Logic Design and Computer Organization

IS234AT

Unit - IV

Dr. Premananda B.S.

Modules

I. Synchronous Sequential Logic

II. Registers, Counters, and Memory

III. A Top-Level View of Computer Function and Interconnection, Cache Memory

IV. Input/ Output and Computer Arithmetic

V. Instruction Sets Characteristics and Functions, Processor Structure and Function, and Parallel Processing

Books

- Computer Organization and Architecture Designing for Performance, William Stallings, 10th Edition, Pearson, ISBN: 978-0134101613

Computer Arithmetic

- Arithmetic and Logic Unit
- Integer Representation
- Integer Arithmetic
  - Addition and Subtraction
  - Multiplication
  - Division
- Floating-point Representation

Arithmetic and Logic Unit (ALU)

- The ALU is that Part of the computer that actually performs arithmetic and logical operations on data.
- All of the other elements of the computer system—control unit, registers, memory, and I/O—are there to bring data into the ALU for it to process and then to take the results back out.
- Based on the use of simple digital logic devices that can store binary digits and perform simple Boolean logic operations.
- Figure 10.1 indicates, how the ALU is interconnected with the rest of the processor.

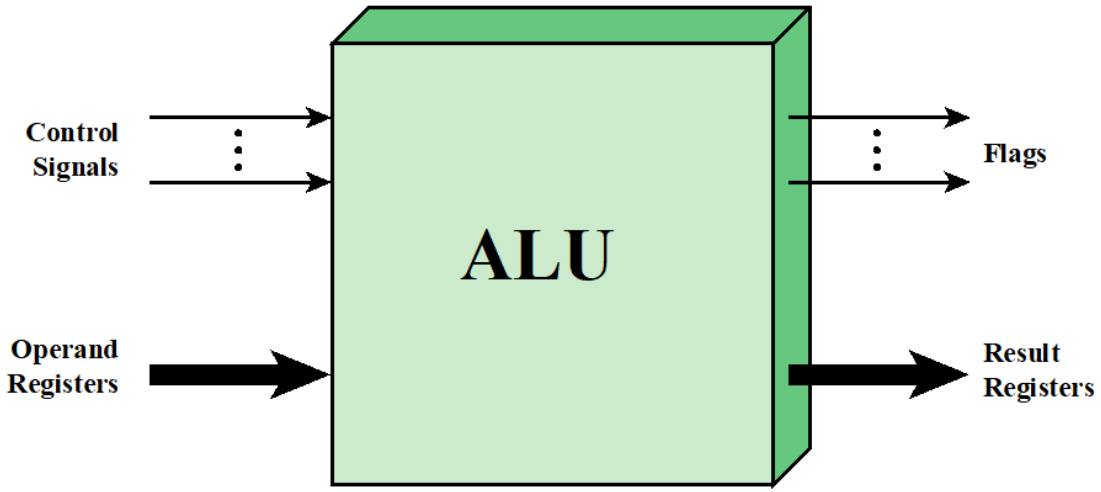


Figure 10.1 ALU Inputs and Outputs

Arithmetic and Logic Unit (ALU)

- Operands for arithmetic and logic operations are given to the ALU in registers, and the results of an operation are stored in registers.
- These registers are temporary storage locations within the processor that are connected by signal paths to the ALU.
- The ALU may also set flags as the result of an operation.
- Example, an overflow flag is set to 1 if the result of a computation exceeds the length of the register into which it is to be stored.
- The flag values are also stored in registers within the processor.
- The processor provides signals that control the operation of the ALU and the movement of the data into and out of the ALU.

Integer Representation

- In the binary number system arbitrary numbers can be represented with:
  - The digits zero and one
  - The minus sign (for negative numbers)
  - The period, or ***radix point*** (for numbers with a fractional component)
- For purposes of computer storage and processing we do not have the benefit of special symbols for the minus sign and radix point.
- Only binary digits (0 and 1) may be used to represent numbers.
- If we limit to non-negative integers, the representation is straightforward.

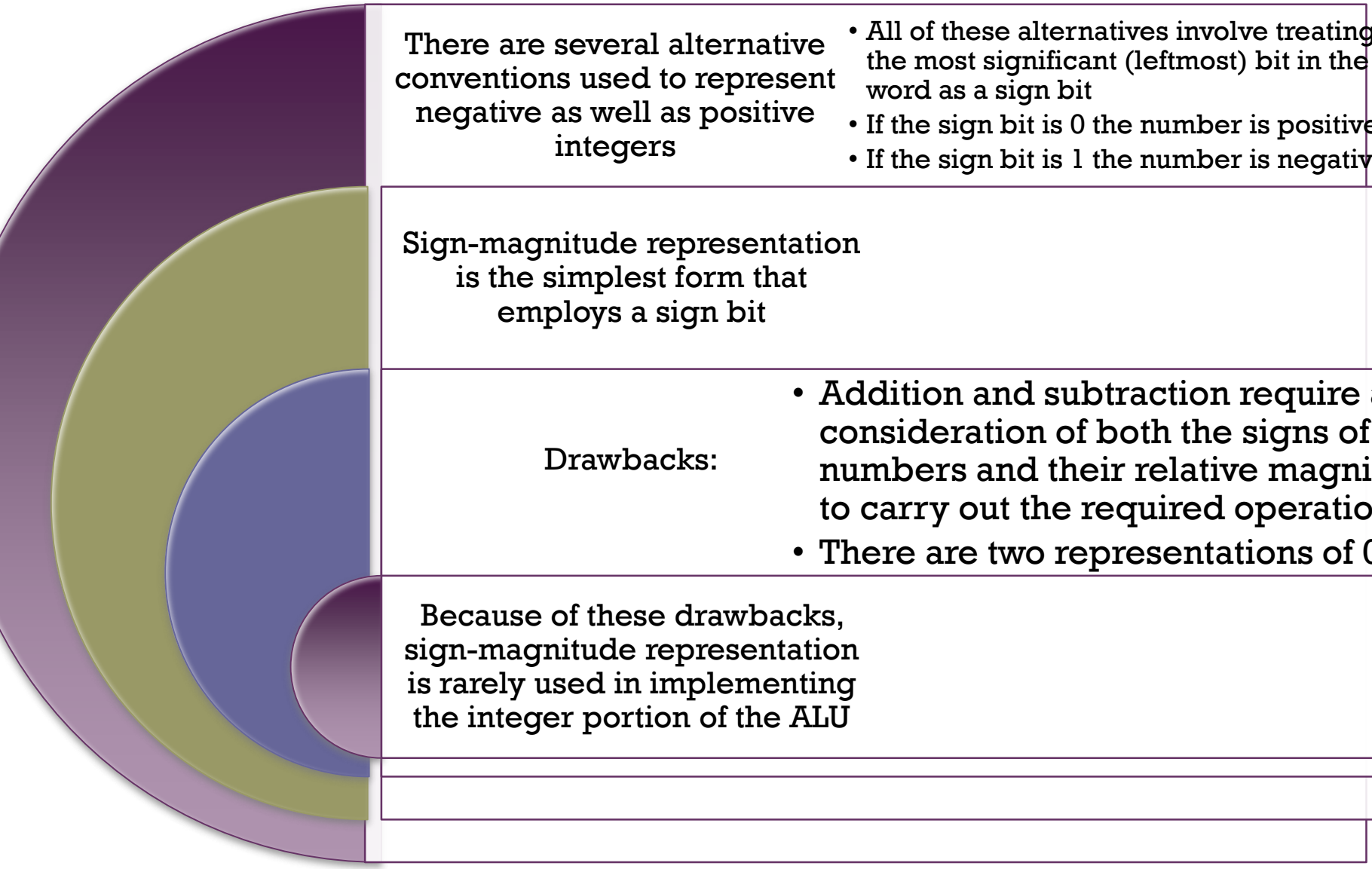
Integer Representation

In general, if an n-bit sequence of binary digits $a_{n-1}a_{n-2} \ldots a_1a_0$ is interpreted as an unsigned integer A, its value is

$$A = \sum_{i=0}^{n-1} 2^i a_i$$

- Sign-Magnitude Representation
- One's Complement Representation
- Two's Complement Representation
- Range Extension

Sign-Magnitude Representation



There are several alternative conventions used to represent negative as well as positive integers

- All of these alternatives involve treating the most significant (leftmost) bit in the word as a sign bit
- If the sign bit is 0 the number is positive
- If the sign bit is 1 the number is negative

Sign-magnitude representation is the simplest form that employs a sign bit

Drawbacks:

- Addition and subtraction require a consideration of both the signs of the numbers and their relative magnitude to carry out the required operation
- There are two representations of 0

Because of these drawbacks, sign-magnitude representation is rarely used in implementing the integer portion of the ALU

Integer Representation:
Sign-Magnitude Representation

- In an *n*-bit word, the rightmost *n* - 1 bits hold the magnitude of the integer.

$$A = \begin{cases} \sum_{i=0}^{n-2} 2^i a_i & \text{if } a_{n-1} = 0 \\ -\sum_{i=0}^{n-2} 2^i a_i & \text{if } a_{n-1} = 1 \end{cases}$$

+18	= 00010010	
−18	= 10010010	(sign magnitude)

Drawbacks

- Addition and subtraction require a consideration of both the signs of the numbers and their relative magnitudes to carry out the required operation.
- There are two representations of 0.
- More difficult to test for 0.

$+0_{10}$	= 00000000	
-0_{10}	= 10000000	(sign magnitude)

Integer Representation:
Two's Complement Representation

- Positive Number

$$A = \sum_{i=0}^{n-2} 2^i a_i \qquad \text{for } A \geq 0$$

- Negative Number

$$A = -2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

Integer Representation:
Two's Complement Representation

Table 10.1 Characteristics of Twos Complement Representation and Arithmetic

Range	-2^{n-1} through $2^{n-1} - 1$
Number of Representations of Zero	One
Negation	Take the Boolean complement of each bit of the corresponding positive number, then add 1 to the resulting bit pattern viewed as an unsigned integer.
Expansion of Bit Length	Add additional bit positions to the left and fill in with the value of the original sign bit.
Overflow Rule	If two numbers with the same sign (both positive or both negative) are added, then overflow occurs if and only if the result has the opposite sign.
Subtraction Rule	To subtract B from A, take the twos complement of B and add it to A.

Table 10.2: Alternative Representations for 4-bit Integers

Decimal Representation	Sign-Magnitude Representation	Twos Complement Representation	Biased Representation
+8	—	—	1111
+7	0111	0111	1110
+6	0110	0110	1101
+5	0101	0101	1100
+4	0100	0100	1011
+3	0011	0011	1010
+2	0010	0010	1001
+1	0001	0001	1000
+0	0000	0000	0111
–0	1000	—	—
–1	1001	1111	0110
–2	1010	1110	0101
–3	1011	1101	0100
–4	1100	1100	0011
–5	1101	1011	0010
–6	1110	1010	0001
–7	1111	1001	0000
–8	—	1000	—

–128	64	32	16	8	4	2	1

(a) An eight-position two's complement value box

–128	64	32	16	8	4	2	1
1	0	0	0	0	0	1	1

–128 +2 +1 = –125

(b) Convert binary 10000011 to decimal

–128	64	32	16	8	4	2	1
1	0	0	0	1	0	0	0

–120 = –128 +8

(c) Convert decimal –120 to binary

Figure 10.2 Use of a Value Box for Conversion Between Twos Complement Binary and Decimal

Integer Representation:
Range Extension

- Range of numbers that can be expressed is extended by increasing the bit length.
- Consider *n*-bit integer and stored in *m* bits, where $m > n$.
- Expansion of bit length is referred to as *range extension*.
- In sign-magnitude notation this is accomplished by moving the sign bit to the new leftmost position and fill in with zeros.

+18	=	00010010	(sign magnitude, 8 bits)
+18	=	0000000000010010	(sign magnitude, 16 bits)
−18	=	10010010	(sign magnitude, 8 bits)
−18	=	1000000000010010	(sign magnitude, 16 bits)

Integer Representation:
Range Extension

- Will not work for twos complement negative integers:
- Rule for twos complement integers is to move the sign bit to the new leftmost position and fill in with copies of the sign bit.
- For positive numbers, fill in with zeros, and for negative numbers, fill in with ones, this is called *sign extension*.

+18	=	00010010	(twos complement, 8 bits)
+18	=	0000000000010010	(twos complement, 16 bits)
−18	=	11101110	(twos complement, 8 bits)
−18	=	1111111111101110	(twos complement, 16 bits)

Fixed-Point Representation

- The representations discussed till now are sometimes referred to as fixed-point.
- The radix point (binary point) is fixed and assumed to be to the right of the rightmost digit.
- The programmer can use the same representation for binary fractions by scaling the numbers so that the binary point is implicitly positioned at some other location.

Computer Arithmetic

- Arithmetic and Logic Unit
- Integer Representation
- **Integer Arithmetic**
  - **Addition and Subtraction**
    - Multiplication
    - Division
- Floating-point Representation

Negation

- Twos complement operation
  1. Take the Boolean complement of each bit of the integer (including the sign bit).
     - Set each 1 to 0 and each 0 to 1
  2. Treating the result as an unsigned binary integer, add 1.

```
                  +18 = 00010010 (twos complement)
bitwise complement = 11101101
                     +        1
                      11101110 = -18
```

- The negative of the negative of that number is itself:

```
                  -18 =  11101110 (twos complement)
bitwise complement =  00010001
                     +        1
                       00010010 = +18
```

Negation Special Case 1

0	=	00000000	(2's complement)
Bitwise complement	=	11111111	
Add 1 to LSB		+ 1	
Result		100000000	= 0

Overflow is ignored

- There is a *carry out* of the most significant bit position, which is ignored.
- The result is that the negation of 0 is 0.

Negation Special Case 2

- Take the negation of the bit pattern of 1 followed by *n - 1* zeros, we get back the same number.

-128	=	10000000	(2's complement)
Bitwise complement	=	01111111	
Add 1 to LSB		+ 1	
Result		10000000	= -128

Monitor MSB (sign bit)

It should change during the negation

Addition and Subtraction

- The addition of 2’s complement is illustrated in Figure next.
- Addition proceeds as if the two numbers were unsigned integers.
- If the result of the operation is positive, we get a positive number in the twos complement form, which is the same as in the unsigned integer form.
- If the result of the operation is negative, we get a negative number in twos complement form.
- In some instances, there is a carry bit beyond the end of the word (indicated by shading), which is ignored.
- The addition of 2’s complement is illustrated in Figure 10.3.

Addition and Subtraction

- In any addition, the result may be larger than the word size used, this condition is called **overflow.**
- When overflow occurs, the ALU must signal this fact so that no attempt is made to use the result.

- OVERFLOW RULE: If two numbers are added, and they are both positive or both negative, then overflow occurs if and only if the result has the *opposite sign*.
  - Figures 10.3e and f show examples of overflow.
  - The overflow can occur whether or not there is a carry.

- SUBTRACTION RULE: To subtract one number (subtrahend) from another (minuend), take the twos complement (negation) of the subtrahend and add it to the minuend.

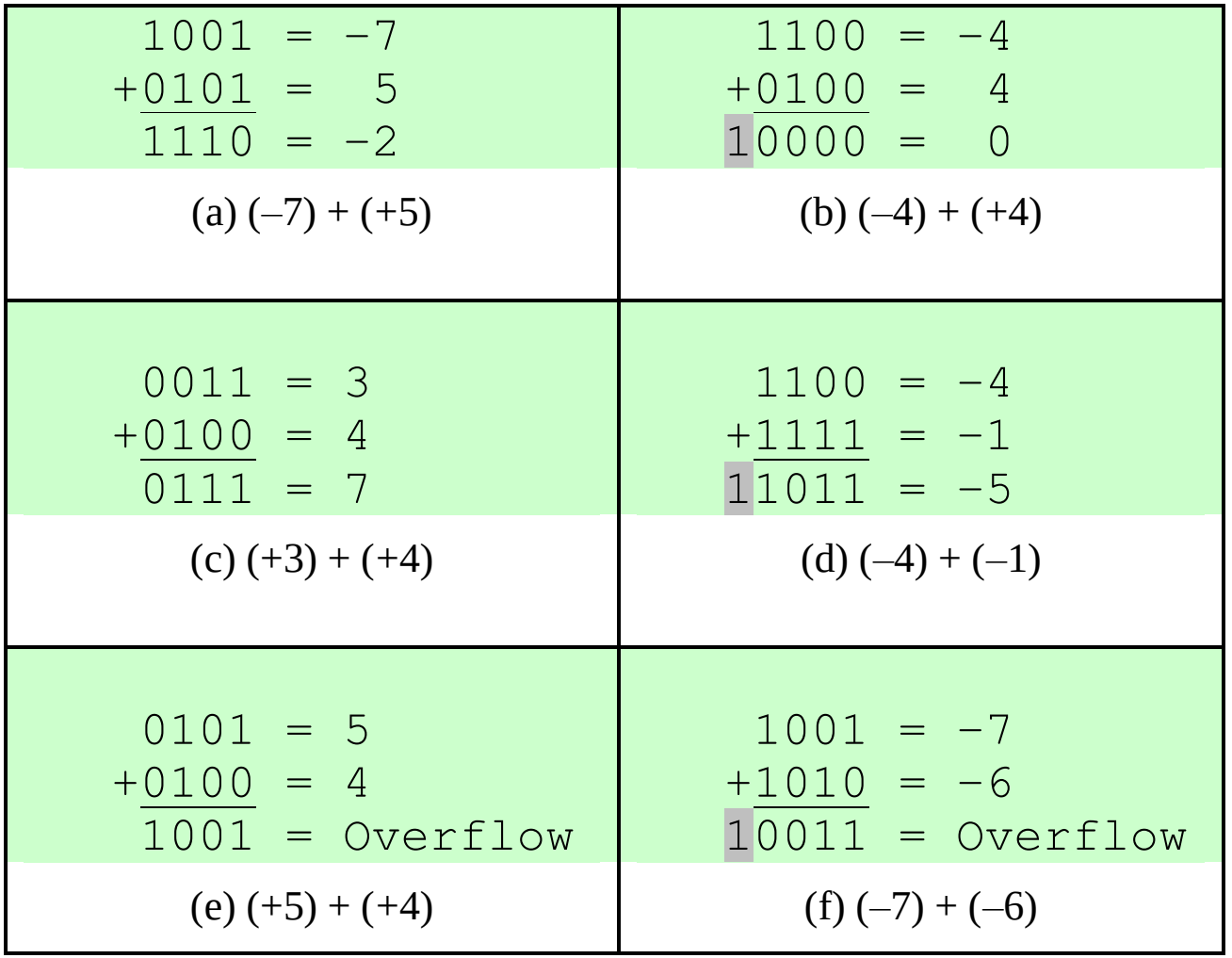


Figure 10.3 Addition of Numbers in Twos Complement Representation

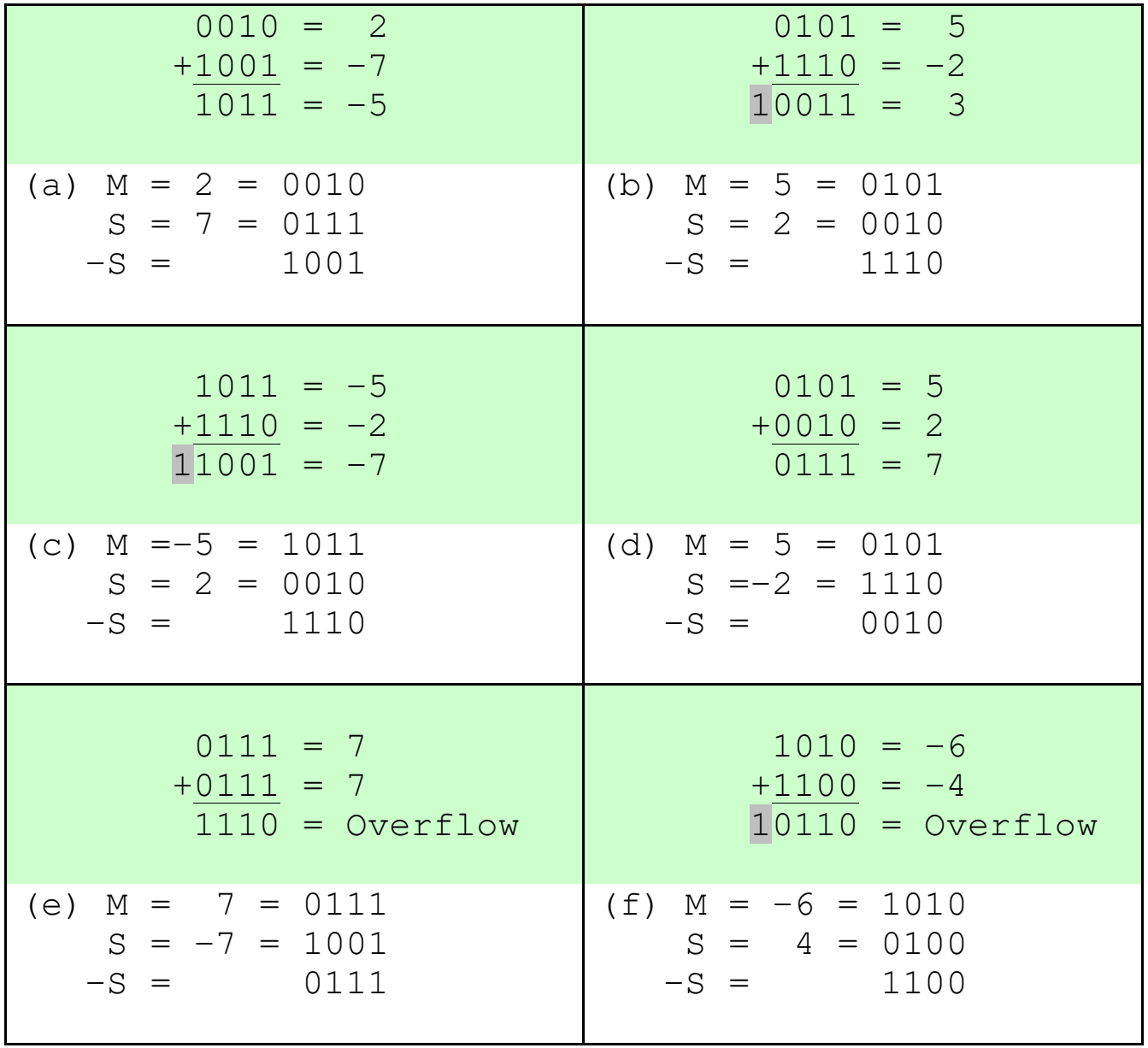


Figure 10.4 Subtraction of Numbers in Twos Complement Representation (M – S)

