Arithmetic

Chapter 4

Addition/subtraction of signed numbers

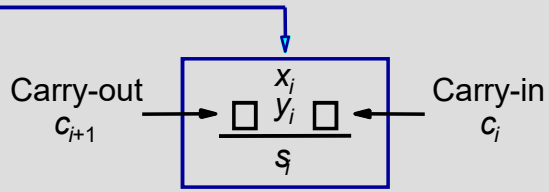
x_i	y_i	Carry-in c_i	Sums s_i	Carry-out c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$s_i = \bar{x}_i \bar{y}_i c_i + \bar{x}_i y_i \bar{c}_i + x_i \bar{y}_i \bar{c}_i + x_i y_i c_i = x_i \oplus y_i \oplus c_i$$

$$c_{i+1} = y_i c_i + x_i c_i + x_i y_i$$

Example:

$$\begin{array}{rcl} X & 7 & 0\ 1\ 1\ 1 \\ +\ Y & +\ 6 & +\ 0\ 1\ 1\ 0\ 0 \\ \hline Z & 13 & 1\ 1\ 0\ 1 \end{array}$$



Legend for stage i

At the i^{th} stage:

Input:

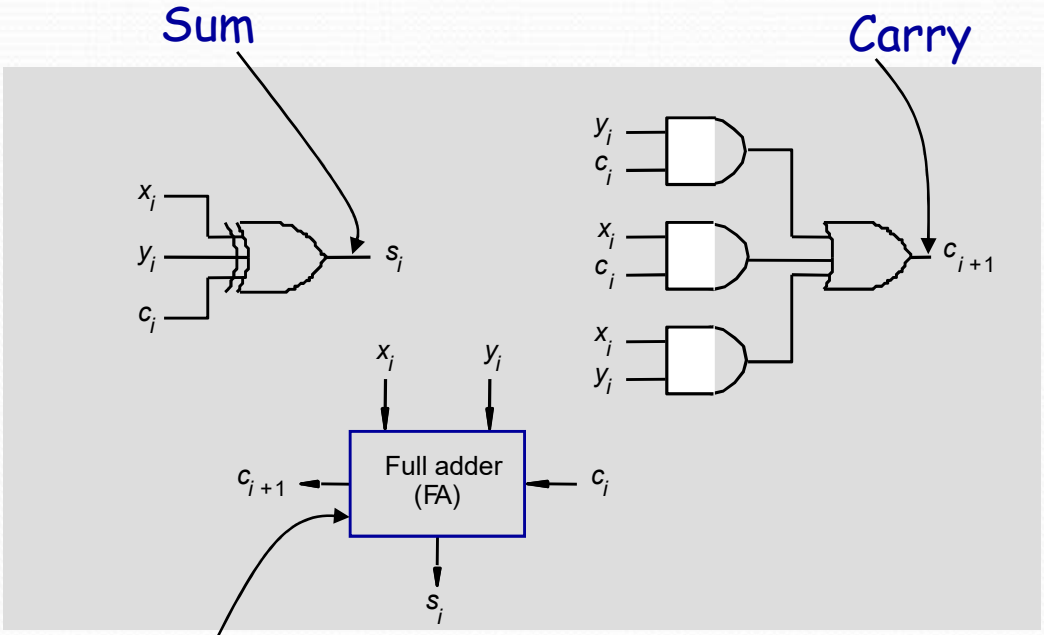
c_i is the carry-in

Output:

s_i is the sum

c_{i+1} carry-out to $(i+1)^{st}$ state

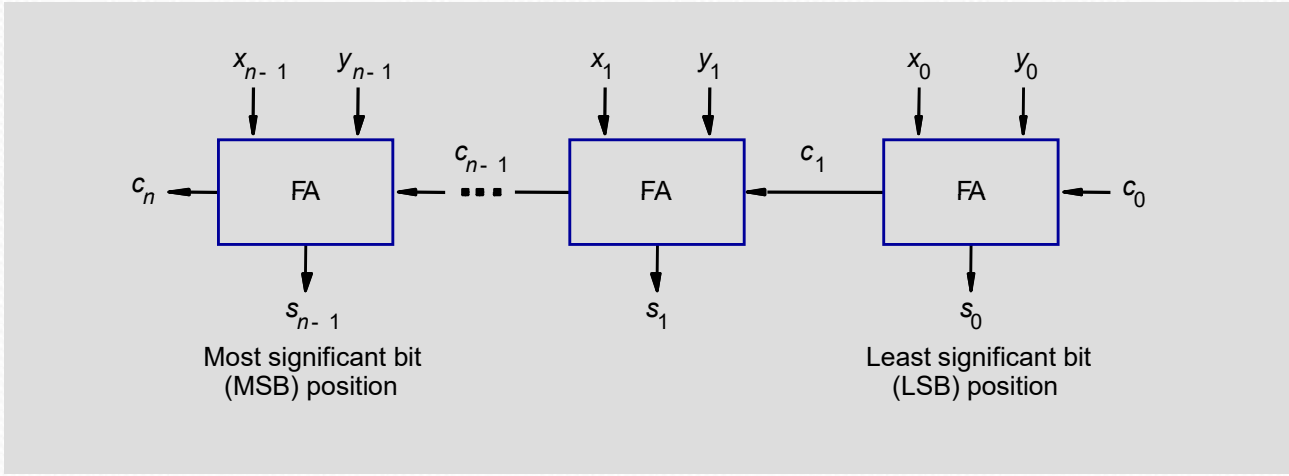
Addition logic for a single stage



Full Adder (FA): Symbol for the complete circuit for a single stage of addition.

n-bit adder

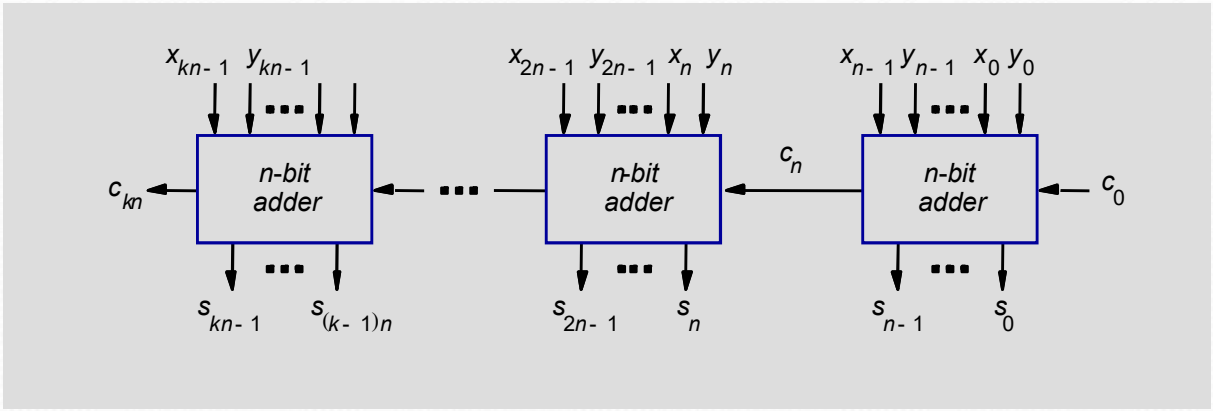
- Cascade *n* full adder (FA) blocks to form a *n*-bit adder.
- Carries propagate or ripple through this cascade, *n*-bit ripple carry adder.



Carry-in c_0 into the LSB position provides a convenient way to perform subtraction.

K n-bit adder

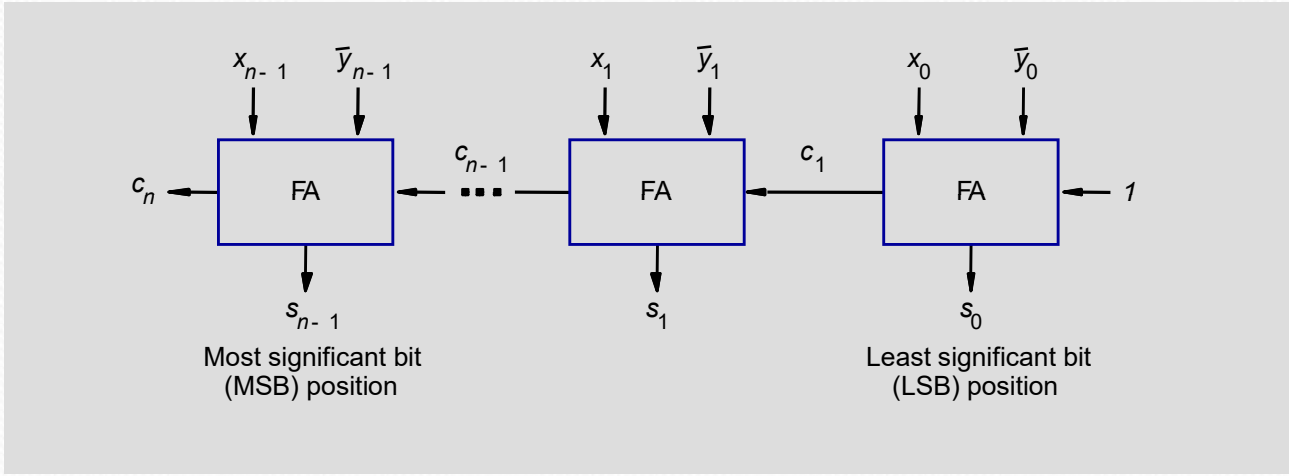
K n-bit numbers can be added by cascading k n-bit adders.



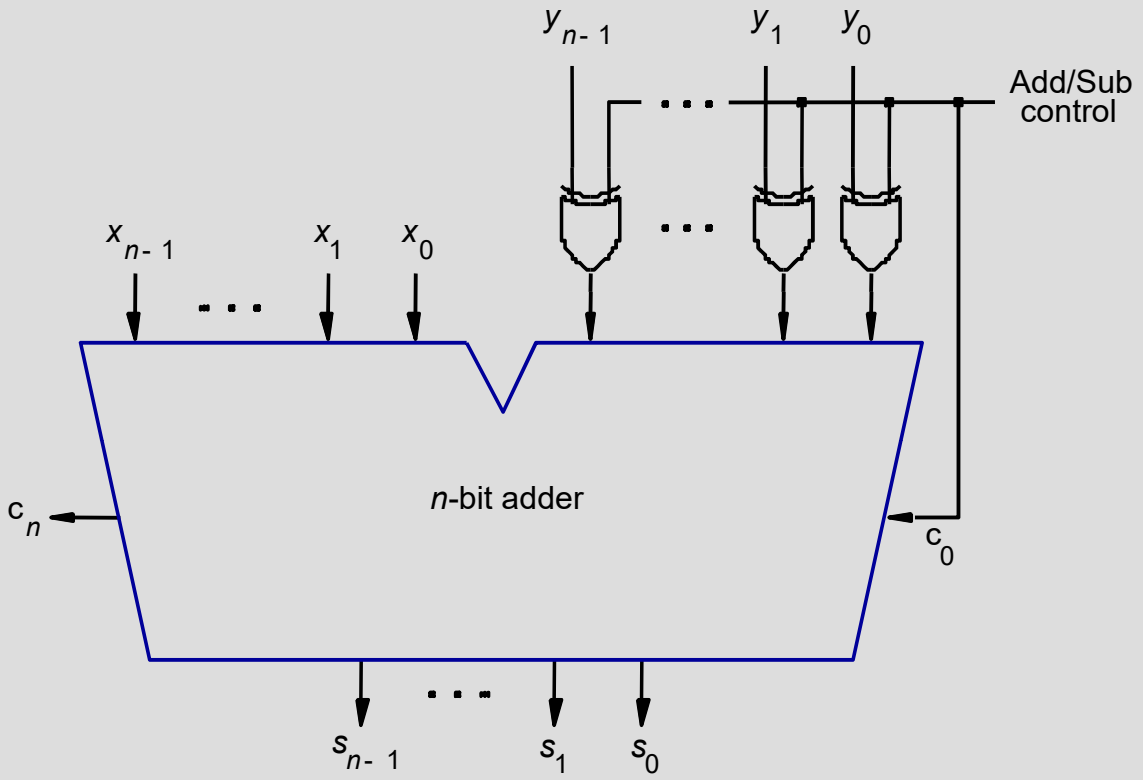
Each *n*-bit adder forms a block, so this is cascading of blocks.
Carries ripple or propagate through blocks, Blocked Ripple Carry Adder

n-bit subtractor

- Recall *X* – *Y* is equivalent to adding 2's complement of *Y* to *X*.
- 2's complement is equivalent to 1's complement + 1.
- $X - Y = X + \bar{Y} + 1$
- 2's complement of positive and negative numbers is computed similarly.



n-bit adder/subtractor (contd..)



- Add/sub control = 0, addition.
- Add/sub control = 1, subtraction.

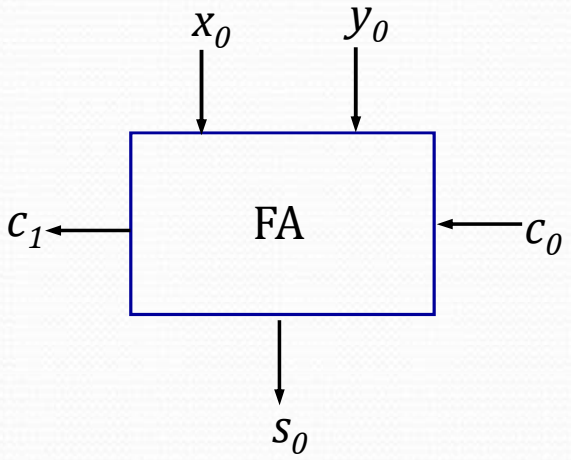
Detecting overflows

- Overflows can only occur when the sign of the two operands is the same.
- Overflow occurs if the sign of the result is different from the sign of the operands.
- Recall that the MSB represents the sign.
  - x_{n-1}, y_{n-1}, s_{n-1} represent the sign of operand *x*, operand *y* and result *s* respectively.
- Circuit to detect overflow can be implemented by the following logic expressions:

$$Overflow = x_{n-1} y_{n-1} \bar{s}_{n-1} + \bar{x}_{n-1} \bar{y}_{n-1} s_{n-1}$$

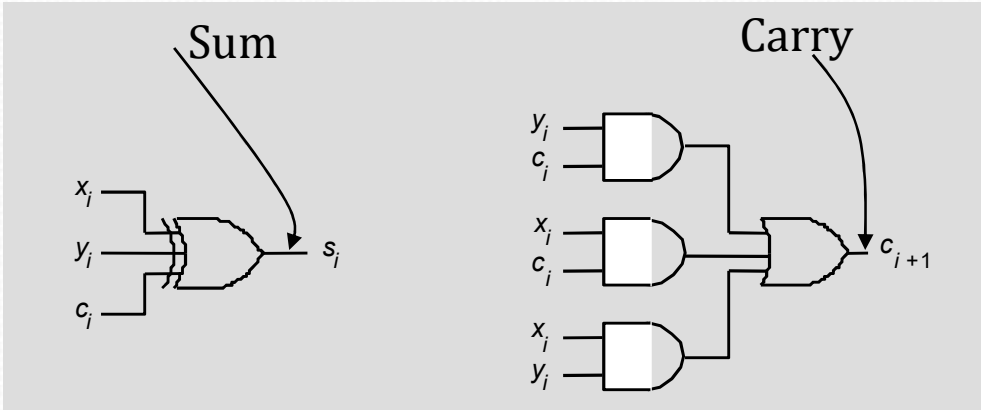
$$Overflow = c_n \oplus c_{n-1}$$

Computing the add time



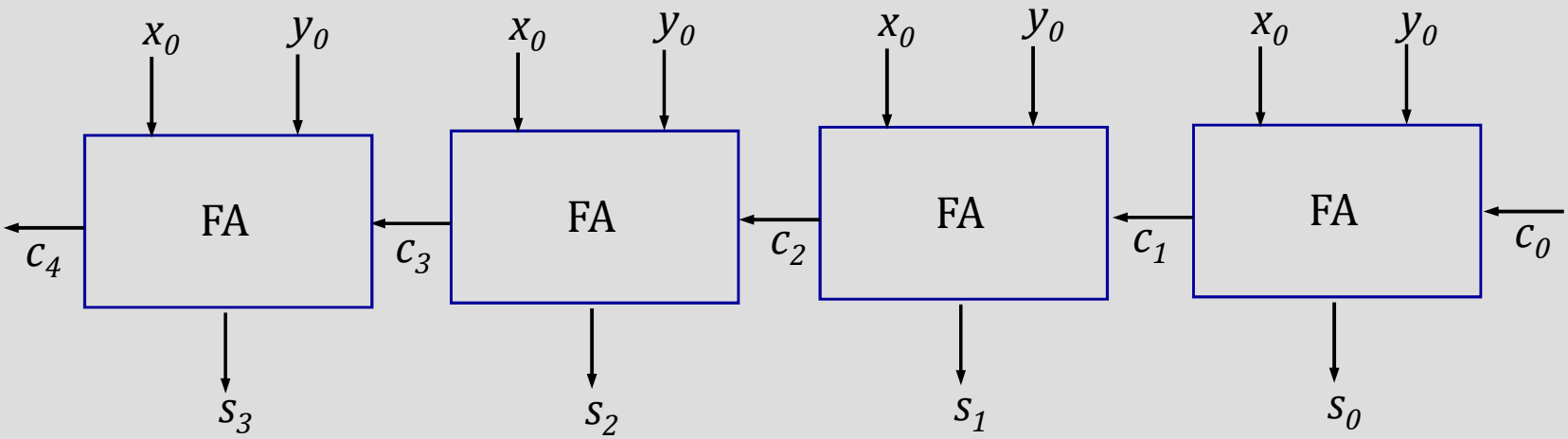
Consider 0^{th} stage:

- c_1 is available after 2 gate delays.
- s_1 is available after 1 gate delay.



Computing the add time (contd..)

Cascade of 4 Full Adders, or a 4-bit adder



- s_0 available after 1 gate delays, c_1 available after 2 gate delays.
- s_1 available after 3 gate delays, c_2 available after 4 gate delays.
- s_2 available after 5 gate delays, c_3 available after 6 gate delays.
- s_3 available after 7 gate delays, c_4 available after 8 gate delays.

For an n-bit adder, s_{n-1} is available after $2n-1$ gate delays
c_n is available after $2n$ gate delays.

Fast addition

Recall the equations:

$$s_i = x_i \oplus y_i \oplus c_i$$

$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

Second equation can be written as:

$$c_{i+1} = x_i y_i + (x_i + y_i) c_i$$

We can write:

$$c_{i+1} = G_i + P_i c_i$$

$$where\ G_i = x_i y_i\ and\ P_i = x_i + y_i$$

- G_i is called generate function and P_i is called propagate function
- G_i and P_i are computed only from x_i and y_i and not c_i, thus they can be computed in one gate delay after X and Y are applied to the inputs of an n-bit adder.

Carry lookahead

$$c_{i+1} = G_i + P_i c_i$$

$$c_i = G_{i-1} + P_{i-1} c_{i-1}$$

$$\Rightarrow c_{i+1} = G_i + P_i(G_{i-1} + P_{i-1} c_{i-1})$$

continuing

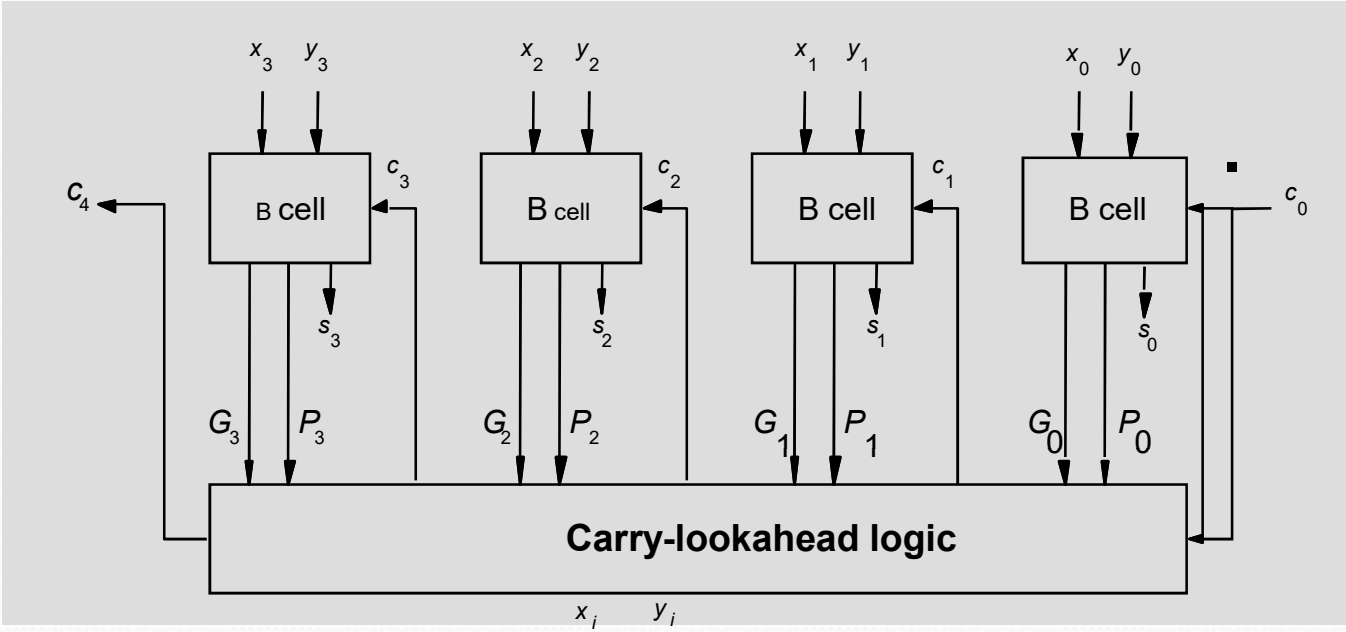
$$\Rightarrow c_{i+1} = G_i + P_i(G_{i-1} + P_{i-1}(G_{i-2} + P_{i-2} c_{i-2}))$$

until

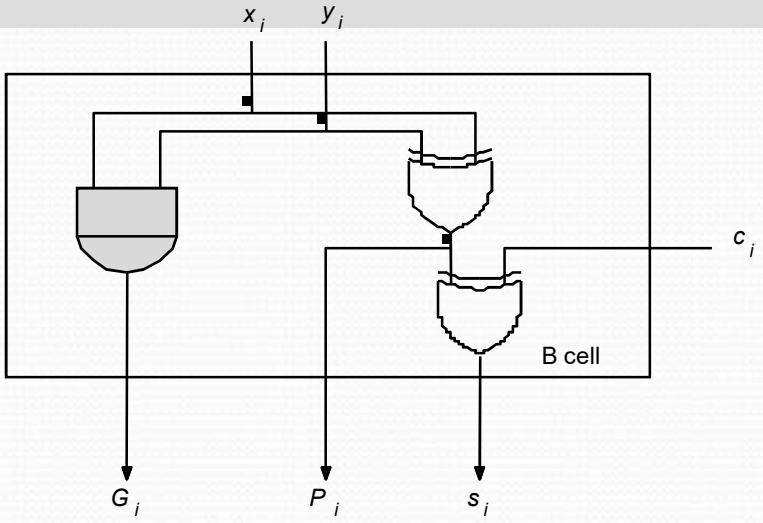
$$c_{i+1} = G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + .. + P_i P_{i-1} .. P_1 G_0 + P_i P_{i-1} ... P_0 c_0$$

- All carries can be obtained 3 gate delays after X, Y and c_0 are applied.
  - -One gate delay for P_i and G_i
  - -Two gate delays in the AND-OR circuit for c_{i+1}
- All sums can be obtained 1 gate delay after the carries are computed.
- Independent of n, n-bit addition requires only 4 gate delays.
- This is called Carry Lookahead adder.

