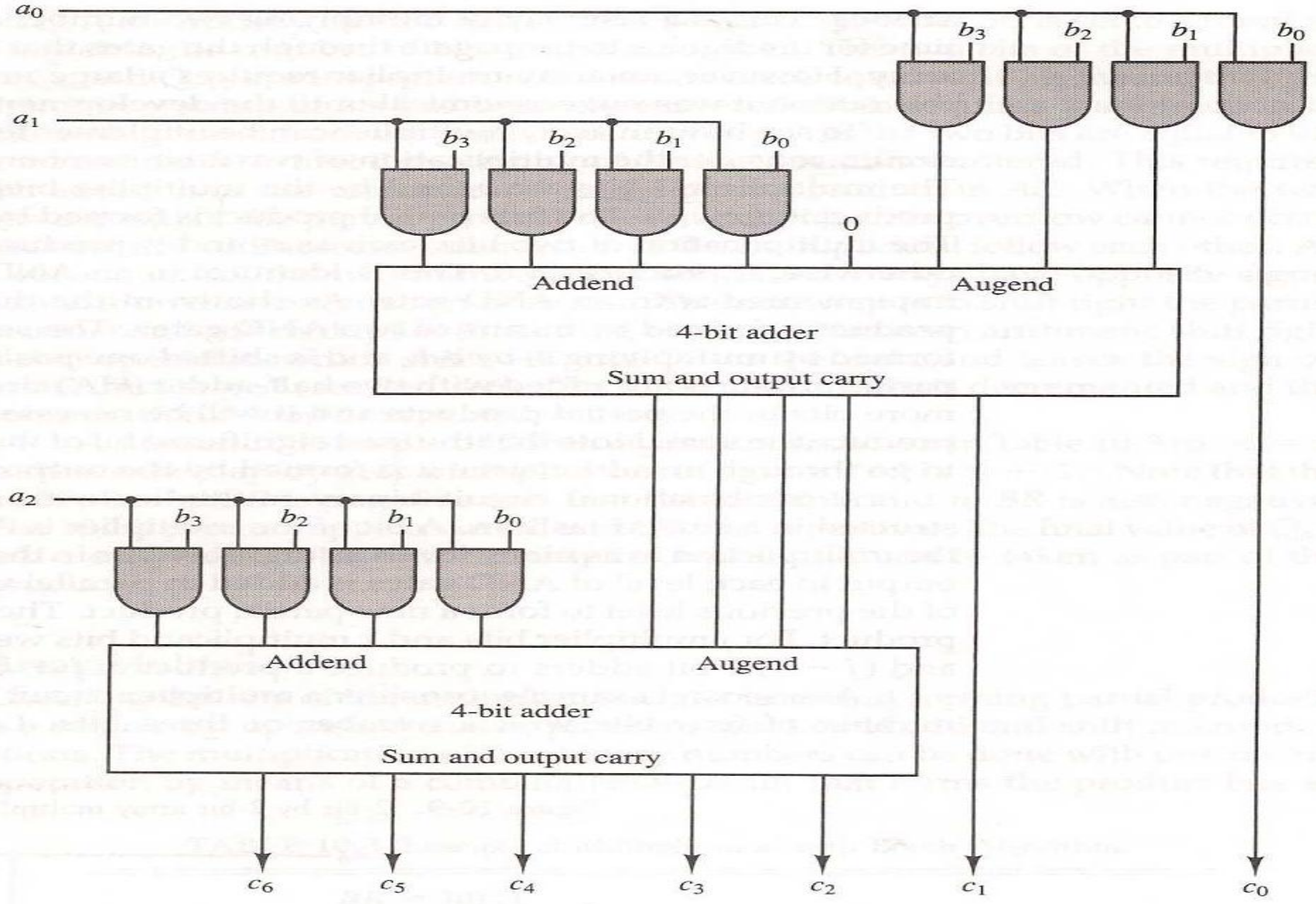


Figure 10-10 4-bit by 3-bit array multiplier.

Array Multiplier

- Example : Multiplicand : 1110 ,Multiplier:111

a0=1,a1=1,a2=1   b0=0,b1=1,b2=1,b3=1

C0=0

```
          0   1   1   1
          1   1   1   0
     ____________________
       1  0   1   0   1          c1=1
       1  1   1   0
____________________
 1     1  0   0   0
```

C0=0,c1=1,c2=0,c3=0,c4=0,c5=1,c6=1

Division Algorithm

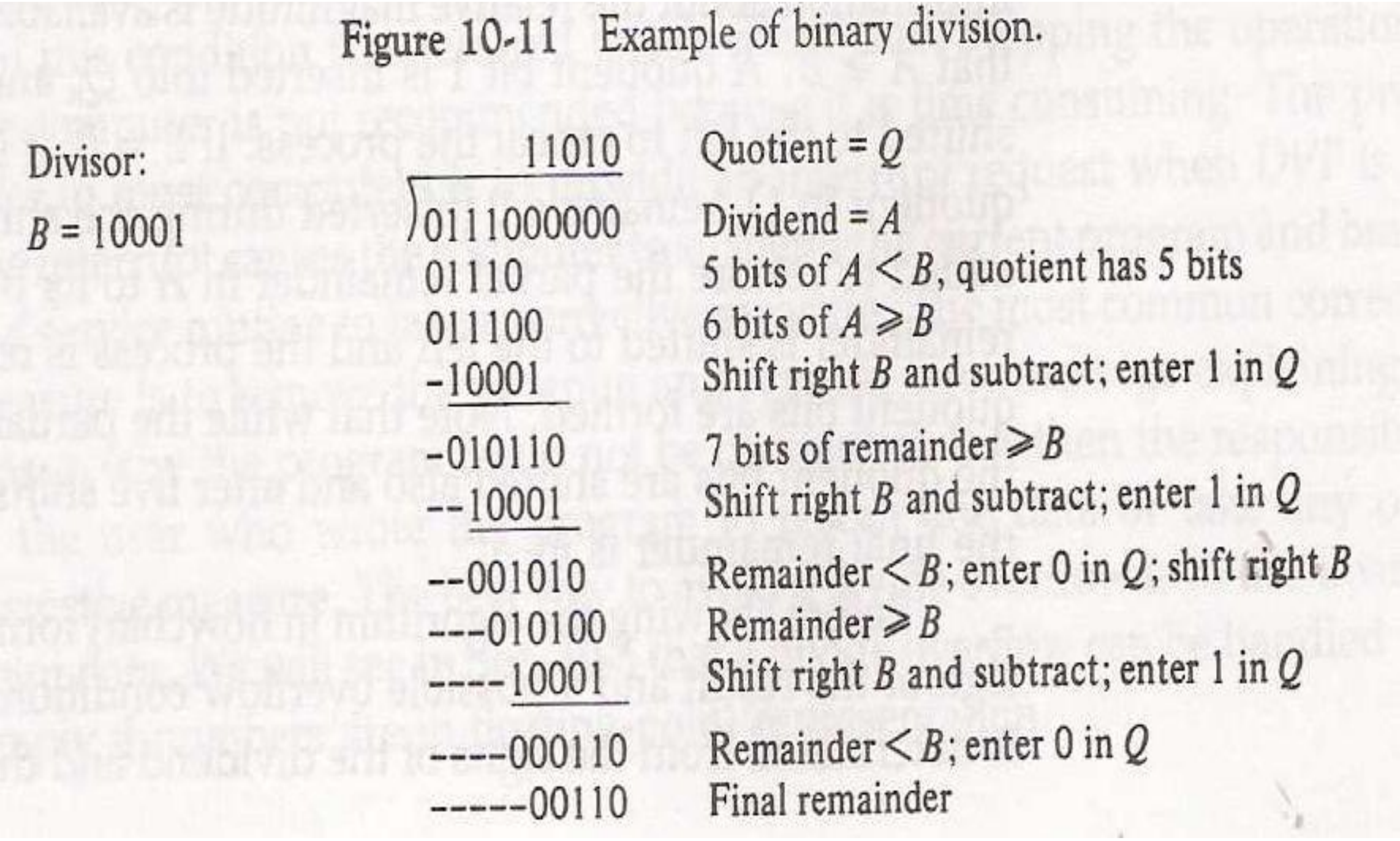


Figure 10-11 Example of binary division.

```
Divisor:                11010         Quotient = Q
                   ___________
B = 10001          )0111000000        Dividend = A
                    01110             5 bits of A < B, quotient has 5 bits
                    011100            6 bits of A ≥ B
                    -10001            Shift right B and subtract; enter 1 in Q
                    ______
                    -010110           7 bits of remainder ≥ B
                    --10001           Shift right B and subtract; enter 1 in Q
                      _____
                    --001010          Remainder < B; enter 0 in Q; shift right B
                    ---010100         Remainder ≥ B
                    ----10001         Shift right B and subtract; enter 1 in Q
                        ______
                    ----000110        Remainder < B; enter 0 in Q
                    ------00110       Final remainder
```

Divisor $B = 10001$, $\overline{B} + 1 = 01111$

	E	A	Q	SC
Dividend:		01110	00000	5
shl EAQ	0	11100	00000	
add $\overline{B} + 1$		$\underline{01111}$		
$E = 1$	1	01011		
Set $Q_n = 1$	1	01011	00001	4
shl EAQ	0	10110	00010	
Add $\overline{B} + 1$		$\underline{01111}$		
$E = 1$	1	00101		
Set $Q_n = 1$	1	00101	00011	3
shl EAQ	0	01010	00110	
Add $\overline{B} + 1$		$\underline{01111}$		
$E = 0$; leave $Q_n = 0$	0	11001	00110	
Add B		$\underline{10001}$		
Restore remainder	1	01010		2
shl EAQ	0	10100	01100	
Add $\overline{B} + 1$		$\underline{01111}$		
$E = 1$	1	00011		
Set $Q_n = 1$	1	00011	01101	1
shl EAQ	0	00110	11010	
Add $\overline{B} + 1$		$\underline{01111}$		
$E = 0$; leave $Q_n = 0$	0	10101	11010	
Add B		$\underline{10001}$		
Restore remainder	1	00110	11010	0
Neglect E				
Remainder in A:		00110		
Quotient in Q:			11010	

Figure 10-12 Example of binary division with digital hardware.

Divide Overflow

Conditions for overflow:

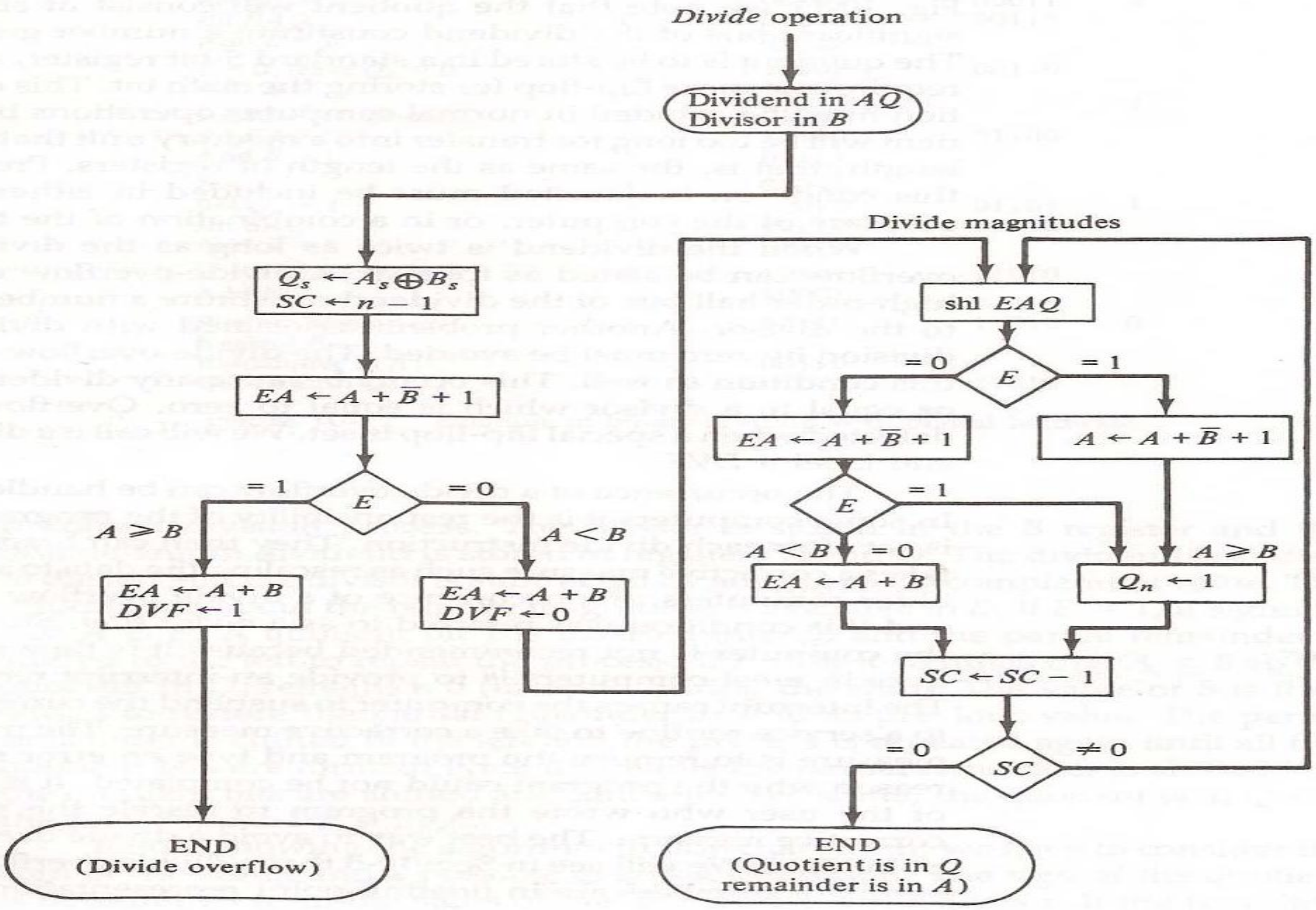
1)A divide overflow condition occurs if the higher order off bits of the dividend constitute a number grater than or equal to the divisor.

2) Division by zero must be Avoided.

- Over flow condition is detected by an special flip flop called divide over flow flip flop(DVF).

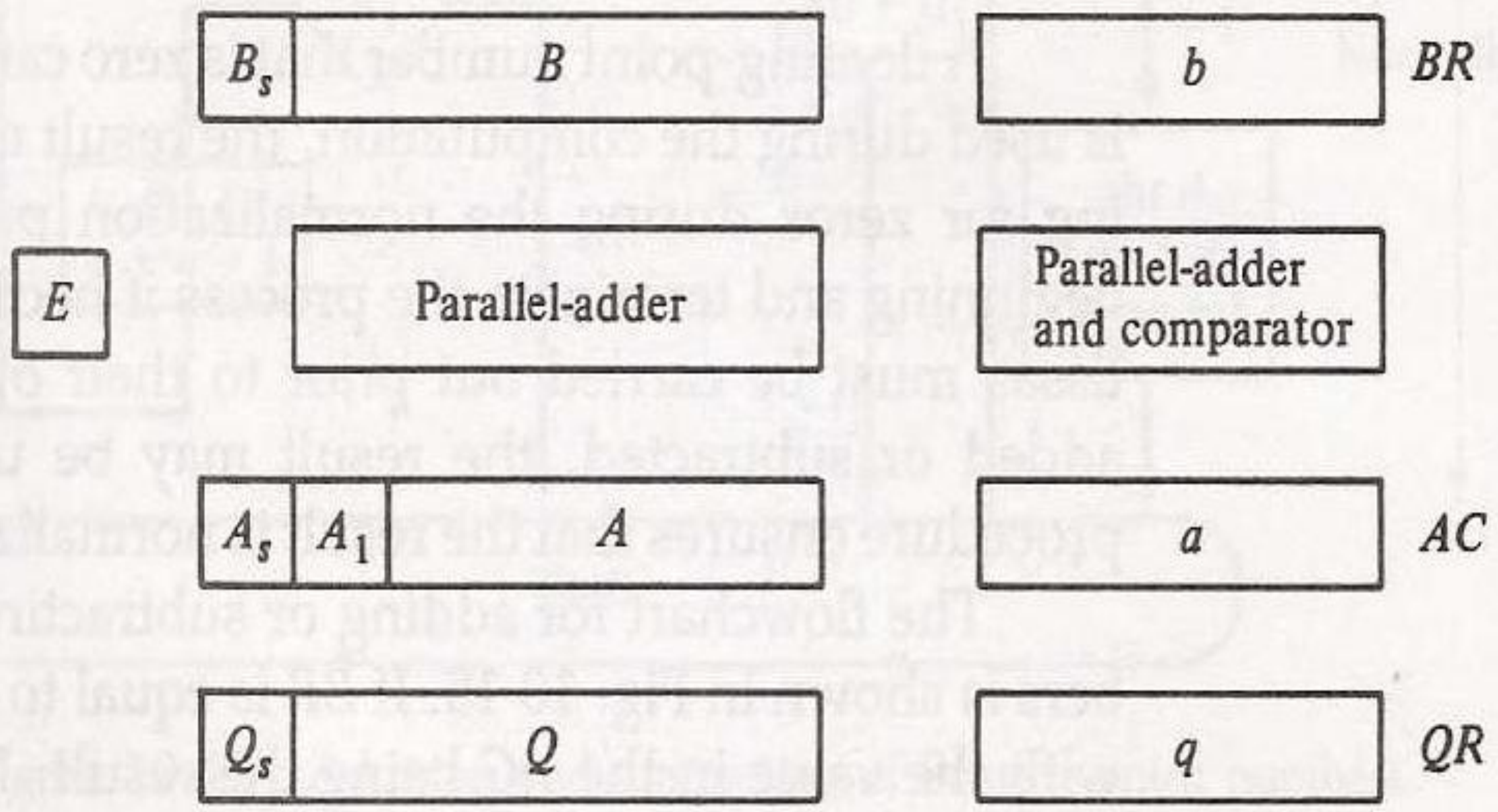
Division Algorithm

Figure 10-13 Flowchart for divide operation.



Floating-Point Algorithms

Figure 10-14 Registers for floating-point arithmetic operations.



Addition and Subtraction Algorithm

1) Check For Zero.
2) Align the Mantissas.
3) Add or Subtract the Mantissas.
4) Normalize the Result.

Addition and Subtraction Algorithm

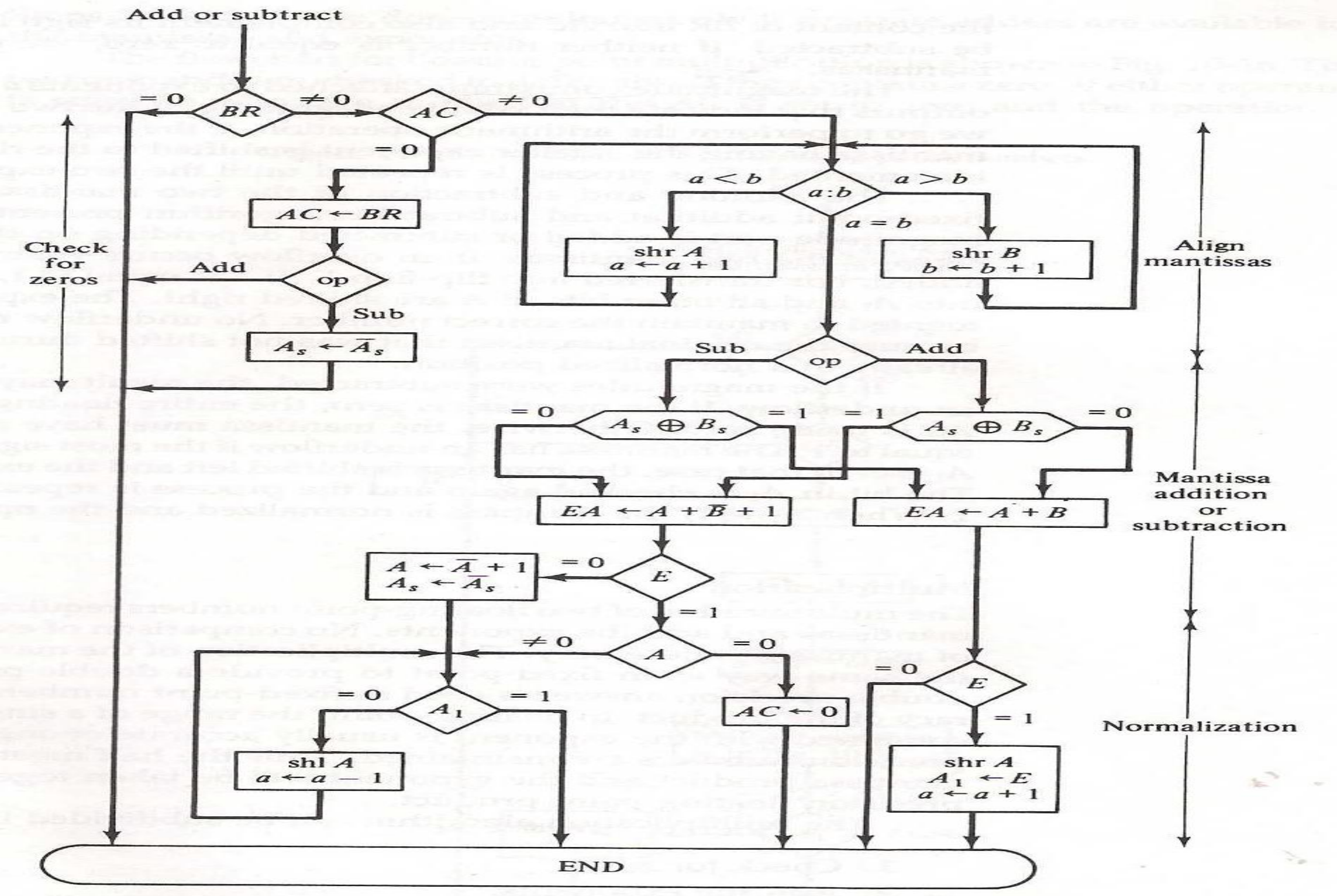


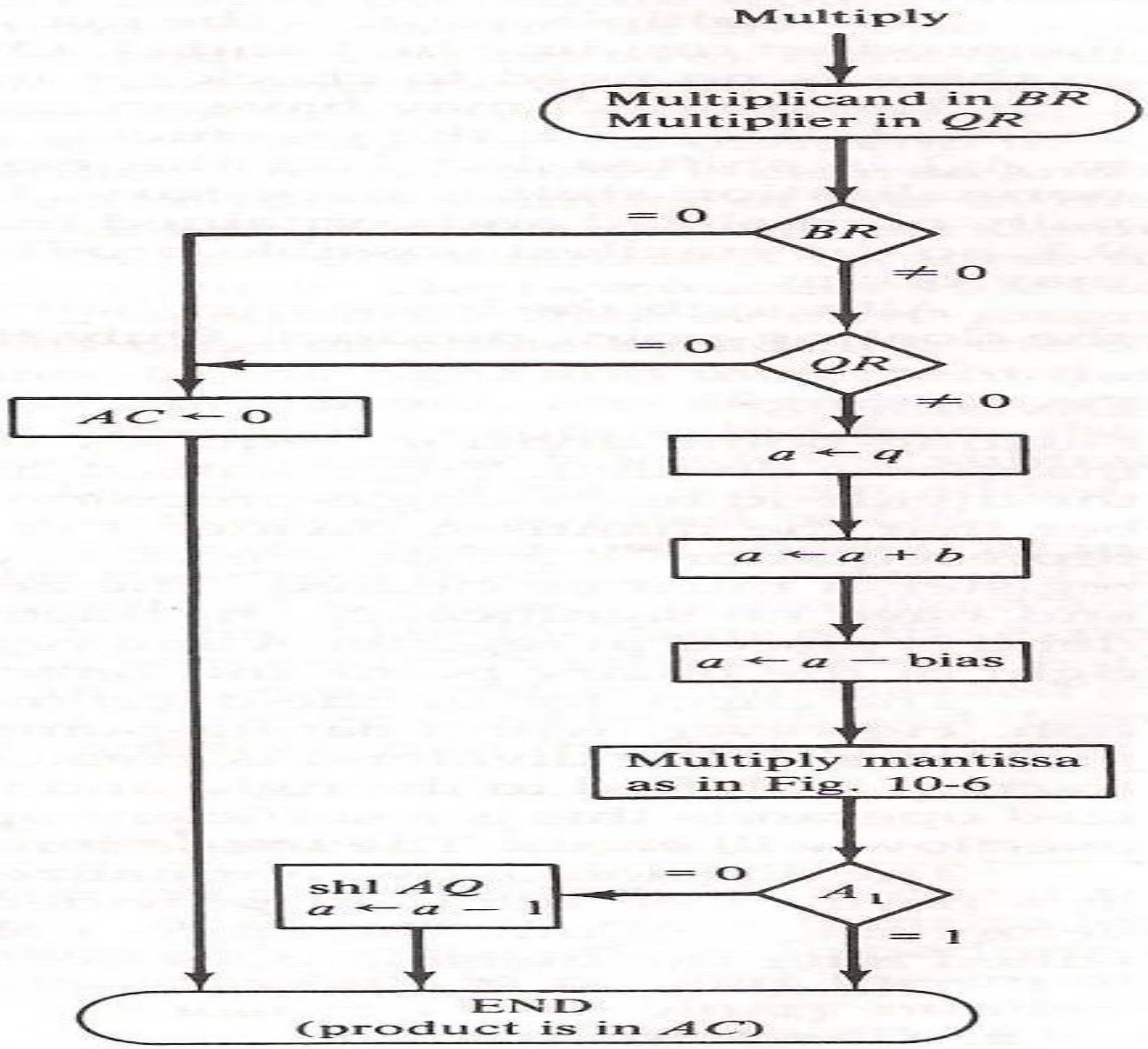
Figure 10-15 Addition and subtraction of floating-point numbers.

Multiplication

1) Check for Zeros
2) Add the Exponents
3) Multiply the Mantissas.
4) Normalize the Product.

Multiplication Algorithm

Figure 10-16 Multiplication of floating-point numbers.



Division

1) Check for Zeros.
2) Initialize the register and evaluate the sign.
3) Align the dividend.
4) Subtract the Exponents.
5) Divide the Mantissas.

Division Algorithm

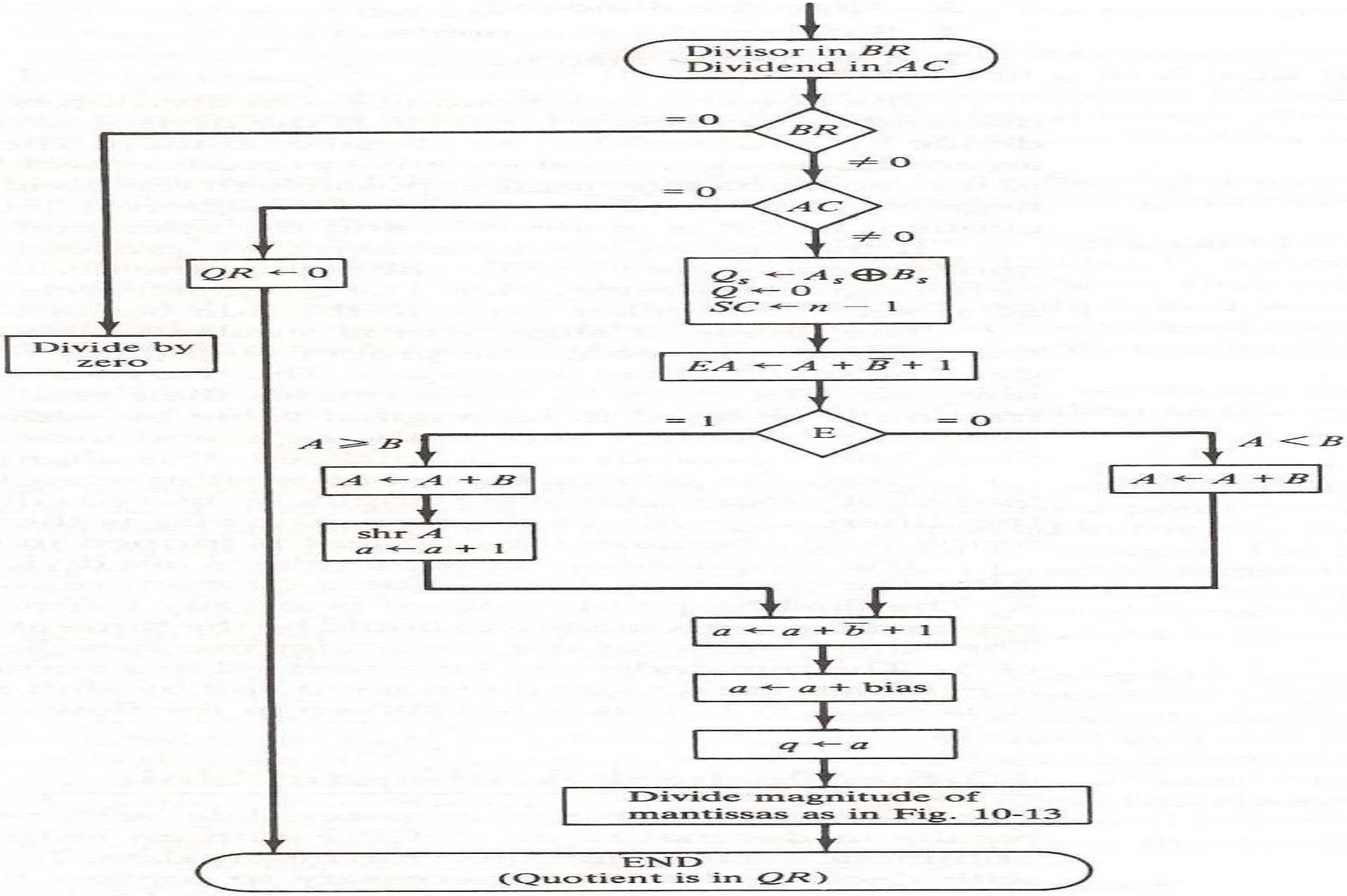


Figure 10-17 Division of floating-point numbers.

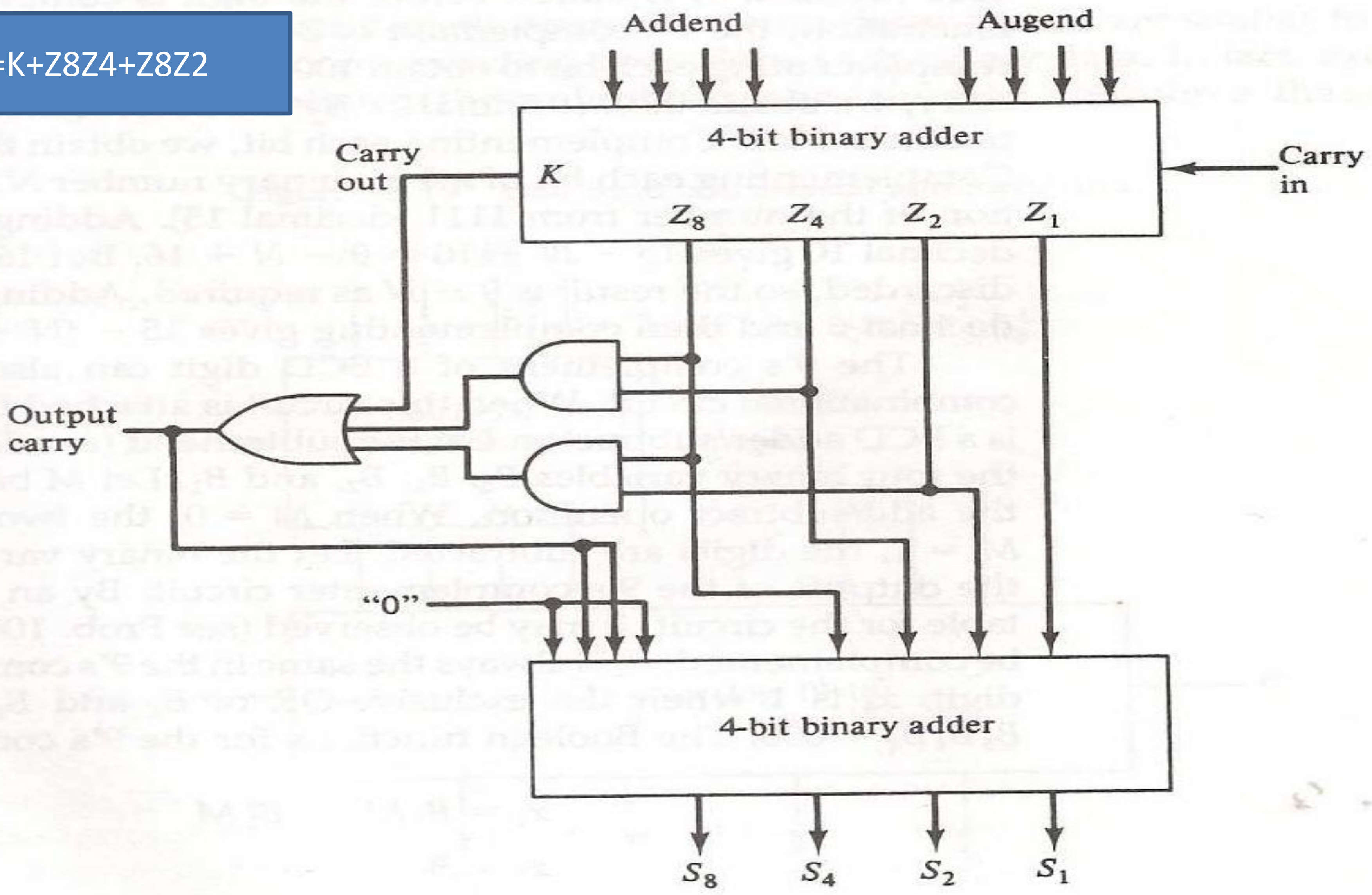
TABLE 10-4 Derivation of BCD Adder

Binary Sum					BCD Sum					
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	Decimal
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

BCD Adder

Figure 10-18 Block diagram of BCD adder.

C=K+Z8Z4+Z8Z2



BCD Subtraction

- There are two methods to obtain 9’s Complement of BCD Numbers.

1)Binary 1010 is added to each complemented digit and the carry discarded after each addition.

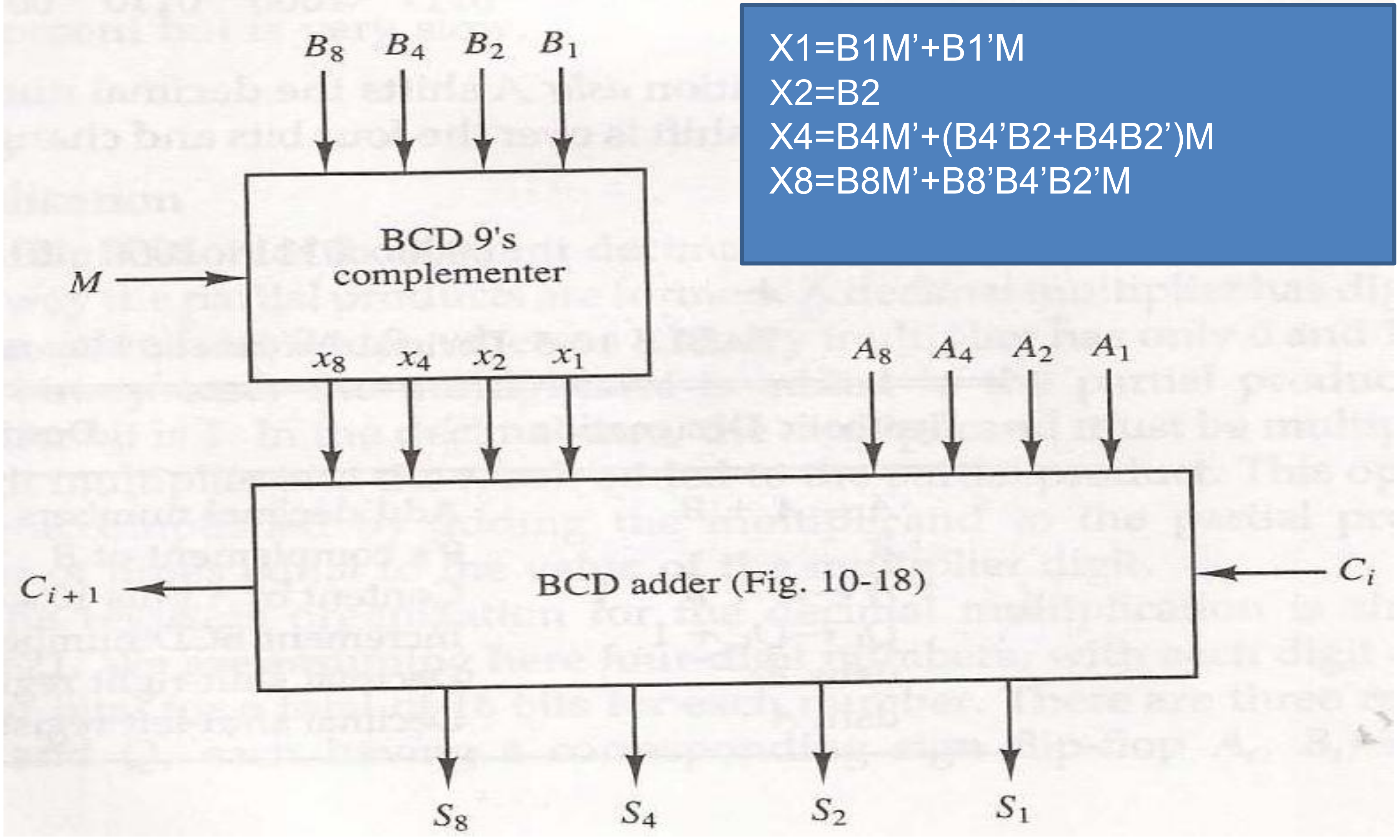
Ex: 9’s Complement of BCD 0111 is Computed by first complementing each bit to obtain 1000. adding binary 1010 and discard the end carry we obtain 0010.

2) Binary 0110 is added before the digit is complemented.

Ex: We add 0110 to 0111 to obtain 1101 complementing each bit we obtain the required result of 0010.

Decimal Arithmetic Unit

Figure 10-19 One stage of a decimal arithmetic unit.



X1=B1M’+B1’M
X2=B2
X4=B4M’+(B4’B2+B4B2’)M
X8=B8M’+B8’B4’B2’M

A decimal shift right or left is preceded by the letter *d* to indicate a shift over the four bits that hold the decimal digits. As a numerical illustration consider a register *A* holding decimal 7860 in BCD. The bit pattern of the 12 flip-flops is

0111 1000 0110 0000

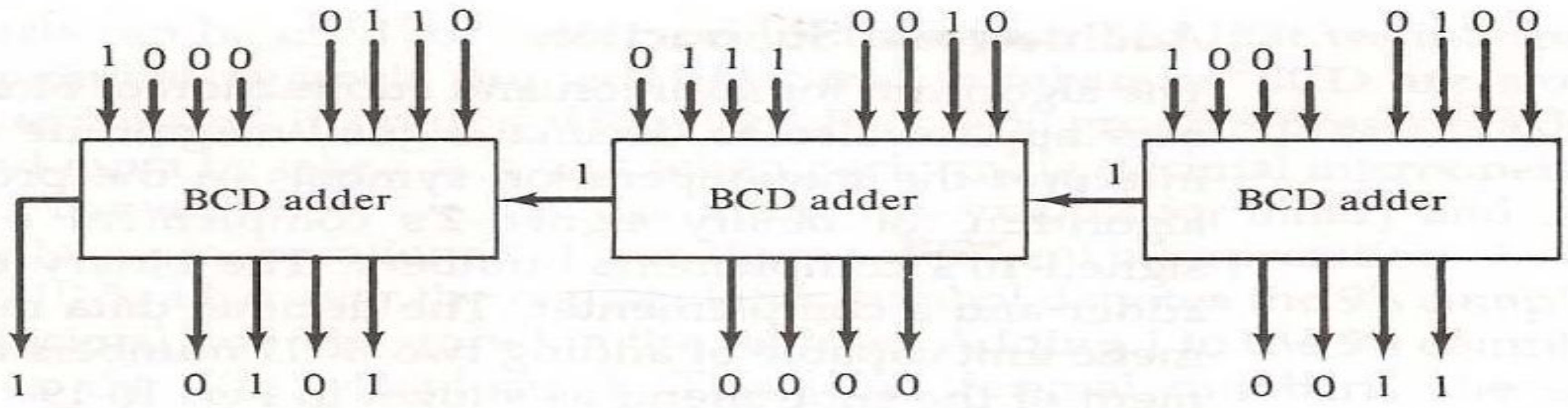
The microoperation *dshr A* shifts the decimal number one digit to the right to give 0786. This shift is over the four bits and changes the content of the register into

0000 0111 1000 0110

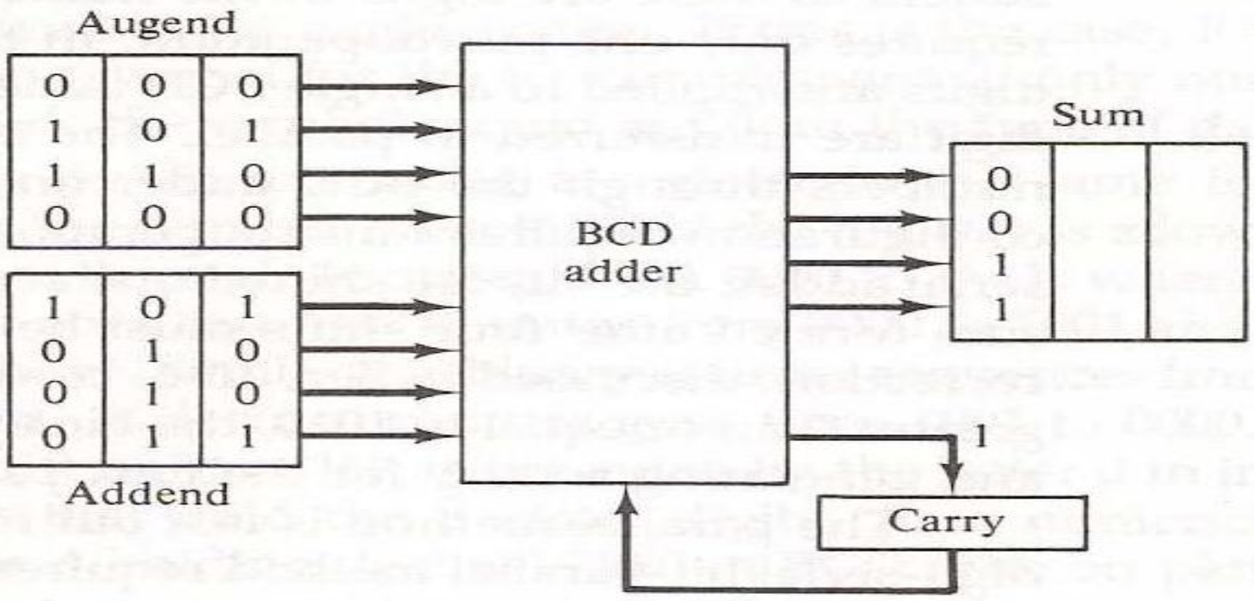
TABLE 10-5 Decimal Arithmetic Microoperation Symbols

Symbolic Designation	Description
$A \leftarrow A + B$	Add decimal numbers and transfer sum into *A*
$\overline{B}$	9's complement of *B*
$A \leftarrow A + \overline{B} + 1$	Content of *A* plus 10's complement of *B* into *A*
$Q_L \leftarrow Q_L + 1$	Increment BCD number in Q_L
dshr *A*	Decimal shift-right register *A*
dshl *A*	Decimal shift-left register *A*

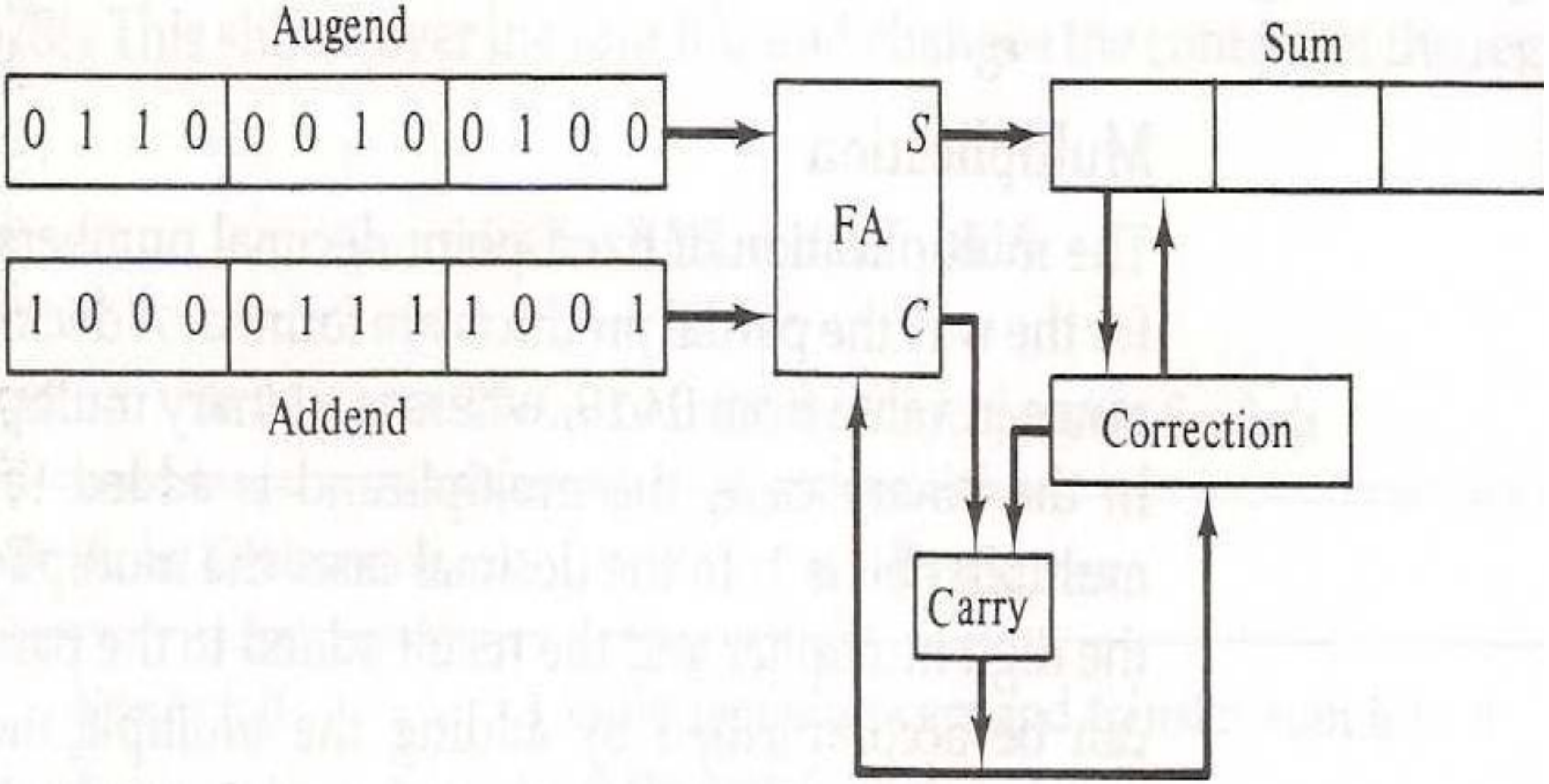
Decimal Arithmetic Addition



(a) Parallel decimal addition: 624 + 879 = 1503



(b) Digit-serial, bit-parallel decimal addition



(c) All serial decimal addition

Figure 10-20 Three ways of adding decimal numbers.