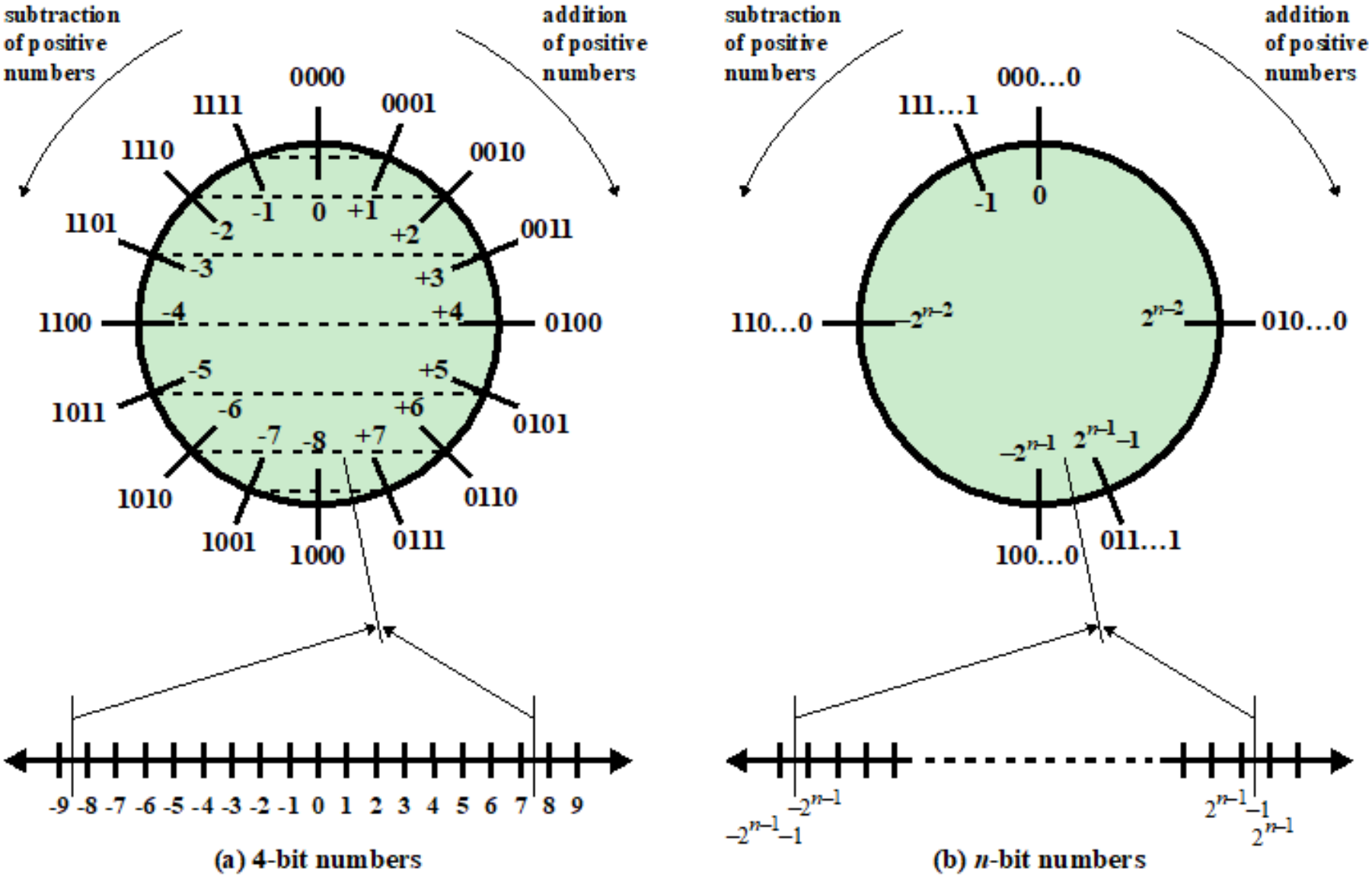
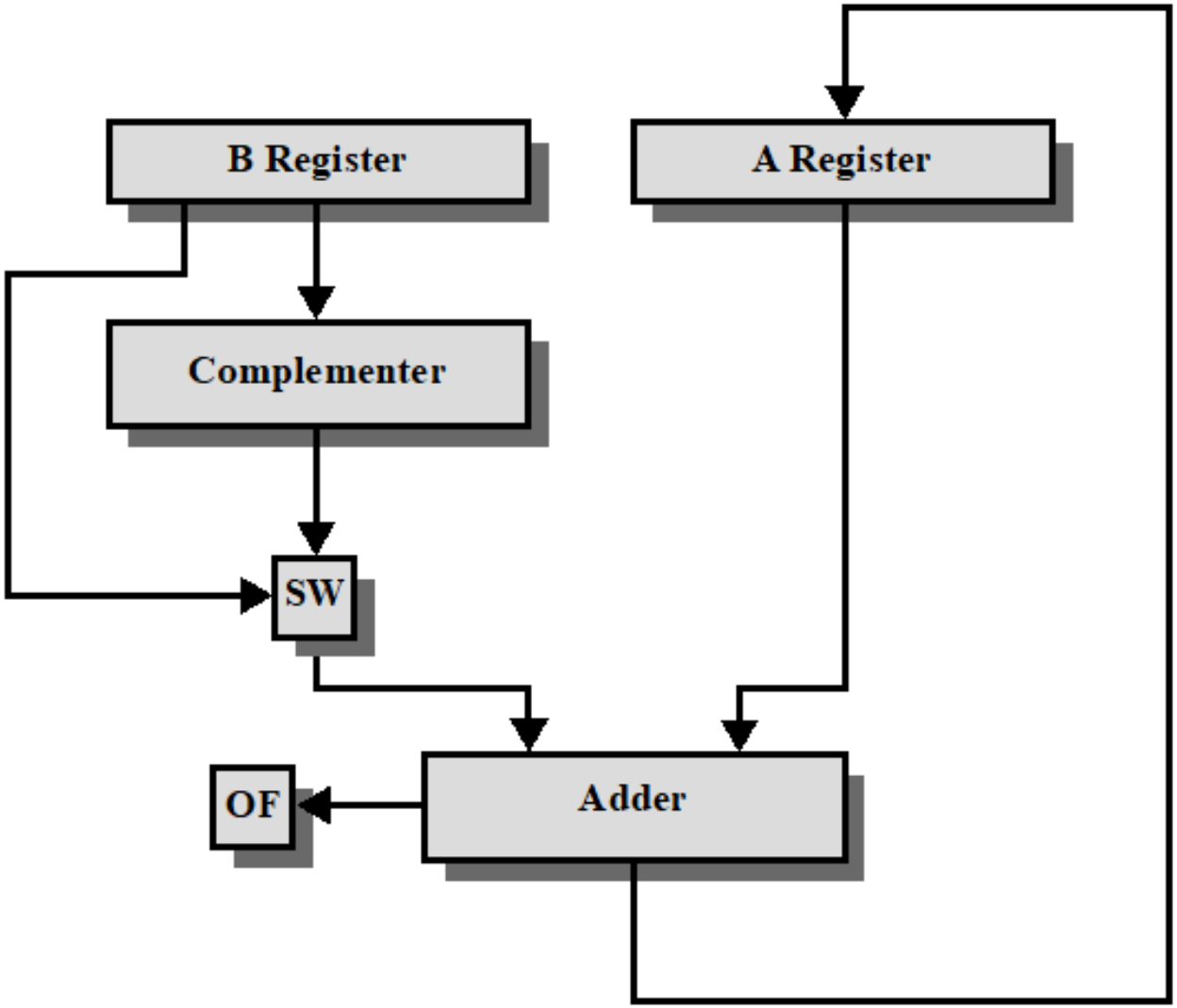


Figure 10.5 Geometric Depiction of Twos Complement Integers

Addition and Subtraction

- Figure 10.6 suggests the data paths and hardware elements needed to perform addition and subtraction.
- A binary adder produces a sum and an overflow indication.
- The binary adder treats the two numbers as unsigned integers.
- In addition, the two numbers are presented to the adder from two registers, designated in this case as A and B registers.
- The result may be stored in one of these registers or in a third.
- The overflow indication is stored in a 1-bit overflow flag: (0 = no overflow; 1 = overflow).
- For subtraction, the subtrahend (B register) is passed through a 2's complementer so that its 2's complement is provided to the adder.
- Control signals are needed to control whether or not the complementer is used, depending on whether the operation is addition or subtraction.



OF = overflow bit
SW = Switch (select addition or subtraction)

Figure 10.6 Block Diagram of Hardware for Addition and Subtraction

Computer Arithmetic

- Arithmetic and Logic Unit
- Integer Representation
- **Integer Arithmetic**
  - Addition and Subtraction
  - **Multiplication**
  - Division
- Floating-point Representation

Multiplication

- Compared with addition and subtraction, multiplication is a complex operation, whether performed in hardware or software.
- A variety of algorithms have been used in various computers.
- Unsigned Integers: Figure 10.7 illustrates the multiplication of unsigned binary integers.

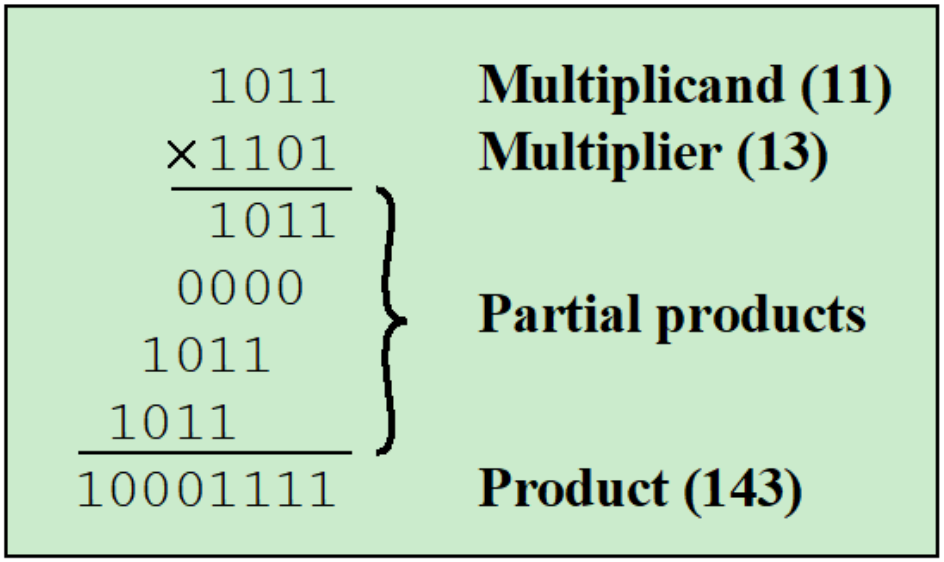


Figure 10.7 Multiplication of Unsigned Binary Integers

Multiplication

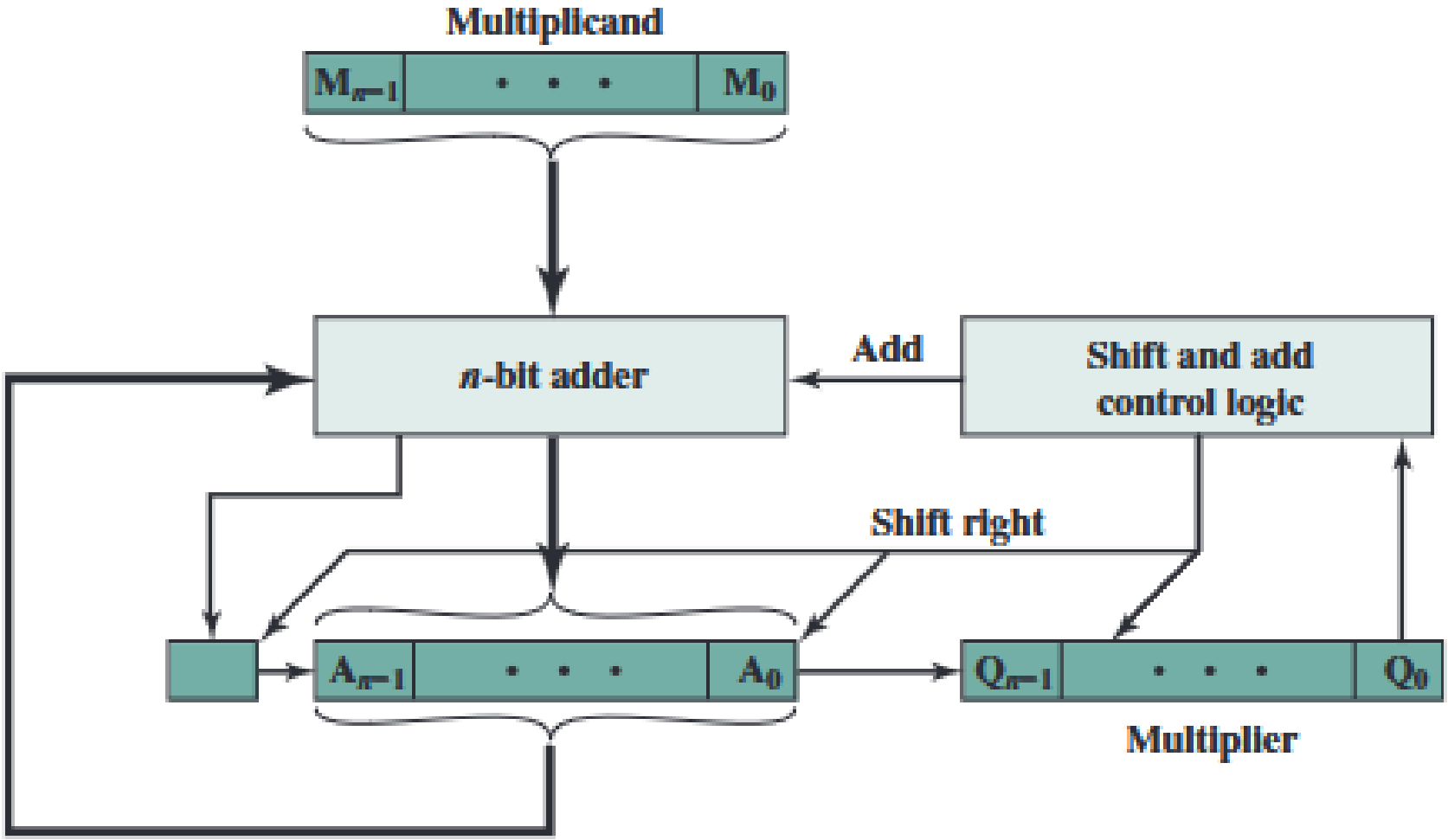
- Several important observations can be made:
  1. Multiplication involves the generation of partial products, one for each digit in the multiplier. These partial products are then summed to produce the final product.
  2. The partial products are easily defined. When the multiplier bit is 0, the partial product is 0. When the multiplier is 1, the partial product is the multiplicand.
  3. The total product is produced by summing the partial products. For this operation, each successive partial product is shifted one position to the left relative to the preceding partial product.
  4. The multiplication of two *n-bit binary integers results in a product of up to 2n* bits in length (e.g., 11 x 11 = 1001).

Multiplication: Making more efficient

- First, perform a running addition on the partial products rather than waiting until the end.
  - This eliminates the need for storage of all the partial products; fewer registers are needed.
- Second, we can save time on the generation of partial products.
  - For each 1 on the multiplier, an add and a shift operation are required; but for each 0, only a shift is required.

- Figure 10.8a shows a possible implementation of unsigned Binary multiplication employing these measures.

Hardware Implementation of Unsigned Binary Multiplication



(a) Block diagram

Figure 10.8 Hardware Implementation of Unsigned Binary Multiplication

Multiplication: Unsigned Binary Multiplication

- The multiplier and multiplicand are loaded into two (Q and M) registers.
- A third register, **A** is initially set to 0.
- A 1-bit **C** register, initialized to 0, which holds a carry bit resulting from addition.
- A flowchart of the operation for unsigned binary multiplication is shown in Figure 10.9.

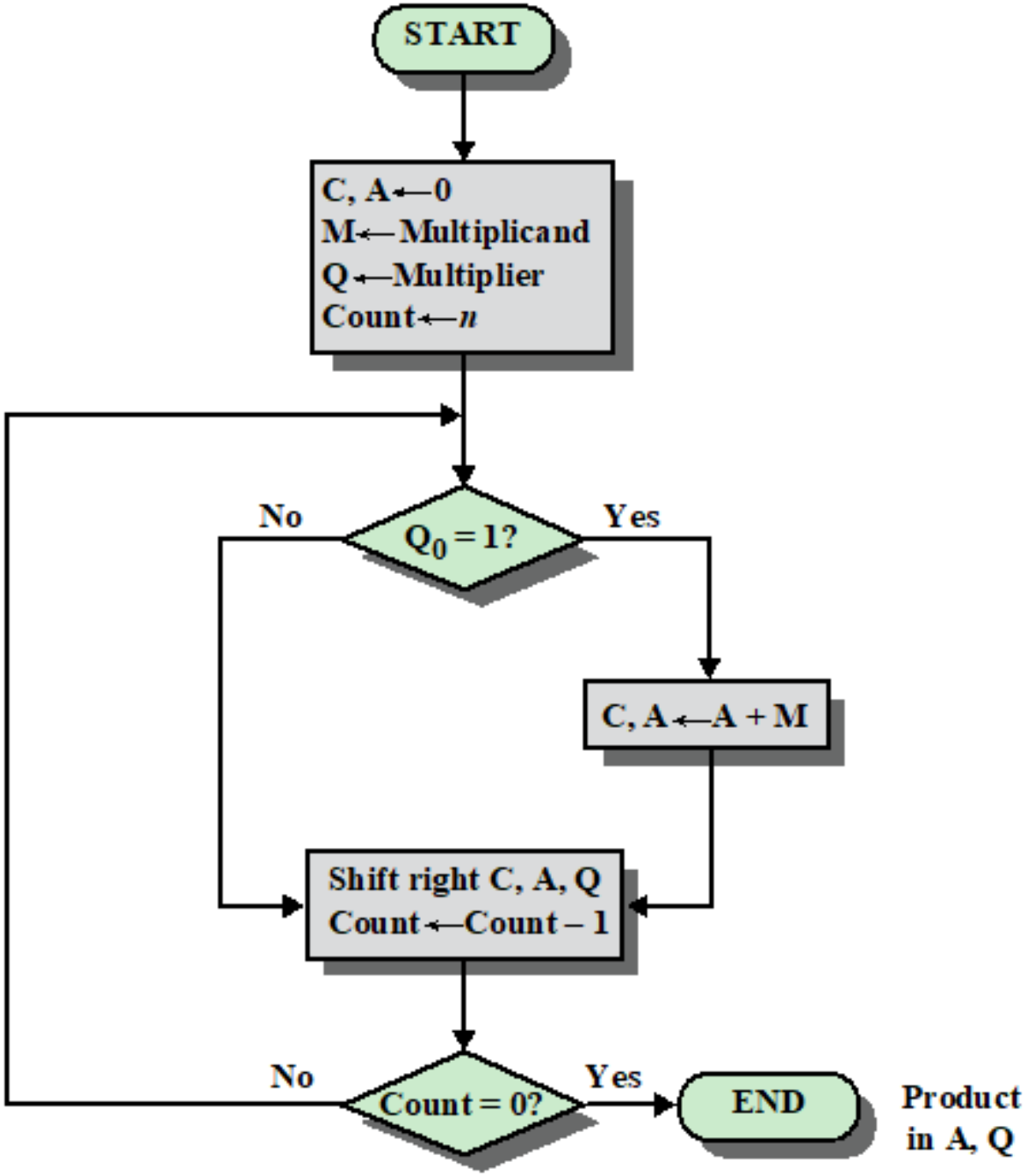


Figure 10.9 Flowchart for Unsigned Binary Multiplication

Multiplication: Unsigned Binary Multiplication

- The operation of the multiplier is as follows.
- Control logic reads the bits of the multiplier one at a time.
- If Q_0 *is 1*, then the multiplicand is added to the A register and the result is stored in the A register, with the C bit used for *overflow*.
- All of the bits of the C, A, and Q registers are shifted to the right one bit so that the C bit goes into A_{n-1}, A_0 *goes into* Q_{n-1} *and* Q_0 *is lost*.
- *If* Q_0 *is 0, then no addition is* performed, just the *shift*.
- This process is repeated for each bit of the original multiplier.
- The resulting 2*n*-bit product is contained in the A and Q registers.

Hardware Implementation of Unsigned Binary Multiplication: Example

multiplication of 11 and 13

```
    1011      Multiplicand (11)
  × 1101      Multiplier (13)
  ------
    1011  }
   0000   }   Partial products
  1011    }
 1011     }
 --------
 10001111     Product (143)
```

C	A	Q	M		
0	0000	1101	1011	**Initial values**	
0	1011	1101	1011	**Add**	**First cycle**
0	0101	1110	1011	**Shift**	
0	0010	1111	1011	**Shift**	**Second cycle**
0	1101	1111	1011	**Add**	**Third cycle**
0	0110	1111	1011	**Shift**	
1	0001	1111	1011	**Add**	**Fourth cycle**
0	1000	1111	1011	**Shift**	

- In Figure 10.10 the partial products are viewed as 2*n*-bit numbers generated from the *n*-bit multiplicand.
- The 4-bit multiplicand 1011 is stored in an 8-bit word as 00001011.
- Each partial product (other than that for 2^0) consists of this number shifted to the left, with the unoccupied positions on the right filled with zeros (e.g., a shift to the left of two places yields 00101100).

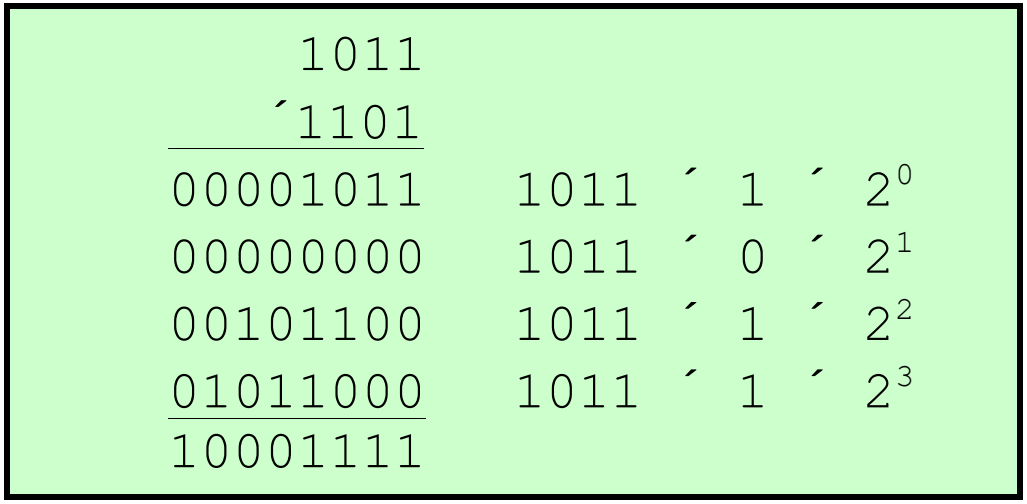


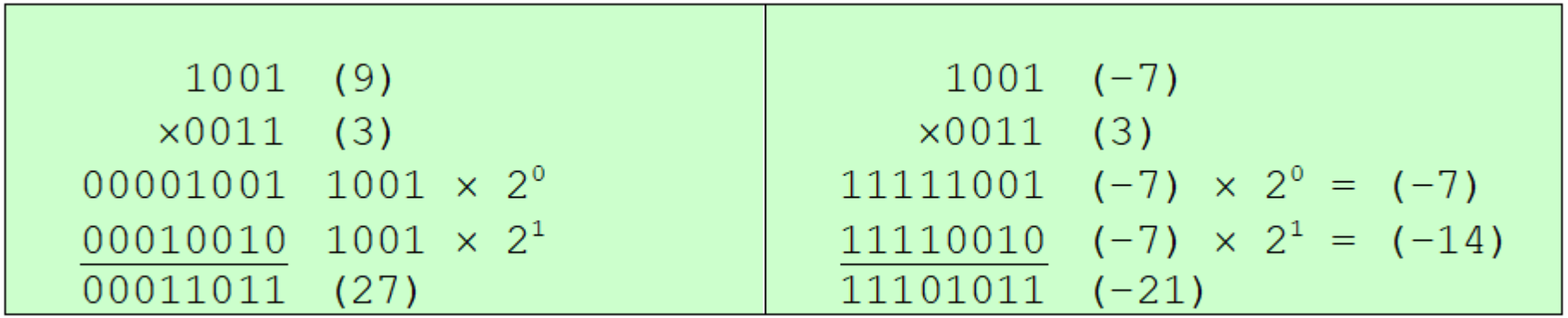
Figure 10.10 Multiplication of Two Unsigned 4-Bit Integers Yielding an 8-Bit Result

Multiplication: Unsigned Binary Multiplication

- The straightforward multiplication will not work if the multiplicand is negative.
- Each contribution of the negative multiplicand as a partial product must be a negative number on a 2*n*-bit field.
- The sign bits of the partial products must line up, as demonstrated in Figure 10.11 next, which shows that multiplication of 1001 by 0011.

Figure 10.11, demonstrates the multiplication of 1001 by 0011

- If treated as unsigned integers, the multiplication of 9 x 3 = 27.
- If 1001 is interpreted as the twos complement value -7, then each partial product must be a negative twos complement number of $2n$ (8) bits, as shown in Figure 10.11b.
- Accomplished by padding out each partial product to the left with binary 1s.



(a) Unsigned integers (b) Twos complement integers

Figure 10.11 Comparison of Multiplication of Unsigned and Twos Complement Integers

Multiplication: Booth's Algorithm

- Booth's algorithm depicted in Figure 10.12 is used to speed up the multiplication process.
- The multiplier and multiplicand are placed in the **Q** and **M** registers.
- The results of the multiplication will appear in the **A** and **Q** registers.
- A 1-bit register is placed to the right of the LSB (Q_0) of the Q register and designated Q_{-1}. A and Q_{-1} are initialized to 0.
- The control logic scans the bits of the multiplier one at a time.
- If the two bits are the same (1–1 or 0–0), then all of the bits of the A, Q, and Q_{-1} registers are shifted to the right 1-bit.
- If the two bits differ, then the multiplicand is added to or subtracted from the A register, depending on whether the two bits are 0–1 or 1–0.
- Then the right shift occurs such that the leftmost bit of A, namely, A_{n-1} *not only is shifted into* A_{n-2}, but also remains in A_{n-1}.
- This is required to preserve the sign of the number in A and Q, it is known as an **arithmetic shift,** because it preserves the sign bit.
- Arithmetic shift preserves the sign bit of the original number.