Carry-lookahead adder



4-bit carry-lookahead adder



B-cell for a single stage

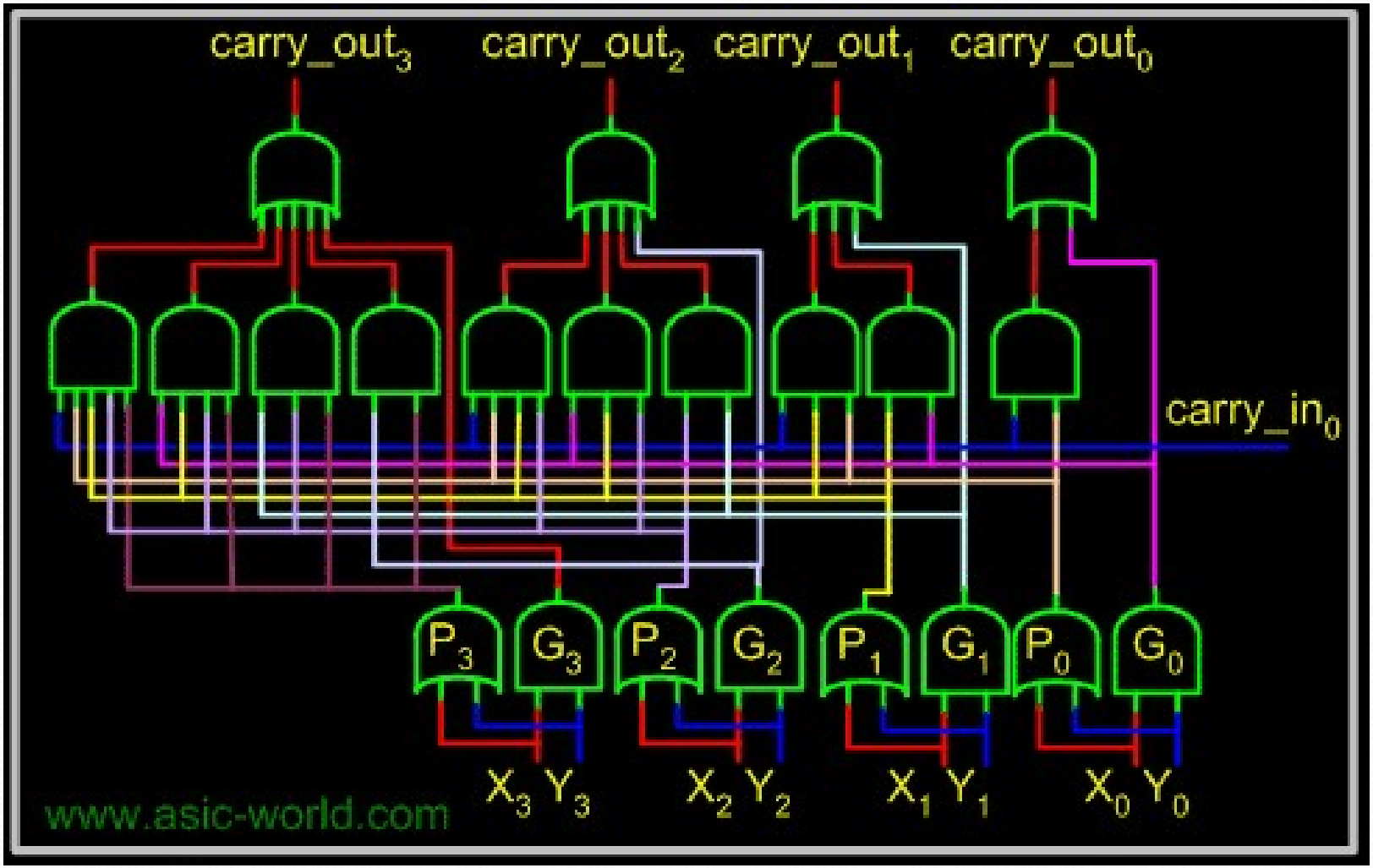
Carry lookahead adder (contd..)

- Performing *n*-bit addition in 4 gate delays independent of *n* is good only theoretically because of fan-in constraints.

$$c_{i+1} = G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + .. + P_i P_{i-1} .. P_1 G_0 + P_i P_{i-1} ... P_0 c_0$$

- Last AND gate and OR gate require a fan-in of (n+1) for a n-bit adder.
  - For a 4-bit adder ($n=4$) fan-in of 5 is required.
  - Practical limit for most gates.
- In order to add operands longer than 4 bits, we can cascade 4-bit Carry-Lookahead adders. Cascade of Carry-Lookahead adders is called Blocked Carry-Lookahead adder.

4-bit carry-lookahead Adder



Blocked Carry-Lookahead adder

Carry-out from a 4-bit block can be given as:

$$c_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0c_0$$

Rewrite this as:

$$P_0^I = P_3P_2P_1P_0$$

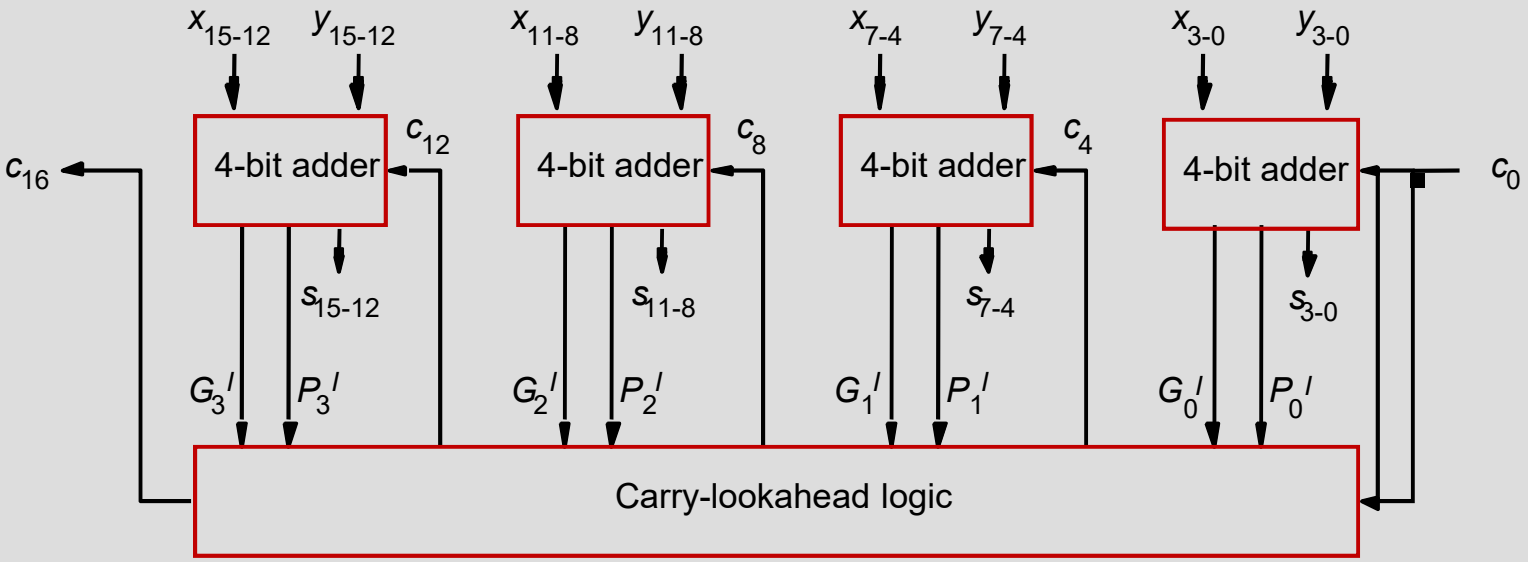
$$G_0^I = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$$

Subscript I denotes the blocked carry lookahead and identifies the block.

Cascade 4 *4*-bit adders, c_{16} can be expressed as:

$$c_{16} = G_3^I + P_3^I G_2^I + P_3^I P_2^I G_1^I + P_3^I P_2^I P_1^0 G_0^I + P_3^I P_2^I P_1^0 P_0^0 c_0$$

Blocked Carry-Lookahead adder

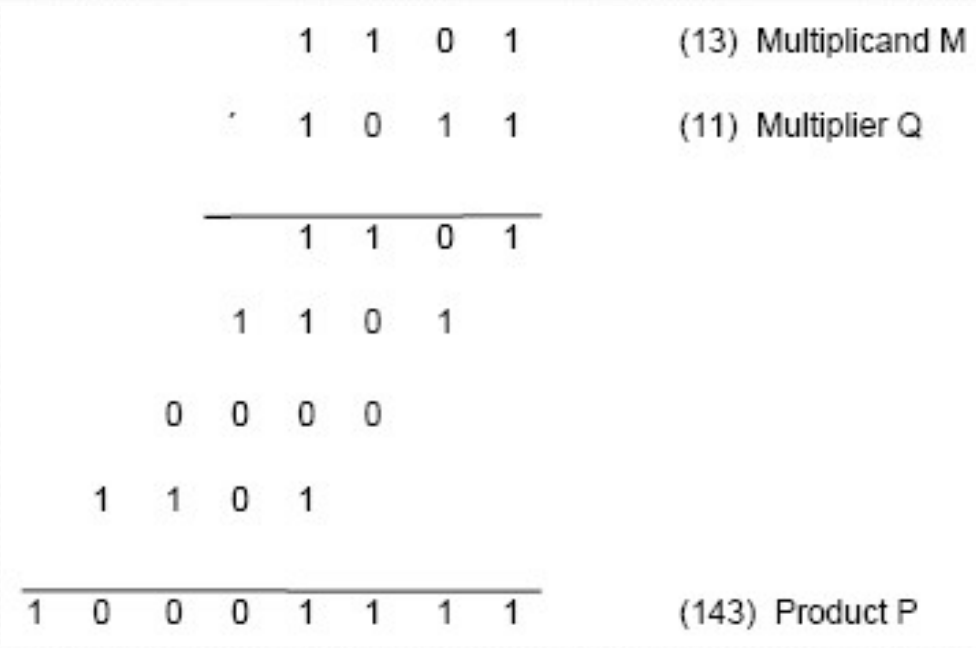


After x_i, y_i and c_0 are applied as inputs:

- G_i and P_i for each stage are available after 1 gate delay.
- P^I is available after 2 and G^I after 3 gate delays.
- All carries are available after 5 gate delays.
- c_{16} is available after 5 gate delays.
- s_{15} which depends on c_{12} is available after 8 (5+3)gate delays (Recall that for a 4-bit carry lookahead adder, the last sum bit is available 3 gate delays after all inputs are available)

Multiplication

Multiplication of unsigned numbers



Product of 2 *n*-bit numbers is at most a *2n*-bit number.

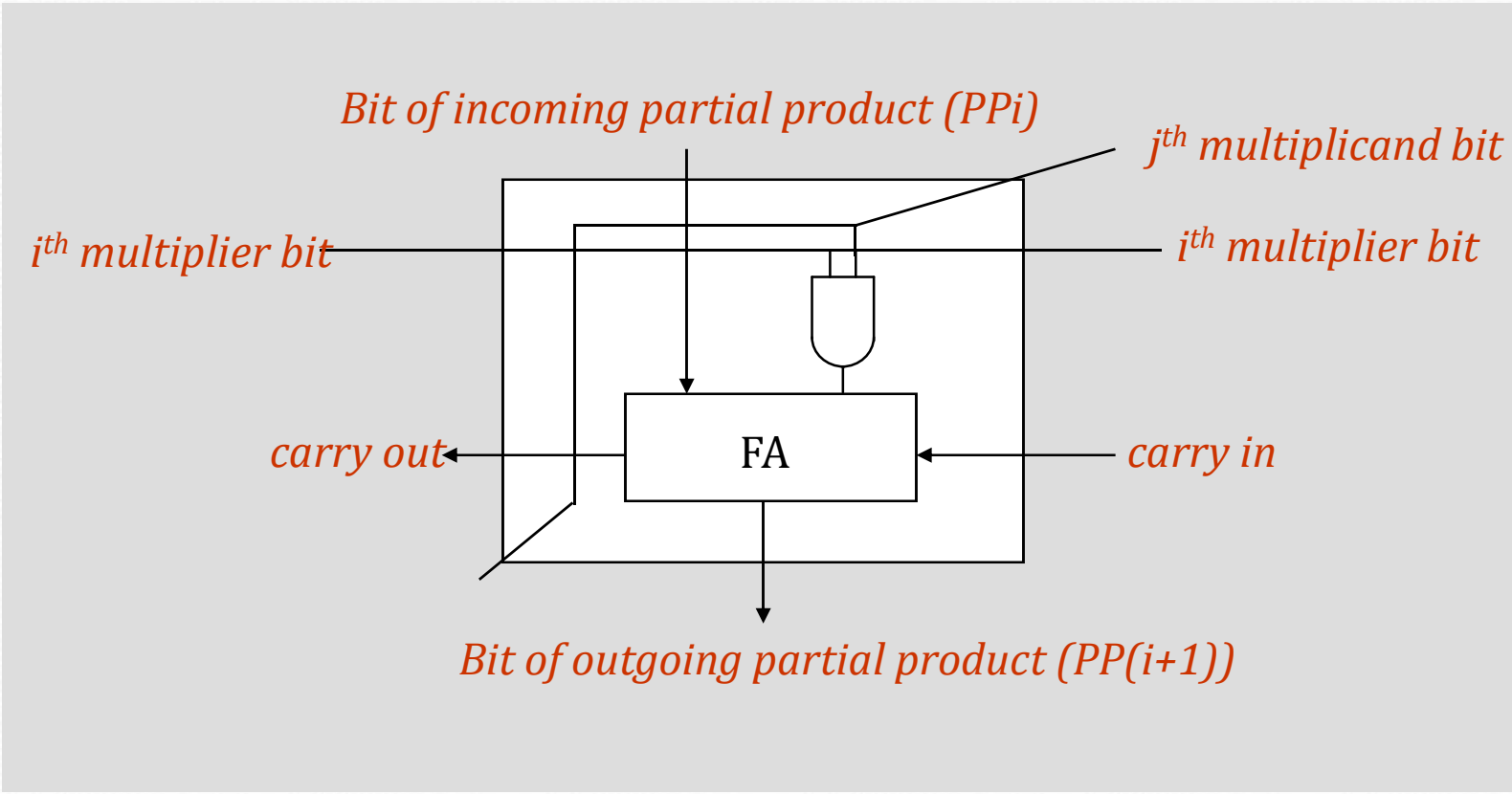
Unsigned multiplication can be viewed as addition of shifted versions of the multiplicand.

Multiplication of unsigned numbers (contd..)

- We added the partial products at end.
  - Alternative would be to add the partial products at each stage.
- Rules to implement multiplication are:
  - If the i^{th} bit of the multiplier is 1, shift the multiplicand and add the shifted multiplicand to the current value of the partial product.
  - Hand over the partial product to the next stage
  - Value of the partial product at the start stage is 0.

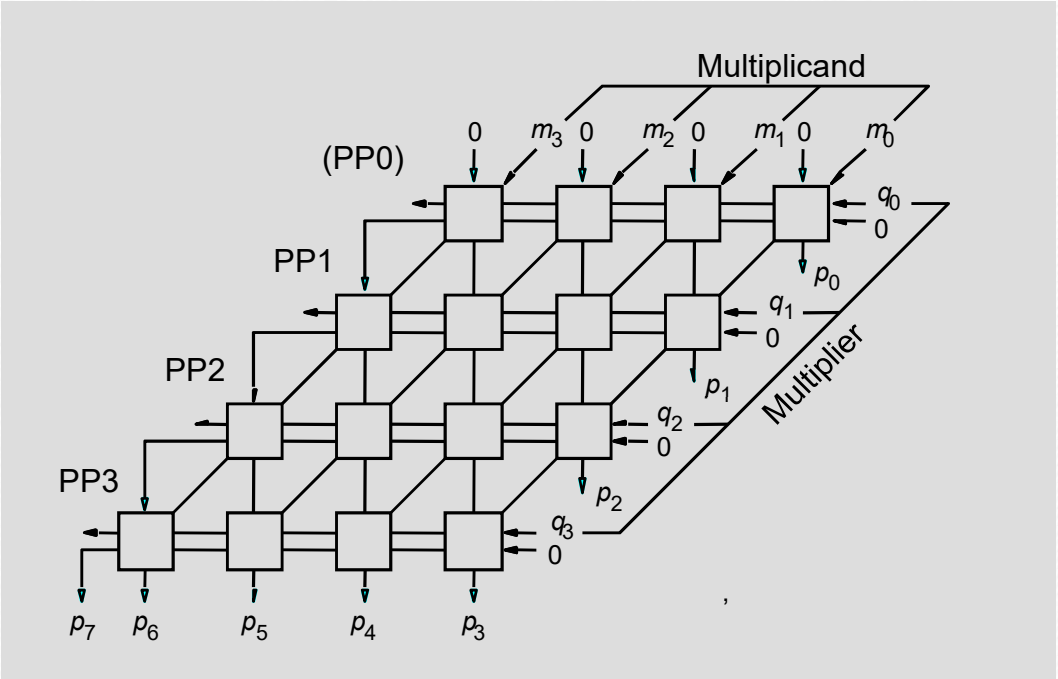
Multiplication of unsigned numbers

Typical multiplication cell



Combinatorial array multiplier

Combinatorial array multiplier



Product is: $p_7, p_6 .. p_0$

Multiplicand is shifted by displacing it through an array of adders.