BCD Multiplication and Division Registers

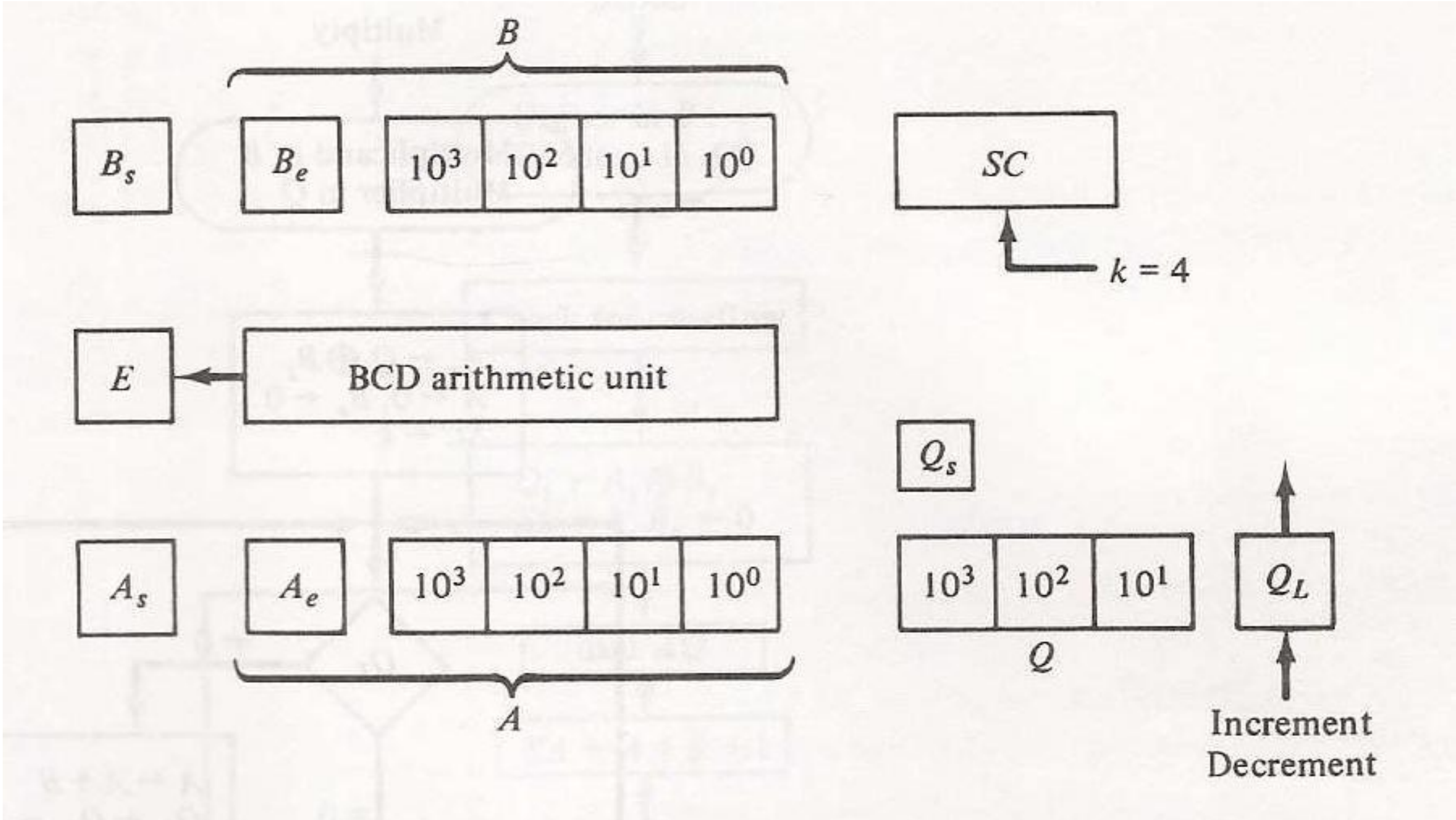


Figure 10-21 Registers for decimal arithmetic multiplication and division.

Decimal Multiplication

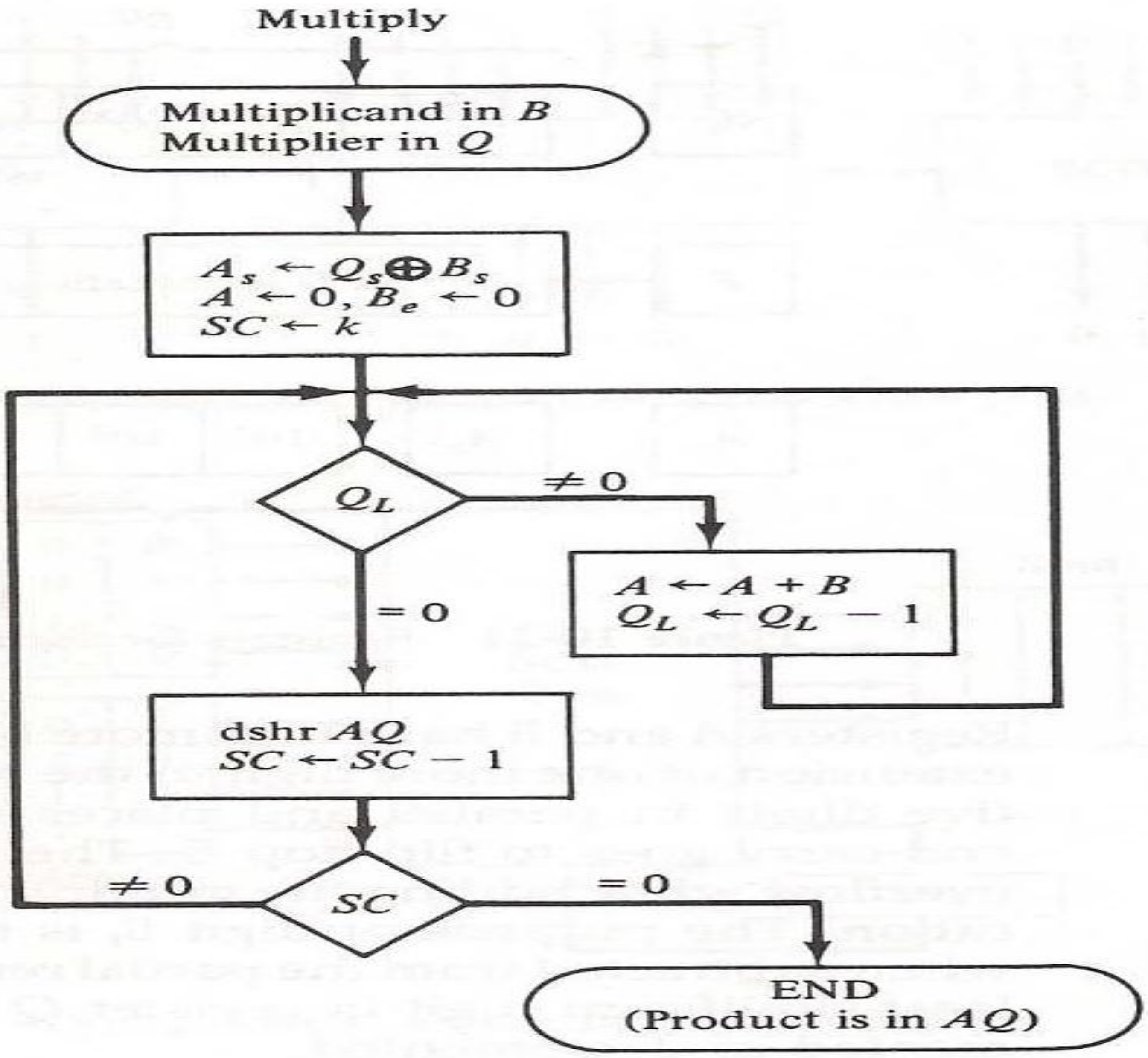


Figure 10-22 Flowchart for decimal multiplication.

Decimal Division

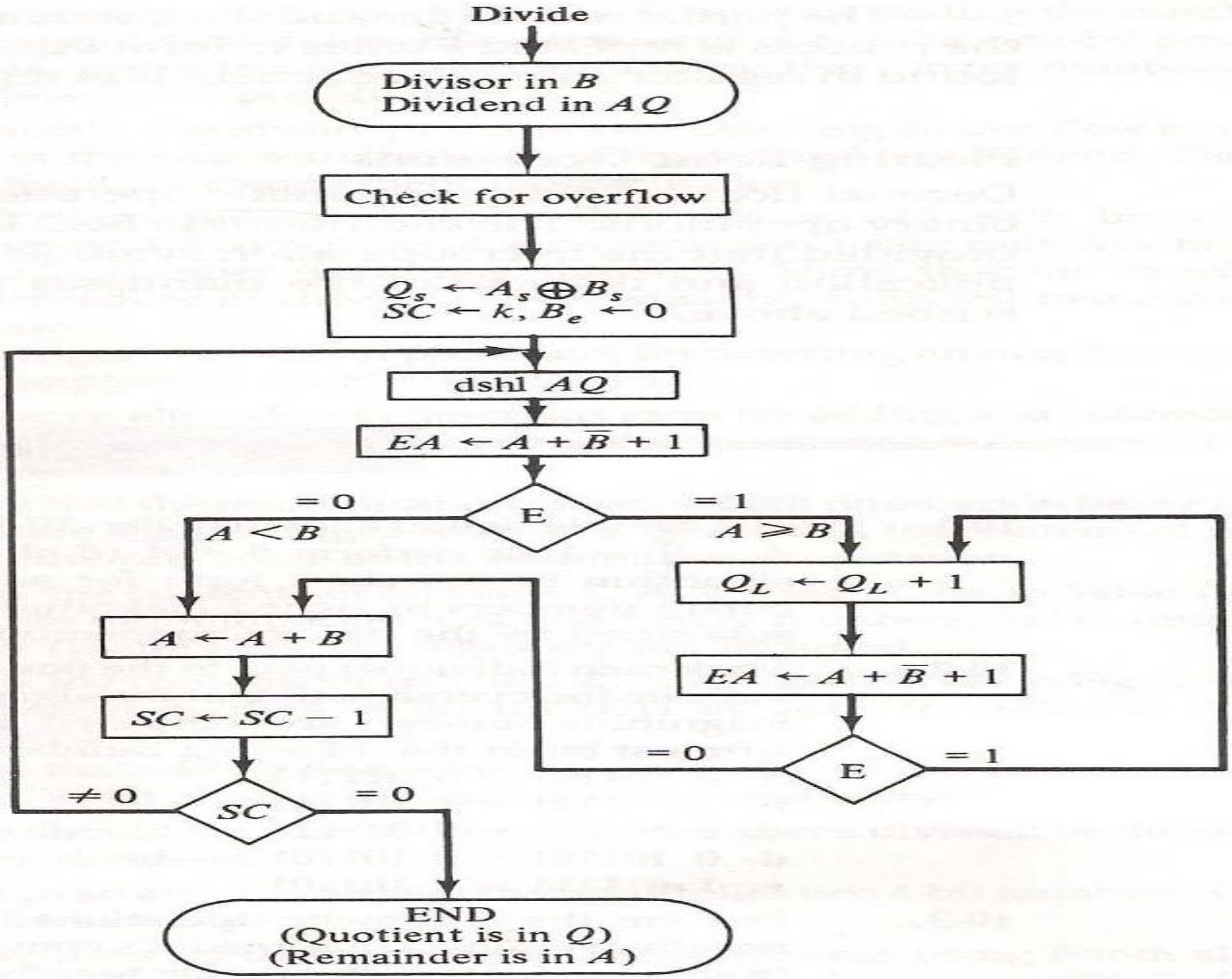


Figure 10-23 Flowchart for decimal division.

Control Unit

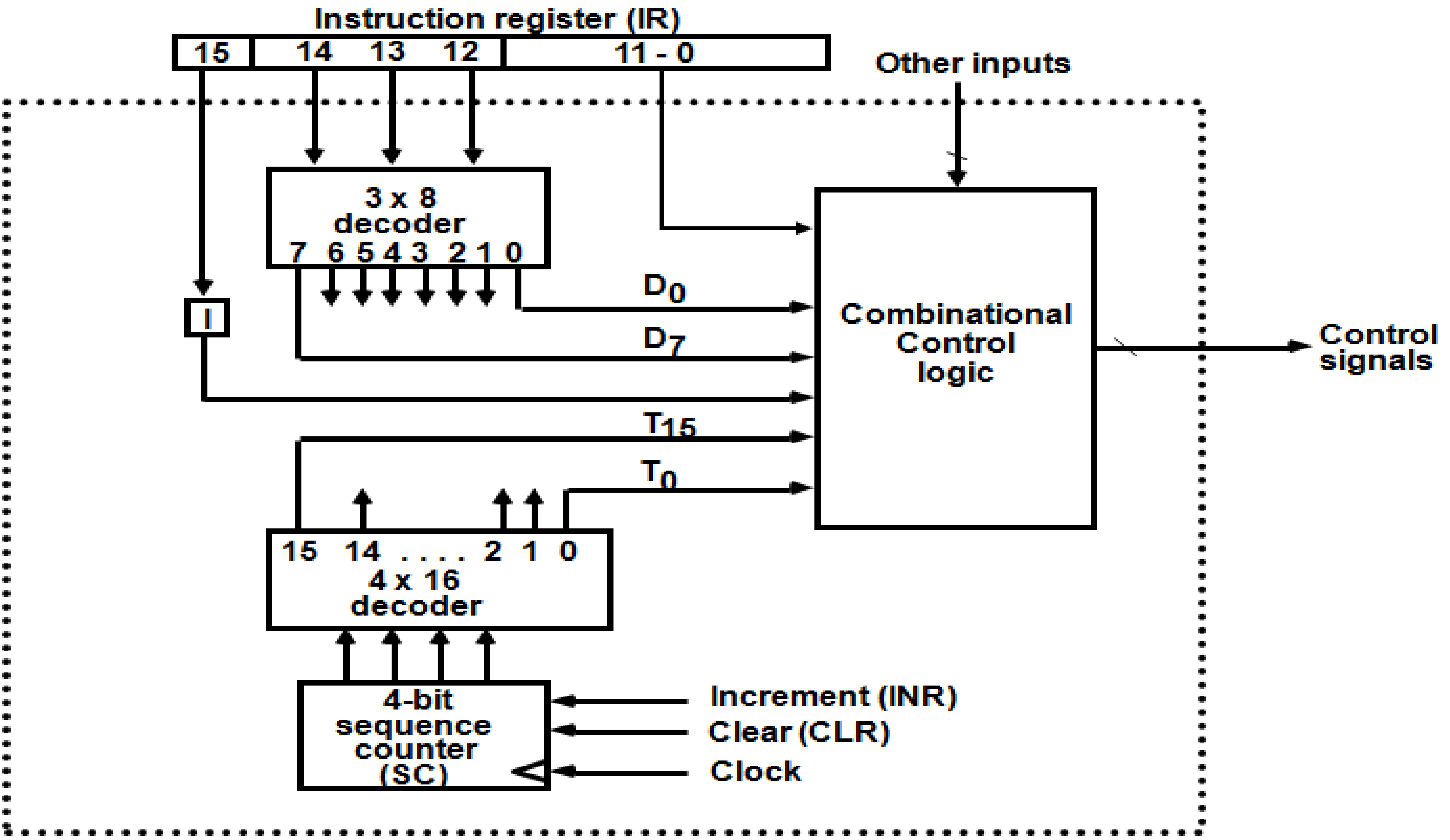


Fig: Hardwired Control Unit

Memory Locations, Addresses, and Operations

Memory Location, Addresses, and Operation

- Memory consists of many millions of storage cells, each of which can store 1 bit.
- Data is usually accessed in *n*-bit groups. *n* is called word length.

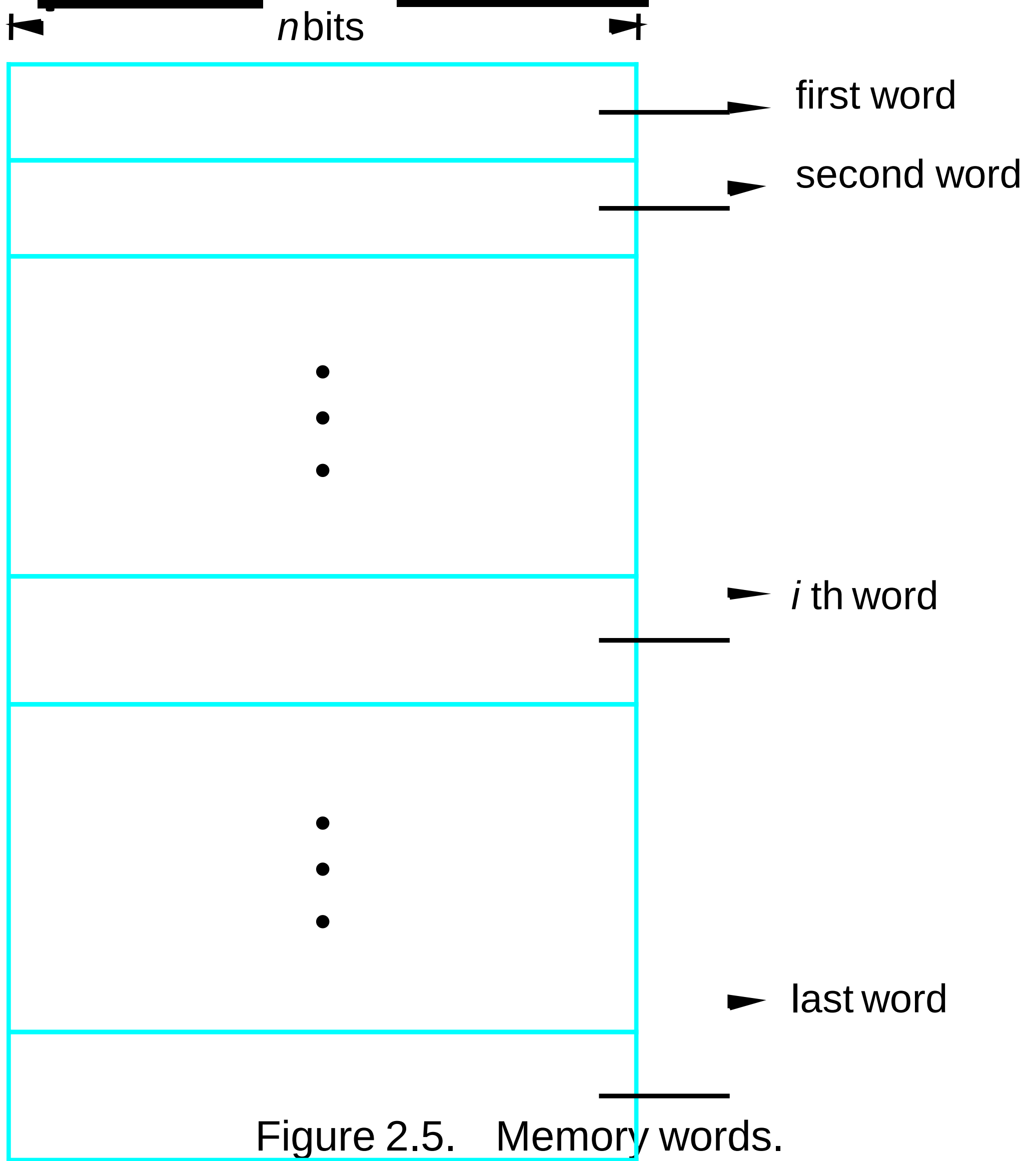
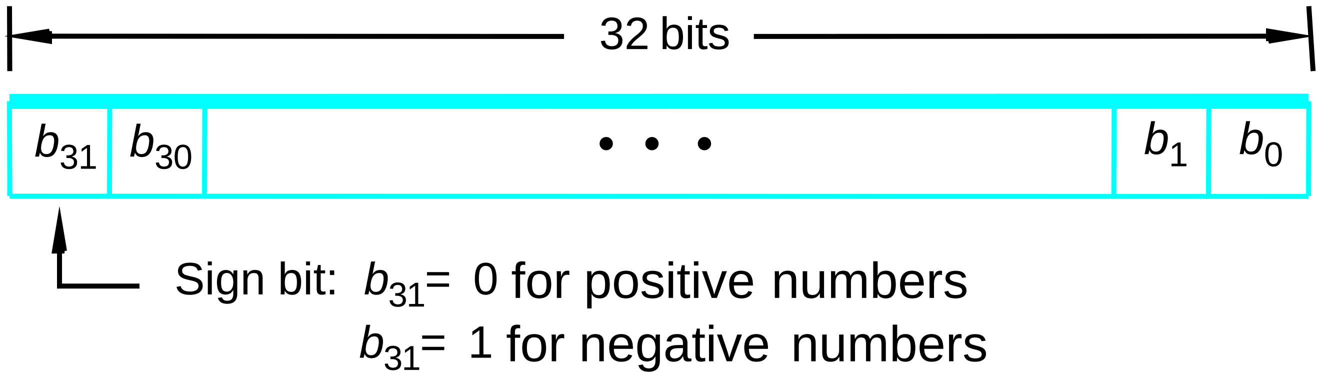


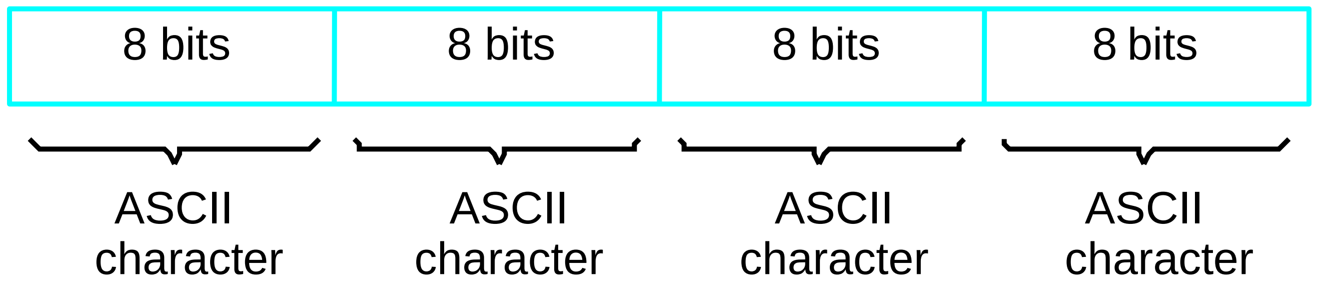
Figure 2.5. Memory words.

Memory Location, Addresses, and Operation

- 32-bit word length example


32 bits
b_{31}
b_{30}
• • •
b_1
b_0
Sign bit: $b_{31} = 0$ for positive numbers
$b_{31} = 1$ for negative numbers

(a) A signed integer



(b) Four characters

Memory Location, Addresses, and Operation

- To retrieve information from memory, either for one word or one byte (8-bit), addresses for each location are needed.
- A *k*-bit address memory has 2^k memory locations, namely $0 – 2^k-1$, called memory space.
- 24-bit memory: $2^{24} = 16,777,216 = 16M$ ($1M=2^{20}$)
- 32-bit memory: $2^{32} = 4G$ ($1G=2^{30}$)
- 1K(kilo)=2^{10}
- 1T(tera)=2^{40}

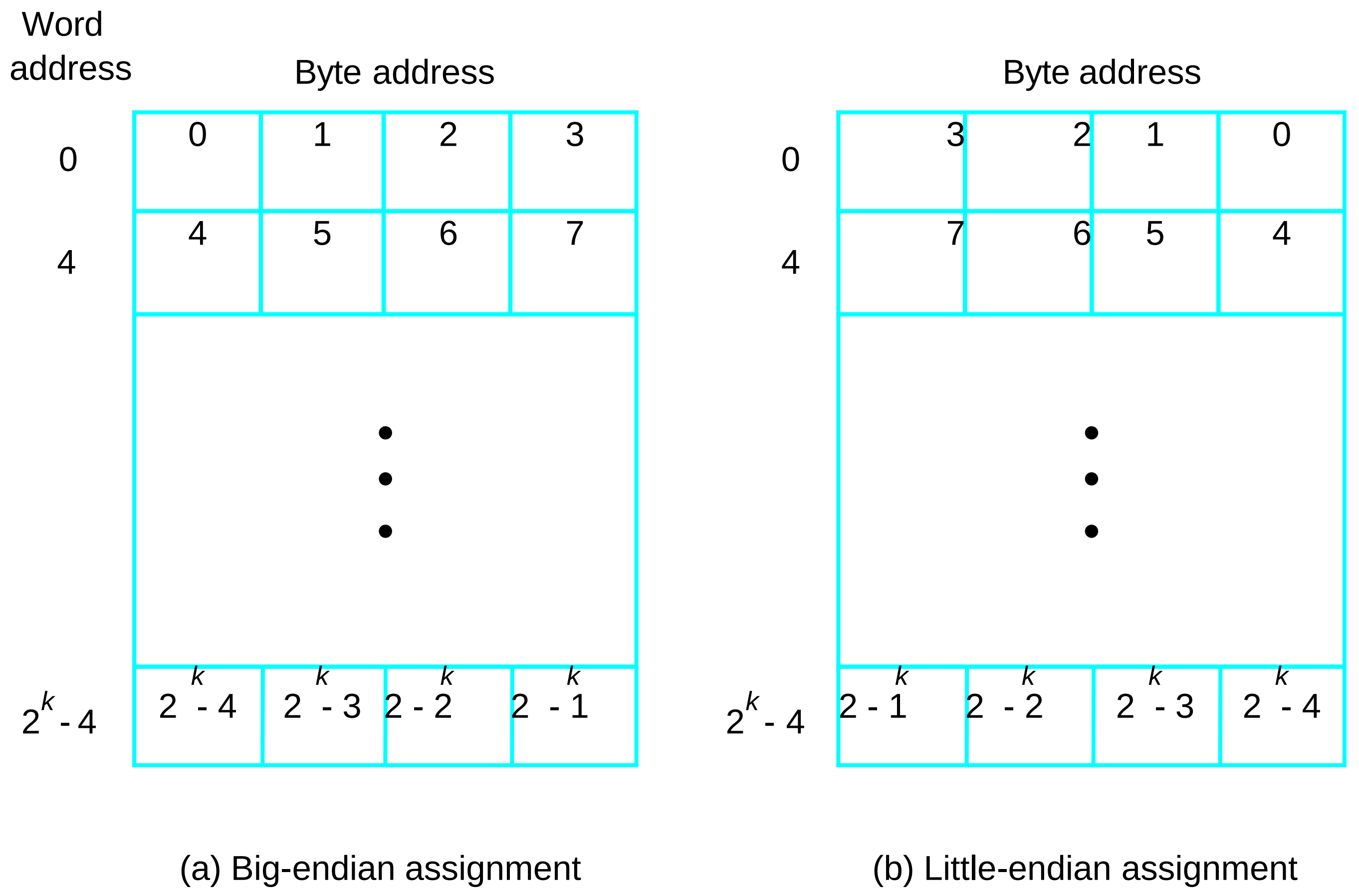
Memory Location, Addresses, and Operation

- It is impractical to assign distinct addresses to individual bit locations in the memory.
- The most practical assignment is to have successive addresses refer to successive byte locations in the memory – byte-addressable memory.
- Byte locations have addresses O, 1, 2, … If word length is 32 bits, they successive words are located at addresses O, 4, 8,…

Big-Endian and Little-Endian Assignments

Big-Endian: lower byte addresses are used for the most significant bytes of the word

Little-Endian: opposite ordering. lower byte addresses are used for the less significant bytes of the word



Word address
Byte address
0
4
0 1 2 3
4 5 6 7
$2^k - 4$
$2^k - 4$ $2^k - 3$ $2^k - 2$ $2^k - 1$
(a) Big-endian assignment
Byte address
0
4
3 2 1 0
7 6 5 4
$2^k - 4$
$2^k - 1$ $2^k - 2$ $2^k - 3$ $2^k - 4$
(b) Little-endian assignment

Figure 2.7. Byte and word addressing.

Memory Location, Addresses, and Operation

- Address ordering of bytes
- Word alignment
  - Words are said to be aligned in memory if they begin at a byte addr. that is a multiple of the num of bytes in a word.
    - 16-bit word: word addresses: O, 2, 4,….
    - 32-bit word: word addresses: O, 4, 8,….
    - 64-bit word: word addresses: O, 8,16,….
- Access numbers, characters, and character strings

Memory Operation

- Load (or Read or Fetch)
  - ➢ Copy the content. The memory content doesn’t change.
  - ➢ Address – Load
  - ➢ Registers can be used
- Store (or Write)
  - ➢ Overwrite the content in memory
  - ➢ Address and Data – Store
  - ➢ Registers can be used

Instruction and Instruction Sequencing

“Must-Perform” Operations

- Data transfers between the memory and the processor registers
- Arithmetic and logic operations on data
- Program sequencing and control
- I/O transfers

Register Transfer Notation

- Identify a location by a symbolic name standing for its hardware binary address (LOC, ‘O,…)
- Contents of a location are denoted by placing square brackets around the name of the location (R1←[LOC], ‘3 ←[‘1]+[‘2])
- Register Transfer Notation (RTN)

Assembly Language Notation

- Represent machine instructions and programs.
- Move LOC, R1 = R1←[LOC]
- Add ‘1, ‘2, ‘3 = ‘3 ←[‘1]+[‘2]

CPU Organization

- Single Accumulator
  - Result usually goes to the Accumulator
  - Accumulator has to be saved to memory quite often
- General Register
  - Registers hold operands thus reduce memory traffic
  - Register bookkeeping
- Stack
  - Operands and result are always in the stack

Instruction Formats

- Computer may have different instructions .
- The number of address fields in the instruction format depends on the internal organization of its registers.
- There are three different types of CPU organizations:

1. Single Accumulator organization
    - Basic Computer is a good example
    - Accumulator is the only general purpose register
2. General Register Organization
    - Used by most modern computer processors
    - Any of the registers can be used as the source or destination for computer operations
3. Stack Organization
    - All operations are done using the hardware stack

Instruction Formats

- Instruction Fields

1. OP-code field - specifies the operation to be performed.
2. Address field - designates memory address(es) or a processor register(s).
3. Mode field - determines how the address field is to be interpreted (to get effective address or the operand).

- The number of address fields in the instruction format depends on the internal organization of CPU.
- The three most common CPU organizations:

1. Single accumulator organization:

ADD X /* $AC \leftarrow AC + M[X]$ */

Instruction Formats

2. General register organization:

ADD R1, R2, R3	/* R1 $\leftarrow$ R2 + R3 */
ADD R1, R2	/* R1 $\leftarrow$ R1 + R2 */
MOV R1, R2	/* R1 $\leftarrow$ R2 */
ADD R1, X	/* R1 $\leftarrow$ R1 + M[X] */

3. Stack organization:

PUSH X	/* TOS $\leftarrow$ M[X] */
ADD	

Instruction Formats

- Three-Address Instructions

Program to evaluate X = (A + B) * (C + D) :

```
ADD R1, A, B      /*  R1 ←M[A] + M[B]*/
ADD R2, C, D      /*  R2 ←M[C] + M[D]*/
MUL X, R1, R2     /*  M[X] ←R1 * R2        */
```

- Results in short programs
- Instruction becomes long (many bits)

- Two-Address Instructions

Program to evaluate X = (A + B) * (C + D) :

```
MOV    R1, A      /* R1 ←M[A]          */
ADD    R1, B      /* R1 ← R1 + M[B]    */
MOV    R2, C      /* R2 ← M[C]         */
ADD    R2, D      /* R2 ← R2 + M[D]    */
MUL    R1, R2     /* R1 ← R1 * R2      */
MOV    X, R1      /* M[X] ←R1          */
```

Instruction Formats

- One-Address Instructions

- Use an implied AC register for all data manipulation

- Program to evaluate X = (A + B) * (C + D) :

```
LOAD     A     /*  AC ← M[A]          */
ADD      B     /*  AC ← AC + M[B]     */
STORE    T     /*  M[T] ← AC          */
LOAD     C     /*  AC ← M[C]          */
ADD      D     /*  AC ← AC + M[D]     */
MUL      T     /*  AC ← AC * M[T]     */
STORE    X     /*  M[X] ← AC          */
```

Instruction Formats

- Zero-Address Instructions.
  - Can be found in a stack-organized computer.
  - Program to evaluate $X = (A + B) * (C + D)$:

```
PUSH    A        /*  TOS ← A   */
PUSH    B          /*  TOS ← B */
ADD              /*  TOS ←(A + B)      */
PUSH    C        /*  TOS ←C    */
PUSH    D        /*  TOS ←D    */
ADD              /*  TOS ←(C + D)      */
MUL              /*  TOS ←(C + D) * (A + B)  */
POP     X        /*  M[X] ←TOS         */
```

Using Registers

- Registers are faster
- Shorter instructions

– The number of registers is smaller (e.g. 32 registers need 5 bits)

- Potential speedup
- Minimize the frequency with which data is moved back and forth between the memory and processor registers.

Instruction Execution and Straight-Line Sequencing

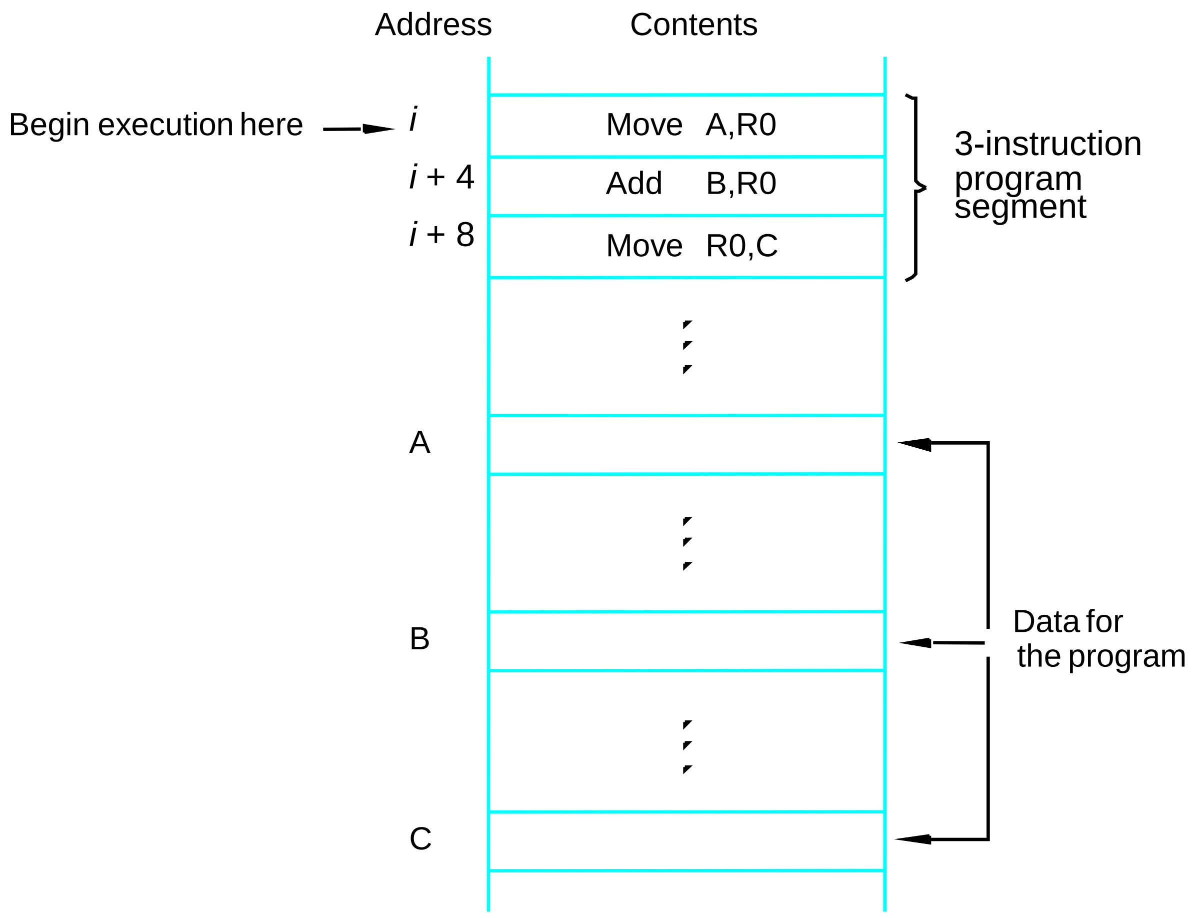


Figure 2.8. A program for C ← [A] + [B].

Assumptions:

-One memory operand per instruction

-32-bit word length

-Memory is byte addressable

-Full memory address can be directly specified in a single-word instruction

Two-phase procedure

-Instruction fetch

-Instruction execute

Page 43

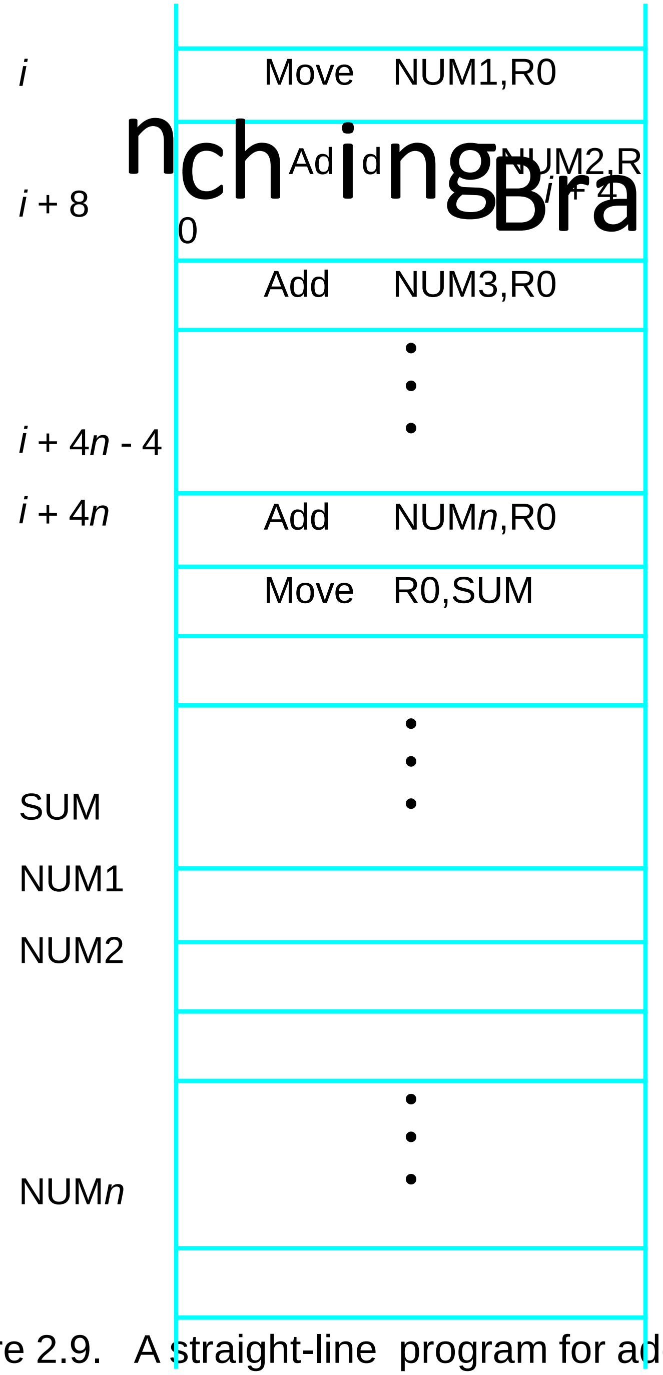
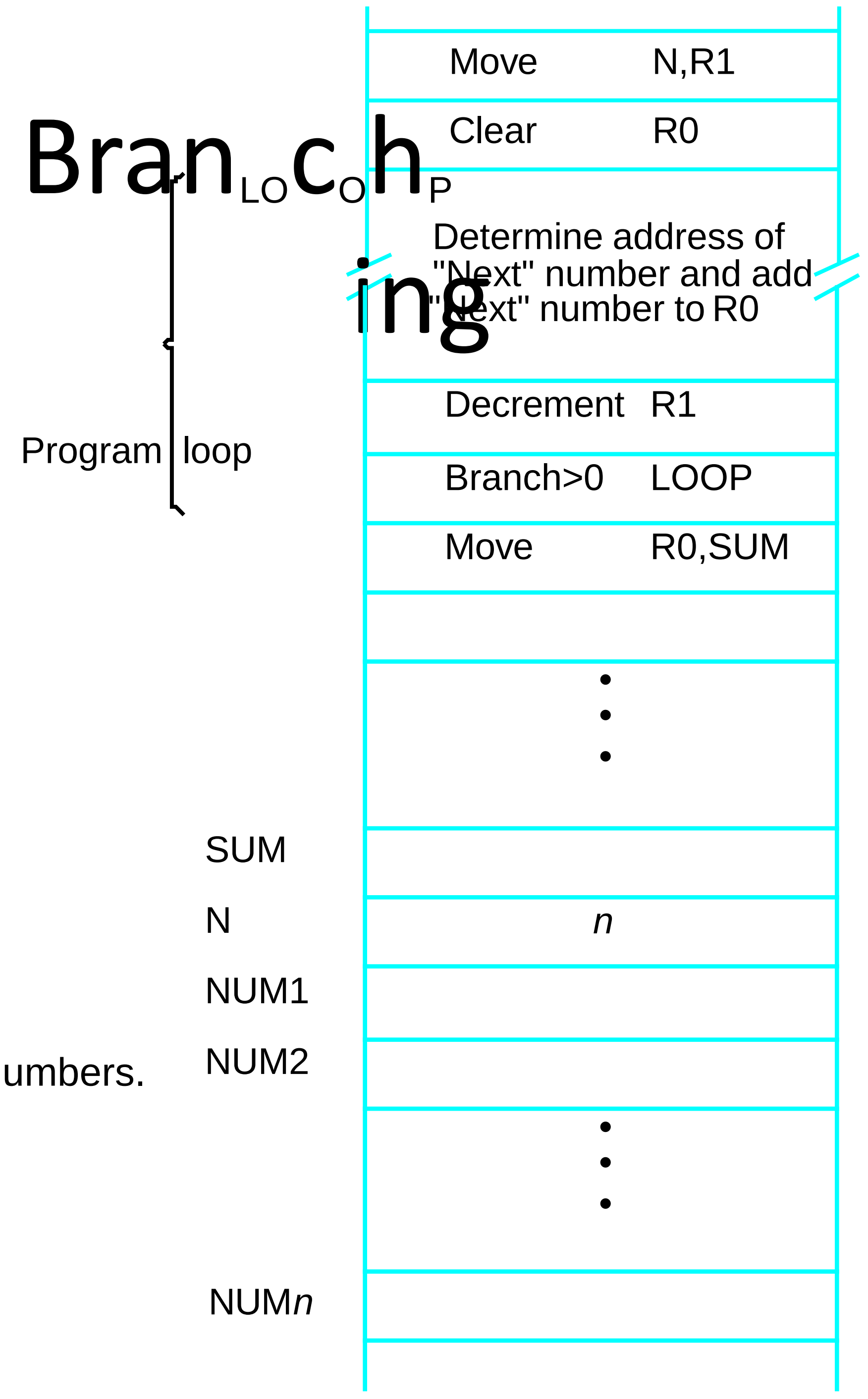


Figure 2.9. A straight-line program for adding *n* numbers.