Booth's Algorithm

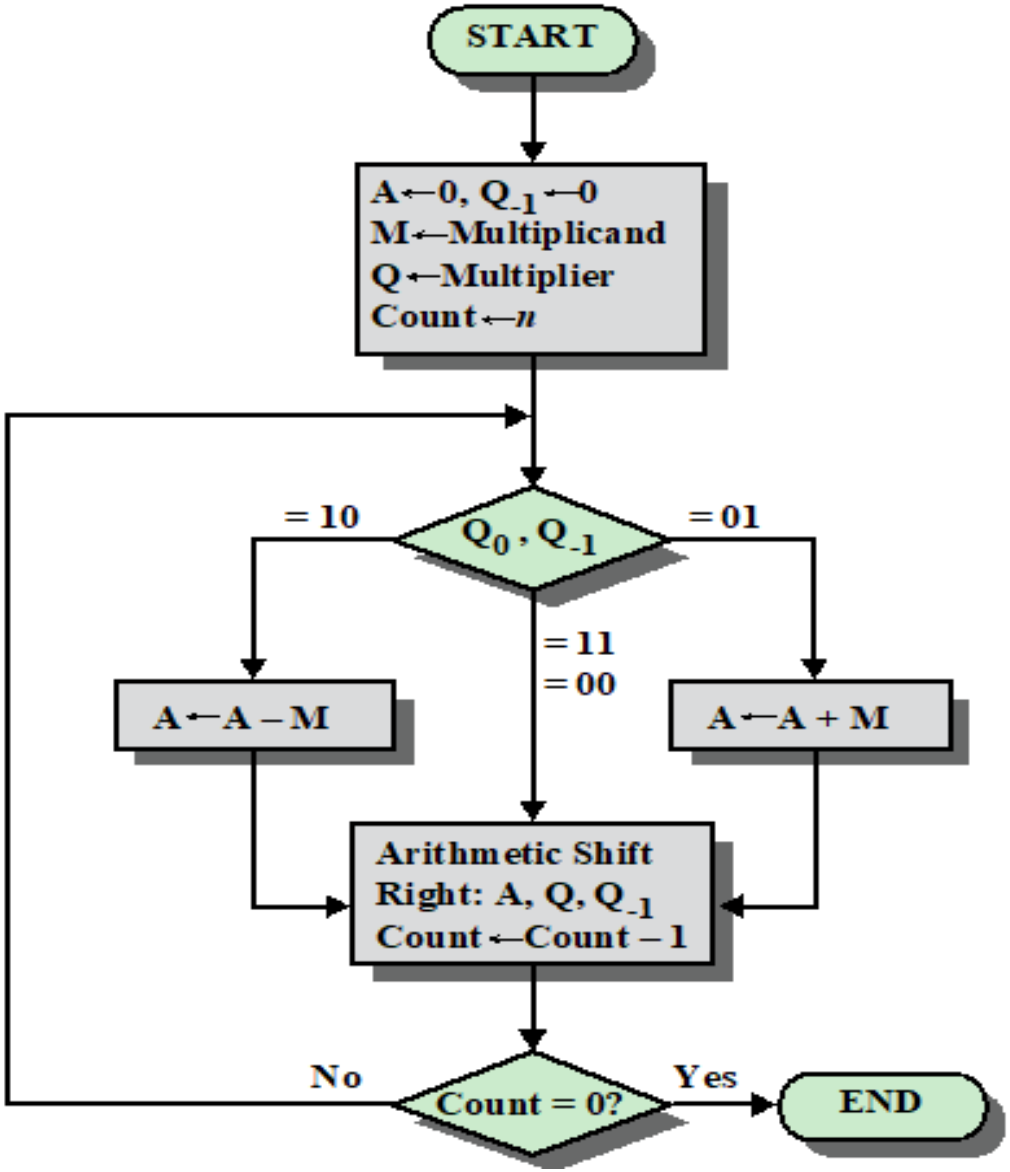


Figure 10.12 Booth's Algorithm for Twos Complement Multiplication

Booth's Algorithm: Example-1

- Figure 10.13 shows the sequence of events in Booth's algorithm for the multiplication of 7 by 3.

A	Q	Q_{-1}	M		
0000	0011	0	0111	Initial Values	
1001	0011	0	0111	$A \leftarrow A - M$	First Cycle
1100	1001	1	0111	Shift	
1110	0100	1	0111	Shift	Second Cycle
0101	0100	1	0111	$A \leftarrow A + M$	Third Cycle
0010	1010	0	0111	Shift	
0001	0101	0	0111	Shift	Fourth Cycle

Figure 10.13 Example of Booth's Algorithm (7× 3)

Figure 10.14 provides other examples of Booth's algorithm

```
     0111
    ×0011      (0)
 11111001      1-0
 0000000       1-1
 000111        0-1
 --------
 00010101      (21)
```

(a) (7) × (3) = (21)

```
     0111
    ×1101      (0)
 11111001      1-0
 0000111       0-1
 111001        1-0
 --------
 11101011      (-21)
```

(b) (7) × (–3) = (–21)

```
     1001
    ×0011      (0)
 00000111      1-0
 0000000       1-1
 111001        0-1
 --------
 11101011      (-21)
```

(c) (–7) × (3) = (–21)

```
     1001
    ×1101      (0)
 00000111      1-0
 1111001       0-1
 000111        1-0
 --------
 00010101      (21)
```

(d) (–7) × (–3) = (21)

Figure 10.14 Examples Using Booth's Algorithm

Booth's Algorithm: Example-2

Perform the multiplication of 7 and 5 using Booth's algorithm.

A	Q	Q–1	M		
0000	0101	0	0111	Initial value	
1001	0101	0	0111	A ← A–M	First cycle
1100	1010	1	0111	shift	
0011	1010	1	0111	A ← A+M	Second cycle
0001	1101	0	0111	shift	
1010	1101	0	0111	A ← A–M	Third cycle
1101	0110	1	0111	shift	
0100	0110	1	0111	A ← A+M	Fourth cycle
0010	0011	0	0111	shift	

Booth's Algorithm: Example-3

Perform the multiplication of 15 and -13 using Booth's algorithm.

A	Q_n	Q_{n+1}	Operation	SC
00000	10011	0		101 (5)
10001	10011	0	A←A – M	
11000	11001	1	Shift	100 (4)
11100	01100	1	Shift	011 (3)
01011	01100	1	A←A + M	
00101	10110	0	Shift	010 (2)
00010	11011	0	Shift	001 (1)
10011	11011	0	A←A – M	
11001	11101	1	Shift	000 (0)
Result = (11001	11101) = – 195(2's complement of +195)			

Booth's Algorithm: Summary

- Booth's algorithm performs:
  - a subtraction when the first 1 is encountered (10),
  - an addition when (01) is encountered,
  - finally, another subtraction when the first 1 of the next block of 1s is encountered.
- Booth's algorithm performs fewer additions and subtractions than a more straightforward algorithm.

Computer Arithmetic

- Arithmetic and Logic Unit
- Integer Representation
- **Integer Arithmetic**
  - Addition and Subtraction
  - Multiplication
  - **Division**
- Floating-point Representation

Division

- Division operation involves repetitive shifting and addition or subtraction.
- Figure 10.15 shows the division of unsigned binary integers.
- First, dividend bits are examined from left to right, until a number greater than or equal to the divisor; this is referred to as the divisor being able to divide the number.
- Until event occurs, 0s are placed in the quotient from left to right.
- When the event occurs, a 1 is placed in the quotient and the divisor is subtracted from the partial dividend.
- The result is referred to as a *partial remainder.*
- At each cycle, additional bits from the dividend are appended to the partial remainder until the result is greater than or equal to the divisor.
- The divisor is subtracted from this number to produce a new partial remainder.

- Figure 10.16 shows a machine algorithm that corresponds to the long division process.
- The divisor is placed in the M register, and the dividend is in the Q register.

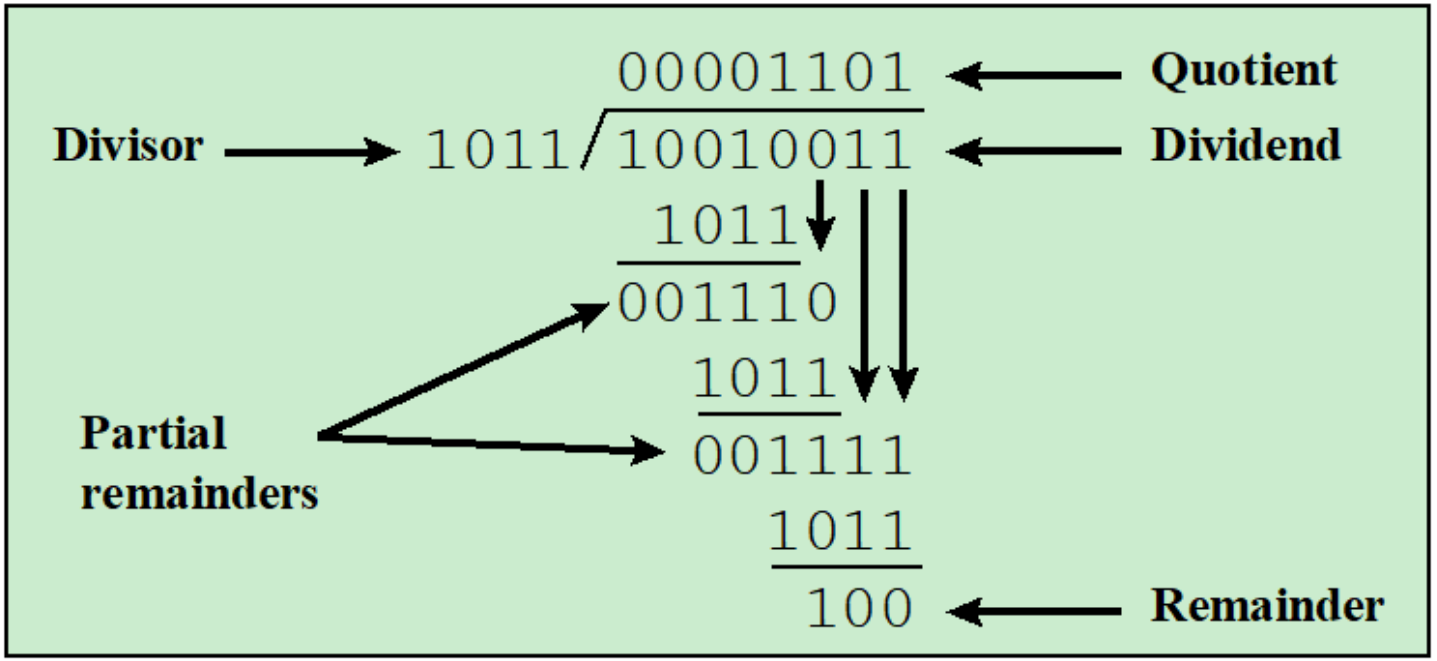
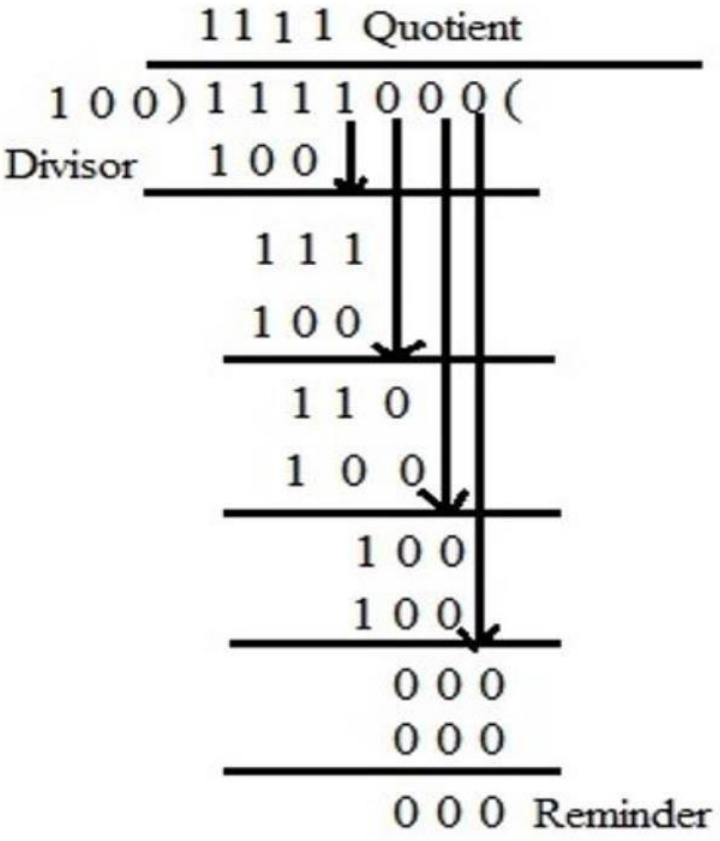
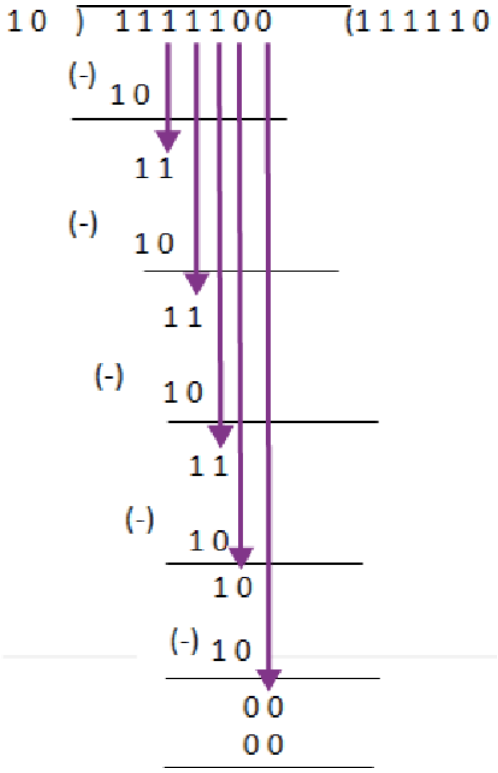
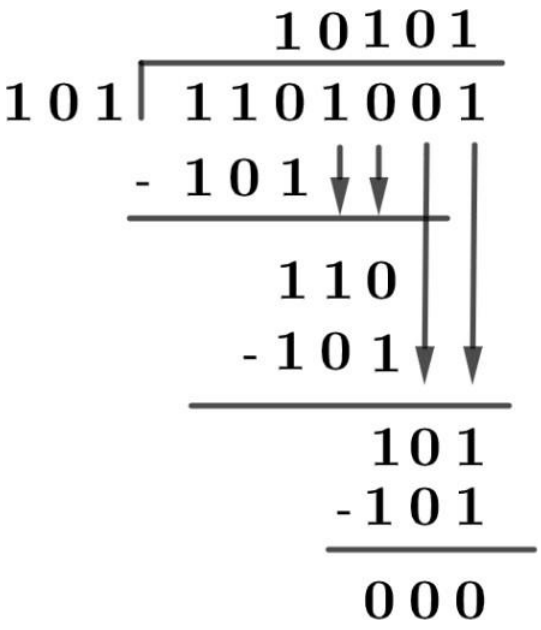


Figure 10.15 Example of Division of Unsigned Binary Integers

Division: Examples







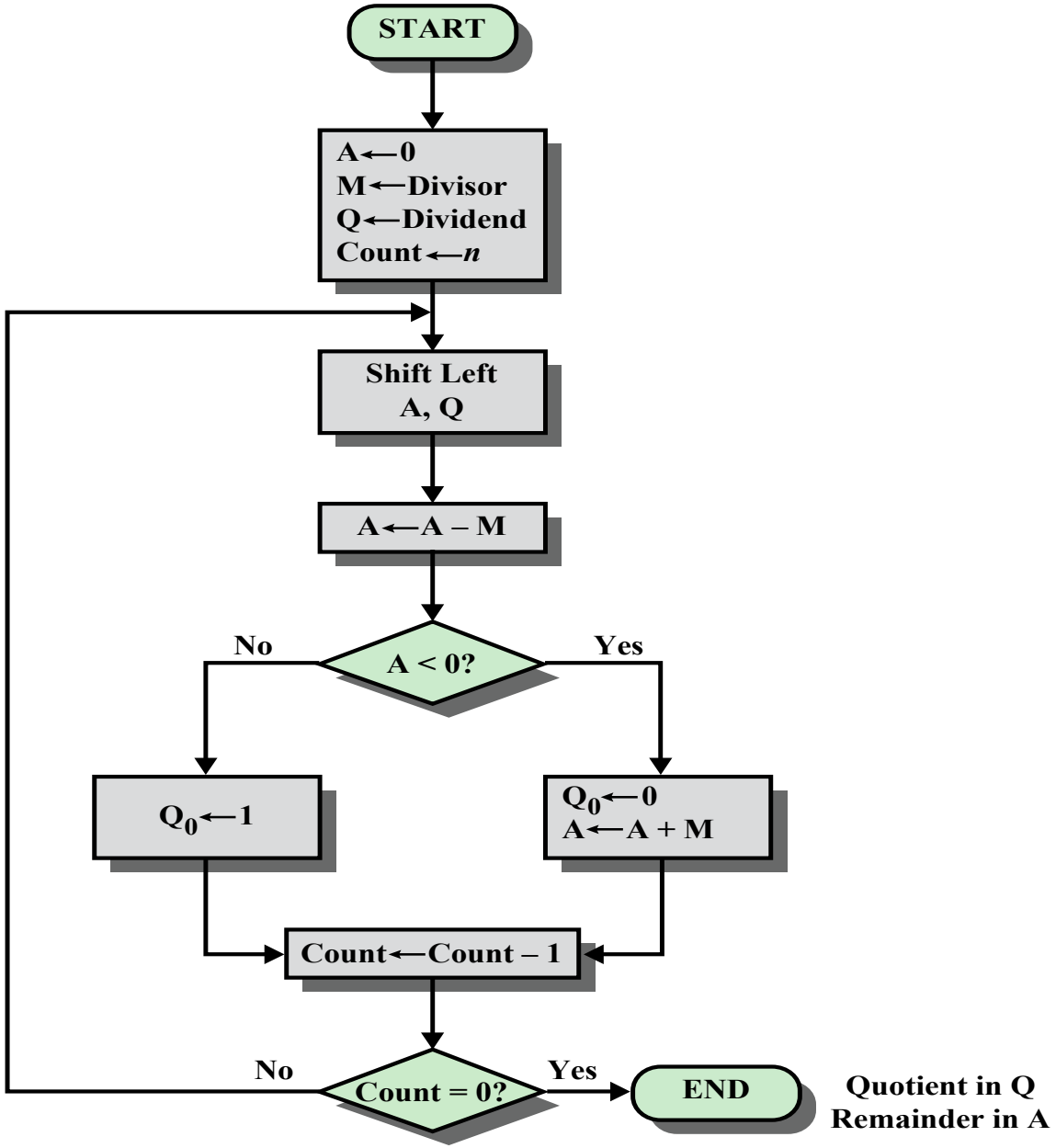


Figure 10.16 Flowchart for Unsigned Binary Division

Division: Examples

A	Q	
0000	0111	Initial value
0000 1101 1101 0000	1110 1110	Shift Use twos complement of 0011 for subtraction Subtract Restore, set Q_0 = 0
0001 1101 1110 0001	1100 1100	Shift Subtract Restore, set Q_0 = 0
0011 1101 0000	1000 1001	Shift Subtract, set Q_0 = 1
0001 1101 1110 0001	0010 0010	Shift Subtract Restore, set Q_0 = 0

Figure 10.17 Example of Restoring Twos Complement Division (7/3)

Division: Examples

The algorithm assumes that the divisor V and the dividend D are positive and that $|V| < |D|$. If $|V| = |D|$, then the quotient $Q = 1$ and the remainder $R = 0$. If $|V| > |D|$, then $Q = 0$ and $R = D$. The algorithm can be summarized as follows:

1. Load the twos complement of the divisor into the M register; that is, the M register contains the negative of the divisor. Load the dividend into the A, Q registers. The dividend must be expressed as a $2n$-bit positive number. Thus, for example, the 4-bit 0111 becomes 00000111.
2. Shift A, Q left 1 bit position.
3. Perform A $\leftarrow$ A $-$ M. This operation subtracts the divisor from the contents of A.
4. a. If the result is nonnegative (most significant bit of A $= 0$), then set $Q_0 \leftarrow 1$.
   b. If the result is negative (most significant bit of A $= 1$), then set $Q_0 \leftarrow 0$. and restore the previous value of A.
5. Repeat steps 2 through 4 as many times as there are bit positions in Q.
6. The remainder is in A and the quotient is in Q.

Computer Arithmetic

- Arithmetic and Logic Unit
- Integer Representation
- Integer Arithmetic
  - Addition and Subtraction
  - Multiplication
  - Division
- **Floating-point Representation**

Floating-Point Representation

- With a fixed-point notation, it is possible to represent a range of positive and negative integers centered on or near 0.
- By assuming a fixed binary or radix point, this format allows the representation of numbers with a fractional component as well.
- Limitations:
  - Very large numbers cannot be represented nor can very small fractions.
  - The fractional part of the quotient in a division of two large numbers could be lost.

- To simplify floating-point operations numbers need to be normalized
- A normal number is one in which the most significant digit of the significand is nonzero.
- For base 2 representation, a normal number has most significant bit of the significand as one.
- Convention/Assumption→ One bit to the left of the radix point.
- Represented $\pm 1.bbb \ldots b \times 2^{\pm E}$

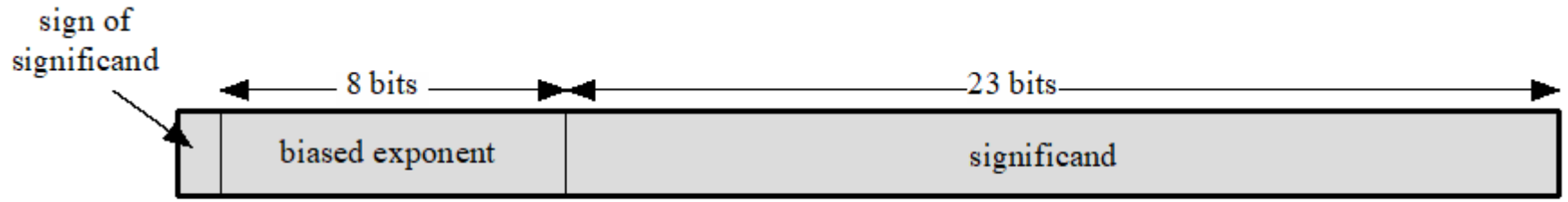
- where b is either binary digit (0 or 1)
- The most significant bit is always one, it is unnecessary to store this bit rather, it is implicit.
- Number may be normalized by shifting the radix point to the right of the leftmost 1 bit and adjusting the exponent accordingly.

The number is in the form

$$\pm S \times B^{\pm E}$$

This number can be stored in a binary word with three fields:

- Sign: plus or minus
- Significand S
- Exponent E



(a) Format

$1.1010001 \times 2^{10100} = 0\ 10010011\ 10100010000000000000000 = 1.6328125 \times 2^{20}$

$-1.1010001 \times 2^{10100} = 1\ 10010011\ 10100010000000000000000 = -1.6328125 \times 2^{20}$

$1.1010001 \times 2^{-10100} = 0\ 01101011\ 10100010000000000000000 = 1.6328125 \times 2^{-20}$

$-1.1010001 \times 2^{-10100} = 1\ 01101011\ 10100010000000000000000 = -1.6328125 \times 2^{-20}$

(b) Examples

Figure 10.18 Typical 32-Bit Floating-Point Format

Table 10.2

Alternative Representations for 4-bit Integers

Decimal Representation	Sign-Magnitude Representation	Twos Complement Representation	Biased Representation
+8	—	—	1111
+7	0111	0111	1110
+6	0110	0110	1101
+5	0101	0101	1100
+4	0100	0100	1011
+3	0011	0011	1010
+2	0010	0010	1001
+1	0001	0001	1000
+0	0000	0000	0111
–0	1000	—	—
–1	1001	1111	0110
–2	1010	1110	0101
–3	1011	1101	0100
–4	1100	1100	0011
–5	1101	1011	0010
–6	1110	1010	0001
–7	1111	1001	0000
–8	—	1000	—

Binary Number Fields

- The leftmost bit stores the sign of the number (0 = positive, 1 = negative)
- Exponent in the next 8 bits(Biased representation)
- Bias equals $(2^{k-1} - 1)$
  - k -bits in exponent
  - 8-bit [0 through 255, -127 to + 128, Bias=127]
- Advantage of biased representation
  - Nonnegative floating-point numbers can be treated as integers for comparison purposes.
  - When the bits of a biased representation are treated as unsigned integers, the relative magnitudes of the numbers do not change
  - Example, in biased and unsigned representations, the largest number is 1111 and the smallest number is 0000

Floating-Point

- The final portion of the word is the significand.
- Any floating-point number can be expressed in many ways

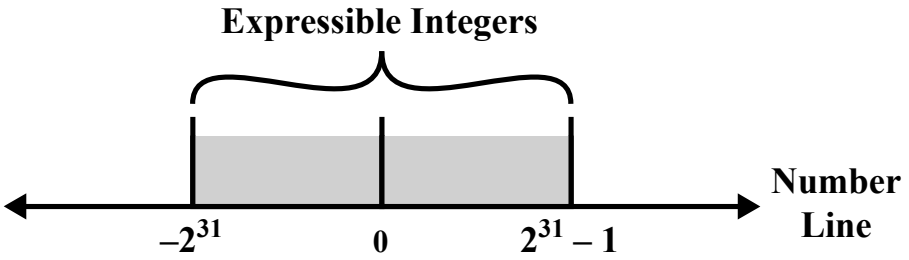
  The following are equivalent, where the significand is expressed in binary form:

  $0.110 * 2^5$

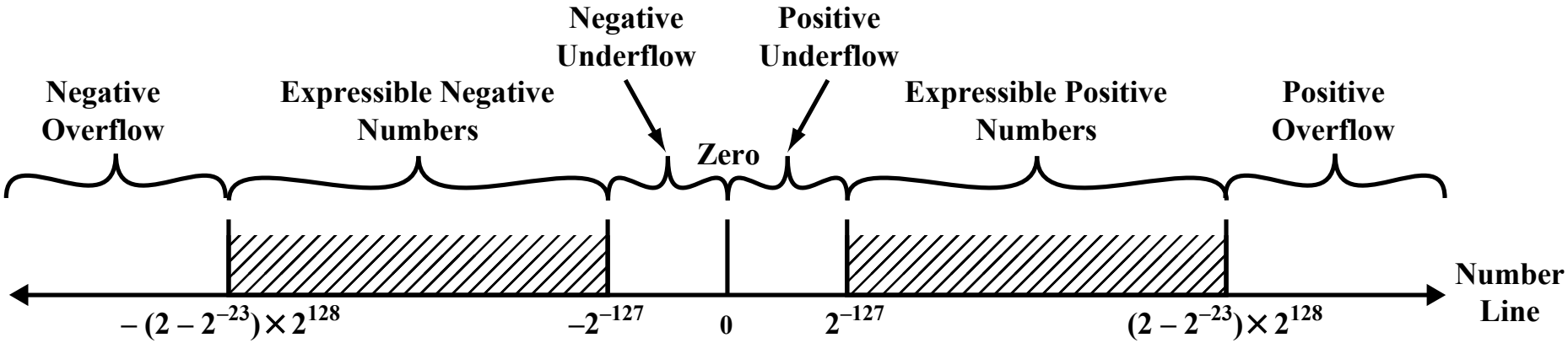
  $110 * 2^2$

  $0.0110 * 2^6$

- *Normal number*
  - The most significant digit of the significand is nonzero



(a) Twos Complement Integers



(b) Floating-Point Numbers

Figure 10.19 Expressible Numbers in Typical 32-Bit Formats

- Negative numbers between $-(2 - 2^{-23}) \times 2^{128}$ and -2^{-127}
- Positive numbers between 2^{-127} and $(2 - 2^{-23}) \times 2^{128}$

Five regions on the number line are not included in these ranges:

- Negative numbers less than $-(2 - 2^{-23}) \times 2^{128}$, called **negative overflow**
- Negative numbers greater than 2^{-127}, called **negative underflow**
- Zero
- Positive numbers less than 2^{-127}, called **positive underflow**
- Positive numbers greater than $(2 - 2^{-23}) \times 2^{128}$, called **positive overflow**

- The numbers represented in floating-point notation are not spaced evenly along the number line, as are fixed-point numbers.
- The possible values get closer together near the origin and farther apart as you move away, as shown in Figure 10.20.