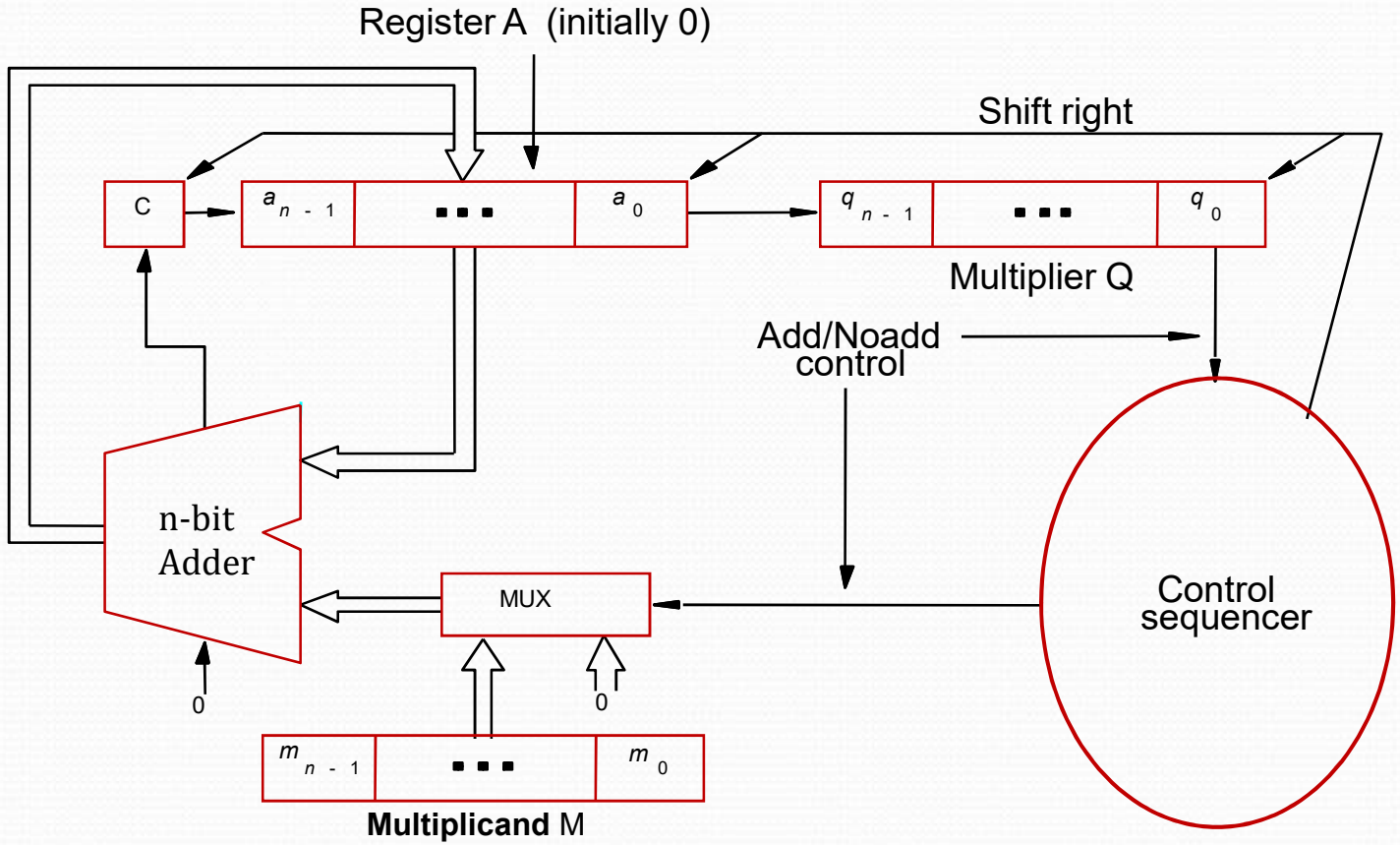
Combinatorial array multiplier (contd..)

- Combinatorial array multipliers are:
  - Extremely inefficient.
  - Have a high gate count for multiplying numbers of practical size such as 32-bit or 64-bit numbers.
  - Perform only one function, namely, unsigned integer product.
- Improve gate efficiency by using a mixture of combinatorial array techniques and sequential techniques requiring less combinational logic.

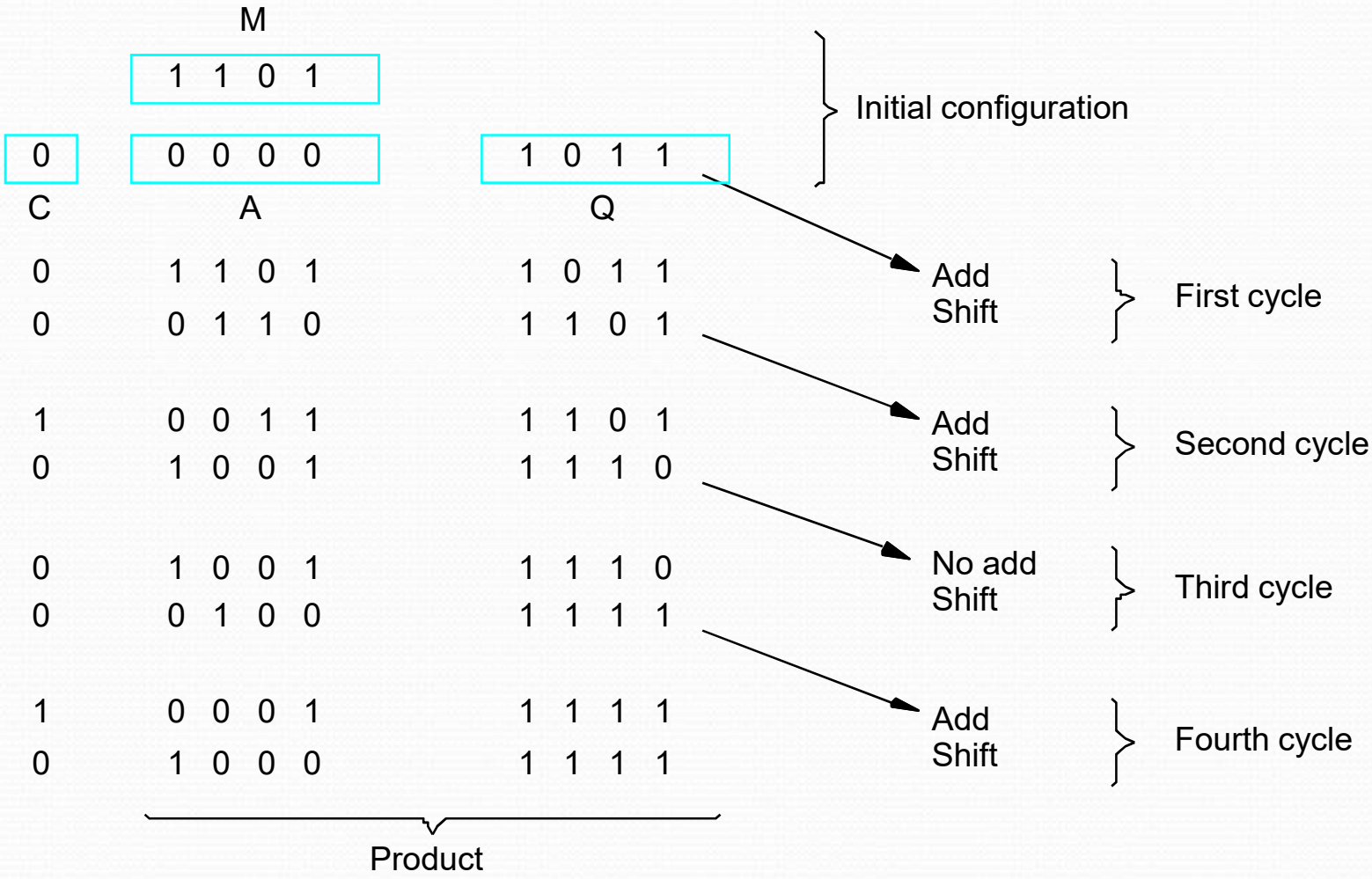
Sequential multiplication

- Recall the rule for generating partial products:
  - If the ith bit of the multiplier is 1, add the appropriately shifted multiplicand to the current partial product.
  - Multiplicand has been shifted left when added to the partial product.
- However, adding a left-shifted multiplicand to an unshifted partial product is equivalent to adding an unshifted multiplicand to a right-shifted partial product.

Sequential Circuit Multiplier



Sequential multiplication (contd..)



Signed Multiplication

Signed Multiplication

- Considering 2's-complement signed operands, what will happen to (-13)×(+11) if following the same method of unsigned multiplication?

					1	0	0	1	1	(- 13)
					0	1	0	1	1	(+ 11)
1	1	1	1	1	1	0	0	1	1	
1	1	1	1	1	0	0	1	1		
0	0	0	0	0	0	0	0			
1	1	1	0	0	1	1				
0	0	0	0	0	0					
1	1	0	1	1	1	0	0	0	1	(- 143)

Sign extension is shown in blue

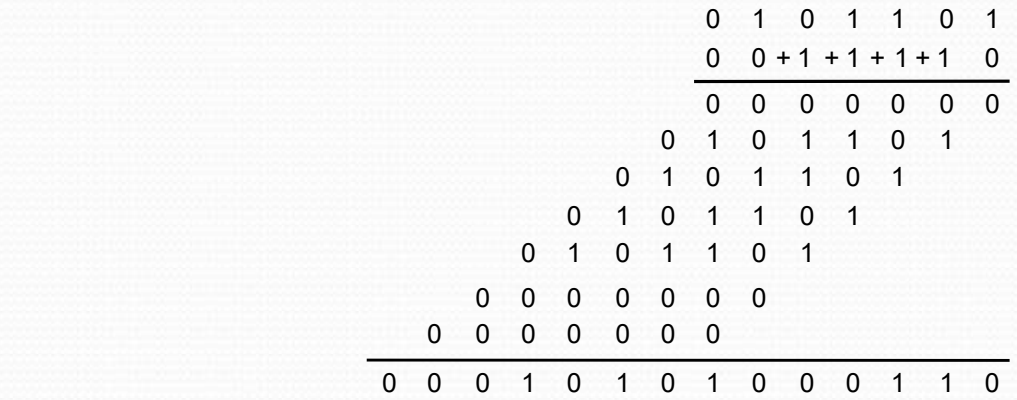
Sign extension of negative multiplicand.

Signed Multiplication

- For a negative multiplier, a straightforward solution is to form the 2's-complement of both the multiplier and the multiplicand and proceed as in the case of a positive multiplier.
- This is possible because complementation of both operands does not change the value or the sign of the product.
- A technique that works equally well for both negative and positive multipliers – Booth algorithm.

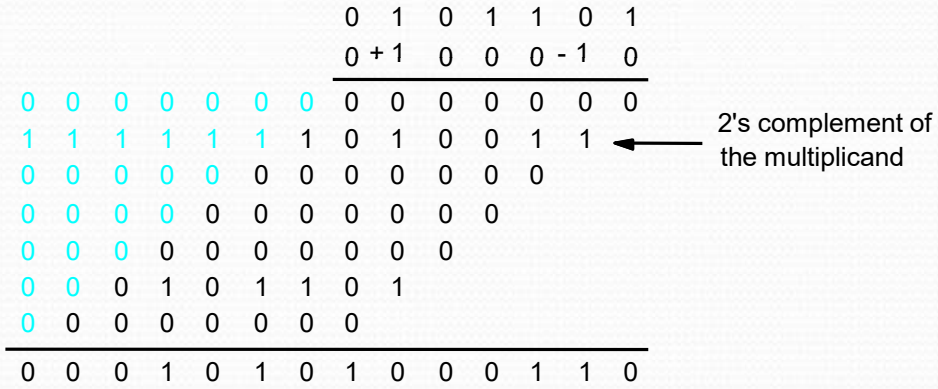
Booth Algorithm

- Consider in a multiplication, the multiplier is positive 0011110, how many appropriately shifted versions of the multiplicand are added in a standard procedure?



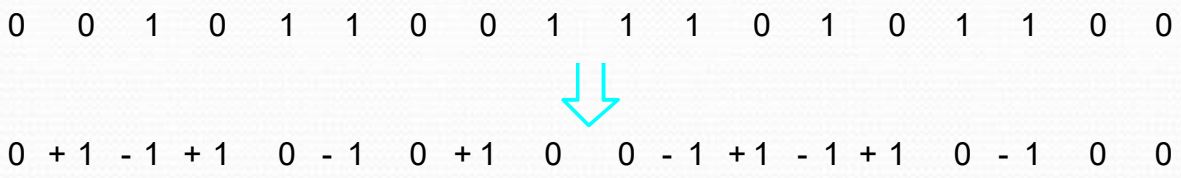
Booth Algorithm

- Since 0011110 = 0100000 – 0000010, if we use the expression to the right, what will happen?



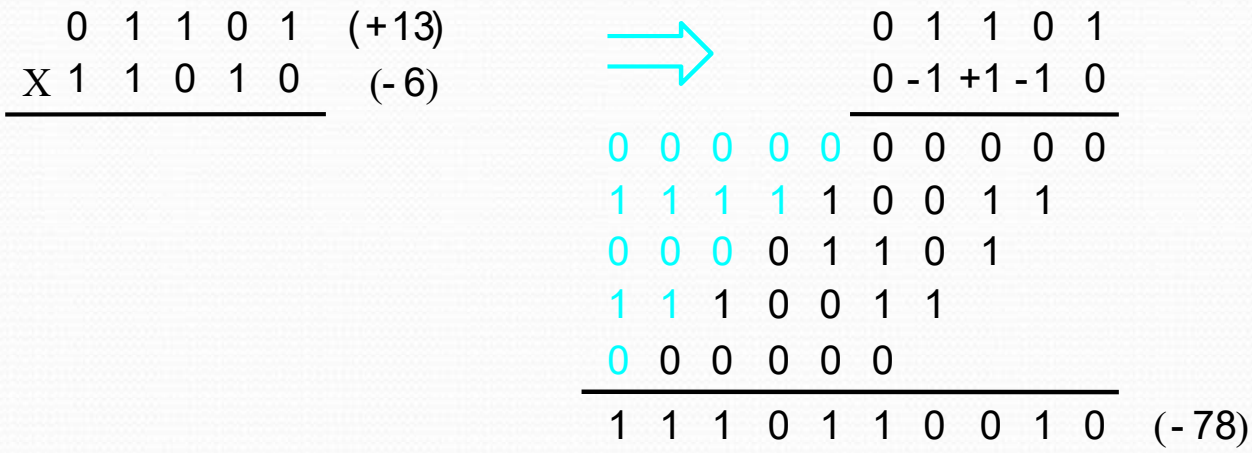
Booth Algorithm

- In general, in the Booth scheme, -1 times the shifted multiplicand is selected when moving from 0 to 1, and +1 times the shifted multiplicand is selected when moving from 1 to 0, as the multiplier is scanned from right to left.



Booth recoding of a multiplier.

Booth Algorithm



Booth multiplication with a negative multiplier.

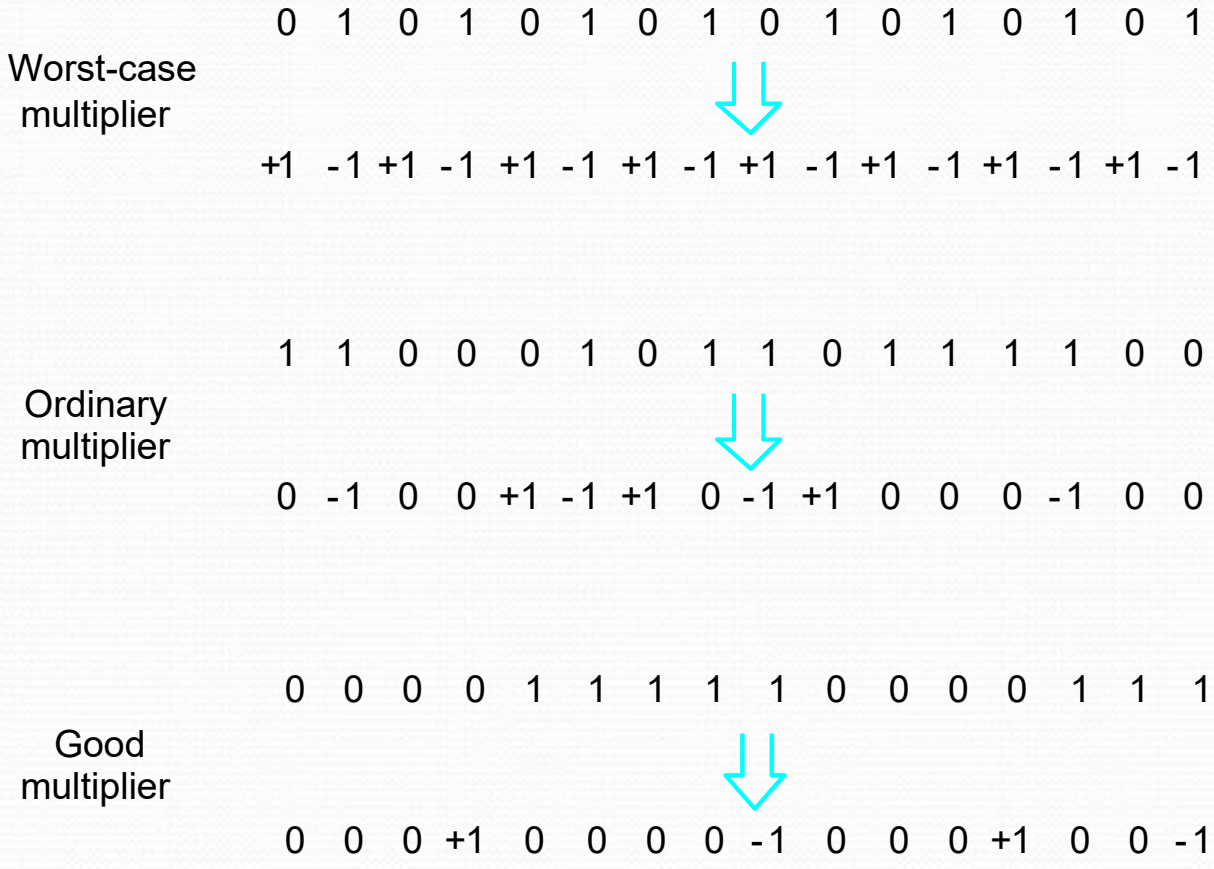
Booth Algorithm

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

Booth multiplier recoding table.

Booth Algorithm

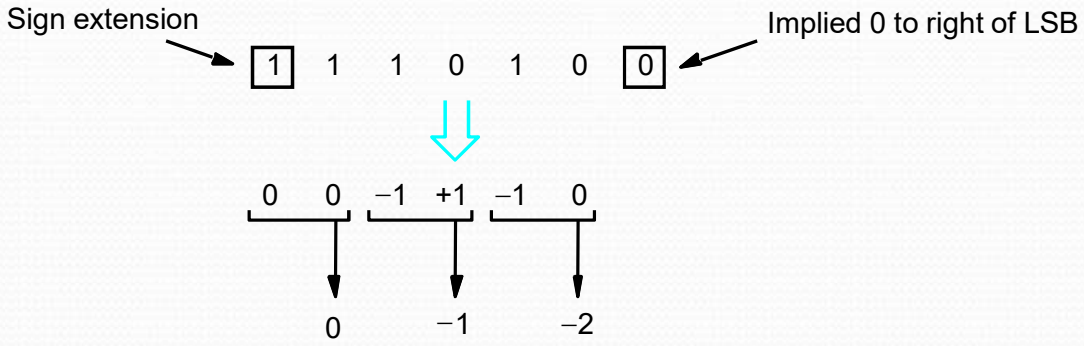
- Best case – a long string of 1's (skipping over 1s)
- Worst case – 0's and 1's are alternating



Fast Multiplication

Bit-Pair Recoding of Multipliers

- Bit-pair recoding halves the maximum number of summands (versions of the multiplicand).