Branching

Branch target

Conditional branch



```
        Move        N,R1
        Clear       R0
LOOP    Determine address of
        "Next" number and add
        "Next" number to R0
        Decrement   R1
        Branch>0    LOOP
        Move        R0,SUM
```

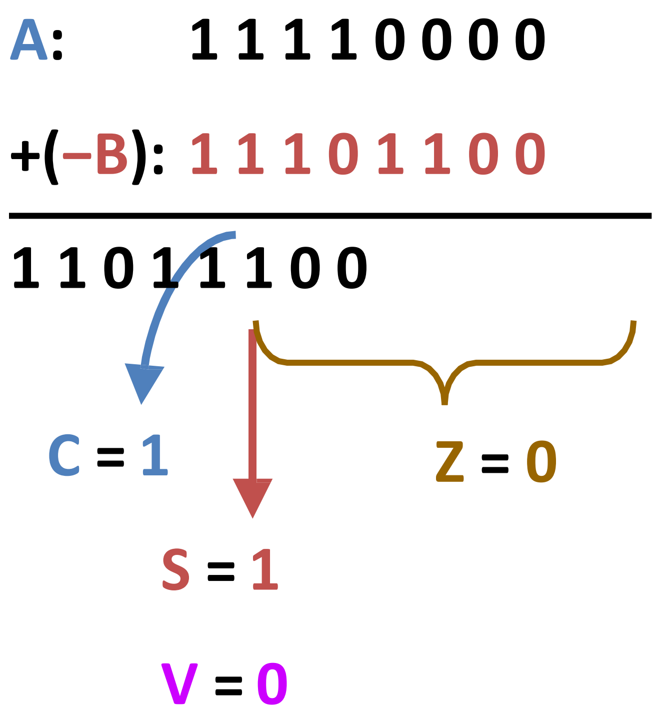
Figure 2.10. Using a loop to add *n* numbers.

Condition Codes

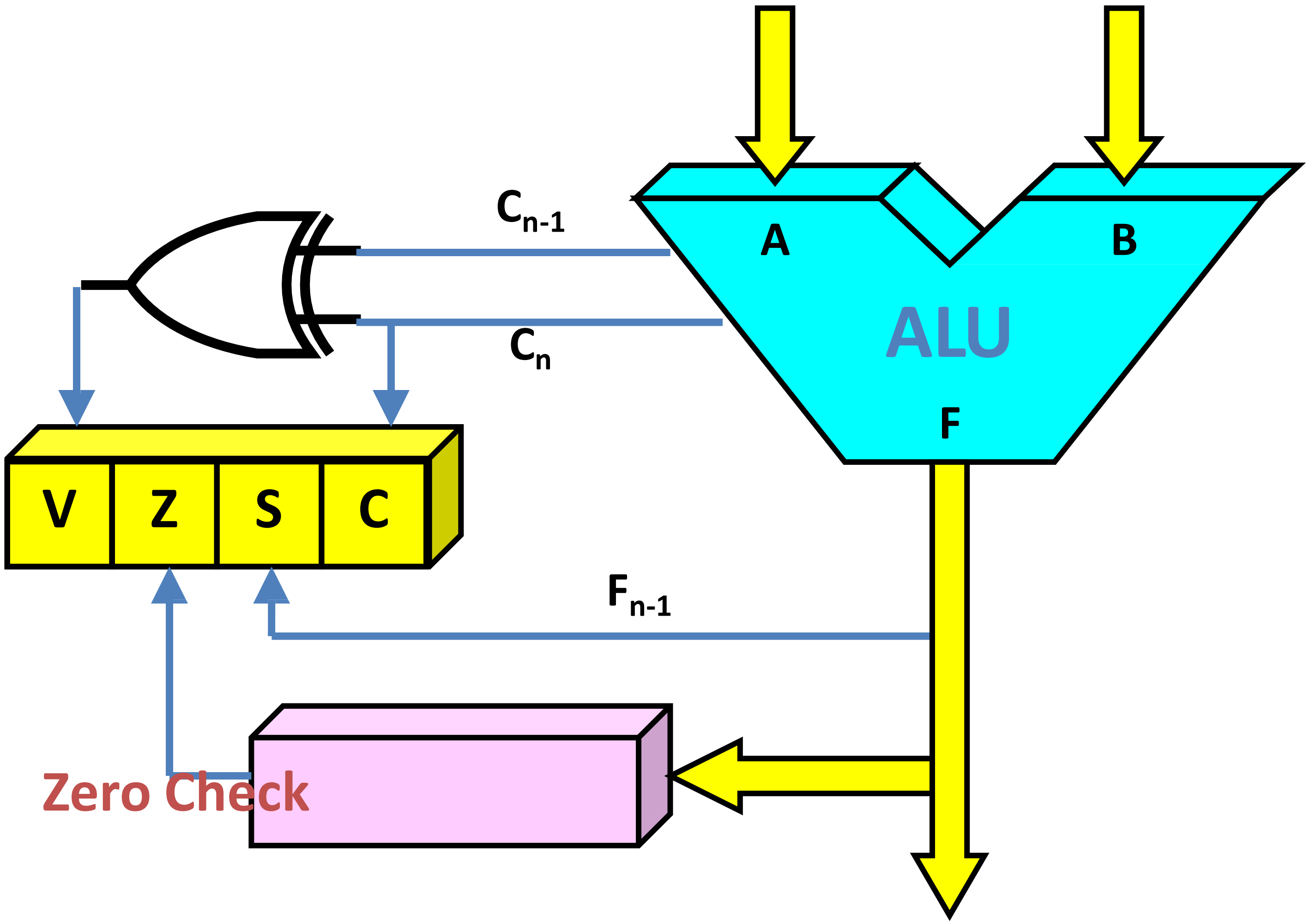
- Condition code flags
- Condition code register / status register
- N (negative)
- Z (zero)
- V (overflow)
- C (carry)
- Different instructions affect different flags

Conditional Branch Instructions

- Example:
  - A: 1 1 1 1 0 0 0 0
  - B: 0 0 0 1 0 1 0 0



Status Bits



Addressing Modes

- The way the operands are chosen during program execution is dependent on the addressing mode of the instruction .

- Specifies a rule for interpreting or modifying the address field of the instruction (before the operand is actually referenced).

Need for different addressing modes

1. To give programming flexibility to the user i.e pointers to memory,Counters for loop control,indexing of data,program relocation.
2. To use the bits in the address field of the instruction efficiently i.e reduce the number of bits.

Addressing Modes

1.Implied Mode

- Address of the operands are specified implicitly in the definition of the instruction.
- No need to specify address in the instruction.
- EA = AC, or EA = Stack[SP].
- Examples CLA, CME, INP

2. Immediate Mode

- Instead of specifying the address of the operand, operand itself is specified.
- No need to specify address in the instruction
- Operand itself needs to be specified
- Fast to acquire an operand

Addressing Modes

3.Register Mode

- Address specified in the instruction is the register address
- Designated operand need to be in a register
- Shorter address than the memory address
- Saving address field in the instruction
- Faster to acquire an operand than the memory addressing
- EA = IR(R) (IR(R): Register field of IR)

4.Register Indirect Mode

- Instruction specifies a register which contains the memory address of the operand .
- Saving instruction bits since register address is shorter than the memory address.
- Slower to acquire an operand than both the register addressing or memory addressing.
- EA = [IR(R)] ([x]: Content of x).

Addressing Modes

4.Autoincrement or Auto decrement Mode

- When the address in the register is used to access memory, the value in the register is incremented or decremented by 1 automatically .

5.Relative Addressing Modes

- The Address fields of an instruction specifies the part of the address (abbreviated address) which can be used along with a designated register to calculate the address of the operand.
- Address field of the instruction is short.
- Large physical memory can be accessed with a small number of address bits.
- EA = f(IR(address), R)

Addressing Modes

- 3 Different Relative Addressing Modes depending on R;

  1. PC Relative Addressing Mode (R = PC)

     Ex: EA = PC + IR(address)

  2. Indexed Addressing Mode (R = IX, where IX: Index Register)

     Ex: EA = IX + IR(address)

  3. Base Register Addressing Mode

     (R = BAR, where BAR: Base Address Register)

     Ex: EA = BAR + IR(address)

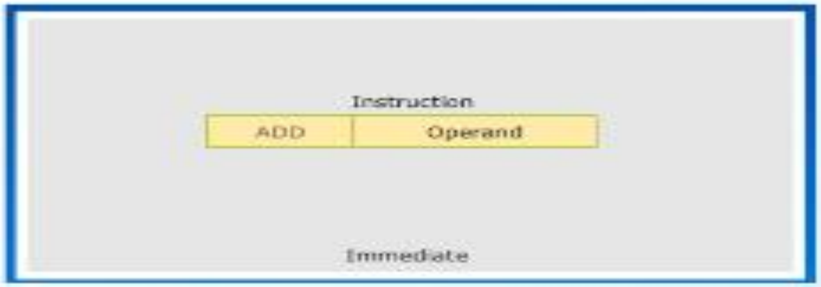
Addressing Modes

6. Direct Address Mode
   - Instruction specifies the memory address which can be used directly to access the memory.
   - Faster than the other memory addressing modes.
   - Too many bits are needed to specify the address for a large physical memory space.
   - EA = IR(addr) (IR(addr): address field of IR)

7. Indirect Addressing Mode
   - The address field of an instruction specifies the address of a memory location that contains the address of the operand.
   - When the abbreviated address is used large physical memory can be addressed with a relatively small number of bits.
   - Slow to acquire an operand because of an additional memory access
   - EA = M[IR(address)]

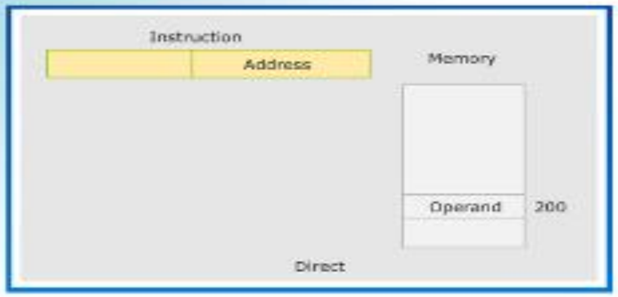
Addressing Modes

Immediate mode



Example: ADD 5

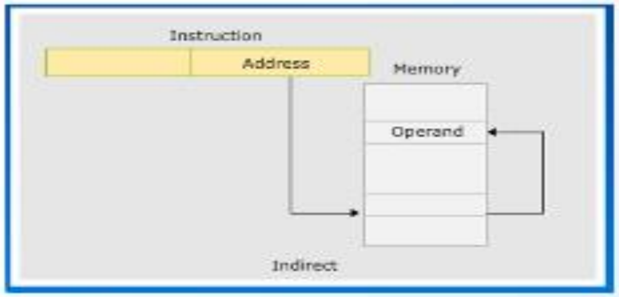
Direct mode



Example: ADD 200

Where 200 is a 12 bit address

Indirect mode

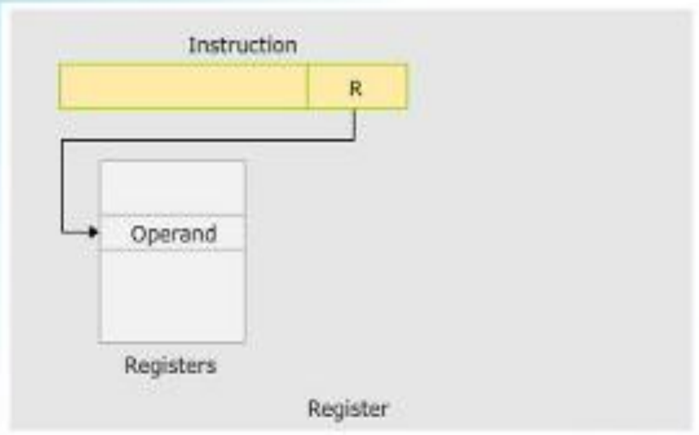


Example: ADD 200

Where 200 is a 12 bit address that holds the effective address

Addressing Modes

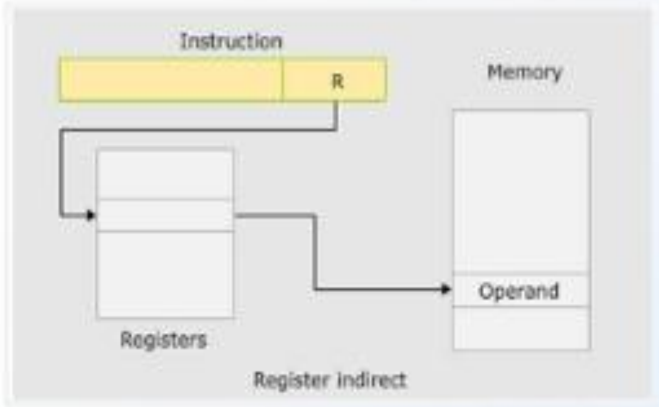
Register mode



Example: ADD R1

Where R1 is a register that contains the operand

Register indirect mode

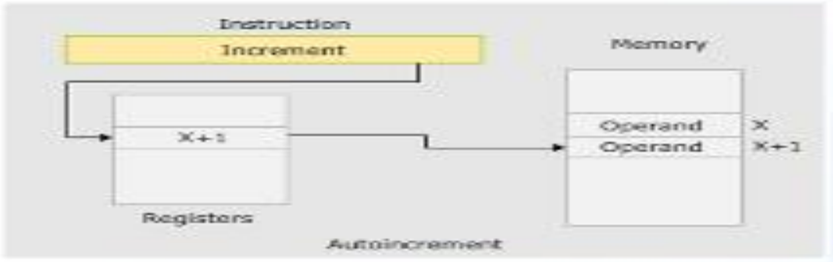


Example: ADD R1

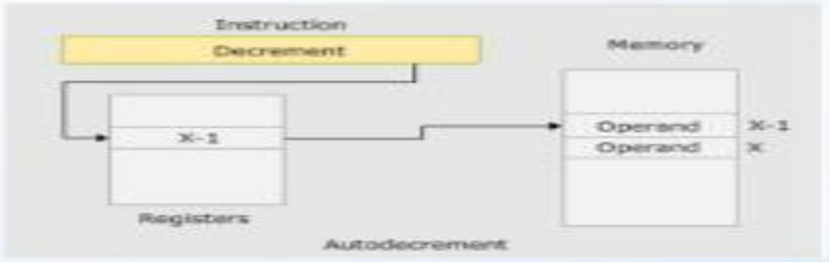
Where R1 is a register that contains the effective address of operand

Addressing Modes

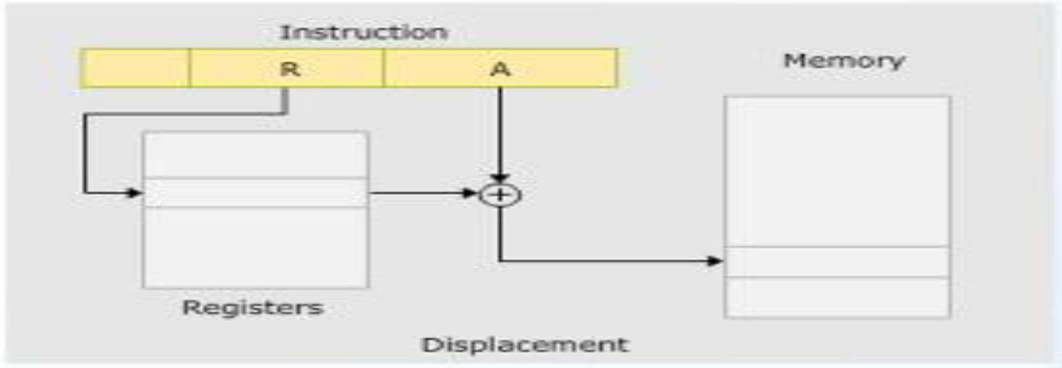
Auto increment mode



Auto decrement mode

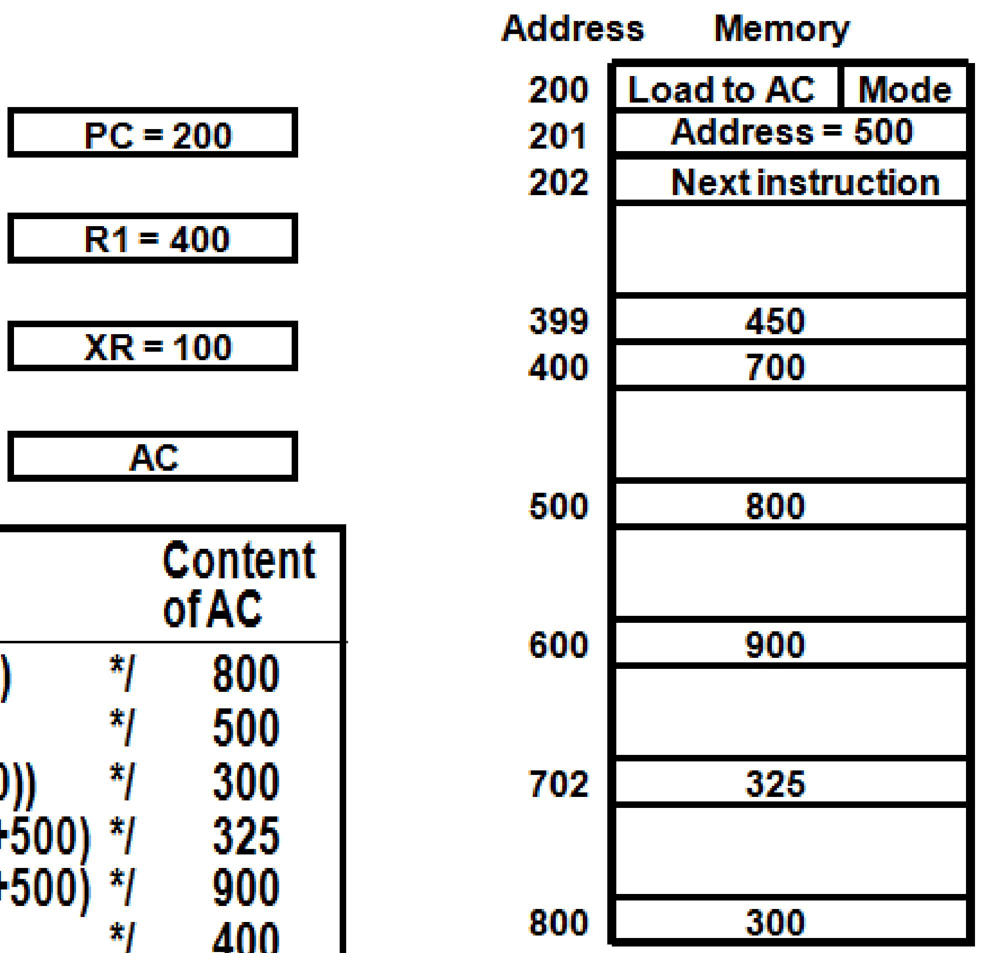


Displacement mode



Addressing Modes

Example :



Addressing Mode	Effective Address		Content of AC
Direct address	500	/* AC ← (500) */	800
Immediate operand	-	/* AC ← 500 */	500
Indirect address	800	/* AC ← ((500)) */	300
Relative address	702	/* AC ← (PC+500) */	325
Indexed address	600	/* AC ← (RX+500) */	900
Register	-	/* AC ← R1 */	400
Register indirect	400	/* AC ← (R1) */	700
Autoincrement	400	/* AC ← (R1)+ */	700
Autodecrement	399	/* AC ← -(R) */	450

Data Transfer and Manipulation Instructions

- Computer instructions can be classified into three categories.
  1. Data Transfer Instructions
  2. Data Manipulation Instructions
  3. Program Control Instructions
- Data Transfer Instructions transfer the data from one location to another location.
- Data manipulation instructions performs arithmetic ,logic and shift operations on the data.
- Program control instructions provide decision making and change the path taken by the program when executed in the computer.

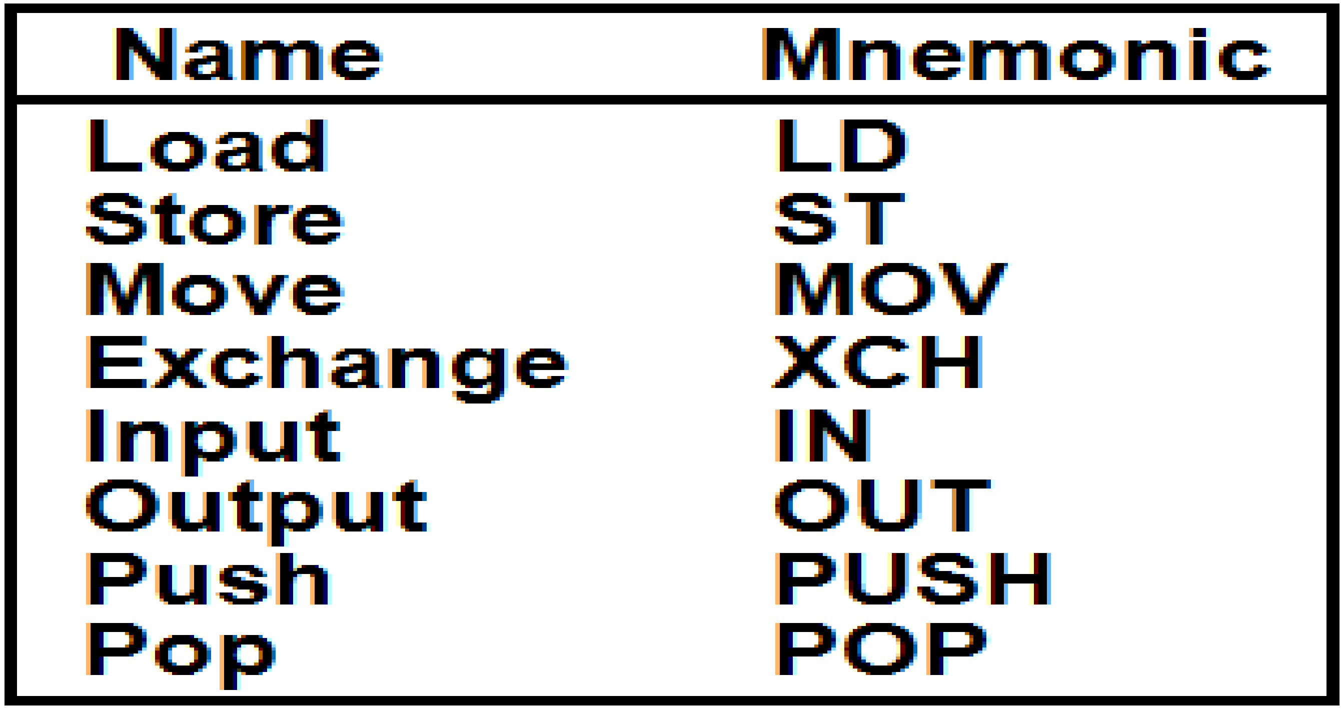
Data Transfer Instructions

- Data Transfer instructions move the data between

  Memory ---- Processor register

  Processor register ---- Input/output

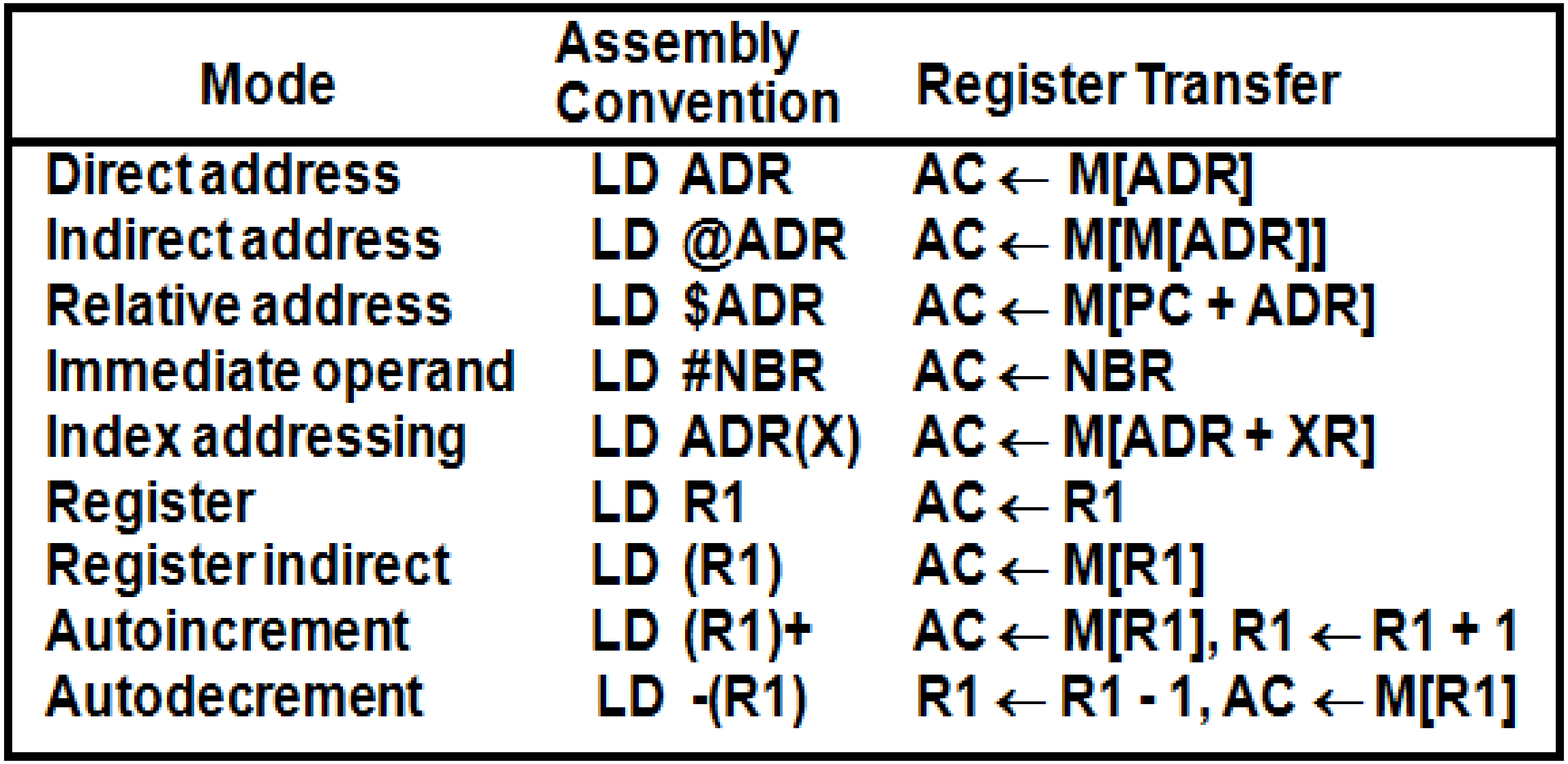
  Processor registers ---- Processor registers



Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Table : Typical data Transfer Instructions

Data Transfer Instructions



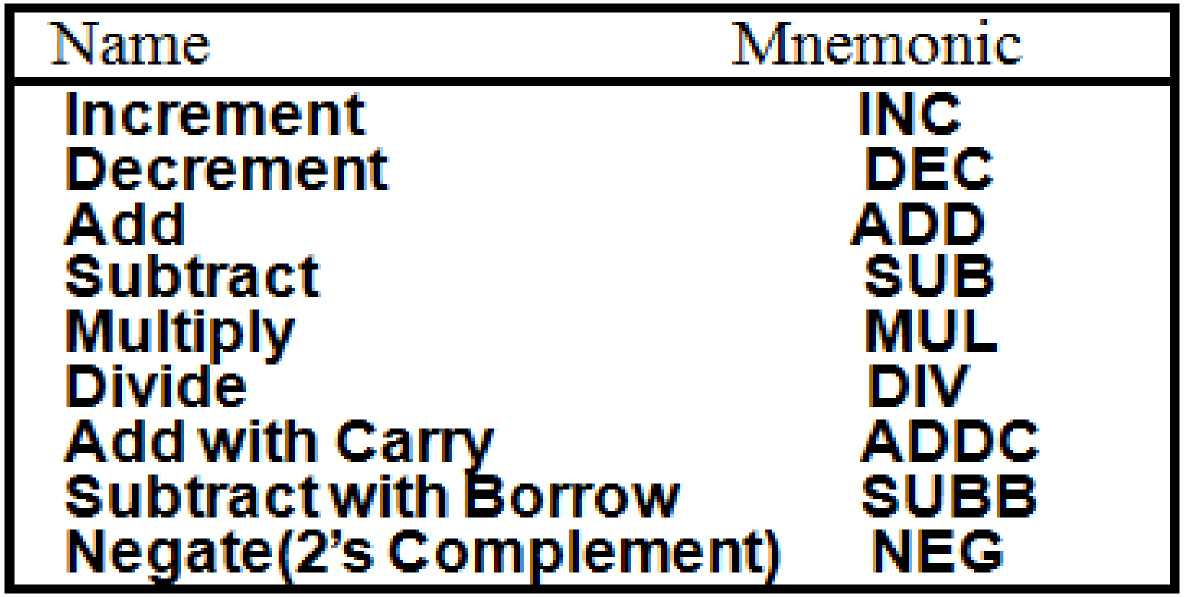
Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	AC ← M[ADR]
Indirect address	LD @ADR	AC ← M[M[ADR]]
Relative address	LD $ADR	AC ← M[PC + ADR]
Immediate operand	LD #NBR	AC ← NBR
Index addressing	LD ADR(X)	AC ← M[ADR + XR]
Register	LD R1	AC ← R1
Register indirect	LD (R1)	AC ← M[R1]
Autoincrement	LD (R1)+	AC ← M[R1], R1 ← R1 + 1
Autodecrement	LD -(R1)	R1 ← R1 - 1, AC ← M[R1]

Table : Addressing modes for load instruction

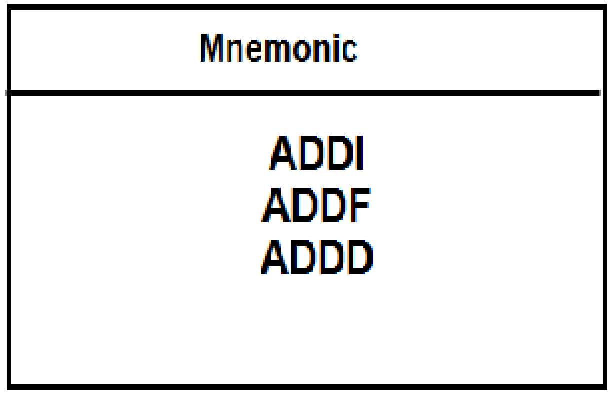
Data Manipulation Instructions

- Data Manipulation Instructions are of three basic types.
  1. Arithmetic Instructions
  2. Logical and Bit Manipulation Instructions
  3. Shift Instructions

1. Arithmetic Instructions



Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with Carry	ADDC
Subtract with Borrow	SUBB
Negate(2's Complement)	NEG

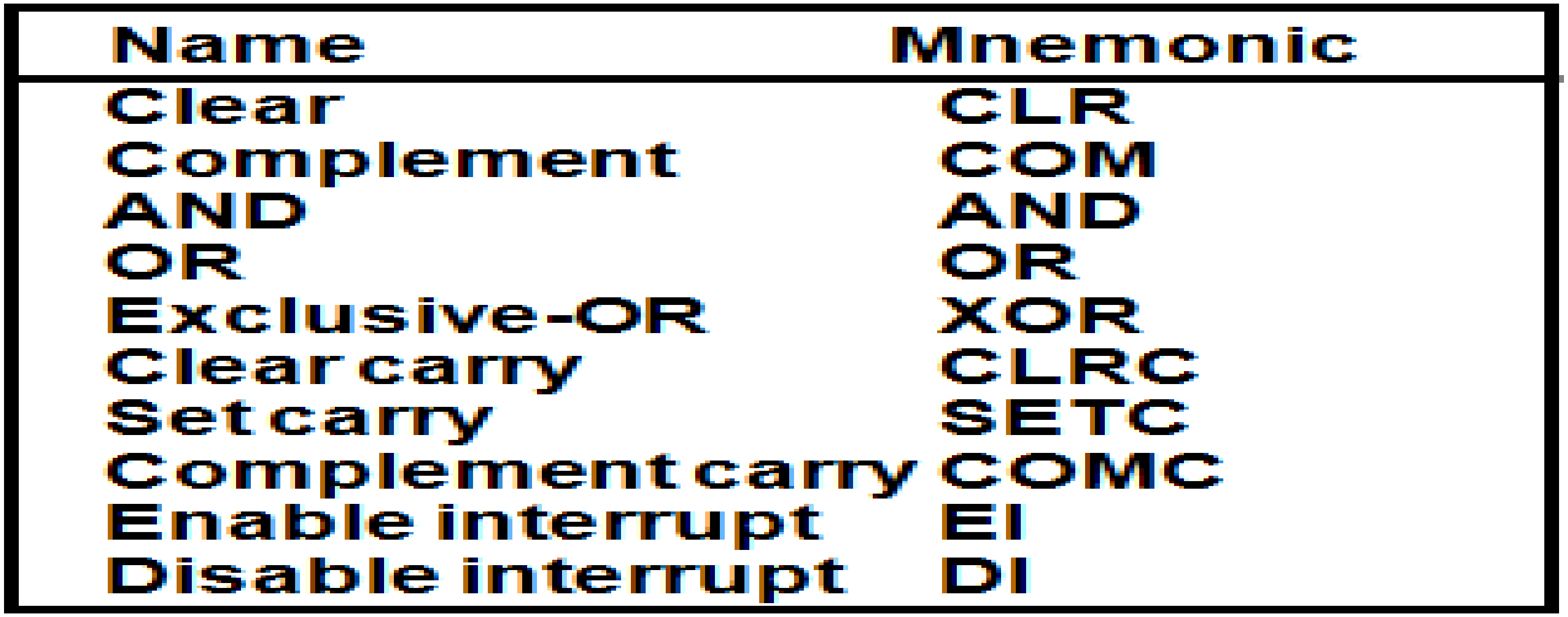


Mnemonic
ADDI
ADDF
ADDD

Table: Typical Arithmetic Instructions

Data Manipulation Instructions

2. Logical and Bit manipulation Instructions



Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Table : Logical and Bit Manipulation Instructions

Data Manipulation Instructions

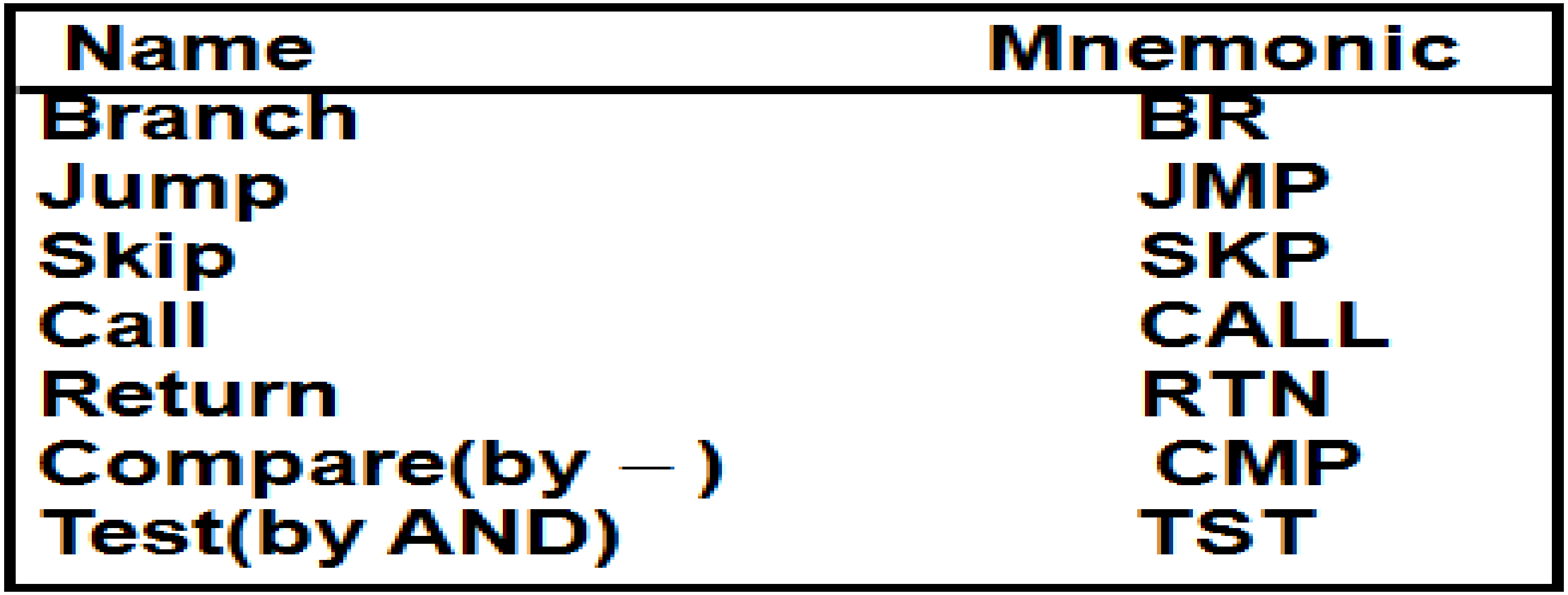
Shift Instructions

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right thru carry	RORC
Rotate left thru carry	ROLC

Op REG TYPE RL COUNT- ShiftInstruction

Program Control

- When the program control instruction is executed it change the address value in the program counter and cause the flow of control to be altered.



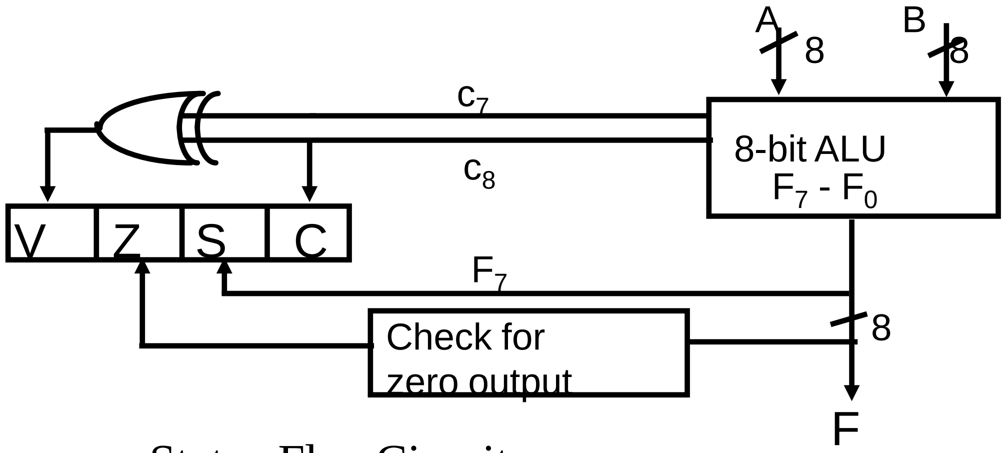
Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RTN
Compare(by –)	CMP
Test(by AND)	TST

Table: Program Control Instructions

- Branch and Jump instructions are conditional and Unconditional Instructions.
- CMP and Test set some of the bits in PSW(Processor status Word).