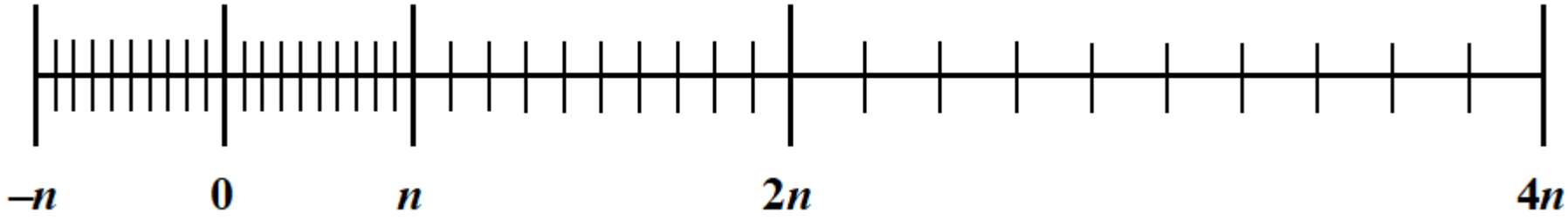


Figure 10.20 Density of Floating-Point Numbers

IEEE Standard 754

Most important floating-point representation is defined

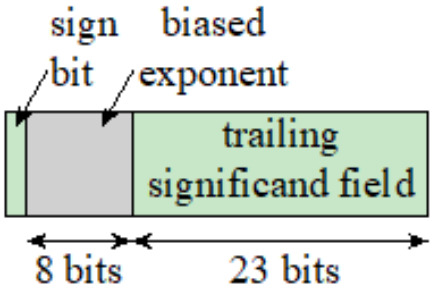
Standard was developed to facilitate the portability of programs from one processor to another and to encourage the development of sophisticated, numerically oriented programs

Standard has been widely adopted and is used on virtually all contemporary processors and arithmetic coprocessors

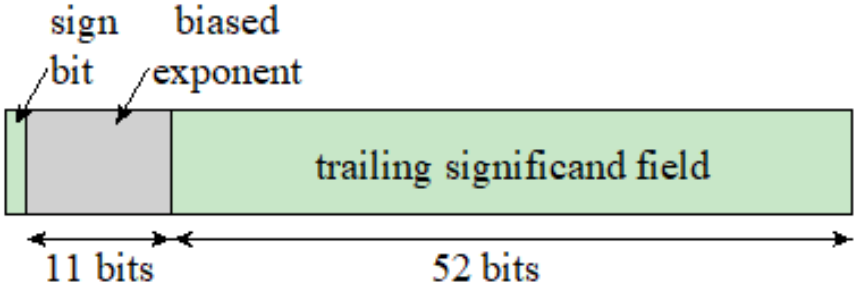
IEEE 754-2008 covers both binary and decimal floating-point representations

IEEE 754-2008

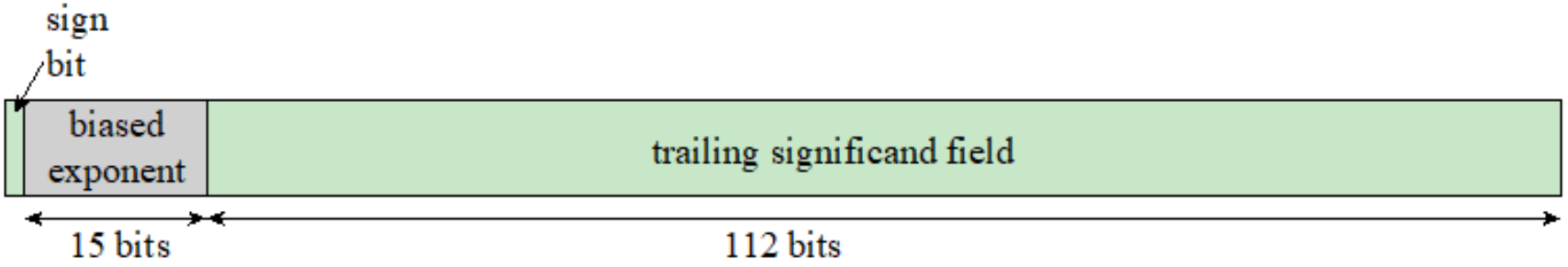
- Defines the following different types of floating-point formats:
- Arithmetic format
  - All the mandatory operations defined by the standard are supported by the format. The format may be used to represent floating-point operands or results for the operations described in the standard.
- Basic format
  - This format covers five floating-point representations, three binary and two decimal, whose encodings are specified by the standard, and which can be used for arithmetic. At least one of the basic formats is implemented in any conforming implementation.
- Interchange format
  - A fully specified, fixed-length binary encoding that allows data interchange between different platforms and that can be used for storage.



(a) binary32 format



(b) binary64 format



(c) binary128 format

Figure 10.21 IEEE 754 Formats

Table 10.4 shows the relationship between defined formats and format types.

Table 10.4

Format	Format Type		
	Arithmetic Format	Basic Format	Interchange Format
binary16			X
binary32	X	X	X
binary64	X	X	X
binary128	X	X	X
binary{*k*} (*k* = *n* ´ 32 for *n* > 4)	X		X
decimal64	X	X	X
decimal128	X	X	X
decimal{*k*} (*k* = *n* ´ 32 for *n* > 4)	X		X
extended precision	X		
extendable precision	X		

Summary

- ALU
- Integer representation
  - Sign-magnitude representation
  - Twos complement representation
  - Range extension
  - Fixed-point representation
- Floating-point representation
  - Principles
  - IEEE standard for binary floating-point representation
- Integer arithmetic
  - Negation
  - Addition and subtraction
  - Multiplication
  - Division

Input/ Output

- External Devices
- I/O Modules
- Programmed I/O
- Interrupt-Driven I/O
- Direct Memory Access
- I/O Channels and Processors
- External Interconnection Standards

Input/ Output (I/O)

- Input/ Output (I/O) module interfaces to the system bus or central switch and controls one or more peripheral devices.
- An I/O module contains logic for performing a communication function between the peripheral and the bus.
- One should not connect peripherals directly to the system bus, because:
  - There are a variety of peripherals with various methods of operation. It is impractical to incorporate the necessary logic within the processor to control a range of devices.
  - The data transfer rate of peripherals is often much slower than that of the memory or processor. It is impractical to use the high-speed system bus to communicate directly with a peripheral.
  - The data transfer rate of some peripherals is faster than that of the memory or processor. The mismatch would lead to inefficiencies if not managed properly.
  - Peripherals often use different data formats and word lengths than the computer to which they are attached.

An I/O module has two major functions as shown in Figure 7.1:

- Interface to the processor and memory via the system bus or central switch.
- Interface to one or more peripheral devices by tailored data links.

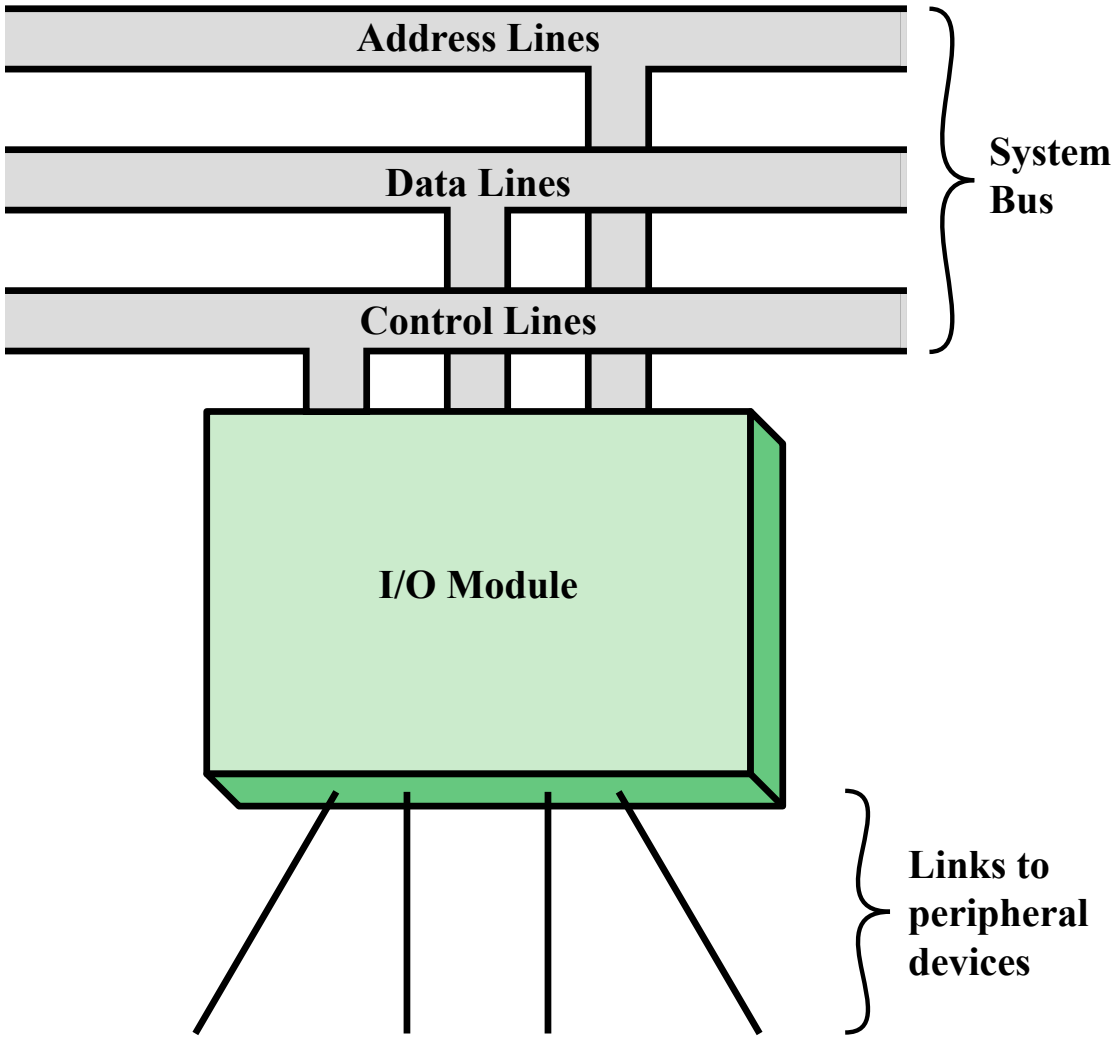


Figure 7.1 Generic Model of an I/O Module

External Devices

- Provide a means of exchanging data between external environment and the computer
- Attach to the computer by a link to an I/O module
  - The link is used to exchange control, status, and data between the I/O module and the external device
- *Peripheral device*
  - An external device connected to an I/O module

Three categories:

- Human readable
  - Suitable for communicating with the computer user
  - Video display terminals (VDTs), printers
- Machine readable
  - Suitable for communicating with equipment
  - Magnetic disk and tape systems, sensors and actuators
- Communication
  - Suitable for communicating with remote devices such as a terminal, a machine readable device, or another computer

External Devices

- The nature of an external device is indicated in Figure 7.2, the interface to the I/O module is in the form of control, data, and status signals.
- *Control signals* determine the function that the device will perform, such as send data to the I/O module (INPUT or READ), accept data from the I/O module (OUTPUT or WRITE), report status, or perform some control function particular to the device (e.g., position a disk head).
- Data are in the form of a set of bits.
- *Status signals* indicate the state of the device.
  - Examples are READY/NOT-READY to show whether the device is ready for data transfer.
- *Control logic* associated with the device controls the device's operation in response to direction from the I/O module.
- The *transducer* converts data from electrical to other forms of energy during output and from other forms to electrical during input.

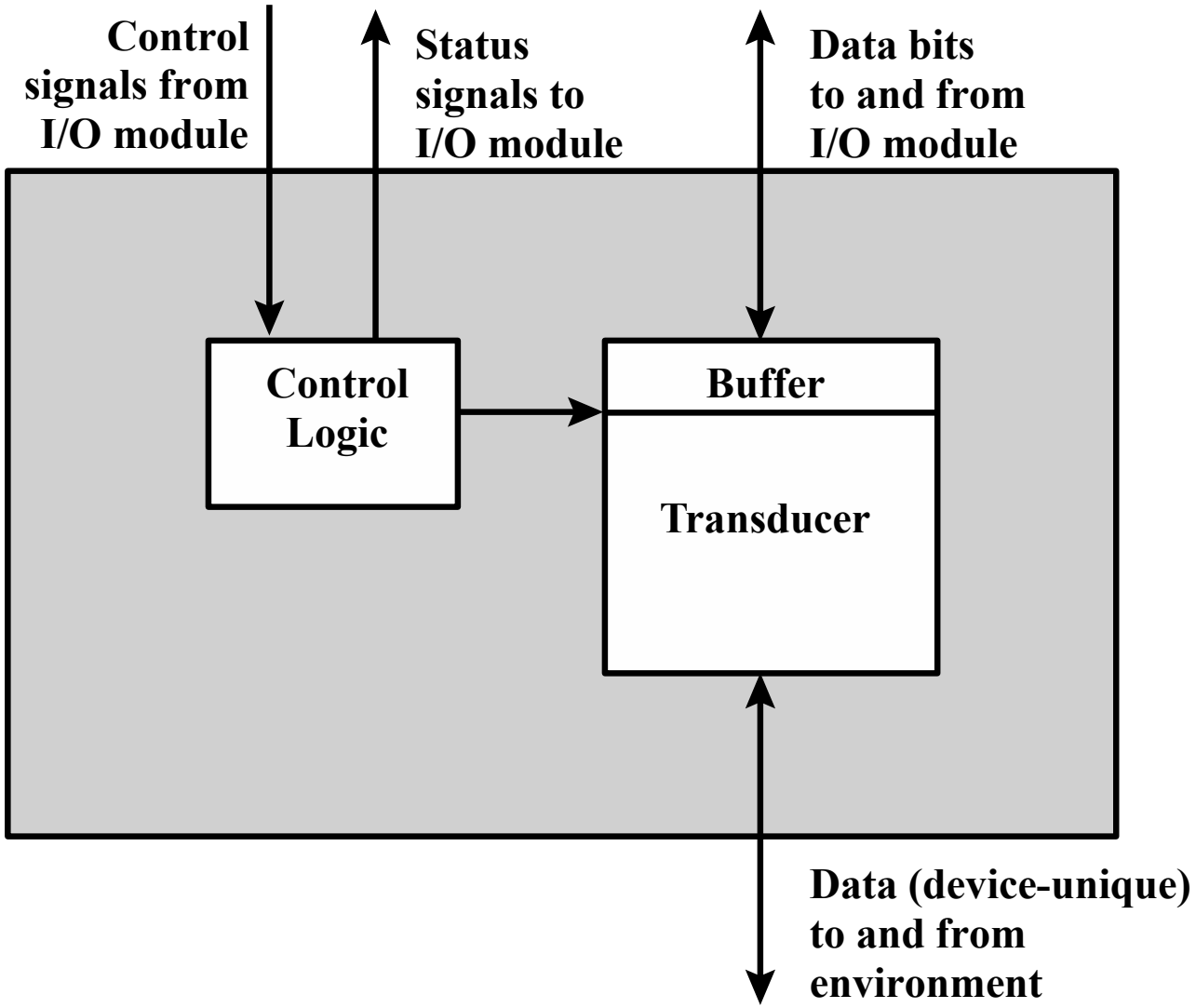


Figure 7.2 Block Diagram of an External Device

External Devices

- A buffer is associated with the transducer to temporarily hold data being transferred between the I/O module and the external environment.
- A buffer size of 8 to 16 bits is common for serial devices, whereas block-oriented devices such as disk drive controllers may have much larger buffers.
- A disk drive contains electronics for exchanging data, control, and status signals with an I/O module plus the electronics for controlling the disk read/write mechanism.
- In a fixed-head disk, the transducer is capable of converting between the magnetic patterns on the moving disk surface and bits in the device's buffer (Figure 7.2).

Keyboard/Monitor

- Basic unit of exchange is the character
  - Associated with each character is a code
  - Commonly used text code is International Reference Alphabet (IRA)
  - Each character in this code is represented by a unique 7-bit binary code
    - 128 different characters are represented
- Characters are of two types:
  - **Printable:**
    - Alphabetic, numeric, and special characters that can be printed on paper or displayed on a screen
  - **Control**
    - Have to do with controlling the printing or displaying of characters
    - Other control characters are concerned with communications procedures

- Most common means of computer/ user interaction.
- User provides input through the keyboard.
- The monitor displays data provided by the computer.

Keyboard Codes

- When the user depresses a key it generates an electronic signal that is interpreted by the transducer in the keyboard and translated into the bit pattern of the corresponding IRA code
- This bit pattern is transmitted to the I/O module in the computer
- On output, IRA code characters are transmitted to an external device from the I/O module
- The transducer interprets the code and sends the required electronic signals to the output device either to display the indicated character or perform the requested control function

Input/ Output

- External Devices

I/O Modules

- Programmed I/O
- Interrupt-Driven I/O
- Direct Memory Access
- I/O Channels and Processors
- External Interconnection Standards

I/O Modules

Module Function

The major functions for an I/O module fall into the following categories:

Control and timing

- Coordinates the flow of traffic between internal resources and external devices

Processor communication

- Involves command decoding, data, status reporting, and address recognition

Device communication

- Involves commands, status information, and data

Data buffering

- Performs the needed buffering operation to balance device and memory speeds

Error detection

- Detects and reports transmission errors

I/O Modules

- The processor may communicate with one or more external devices in unpredictable patterns, depending on the program's need for I/O.
- The internal resources, such as main memory and the system bus, must be shared among a number of activities, including data I/O.
- The I/O function includes a **control** and **timing** requirement, to coordinate the flow of traffic between internal resources and external devices.
- For example, the control of the transfer of data from an external device to the processor might involve the following sequence of steps:
  i. The processor interrogates the I/O module to check the status of the attached device.
  ii. The I/O module returns the device status.
  iii. If the device is operational and ready to transmit, the processor requests the transfer of data, using a command to the I/O module.
  iv. The I/O module obtains a unit of data (e.g., 8 or 16 bits) from the external device.
  v. The data are transferred from the I/O module to the processor.
- If the system employs a bus, then each of the interactions between the processor and the I/O module involves one or more bus arbitrations.

I/O Modules

- **Processor communication** involves the following:
  - **Command decoding:** The I/O module accepts commands from the processor, sent as signals on the control bus.
  - **Data:** Data are exchanged between the processor and the I/O module over the data bus.
  - **Status reporting:** Because peripherals are so slow, it is necessary to know the status of the I/O module. Common status signals are BUSY and READY. Signals to report various error conditions are used.
  - **Address recognition:** **E**ach I/O device has an address. An I/O module must recognize one unique address for each peripheral it controls.
- The I/O module must be able to perform **device communication.**
- Communication involves commands, status information, and data.

I/O Modules: Data Buffering

- An essential task of an I/O module is **data buffering.**
- The transfer rate into and out of main memory or the processor is quite high, the rate is orders of magnitude lower for many peripheral devices and covers a wide range.
- Data coming from main memory are sent to an I/O module in a rapid burst.
- The data are buffered in the I/O module and then sent to the peripheral device at its data rate.
- In the opposite direction, data are buffered so as not to tie up the memory in a slow transfer operation.
- An I/O module must be able to operate at both device and memory speeds.
- If the I/O device operates at a rate higher than the memory access rate, then the I/O module performs the needed buffering operation.

I/O Modules: Error Detection

- An I/O module is often responsible for **error detection** and for subsequently reporting errors to the processor.
- One class of errors includes mechanical and electrical malfunctions reported by the device (e.g., paper jam, bad disk track).
- Another class consists of unintentional changes to the bit pattern as it is transmitted from device to I/O module.
- An error-detecting code is often used to detect transmission errors.
- A simple example is the use of a parity bit on each character of data.
- For example, the IRA character code occupies 7 bits of a byte:
  - The eighth bit is set so that the total number of 1s in the byte is even (even parity) or odd (odd parity).
  - When a byte is received, the I/O module checks the parity to determine whether an error has occurred.

I/O Module Structure

- Figure 7.3 provides a general block diagram of an I/O module.
- The module connects to the rest of the computer through a set of signal lines.

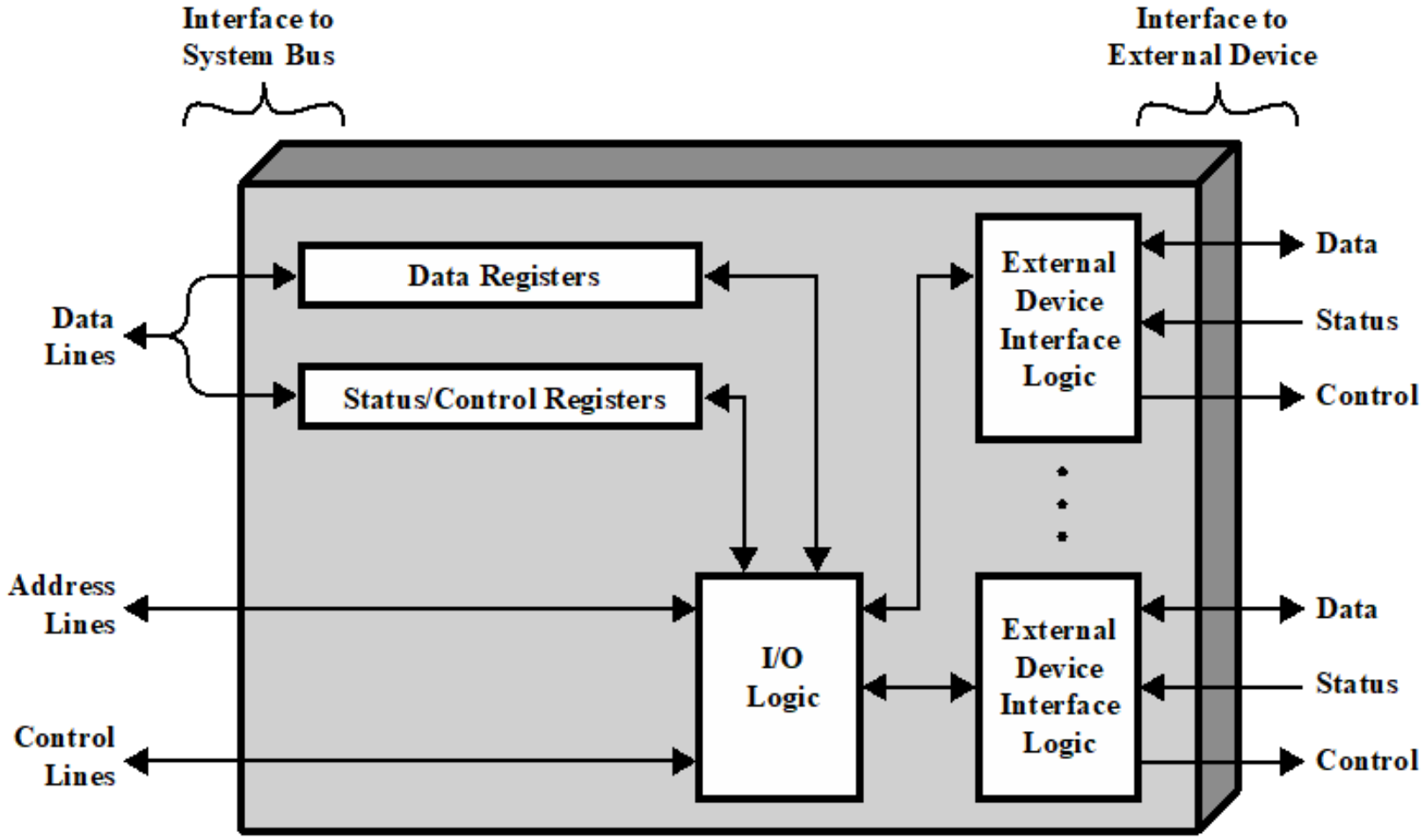


Figure 7.3 Block Diagram of an I/O Module

I/O Module Structure

- Data transferred to and from the module are buffered in one or more data registers.
- One or more status registers to provide current status information.
  - A status register may also function as a control register, to accept detailed control information from the processor.
- The logic within the module interacts with the processor via a set of control lines.
- Processor uses the control lines to issue commands to I/O module.
- Control lines may be used by the I/O module (e.g., status signals).
- The module must also be able to recognize and generate addresses associated with the devices it controls.
- Each I/O module has a unique address or, if it controls more than one external device, a unique set of addresses.