(a) Example of bit-pair recoding derived from Booth recoding

Bit-Pair Recoding of Multipliers

Multiplier bit-pair		Multiplier bit on the right	Multiplicand selected at position i
$i+1$	i	$i-1$	
0	0	0	0 X M
0	0	1	+1 X M
0	1	0	+1 X M
0	1	1	+2 X M
1	0	0	−2 X M
1	0	1	−1 X M
1	1	0	−1 X M
1	1	1	0 X M

(b) Table of multiplicand selection decisions

Bit-Pair Recoding of Multipliers

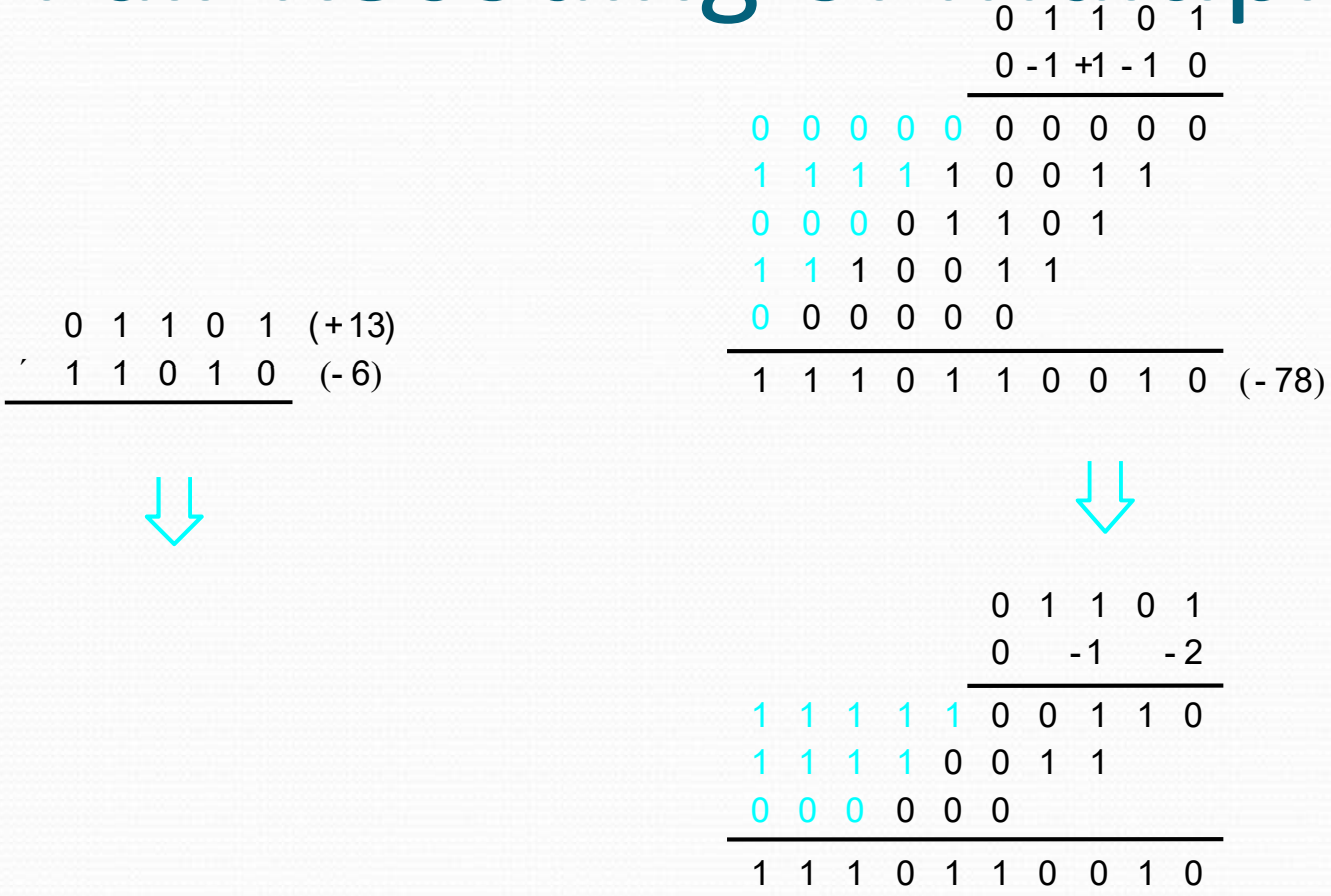
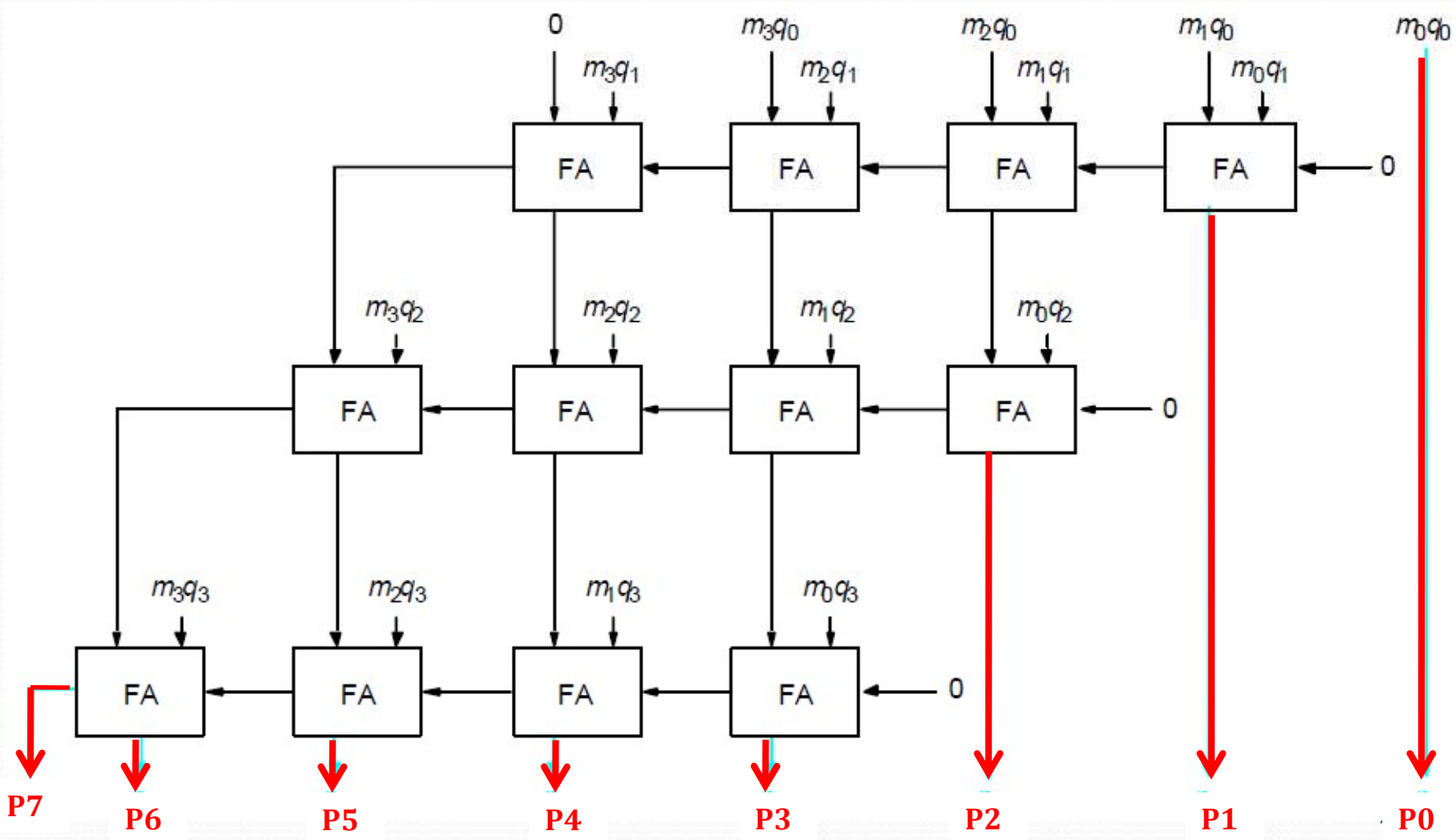


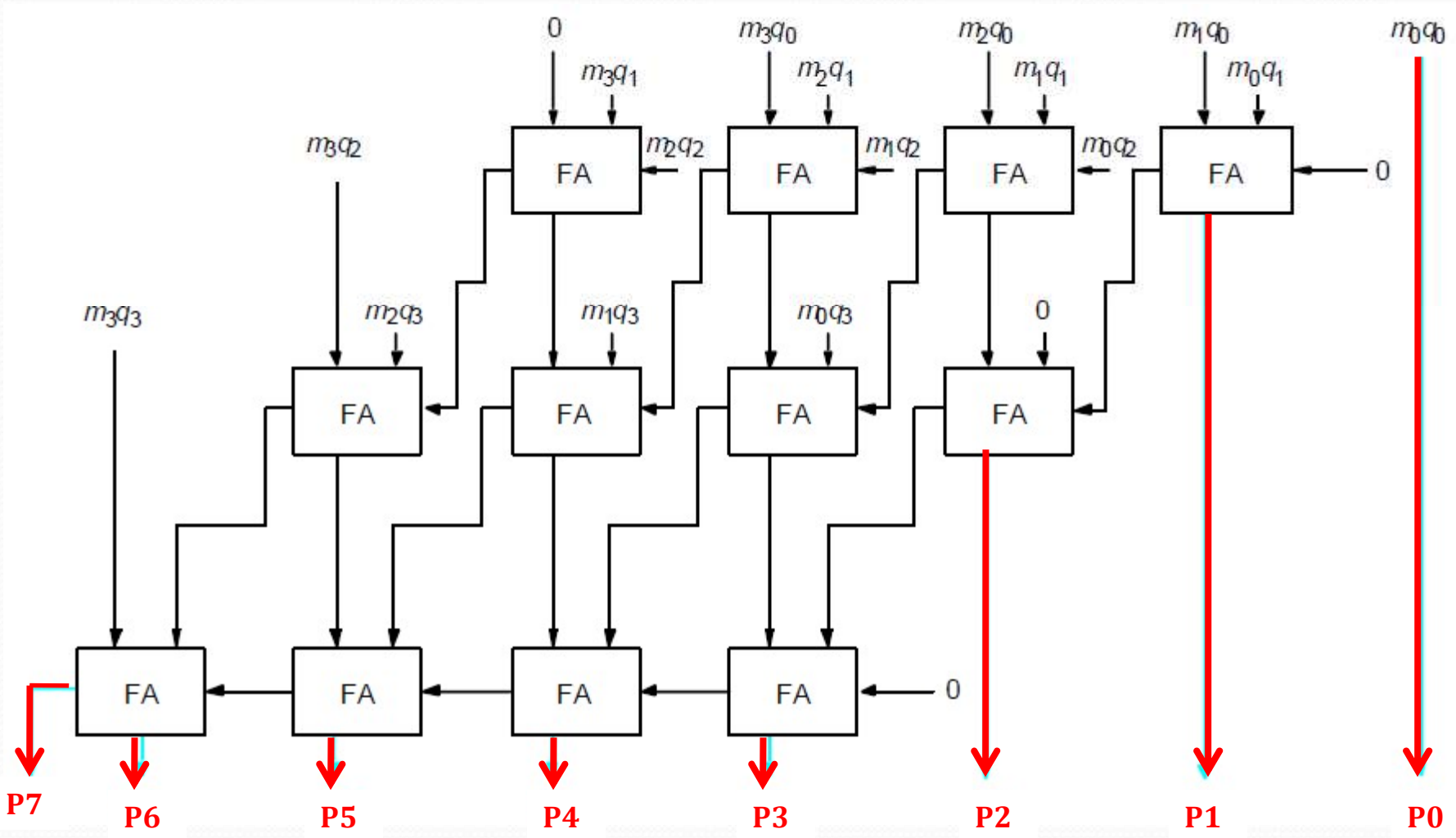
Figure 6.15. Multiplication requiring only *n*/2 summands.

Carry-Save Addition of Summands

- CSA speeds up the addition process.



Carry-Save Addition of Summands(Cont.,)



Carry-Save Addition of Summands(Cont.,)

- Consider the addition of many summands, we can:

- Group the summands in threes and perform carry-save addition on each of these groups in parallel to generate a set of S and C vectors in one full-adder delay
- Group all of the S and C vectors into threes, and perform carry-save addition on them, generating a further set of S and C vectors in one more full-adder delay
- Continue with this process until there are only two vectors remaining
- They can be added in a RCA or CLA to produce the desired product

Carry-Save Addition of Summands

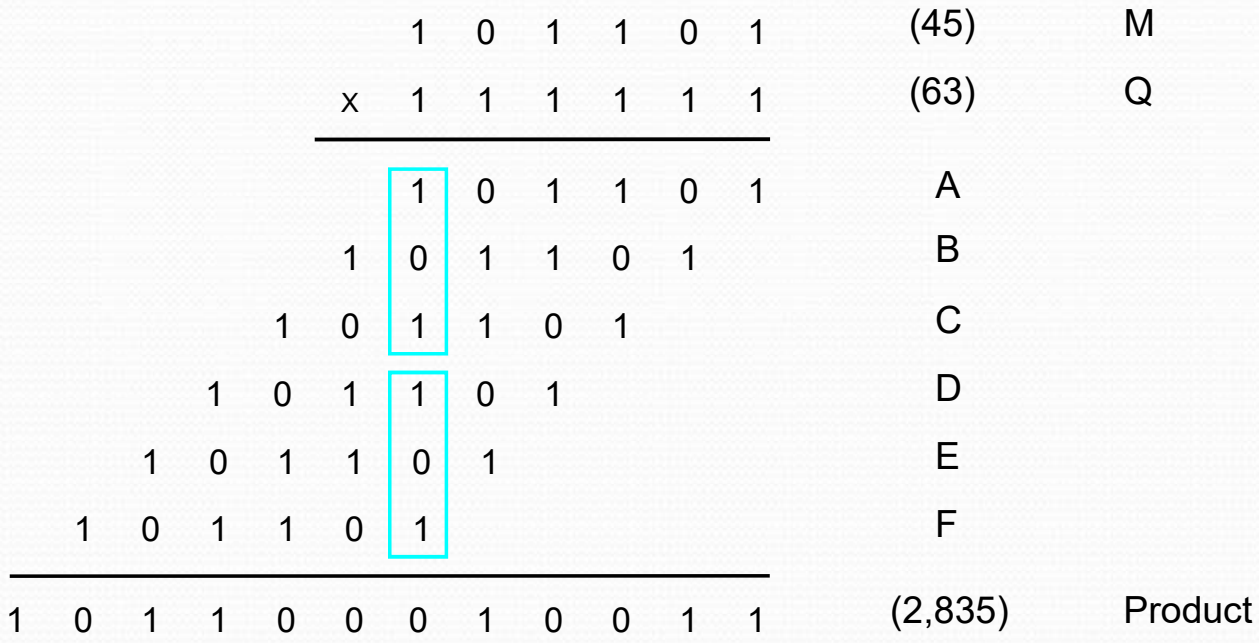


Figure 6.17. A multiplication example used to illustrate carry-save addition as shown in Figure 6.18.

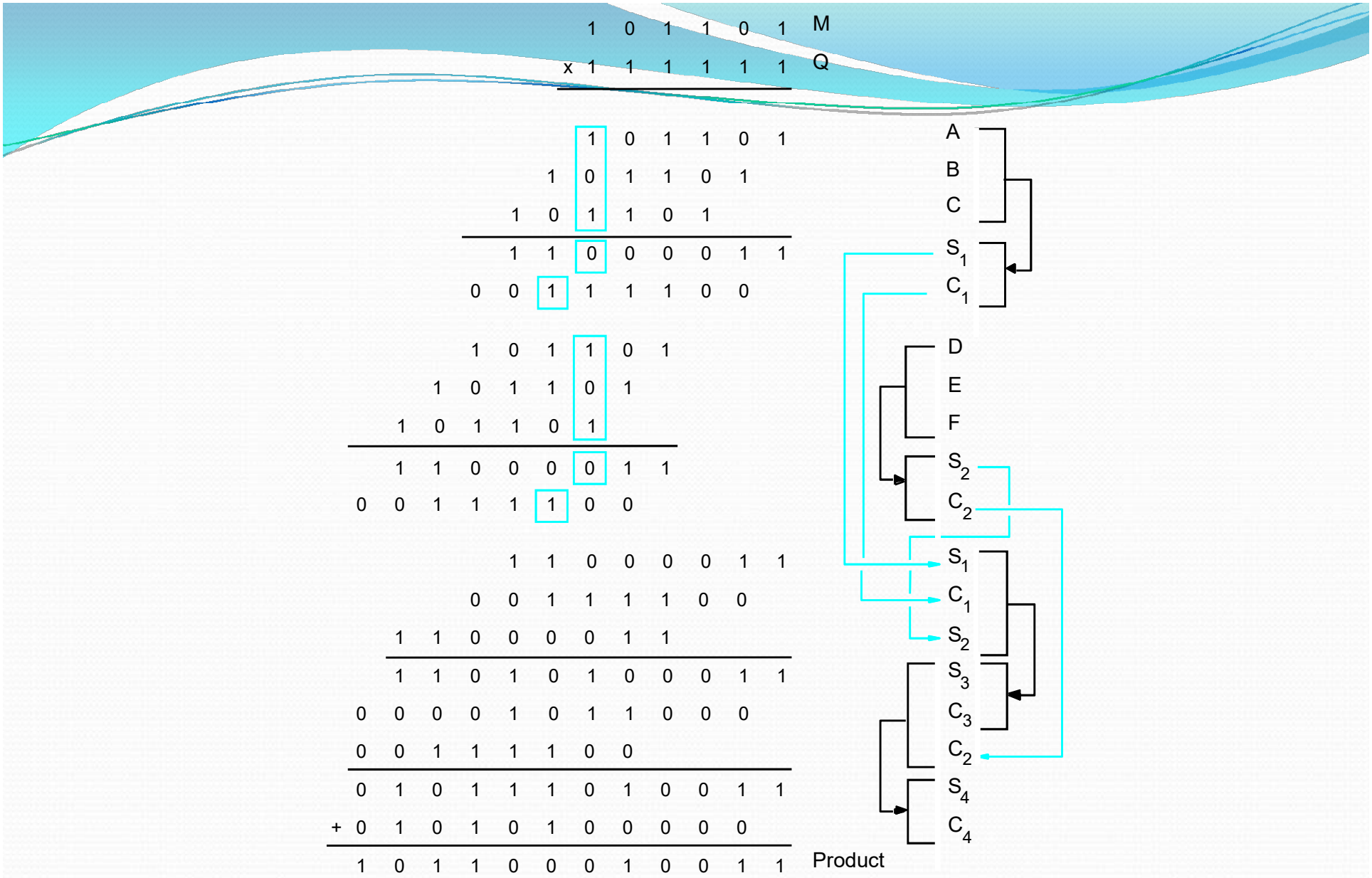


Figure 6.18. The multiplication example from Figure 6.17 performed using carry-save addition.

Integer Division

Manual Division

```
      21
   ------
13 ) 274
     26
    ---
     14
     13
    ---
      1
```

```
          10101
      ----------
1101 ) 100010010
       1101
      -------
        10000
         1101
        ------
          1110
          1101
         -----
             1
```

Longhand division examples.

Longhand Division Steps

- Position the divisor appropriately with respect to the dividend and performs a subtraction.
- If the remainder is zero or positive, a quotient bit of 1 is determined, the remainder is extended by another bit of the dividend, the divisor is repositioned, and another subtraction is performed.
- If the remainder is negative, a quotient bit of 0 is determined, the dividend is restored by adding back the divisor, and the divisor is repositioned for another subtraction.

Circuit Arrangement

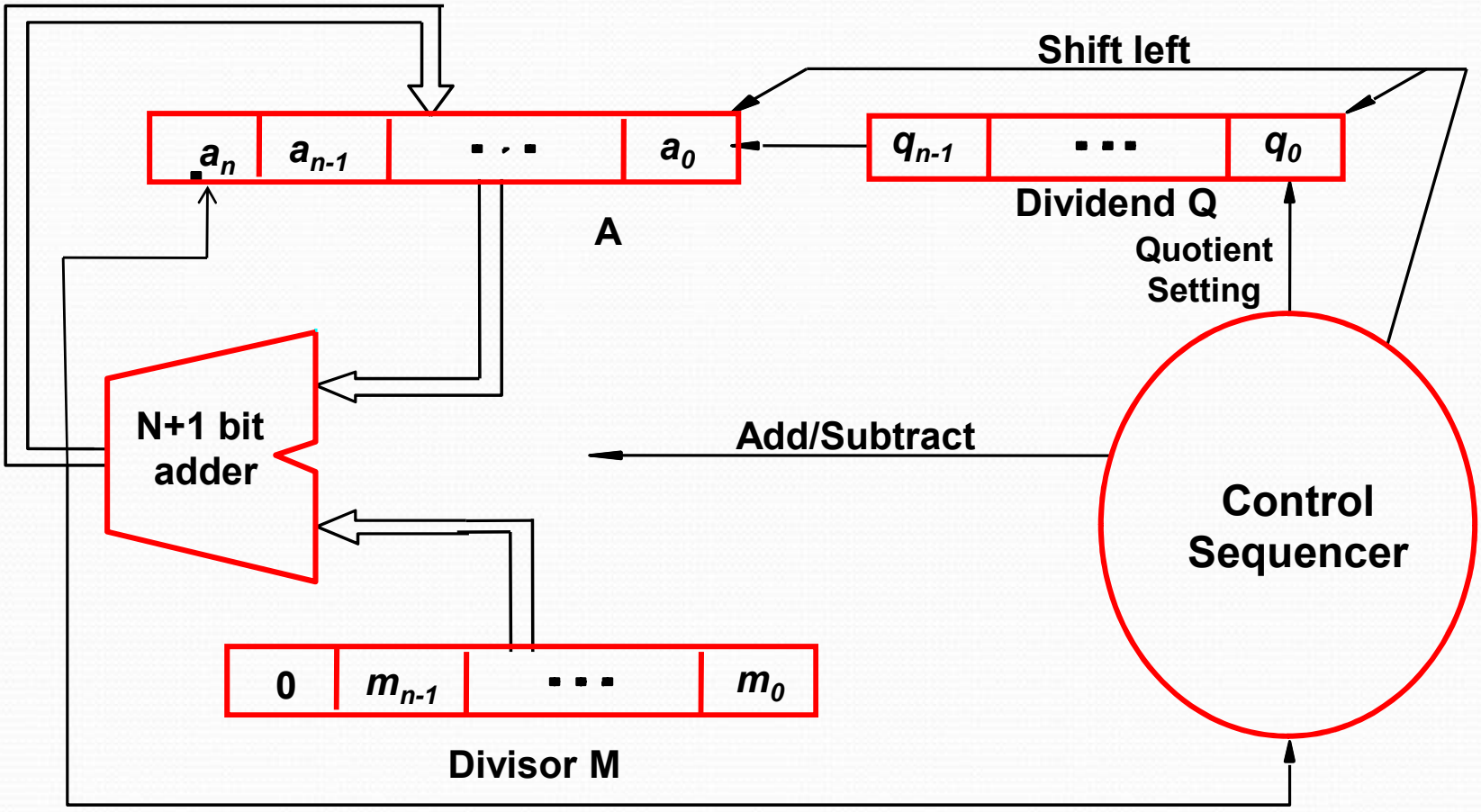


Figure 6.21. Circuit arrangement for binary division.