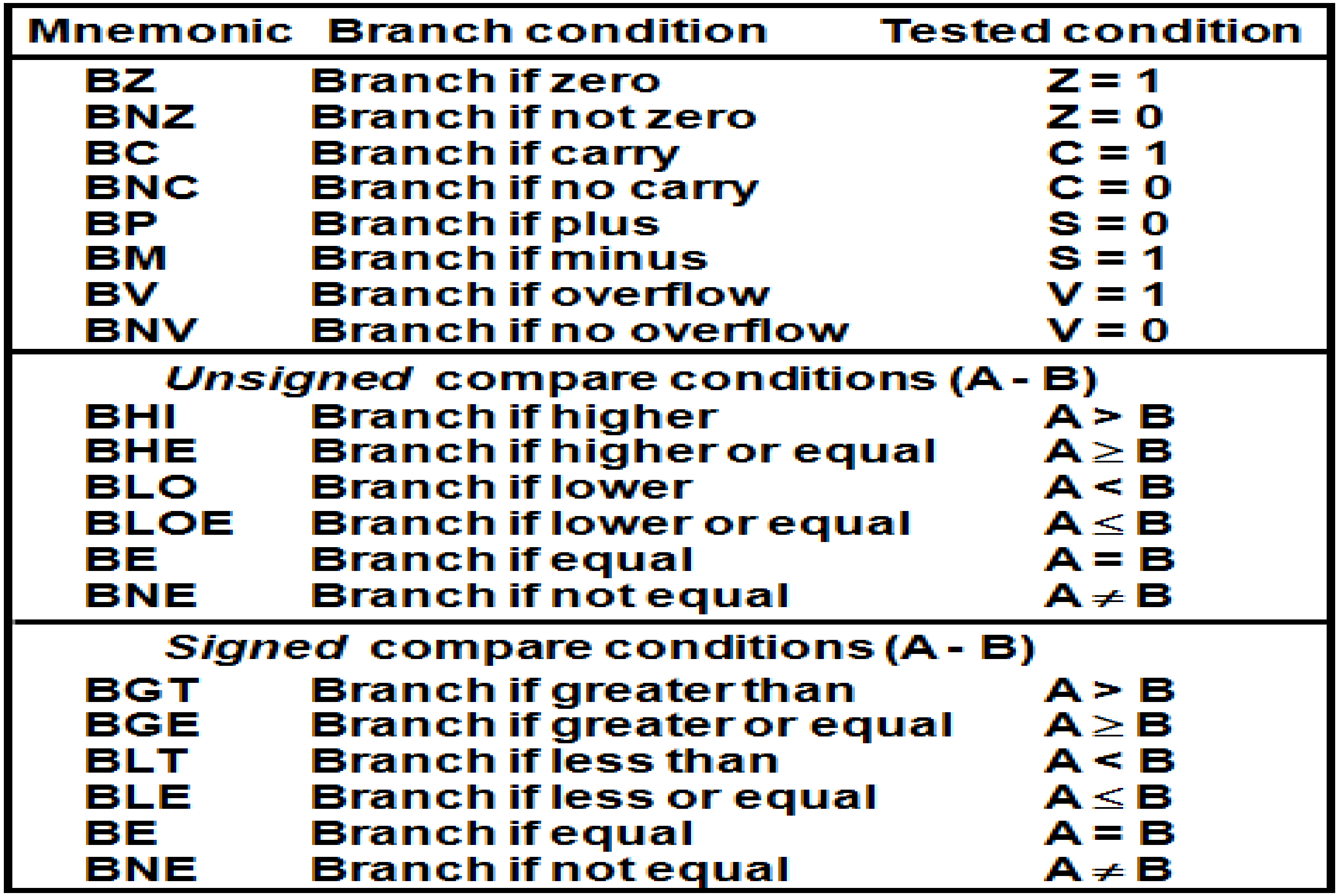
Program Control

- In Basic Computer, the processor had several (status) flags – 1 bit value that indicated various information about the processor's state – E, FGI, FGO, I, IEN, R.
- In some processors, flags like these are often combined into a register – the processor status register (PSR); sometimes called a processor status word (PSW).
- Common flags in PSW are
  - C (Carry): Set to 1 if the carry out of the ALU is 1
  - S (Sign): The MSB bit of the ALU's output
  - Z (Zero): Set to 1 if the ALU's output is all 0's
  - V (Overflow): Set to 1 if there is an overflow



Status Flag Circuit

Program Control



Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	$Z = 1$
BNZ	Branch if not zero	$Z = 0$
BC	Branch if carry	$C = 1$
BNC	Branch if no carry	$C = 0$
BP	Branch if plus	$S = 0$
BM	Branch if minus	$S = 1$
BV	Branch if overflow	$V = 1$
BNV	Branch if no overflow	$V = 0$
	Unsigned compare conditions (A - B)	
BHI	Branch if higher	$A > B$
BHE	Branch if higher or equal	$A \geq B$
BLO	Branch if lower	$A < B$
BLOE	Branch if lower or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$
	Signed compare conditions (A - B)	
BGT	Branch if greater than	$A > B$
BGE	Branch if greater or equal	$A \geq B$
BLT	Branch if less than	$A < B$
BLE	Branch if less or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$

Table: conditional branch instructions

Program Control

Example:- A: 11110000 =240 B: 00010100 =20

Unsigned

```
      A   : 11110000
    B'+1: 11101100
     A-B: 11011100      =220
   C=1  S=1  V=0    Z=0
```

Signed

```
 A   :   11110000        =-16
 B   :   00010100        =20
A-B  :   11011100     2's Complement of(36)
```

Subroutine Call and Return

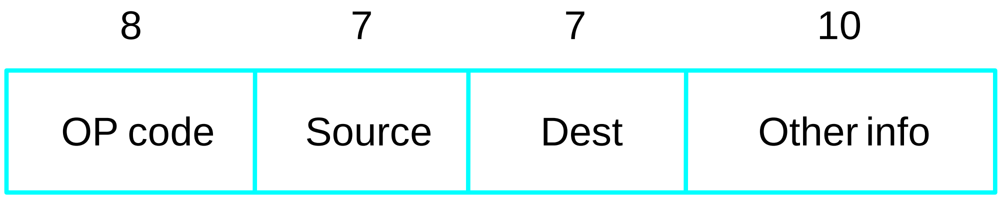
- Call subroutine
  - Jump to subroutine
  - Branch to subroutine
  - Branch and save return address
- Two Most Important Operations are
  * Branch to the beginning of the Subroutine
    - Same as the Branch or Conditional Branch
  * Save the Return Address to get the address of the location in the Calling Program upon exit from the Subroutine
- Return Subroutine from memory.

```
CALL
         SP ← SP - 1
         M[SP] ← PC
      PC ← EA

RTN
         PC ← M[SP]
          SP ← SP + 1
```

Encoding of Machine Instructions

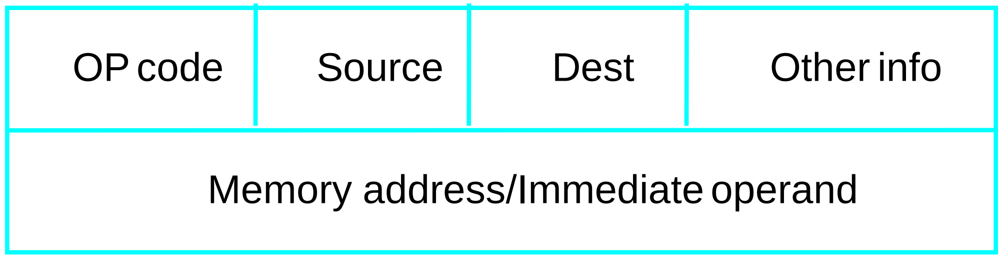
- Assembly language program needs to be converted into machine instructions. (ADD = 0100 in ARM instruction set)
- In the previous section, an assumption was made that all instructions are one word in length.
- OP code: the type of operation to be performed and the type of operands used may be specified using an encoded binary pattern
- Suppose 32-bit word length, 8-bit OP code (how many instructions can we have?), 16 registers in total (how many bits?), 3-bit addressing mode indicator.
- Add R1, R2
- Move 24(R0), R5
- LshiftR #2, R0
- Move #$3A, R1
- Branch>0 LOOP



(a) One-word instruction

Encoding of Machine Instructions

- What happens if we want to specify a memory operand using the Absolute addressing mode?
- Move R2, LOC
- 14-bit for LOC – insufficient
- Solution – use two words



(b) Two-word instruction

Encoding of Machine Instructions

- Then what if an instruction in which two operands can be specified using the Absolute addressing mode?
- Move LOC1, LOC2
- Solution – use two additional words
- This approach results in instructions of variable length. Complex instructions can be implemented, closely resembling operations in high-level programming languages – Complex Instruction Set Computer (CISC)

Encoding of Machine Instructions

- If we insist that all instructions must fit into a single 32-bit word, it is not possible to provide a 32-bit address or a 32-bit immediate operand within the instruction.
- It is still possible to define a highly functional instruction set, which makes extensive use of the processor registers.
- Add R1, R2 ----- yes
- Add LOC, R2 ----- no
- Add (R3), R2 ----- yes

UNIT-IV

Register Transfer

- Registers are designated by capital letters, sometimes followed by numbers (e.g., A, R13, IR)
- Often the names indicate function:

– MAR	- memory address register
– PC	- program counter
– IR	- instruction register

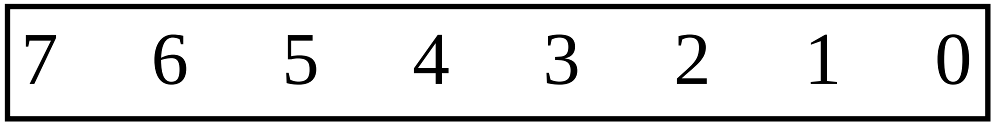
- Information Transfer from one register to another register is designated in symbolic form by using a replacement operator.

$$R2 \leftarrow R1$$

- Registers and their contents can be viewed and represented in *various ways:*
  - A register can be viewed as a single entity:

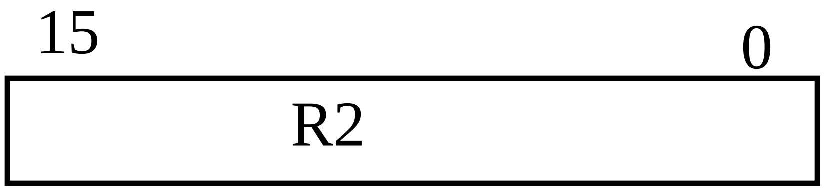
Register Transfer

- Registers may also be represented showing the bits of data they contain



Showing individual bits

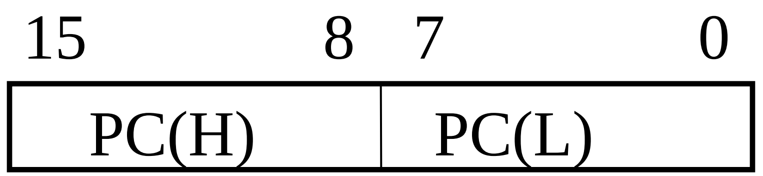
- Numbering of a registers



Numbering of bits

Register Transfer

- Portion of register



Subfields

Control Function

- The actions are performed only when certain conditions are true.
- This is similar to an “if” statement in a programming language.
- In digital systems, this is often done via a control signal, called a control function . control function is a Boolean variable.
  - If the signal is 1, the action takes place
- This is represented as:

  $P: R2 \leftarrow R1$

Register Transfer

- Which means "if P = 1, then load the contents of register R1 into register R2", i.e., if (P = 1) then (R2 ← R1).
- Implementation of controlled transfer

P: R2 ← R1

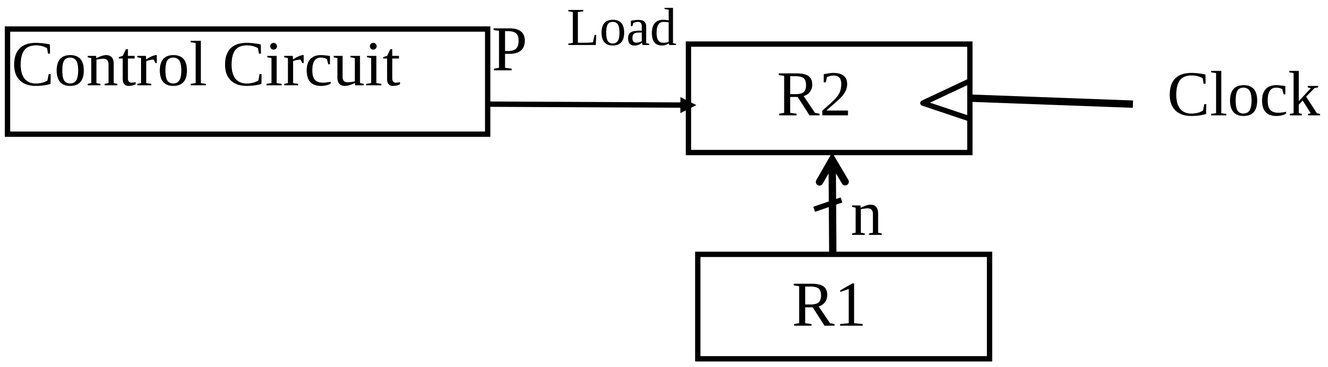


Fig: Block Diagram

Register Transfer

- The same clock controls the circuits that generate the control function and the destination register.

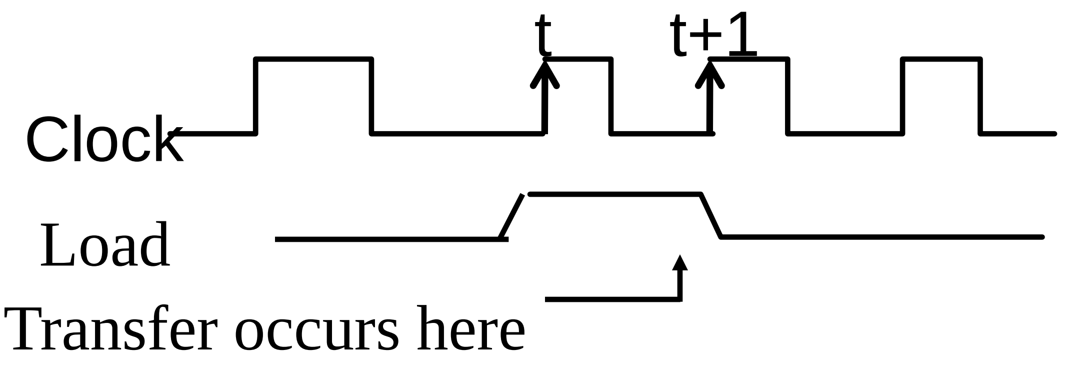


Fig :Timing Diagram

Register Transfer

- If two or more operations are to occur simultaneously, they are separated with commas

$P: R3 \leftarrow R5, MAR \leftarrow IR$

- Here, if the control function P = 1, load the contents of R5 into R3, and at the same time (clock), load the contents of register IR into register MAR
- Basic Symbols Used for Register Transfer is

- Letters and Numerals to denote a registers. Ex: MAR,IR,R2 .
- Parentheses ( ) to denote a part of a register . Ex: R2(0-7),R2(L).
- Arrow $\longrightarrow$ to denote transfer of Information. Ex: $R2 \leftarrow R1$
- Comma , Separates two micro operations . Ex: $R2 \leftarrow R1, R3 \leftarrow R4$

Overview

- Instruction Set Processor (ISP)
- Central Processing Unit (CPU)
- A typical computing task consists of a series of steps specified by a sequence of machine instructions that constitute a program.
- An instruction is executed by carrying out a sequence of more rudimentary operations.

Fundamental Concepts

- Processor fetches one instruction at a time and perform the operation specified.
- Instructions are fetched from successive memory locations until a branch or a jump instruction is encountered.
- Processor keeps track of the address of the memory location containing the next instruction to be fetched using Program Counter (PC).
- Instruction Register (IR)

Executing an Instruction

- Fetch the contents of the memory location pointed to by the PC. The contents of this location are loaded into the IR (fetch phase).

$$IR \leftarrow [[PC]]$$

- Assuming that the memory is byte addressable, increment the contents of the PC by 4 (fetch phase).

$$PC \leftarrow [PC] + 4$$

- Carry out the actions specified by the instruction in the IR (execution phase).

Processor Organization

MDR HAS TWO INPUTS AND TWO OUTPUTS

Datapath

Textbook Page 413

Executing an Instruction

- Transfer a word of data from one processor register to another or to the ALU.
- Perform an arithmetic or a logic operation and store the result in a processor register.
- Fetch the contents of a given memory location and load them into a processor register.
- Store a word of data from a processor register into a given memory location.

Register Transfers

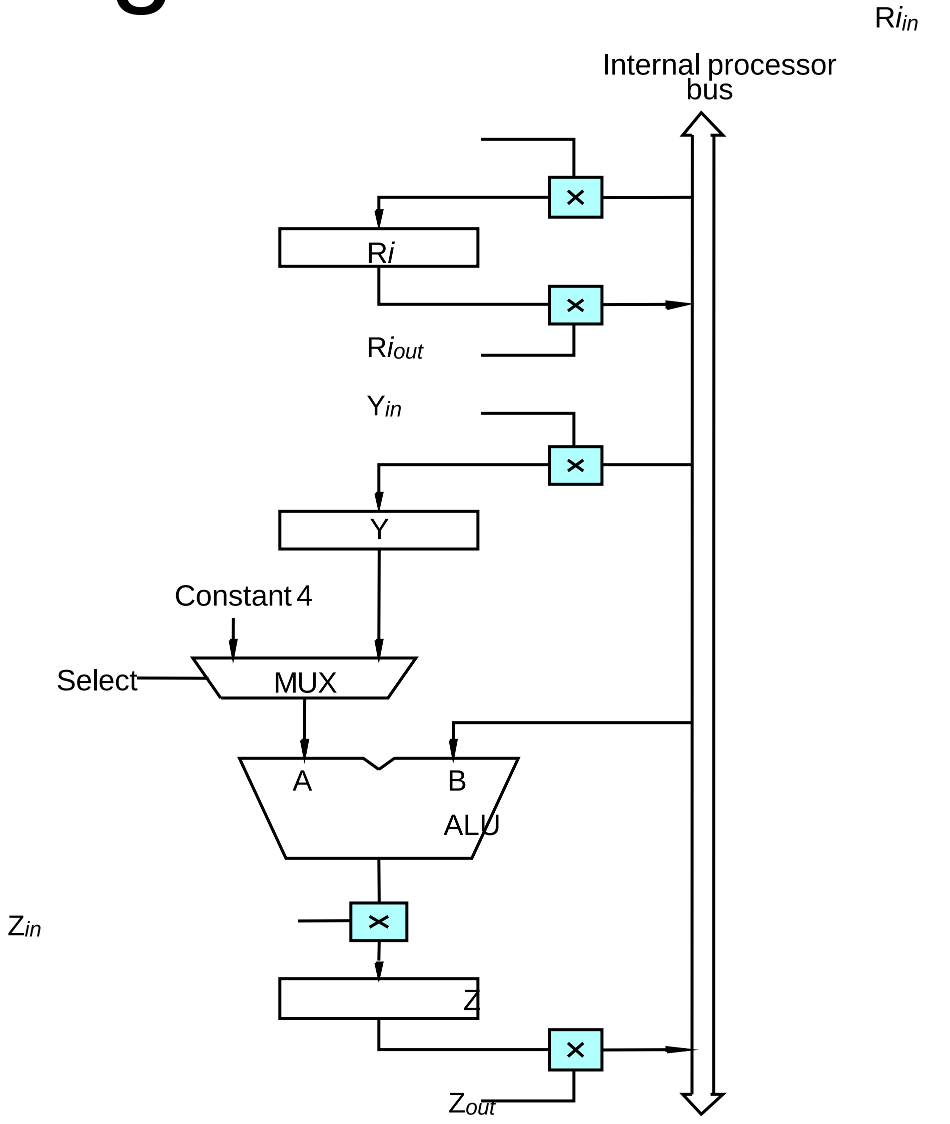


Figure 7.2. Input and output gating for the registers in Figure 7.1.

Performing an Arithmetic or Logic Operation

- The ALU is a combinational circuit that has no internal storage.
- ALU gets the two operands from MUX and bus. The result is temporarily stored in register Z.
- What is the sequence of operations to add the contents of register R1 to those of R2 and store the result in R3?
  1. R1out, Yin
  2. R2out, SelectY, Add, Zin
  3. Zout, R3in

Fetching a Word from Memory

- The response time of each memory access varies (cache miss, memory-mapped I/O,…).
- To accommodate this, the processor waits until it receives an indication that the requested operation has been completed (Memory-Function-Completed, MFC).
- Move (R1), R2
  - MAR $\leftarrow$ [R1]
  - Start a Read operation on the memory bus
  - Wait for the MFC response from the memory
  - Load MDR from the memory bus
  - R2 $\leftarrow$ [MDR]

Timing

MAR ← [R1]

Assume MAR
is always available
on the address lines
of the memory bus.

Start a Read operation on the memory bus

Wait for the MFC response from the memory

Load MDR from the memory bus

R2 ← [MDR]

Execution of a Complete Instruction

- Add (R3), R1
- Fetch the instruction
- Fetch the first operand (the contents of the memory location pointed to by R3)
- Perform the addition
- Load the result into R1

Architecture

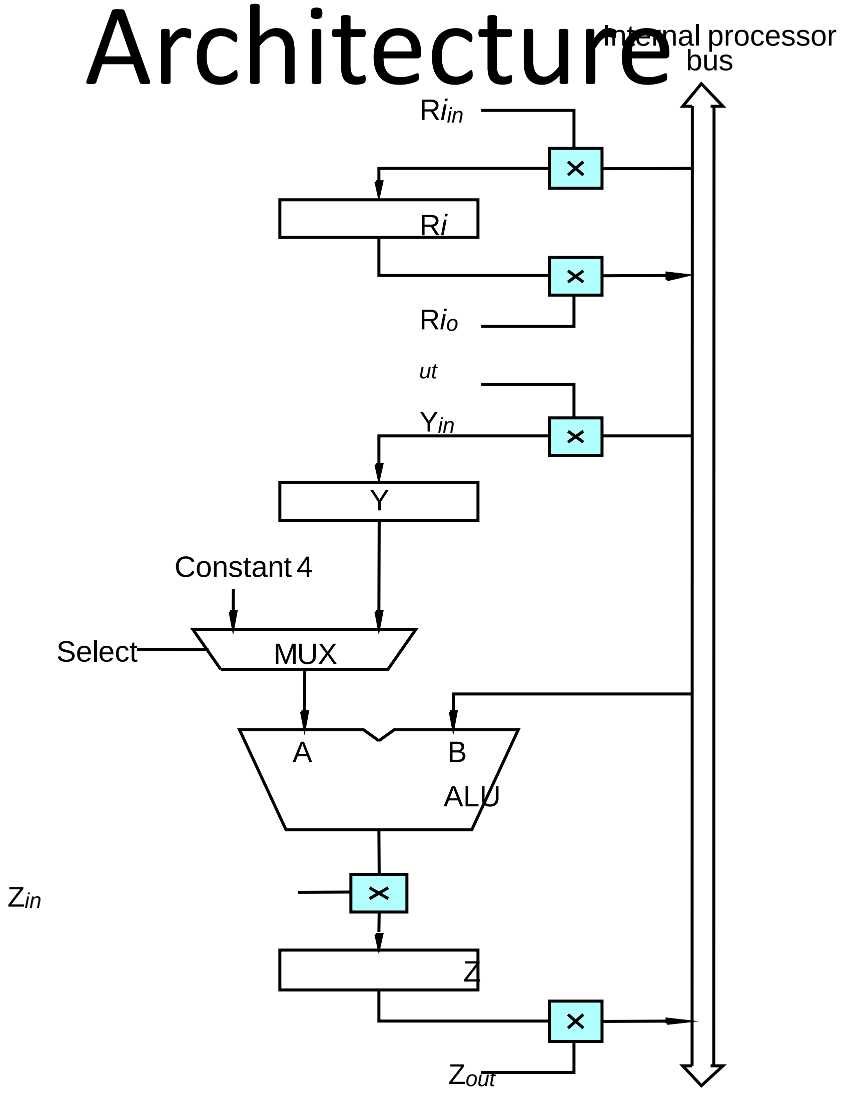


Figure 7.2. Input and output gating for the registers in Figure 7.1.

Execution of Branch Instructions

- A branch instruction replaces the contents of PC with the branch target address, which is usually obtained by adding an offset X given in the branch instruction.
- The offset X is usually the difference between the branch target address and the address immediately following the branch instruction.
- Conditional branch

Execution of Branch Instructions

Step	Action
1	PC_{out} , MAR_{in} , Read, Select4,Add, Z_{in}
2	Z_{out}, PC_{in} , Y_{in}, WMF C MDR_{out} , IR_{in}
3	Offset-field-of-IR_{out}, Add, Z_{in}
4	Z_{out}, PC_{in} , End
5	

Figure 7.7. Control sequence for an unconditional branch instruction.

Multiple-Bus Organization

- Add R4, R5, R6

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

Figure 7.9. Control sequence for the instruction. Add R4,R5,R6, for the three-bus organization in Figure 7.8.

Hardwired Control

Overview

- To execute instructions, the processor must have some means of generating the control signals needed in the proper sequence.
- Two categories: hardwired control and microprogrammed control
- Hardwired system can operate at high speed; but with little flexibility.

Control Unit Organization

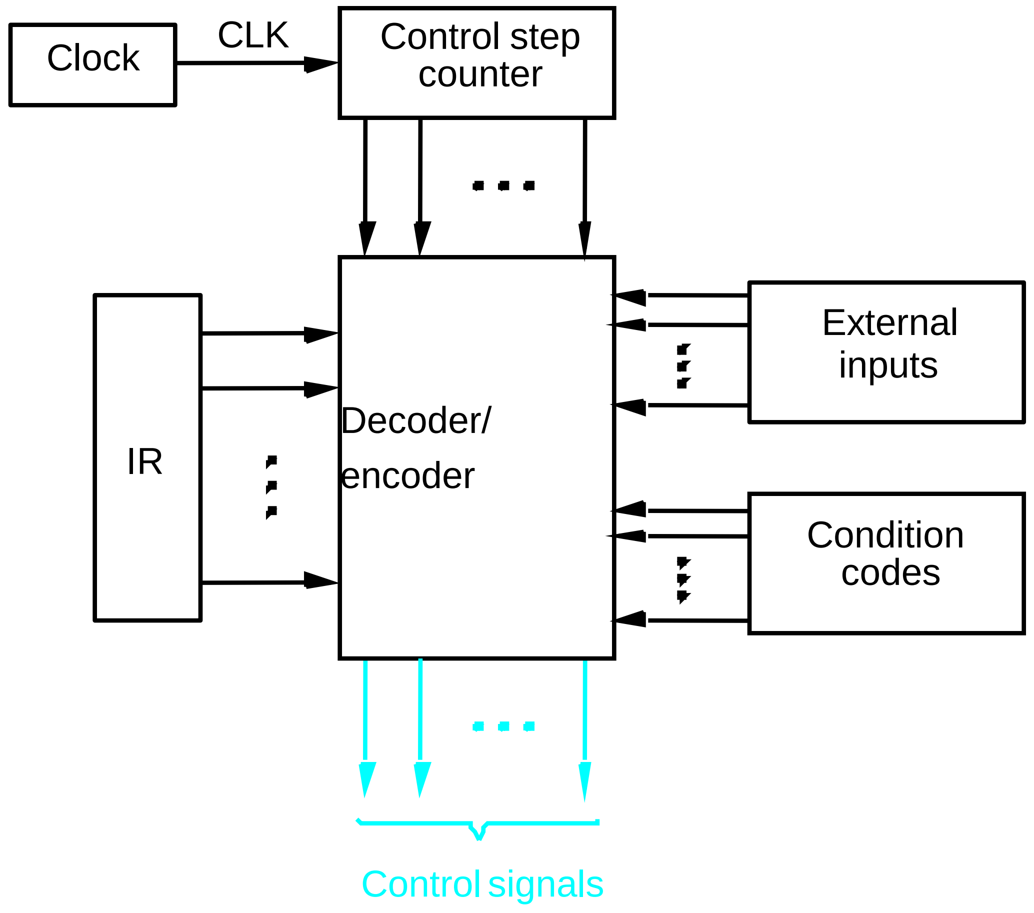


Figure 7.10. Control unit organization.

Generating Z_{in}

- $Z_{in} = T_1 + T_6 \bullet ADD + T_4 \bullet BR + \ldots$

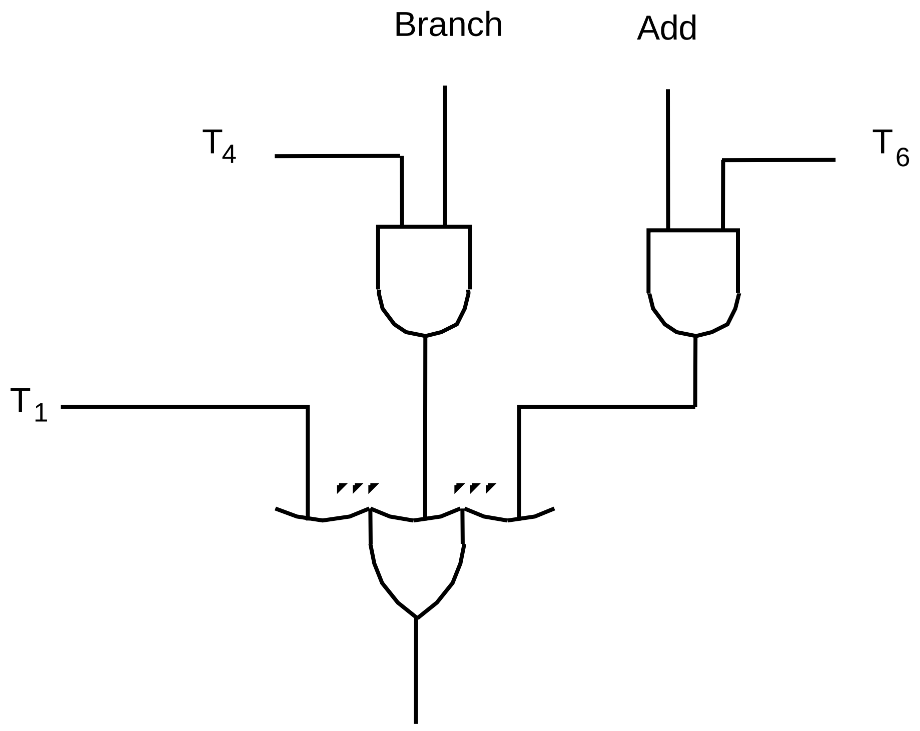


Figure 7.12. Generation of the Z_{in} control signal for the processor in Figure 7.1.

Generating End

- End = $T_7 \cdot ADD + T_5 \cdot BR + (T_5 \cdot N + T_4 \cdot \overline{N}) \cdot BRN + \ldots$

Microprogrammed Control

Overview

- Control signals are generated by a program similar to machine language programs.
- Control Word (CW); microroutine; microinstruction

Overview

- The previous organization cannot handle the situation when the control unit is required to check the status of the condition codes or external inputs to choose between alternative courses of action.
- Use conditional branch microinstruction.

Address	**Microinstruction**
0	PC_{out} , MAR_{in} , Read, Select4,Add, Z_{in} Z_{out} , PC_{in} ,
1	Y_{in} , WMFC
2	MDR_{out} , IR_{in}
3	Branch to startingaddressof appropriatemicroroutine
. . . .	. . . .
25	If N=0, then branch to microinstruction0
26	Offset-field-of-IR_{out} , SelectY, Add, Z_{in}
27	Z_{out} , PC_{in} , End

Figure 7.17. Microroutine for the instruction Branch<0.

Overview

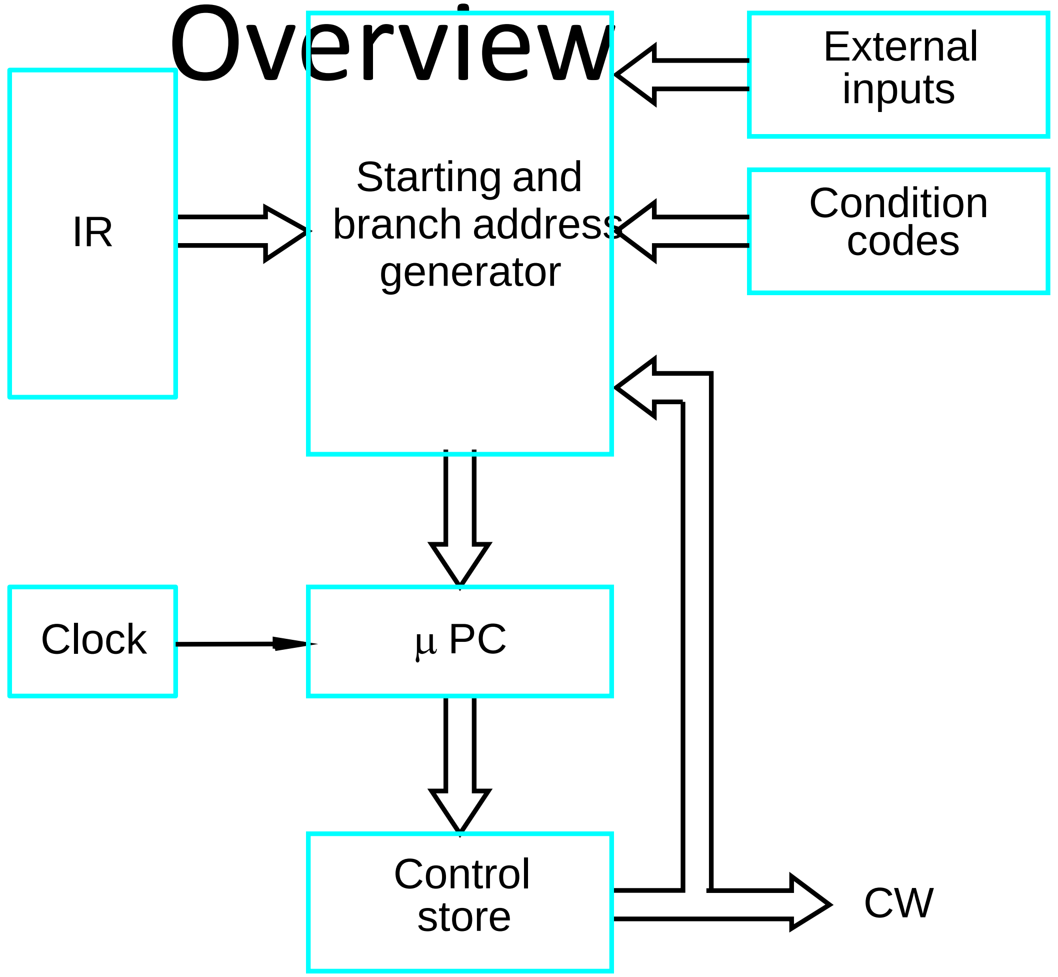


Figure 7.18. Organization of the control unit to allow conditional branching in the microprogram.

Microinstructions

- A straightforward way to structure microinstructions is to assign one bit position to each control signal.
- However, this is very inefficient.
- The length can be reduced: most signals are not needed simultaneously, and many signals are mutually exclusive.
- All mutually exclusive signals are placed in the same group in binary coding.

Further Improvement

- Enumerate the patterns of required signals in all possible microinstructions. Each meaningful combination of active control signals can then be assigned a distinct code.
- Vertical organization
- Horizontal organization

Microprogram Sequencing

- If all microprograms require only straightforward sequential execution of microinstructions except for branches, letting a μPC governs the sequencing would be efficient.
- However, two disadvantages:
  - Having a separate microroutine for each machine instruction results in a large total number of microinstructions and a large control store.
  - Longer execution time because it takes more time to carry out the required branches.
- Example: Add src, Rdst
- Four addressing modes: register, autoincrement, autodecrement, and indexed (with indirect forms).

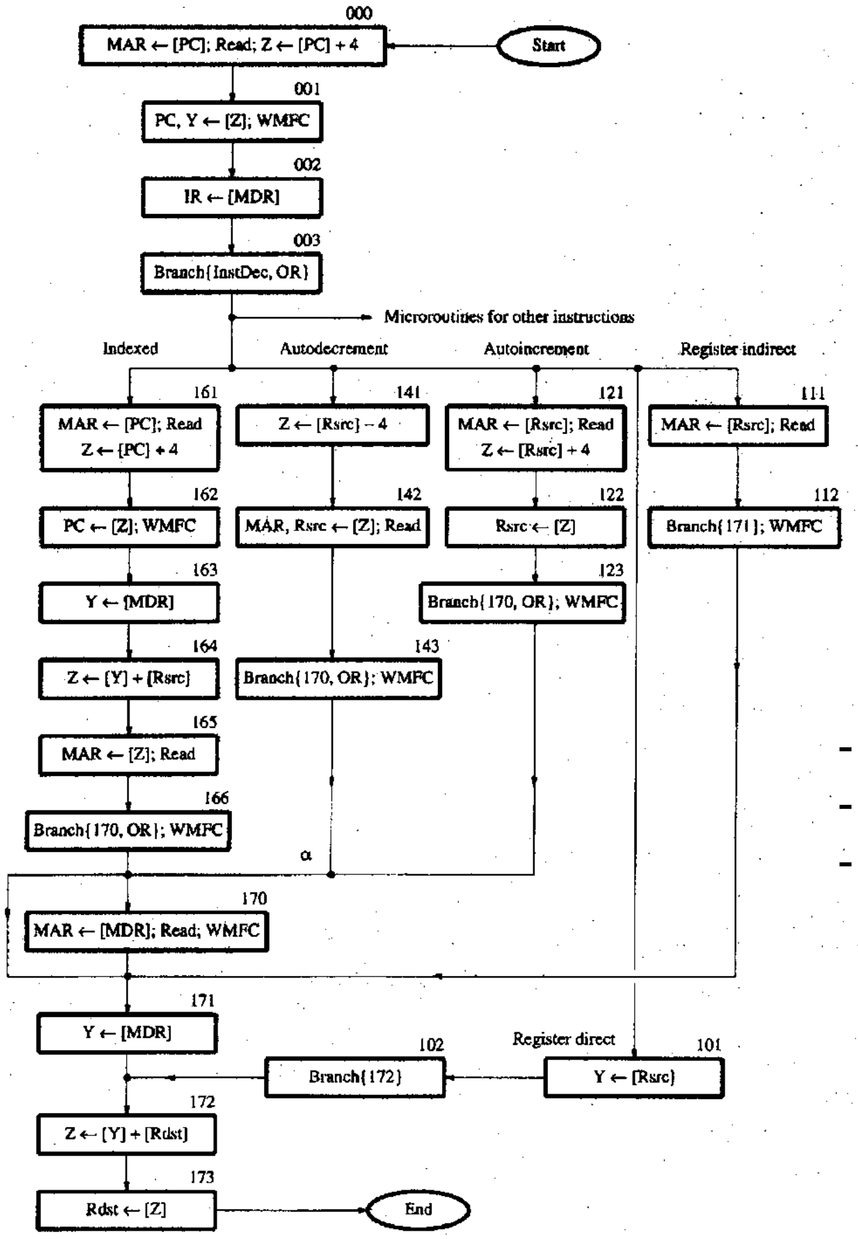
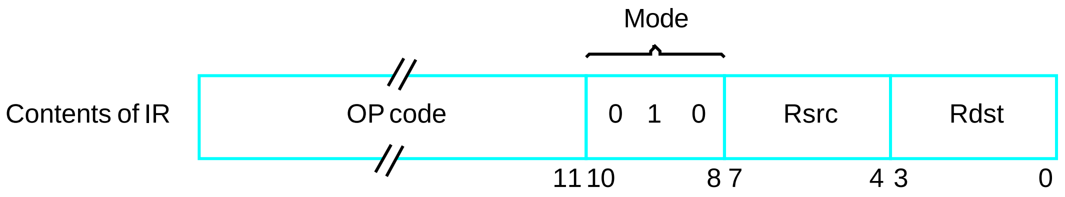


Figure 7.20. Flowchart of a microprogram for the Add src,Rdst instruction.

- Bit-ORing
- Wide-Branch Addressing
- WMFC



Address (octal)	Microinstruction
000	PC_{out}, MAR_{in}, Read, Selec4t, Add, Z_{in} Z_{out},
001	PC_{in}, Y_{in}, WMFC
002	MDR_{out}, IR_{in}
003	μBranch {μ PC← 101 (from Instruction decoder);
121	$\mu PC_{5,4} \leftarrow [IR_{10,9}]$; $\mu PC_3 \leftarrow [\overline{IR_{10}}] \cdot [\overline{IR_9}] \cdot [IR_8]\}$
122	$Rsrc_{out}$, MAR_{in} , Read, Select4, Add,$_{in}$Z Z_{out},
123	$Rsrc_{in}$
170	μBranch {μPC← 170;$\mu PC_0 \leftarrow [\overline{IR_8}]$}, WMFC
171	MDR_{out}, MAR_{in}, Read, WMFC MDR_{out}, Y_{in}
172	Rds$_o$t$_{ut}$, SelectY, Add, Z_{in} Z_{out}, Rds$_i$t$_n$, End
173	

Figure 7.21. Microinstruction for Add (Rsrc)+,Rdst.
*Note:*Microinstruction at location 170 is not executed for this addressing mode.

Microinstructions with Next-Address Field

- The microprogram we discussed requires several branch microinstructions, which perform no useful operation in the datapath.
- A powerful alternative approach is to include an address field as a part of every microinstruction to indicate the location of the next microinstruction to be fetched.
- Pros: separate branch microinstructions are virtually eliminated; few limitations in assigning addresses to microinstructions.
- Cons: additional bits for the address field (around 1/6)

bit-ORing

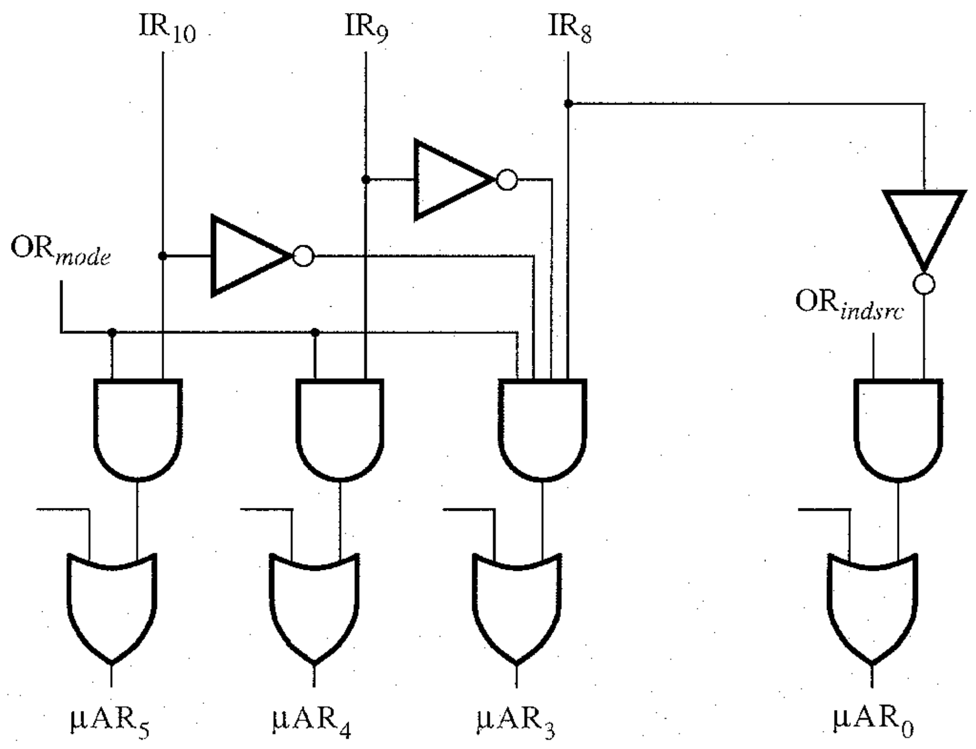


Figure 7.26. Control circuitry for bit-ORing

(part of the decoding circuits in Figure 7.25).

Chapter 5. The Memory System

Overview

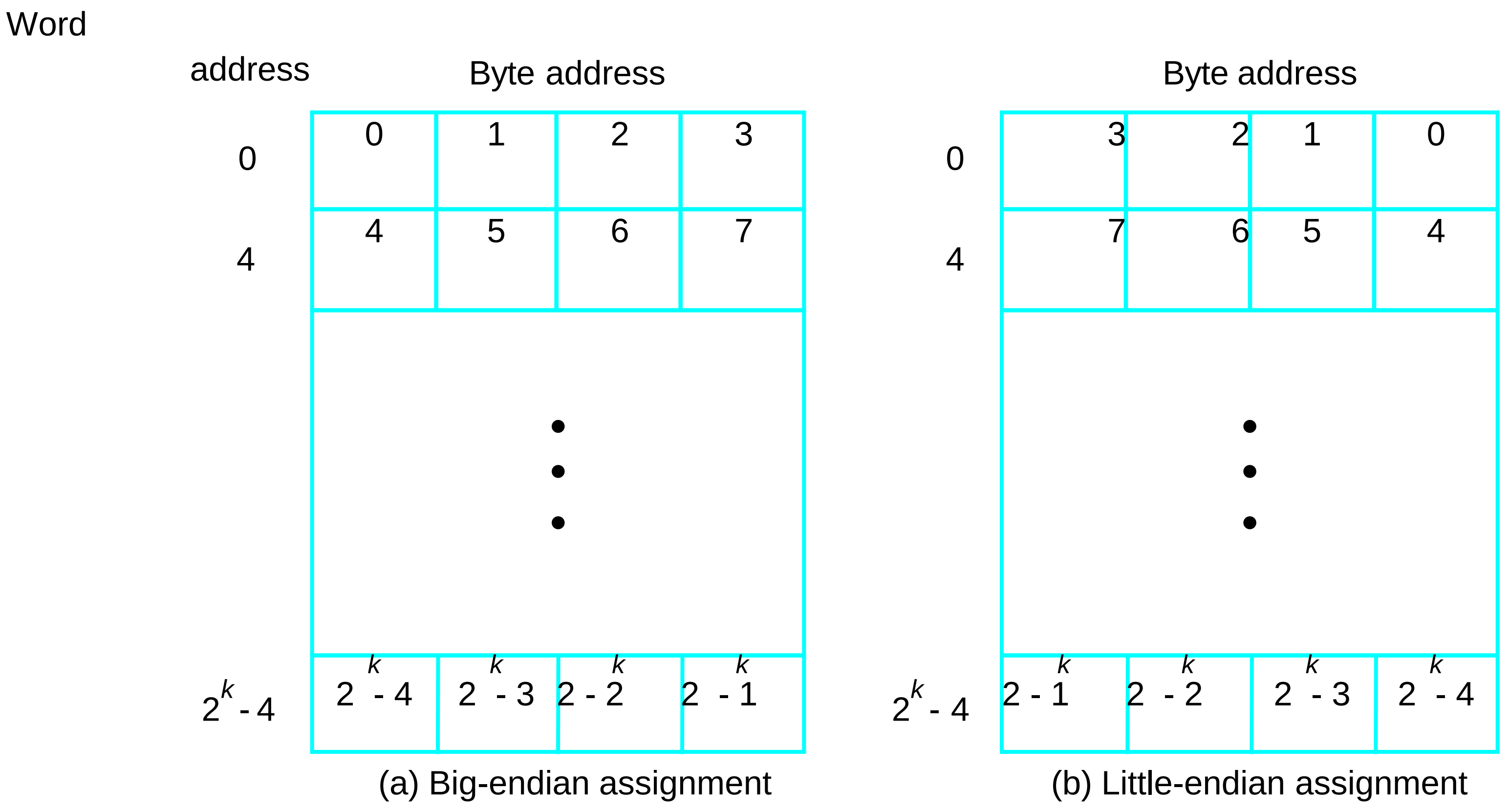
- Basic memory circuits
- Organization of the main memory
- Cache memory concept
- Virtual memory mechanism
- Secondary storage

Basic Concepts

- The maximum size of the memory that can be used in any computer is determined by the addressing scheme.