I/O Module Structure

- The I/O module contains logic specific to the interface with each device that it controls.
- The I/O module may hide the details of timing, formats, and the electromechanics of an external device so that the processor can function in terms of simple read and write commands, and possibly open and close file commands.
- An I/O module that takes the processing burden, presenting a high-level interface to the processor, is referred to as an *I/O channel* or *I/O processor.*
- An I/O module that is quite primitive and requires detailed control is referred to as an *I/O controller* or *device controller*.
- I/O controllers are seen on microcomputers, whereas I/O channels are used on mainframes.

Input/ Output

- External Devices
- I/O Modules

Programmed I/O

- Interrupt-Driven I/O
- Direct Memory Access
- I/O Channels and Processors
- External Interconnection Standards

Programmed I/O

Three techniques are possible for I/O operations:

- Programmed I/O:
  - Data are exchanged between the processor and the I/O module
  - Processor executes a program that gives it direct control of the I/O operation
  - When the processor issues a command it must wait until the I/O operation is complete
  - If the processor is faster than the I/O module this is wasteful of processor time
- Interrupt-driven I/O
  - Processor issues an *I/O command*, continues to execute other instructions, and is interrupted by the I/O module when the latter has completed its work
- Direct memory access
  - The I/O module and main memory exchange data directly without processor involvement

Table 7.1 Indicates the relationship among these three techniques.

Table 7.1: I/O Techniques

	No Interrupts	**Use of Interrupts**
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

I/O Commands

- To execute an I/O-related instruction, the processor issues an address, specifying the particular I/O module and external device, and an I/O command.
- There are four types of I/O commands that an I/O module may receive when it is addressed by a processor:
  1. Control: used to activate a peripheral and tell it what to do.
  2. Test: used to test various status conditions associated with an I/O module and its peripherals.
  3. Read: causes the I/O module to obtain an item of data from the peripheral and place it in an internal buffer.
  4. Write: causes the I/O module to take an item of data from the data bus and subsequently transmit that data item to the peripheral.

I/O Commands

- Three techniques for input of a block of data is shown in Figure 7.4.
- Figure 7.4a gives an example of the use of programmed I/O to read in a block of data from a peripheral device (e.g., a record from tape) into memory.
- Data are read in one word (e.g., 16 bits) at a time.
- For each word that is read in, the processor must remain in a status-checking cycle until it determines that the word is available in the I/O module's data register.
- Disadvantage: it is a time-consuming process that keeps the processor busy needlessly.

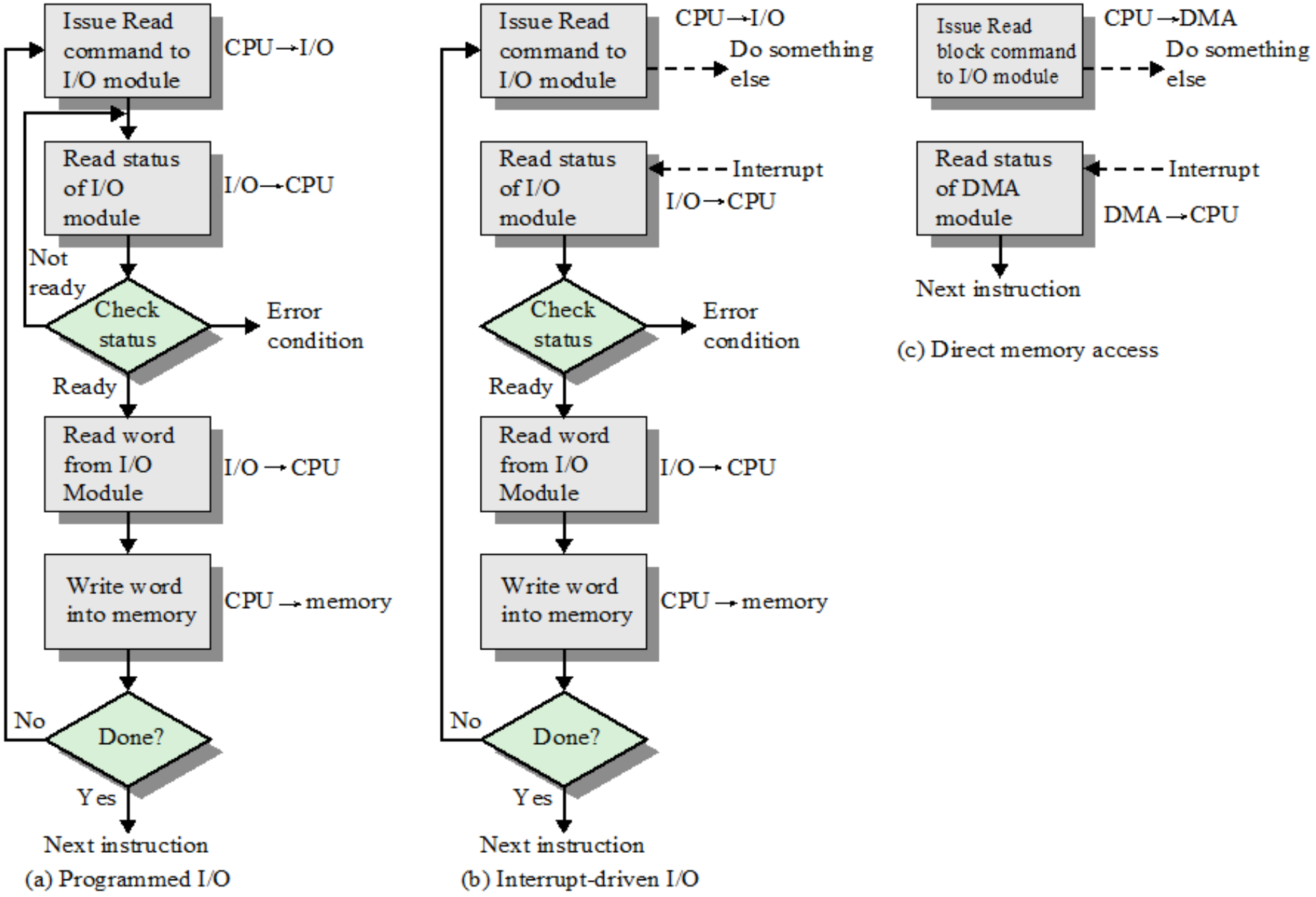


Figure 7.4 Three Techniques for Input of a Block of Data

I/O Instructions

With programmed I/O there is a close correspondence between the I/O-related instructions that the processor fetches from memory and the I/O commands that the processor issues to an I/O module to execute the instructions

The form of the instruction depends on the way in which external devices are addressed

Each I/O device connected through I/O modules is given a unique identifier or address

When the processor issues an I/O command, the command contains the address of the desired device

Each I/O module must interpret the address lines to determine if the command is for itself

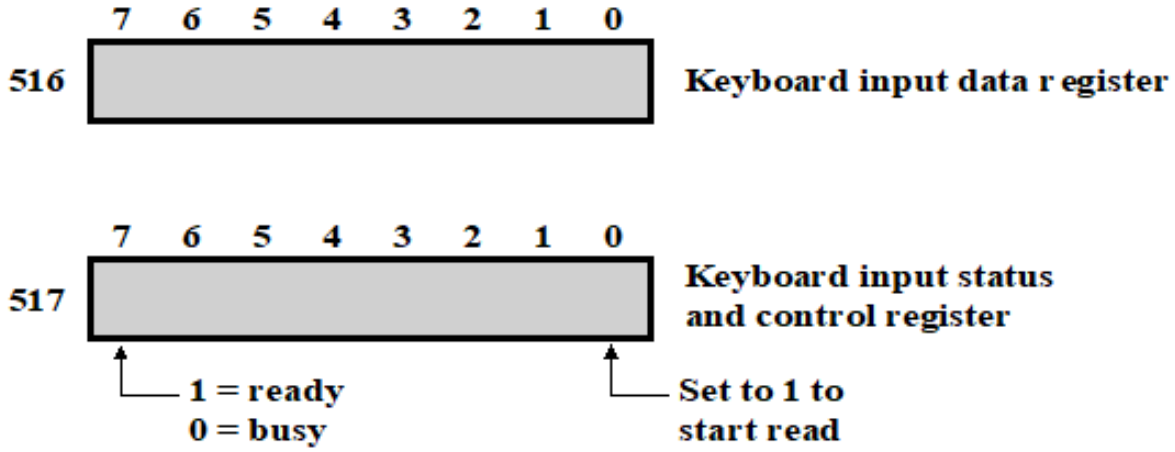
Memory-mapped I/O

There is a single address space for memory locations and I/O devices

A single read line and a single write line are needed on the bus

I/O Mapping

- Figure 7.5 shows two programmed I/O techniques.
- Assume a 10-bit address, with a 512-bit memory (locations 0–511) and up to 512 I/O addresses.
- Memory mapped I/O
  - Devices and memory share an address space
  - I/O looks just like memory read/write
  - No special commands for I/O
    - Large selection of memory access commands available
- Isolated I/O
  - Separate address spaces
  - Need I/O or memory select lines
  - Special commands for I/O
    - Limited set



ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	Load AC	"1"	Load accumulator
	Store AC	517	Initiate keyboard read
202	Load AC	517	Get status byte
	Branch if Sign = 0	202	Loop until ready
	Load AC	516	Load data byte

(a) Memory-mapped I/O

ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	Load I/O	5	Initiate keyboard read
201	Test I/O	5	Check for completion
	Branch Not Ready	201	Loop until complete
	In	5	Load data byte

(b) Isolated I/O

Figure 7.5 Memory-Mapped and Isolated I/O

Input/ Output

- External Devices
- I/O Modules
- Programmed I/O
- **Interrupt-Driven I/O**
- Direct Memory Access
- I/O Channels and Processors
- External Interconnection Standards

Interrupt-Driven I/O

The problem with programmed I/O is that the processor has to wait a long time for the I/O module to be ready for either reception or transmission of data

An alternative is for the processor to issue an I/O command to a module and then go on to do some other useful work

The I/O module will then interrupt the processor to request service when it is ready to exchange data with the processor

The processor executes the data transfer and resumes its former processing

Interrupt-Driven I/O

- The occurrence of an interrupt triggers a number of events, both in the processor hardware and in software. Figure 7.6 shows a typical sequence.

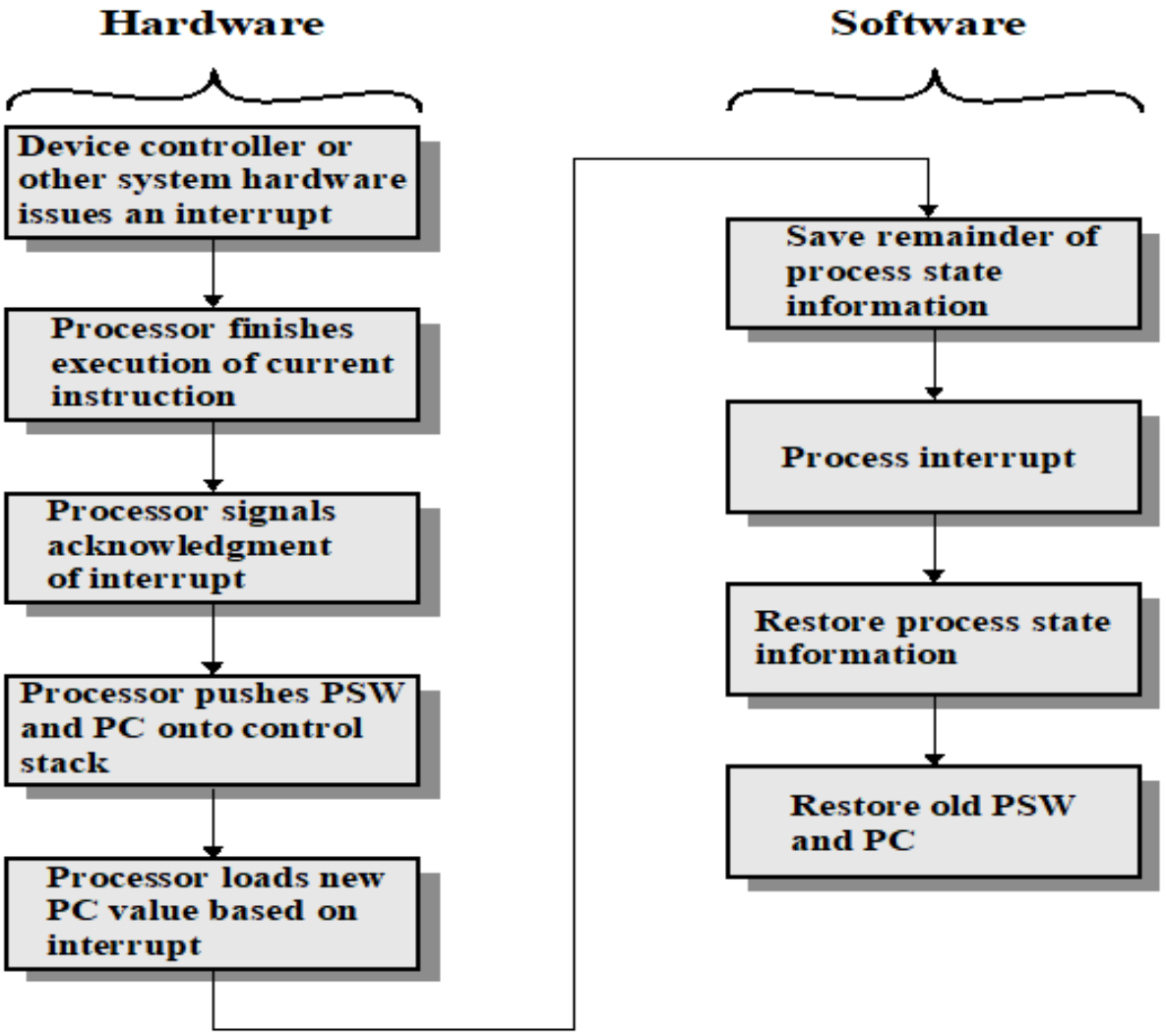


Figure 7.6 Simple interrupt processing

Interrupt-Driven I/O: Interrupt Processing

- When an I/O device completes an I/O operation, the following sequence of hardware events occurs:

1. The device issues an interrupt signal to the processor.
2. Processor finishes execution of the current instruction before responding to the interrupt.
3. Processor tests for an interrupt, determines that there is one, and sends an ack signal to the device that issued the interrupt. The acknowledgment allows the device to remove its interrupt signal.
4. The processor needs to prepare to transfer control to the interrupt routine.
   - Save information needed to resume the current program at the point of interrupt.
   - The information required is:
     (a) status of the processor, contained in a register called the program status word (PSW)
     (b) the location of the next instruction to be executed, which is contained in the program counter
   - These can be pushed onto the system control stack.
5. The processor loads the program counter with the entry location of the interrupt-handling program that will respond to this interrupt.
   - Depending on computer architecture and operating system, there may be one program for each type of interrupt; or one program for each device and each type of interrupt.
   - If there is more than one interrupt-handling routine, the processor must determine which one to invoke.

Interrupt-Driven I/O: Interrupt Processing

- Once the program counter has been loaded, the processor proceeds to the next instruction cycle, which begins with an instruction fetch.
- The execution of this program results in the following operations:

- 6. The PC and PSW relating to the interrupted program saved on the system stack. The contents of the processor registers need to be saved, because these registers may be used by the interrupt handler. The interrupt handler will begin by saving the contents of all registers on the stack.
- Figure 7.7a shows a simple example. A user program is interrupted after the instruction at location *N*.
- 7. The interrupt handler next processes the interrupt. Examines status information relating to the I/O operation/interrupt. Send additional commands or acknowledgments to the I/O device.
- 8. When interrupt processing is complete, the saved register values are retrieved from the stack and restored to the registers (e.g., see Figure 7.7b).
- 9. Restore the PSW and PC values from the stack. As a result, the next instruction to be executed will be from the previously interrupted program.

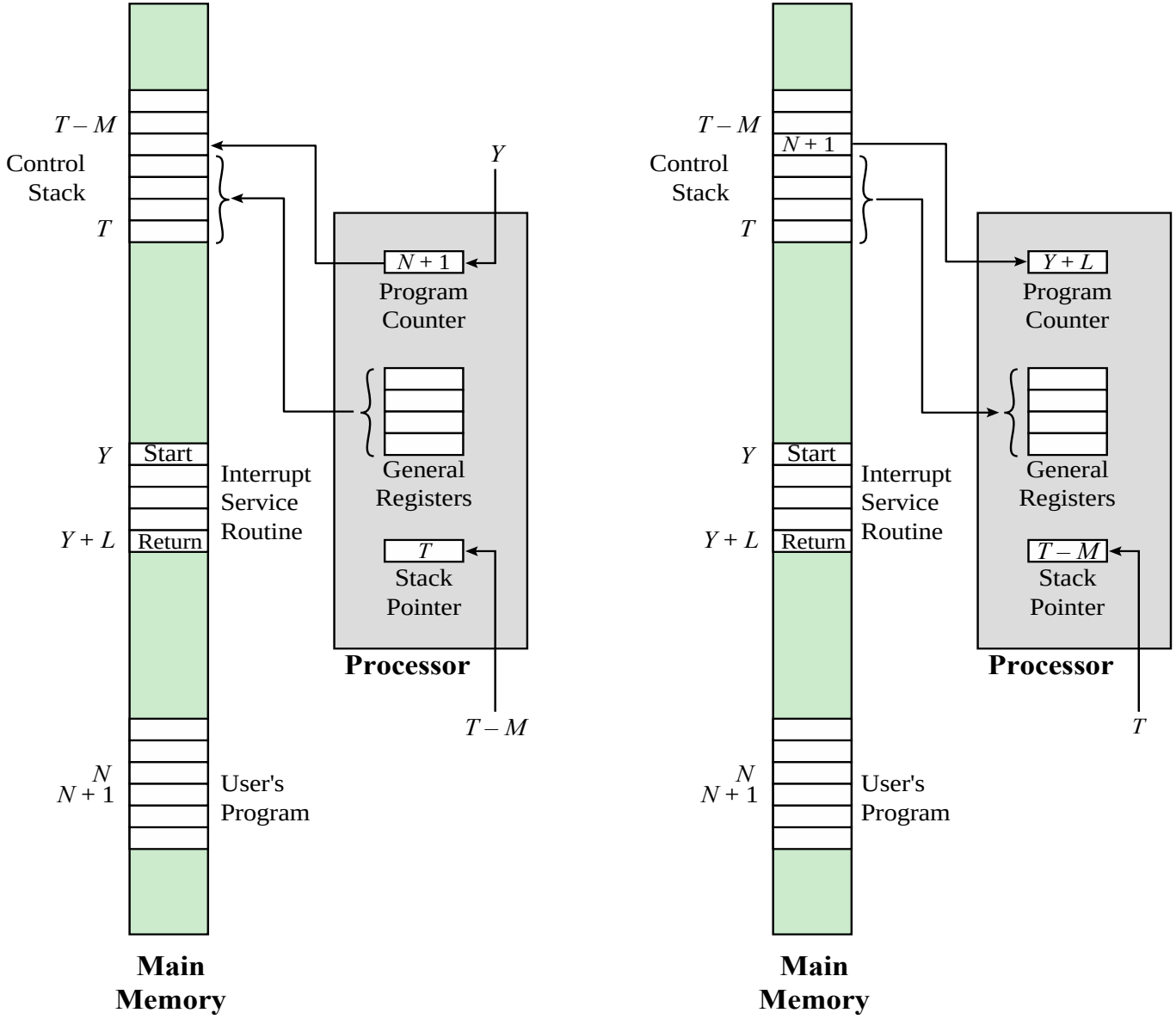


Figure 7.7 Changes in Memory and Registers for an Interrupt

Design Issues

- Two design issues arise in implementing interrupt I/O”
  i. There will almost invariably be multiple I/O modules, how does the processor determine which device issued the interrupt?
  ii. If multiple interrupts have occurred, how does the processor decide which one to process?
- Four general categories of techniques are in common use:
  - Multiple interrupt lines
  - Software poll
  - Daisy chain (hardware poll, vectored)
  - Bus arbitration (vectored)

Four General Categories of Techniques

- **Multiple interrupt lines**
  - Between the processor and the I/O modules
  - Most straightforward approach to the problem
  - Consequently even if multiple lines are used, it is likely that each line will have multiple I/O modules attached to it
- **Software poll**
  - When processor detects an interrupt it branches to an interrupt-service routine whose job is to poll each I/O module to determine which module caused the interrupt
  - Time consuming
- **Daisy chain (hardware poll, vectored)**
  - The interrupt acknowledge line is daisy chained through the modules
  - Vector – address of the I/O module or some other unique identifier
  - Vectored interrupt – processor uses the vector as a pointer to the appropriate device-service routine, avoiding the need to execute a general interrupt-service routine first
- **Bus arbitration (vectored)**
  - I/O module must first gain control of the bus before it can raise the interrupt request line
  - When the processor detects the interrupt it responds on the interrupt acknowledge line
  - Then the requesting module places its vector on the data lines

Intel 82C59A Interrupt Controller

- The Intel 80386 provides a single Interrupt Request (INTR) and a single Interrupt Acknowledge (INTA) line.
- To allow the 80386 to handle a variety of devices and priority structures, it is configured with an external interrupt arbiter, the 82C59A.
- External devices are connected to the 82C59A, which in turn connects to the 80386.
- Figure 7.8 shows the use of the 82C59A to connect multiple I/O modules for the 80386.
- A single 82C59A can handle up to eight modules.
- If control for more than eight modules is required, a cascade arrangement can be used to handle up to 64 modules.
- The 82C59A is programmable.