Restoring Division

- Shift A and Q left one binary position
- Subtract M from A, and place the answer back in A
- If the sign of A is 1, set q_0 to 0 and add M back to A (restore A); otherwise, set q_0 to 1
- Repeat these steps *n* times

Examples

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

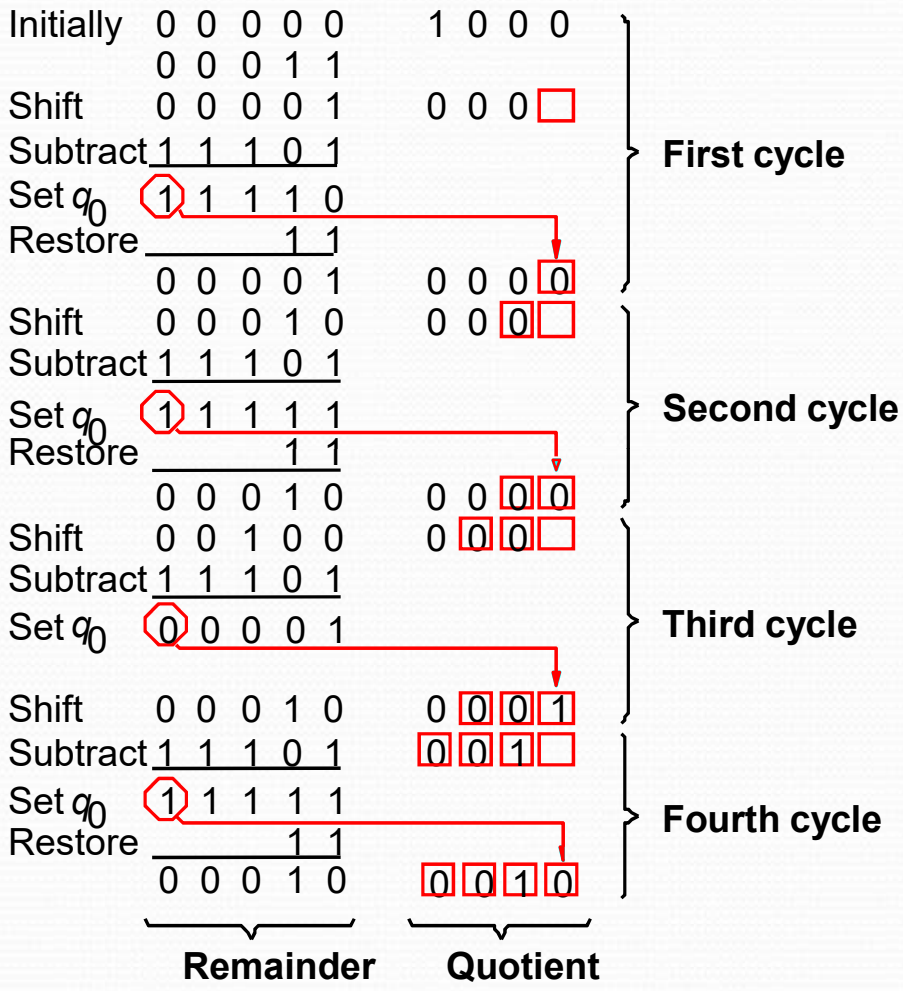
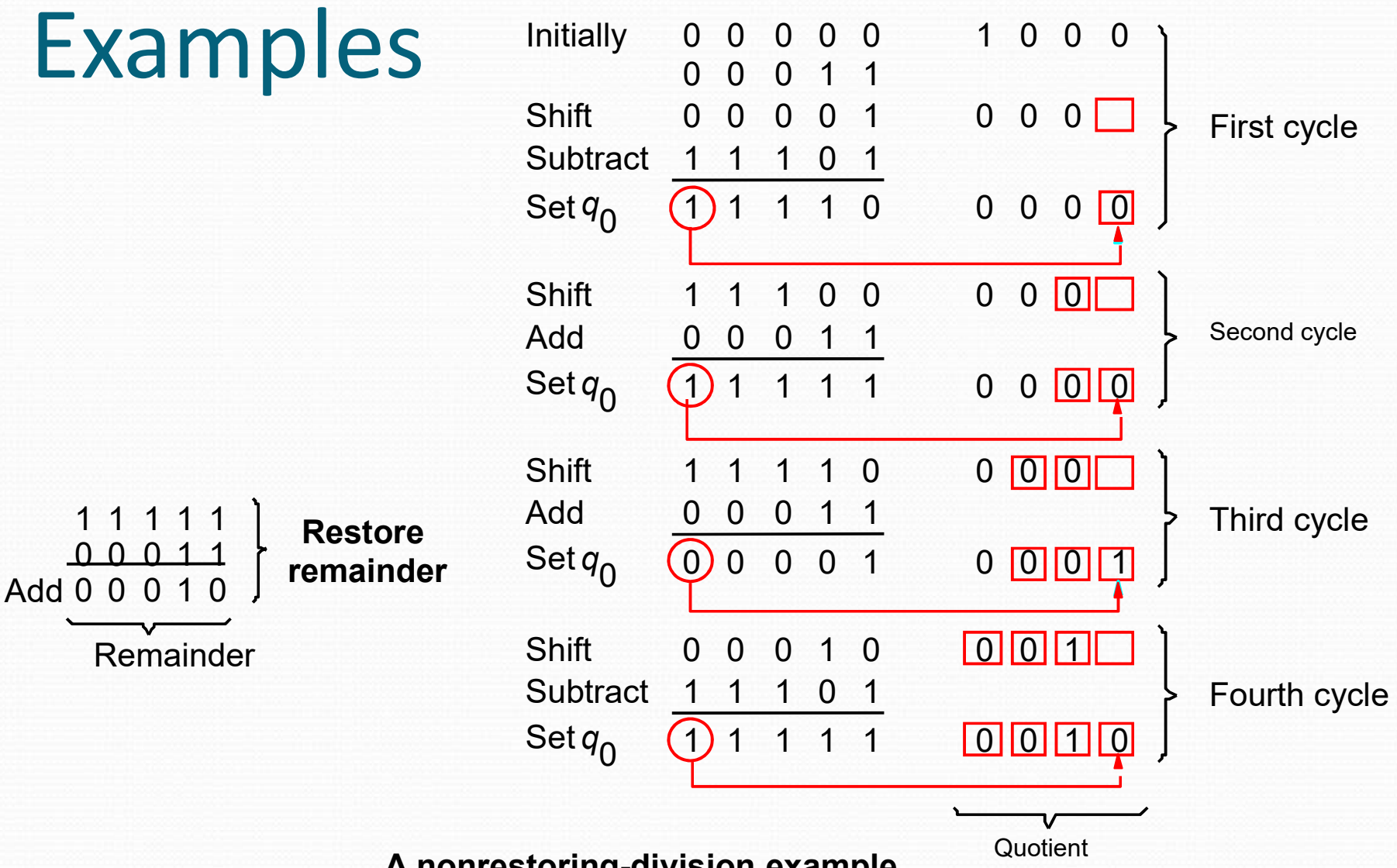


Figure 6.22. A restoring-division example.

Nonrestoring Division

- Avoid the need for restoring A after an unsuccessful subtraction.
- Any idea?
- Step 1: (Repeat *n* times)
  - If the sign of A is 0, shift A and Q left one bit position and subtract M from A; otherwise, shift A and Q left and add M to A.
  - Now, if the sign of A is 0, set q_0 to 1; otherwise, set q_0 to 0.
- Step2: If the sign of A is 1, add M to A

Examples



A nonrestoring-division example.

Floating-Point Numbers and Operations

Fractions

If b is a binary vector, then we have seen that it can be interpreted as an unsigned integer by:

$$V(b) = b_{31}.2^{31} + b_{30}.2^{30} + b_{n-3}.2^{29} + ... + b_1.2^1 + b_0.2^0$$

This vector has an implicit binary point to its immediate right:

$b_{31}b_{30}b_{29}.....b_1b_0.$ implicit binary point

Suppose if the binary vector is interpreted with the implicit binary point is just left of the sign bit:

implicit binary point $.b_{31}b_{30}b_{29}.....b_1b_0$

The value of b is then given by:

$$V(b) = b_{31}.2^{-1} + b_{30}.2^{-2} + b_{29}.2^{-3} + ... + b_1.2^{-31} + b_0.2^{-32}$$

Range of fractions

The value of the unsigned binary fraction is:

$$V(b) = b_{31}.2^{-1} + b_{30}.2^{-2} + b_{29}.2^{-3} + ... + b_{1}.2^{-31} + b_{0}.2^{-32}$$

The range of the numbers represented in this format is:

$$0 \le V(b) \le 1 - 2^{-32} \approx 0.9999999998$$

In general for a *n*-bit binary fraction (a number with an assumed binary point at the immediate left of the vector), then the range of values is:

$$0 \le V(b) \le 1 - 2^{-n}$$

Scientific notation

- Previous representations have a fixed point. Either the point is to the immediate right or it is to the immediate left. This is called Fixed point representation.
- Fixed point representation suffers from a drawback that the representation can only represent a finite range (and quite small) range of numbers.

A more convenient representation is the scientific representation, where the numbers are represented in the form:

$$x = m_1.m_2m_3m_4 \times b^{\pm e}$$

Components of these numbers are:

Mantissa (m), implied base (b), and exponent (e)

Significant digits

A number such as the following is said to have 7 significant digits

$$x = \pm 0.m_1 m_2 m_3 m_4 m_5 m_6 m_7 \times b^{\pm e}$$

Fractions in the range 0.0 to 0.9999999 need about 24 bits of precision (in binary). For example the binary fraction with 24 1's:

$$111111111111111111111111 = 0.9999999404$$

Not every real number between 0 and 0.9999999404 can be represented by a 24-bit fractional number.
The smallest non-zero number that can be represented is:

$$000000000000000000000001 = 5.96046 \times 10^{-8}$$

Every other non-zero number is constructed in increments of this value.

Sign and exponent digits

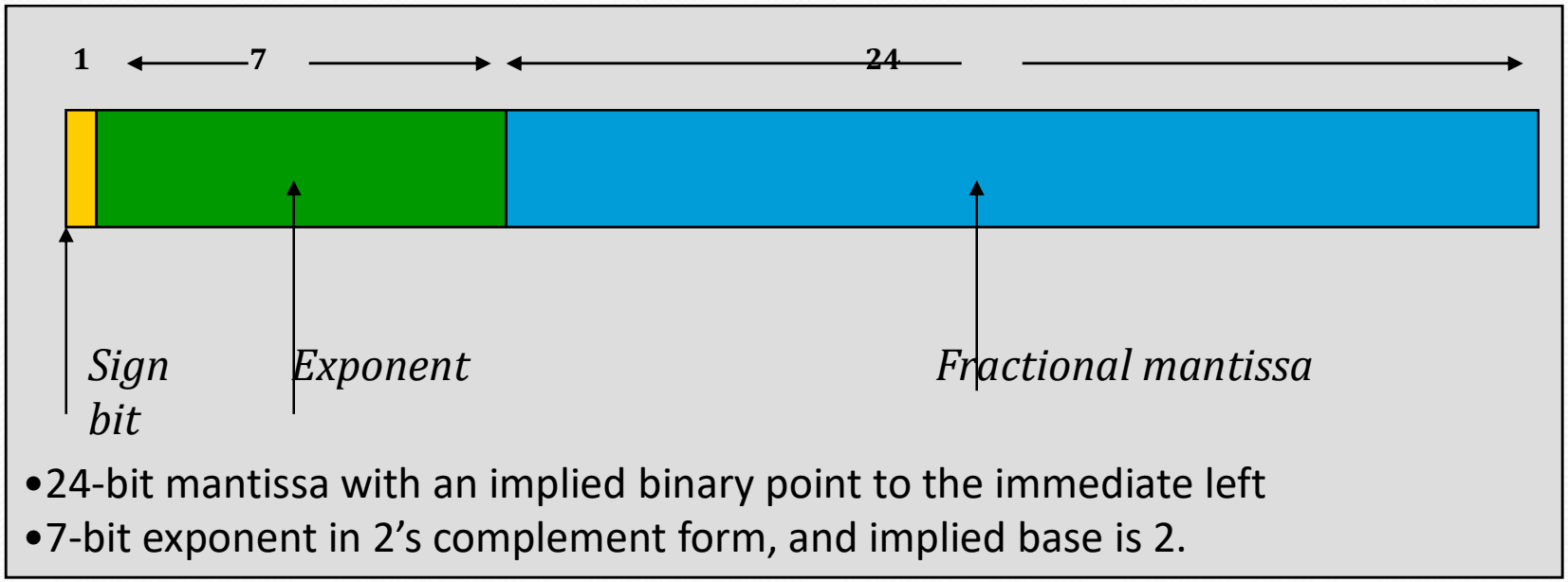
- In a 32-bit number, suppose we allocate 24 bits to represent a fractional mantissa.
- Assume that the mantissa is represented in sign and magnitude format, and we have allocated one bit to represent the sign.
- We allocate 7 bits to represent the exponent, and assume that the exponent is represented as a 2's complement integer.
- There are no bits allocated to represent the base, we assume that the base is implied for now, that is the base is 2.
- Since a 7-bit 2's complement number can represent values in the range -64 to 63, the range of numbers that can be represented is:

$$0.0000001 \times 2^{-64} \quad <= \ |x| <= \ 0.9999999 \times 2^{63}$$

- In decimal representation this range is:

$$0.5421 \times 10^{-20} \quad <= \ |x| <= \ 9.2237 \times 10^{18}$$

A sample representation



- 24-bit mantissa with an implied binary point to the immediate left
- 7-bit exponent in 2's complement form, and implied base is 2.

Normalization

Consider the number: $x = 0.0004056781 \times 10^{12}$

If the number is to be represented using only 7 significant mantissa digits, the representation ignoring rounding is: $x = 0.0004056 \times 10^{12}$

If the number is shifted so that as many significant digits are brought into 7 available slots: $x = 0.4056781 \times 10^{9} = 0.0004056 \times 10^{12}$

Exponent of *x* was decreased by 1 for every left shift of *x*.

A number which is brought into a form so that all of the available mantissa digits are optimally used (this is different from all occupied which may not hold), is called a normalized number.

Same methodology holds in the case of binary mantissas

$0001101000(10110) \times 2^{8} = 1101000101(10) \times 2^{5}$

Normalization (contd..)

- A floating point number is in normalized form if the most significant 1 in the mantissa is in the most significant bit of the mantissa.
- All normalized floating point numbers in this system will be of the form:

$$0.1xxxxx.....xx$$

Range of numbers representable in this system, if every number must be normalized is:

$$0.5 \times 2^{-64} <= |x| < 1 \times 2^{63}$$

Normalization, overflow and underflow

The procedure for normalizing a floating point number is:

Do (until MSB of mantissa = = 1)

Shift the mantissa left (or right)

Decrement (increment) the exponent by 1

end do

Applying the normalization procedure to: $.000111001110....0010 \times 2^{-62}$

gives: $.111001110..... \times 2^{-65}$

But we cannot represent an exponent of –65, in trying to normalize the number we have underflowed our representation.

Applying the normalization procedure to: $1.00111000.....x\ 2^{63}$

gives: $0.100111.....x\ 2^{64}$

This overflows the representation.

Changing the implied base

So far we have assumed an implied base of 2, that is our floating point numbers are of the form:

$$x = m\ 2^e$$

If we choose an implied base of 16, then:

$$x = m\ 16^e$$

Then:

$$y = (m.16)\ .16^{e-1}\ (m.2^4)\ .16^{e-1} = m\ .\ 16^e = x$$

- Thus, every four left shifts of a binary mantissa results in a decrease of 1 in a base 16 exponent.
- Normalization in this case means shifting the mantissa until there is a 1 in the first four bits of the mantissa.

Excess notation

- Rather than representing an exponent in 2's complement form, it turns out to be more beneficial to represent the exponent in excess notation.
- If 7 bits are allocated to the exponent, exponents can be represented in the range of -64 to +63, that is:

$$-64 <= e <= 63$$

Exponent can also be represented using the following coding called as excess-64:

$$E' = E_{true} + 64$$

In general, excess-p coding is represented as:

$$E' = E_{true} + p$$

True exponent of -64 is represented as 0
0 is represented as 64
63 is represented as 127

This enables efficient comparison of the relative sizes of two floating point numbers.

IEEE notation

IEEE Floating Point notation is the standard representation in use. There are two representations:

- Single precision.
- Double precision.

Both have an implied base of 2.

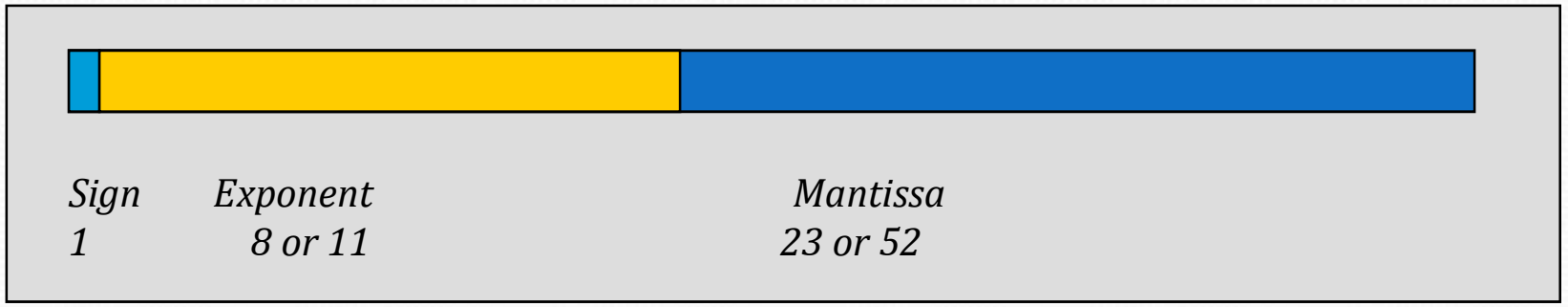
Single precision:

- 32 bits (23-bit mantissa, 8-bit exponent in excess-127 representation)

Double precision:

- 64 bits (52-bit mantissa, 11-bit exponent in excess-1023 representation)

Fractional mantissa, with an implied binary point at immediate left.



Peculiarities of IEEE notation

- Floating point numbers have to be represented in a normalized form to maximize the use of available mantissa digits.
- In a base-2 representation, this implies that the MSB of the mantissa is always equal to 1.
- If every number is normalized, then the MSB of the mantissa is always 1. We can do away without storing the MSB.
- IEEE notation assumes that all numbers are normalized so that the MSB of the mantissa is a 1 and does not store this bit.
- So the real MSB of a number in the IEEE notation is either a 0 or a 1.
- The values of the numbers represented in the IEEE single precision notation are of the form:

$$(+,-)\ 1.M \times 2^{(E - 127)}$$

- The hidden 1 forms the integer part of the mantissa.
- Note that excess-127 and excess-1023 (not excess-128 or excess-1024) are used to represent the exponent.

Exponent field

In the IEEE representation, the exponent is in excess-127 (excess-1023) notation.
The actual exponents represented are:

$$-126 <= E <= 127 \quad and \quad -1022 <= E <= 1023$$
$$not$$
$$-127 <= E <= 128 \quad and \quad -1023 <= E <= 1024$$

This is because the IEEE uses the exponents -127 and 128 (and -1023 and 1024), that is the actual values 0 and 255 to represent special conditions:

- Exact zero
- Infinity

Floating point arithmetic

Addition:

$$3.1415 \times 10^8 + 1.19 \times 10^6 = 3.1415 \times 10^8 + 0.0119 \times 10^8 = 3.1534 \times 10^8$$

Multiplication:

$$3.1415 \times 10^8 \times 1.19 \times 10^6 = (3.1415 \times 1.19) \times 10^{(8+6)}$$

Division:

$$3.1415 \times 10^8 / 1.19 \times 10^6 = (3.1415 / 1.19) \times 10^{(8-6)}$$

Biased exponent problem:

If a true exponent e is represented in excess-p notation, that is as e+p.

Then consider what happens under multiplication:

$$a.\ 10^{(x+p)} * b.\ 10^{(y+p)} = (a.b).\ 10^{(x+p+y+p)} = (a.b).\ 10^{(x+y+2p)}$$

Representing the result in excess-p notation implies that the exponent should be *x+y+p*. Instead it is *x+y+2p*.

Biases should be handled in floating point arithmetic.

Floating point arithmetic: ADD/SUB rule

- Choose the number with the smaller exponent.
- Shift its mantissa right until the exponents of both the numbers are equal.
- Add or subtract the mantissas.
- Determine the sign of the result.
- Normalize the result if necessary and truncate/round to the number of mantissa bits.

Note: This does not consider the possibility of overflow/underflow.

Floating point arithmetic: MUL rule

- Add the exponents.
- Subtract the bias.
- Multiply the mantissas and determine the sign of the result.
- Normalize the result (if necessary).
- Truncate/round the mantissa of the result.

Floating point arithmetic: DIV rule

- Subtract the exponents
- Add the bias.
- Divide the mantissas and determine the sign of the result.
- Normalize the result if necessary.
- Truncate/round the mantissa of the result.

Note: Multiplication and division does not require alignment of the mantissas the way addition and subtraction does.

Guard bits

While adding two floating point numbers with 24-bit mantissas, we shift the mantissa of the number with the smaller exponent to the right until the two exponents are equalized.

This implies that mantissa bits may be lost during the right shift (that is, bits of precision may be shifted out of the mantissa being shifted).

To prevent this, floating point operations are implemented by keeping guard bits, that is, extra bits of precision at the least significant end of the mantissa.

The arithmetic on the mantissas is performed with these extra bits of precision.

After an arithmetic operation, the guarded mantissas are:

- Normalized (if necessary)
- Converted back by a process called truncation/rounding to a 24-bit mantissa.

Truncation/rounding

- Straight chopping:
  - The guard bits (excess bits of precision) are dropped.
- Von Neumann rounding:
  - If the guard bits are all 0, they are dropped.
  - However, if any bit of the guard bit is a 1, then the LSB of the retained bit is set to 1.
- Rounding:
  - If there is a 1 in the MSB of the guard bit then a 1 is added to the LSB of the retained bits.

Rounding

- Rounding is evidently the most accurate truncation method.
- However,
  - Rounding requires an addition operation.
  - Rounding may require a renormalization, if the addition operation de-normalizes the truncated number.