16-bit addresses = 2^{16} = 64K memory locations

- Most modern computers are byte addressable.



(a) Big-endian assignment

(b) Little-endian assignment

Traditional Architecture

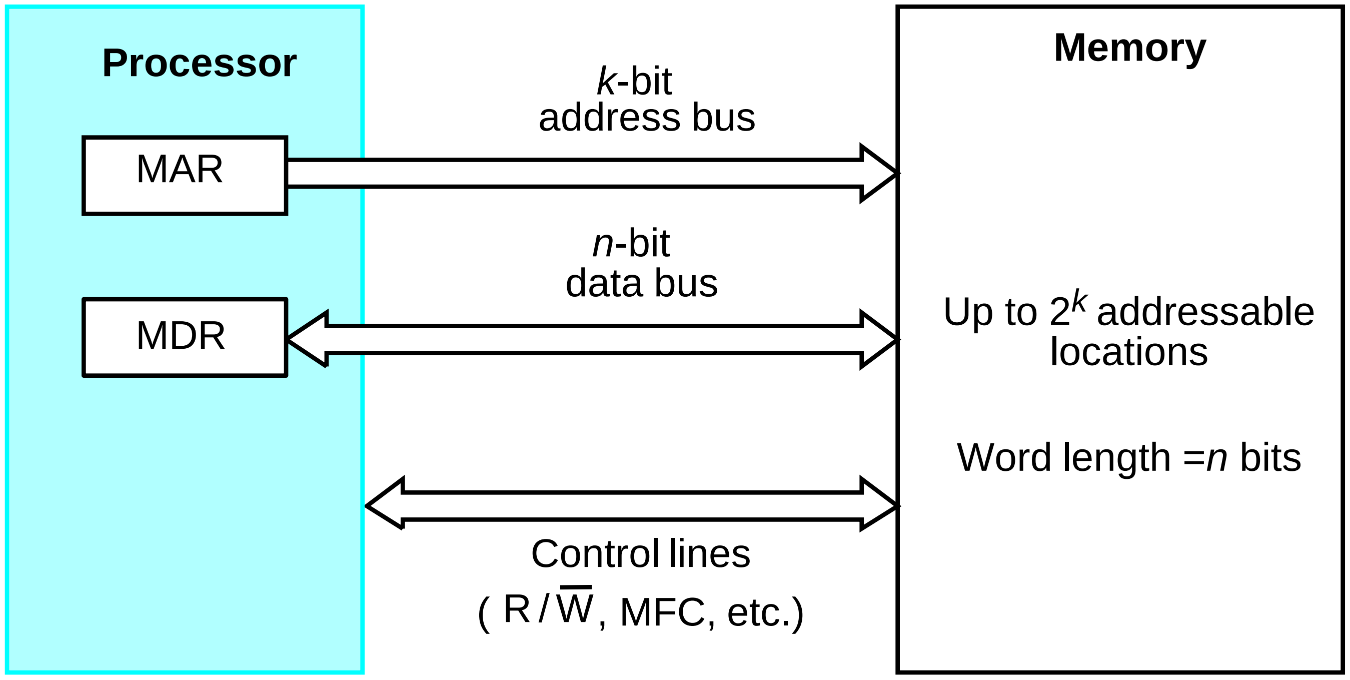


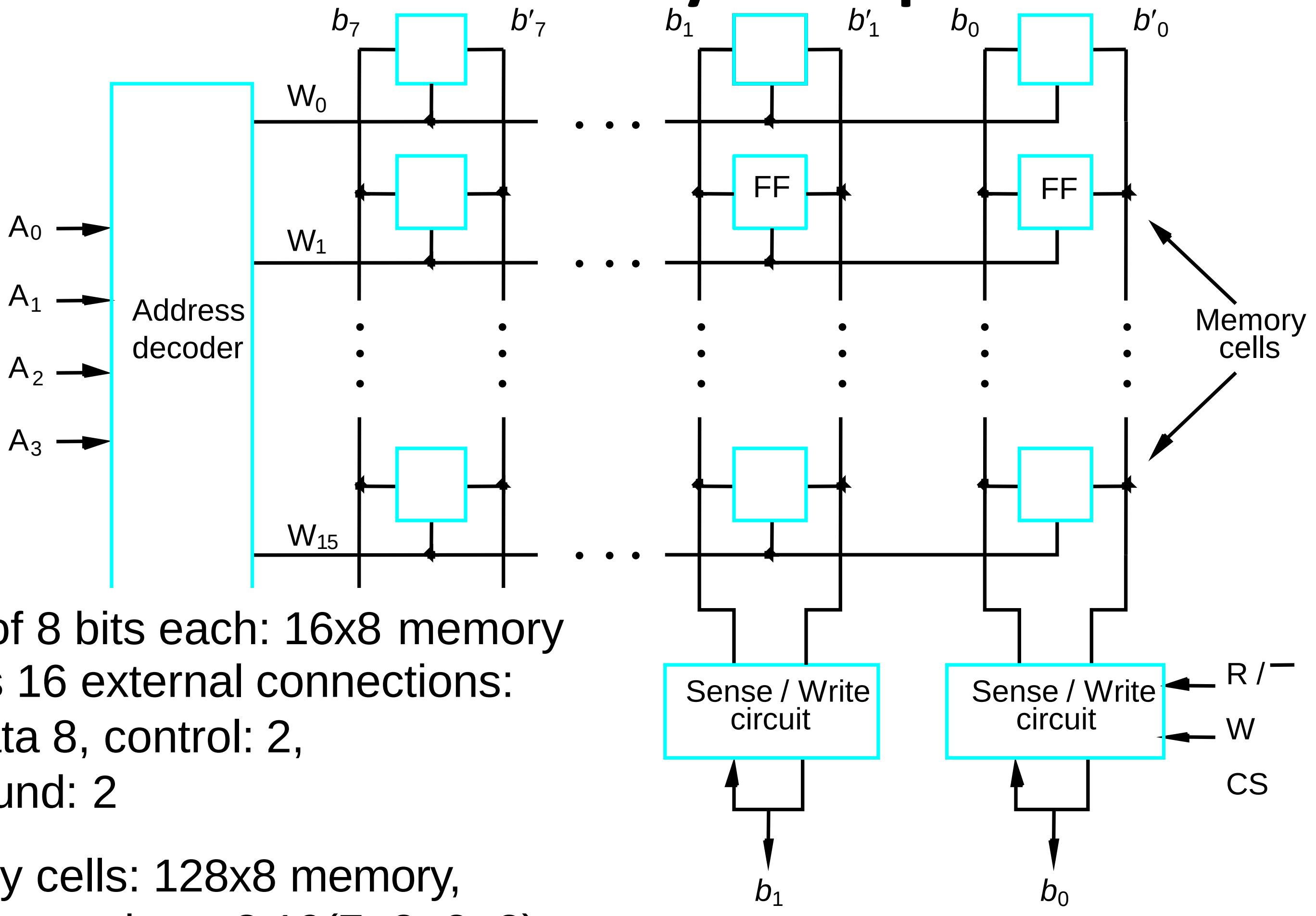
Figure 5.1. Connection of the memory to the processor.

Basic Concepts

- "Block transfer" – bulk data transfer
- *Memory access time*
- *Memory cycle time*
- RAM – any location can be accessed for a Read or Write operation in some fixed amount of time that is independent of the location's address.
- Cache memory
- Virtual memory, memory management unit

Semiconductor RAM Memories

Internal Organization of Memory Chips



16 words of 8 bits each: 16x8 memory org.. It has 16 external connections: addr. 4, data 8, control: 2, power/ground: 2

1K memory cells: 128x8 memory, external connections: ? 19(7+8+2+2)

1Kx1:? 15 (10+1+2+2)

^{i}zation of bit cells in a memory chip.

A Memory Chip

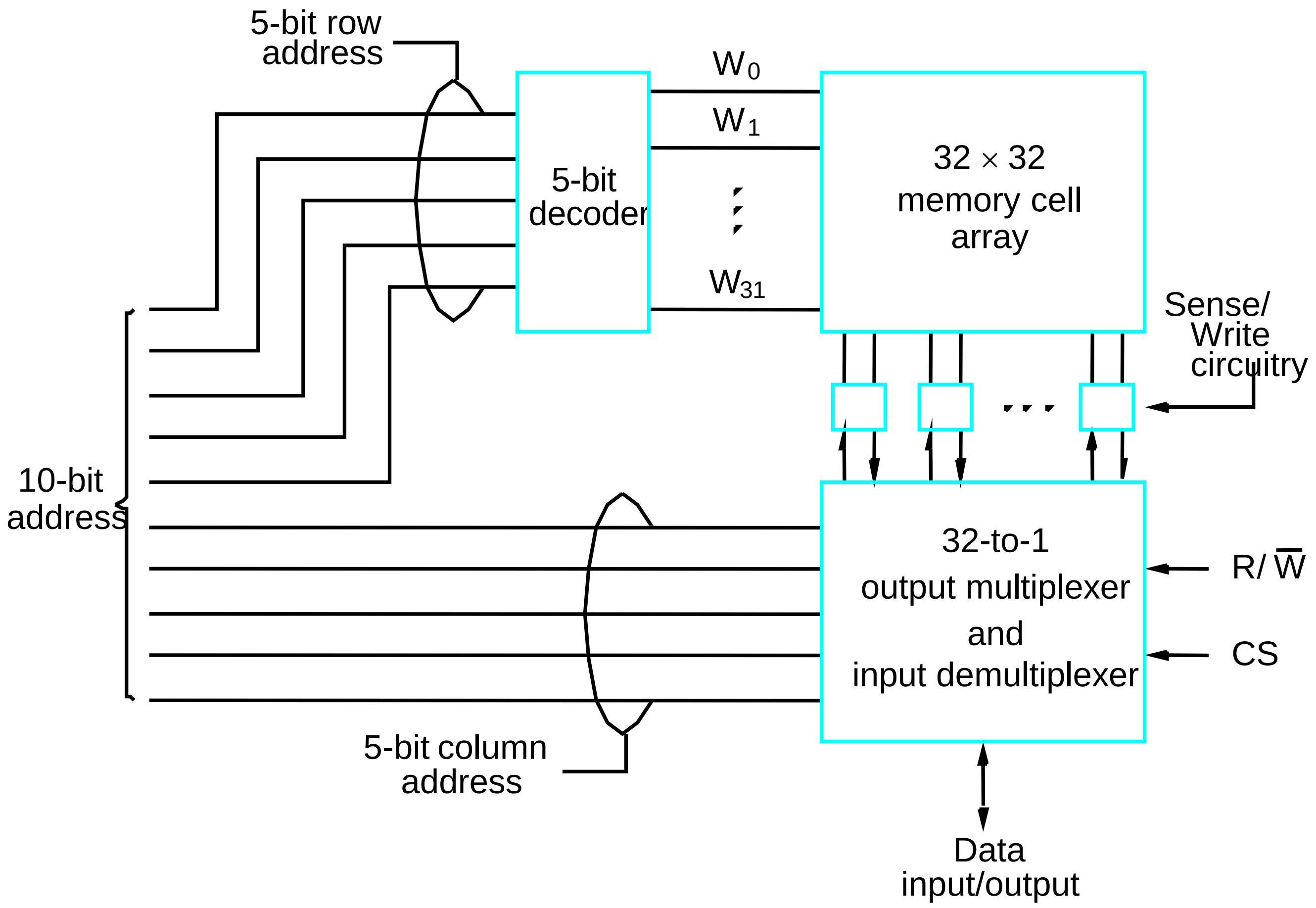


Figure 5.3. Organization of a 1K × 1 memory chip.

Static Memories

- The circuits are capable of retaining their state as long as power is applied.

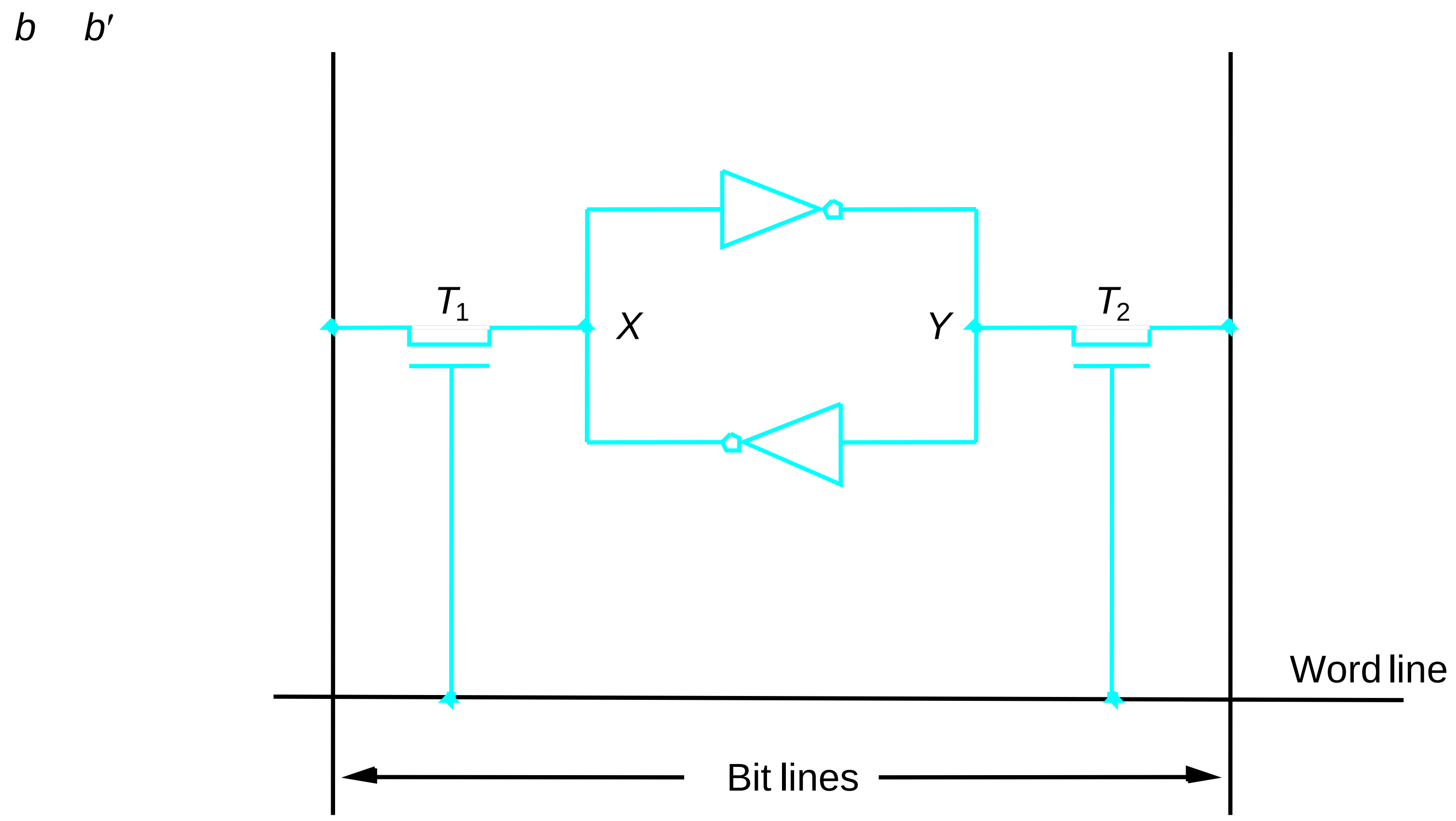


Figure 5.4. A static RAM cell.

Static Memories

- CMOS cell: low power consumption

Asynchronous DRAMs

- Static RAMs are fast, but they cost more area and are more expensive.
- Dynamic RAMs (DRAMs) are cheap and area efficient, but they can not retain their state indefinitely – need to be periodically refreshed.

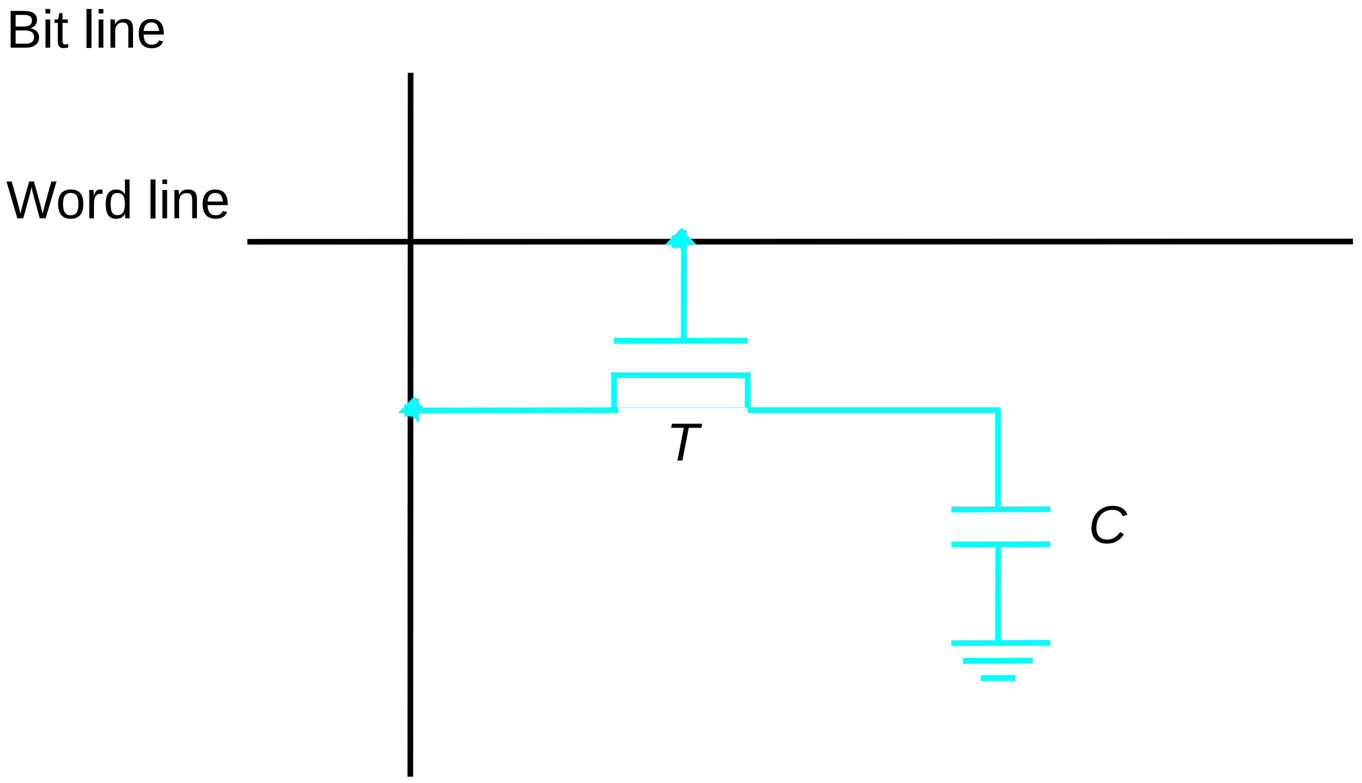


Figure 5.6. A single-transistor dynamic memory cell

A Dynamic Memory Chip

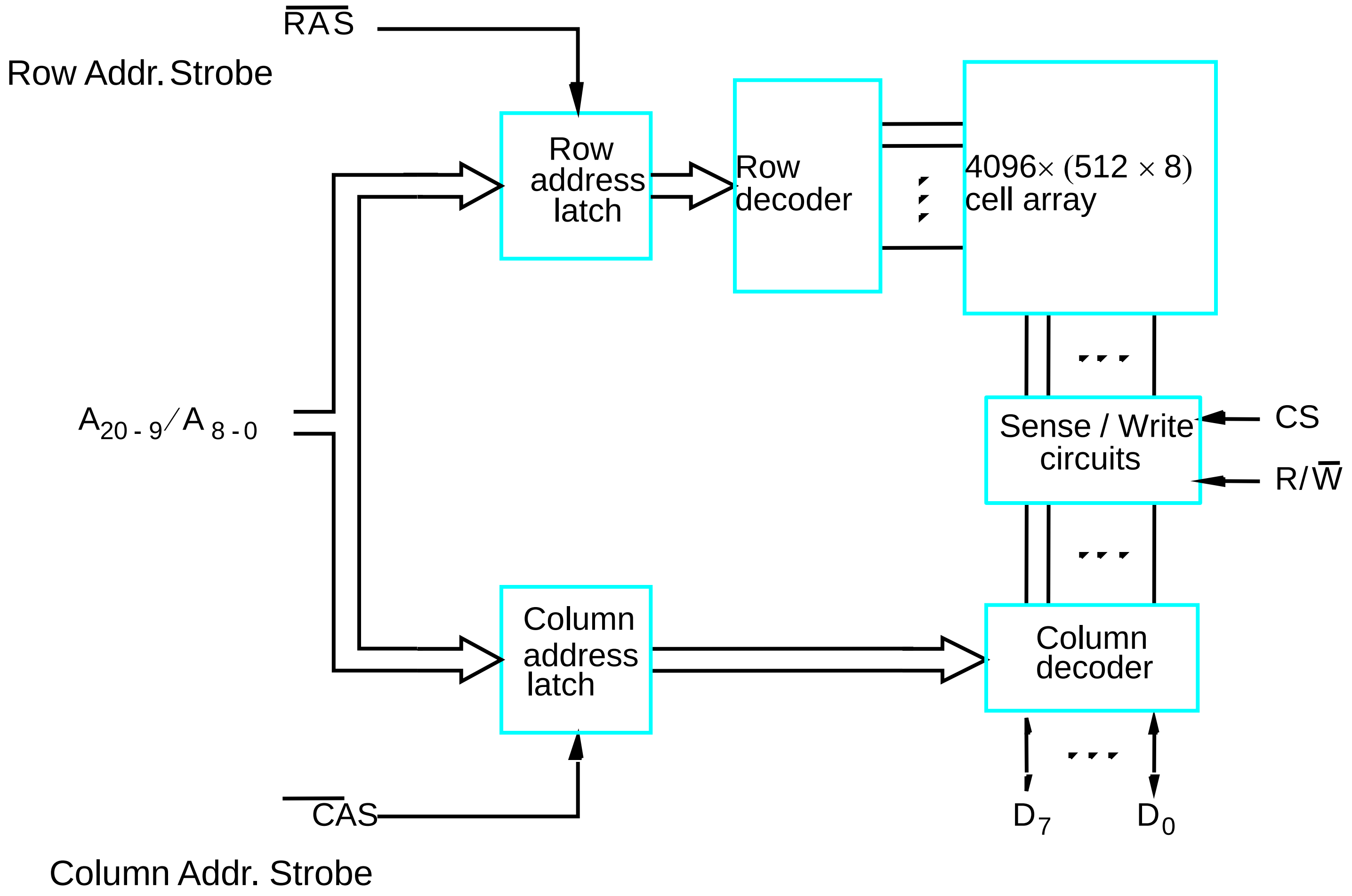


Figure 5.7. Internal organization of a 2M × 8 dynamic memory chip.

Fast Page Mode

- When the DRAM in last slide is accessed, the contents of all 4096 cells in the selected row are sensed, but only 8 bits are placed on the data lines $D_{7\text{-}0}$, as selected by $A_{8\text{-}0}$.
- Fast page mode – make it possible to access the other bytes in the same row without having to reselect the row.
- A latch is added at the output of the sense amplifier in each column.
- Good for bulk transfer.

Synchronous DRAMs

- The operations of SDRAM are controlled by a clock signal.

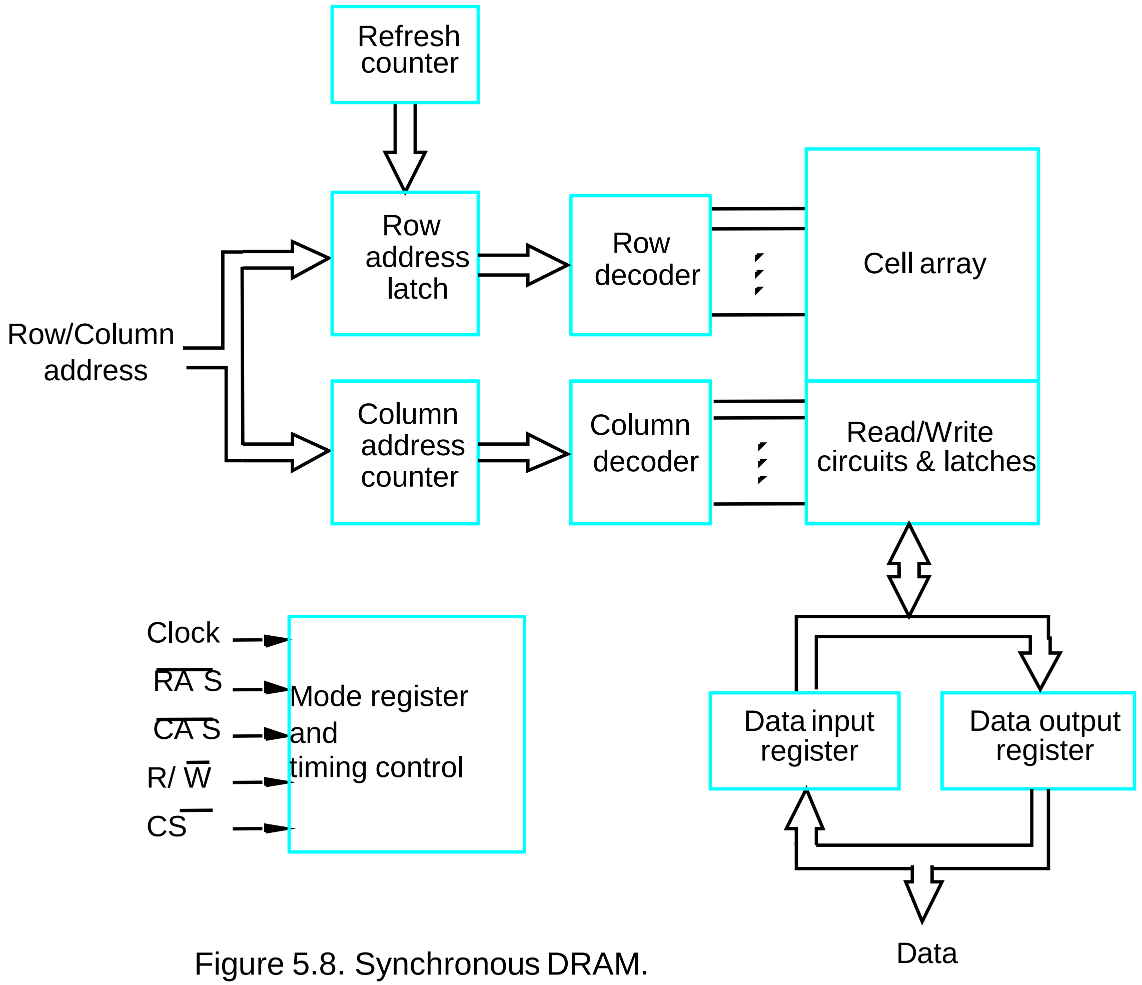


Figure 5.8. Synchronous DRAM.

Synchronous DRAMs

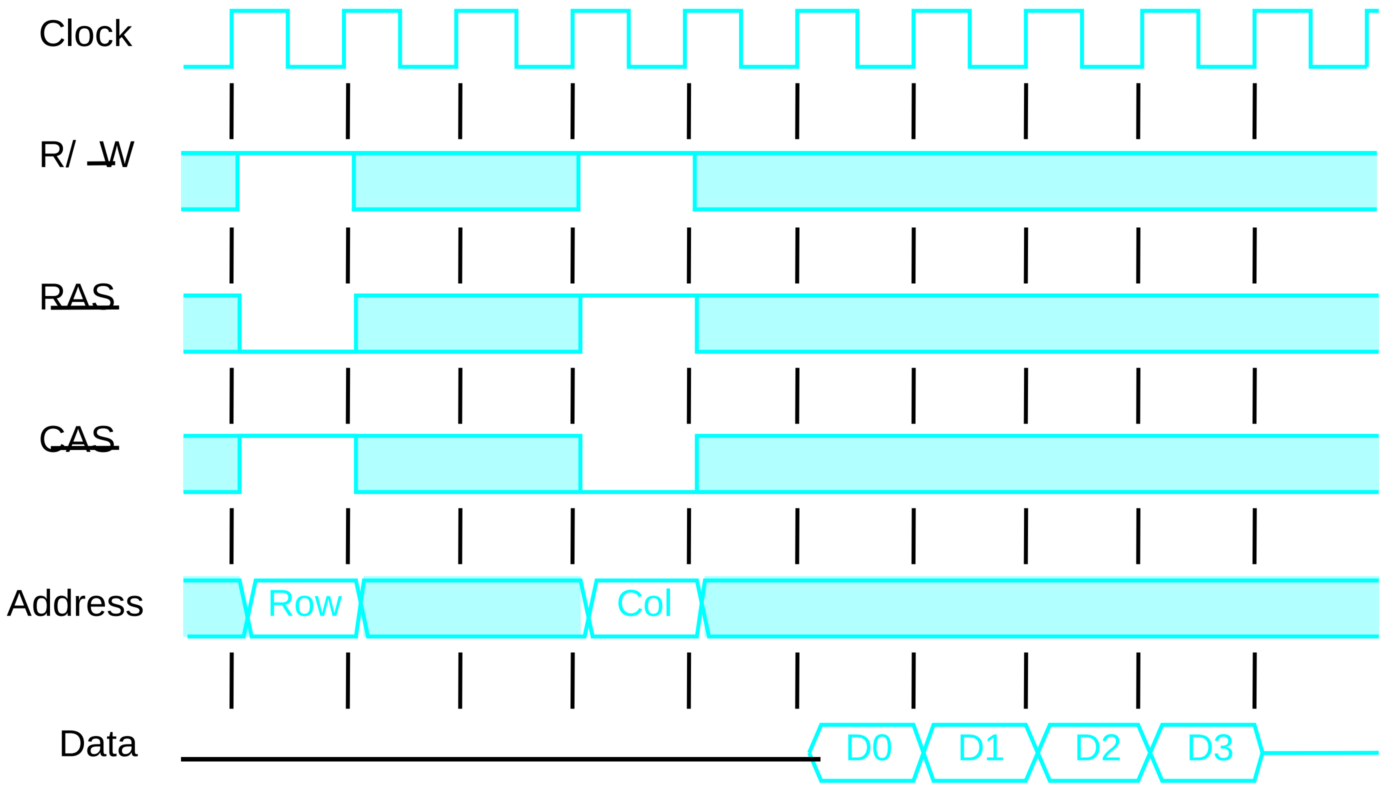


Figure 5.9. Burst read of length 4 in an SDRAM.

Synchronous DRAMs

- No CAS pulses is needed in burst operation.
- Refresh circuits are included (every 64ms).
- Clock frequency > 100 MHz
- Intel PC100 and PC133

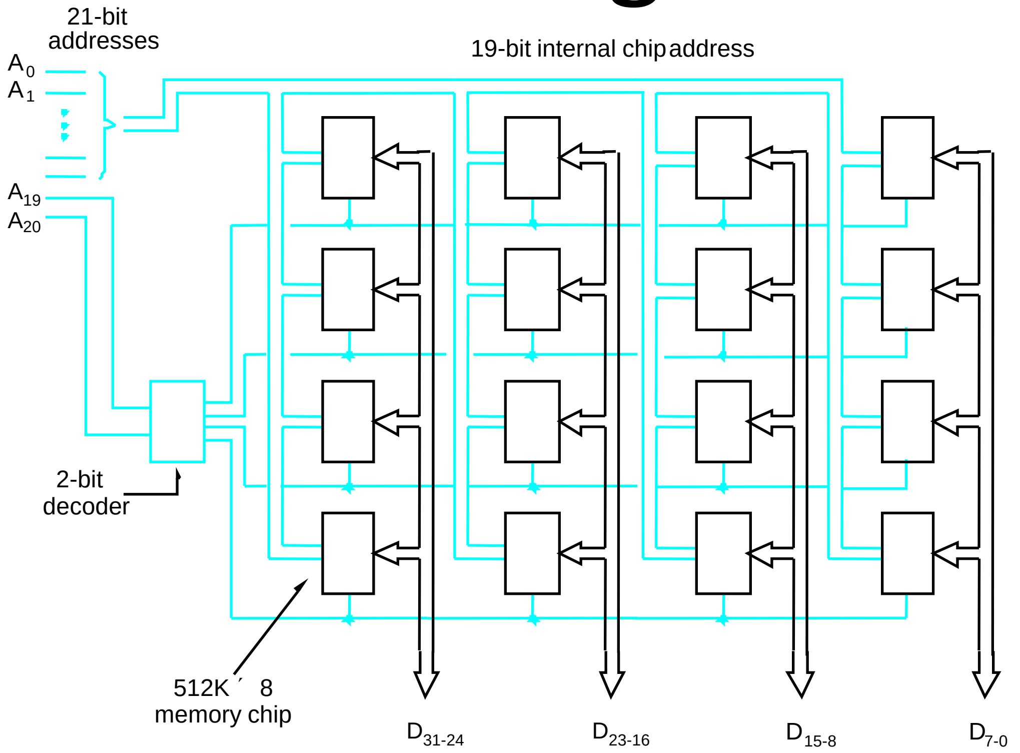
Latency and Bandwidth

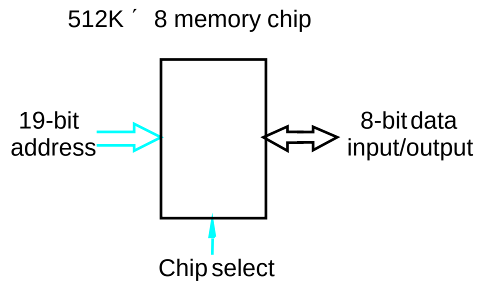
- The speed and efficiency of data transfers among memory, processor, and disk have a large impact on the performance of a computer system.
- Memory latency – the amount of time it takes to transfer a word of data to or from the memory.
- Memory bandwidth – the number of bits or bytes that can be transferred in one second. It is used to measure how much time is needed to transfer an entire block of data.
- Bandwidth is not determined solely by memory. It is the product of the rate at which data are transferred (and accessed) and the width of the data bus.

DDR SDRAM

- Double-Data-Rate SDRAM
- Standard SDRAM performs all actions on the rising edge of the clock signal.
- DDR SDRAM accesses the cell array in the same way, but transfers the data on both edges of the clock.
- The cell array is organized in two banks. Each can be accessed separately.
- DDR SDRAMs and standard SDRAMs are most efficiently used in applications where block transfers are prevalent.

Structures of Larger Memories





Memory System Considerations

- The choice of a RAM chip for a given application depends on several factors:

Cost, speed, power, size…

- SRAMs are faster, more expensive, smaller.
- DRAMs are slower, cheaper, larger.
- Which one for cache and main memory, respectively?
- Refresh overhead – suppose a SDRAM whose cells are in 8K rows; 4 clock cycles are needed to access each row; then it takes $8192 \times 4 = 32{,}768$ cycles to refresh all rows; if the clock rate is 133 MHz, then it takes $32{,}768/(133 \times 10^{-6}) = 246 \times 10^{-6}$ seconds; suppose the typical refreshing period is 64 ms, then the refresh overhead is $0.246/64 = 0.0038 < 0.4\%$ of the total time available for accessing the memory.

Memory Controller

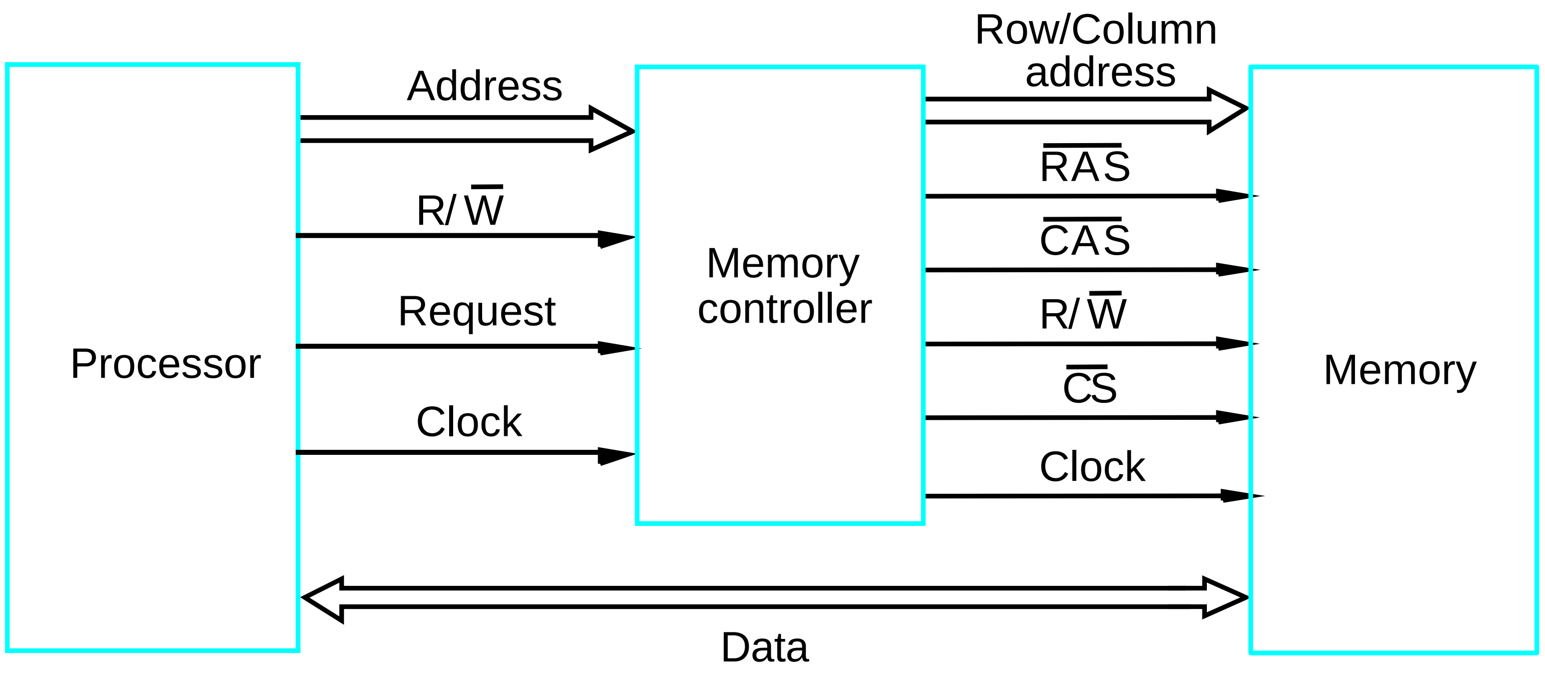


Figure 5.11. Use of a memory controller.

Read-Only Memories

Read-Only-Memory

- Volatile / non-volatile memory
- ROM
- PROM: programmable ROM
- EPROM: erasable, reprogrammable ROM
- EEPROM: can be programmed and erased electrically

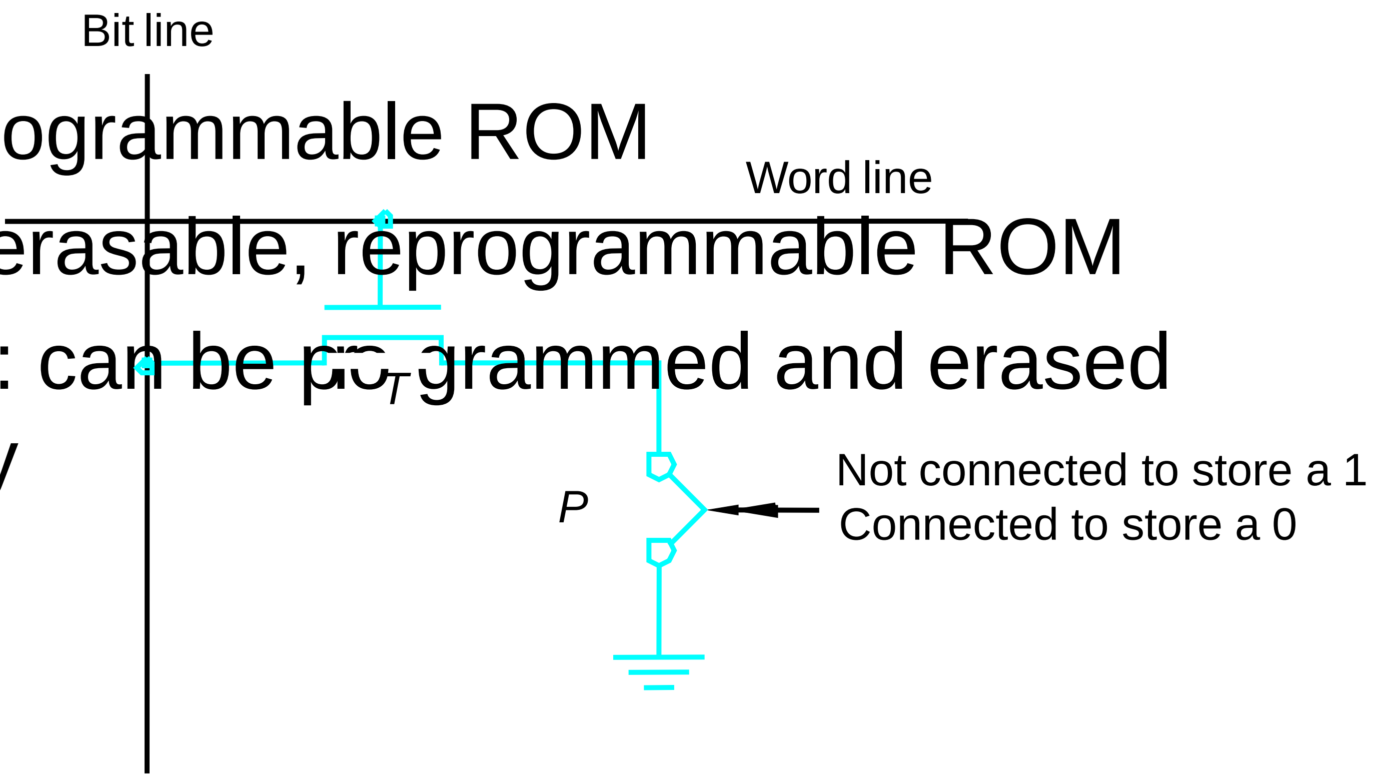


Figure 5.12. A ROM cell.