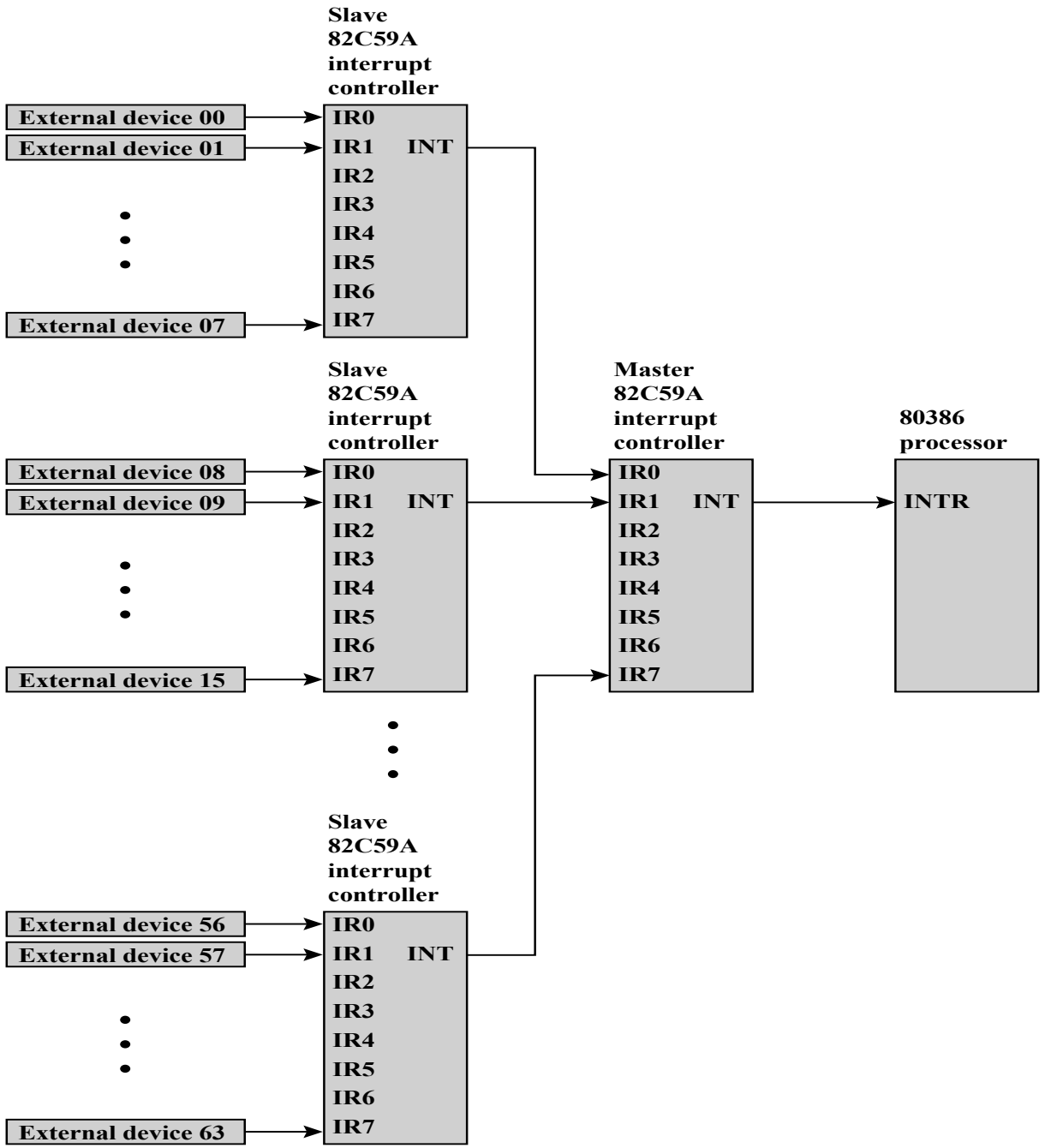


Figure 7.8 Use of the 82C59A Interrupt Controller

Intel 82C59A Interrupt Controller

- The 82C59A's sole responsibility is the management of interrupts.
- It accepts interrupt requests from attached modules, determines which interrupt has the highest priority, and then signals the processor by raising the INTR line.
- The processor acknowledges via the INTA line, prompts the 82C59A to place the appropriate vector information on the data bus.
- The processor can then proceed to process the interrupt and to communicate directly with the I/O module to read or write data.
- The following interrupt modes are possible:
  - **Fully nested:** The interrupt requests are ordered in priority from 0 (IR0) through 7 (IR7).
  - **Rotating:** If number of interrupting devices are of equal priority, a device, after being serviced, receives the lowest priority in the group.
  - **Special mask:** Allows the processor to inhibit interrupts from certain devices.

Intel 8255A Programmable Peripheral Interface

- The 8255A is a single-chip, general-purpose I/O module designed for use with the Intel 80386 processor.
- Its uses include as a controller for simple I/O devices for microprocessors and in embedded systems, including microcontroller systems.
- Figure 7.9 shows a block diagram plus the pin assignment for the 40-pin package in which it is housed.

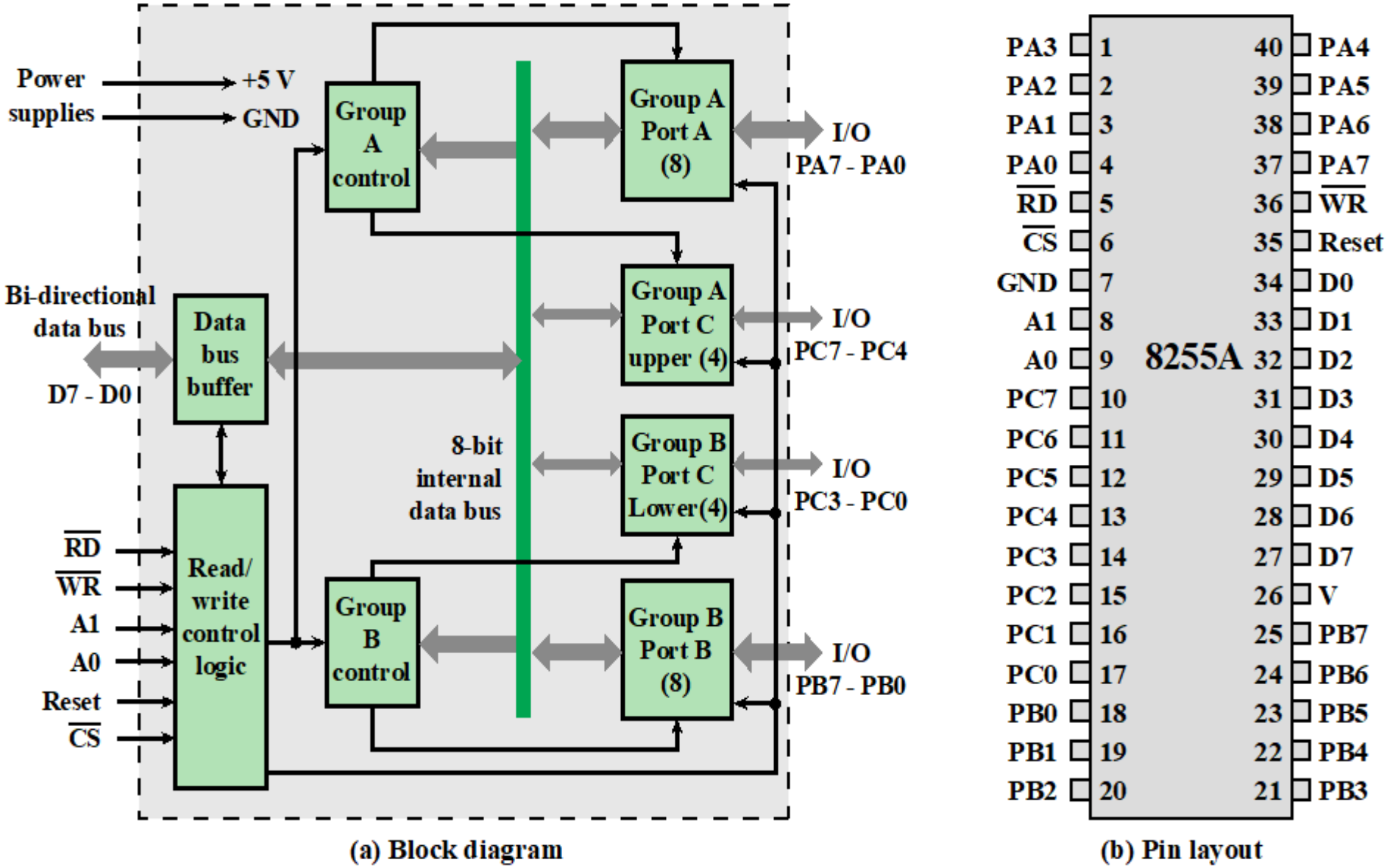


Figure 7.9 The Intel 8255A Programmable Peripheral Interface

Intel 8255A PPI

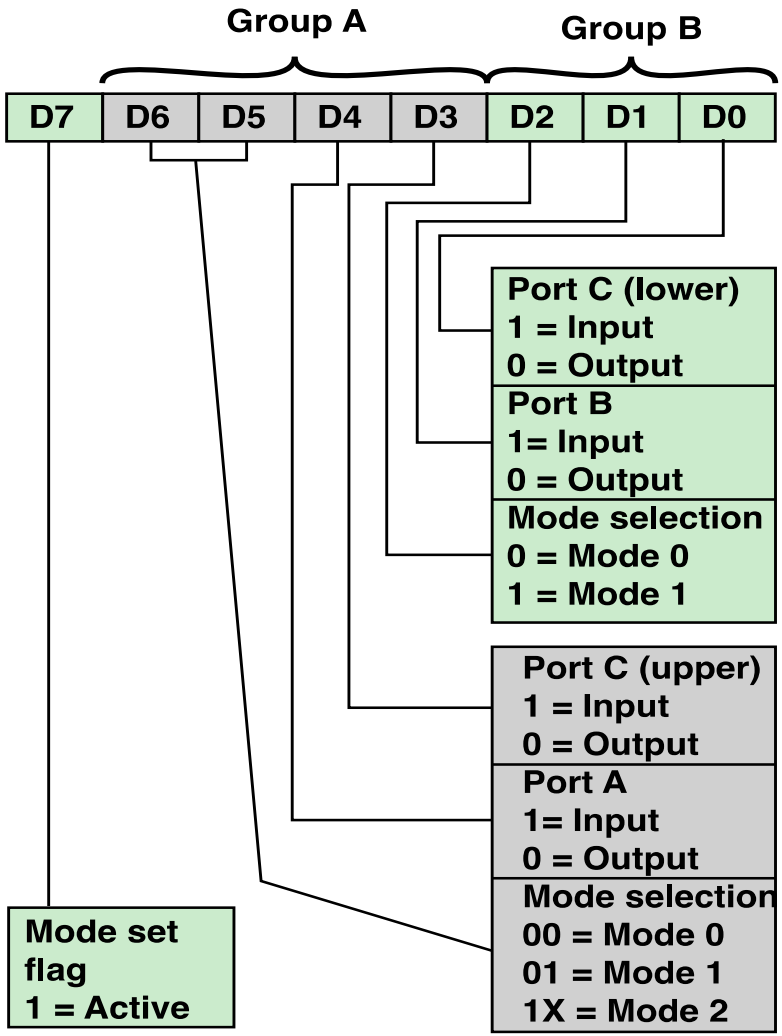
- The Intel 8255A includes the following lines:
  - **D0–D7:** These are the data I/O lines for the device. All information read from and written to the 8255A occurs via these eight data lines.
  - $\overline{CS}$ **(Chip Select Input):** If this line is a logical 0, the microprocessor can read and write to the 8255A.
  - $\overline{RD}$ **(Read Input):** If this line is a logical 0 and the CS input is a logical 0, the 8255A data outputs are enabled onto the system data bus.
  - $\overline{WR}$ **(Write Input):** If this input line is a logical 0 and the CS input is a logical 0, data are written to the 8255A from the system data bus.
  - **RESET:** The 8255A is placed into its reset state if this input line is a logical 1. All peripheral ports are set to the input mode.
  - **PA0–PA7, PB0–PB7, PC0–PC7:** These signal lines are used as 8-bit I/O ports. They can be connected to peripheral devices.
  - **A0, A1:** The logical combination of these two input lines determine which internal register of the 8255A data are written to or read from.

Intel 8255A PPI

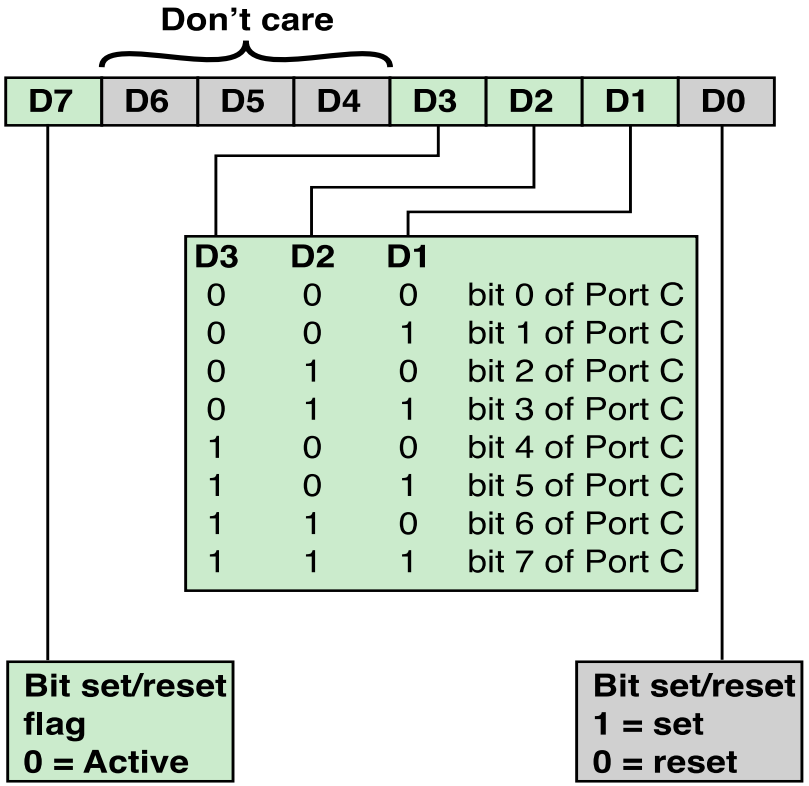
- The right side of Figure 7.9a is the external interface of the 8255A.
- The 24 I/O lines are divided into three 8-bit groups (A, B, C).
- Each group can function as an 8-bit I/O port.
- Group C is subdivided into 4-bit groups (C_A and C_B), which may be used in conjunction with the A and B I/O ports.
- The left side of Figure 7.9a is the internal interface to the microprocessor (MP) system bus, includes an 8-bit bidirectional data bus (D0 to D7), used to transfer data between the MP and I/O ports and to transfer control information
- The processor controls the 8255A using 8-bit control register in the processor.
- The processor can set the value of the control register to specify a variety of operating modes and configurations.
- The control register bits are set in the processor and then sent to the control port over lines D0-D7.
- When the processor sets both A1 and A2 to 1, the 8255A interprets the 8-bit value on the data bus as a control word.

Intel 8255A PPI

- The processor transfers an 8-bit control word with line D7 set to 1 (Figure 7.10a), control word is used to configure the operating mode of the 24 I/O lines, the three modes are:
- **Mode 0:** This is the basic I/O mode. The three groups of eight external lines function as three 8-bit I/O ports. Each port can be designated as input or output.
  - Data may only be sent to a port if the port is defined as output, and data may only be read from a port if the port is set to input.
- **Mode 1:** Ports A and B can be configured as either input or output, and lines from port C serve as control lines for A and B.
  - The control signals serve two principal purposes: “handshaking” and interrupt request.
  - Handshaking: One control line is used by the sender as a DATA READY line. Another line is used by the receiver as an ACKNOWLEDGE, indicating that the data have been read and the data lines may be cleared.
  - INTERRUPT REQUEST line is tied back to the system bus.
- **Mode 2 (**bidirectional mode): Port A can be configured as either the input or output lines for bidirectional traffic on port B, with the port B lines providing the opposite direction.
  - Again, port C lines are used for control signaling.
  - When the processor sets D7 to 0 (Figure 7.10b), the control word is used to program the bit values of port C individually. This feature is rarely used.



(a) Mode definition of the 8255 control register to configure the 8255



(b) Bit definitions of the 8255 control register to modify single bits of port C

Figure 7.10 The Intel 8255A Control Word

Keyboard/Display: Example

- The 8255A is programmable via the control register, it can be used to control a variety of simple peripheral devices.
- Figure 7.11 illustrates its use to control a keyboard/display terminal.
- The keyboard provides 8 bits of input.
- Two of these bits, SHIFT and CONTROL, have special meaning to the keyboard-handling program executing in the processor.
- Two handshaking control lines are provided for use with the keyboard.
- The display is also linked by an 8-bit data port.
- Two of the bits have special meanings that are transparent to the 8255A.
- In addition to two handshaking lines, two lines provide additional control functions.

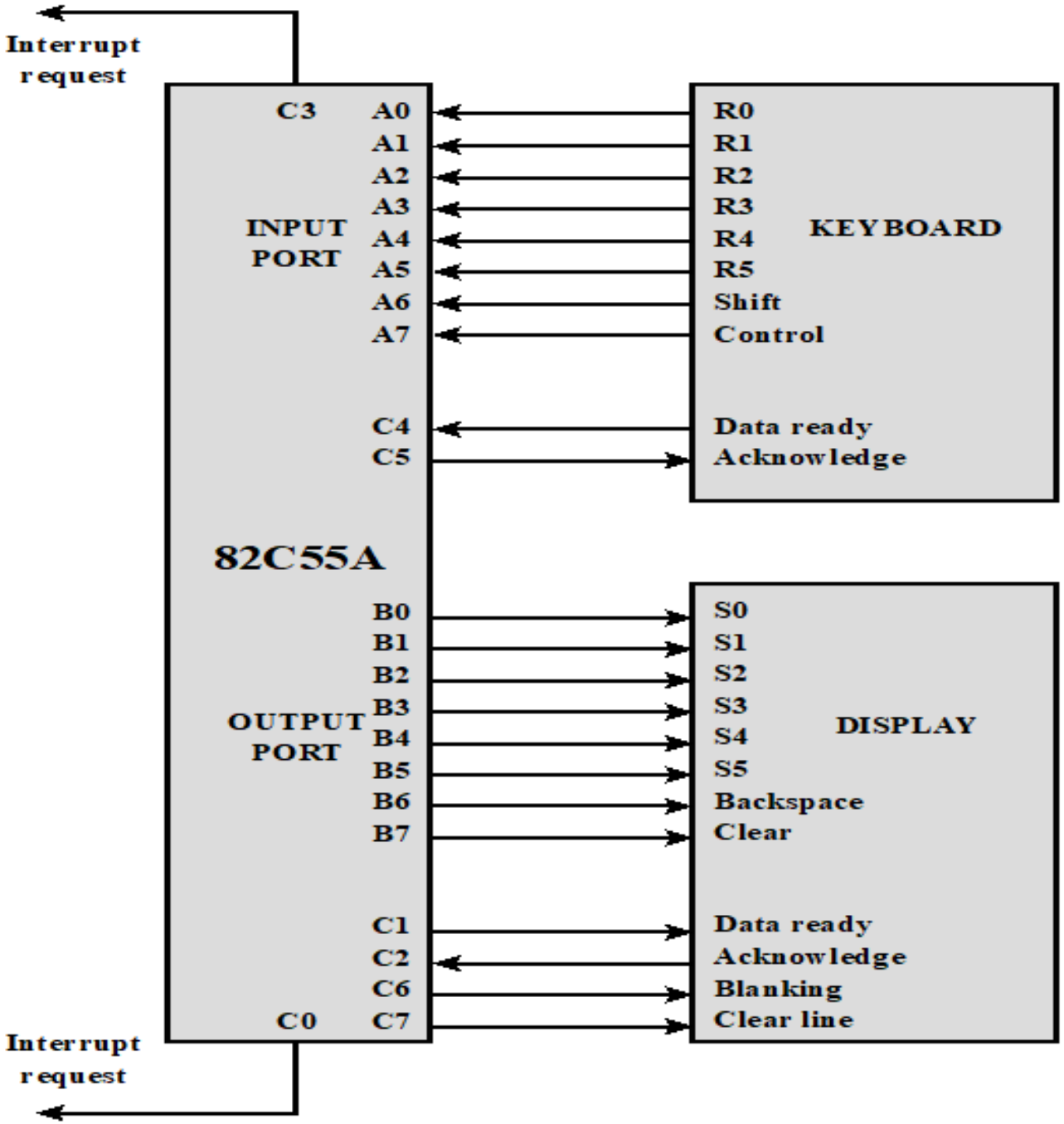


Figure 7.11 Keyboard/Display Interface to 82C55A

Drawbacks of Programmed and Interrupt-Driven I/O

- Interrupt-driven I/O, is efficient than simple programmed I/O, still requires the active intervention of the processor to transfer data between memory and an I/O module, and any data transfer must traverse a path through the processor.
- Both of these forms suffer from two inherent drawbacks:
  1. The I/O transfer rate is limited by the speed with which the processor can test and service a device.
  2. The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer.
- Consider the transfer of a block of data:
  - Using simple programmed I/O, the processor is dedicated to the task of I/O and can move data at a rather high rate, at the cost of doing nothing else.
  - Interrupt I/O frees up the processor to some extent at the expense of the I/O transfer rate.
- Both methods have an impact on both processor activity and I/O transfer rate.
- When large volumes of data are to be moved, a more efficient technique is required: direct memory access (DMA).

Input/ Output

- External Devices
- I/O Modules
- Programmed I/O
- Interrupt-Driven I/O

- **Direct Memory Access**

- I/O Channels and Processors
- External Interconnection Standards

Direct Memory Access

- DMA involves an additional module on the system bus.
- The DMA module (Figure 7.12) is capable of mimicking the processor and, indeed, of taking over control of the system from the processor.
- Do this to transfer data to and from memory over the system bus.
- The DMA module must use the bus only when the processor does not need it, or it must force processor to suspend operation temporarily.
- The latter technique is referred to as *cycle stealing*, because the DMA module in effect steals a bus cycle.

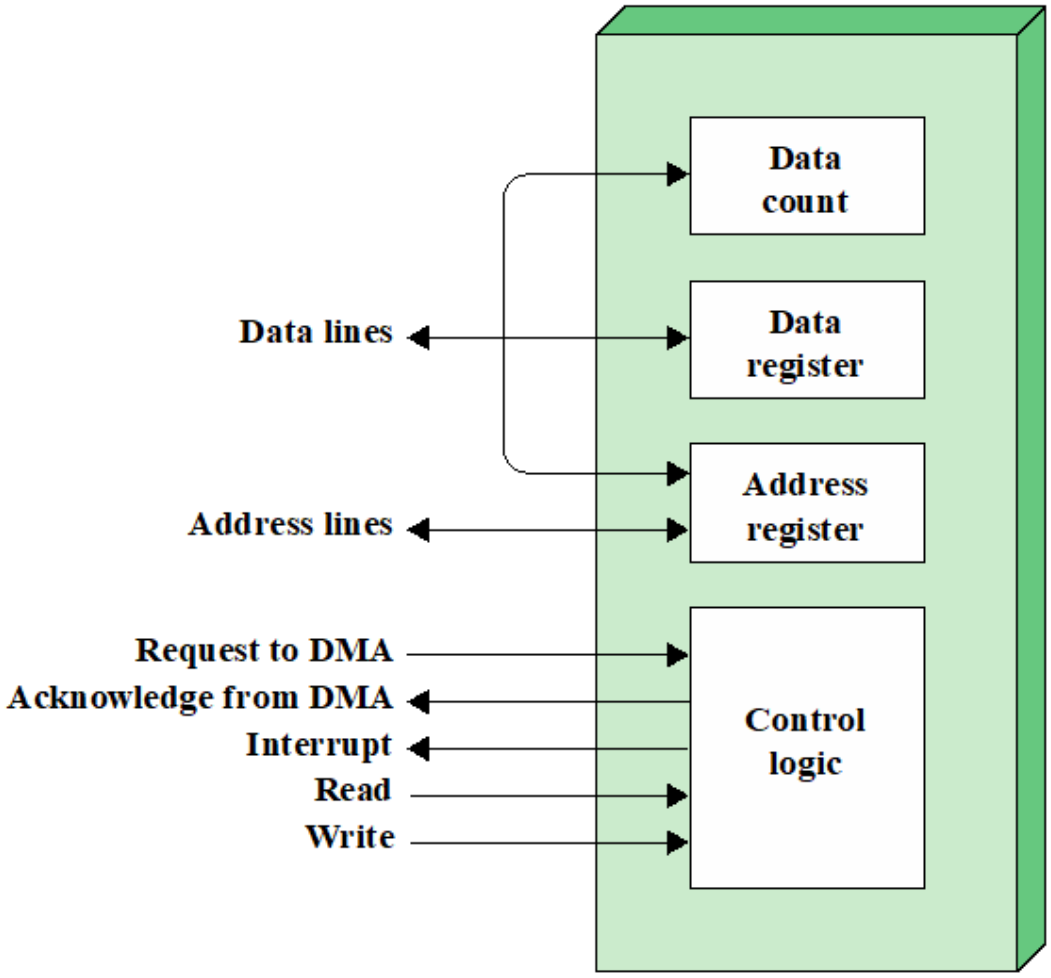


Figure 7.12 Typical DMA Block Diagram

Direct Memory Access

- When the processor wishes to read or write a block of data, it issues a command to the DMA module, by sending following to the DMA module:
  - Whether a read or write is requested, using the read or write control line between the processor and the DMA module.
  - The address of the I/O device involved, communicated on the data lines.
  - The starting location in memory to read from or write to, communicated on the data lines and stored by the DMA module in its address register.
  - The number of words to be read or written, again communicated via the data lines and stored in the data count register.
- The processor then continues with other work. It has delegated this I/O operation to the DMA module.
- The DMA module transfers the entire block of data, one word at a time, directly to or from memory, without going through the processor.
- When the transfer is complete, the DMA module sends an interrupt signal to the processor.

Direct Memory Access

- Figure 7.13 shows where in the instruction cycle the processor may be suspended.
- In each case, the processor is suspended just before it needs to use the bus.
- The DMA module then transfers one word and returns control to the processor.
- The processor pauses for one bus cycle, to cause the processor to execute slowly.
- For a multiple-word I/O transfer, DMA is far more efficient than interrupt-driven or programmed I/O.