> *0.111111100000 rounds to 0.111111 + 0.000001*
> *=1.000000 which must be renormalized to 0.100000*

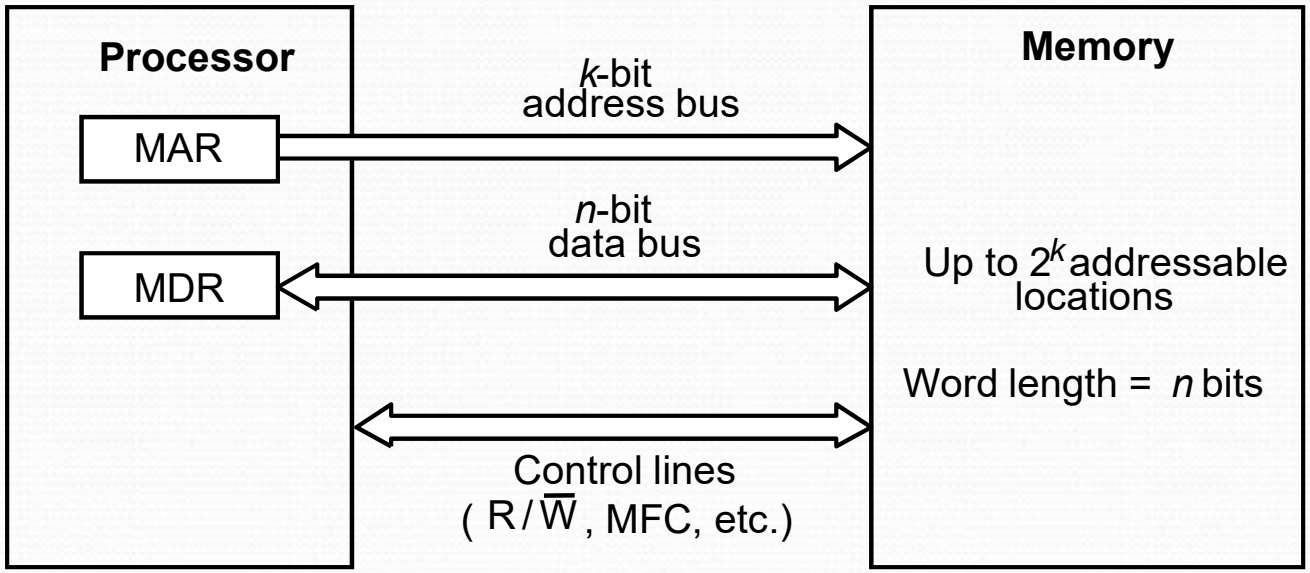
- IEEE uses the rounding method.

Fundamental Concepts

The Memory System

Some basic concepts

- Maximum size of the Main Memory
- byte-addressable
- CPU-Main Memory Connection



Some basic concepts(Contd.,)

- Measures for the speed of a memory:
  - memory access time.
  - memory cycle time.
- An important design issue is to provide a computer system with as large and fast a memory as possible, within a given cost target.
- Several techniques to increase the effective size and speed of the memory:
  - Cache memory (to increase the effective speed).
  - Virtual memory (to increase the effective size).

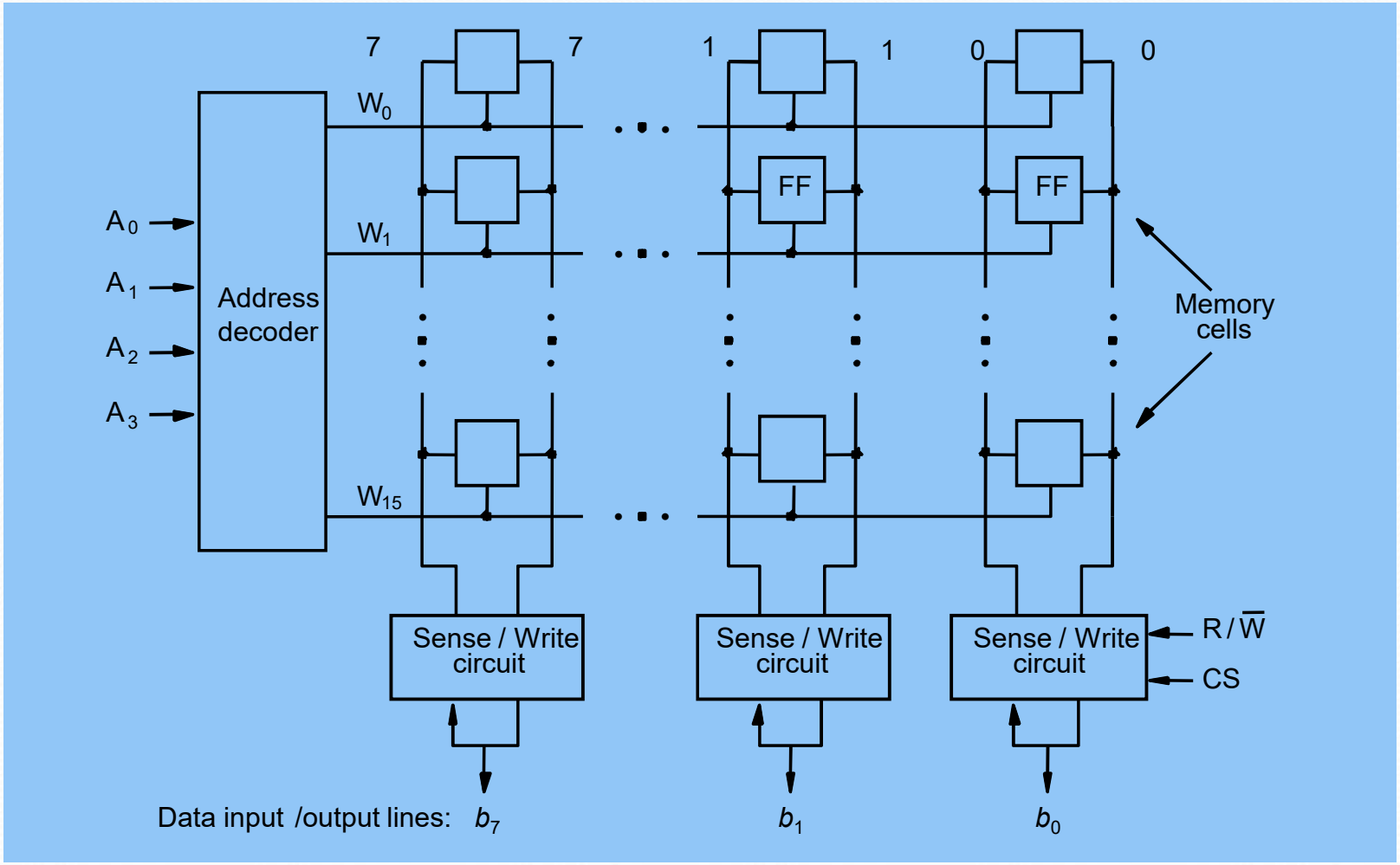
Semiconductor RAM memories

The Memory System

Internal organization of memory chips

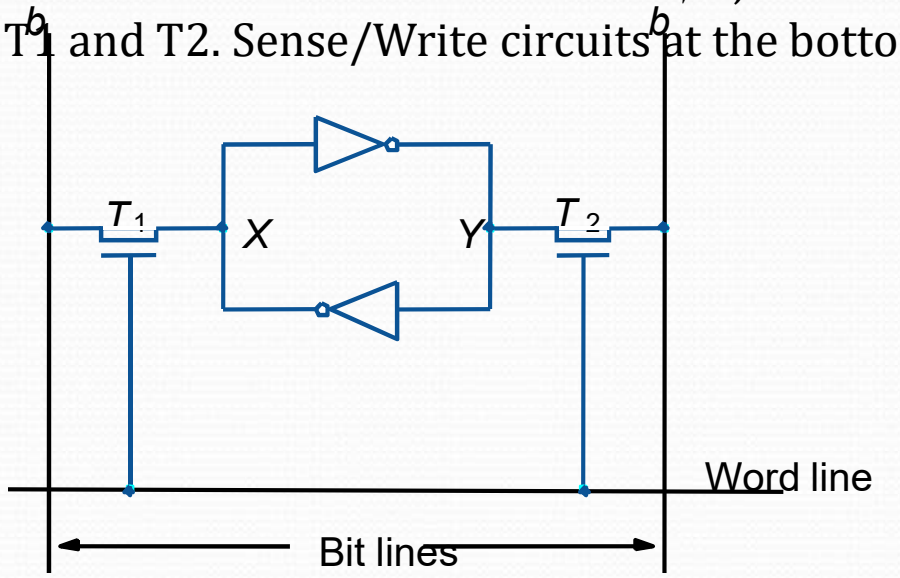
- Each memory cell can hold one bit of information.
- Memory cells are organized in the form of an array.
- One row is one memory word.
- All cells of a row are connected to a common line, known as the "word line".
- Word line is connected to the address decoder.
- Sense/write circuits are connected to the data input/output lines of the memory chip.

Internal organization of memory chips (Contd.,)



SRAM Cell

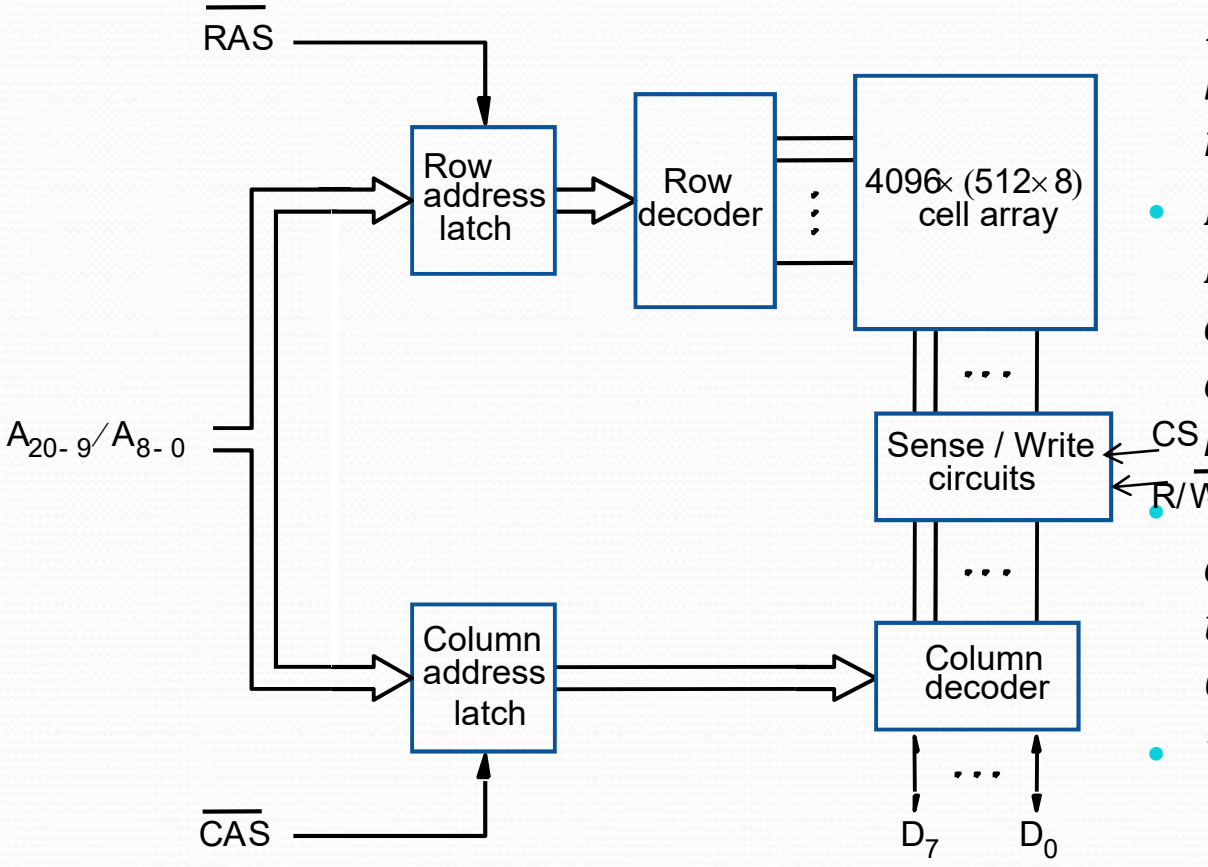
- Two transistor inverters are cross connected to implement a basic flip-flop.
- The cell is connected to one word line and two bits lines by transistors T1 and T2
- When word line is at ground level, the transistors are turned off and the latch retains its state
- Read operation: In order to read state of SRAM cell, the word line is activated to close switches T1 and T2. Sense/Write circuits at the bottom monitor the state of b and b'



Asynchronous DRAMs

- Static RAMs (SRAMs):
  - Consist of circuits that are capable of retaining their state as long as the power is applied.
  - Volatile memories, because their contents are lost when power is interrupted.
  - Access times of static RAMs are in the range of few nanoseconds.
  - However, the cost is usually high.
- Dynamic RAMs (DRAMs):
  - Do not retain their state indefinitely.
  - Contents must be periodically refreshed.
  - Contents may be refreshed while accessing them for reading.

Asynchronous DRAMs

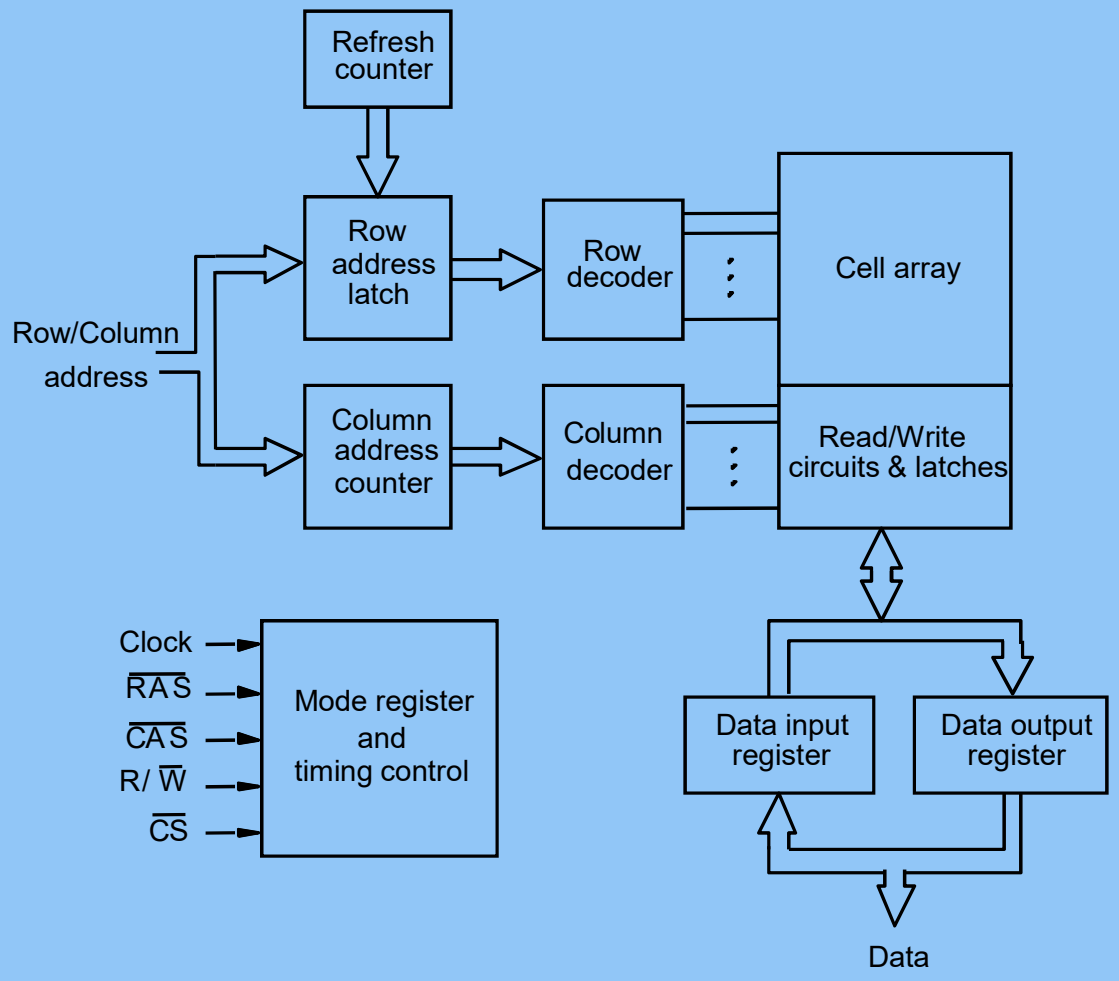


- ***Each row can store 512 bytes. 12 bits to select a row, and 9 bits to select a group in a row. Total of 21 bits.***
- ***First apply the row address, RAS signal latches the row address. Then apply the column address, CAS signal latches the address.***
- ***Timing of the memory unit is controlled by a specialized unit which generates RAS and CAS.***
- ***This is asynchronous DRAM***

Fast Page Mode

- Suppose if we want to access the consecutive bytes in the selected row.
- This can be done without having to reselect the row.
  - Add a latch at the output of the sense circuits in each row.
  - All the latches are loaded when the row is selected.
  - Different column addresses can be applied to select and place different bytes on the data lines.
- Consecutive sequence of column addresses can be applied under the control signal CAS, without reselecting the row.
  - Allows a block of data to be transferred at a much faster rate than random accesses.
  - A small collection/group of bytes is usually referred to as a block.
- This transfer capability is referred to as the fast page mode feature.

Synchronous DRAMs

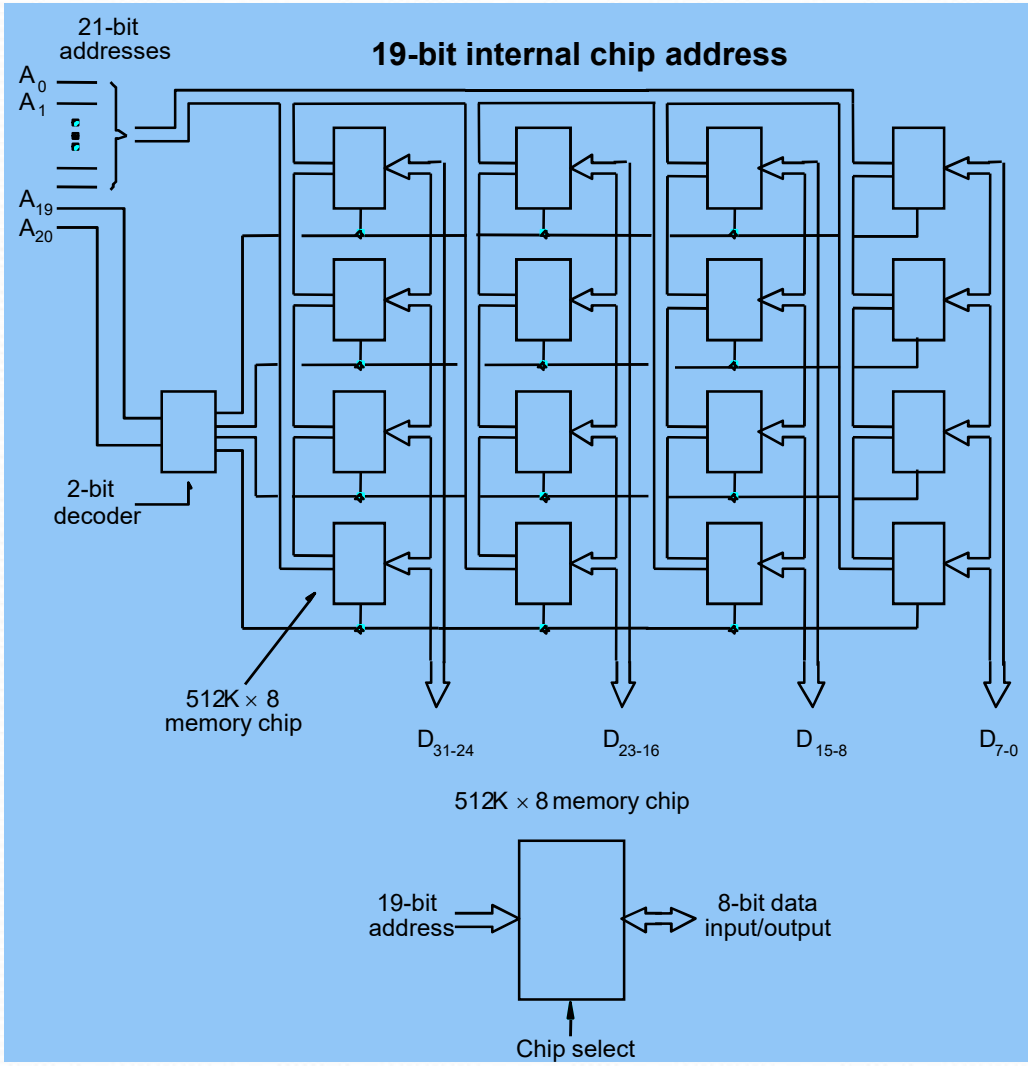


- *Operation is directly synchronized with processor clock signal.*
- *The outputs of the sense circuits are connected to a latch.*
- *During a Read operation, the contents of the cells in a row are loaded onto the latches.*
- *During a refresh operation, the contents of the cells are refreshed without changing the contents of the latches.*
- *Data held in the latches correspond to the selected columns are transferred to the output.*
- *For a burst mode of operation, successive columns are selected using column address counter and clock. $\overline{CAS}$ signal need not be generated externally. A new data is placed during raising edge of the clock*

Latency, Bandwidth, and DDRSDRAMs

- Memory latency is the time it takes to transfer a word of data to or from memory
- Memory bandwidth is the number of bits or bytes that can be transferred in one second.
- DDRSDRAMs
  - Cell array is organized in two banks

Static memories



Implement a memory unit of 2M words of 32 bits each.
Use 512x8 static memory chips.
Each column consists of 4 chips.
Each chip implements one byte position.
A chip is selected by setting its chip select control line to 1.
Selected chip places its data on the data output line, outputs of other chips are in high impedance state.
21 bits to address a 32-bit word.
High order 2 bits are needed to select the row, by activating the four Chip Select signals.
19 bits are used to access specific byte locations inside the selected chip.