Flash Memory

- Similar to EEPROM
- Difference: only possible to write an entire block of cells instead of a single cell
- Low power
- Use in portable equipment
- Implementation of such modules
  - Flash cards
  - Flash drives

Speed, Size, and Cost

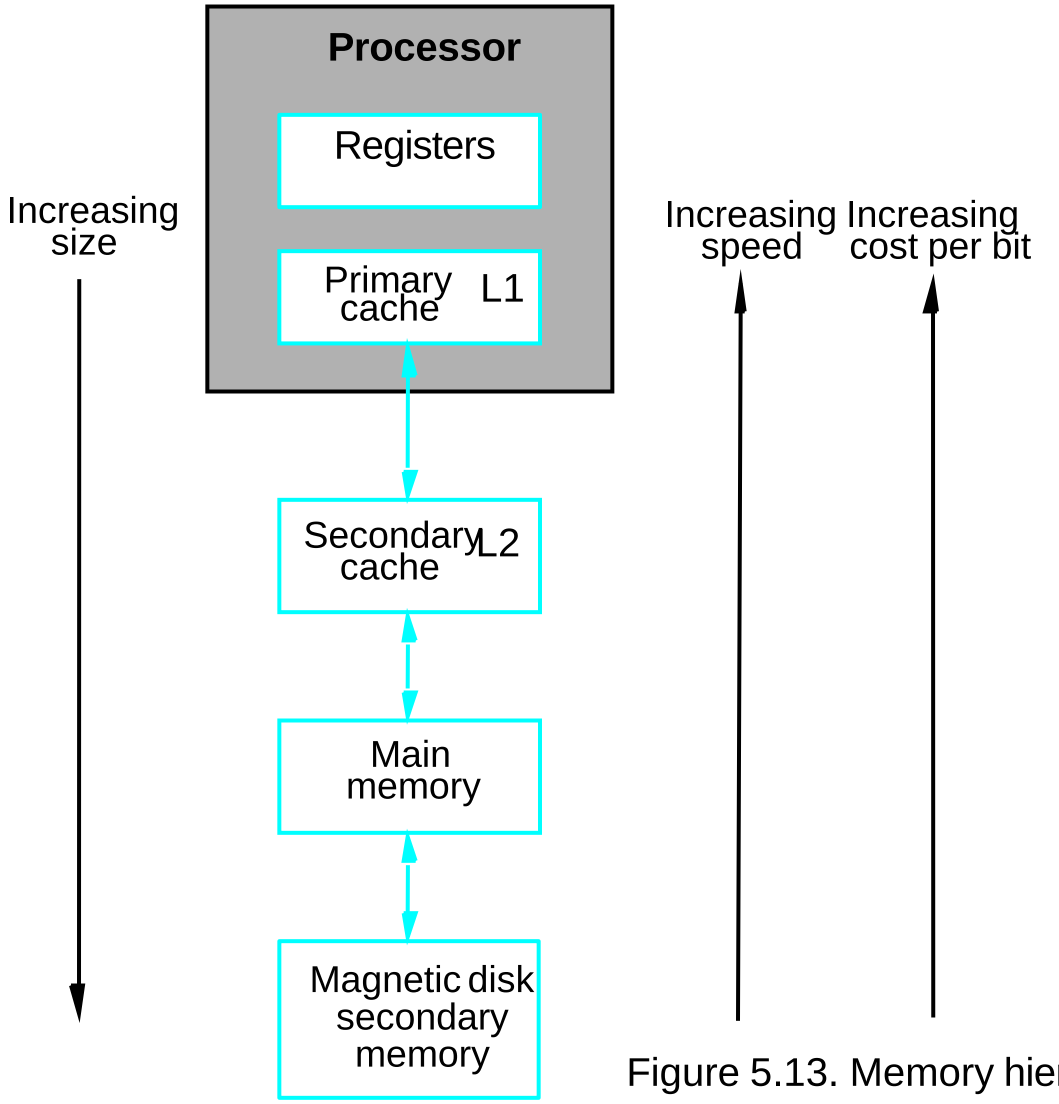


Figure 5.13. Memory hierarchy.

Cache Memories

Cache

- What is cache?

Page 315

- Why we need it?
- Locality of reference (very important)

  - temporal

  - spatial
- Cache block – *cache line*
  - *A set of contiguous address locations of some size*

Cache

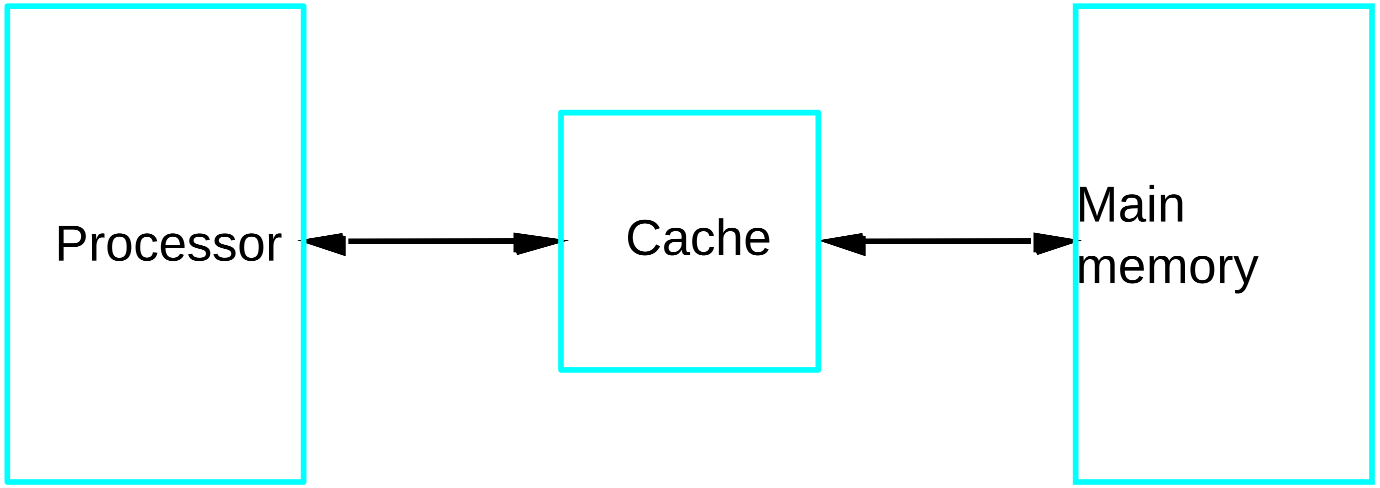
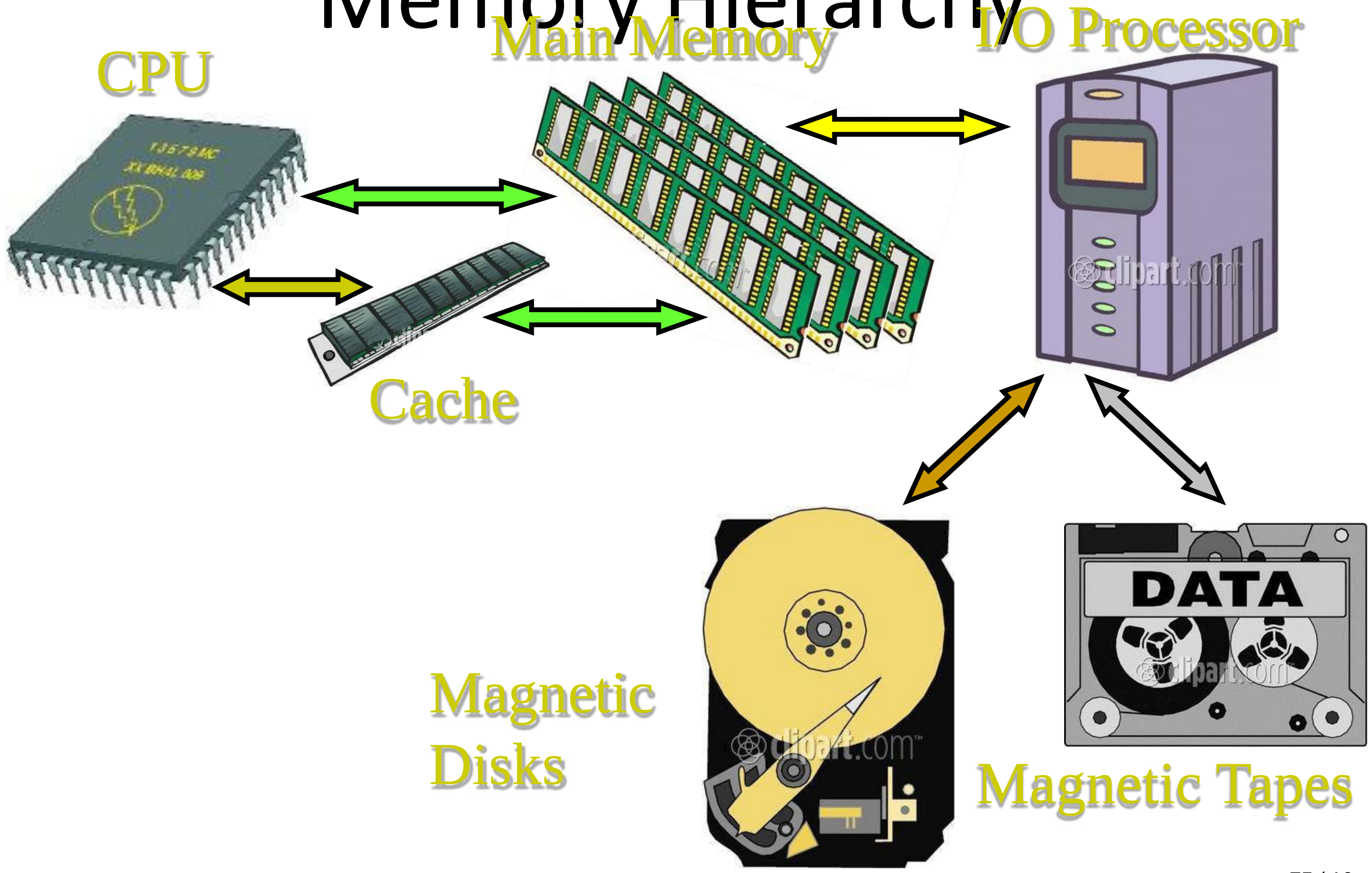


Figure 5.14. Use of a cache memory.

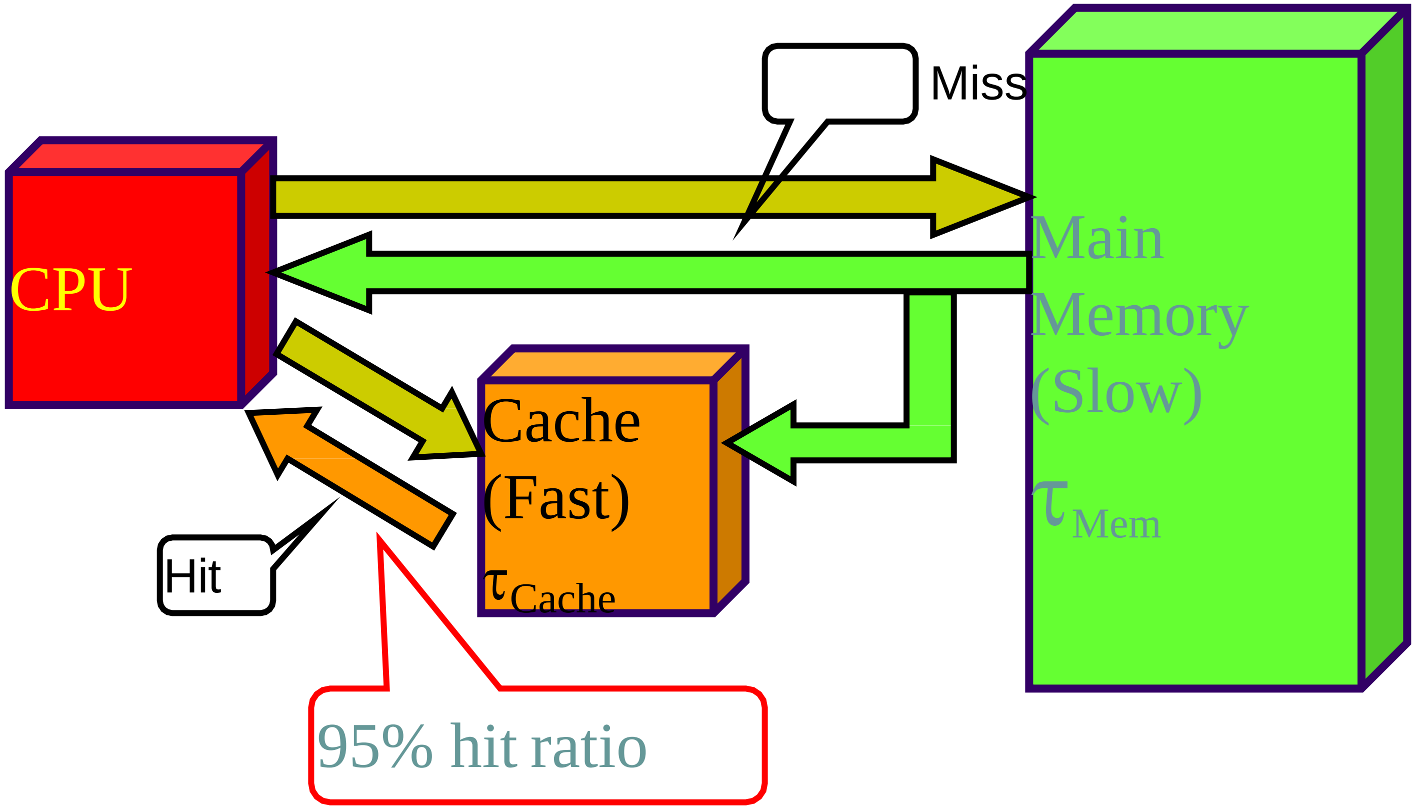
- Replacement algorithm
- Hit / miss
- Write-through / Write-back
- Load through

Memory Hierarchy



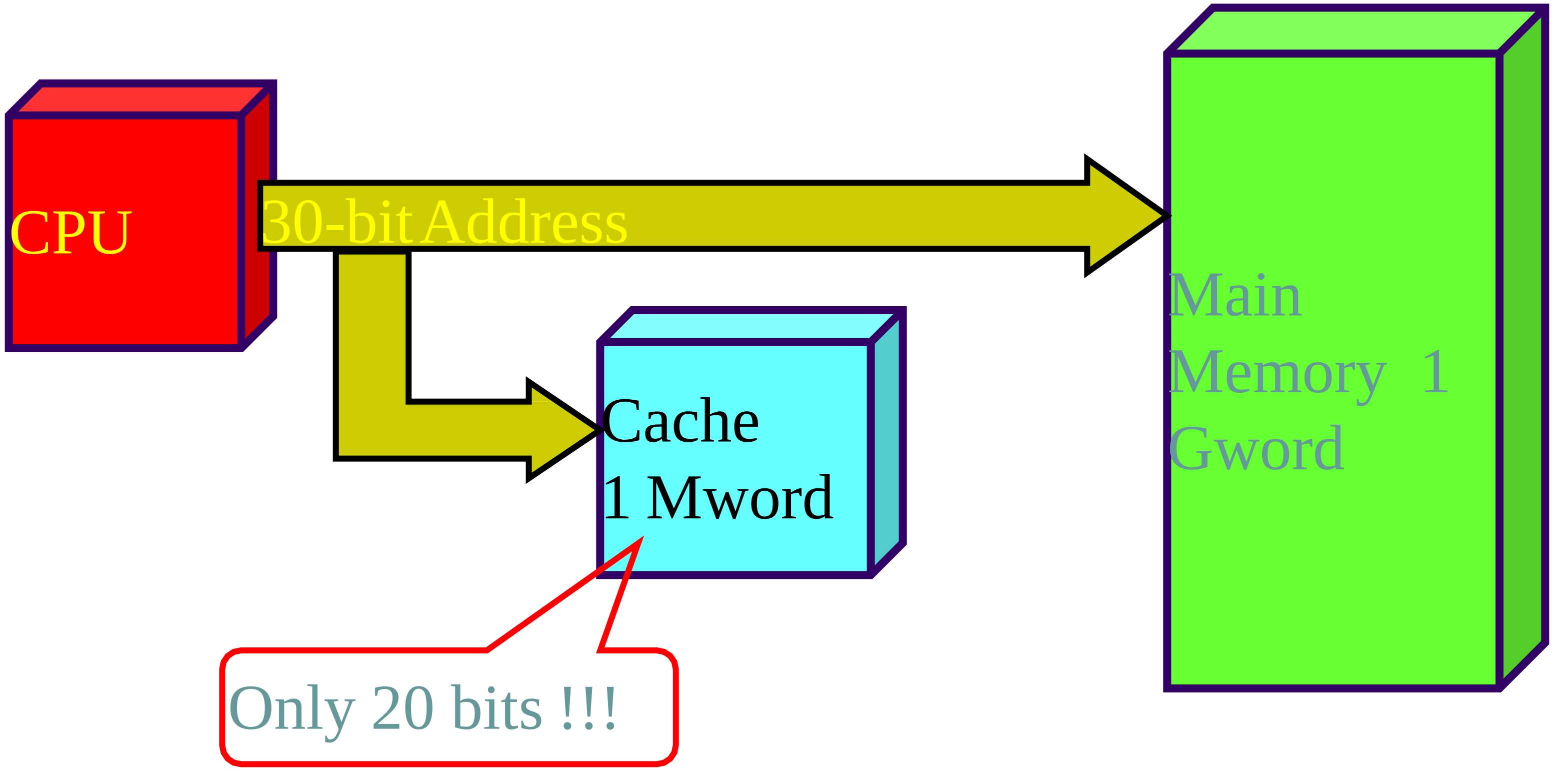
Cache Memory

- High speed (towards CPU speed)
- Small size (power & cost)

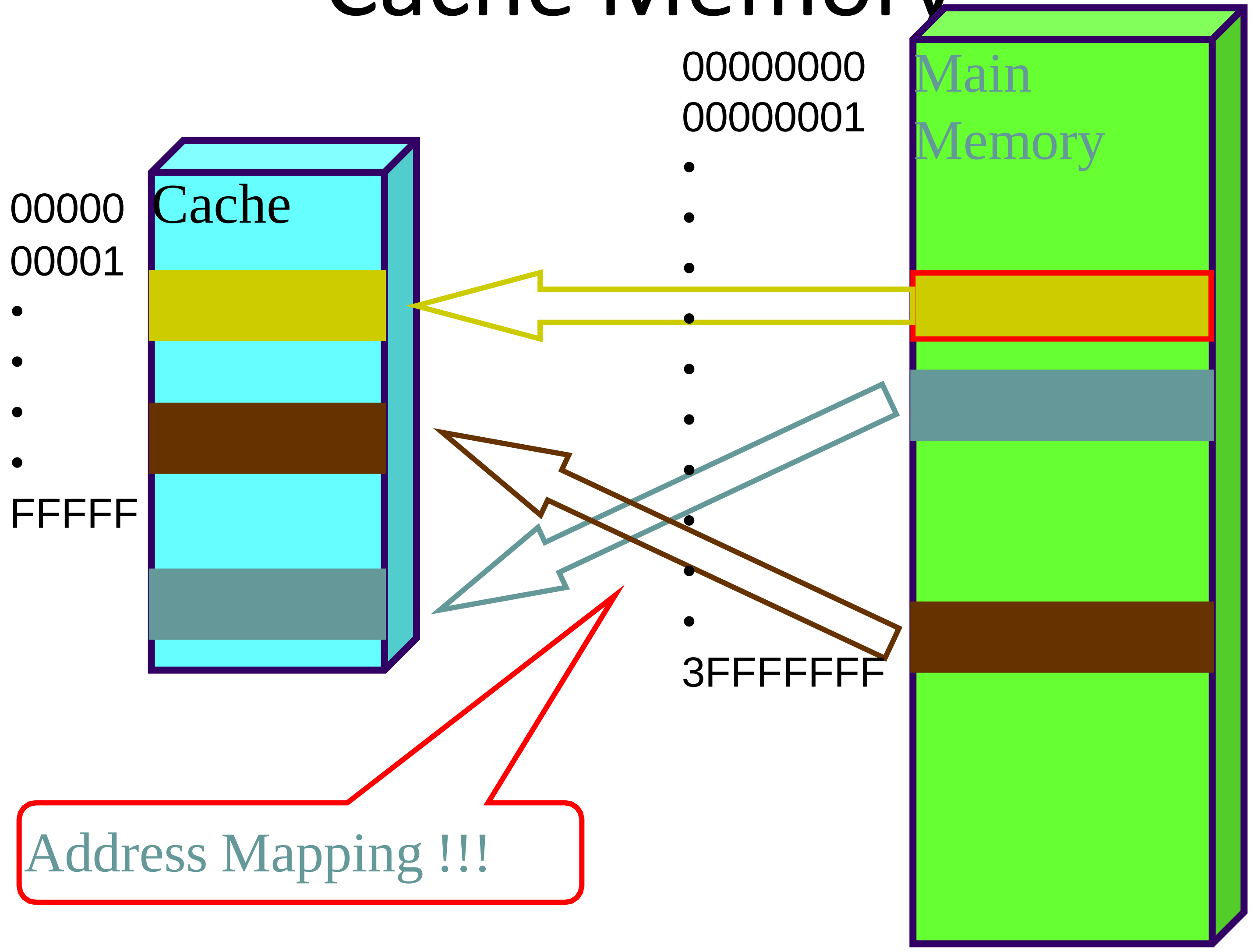


$$\tau_{Access} = 0.95\,\tau_{Cache} + 0.05\,\tau_{Mem}$$

Cache Memory



Cache Memory



Direct Mapping

Block j of main memory maps onto block j modulo 128 of the cache

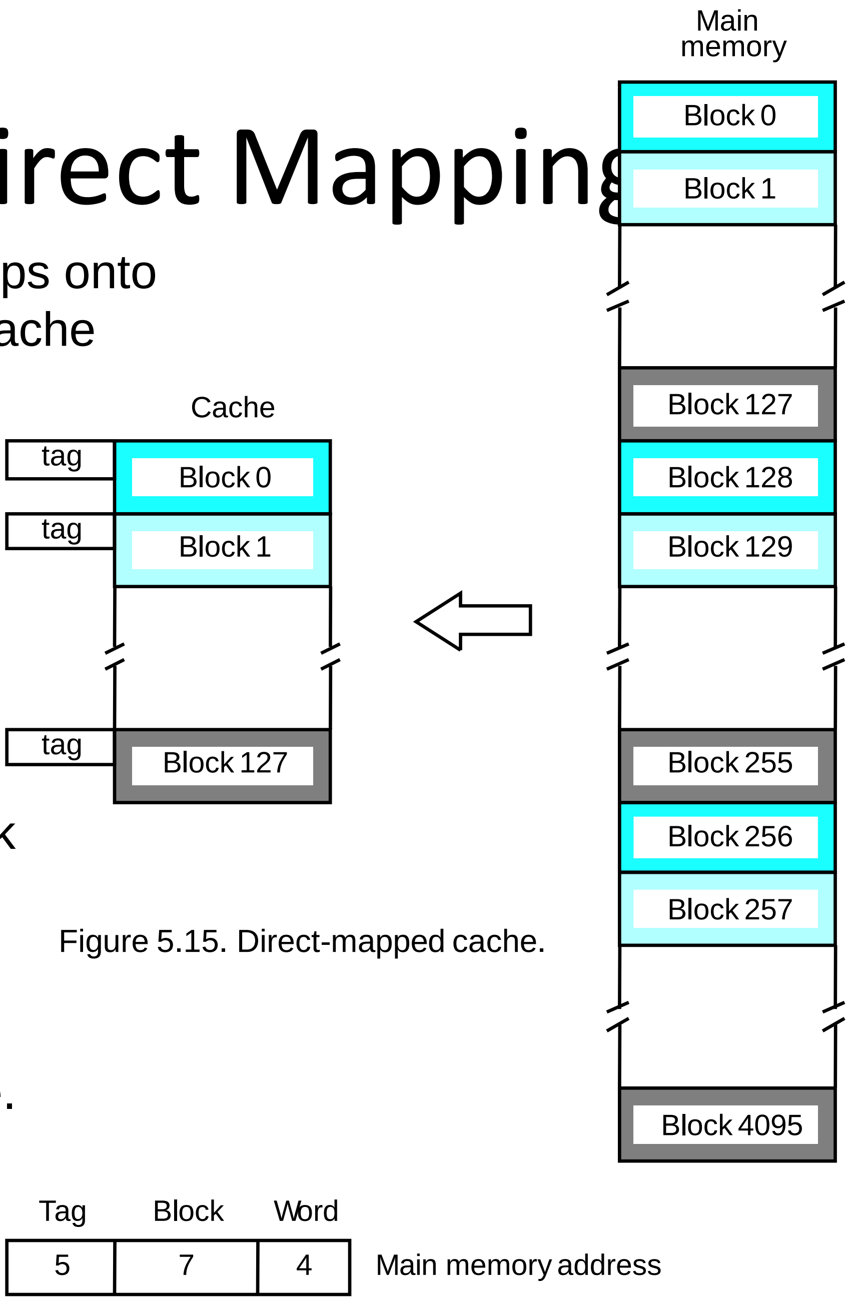


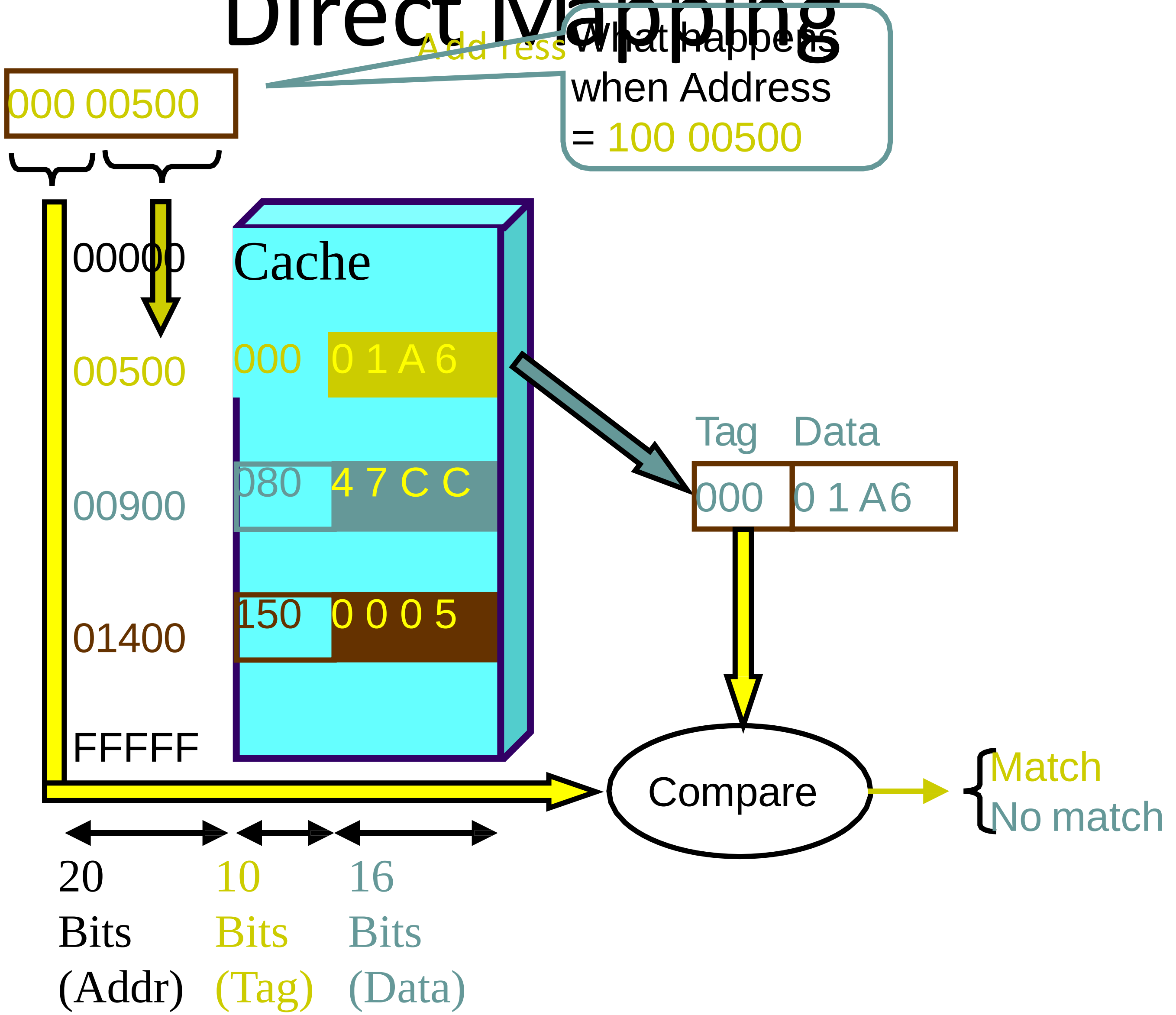
Figure 5.15. Direct-mapped cache.

4: one of 16 words. (each block has $16=2^4$ words)

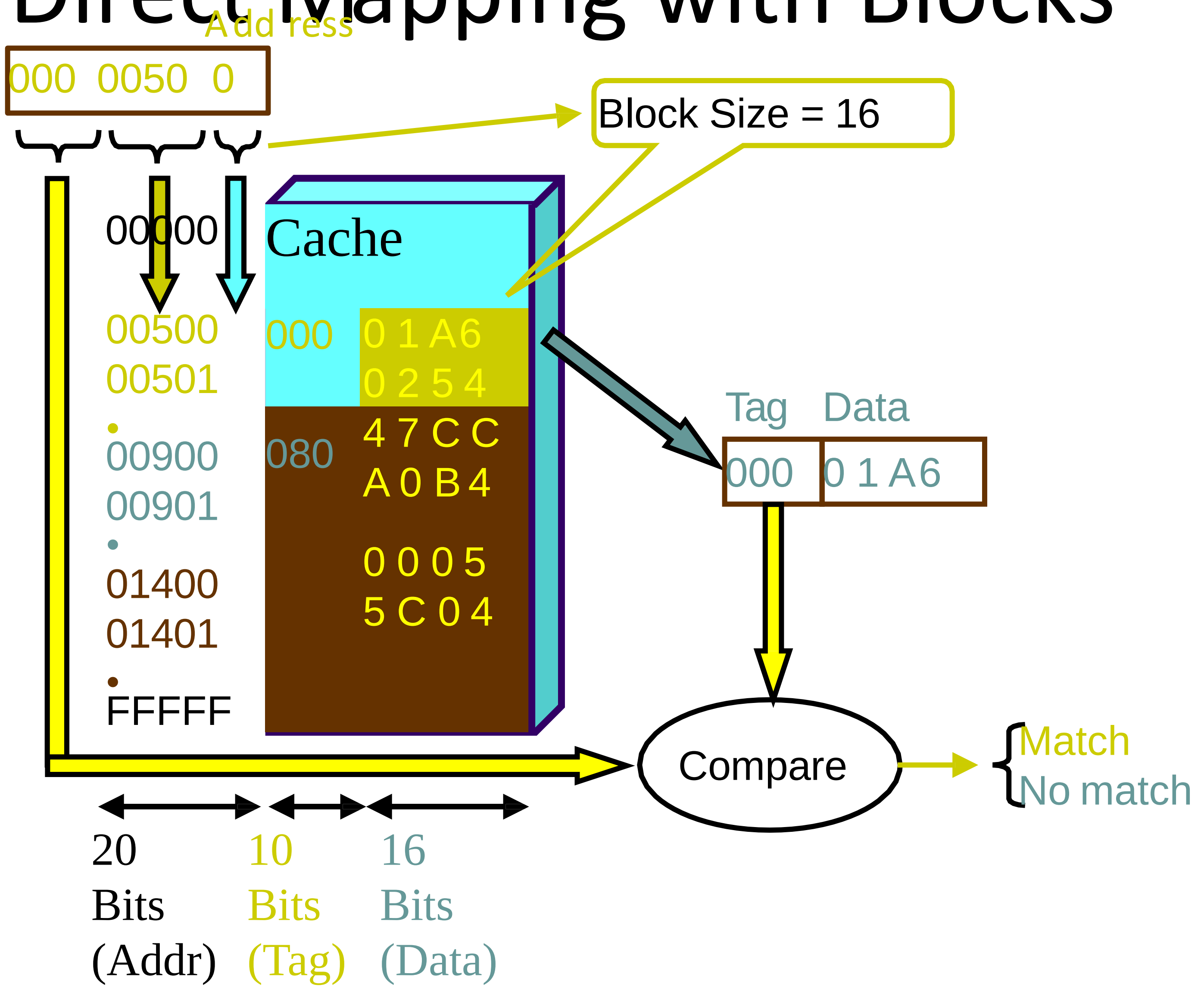
7: points to a particular block in the cache ($128=2^7$)

5: 5 tag bits are compared with the tag bits associated with its location in the cache. Identify which of the 32 blocks that are resident in the cache (4096/128).

Direct Mapping



Direct Mapping with Blocks



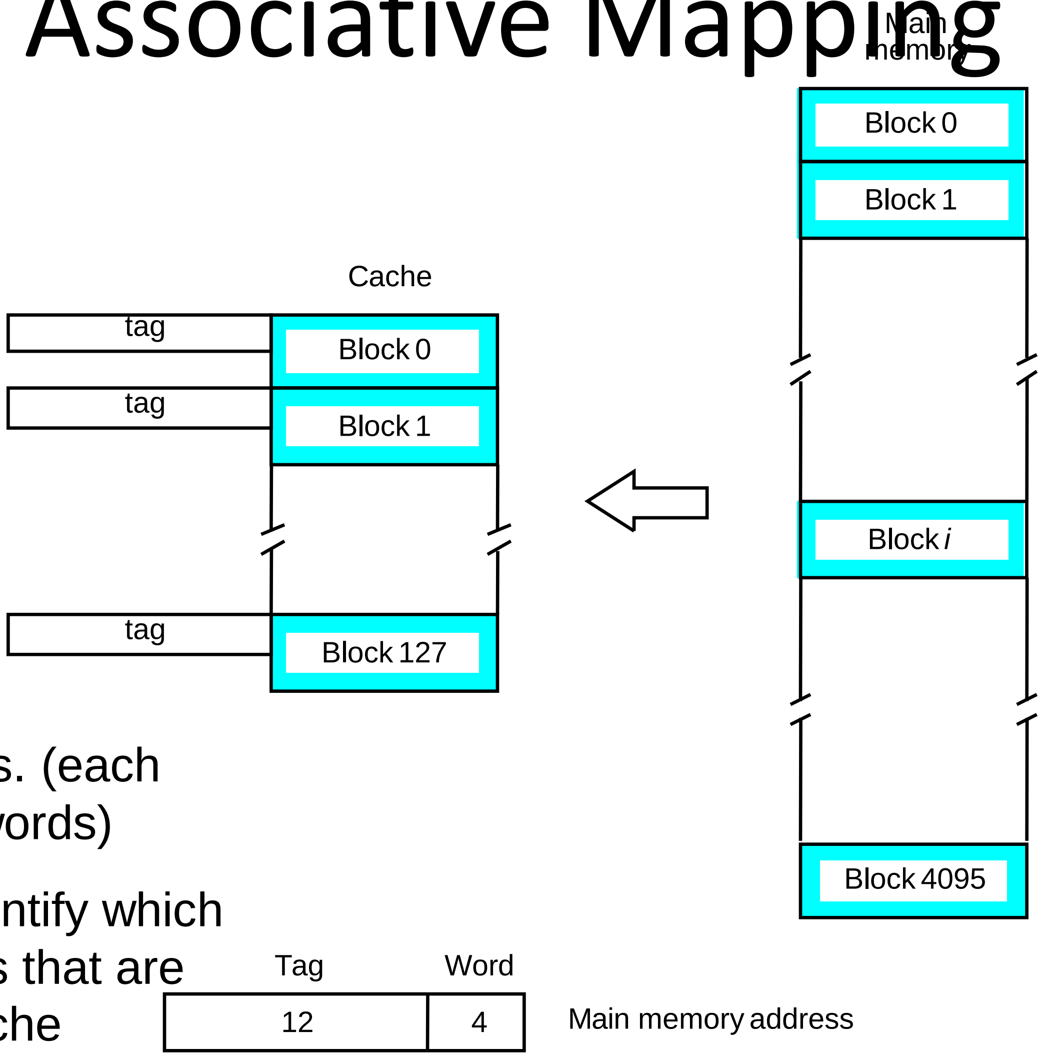
Direct Mapping

Tag	Block	Word	
5	7	4	Main memory address

11101,1111111,1100

- Tag: 11101
- Block: 1111111=127, in the 127th block of the cache
- Word:1100=12, the 12th word of the 127th block in the cache

Associative Mapping

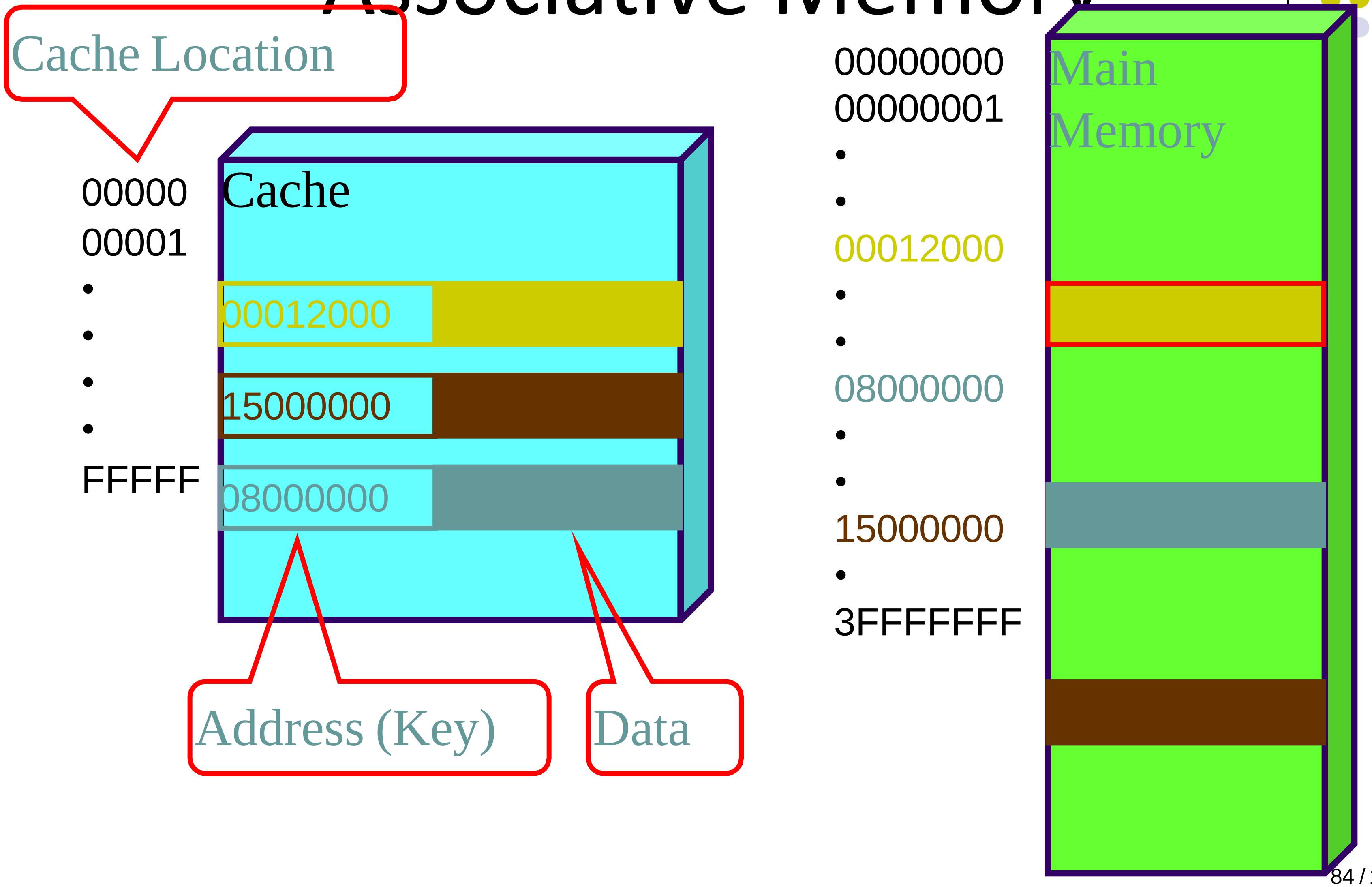


4: one of 16 words. (each block has 16=2^4 words)

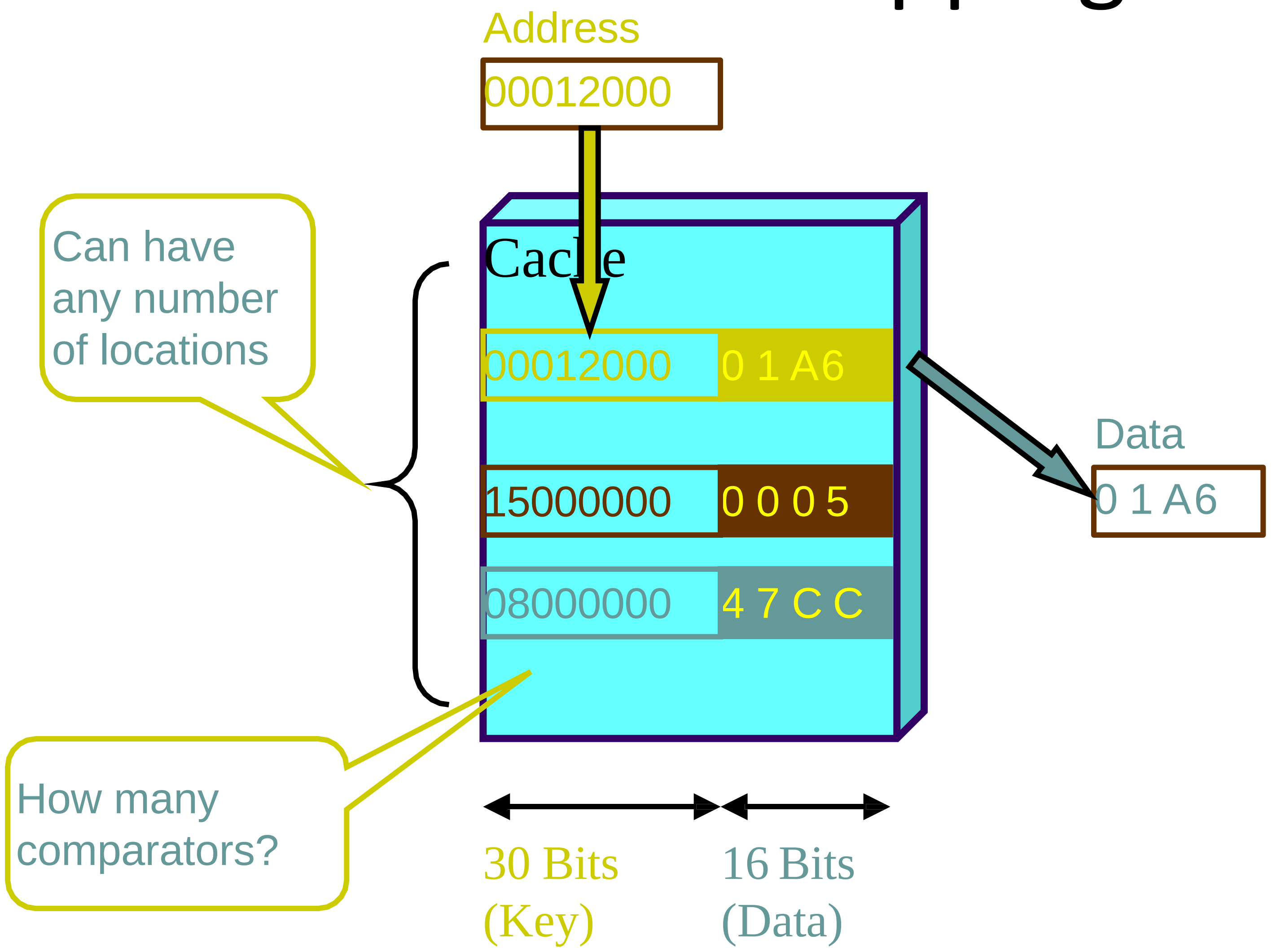
12: 12 tag bits Identify which of the 4096 blocks that are resident in the cache 4096=2^{12}.

Figure 5.16. Associative-mapped cache.