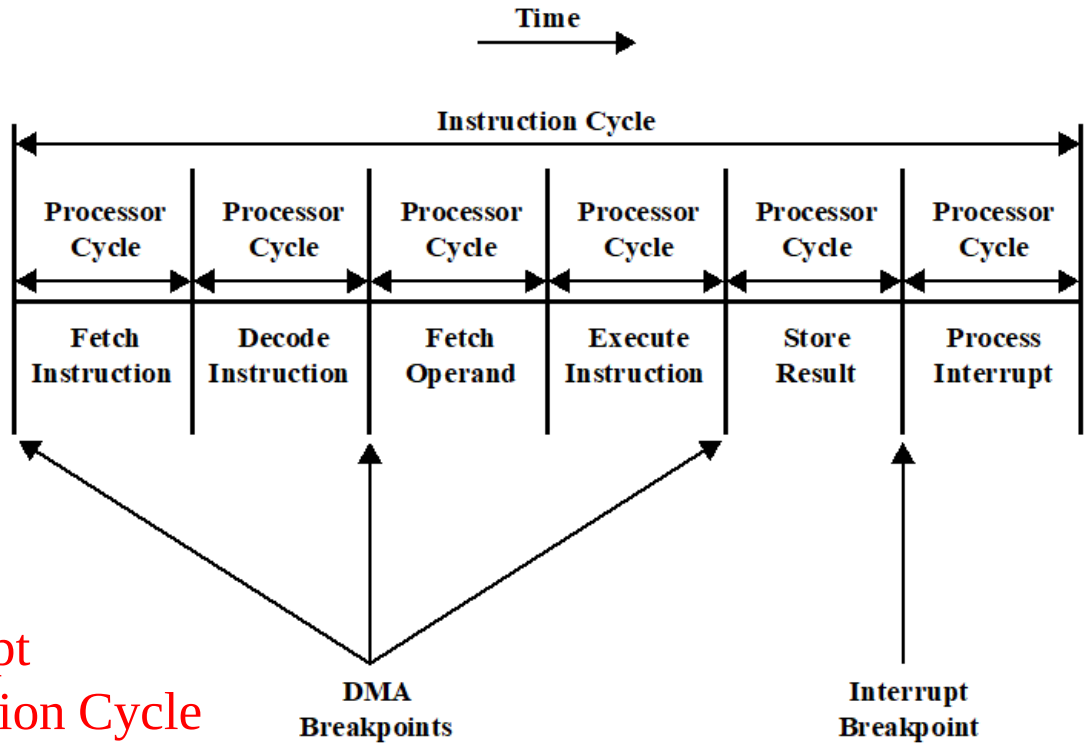


Figure 7.13 DMA and Interrupt Breakpoints during an Instruction Cycle

Direct Memory Access

- DMA mechanism can be configured in a variety of ways as shown in Figure 7.14.
- In Figure 7.14a, all modules share the *same system bus*.
- DMA module, acting as a surrogate processor, uses programmed I/O to exchange data between memory and an I/O module through DMA module.
- This configuration, is inexpensive, but inefficient.
- Each transfer of a word consumes two bus cycles.

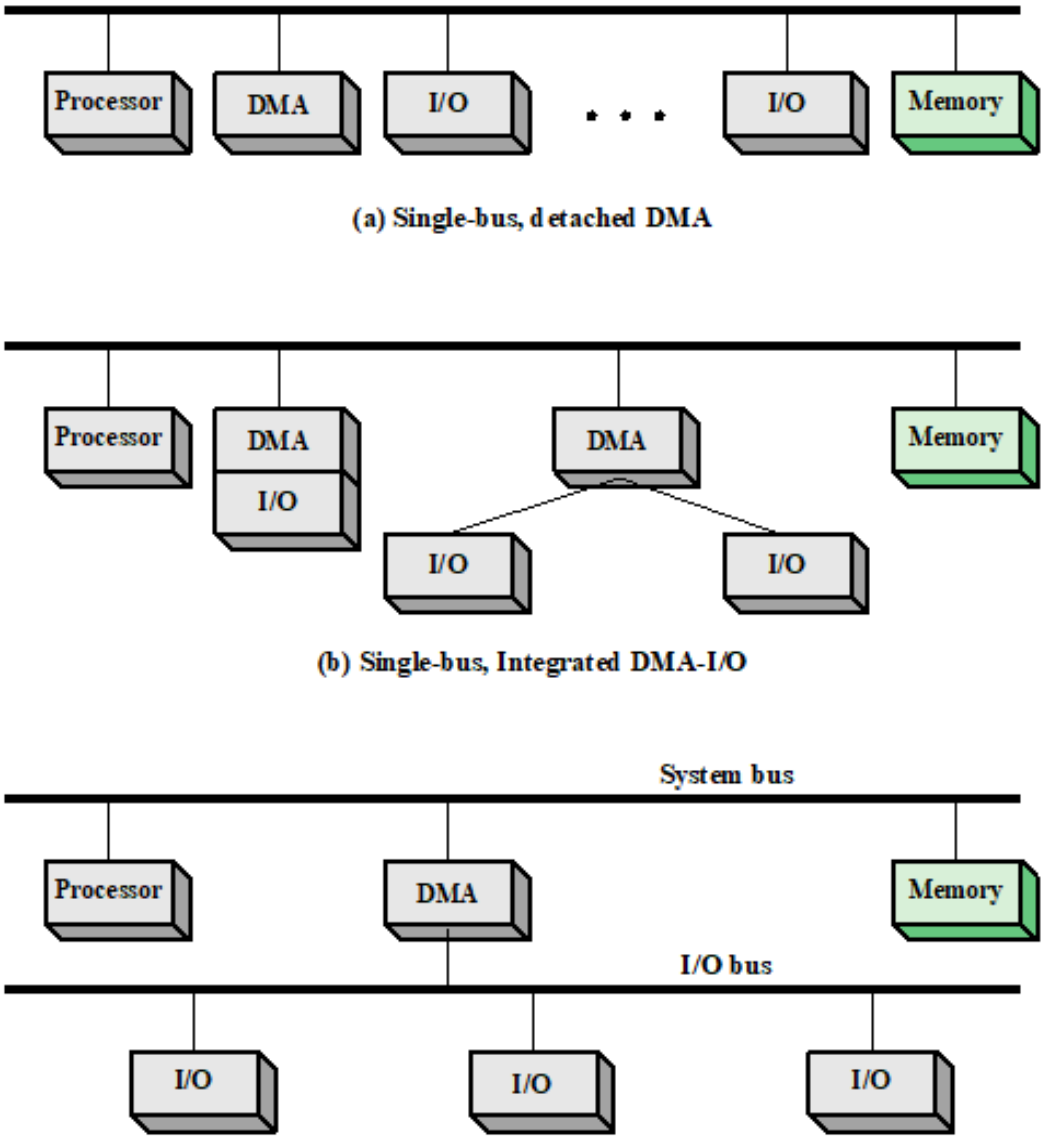


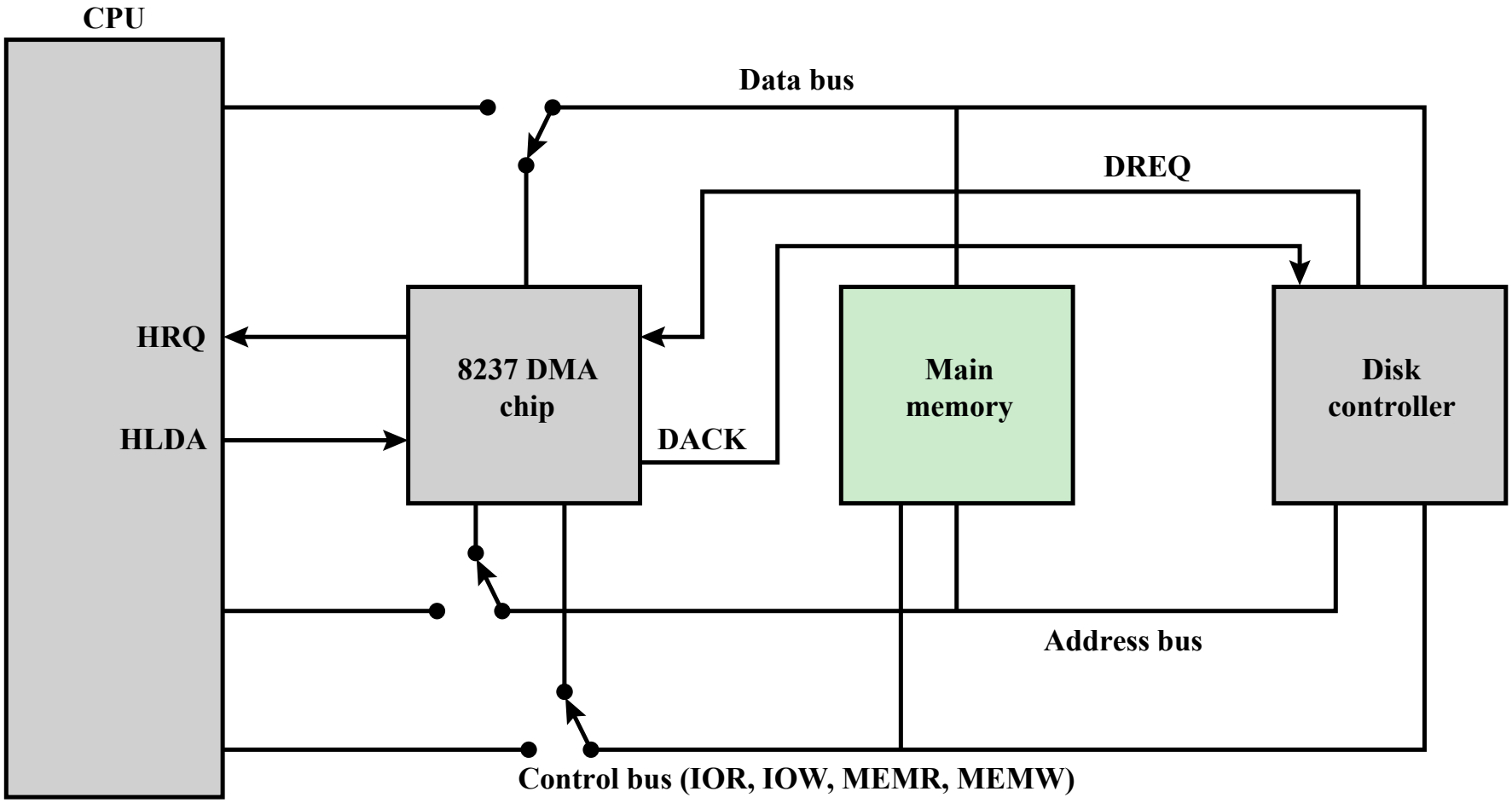
Figure 7.14 Alternative DMA Configurations

Direct Memory Access

- In Figure 7.14b, there is a path between the DMA module and one or more I/O modules that does *not include the system bus*.
- The DMA logic may be a part of an I/O module, or it may be a separate module that controls one or more I/O modules.
- This concept can be extended by connecting I/O modules to the DMA module using an I/O bus (Fig. 7.14c), reduces the number of I/O interfaces in DMA module to one and provides for an expandable configuration.
- In (Figures 7.14b and c), system bus that DMA module shares with the processor and memory is used by the DMA module only to exchange data with memory.
- The exchange of data between the DMA and I/O modules takes place off the system bus.

Intel 8237A DMA Controller

- The Intel 8237A DMA controller interfaces to the 80 x 86 family of processors and to DRAM memory to provide a DMA capability.
- Figure 7.15 indicates the location of the DMA module.
- When the DMA module needs to use the system buses (data, address, and control) to transfer data, it sends a signal called HOLD to the processor.
- The processor responds with the HLDA (hold acknowledge) signal, indicating that the DMA module can use the buses.



DACK = DMA acknowledge
DREQ = DMA request
HLDA = HOLD acknowledge
HRQ = HOLD request

Figure 7.15 8237 DMA Usage of System Bus

Intel 8237A DMA controller

- If the DMA module is to transfer a block of data from memory to disk, it will do:
  1. The peripheral device (such as the disk controller) will request the service of DMA by pulling DREQ (DMA request) high.
  2. The DMA will put a high on its HRQ (hold request), signaling the CPU through its HOLD pin that it needs to use the buses.
  3. The CPU will finish the present bus cycle and respond to the DMA request by putting high on its HDLA, telling the 8237 DMA that it can go ahead and use the buses to perform its task. HOLD must remain active high.
  4. DMA will activate DACK (DMA acknowledge), which tells the peripheral device that it will start to transfer the data.
  5. DMA transfer the data from memory to peripheral by placing the address of the first byte of the block on the address bus and activating MEMR, thereby reading the byte from memory into the data bus; it then activates IOW to write it to the peripheral. DMA decrements the counter and increments the address pointer and repeats this process until count reaches zero and task is finished.
  6. After the DMA has finished its job it will deactivate HRQ, signaling the CPU that it can regain control over its buses.

Intel 8237A DMA controller

- While the DMA is using the buses to transfer data, the processor is idle.
- Similarly, when the processor is using the bus, the DMA is idle.
- The 8237 DMA is known as a *fly-by DMA controller.*
- *The data being moved from* one location to another does not pass through the DMA chip and is not stored in the DMA chip.
- The DMA can only transfer data between an I/O port and a memory address, but not between two I/O ports or two memory locations.
- The DMA chip can perform a memory-to-memory transfer via a register.
- The 8237 contains four DMA channels that can be programmed independently, and any one of the channels may be active at any moment.
- These channels are numbered 0, 1, 2, and 3.
- The 8237 has a set of five control/command registers to program and control DMA operation over one of its channels (Table 7.2).

Table 7.2: Intel 8237A Registers

Bit	Command	Status	Mode	Single Mask	All Mask
D0	Memory-to-memory E/D	Channel 0 has reached TC	Channel select	Select channal mask bit	Clear/set channel 0 mask bit
D1	Channel 0 address hold E/D	Channel 1 has reached TC			Clear/set channel 1 mask bit
D2	Controller E/D	Channel 2 has reached TC	Verify/write/ read transfer	Clear/set mask bit	Clear/set channel 2 mask bit
D3	Normal/compressed timing	Channel 3 has reached TC		Not used	Clear/set channel 3 mask bit
D4	Fixed/rotating priority	Channel 0 request	Auto-initialization E/D		Not used
D5	Late/extended write selection	Channel 0 request	Address increment/ decrement select		
D6	DREQ sense active high/low	Channel 0 request			
D7	DACK sense active high/low	Channel 0 request	Demand/single/ block/cascade mode select		

E/D = enable/disable
TC = terminal count

Direct Memory Access

- DMA is not able to scale to meet the increased demand due to dramatic increases in data rates for network I/O.
- Demand is coming primarily from the widespread deployment of 10-Gbps and 100-Gbps Ethernet switches to handle massive amounts of data transfer to and from database servers and other high-performance systems.
- Another source of traffic comes from Wi-Fi in the gigabit range.
- Network Wi-Fi devices that handle 3.2 Gbps and 6.76 Gbps are becoming widely available and producing demand on enterprise systems.

Direct Cache Access

- Enabling the I/O function to have direct access to the cache can enhance performance, a technique known as direct cache access (DCA).
- Describe the way in which contemporary multicore systems use on-chip shared cache to enhance DMA performance.
  - This approach involves enabling the DMA function to have direct access to the last-level cache.
- Examine cache-related performance issues that manifest when high-speed network traffic is processed.
- Several different strategies for DCA that are designed to enhance network protocol processing performance.
- DCA approach implemented by Intel, referred to as Direct Data I/O.

DMA Using Shared Last-Level Cache: Intel Xeon Multicore Processor

- Intel Xeon is Intel's high-end, high-performance processor family, used in servers, high-performance workstations, and supercomputers.
- Many of the members of the Xeon family use a ring interconnect system, as illustrated for the Xeon E5-2600/4600 in Figure 7.16.
- E5-2600/4600 can be configured with up to 8 cores on a single chip.
- Each core has dedicated L1 and L2 caches.
- There is a shared L3 cache of up to 20 MB.
- The L3 cache is divided into slices, one associated with each core although each core can address the entire cache.
- Further, each slice has its own cache pipeline, so that requests can be sent in parallel to the slices.
- The bidirectional high-speed ring interconnect links cores, last-level cache, PCIe, and *integrated memory controller* (IMC).

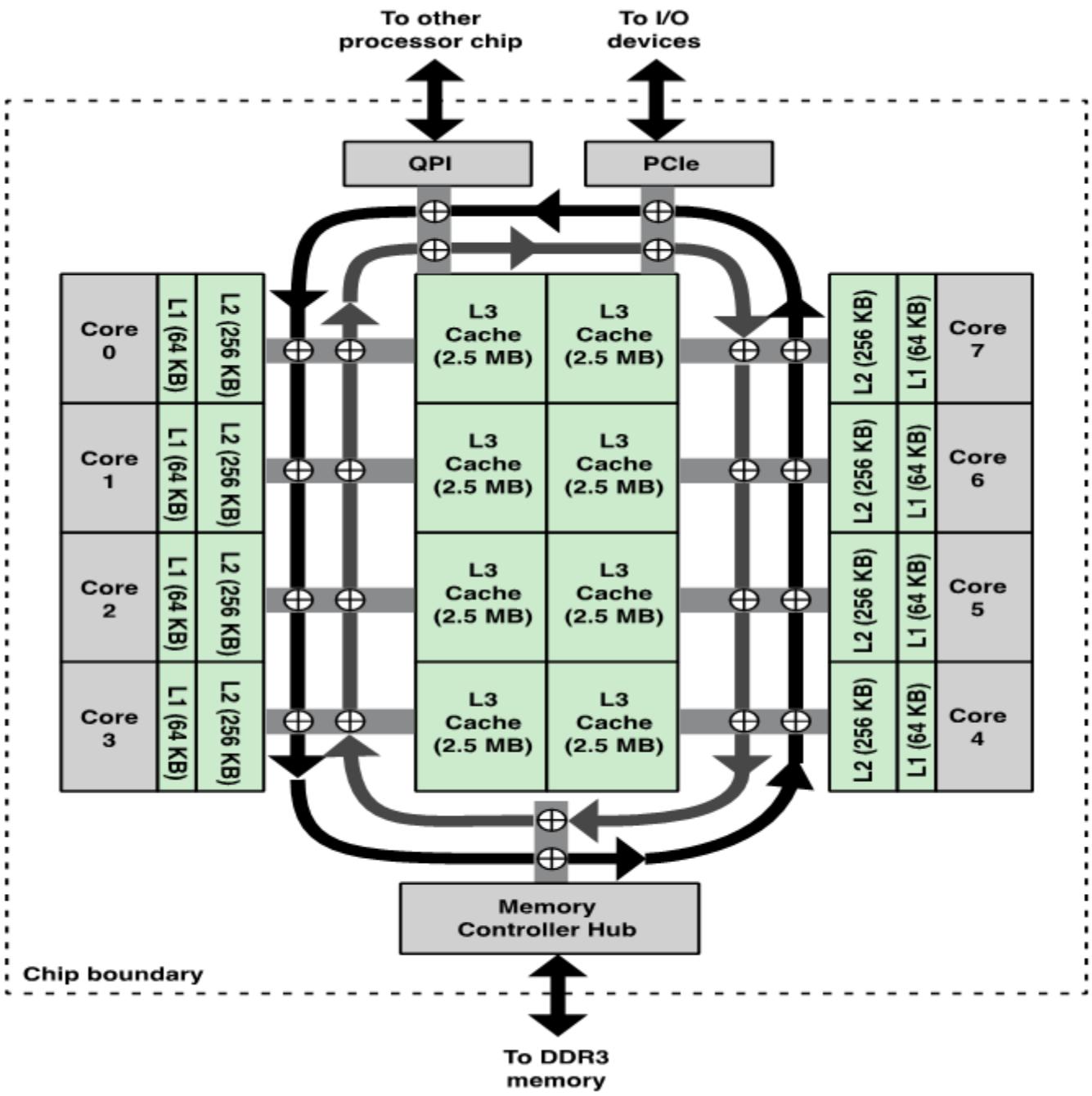


Figure 7.16 Xeon E5-2600/4600 Chip Architecture

Intel Xeon Multicore Processor

- In essence, the ring operates as follows:
  1. Each component that attaches to the bidirectional ring (QPI, PCIe, L3 cache, L2 cache) is considered a ring agent, and implements ring agent logic.
  2. The ring agents cooperate via a distributed protocol to request and allocate access to the ring, in the form of time slots.
  3. When an agent has data to send, it chooses the ring direction that results in the shortest path to the destination and transmits when a scheduling slot is available.
- The ring architecture provides good performance and scales well for multiple cores, up to a point.
- For systems with a greater number of cores, multiple rings are used, with each ring supporting some of the cores.

Intel Xeon Multicore Processor

- In DMA operation, data is exchanged between main memory and an I/O device by means of the system interconnection structure, such as a bus, ring, ...
- If the Xeon E5 used a DMA technique, output would proceed as follows:
- An I/O driver running on a core would send an I/O command to the I/O controller (labeled PCIe in Figure 7.16) with the location and size of the buffer in main memory containing the data to be transferred.
- The I/O controller issues a read request that is routed to the memory controller hub (MCH), which accesses the data on DDR3 memory and puts it on the system ring for delivery to the I/O controller.
- The L3 cache is not involved in this transaction and one or more off-chip memory reads are required.
- For input, data arrives from the I/O controller and is delivered over the system ring to the MCH and written out to main memory.
- The MCH must also invalidate any L3 cache lines corresponding to the updated memory locations, one or more off-chip memory writes are required.
- If an application wants to access the new data, a main memory read is required.

Intel Xeon Multicore Processor

- For output, when the I/O controller issues a read request, the MCH first checks to see if the data is in the L3 cache.
- Causes the data to be evicted from cache, that is, the act of reading by an I/O device causes data to be evicted.
- The I/O operation proceeds efficiently because it does not require main memory access.
- But, if an application does need that data in the future, it must be read back into the L3 cache from main memory; the L3 cache is not involved.
- Thus, the performance improvement involves only output operations.
- Although the output transfer is directly from cache to the I/O controller, the term direct cache access is not used for this feature.

Cache-Related Performance Issues

- Network traffic is transmitted in the form of a sequence of protocol blocks, called packets or protocol data units.
- The lowest, or link, level protocol is Ethernet, so that each arriving and departing block of data consists of an Ethernet packet containing as payload the higher-level protocol packet.
- The higher-level protocols are the Internet Protocol (IP), operating on top of Ethernet, and the Transmission Control Protocol (TCP), operating on top of IP.
- Ethernet payload consists of a block of data with a TCP header and an IP header.
- For outgoing data, Ethernet packets are formed in a peripheral component (I/O controller or network interface controller (NIC)).
- For incoming traffic, the I/O controller strips off the Ethernet information and delivers the TCP/ IP packet to the host CPU.

Cache-Related Performance Issues

- For both outgoing and incoming traffic, the core, main memory, and cache are all involved.
- In a DMA scheme, when an application wishes to transmit data, it places that data in an application-assigned buffer in main memory.
- The core transfers this to a system buffer in main memory and creates the necessary TCP and IP headers, which are also buffered in system memory.
- The packet is then picked up via DMA for transfer via the NIC.
- When large volumes of protocol traffic are processed, two factors in this scenario degrade performance:
  - First, the core consumes valuable clock cycles in copying data between system and application buffers.
  - Second, because memory speeds have not kept up with CPU speeds, the core loses time waiting on memory reads and writes.
- In this way of processing protocol traffic, the cache does not help because the data and protocol headers are constantly changing and thus the cache must constantly be updated.

Packet Traffic Steps:

- Incoming
  - Packet arrives
  - DMA
  - NIC interrupts host
  - Retrieve descriptors and headers
  - Cache miss occurs
  - Header is processed
  - Payload transferred
- Outgoing
  - Packet transfer requested
  - Packet created
  - Output operation invoked
  - DMA transfer
  - NIC signals completion
  - Driver frees buffer

Direct Cache Access Strategies

- The strategy implemented as a prototype on a number of Intel Xeon processors, applies only to incoming network traffic.
- The DCA function in the memory controller sends a prefetch hint to the core as soon as the data is available in system memory.
- This enables the core to prefetch the data packet from the system buffer, thus avoiding cache misses and the associated waste of core cycles.
- Much more substantial gains can be realized by avoiding the system buffer in main memory altogether.
- The packet and packet descriptor information are accessed only once in the system buffer by the core.
- For incoming packets, the core reads the data from the buffer and transfers the packet payload to an application buffer, no need to access that data in the system buffer again.
- For outgoing packets, once the core has placed the data in the system buffer, it has no need to access that data again.
- **Cache injection**, a version of this more complete form of DCA is implemented in Intel's Xeon processor line, referred to as **Direct Data I/O**.