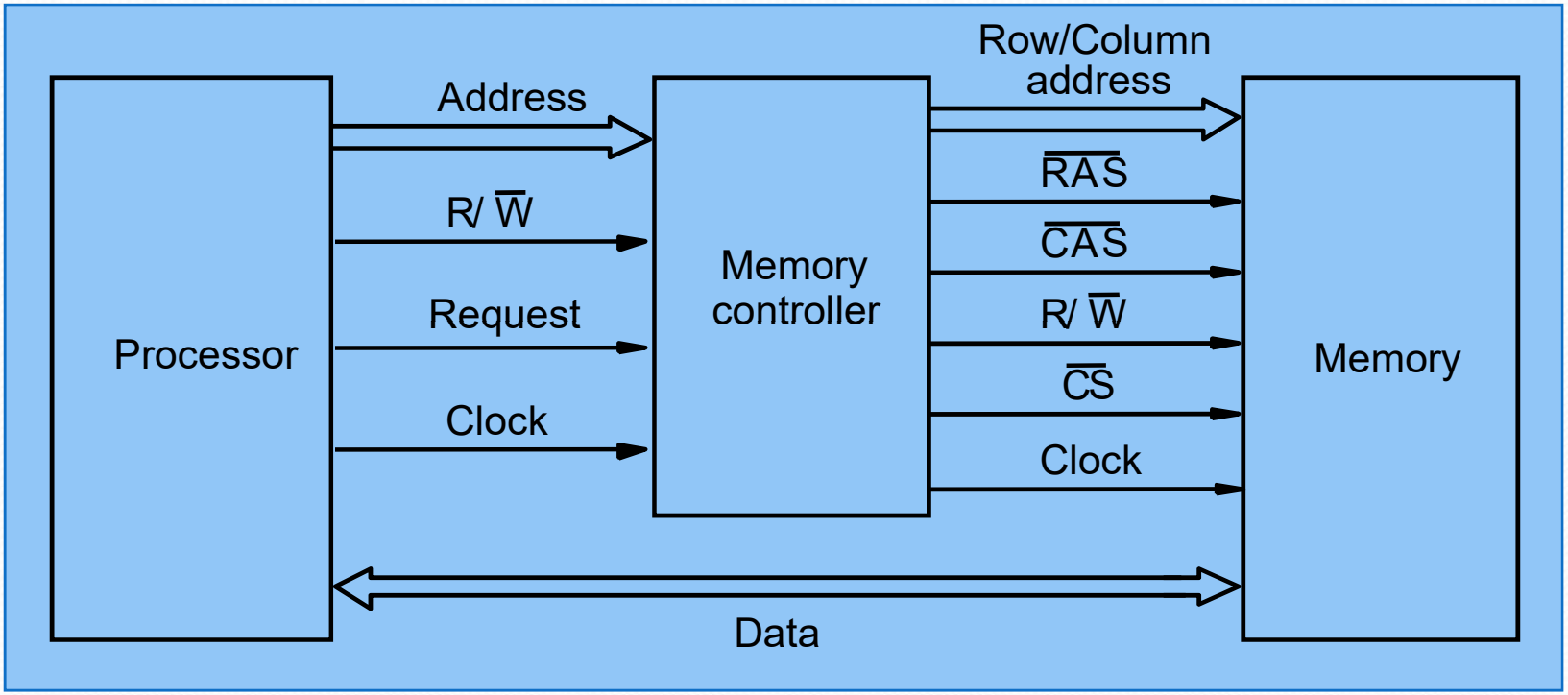
Dynamic memories

- Large dynamic memory systems can be implemented using DRAM chips in a similar way to static memory systems.
- Placing large memory systems directly on the motherboard will occupy a large amount of space.
  - Also, this arrangement is inflexible since the memory system cannot be expanded easily.
- Packaging considerations have led to the development of larger memory units known as SIMMs (Single In-line Memory Modules) and DIMMs (Dual In-line Memory Modules).
- Memory modules are an assembly of memory chips on a small board that plugs vertically onto a single socket on the motherboard.
  - Occupy less space on the motherboard.
  - Allows for easy expansion by replacement.

Memory controller

- Recall that in a dynamic memory chip, to reduce the number of pins, multiplexed addresses are used.
- Address is divided into two parts:
  - High-order address bits select a row in the array.
  - They are provided first, and latched using RAS signal.
  - Low-order address bits select a column in the row.
  - They are provided later, and latched using CAS signal.
- However, a processor issues all address bits at the same time.
- In order to achieve the multiplexing, memory controller circuit is inserted between the processor and memory.

Memory controller (contd..)



Read-Only Memories (ROMs)

The Memory System

Read-Only Memories (ROMs)

- SRAM and SDRAM chips are volatile:
  - Lose the contents when the power is turned off.
- Many applications need memory devices to retain contents after the power is turned off.
  - For example, computer is turned on, the operating system must be loaded from the disk into the memory.
  - Store instructions which would load the OS from the disk.
  - Need to store these instructions so that they will not be lost after the power is turned off.
  - We need to store the instructions into a non-volatile memory.
- Non-volatile memory is read in the same manner as volatile memory.
  - Separate writing process is needed to place information in this memory.
  - Normal operation involves only reading of data, this type of memory is called Read-Only memory (ROM).

Read-Only Memories (Contd.,)

- Read-Only Memory:
  - Data are written into a ROM when it is manufactured.
- Programmable Read-Only Memory (PROM):
  - Allow the data to be loaded by a user.
  - Process of inserting the data is irreversible.
  - Storing information specific to a user in a ROM is expensive.
  - Providing programming capability to a user may be better.
- Erasable Programmable Read-Only Memory (EPROM):
  - Stored data to be erased and new data to be loaded.
  - Flexibility, useful during the development phase of digital systems.
  - Erasable, reprogrammable ROM.
  - Erasure requires exposing the ROM to UV light.

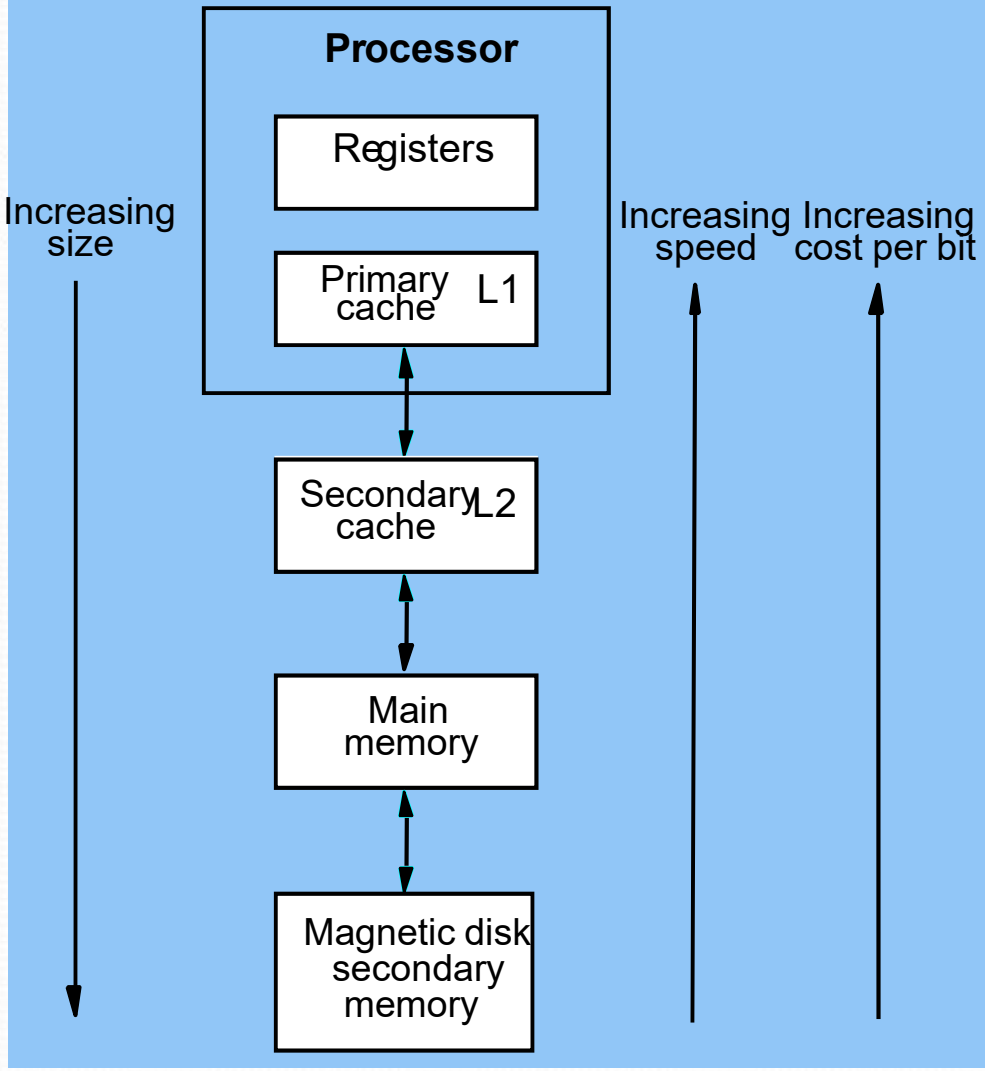
Read-Only Memories (Contd.,)

- Electrically Erasable Programmable Read-Only Memory (EEPROM):
  - To erase the contents of EPROMs, they have to be exposed to ultraviolet light.
  - Physically removed from the circuit.
  - EEPROMs the contents can be stored and erased electrically.
- Flash memory:
  - Has similar approach to EEPROM.
  - Read the contents of a single cell, but write the contents of an entire block of cells.
  - Flash devices have greater density.
    - Higher capacity and low storage cost per bit.
  - Power consumption of flash memory is very low, making it attractive for use in equipment that is battery-driven.
  - Single flash chips are not sufficiently large, so larger memory modules are implemented using flash cards and flash drives.

Speed, Size, and Cost

- A big challenge in the design of a computer system is to provide a sufficiently large memory, with a reasonable speed at an affordable cost.
- Static RAM:
  - Very fast, but expensive, because a basic SRAM cell has a complex circuit making it impossible to pack a large number of cells onto a single chip.
- Dynamic RAM:
  - Simpler basic cell circuit, hence are much less expensive, but significantly slower than SRAMs.
- Magnetic disks:
  - Storage provided by DRAMs is higher than SRAMs, but is still less than what is necessary.
  - Secondary storage such as magnetic disks provide a large amount of storage, but is much slower than DRAMs.

Memory Hierarchy



- *Fastest access is to the data held in processor registers. Registers are at the top of the memory hierarchy.*
- *Relatively small amount of memory that can be implemented on the processor chip. This is processor cache.*
- *Two levels of cache. Level 1 (L1) cache is on the processor chip. Level 2 (L2) cache is in between main memory and processor.*
- *Next level is main memory, implemented as SIMMs. Much larger, but much slower than cache memory.*
- *Next level is magnetic disks. Huge amount of inexepensive storage.*
- *Speed of memory access is critical, the idea is to bring instructions and data that will be used in the near future as close to the processor as possible.*

Cache Memories

The Memory System

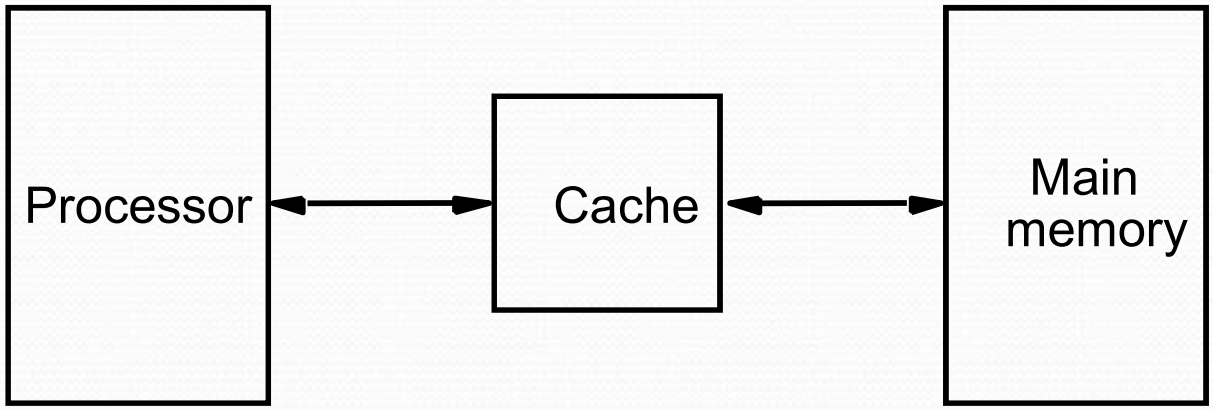
Cache Memories

- Processor is much faster than the main memory.
  - As a result, the processor has to spend much of its time waiting while instructions and data are being fetched from the main memory.
  - Major obstacle towards achieving good performance.
- Speed of the main memory cannot be increased beyond a certain point.
- Cache memory is an architectural arrangement which makes the main memory appear faster to the processor than it really is.
- Cache memory is based on the property of computer programs known as "locality of reference".

Locality of Reference

- Analysis of programs indicates that many instructions in localized areas of a program are executed repeatedly during some period of time, while the others are accessed relatively less frequently.
  - These instructions may be the ones in a loop, nested loop or few procedures calling each other repeatedly.
  - This is called "locality of reference".
- Temporal locality of reference:
  - Recently executed instruction is likely to be executed again very soon.
- Spatial locality of reference:
  - Instructions with addresses close to a recently instruction are likely to be executed soon.

Cache memories



- *Processor issues a Read request, a block of words is transferred from the main memory to the cache, one word at a time.*
- *Subsequent references to the data in this block of words are found in the cache.*
- *At any given time, only some blocks in the main memory are held in the cache. Which blocks in the main memory are in the cache is determined by a "mapping function".*
- *When the cache is full, and a block of words needs to be transferred from the main memory, some block of words in the cache must be replaced. This is determined by a "replacement algorithm".*

Cache hit

- *Existence of a cache is transparent to the processor. The processor issues Read and Write requests in the same manner.*
- *If the data is in the cache it is called a Read or Write hit.*
- *Read hit:*
  - *The data is obtained from the cache.*
- *Write hit:*
  - *Cache has a replica of the contents of the main memory.*
  - *Contents of the cache and the main memory may be updated simultaneously. This is the write-through protocol.*
  - *Update the contents of the cache, and mark it as updated by setting a bit known as the dirty bit or modified bit. The contents of the main memory are updated when this block is replaced. This is write-back or copy-back protocol.*

Cache miss

- *If the data is not present in the cache, then a Read miss or Write miss occurs.*
- *Read miss:*
  - *Block of words containing this requested word is transferred from the memory.*
  - *After the block is transferred, the desired word is forwarded to the processor.*
  - *The desired word may also be forwarded to the processor as soon as it is transferred without waiting for the entire block to be transferred. This is called load-through or early-restart.*
- *Write-miss:*
  - *Write-through protocol is used, then the contents of the main memory are updated directly.*
  - *If write-back protocol is used, the block containing the addressed word is first brought into the cache. The desired word is overwritten with new information.*

Cache Coherence Problem

- *A bit called as "valid bit" is provided for each block.*
- *If the block contains valid data, then the bit is set to 1, else it is 0.*
- *Valid bits are set to 0, when the power is just turned on.*
- *When a block is loaded into the cache for the first time, the valid bit is set to 1.*

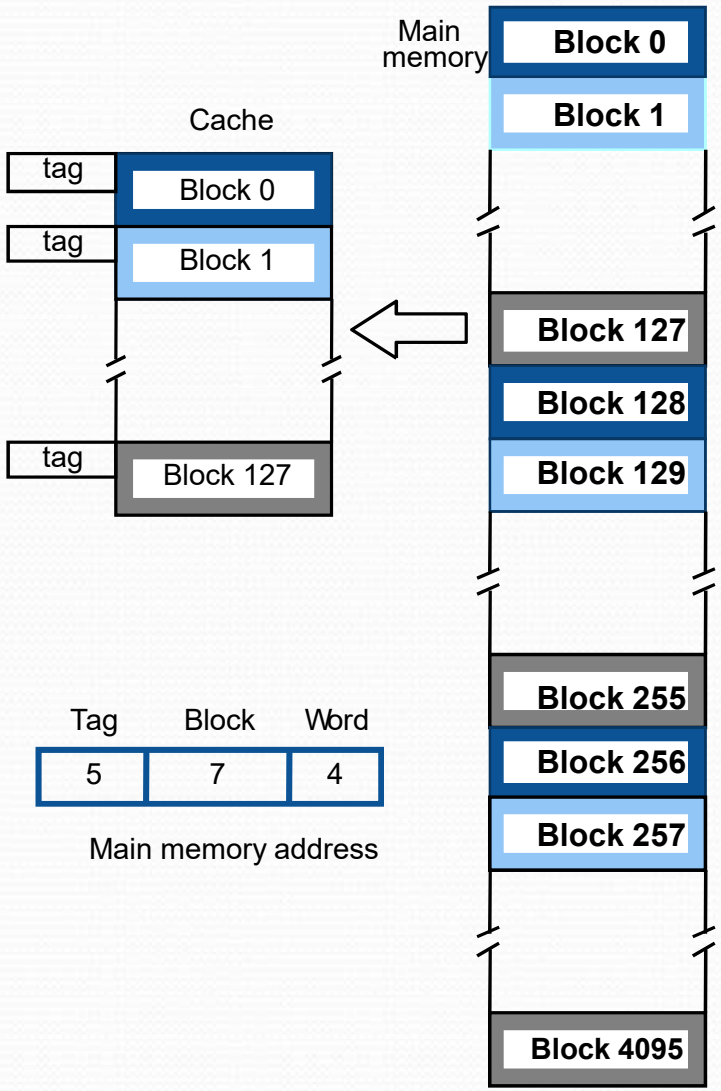
- *Data transfers between main memory and disk occur directly bypassing the cache.*
- *When the data on a disk changes, the main memory block is also updated.*
- *However, if the data is also resident in the cache, then the valid bit is set to 0.*

- *What happens if the data in the disk and main memory changes and the write-back protocol is being used?*
- *In this case, the data in the cache may also have changed and is indicated by the dirty bit.*
- *The copies of the data in the cache, and the main memory are different. This is called the cache coherence problem.*
- *One option is to force a write-back before the main memory is updated from the disk.*

Mapping functions

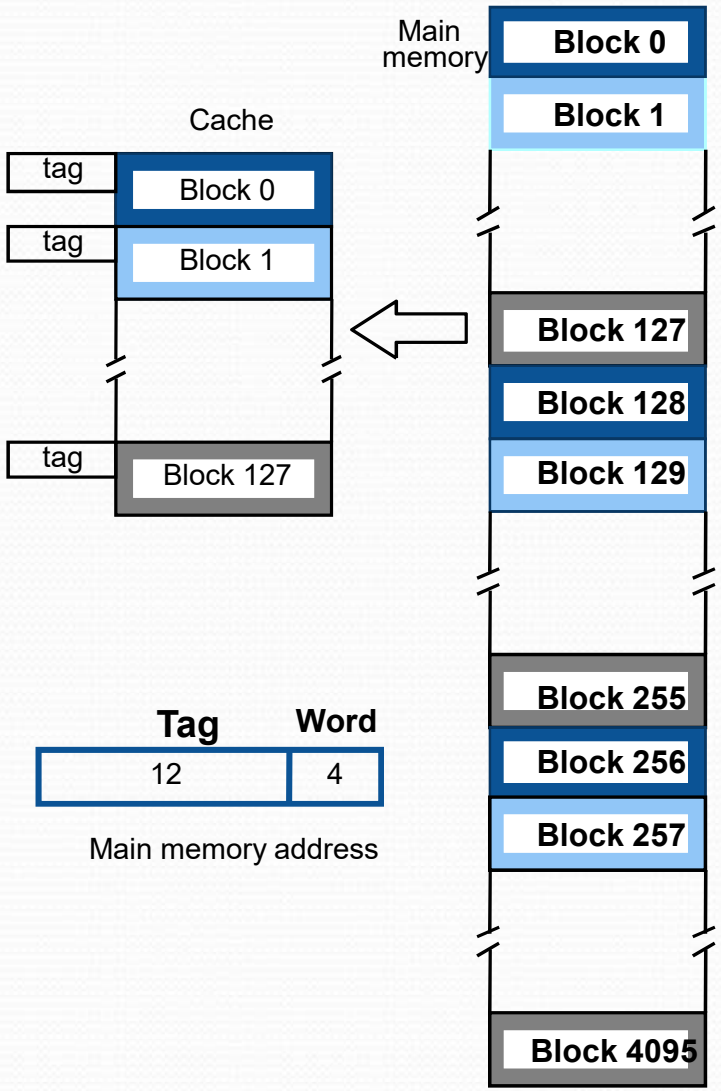
- Mapping functions determine how memory blocks are placed in the cache.
- A simple processor example:
  - Cache consisting of 128 blocks of 16 words each.
  - Total size of cache is 2048 (2K) words.
  - Main memory is addressable by a 16-bit address.
  - Main memory has 64K words.
  - Main memory has 4K blocks of 16 words each.
- Three mapping functions:
  - Direct mapping
  - Associative mapping
  - Set-associative mapping.