Associative Memory



Associative Mapping



Associative Mapping

Tag	Word	
12	4	Main memory address

111011111111,1100

- Tag: 111011111111
- Word:1100=12, the 12th word of a block in the cache

Set-Associative Mapping

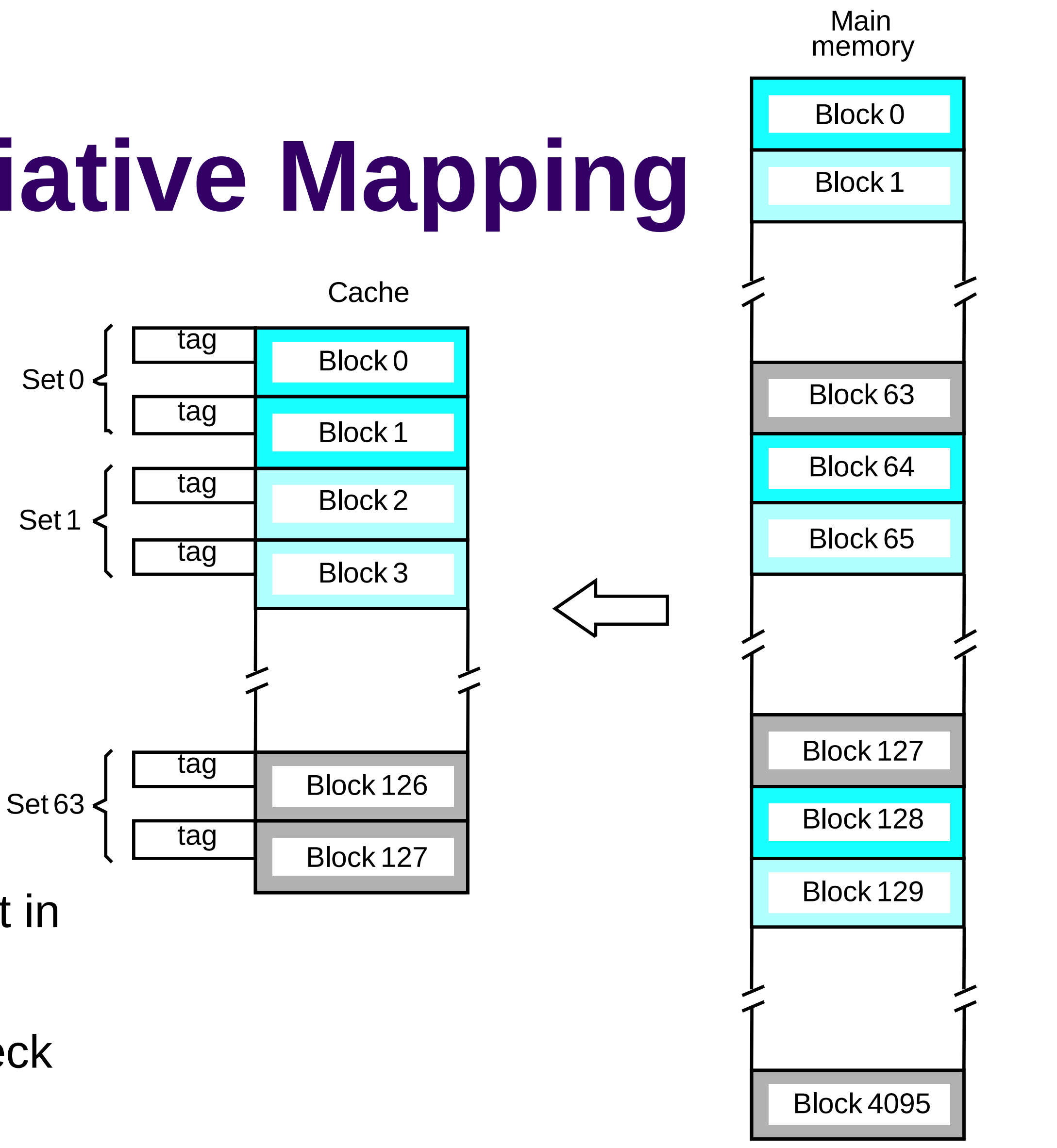


Figure 5.17. Set-associative-mapped cache with two blocks per set.

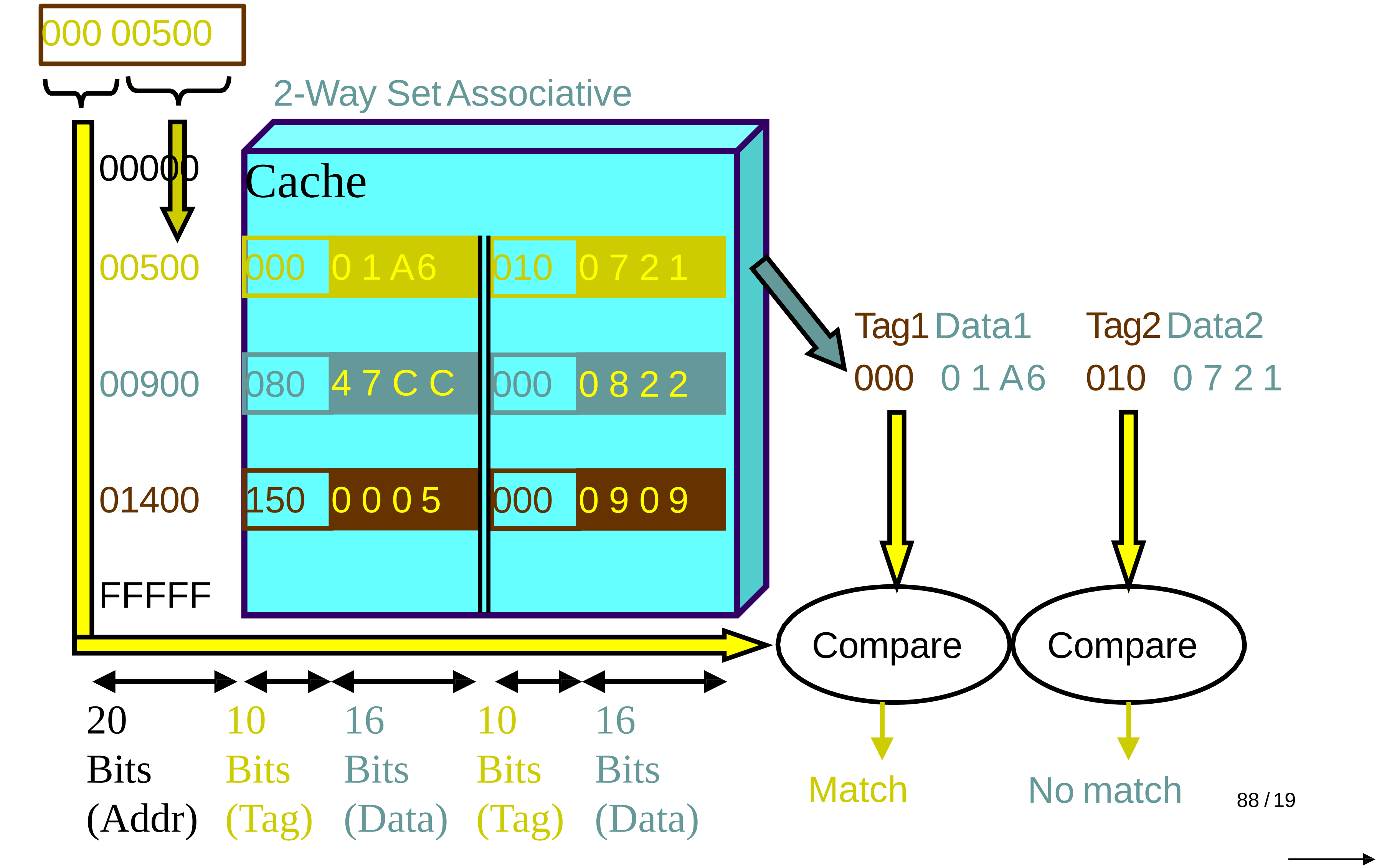
Tag	Set	Word	
6	6	4	Main memory address

4: one of 16 words. (each block has 16=2^4 words)

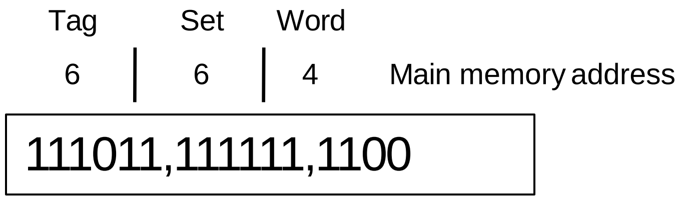
6: points to a particular set in the cache (128/2=64=2^6)

6: 6 tag bits is used to check if the desired block is present (4096/64=2^6).

Set-Associative Mapping



Set-Associative Mapping



- Tag: 111011
- Set: 111111=63, in the 63th set of the cache
- Word:1100=12, the 12th word of the 63th set in the cache

Replacement Algorithms

- Difficult to determine which blocks to kick out
- Least Recently Used (LRU) block
- The cache controller tracks references to all blocks as computation proceeds.
- Increase / clear track counters when a hit/miss occurs

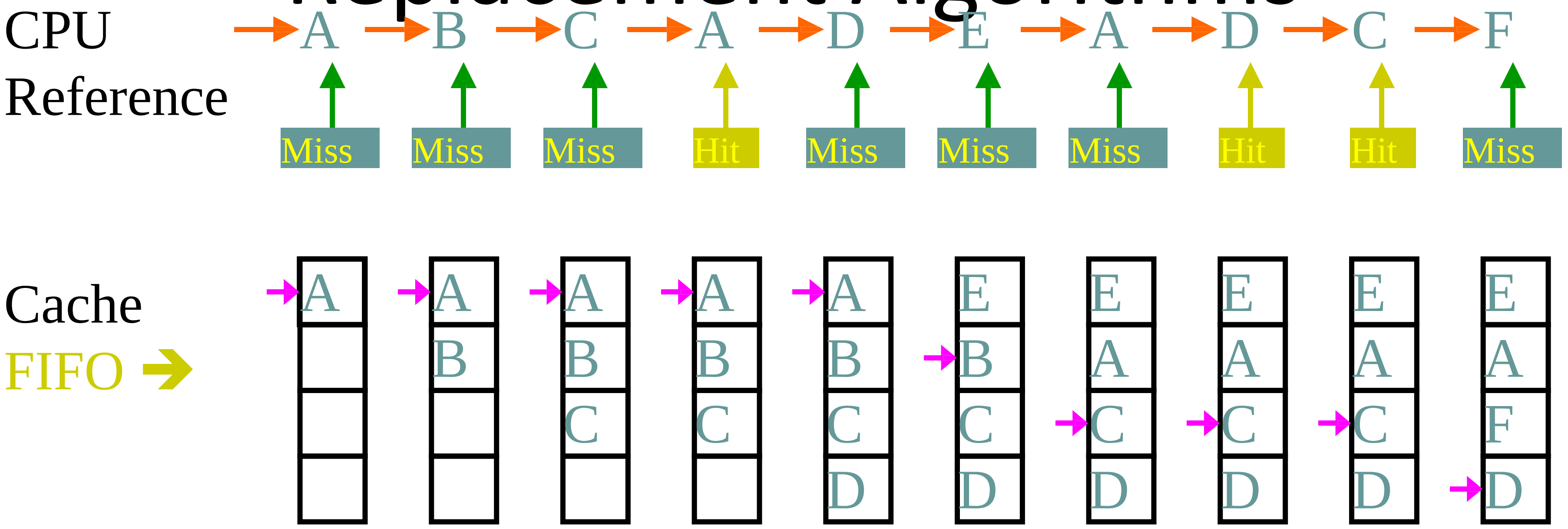
Replacement Algorithms

- For Associative & Set-Associative Cache

  Which location should be emptied when the cache is full and a miss occurs?

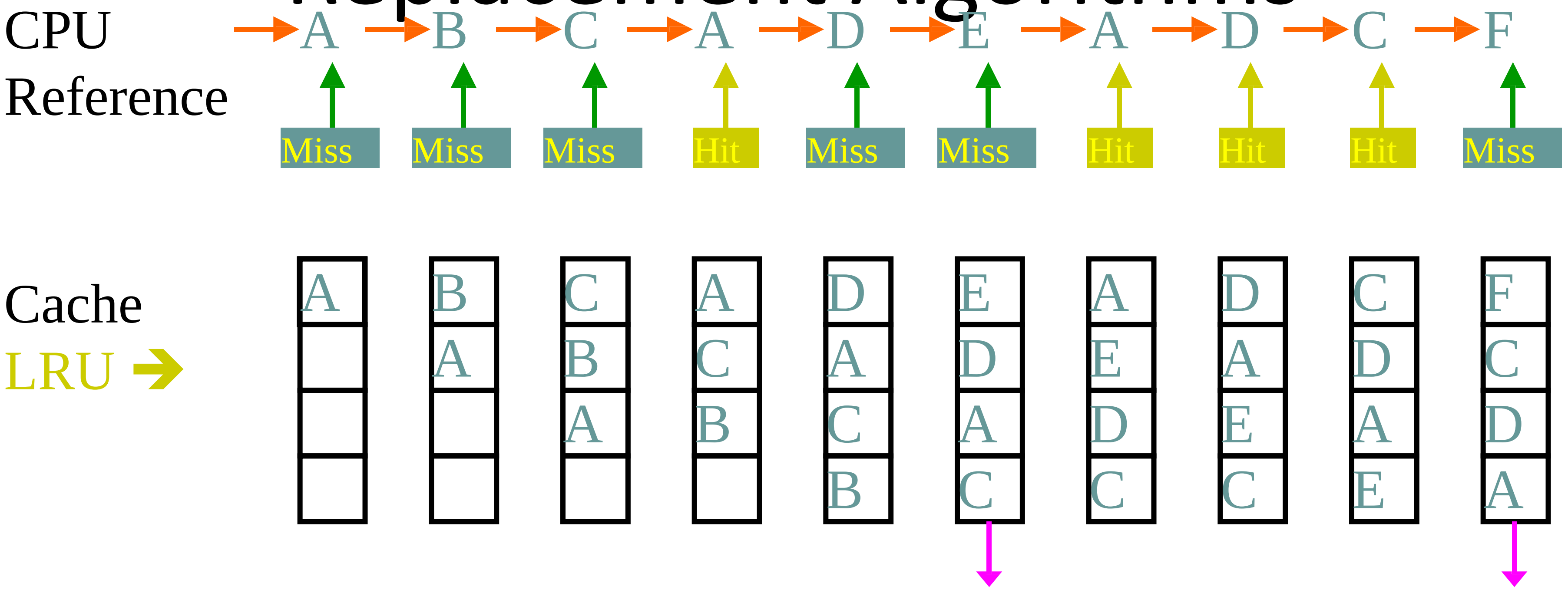
  - First In First Out (FIFO)
  - Least Recently Used (LRU)
- Distinguish an *Empty* location from a *Full* one
  - Valid Bit

Replacement Algorithms



Hit Ratio = 3 / 10 = 0.3

Replacement Algorithms



Hit Ratio = 4 / 10 = 0.4

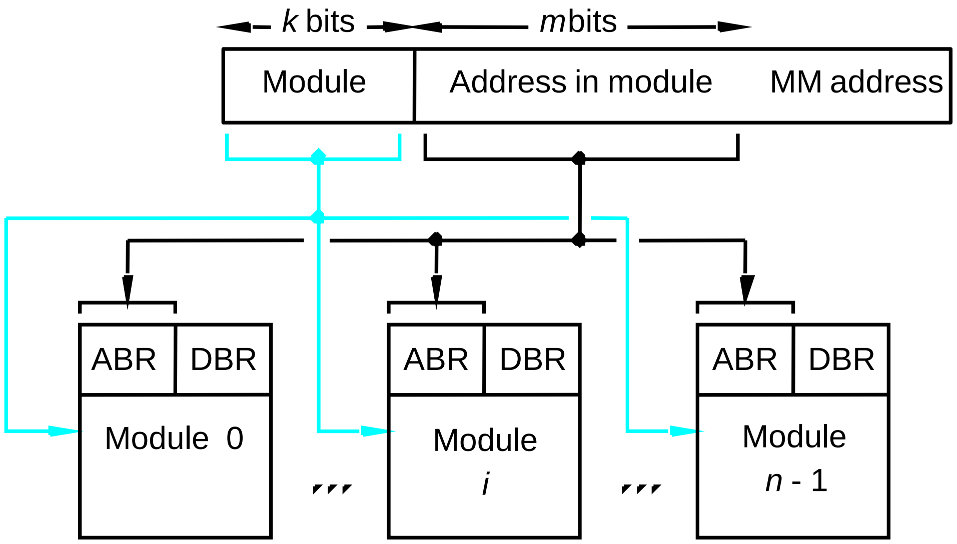
Performance Considerations

Overview

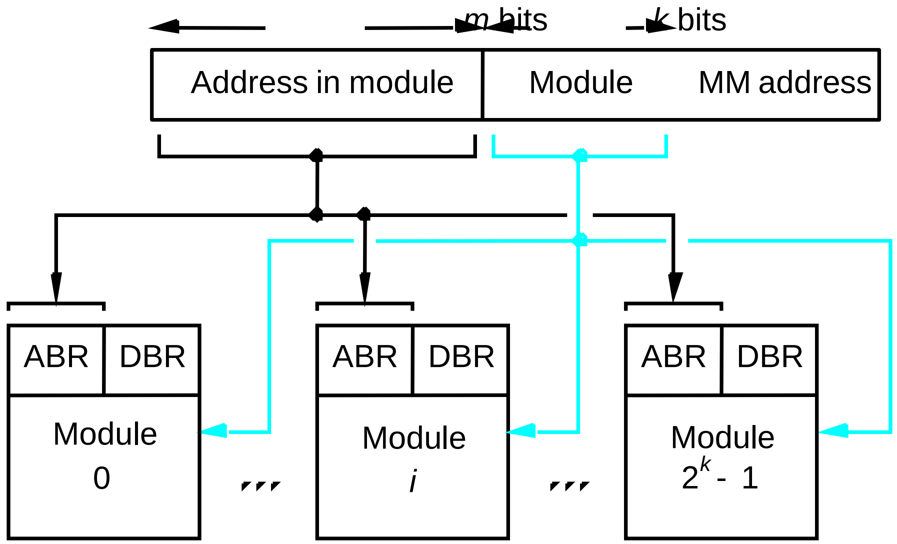
- Two key factors: performance and cost
- Price/performance ratio
- Performance depends on how fast machine instructions can be brought into the processor for execution and how fast they can be executed.
- For memory hierarchy, it is beneficial if transfers to and from the faster units can be done at a rate equal to that of the faster unit.
- This is not possible if both the slow and the fast units are accessed in the same manner.
- However, it can be achieved when parallelism is used in the organizations of the slower unit.

Interleaving

- If the main memory is structured as a collection of physically separated modules, each with its own ABR (Address buffer register) and DBR(Data buffer register), memory access operations may proceed in more than one module at the same time.



(a) Consecutive words in a module



(b) Consecutive words in consecutive modules

Figure 5.25. Addressing multiple-module memory systems.

Hit Rate and Miss Penalty

- The success rate in accessing information at various levels of the memory hierarchy – hit rate / miss rate.
- Ideally, the entire memory hierarchy would appear to the processor as a single memory unit that has the access time of a cache on the processor chip and the size of a magnetic disk – depends on the hit rate (>>0.9).
- A miss causes extra time needed to bring the desired information into the cache.
- Example 5.2, page 332.

Hit Rate and Miss Penalty (cont.)

- $T_{ave}=hC+(1-h)M$
  - $T_{ave:}$ average access time experienced by the processor
  - h: hit rate
  - M: miss penalty, the time to access information in the main memory
  - C: the time to access information in the cache
- Example:
  - Assume that 30 percent of the instructions in a typical program perform a read/write operation, which means that there are 130 memory accesses for every 100 instructions executed.
  - h=0.95 for instructions, h=0.9 for data
  - C=10 clock cycles, M=17 clock cycles, interleaved memory

$$\frac{\text{Time without cache} \qquad 130\times10}{\text{Time with cache} \qquad 100(0.95\times1+0.05\times17)+30(0.9\times1+0.1\times17)} = 5.04$$

- The computer with the cache performs five times better

How to Improve Hit Rate?

- Use larger cache – increased cost
- Increase the block size while keeping the total cache size constant.
- However, if the block size is too large, some items may not be referenced before the block is replaced – miss penalty increases.
- Load-through approach

Caches on the Processor Chip

- On chip vs. off chip
- Two separate caches for instructions and data, respectively
- Single cache for both
- Which one has better hit rate? – Single cache
-

What's the advantage of separating caches? – parallelism, better performance

- Level 1 and Level 2 caches
- L1 cache – faster and smaller. Access more than one word simultaneously and let the processor use them one at a time.
- L2 cache – slower and larger.
- How about the average access time?
- Average access time: $t_{ave} = h_1C_1 + (1-h_1)h_2C_2 + (1-h_1)(1-h_2)M$

where h is the hit rate, C is the time to access information in cache, M is the time to access information in main memory.

Other Enhancements

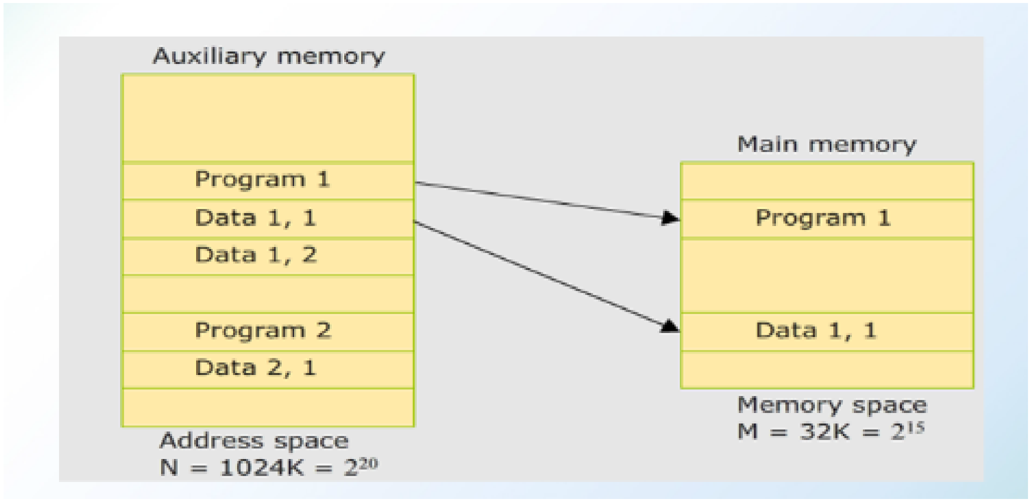
- Write buffer – processor doesn’t need to wait for the memory write to be completed
- Prefetching – prefetch the data into the cache before they are needed
- Lockup-Free cache – processor is able to access the cache while a miss is being serviced.

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
- 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.
- A virtual memory system provides a mechanism for translating program-generated addresses in to correct main memory locations
- The translation or mapping is handled automatically by the hardware by means of a mapping table.

Address space and memory space

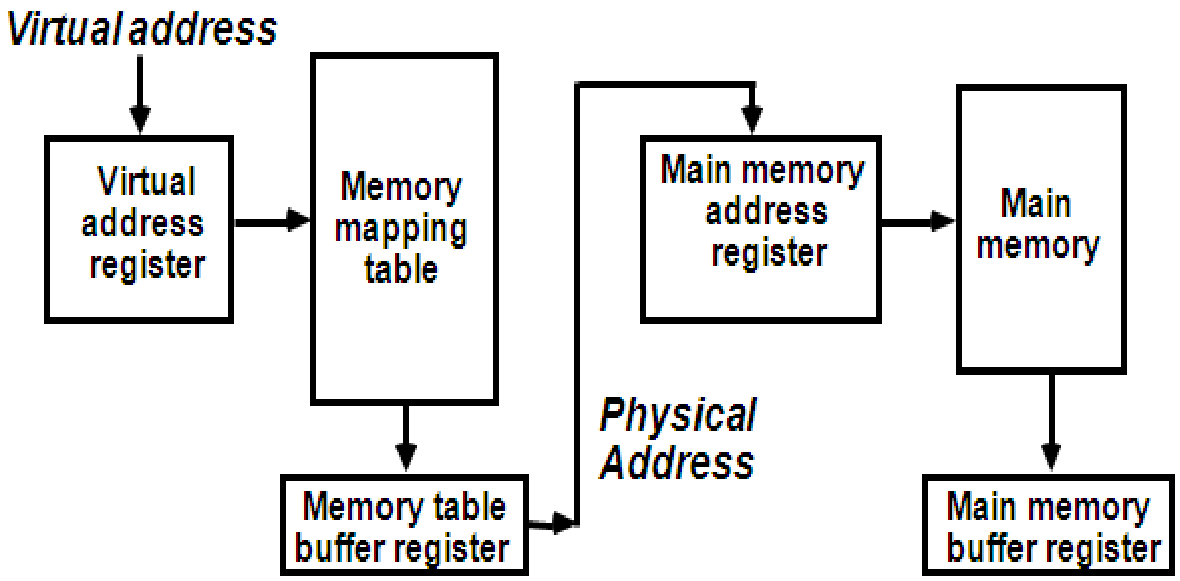
- An address used by a programmer will be called a virtual address, and the set of such addresses the address space.
- An address in main memory is called a location or physical address. the set of such location is called the memory space.



Relation between Address and memory space in a virtual memory system

Mapping using Memory table

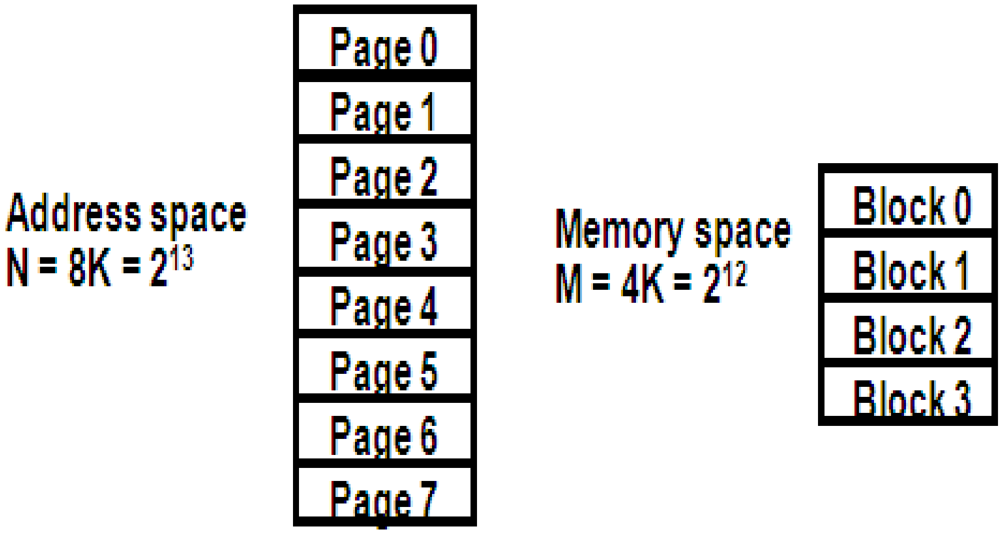
- The mapping table may be stored in a separate memory or in main memory



Memory table for mapping a virtual address

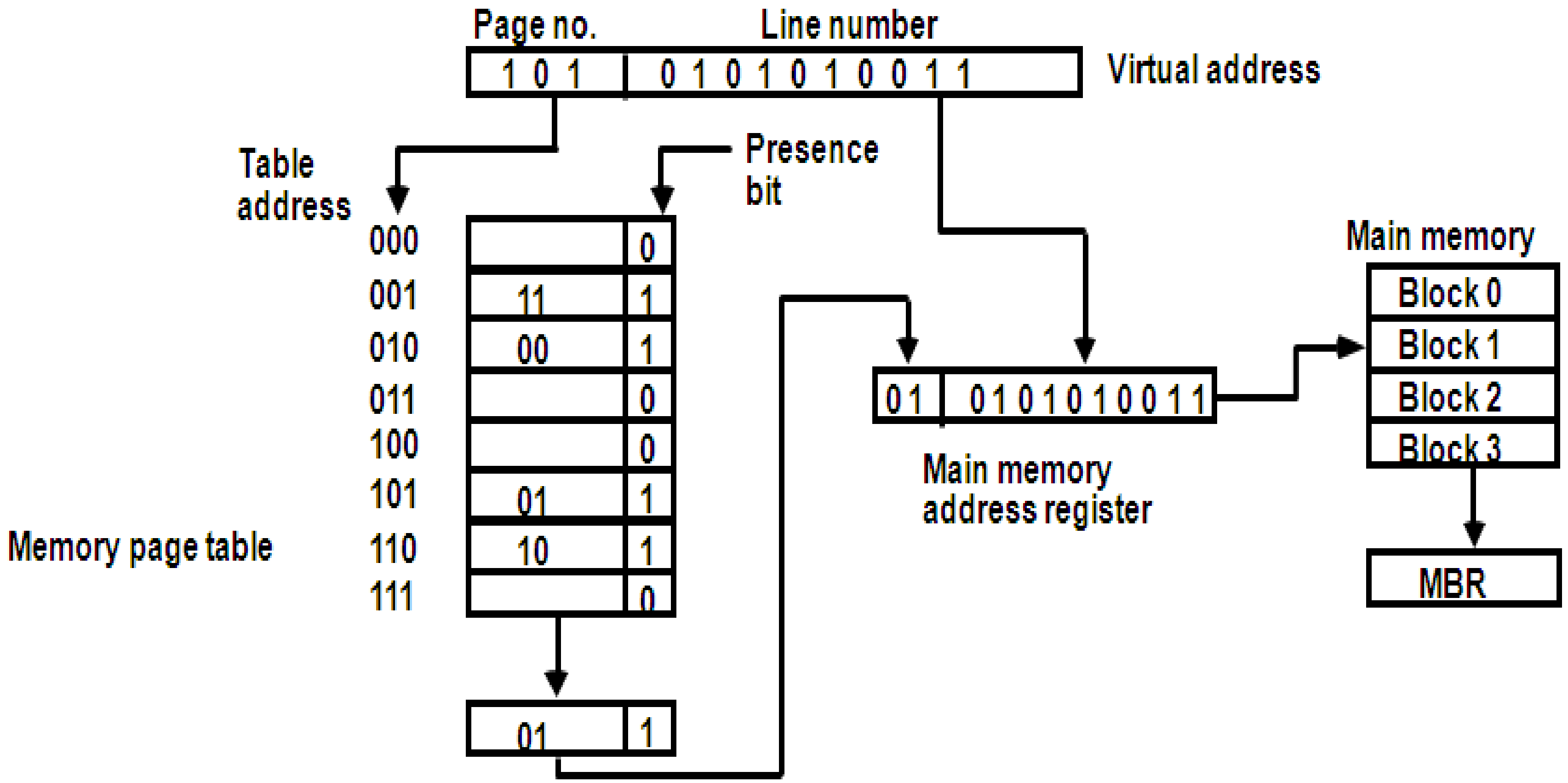
Mapping using paging or page table

- The physical memory is broken down in to groups of equal size blocks.
- The term page refers to groups to groups of address space of the same size as block.



Address space and memory space split in to group of 1K words

Organization of memory Mapping Table in a paged system

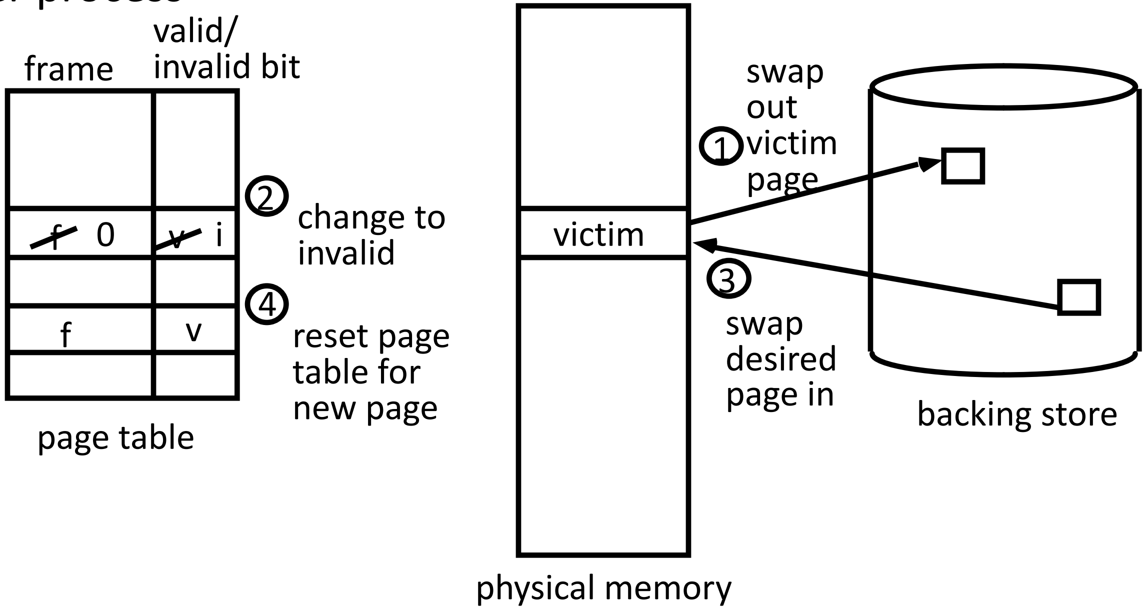


PAGE REPLACEMENT

Decision on which page to displace to make room for an incoming page when no free frame is available

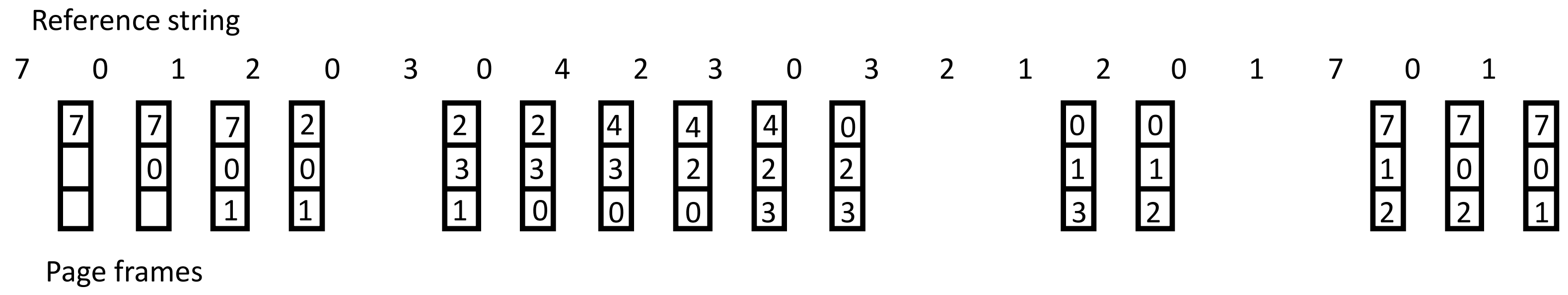
Modified page fault service routine

1. Find the location of the desired page on the backing store
2. Find a free frame
   - If there is a free frame, use it
   - Otherwise, use a page-replacement algorithm to select a *victim* frame
   - Write the victim page to the backing store
3. Read the desired page into the (newly) free frame
4. Restart the user process



PAGE REPLACEMENT ALGORITHMS

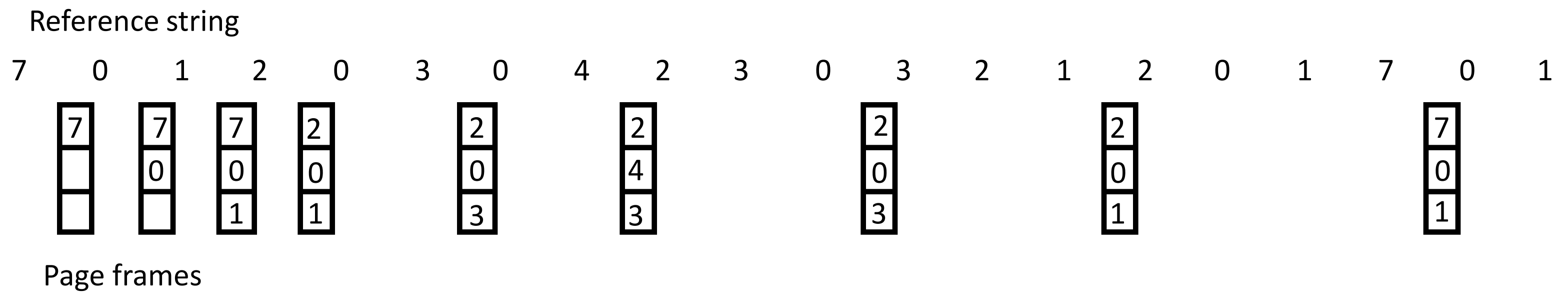
FIFO



FIFO algorithm selects the page that has been in memory the longest time
Using a queue - every time a page is loaded, its
identification is inserted in the queue
Easy to implement
May result in a frequent page fault

Optimal Replacement (OPT) - Lowest page fault rate of all algorithms

Replace that page which will not be used for the longest period of time

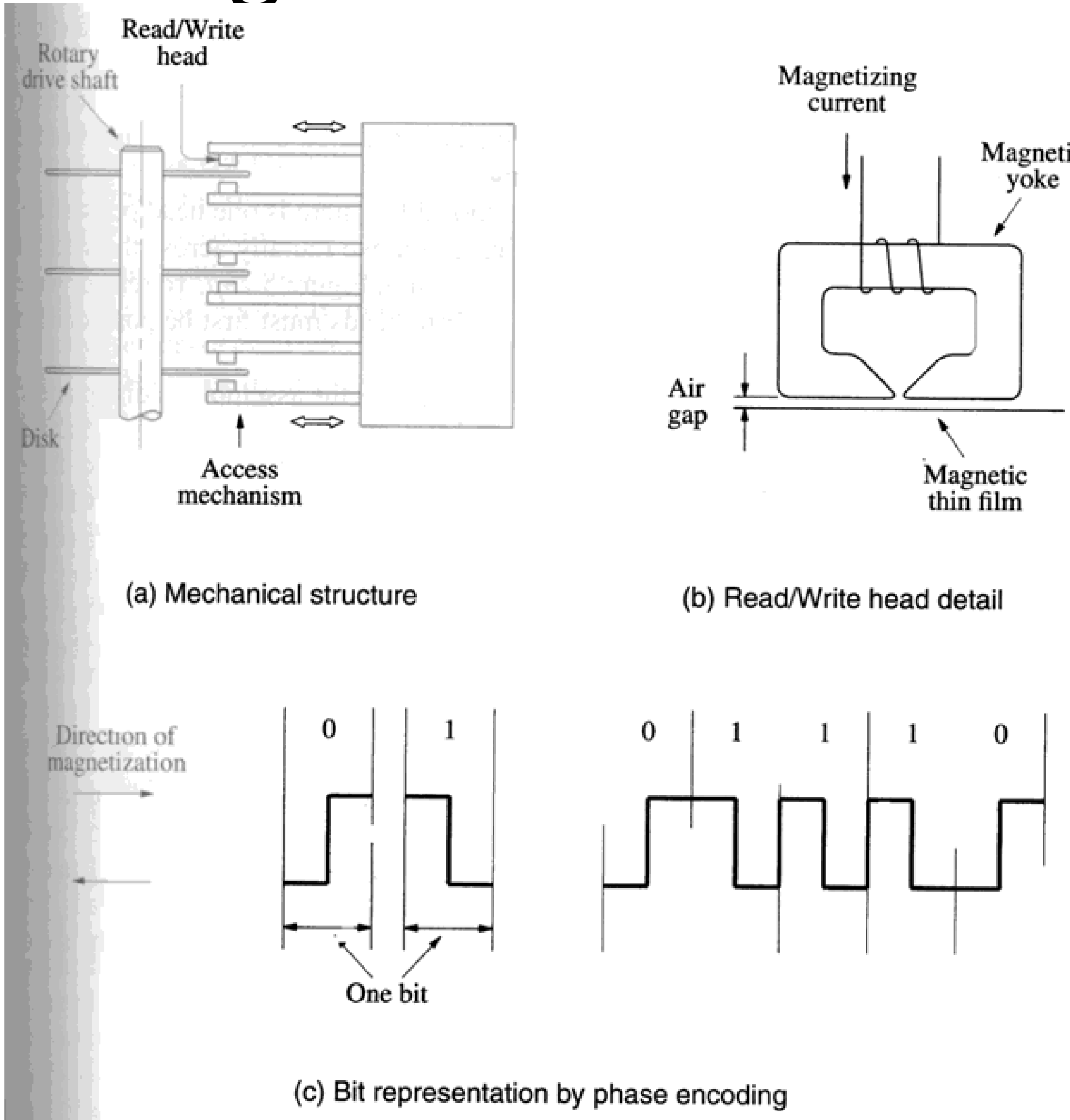


Memory Management Requirements

- Multiple programs
- System space / user space
- Protection (supervisor / user state, privileged instructions)
- Shared pages

Secondary Storage

Magnetic Hard Disks



(a) Mechanical structure

(b) Read/Write head detail

(c) Bit representation by phase encoding

Disk

Disk drive

Disk controller

Organization of Data on a Disk

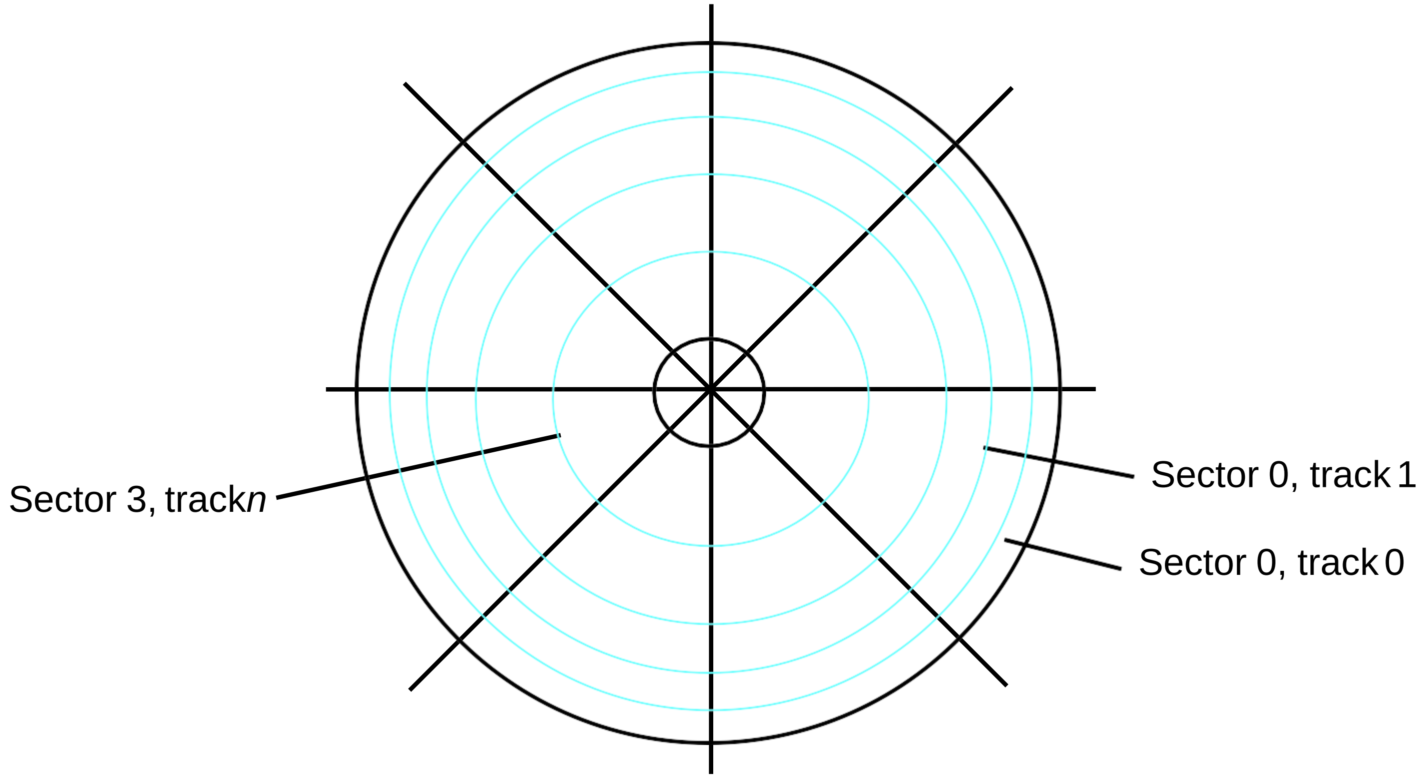


Figure 5.30. Organization of one surface of a disk.