Direct Data I/O

- Intel Direct Data I/O (DDIO) is implemented on all of the Xeon E5 family of processors.
- Figure 7.17a shows the steps involved for a DMA operation.
  - The NIC initiates a memory write (1).
  - The NIC invalidates the cache lines corresponding to the system buffer (2).
  - The DMA operation is performed, depositing the packet directly into main memory (3).
  - After the appropriate core receives a DMA interrupt signal, the core can read the packet data from memory through the cache (4).
- Two techniques for dealing with an update to a cache line are:
  - **Write through:** All write operations are made to main memory and to the cache, ensuring that main memory is always valid. Any other core–cache module can monitor traffic to main memory to maintain consistency within its own local cache.
  - **Write back:** Updates are made only in the cache. When an update occurs, a dirty bit associated with the line is set. Then, when a block is replaced, it is written back to main memory if and only if the dirty bit is set.

Direct Data I/O

- DDIO uses the write-back strategy in the L3 cache.
- A cache write operation may encounter a cache miss, which is dealt with by one of two strategies:
  - **Write allocate:** The required line is loaded into the cache from main memory. The line in the cache is updated by the write operation, this scheme is typically used with the *write-back* method.
  - **Non-write allocate:** The block is modified directly in main memory. No change is made to the cache, used with the write-through method.
- The DDIO strategy for inbound transfers initiated by the NIC are:
  1. If there is a cache hit, the cache line is updated, but not main memory; this is the write-back strategy for a cache hit (write update).
  2. If there is a cache miss, the write operation occurs to a line in the cache that will not be written back to main memory (write allocate).

Direct Data I/O

- Figure 7.17b shows the operation for DDIO input:
  - The NIC initiates a memory write (1).
  - The NIC invalidates the cache lines corresponding to the system buffer and deposits the incoming data in the cache (2).
  - After the appropriate core receives a DCA interrupt signal, the core can read the packet data from the cache (3).
- Figure 7.17c shows the steps involved for a DMA operation for outbound packet transmission:
  - The TCP/IP protocol handler executing on the core reads data in from an application buffer and writes it out to a system buffer. These data access operations result in cache misses and cause data to be read from memory and into the L3 cache (1).
  - When the NIC receives notification for starting a transmit operation, it reads the data from the L3 cache and transmits it (2).
  - The cache access by the NIC causes the data to be evicted from the cache and written back to main memory (3).

Direct Data I/O

- Figure 7.17d shows the steps involved for a DDIO operation for packet transmission:
  - TCP/IP protocol handler creates the packet to be transmitted and stores it in allocated space in the L3 cache (1), but not in main memory (2).
  - The read operation initiated by the NIC is satisfied by data from the cache, without causing evictions to main memory.
- The DDIO strategy is effective for a network protocol application because the incoming data need not be retained for future use.
- The protocol application is going to write the data to an application buffer, and there is no need to temporarily store it in a system buffer.
- DDIO is efficient than DMA for both incoming and outgoing packets and is better to keep up with the high packet traffic rate.

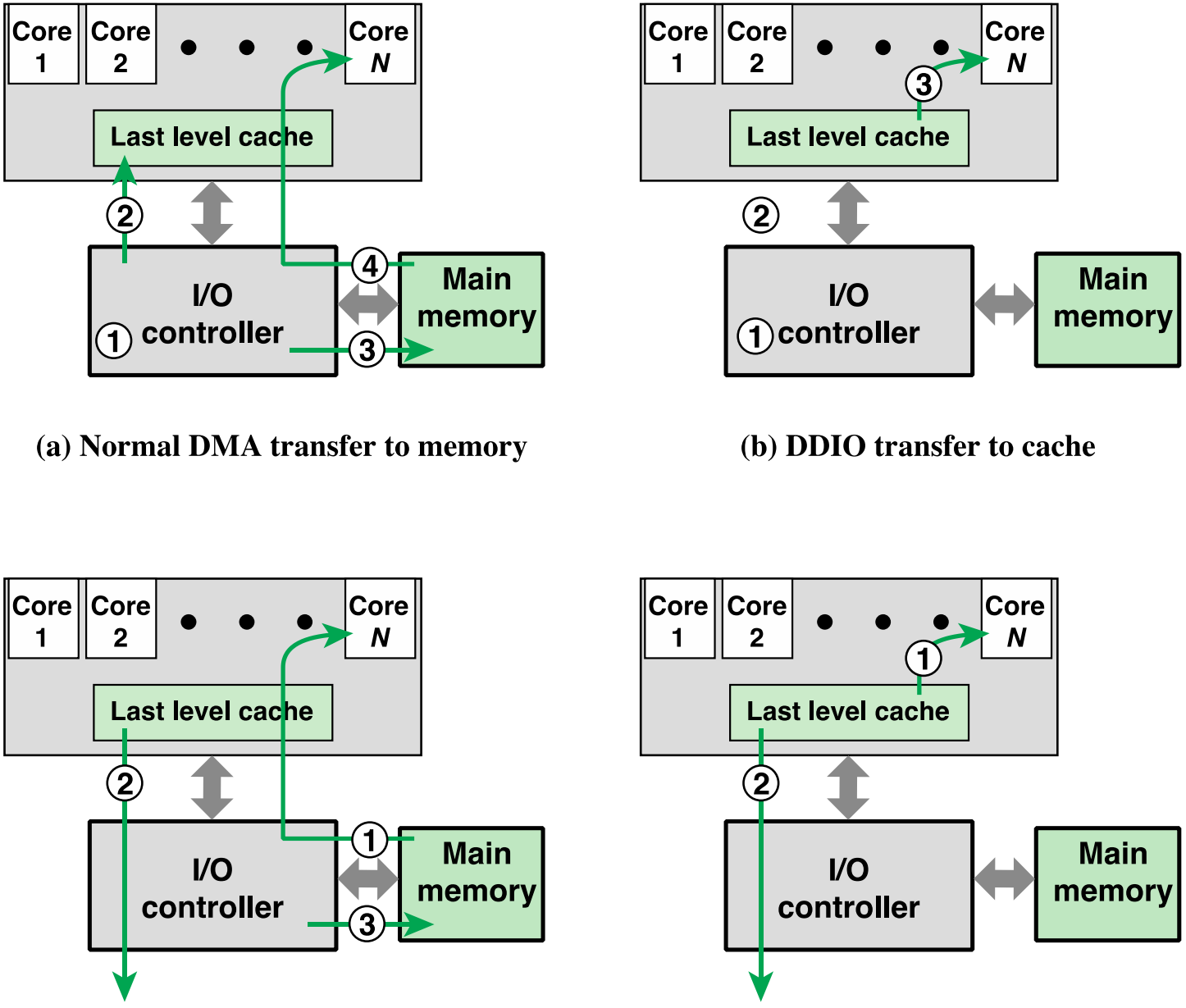


Figure 7.17 Comparison of DMA and DDIO

Input/ Output

- External Devices
- I/O Module
- Programmed I/O
- Interrupt-Driven I/O
- Direct Memory Access

I/O Channels and Processors

- External Interconnection Standards

I/O Channels and Processors

The Evolution of the I/O Function

- The evolutionary steps can be summarized as follows:
  1. The CPU directly controls a peripheral device.
  2. A controller or I/O module is added. The CPU uses programmed I/O without interrupts.
  3. Configuration as in step 2 is used, but now interrupts are employed. The CPU need not spend time waiting for an I/O operation to be performed, thus increasing efficiency.
  4. The I/O module is given direct access to memory via DMA. It can now move a block of data to or from memory without involving the CPU, except at the beginning and end of the transfer.
  5. The I/O module is enhanced to become a processor in its own right, with a specialized instruction set tailored for I/O
  6. The I/O module has a local memory of its own and is, in fact, a computer in its own right. With this architecture a large set of I/O devices can be controlled with minimal CPU involvement.

I/O Channels and Processors

Characteristics of I/O Channels

- An I/O channel has the ability to execute I/O instructions.
- The instructions are stored in main memory to be executed by a special-purpose processor in the I/O channel itself.
- The CPU initiates an I/O transfer by instructing the I/O channel to execute a program in memory.
- Two types of I/O channels are common, as illustrated in Figure 7.18.
- A *selector channe*l controls multiple high-speed devices and, at any one time, is dedicated to the transfer of data with one of those devices. Each device is handled by a *controller, or I/O module*. The I/O channel serves in place of the CPU in controlling these I/O controllers.
- A *multiplexor channel* can handle I/O with multiple devices at the same time. For low-speed devices, a *byte multiplexor* accepts or transmits characters as fast as possible to multiple devices. For example, the resultant character stream from three devices with different rates and individual streams $A_1A_2A_3A_4$ …, $B_1B_2B_3B_4$ …, and $C_1C_2C_3C_4$ … might be $A_1B_1C_1A_2C_2A_3B_2C_3A_4$, and so on. For high-speed devices, a *block multiplexor* interleaves blocks of data from several devices.

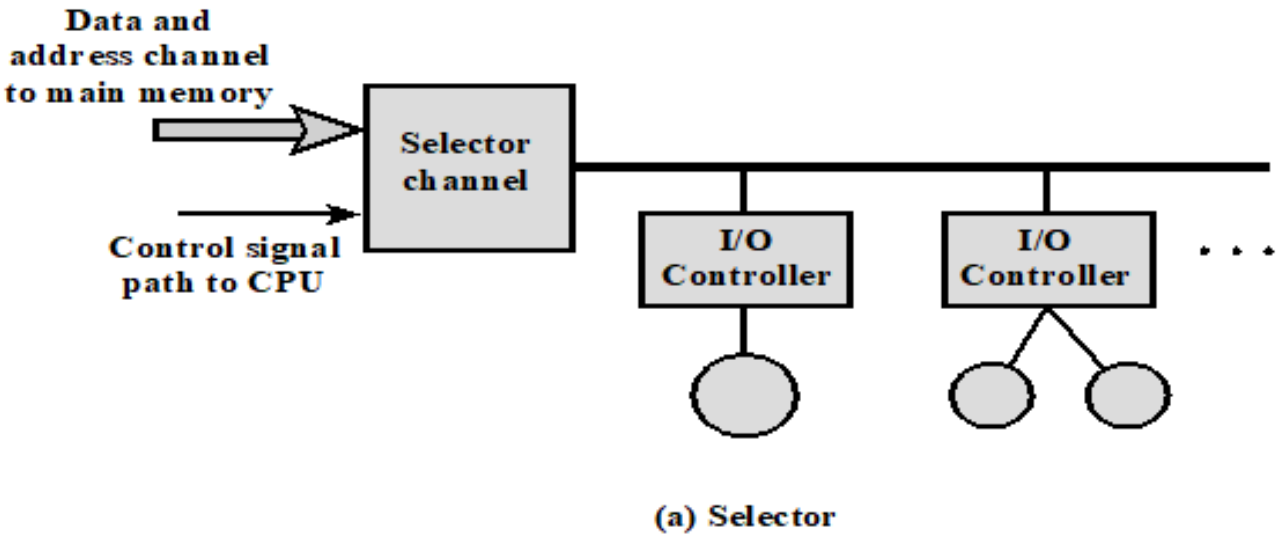


Figure 7.18 I/O Channel Architecture

Input/ Output

- External Devices
- I/O Modules
- Programmed I/O
- Interrupt-Driven I/O
- Direct Memory Access
- I/O Channels and Processors

- **External Interconnection Standards**

External Interconnection Standards

- Most widely used external interface standards to support I/O are:
  - Universal serial bus (USB)
  - FireWire serial bus
  - Small computer system interface (SCSI)
  - Thunderbolt
  - InfiniBand
  - PCI Express
  - Serial advanced technology attachment (SATA)
  - Ethernet
  - Wi-Fi

Universal Serial Bus (USB)

- Widely used for peripheral connections and high-speed I/O.
- Is the default interface for slower speed devices.
- Has gone through multiple generations:
  - USB 1.0: *Low Speed* data rate of 1.5 Mbps and a *Full Speed* rate of 12 Mbps
  - USB 2.0: Provides a data rate of 480 Mbps
  - USB 3.0
    - Higher speed bus called *SuperSpeed* in parallel with the USB 2.0 bus
    - Signaling speed of *SuperSpeed* is 5 Gbps, but due to signaling overhead the usable data rate is up to 4 Gbps
  - USB 3.1
    - Includes a faster transfer mode called *SuperSpeed+*
    - Achieves a signaling rate of 10 Gbps and a usable data rate of 9.7 Gbps
- A USB system is controlled by a root host controller which attaches to devices to create a local network with a hierarchical tree topology.

FireWire Serial Bus

- Developed as an alternative to small computer system interface (SCSI) to be used on smaller systems, such as personal computers, workstations, and servers.
- Objective was to meet the increasing demands for high I/O rates while avoiding the bulky and expensive I/O channel technologies developed for mainframe and supercomputer systems.
- IEEE standard 1394, for a High Performance Serial Bus.
- Uses a daisy chain configuration, with up to 63 devices connected off a single port.
- 1022 FireWire buses can be interconnected using bridges.
- Provides for hot plugging which makes it possible to connect and disconnect peripherals without having to power the computer system down or reconfigure the system.
- Provides for automatic configuration.
- No terminations and the system automatically performs a configuration function to assign addresses.

SCSI

- Small Computer System Interface (SCSI).
- A once common standard for connecting peripheral devices to small and medium-sized computers.
- Has lost popularity to USB and FireWire in smaller systems.
- High-speed versions remain popular for mass memory support on enterprise systems.
- Physical organization is a shared bus, which can support up to 16 or 32 devices, depending on the generation of the standard:
  - The bus provides for parallel transmission rather than serial, with a bus width of 16 bits on earlier generations and 32 bits on later generations.
  - Speeds range from 5 Mbps on the original SCSI-1 specification to 160 Mbps on SCSI-3 U3.

Thunderbolt

- Most recent and fastest peripheral connection technology to become available for general-purpose use.
- Developed by Intel with collaboration from Apple.
- The technology combines data, video, audio, and power into a single high-speed connection for peripherals such as hard drives, RAID (Redundant Array of Independent Disks) arrays, video-capture boxes, and network interfaces.
- Provides up to 10 Gbps throughput in each direction and up to 10 Watts of power to connected peripherals.

InfiniBand

- I/O specification aimed at the high-end server market.
- First version was released in early 2001.
- Heavily relied on by IBM zEnterprise series of mainframes.
- Standard describes an architecture and specifications for data flow among processors and intelligent I/O devices.
- Has become a popular interface for storage area networking and other large storage configurations.
- Enables servers, remote storage, and other network devices to be attached in a central fabric of switches and links.
- The switch-based architecture can connect up to 64,000 servers, storage systems, and networking devices.

InfiniBand

PCE Express:

- PCI Express is a high-speed bus system for connecting peripherals of a wide variety of types and speeds.

SATA:

- Serial ATA (Serial Advanced Technology Attachment) is an interface for disk storage systems.
- It provides data rates of up to 6 Gbps, with a maximum per device of 300 Mbps.
- SATA is widely used in desktop computers, and in industrial and embedded applications.

Ethernet

- The predominant wired networking technology.
- Has evolved to support data rates up to 100 Gbps and distances from a few meters to tens of km.
- Essential for supporting personal computers, workstations, servers, and massive data storage devices in organizations large and small.
- Began as an experimental bus-based 3-Mbps system.
- Has moved from bus-based to switch-based:
  - Data rate has periodically increased by an order of magnitude
  - There is a central switch with all of the devices connected directly to the switch
- Ethernet systems are currently available at speeds up to 100 Gbps.

Wi-Fi

- Wi-Fi is the predominant Wireless Internet access technology.
- Connects computers, tablets, smart phones, and other electronic devices.
- Wi-Fi in the enterprise has become an essential means of enhancing worker productivity and network effectiveness.
- Public hotspots have expanded dramatically to provide free Internet access in most public places.
- The IEEE 802.11 committee has been able to introduce standards for new versions of Wi-Fi at higher speeds:
  - 802.11 (1997): 2 Mbps (megabit per second, million bits per second)
  - 802.11a (1999): 54 Mbps
  - 802.11b (1999): 11 Mbps
  - 802.11n (1999): 600 Mbps
  - 802.11g (2003): 54 Mbps
  - 802.11ad (2012): 6.76 Gbps (billion bits per second)
  - 802.11ac (2014): 3.2 Gbps

Summary

- External devices
  - Keyboard/monitor
  - Disk drive
- I/O modules
  - Module function
  - I/O module structure
- Programmed I/O
  - Overview of programmed I/O
  - I/O commands/instructions
- Direct memory access
  - Drawbacks of programmed and interrupt-driven I/O
  - DMA function
  - Intel 8237A DMA controller
- Interrupt-driven I/O
  - Interrupt processing
  - Design issues
  - Intel 82C59A interrupt controller
  - Intel 82C55A programmable peripheral interface
- Direct Cache Access
  - DMA using shared last-level cache
  - Cache-related performance issues
  - Direct cache access strategies
  - Direct data I/O
- I/O channels and processors
  - The evolution of the I/O function
  - Characteristics of I/O channels

Thank You

Additional Formats

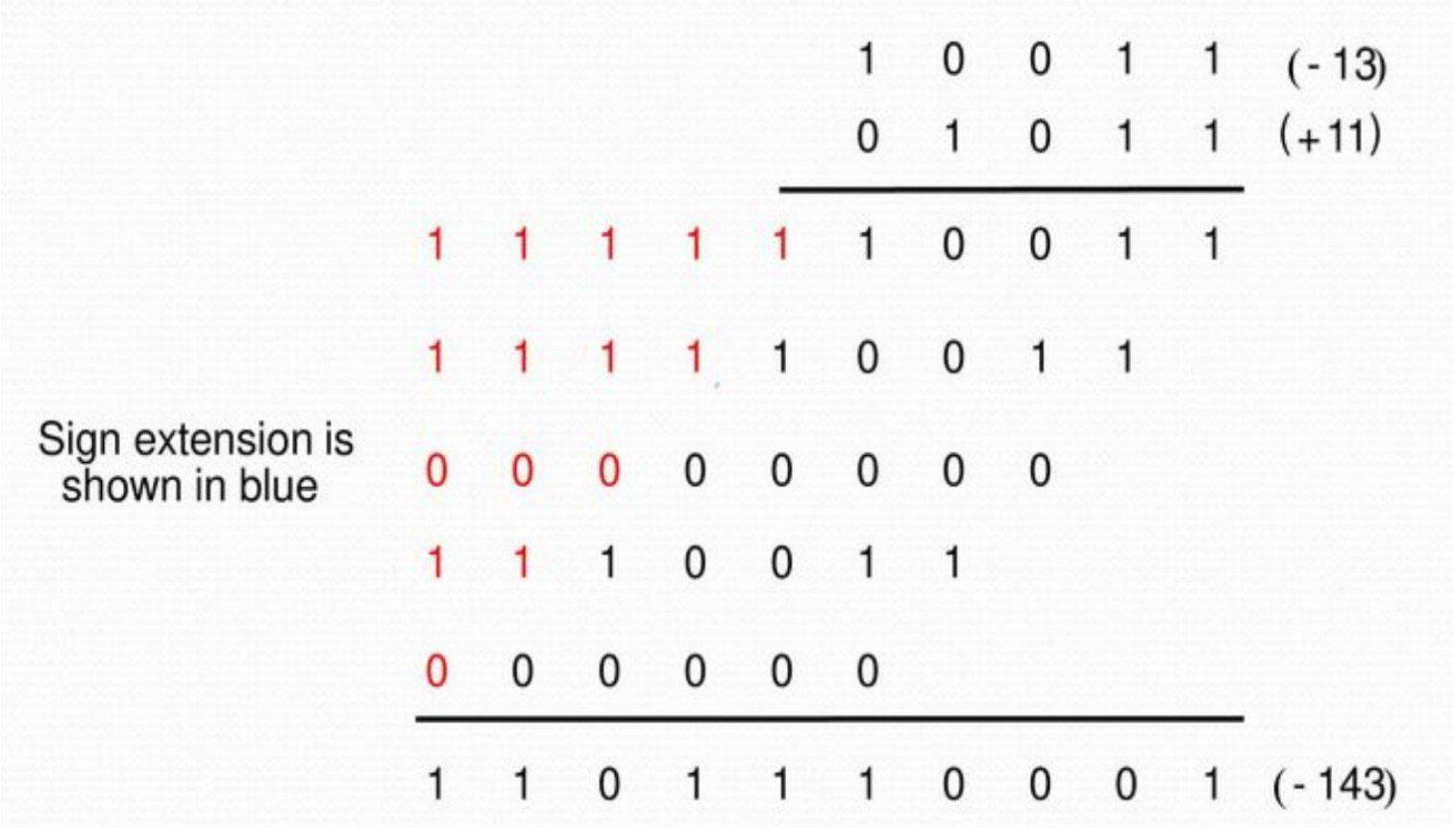
Extended Precision Formats

- Provide additional bits in the exponent (extended range) and in the significand (extended precision)
- Lessens the chance of a final result that has been contaminated by excessive roundoff error
- Lessens the chance of an intermediate overflow aborting a computation whose final result would have been representable in a basic format
- Affords some of the benefits of a larger basic format without incurring the time penalty usually associated with higher precision

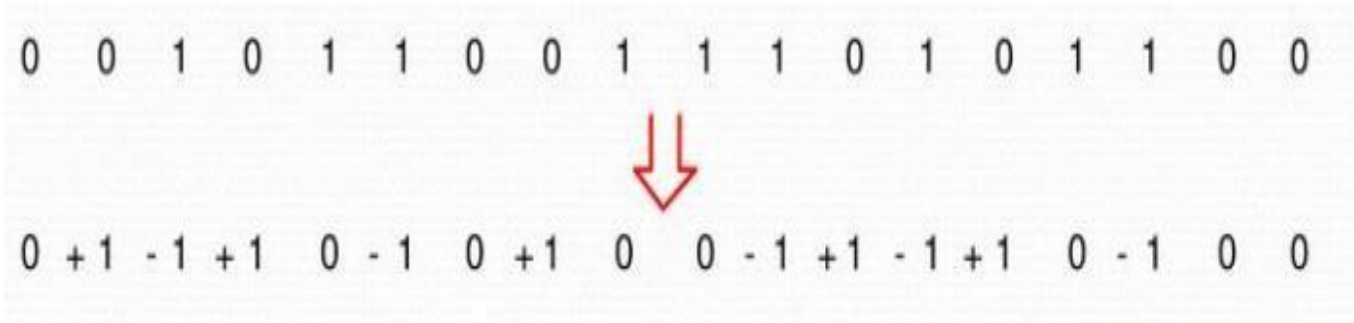
Extendable Precision Format

- Precision and range are defined under user control
- May be used for intermediate calculations but the standard places no constraint or format or length

Multiplication-Signed integers: Example



Booth Algorithm: Example

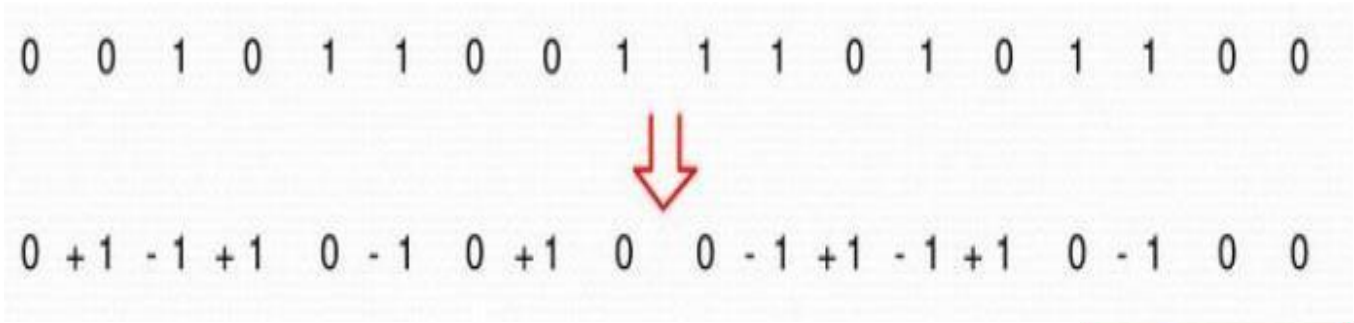


Multiplier		Version of multiplicand selected by bit
Bit *i*	Bit *i*-1	
0	0	0 × M
0	1	+1 × M
1	0	−1 × M
1	1	0 × M

```
                1 1 0 0
              x 0 1 0 1
               +1-1+1-1
          -----------------
        0 0 0 0 0 1 0 0
        1 1 1 1 1 0 0
        0 0 0 1 0 0
    +   1 1 1 0 0
-------------------------
        1 1 1 0 1 1 0 0
        0 0 0 1 0 1 0 0  = -20
```

```
   -4
x   5
-----
  -20
```

Booth Algorithm: Example




```
          0 1 0 1
        x 1 1 0 0
          0-1 0 0
  -------------------
  0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0
    1 1 1 0 1 1
+   0 0 0 0 0
-------------------
  1 1 1 0 1 1 0 0
  0 0 0 1 0 1 0 0  = -20
```

Multiplier		Version of multiplicand selected by bit
Bit *i*	Bit *i*-1	
0	0	0 × M
0	1	+1 × M
1	0	−1 × M
1	1	0 × M

```
   5
 x-4
-20
```