Access Data on a Disk

- Sector header
- Following the data, there is an error-correction code (ECC).
- Formatting process
- Difference between inner tracks and outer tracks
- Access time – seek time / rotational delay (latency time)
- Data buffer/cache

Disk Controller

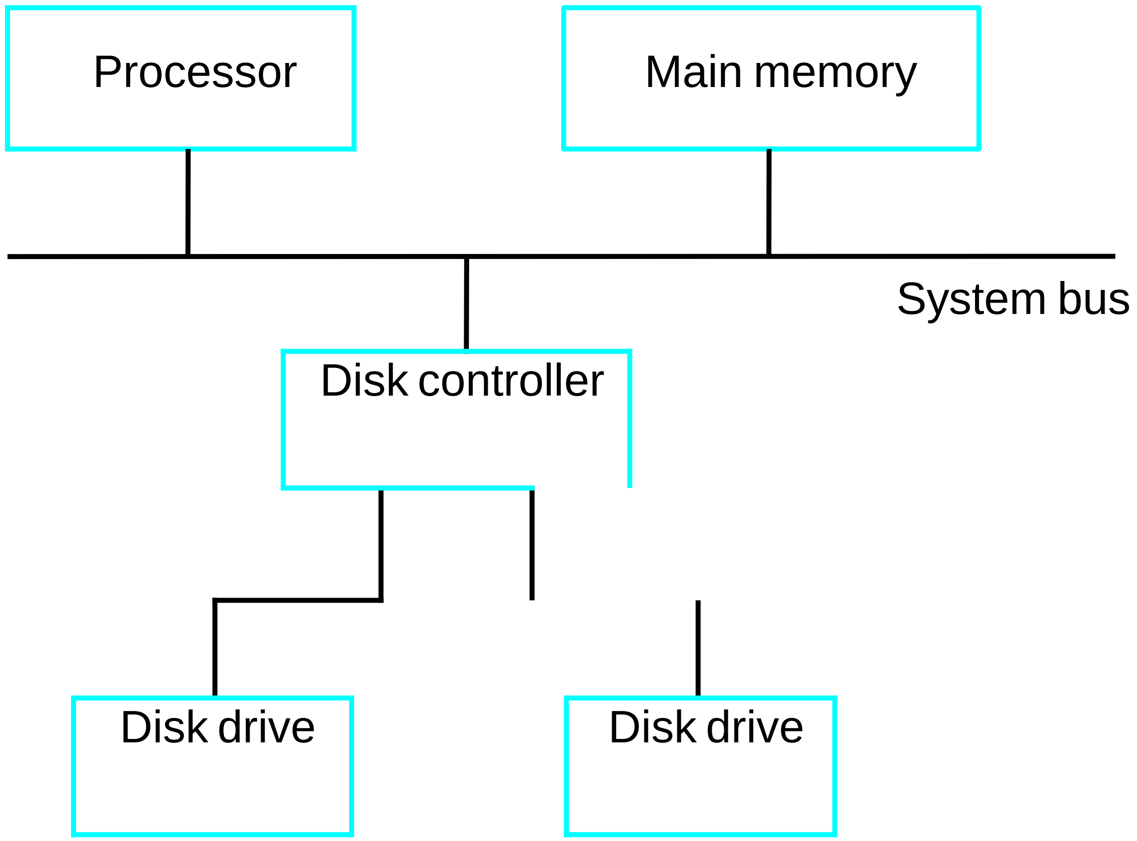


Figure 5.31. Disks connected to the system bus.

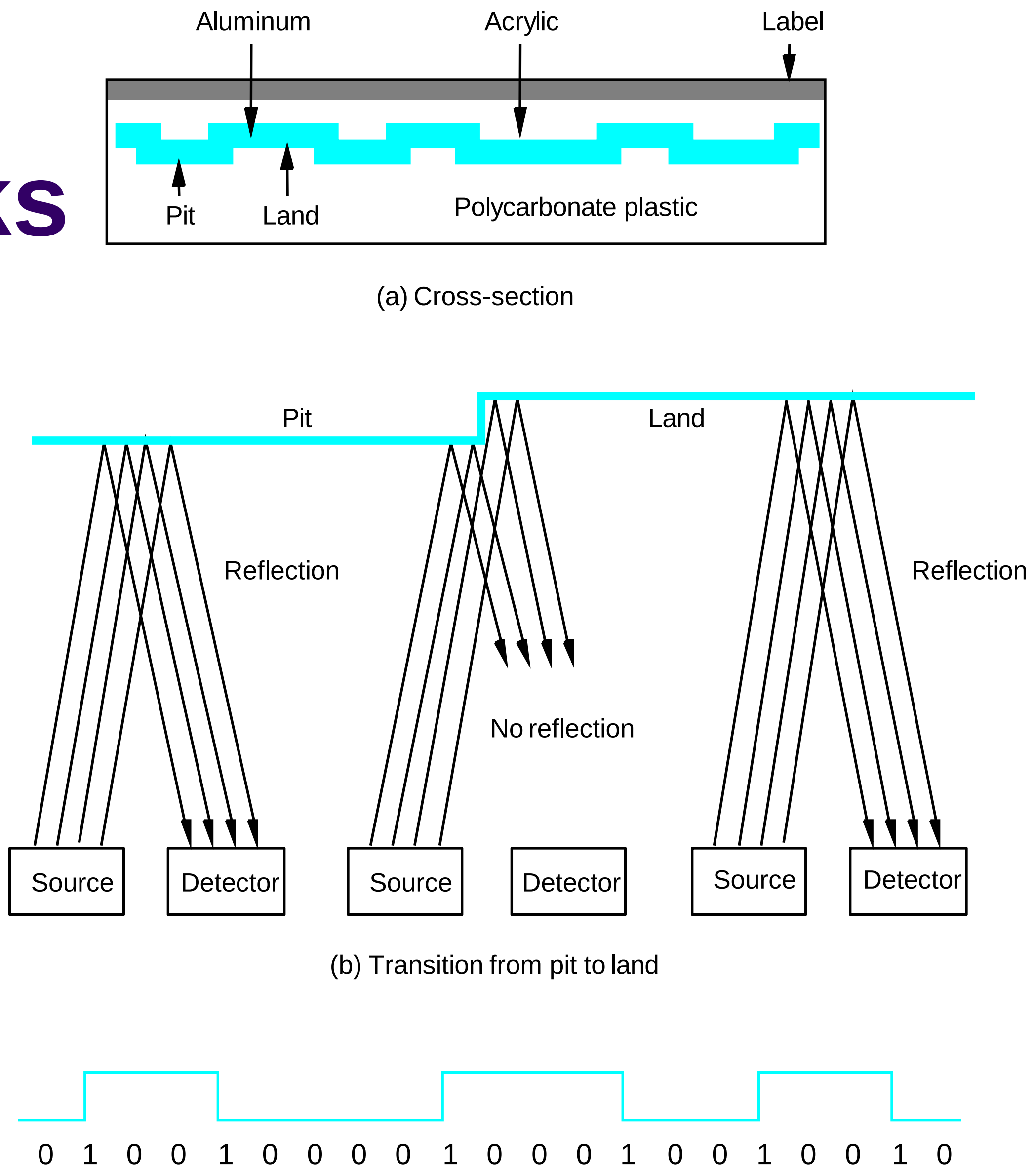
Disk Controller

- Seek
- Read
- Write
- Error checking

RAID Disk Arrays

- Redundant Array of Inexpensive Disks
- Using multiple disks makes it cheaper for huge storage, and also possible to improve the reliability of the overall system.
- RAID0 – data striping
- RAID1 – identical copies of data on two disks
- RAID2, 3, 4 – increased reliability
- RAID5 – parity-based error-recovery

Optical Disks



(a) Cross-section

(b) Transition from pit to land

(c) Stored binary pattern

Figure 5.32. Optical disk.

Optical Disks

- CD-ROM
- CD-Recordable (CD-R)
- CD-ReWritable (CD-RW)
- DVD
- DVD-RAM

Magnetic Tape Systems

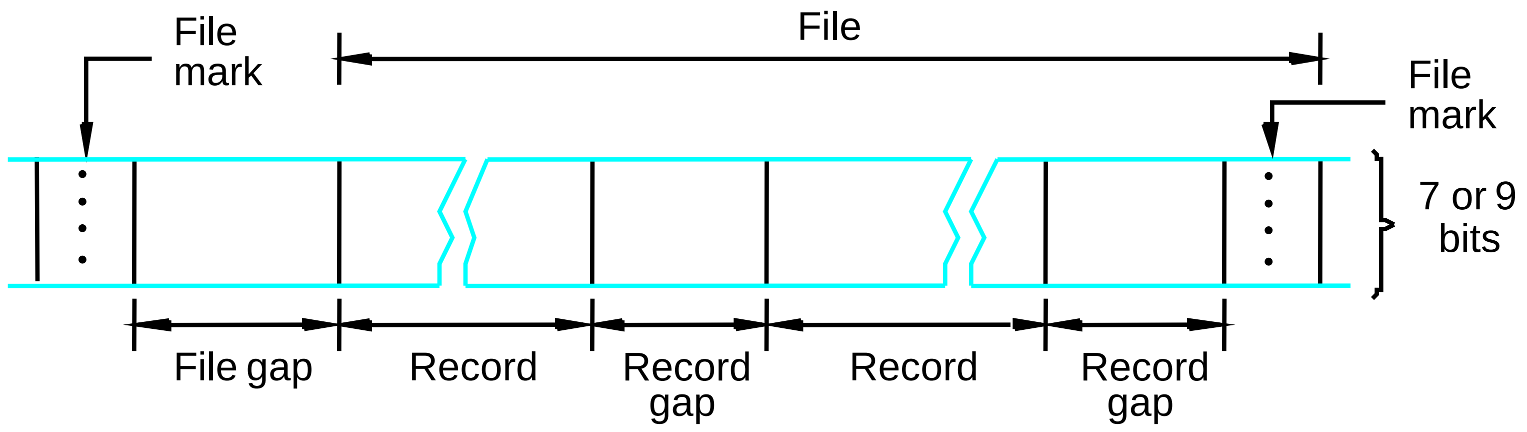


Figure 5.33. Organization of data on magnetic tape.

UNIT-V

Chapter 4. Input/Output Organization

Overview

- Computer has ability to exchange data with other devices.
- Human-computer communication
- Computer-computer communication
- Computer-device communication
- …

Accessing I/O Devices

Single Bus

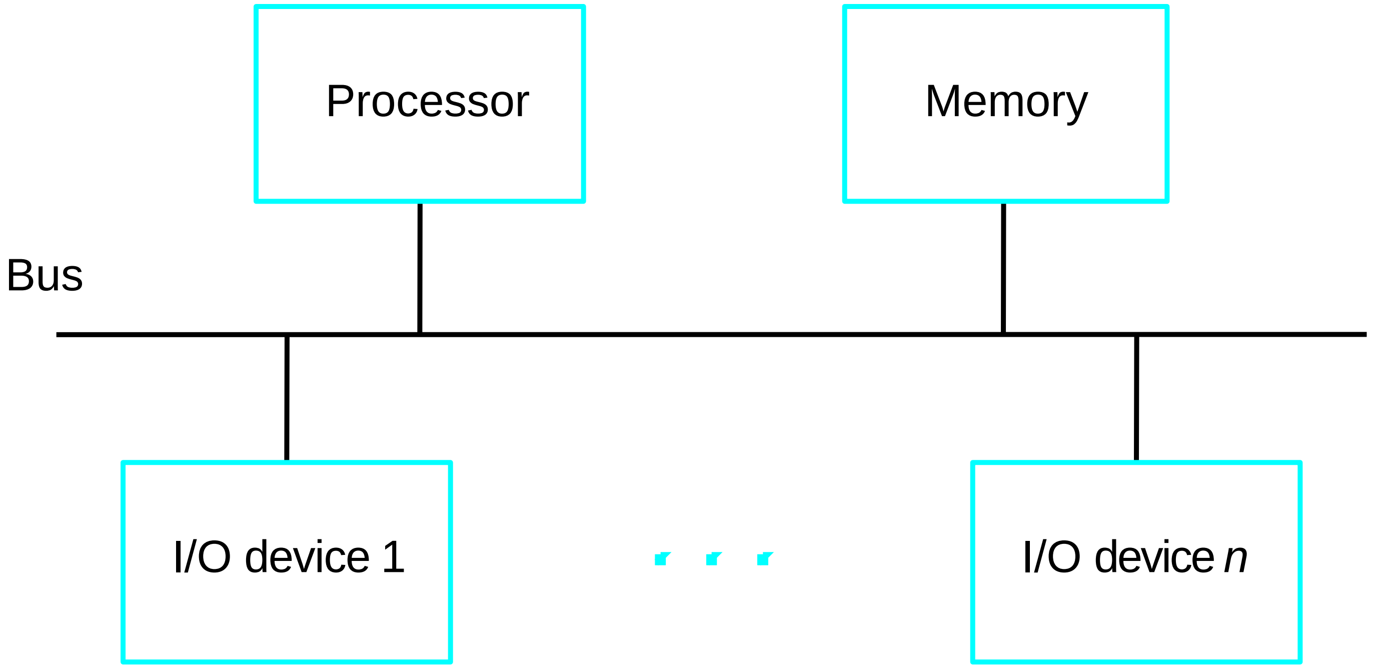


Figure 4.1. A single-bus structure.

Memory-Mapped I/O

- When I/O devices and the memory share the same address space, the arrangement is called memory-mapped I/O.
- Any machine instruction that can access memory can be used to transfer data to or from an I/O device.

Move DATAIN, R0 Move R0, DATAOUT

- Some processors have special In and Out instructions to perform I/O transfer.

Program-Controlled I/O

- I/O devices operate at speeds that are very much different from that of the processor.
- Keyboard, for example, is very slow.
- It needs to make sure that only after a character is available in the input buffer of the keyboard interface; also, this character must be read only once.

Three Major Mechanisms

- Program-controlled I/O – processor polls the device.
- Interrupt
- Direct Memory Access (DMA)

Interrupts

Overview

- In program-controlled I/O, the program enters a wait loop in which it repeatedly tests the device status. During the period, the processor is not performing any useful computation.
- However, in many situations other tasks can be performed while waiting for an I/O device to become ready.
- Let the device alert the processor.

Enabling and Disabling Interrupts

- Since the interrupt request can come at any time, it may alter the sequence of events from that envisaged by the programmer.
- Interrupts must be controlled.

Enabling and Disabling Interrupts

- The interrupt request signal will be active until it learns that the processor has responded to its request. This must be handled to avoid successive interruptions.

➢ Let the interrupt be disabled/enabled in the interrupt-service routine.

➢ Let the processor automatically disable interrupts before starting the execution of the interrupt-service routine.

Handling Multiple Devices

- How can the processor recognize the device requesting an interrupt?
- Given that different devices are likely to require different interrupt-service routines, how can the processor obtain the starting address of the appropriate routine in each case?
- (Vectored interrupts)
- Should a device be allowed to interrupt the processor while another interrupt is being serviced?
- (Interrupt nesting)
- How should two or more simultaneous interrupt requests be handled?
- (Daisy-chain)

Vectored Interrupts

- A device requesting an interrupt can identify itself by sending a special code to the processor over the bus.
- Interrupt vector
- Avoid bus collision

Interrupt Nesting

- Simple solution: only accept one interrupt at a time, then disable all others.
- Problem: some interrupts cannot be held too long.
- Priority structure

Controlling Device Requests

- Some I/O devices may not be allowed to issue interrupt requests to the processor.
- At device end, an interrupt-enable bit in a control register determines whether the device is allowed to generate an interrupt request.
- At processor end, either an interrupt enable bit in the PS register or a priority structure determines whether a given interrupt request will be accepted.

Exceptions

- Recovery from errors
- Debugging
  - Trace
  - Breakpoint
- Privilege exception

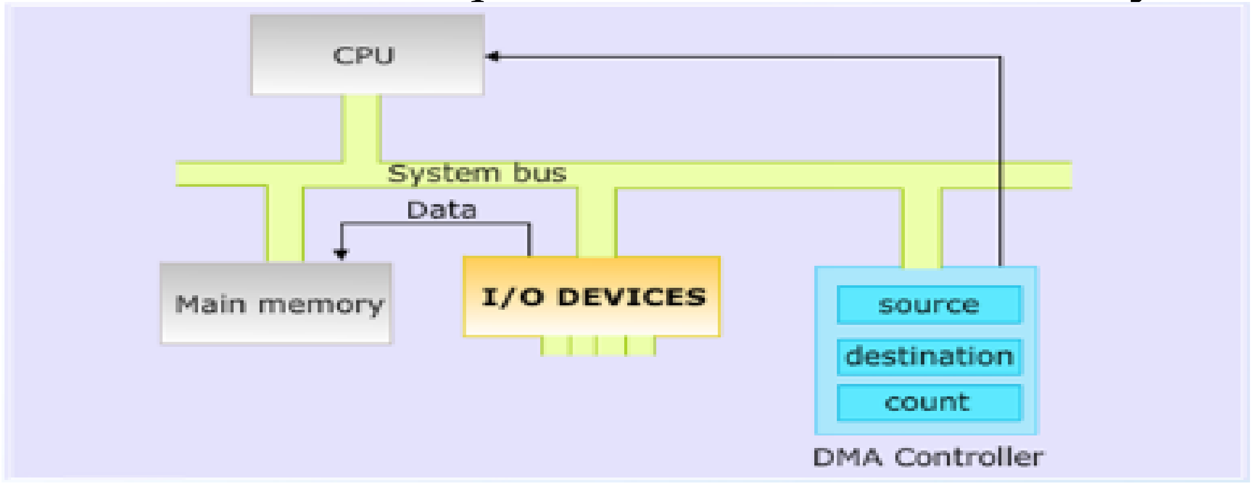
Use of Interrupts in Operating Systems

- The OS and the application program pass control back and forth using software interrupts.
- Supervisor mode / user mode
- Multitasking (time-slicing)
- Process – running, runnable, blocked
- Program state

DIRECT MEMORY ACCESS

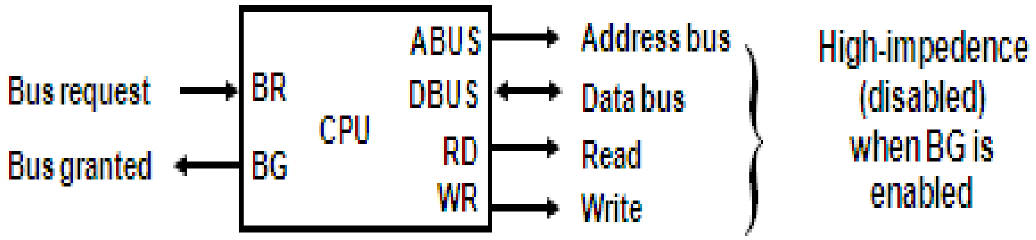
- The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU.
- Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer.

This transfer technique is called direct memory access(DMA)



DMA Function

- During DMA transfer, the CPU is idle and has no control of the memory buses.
- A DMA controller takes over the buses to manage the transfer directly between the I/O devices and memory.
- The CPU may be placed in an idle state in variety of ways.
- One common method exclusively used in microprocessors is to disable the Buses through Special control signals.



DIRECT MEMORY ACCESS

- **BUS REQUEST**

  The bus request(BR) input is used by the DMA controller to request the CPU to relinquish control of the buses.

- **BUS GRANT**

  The CPU activates the bus grant(BG) output to inform the external DMA that the buses are in the high-impedance state

- **BUS TRANSFER**

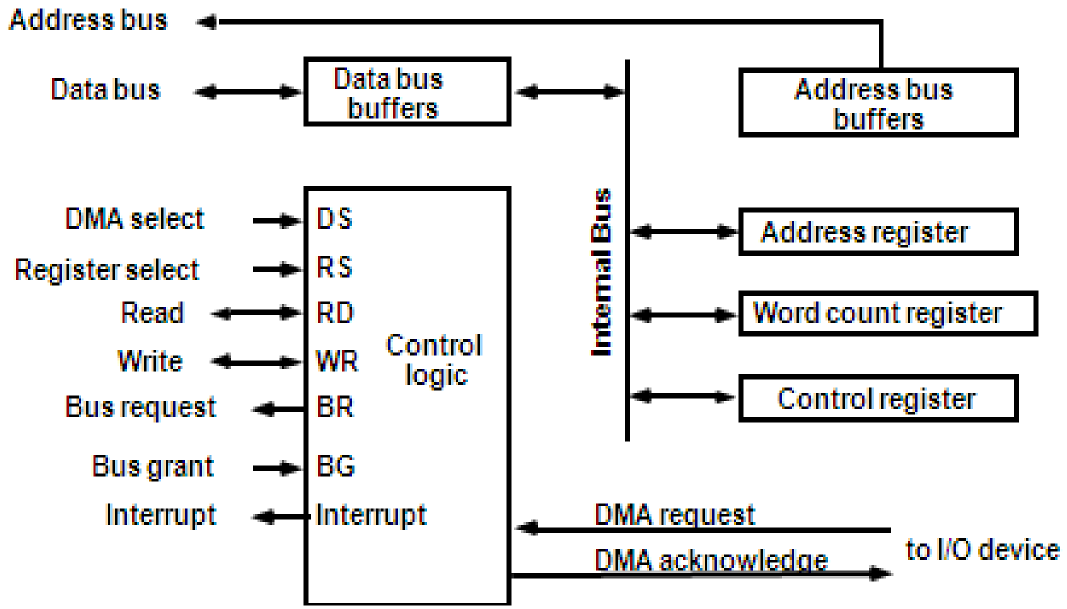
  In DMA burst transfer, a block sequence consisting of a number of memory words is transferred in a continuous burst while the DMA controller is mater of the memory buses.

- **CYCLE STEALING**

  An alternative technique called cycle stealing allows the DMA controller to transfer one data word at a time, after which it must return control of the buses to the CPU.

DMA Controller

- The DMA controller needs the usual circuits of an interface to communicate with the CPU and I/O device.
- The address register and lines are used for direct communication with the memory.
- The word count register specifies the number of words that must be transferred. The data transfer may be done directly between the device and memory under control of the DMA.



Working Of DMA Controller

- The unit communicate with the CPU via the data bus and control lines. The registers in the DMA are selected by the CPU through the address bus by enabling the DS(DMA select) and RS(register select) inputs. The RD(read) and WR(write) input are bidirectional.
- When the BG(bus grant) input is 0,the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers. when BG=1,the CPU has relinquished the buses and the DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control.
- The DMA communicate with the external peripheral through the request and acknowledge lines by using a prescribed handshaking procedure. The DMA controller has three registers: an address register, a word count register, and a control register.
- The address register contains an address to specify the desired location in memory. The address bits go through bus buffers in to the address bus.

DMA controller working

The CPU initializes the DMA by sending the following information through the data bus.

1.The starting address of the memory block where data are available(for read) or where data are to be stored(for write).

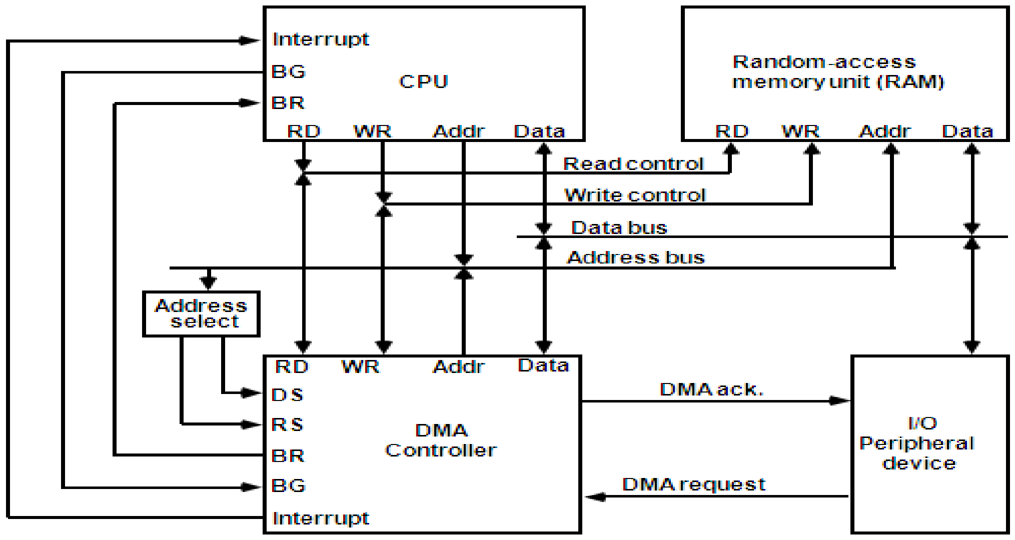
2.The word count, which is the number of words in the memory block.

3.Control to specify the mode of transfer such as read or write.

4.A control to start the DMA transfer.

DMA TRANSFER

- The CPU communicate with the DMA through the address and data buses as with any interface unit.
- The DMA has its own address, which activates the DS and RS lines
- The CPU initializes the DMA through the data bus. once the DMA receives the start control command, it can start the transfer between the peripheral device and the memory.



Bus Arbitration

- The device that is allowed to initiate data transfers on the bus at any given time is called the bus master.
- Bus arbitration is the process by which the next device to become the bus master is selected and bus mastership is transferred to it.
- Need to establish a priority system.
- Two approaches: centralized and distributed

Buses

Overview

- The primary function of a bus is to provide a communications path for the transfer of data.
- A bus protocol is the set of rules that govern the behavior of various devices connected to the bus as to when to place information on the bus, assert control signals, etc.
- Three types of bus lines: data, address, control
- The bus control signals also carry timing information.
- Bus master (initiator) / slave (target)

Discussion

- Trade-offs
  - Simplicity of the device interface
  - Ability to accommodate device interfaces that introduce different amounts of delay
  - Total time required for a bus transfer
  - Ability to detect errors resulting from addressing a nonexistent device or from an interface malfunction
- Asynchronous bus is simpler to design.
- Synchronous bus is faster.

Interface Circuits

Function of I/O Interface

- Provide a storage buffer for at least one word of data;
- Contain status flags that can be accessed by the processor to determine whether the buffer is full or empty;
- Contain address-decoding circuitry to determine when it is being addressed by the processor;
- Generate the appropriate timing signals required by the bus control scheme;
- Perform any format conversion that may be necessary to transfer data between the bus and the I/O device.

Parallel Port

- A parallel port transfers data in the form of a number of bits, typically 8 or 16, simultaneously to or from the device.
- For faster communications

Parallel Port – Input Interface (Keyboard to Processor Connection)

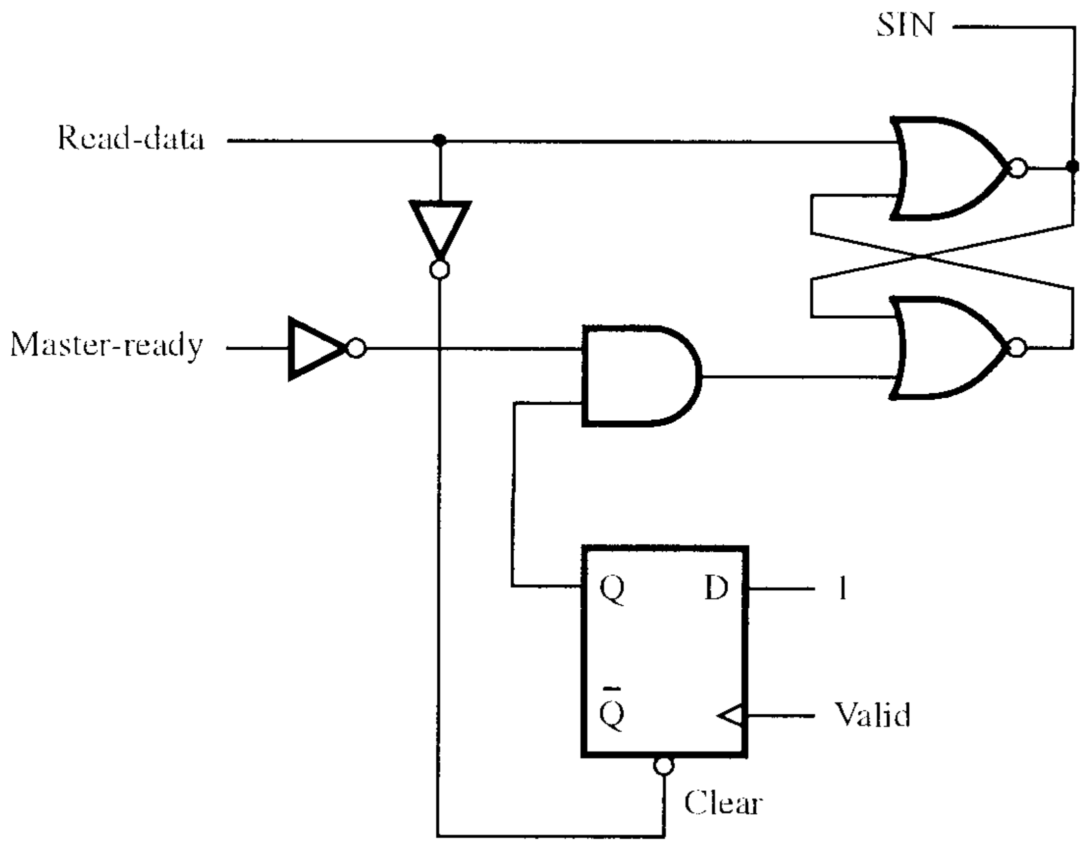


Figure 4.30. Circuit for the status flag block in Figure 4.29.

Parallel Port – Output Interface (Printer to Processor Connection)

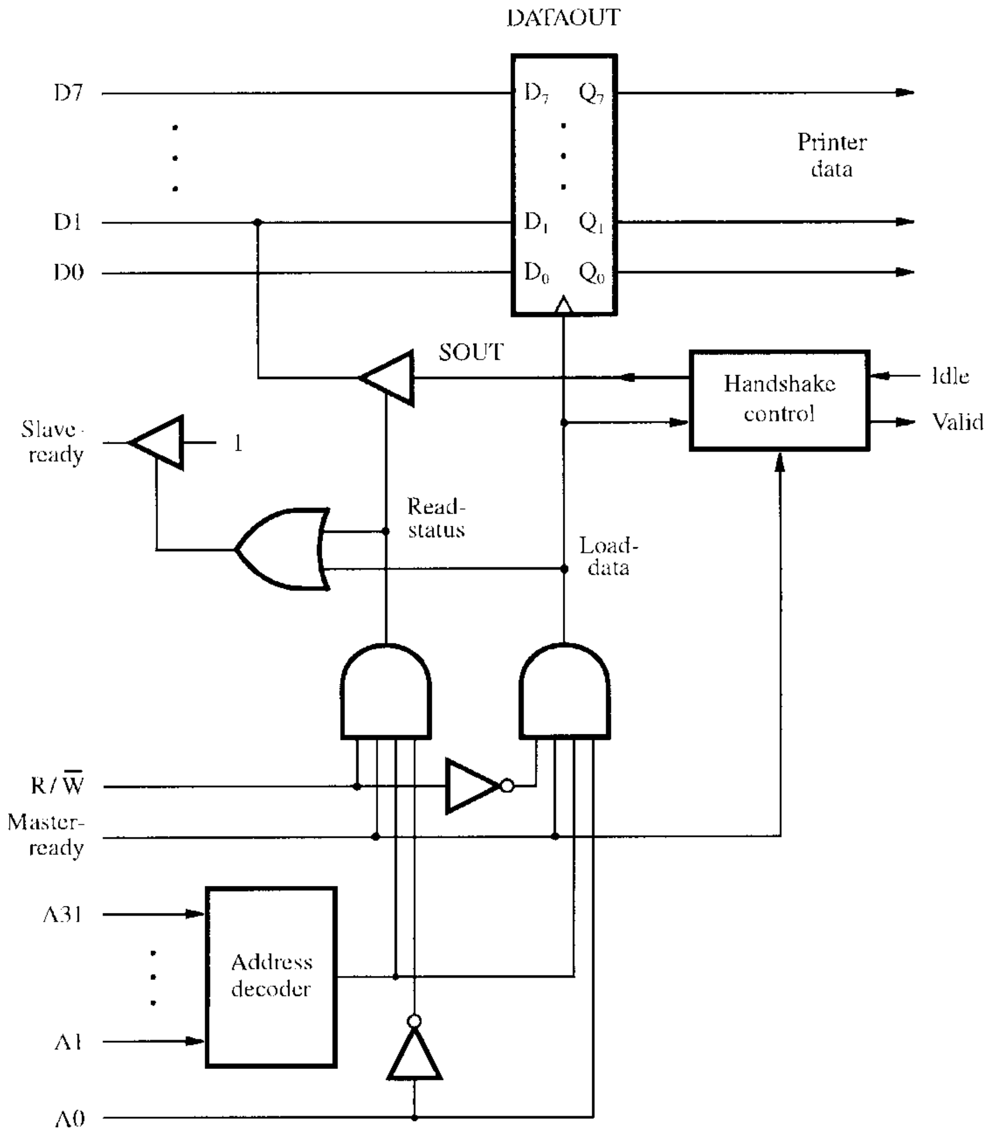


Figure 4.32. Output interface circuit.

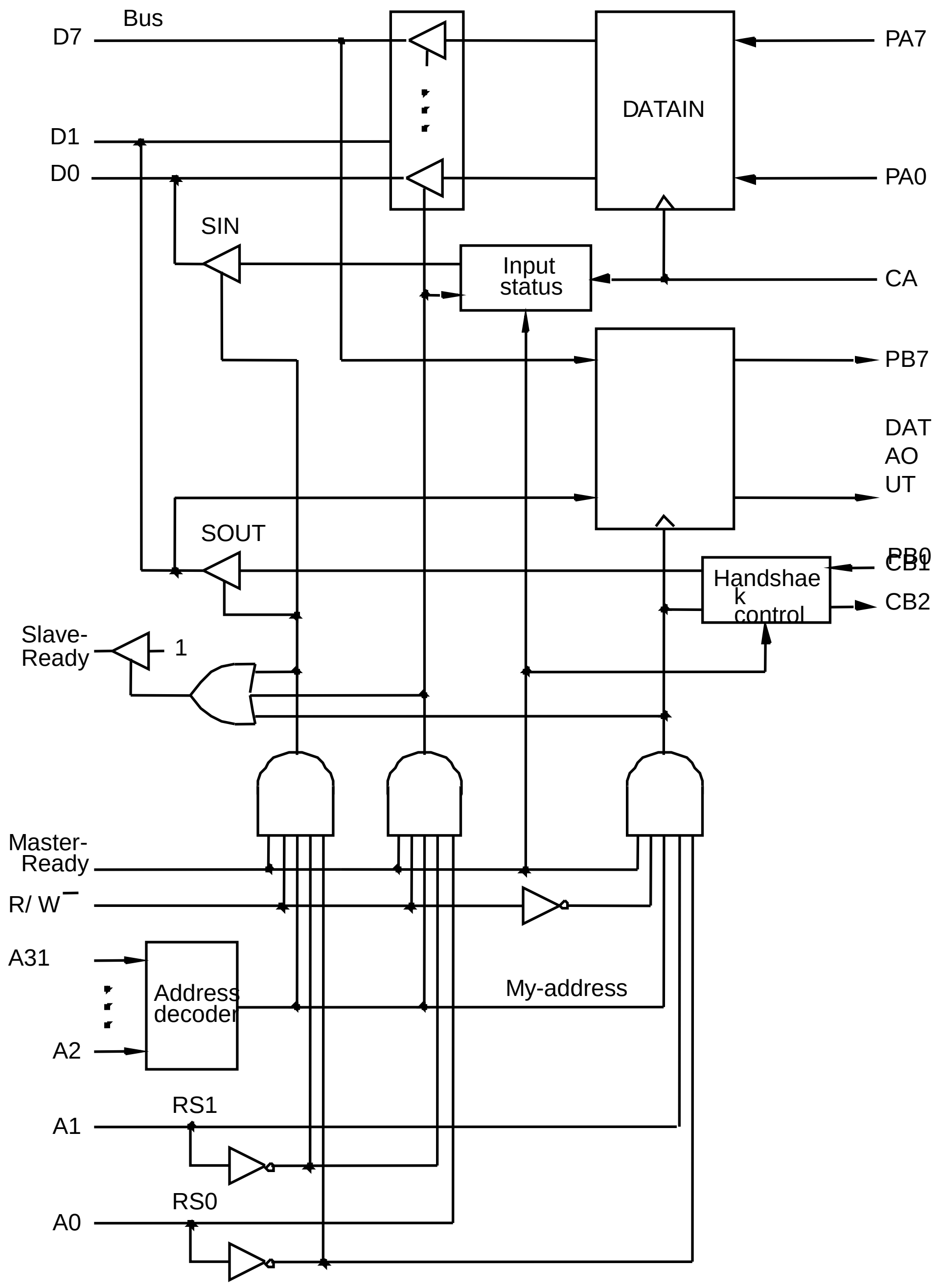


Figure 4.33. Combined input/output interface circuit.

Recall the Timing Protocol

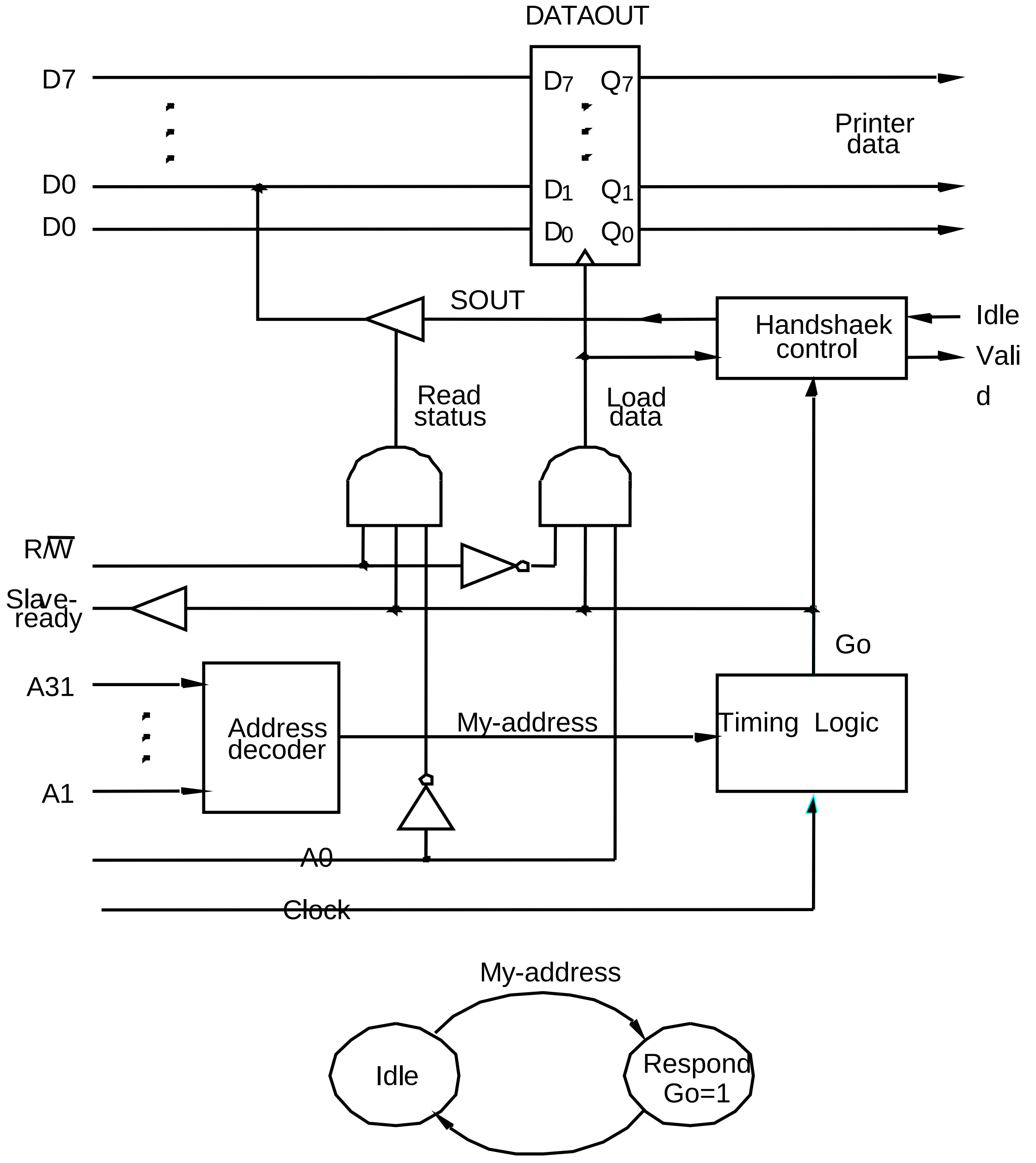


Figure 4.35. A parallel point interface for the bus of Figure 4.25, with a state-diagram for the timing logic.

Serial Port

- A serial port is used to connect the processor to I/O devices that require transmission of data one bit at a time.
- The key feature of an interface circuit for a serial port is that it is capable of communicating in bit-serial fashion on the device side and in a bit-parallel fashion on the bus side.
- Capable of longer distance communication than parallel transmission.

Standard I/O Interfaces

Overview

- The needs for standardized interface signals and protocols.
- Motherboard
- Bridge: circuit to connect two buses
- Expansion bus
- ISA, PCI, SCSI, USB,…

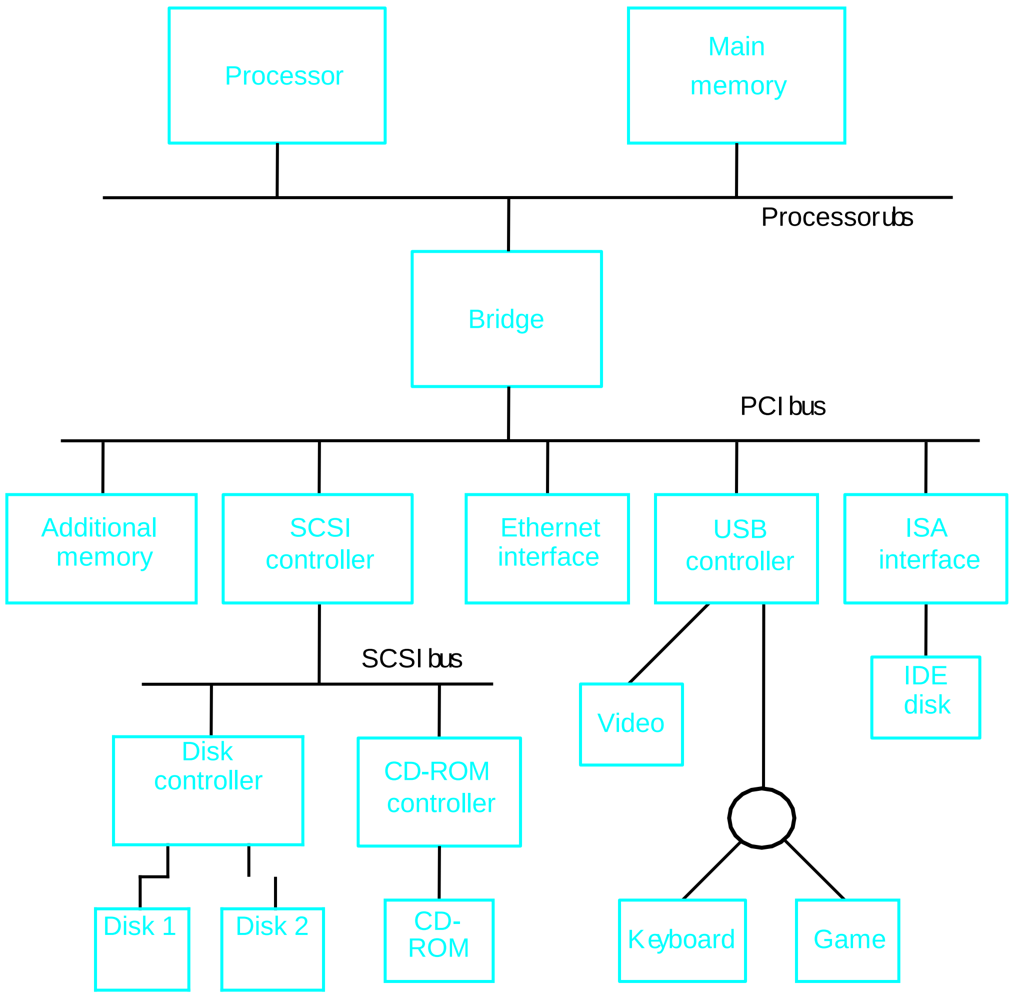


Figure 4.38. An example of a computer system using different interface standards.