Direct mapping



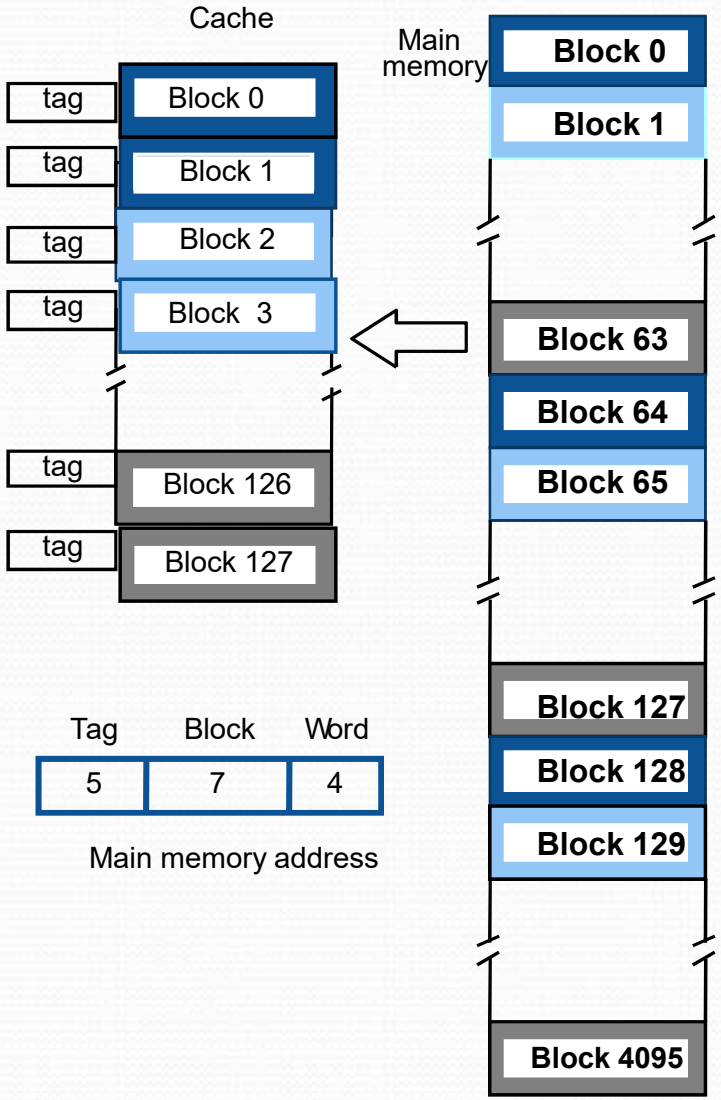
- *Block j of the main memory maps to j modulo 128 of the cache. 0 maps to 0, 129 maps to 1.*
- *More than one memory block is mapped onto the same position in the cache.*
- *May lead to contention for cache blocks even if the cache is not full.*
- *Resolve the contention by allowing new block to replace the old block, leading to a trivial replacement algorithm.*
- *Memory address is divided into three fields:*
  - *Low order 4 bits determine one of the 16 words in a block.*
  - *When a new block is brought into the cache, the the next 7 bits determine which cache block this new block is placed in.*
  - *High order 5 bits determine which of the possible 32 blocks is currently present in the cache. These are tag bits.*
- *Simple to implement but not very flexible.*

Associative mapping



- *Main memory block can be placed into any cache position.*
- *Memory address is divided into two fields:*
  - *Low order 4 bits identify the word within a block.*
  - *High order 12 bits or tag bits identify a memory block when it is resident in the cache.*
- *Flexible, and uses cache space efficiently.*
- *Replacement algorithms can be used to replace an existing block in the cache when the cache is full.*
- *Cost is higher than direct-mapped cache because of the need to search all 128 patterns to determine whether a given block is in the cache.*

Set-Associative mapping



Blocks of cache are grouped into sets.
Mapping function allows a block of the main memory to reside in any block of a specific set.
Divide the cache into 64 sets, with two blocks per set.
Memory block 0, 64, 128 etc. map to block 0, and they can occupy either of the two positions.
Memory address is divided into three fields:

- *6 bit field determines the set number.*
- *High order 6 bit fields are compared to the tag fields of the two blocks in a set.*

Set-associative mapping combination of direct and associative mapping.
Number of blocks per set is a design parameter.

- *One extreme is to have all the blocks in one set, requiring no set bits (fully associative mapping).*
- *Other extreme is to have one block per set, is the same as direct mapping.*

Performance considerations

The Memory System

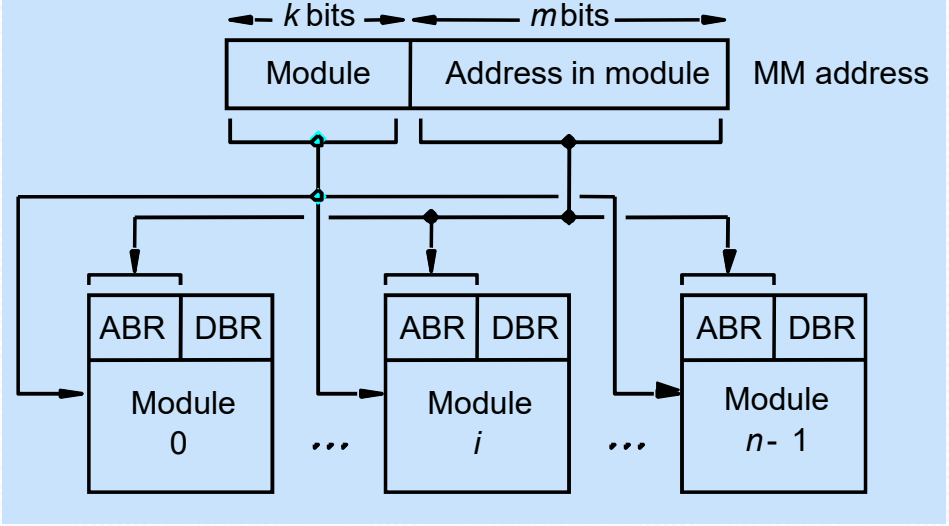
Performance considerations

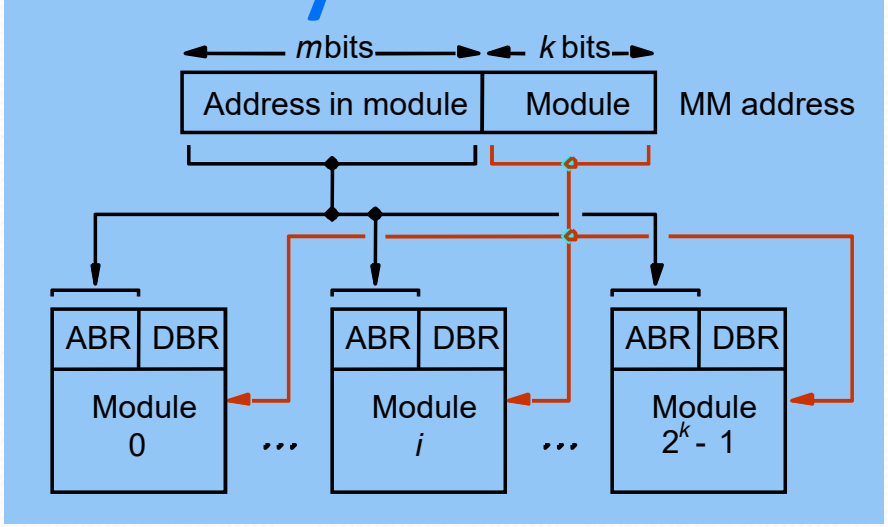
- A key design objective of a computer system is to achieve the best possible performance at the lowest possible cost.
  - Price/performance ratio is a common measure of success.
- Performance of a processor depends on:
  - How fast machine instructions can be brought into the processor for execution.
  - How fast the instructions can be executed.

Interleaving

- Divides the memory system into a number of memory modules. Each module has its own address buffer register (ABR) and data buffer register (DBR).
- Arranges addressing so that successive words in the address space are placed in different modules.
- When requests for memory access involve consecutive addresses, the access will be to different modules.
- Since parallel access to these modules is possible, the average rate of fetching words from the Main Memory can be increased.

Methods of address layouts





- *Consecutive words are placed in a module.*
- *High-order k bits of a memory address determine the module.*
- *Low-order m bits of a memory address determine the word within a module.*
- *When a block of words is transferred from main memory to cache, only one module is busy at a time.*

- ***Consecutive words are located in consecutive modules.***
- ***Consecutive addresses can be located in consecutive modules.***
- ***While transferring a block of data, several memory modules can be kept busy at the same time.***

Hit Rate and Miss Penalty

- Hit rate
- Miss penalty
- Hit rate can be improved by increasing block size, while keeping cache size constant
- Block sizes that are neither very small nor very large give best results.
- Miss penalty can be reduced if load-through approach is used when loading new blocks into cache.

Caches on the processor chip

- In high performance processors 2 levels of caches are normally used.
- Avg access time in a system with 2 levels of caches is

$$T_{ave} = h1c1+(1-h1)h2c2+(1-h1)(1-h2)M$$

Other Performance Enhancements

Write buffer

- *Write-through:*
- *Each write operation involves writing to the main memory.*
- *If the processor has to wait for the write operation to be complete, it slows down the processor.*
- *Processor does not depend on the results of the write operation.*
- *Write buffer can be included for temporary storage of write requests.*
- *Processor places each write request into the buffer and continues execution.*
- *If a subsequent Read request references data which is still in the write buffer, then this data is referenced in the write buffer.*

- *Write-back:*
- *Block is written back to the main memory when it is replaced.*
- *If the processor waits for this write to complete, before reading the new block, it is slowed down.*
- *Fast write buffer can hold the block to be written, and the new block can be read first.*

Other Performance Enhancements (Contd.,)

Prefetching

- *New data are brought into the processor when they are first needed.*
- *Processor has to wait before the data transfer is complete.*
- *Prefetch the data into the cache before they are actually needed, or a before a Read miss occurs.*
- *Prefetching can be accomplished through software by including a special instruction in the machine language of the processor.*
  - ***Inclusion of prefetch instructions increases the length of the programs.***
- *Prefetching can also be accomplished using hardware:*
  - ***Circuitry that attempts to discover patterns in memory references and then prefetches according to this pattern.***

Other Performance Enhancements (Contd.,)

Lockup-Free Cache

- *Prefetching scheme does not work if it stops other accesses to the cache until the prefetch is completed.*
- *A cache of this type is said to be "locked" while it services a miss.*
- *Cache structure which supports multiple outstanding misses is called a lockup free cache.*
- *Since only one miss can be serviced at a time, a lockup free cache must include circuits that keep track of all the outstanding misses.*
- *Special registers may hold the necessary information about these misses.*

Virtual Memory

The Memory System

Virtual memories

- Recall that an important challenge in the design of a computer system is to provide a large, fast memory system at an affordable cost.
- Architectural solutions to increase the effective speed and size of the memory system.
- Cache memories were developed to increase the effective speed of the memory system.
- Virtual memory is an architectural solution to increase the effective size of the memory system.

Virtual memories (contd..)

- Recall that the addressable memory space depends on the number of address bits in a computer.
  - For example, if a computer issues 32-bit addresses, the addressable memory space is 4G bytes.
- Physical main memory in a computer is generally not as large as the entire possible addressable space.
  - Physical memory typically ranges from a few hundred megabytes to 1G bytes.
- Large programs that cannot fit completely into the main memory have their parts stored on secondary storage devices such as magnetic disks.
  - Pieces of programs must be transferred to the main memory from secondary storage before they can be executed.

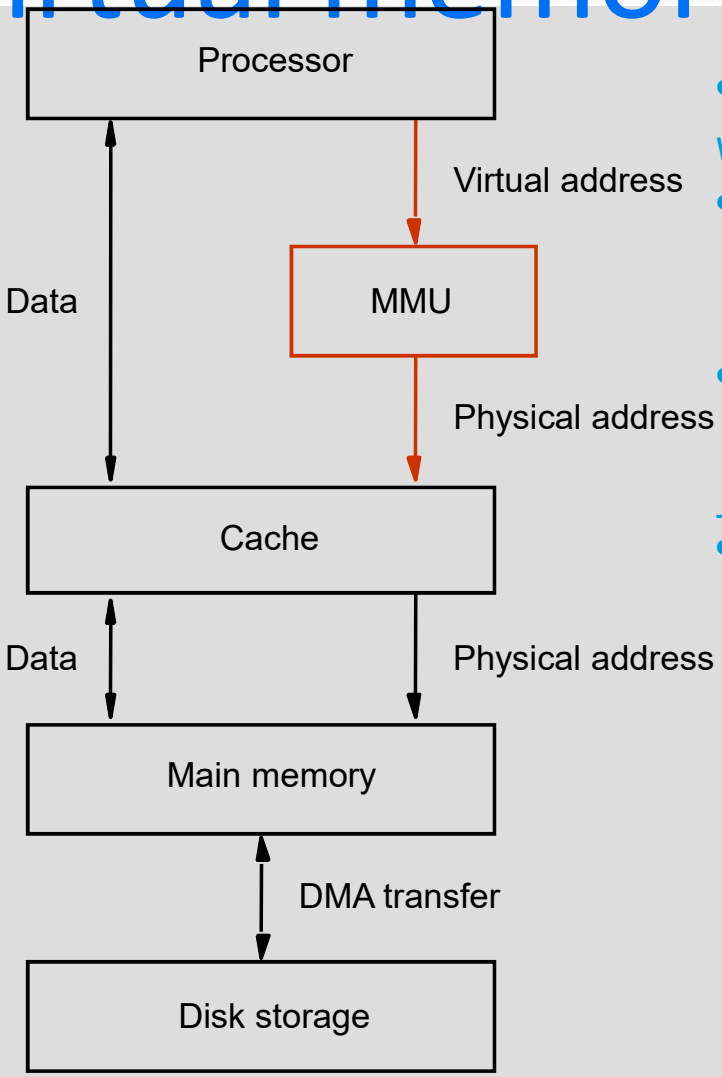
Virtual memories (contd..)

- When a new piece of a program is to be transferred to the main memory, and the main memory is full, then some other piece in the main memory must be replaced.
  - Recall this is very similar to what we studied in case of cache memories.
- Operating system automatically transfers data between the main memory and secondary storage.
  - Application programmer need not be concerned with this transfer.
  - Also, application programmer does not need to be aware of the limitations imposed by the available physical memory.

Virtual memories (contd..)

- Techniques that automatically move program and data between main memory and secondary storage when they are required for execution are called virtual-memory techniques.
- Programs and processors reference an instruction or data independent of the size of the main memory.
- Processor issues binary addresses for instructions and data.
  - These binary addresses are called logical or virtual addresses.
- Virtual addresses are translated into physical addresses by a combination of hardware and software subsystems.
  - If virtual address refers to a part of the program that is currently in the main memory, it is accessed immediately.
  - If the address refers to a part of the program that is not currently in the main memory, it is first transferred to the main memory before it can be used.

Virtual memory organization



- *Memory management unit (MMU) translates virtual addresses into physical addresses.*
- *If the desired data or instructions are in the main memory they are fetched as described previously.*
- *If the desired data or instructions are not in the main memory, they must be transferred from secondary storage to the main memory.*
- *MMU causes the operating system to bring the data from the secondary storage into the main memory.*

Address translation

- Assume that program and data are composed of fixed-length units called pages.
- A page consists of a block of words that occupy contiguous locations in the main memory.
- Page is a basic unit of information that is transferred between secondary storage and main memory.
- Size of a page commonly ranges from 2K to 16K bytes.
  - Pages should not be too small, because the access time of a secondary storage device is much larger than the main memory.
  - Pages should not be too large, else a large portion of the page may not be used, and it will occupy valuable space in the main memory.

Address translation (contd..)

- Concepts of virtual memory are similar to the concepts of cache memory.
- Cache memory:
  - Introduced to bridge the speed gap between the processor and the main memory.
  - Implemented in hardware.
- Virtual memory:
  - Introduced to bridge the speed gap between the main memory and secondary storage.
  - Implemented in part by software.

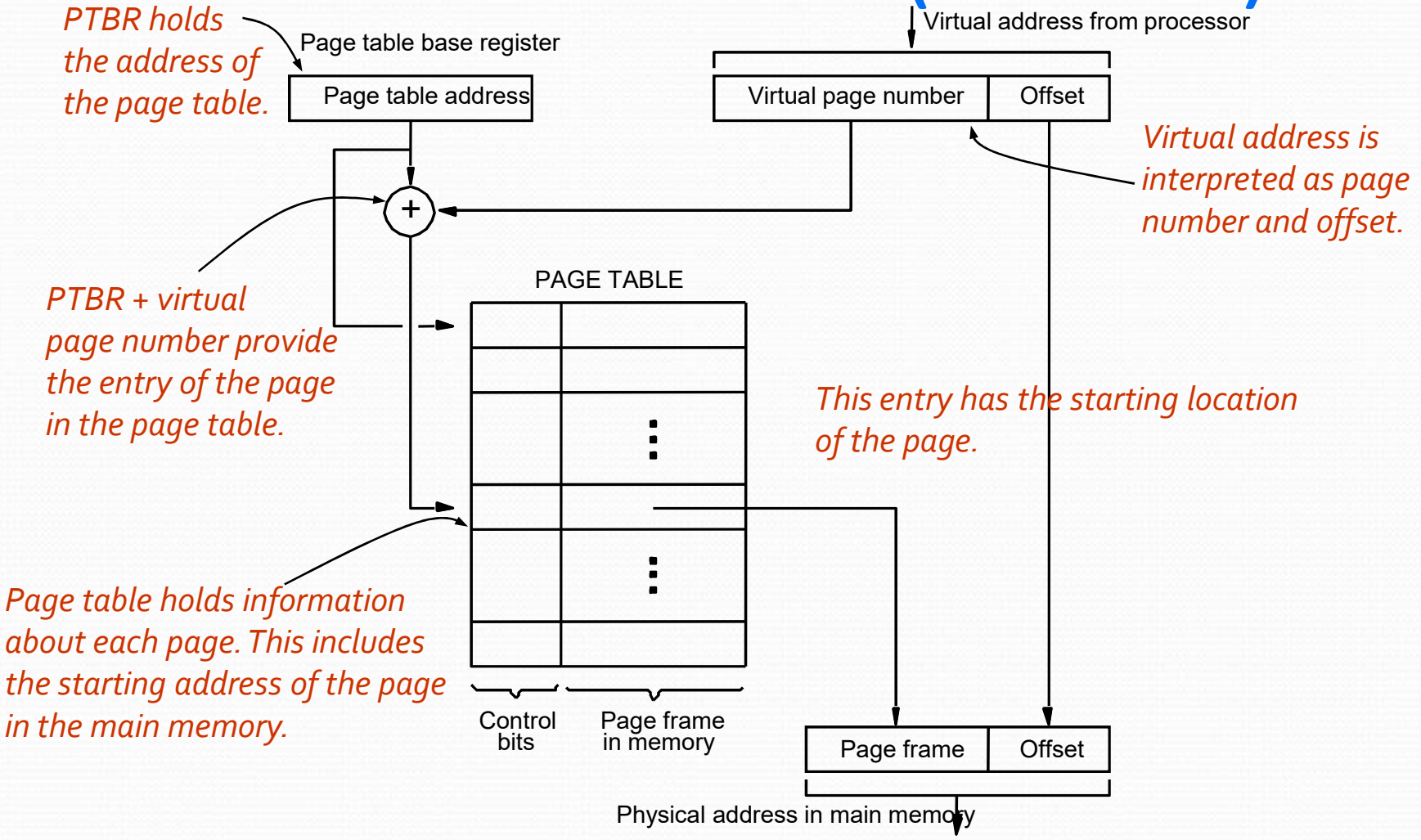
Address translation (contd..)

- Each virtual or logical address generated by a processor is interpreted as a virtual page number (high-order bits) plus an offset (low-order bits) that specifies the location of a particular byte within that page.
- Information about the main memory location of each page is kept in the page table.
  - Main memory address where the page is stored.
  - Current status of the page.
- Area of the main memory that can hold a page is called as page frame.
- Starting address of the page table is kept in a page table base register.

Address translation (contd..)

- Virtual page number generated by the processor is added to the contents of the page table base register.
  - This provides the address of the corresponding entry in the page table.
- The contents of this location in the page table give the starting address of the page if the page is currently in the main memory.

Address translation (contd..)



Address translation (contd..)

- Page table entry for a page also includes some control bits which describe the status of the page while it is in the main memory.
- One bit indicates the validity of the page.
  - Indicates whether the page is actually loaded into the main memory.
  - Allows the operating system to invalidate the page without actually removing it.
- One bit indicates whether the page has been modified during its residency in the main memory.
  - This bit determines whether the page should be written back to the disk when it is removed from the main memory.
  - Similar to the dirty or modified bit in case of cache memory.

Address translation (contd..)

- Other control bits for various other types of restrictions that may be imposed.
  - For example, a program may only have read permission for a page, but not write or modify permissions.

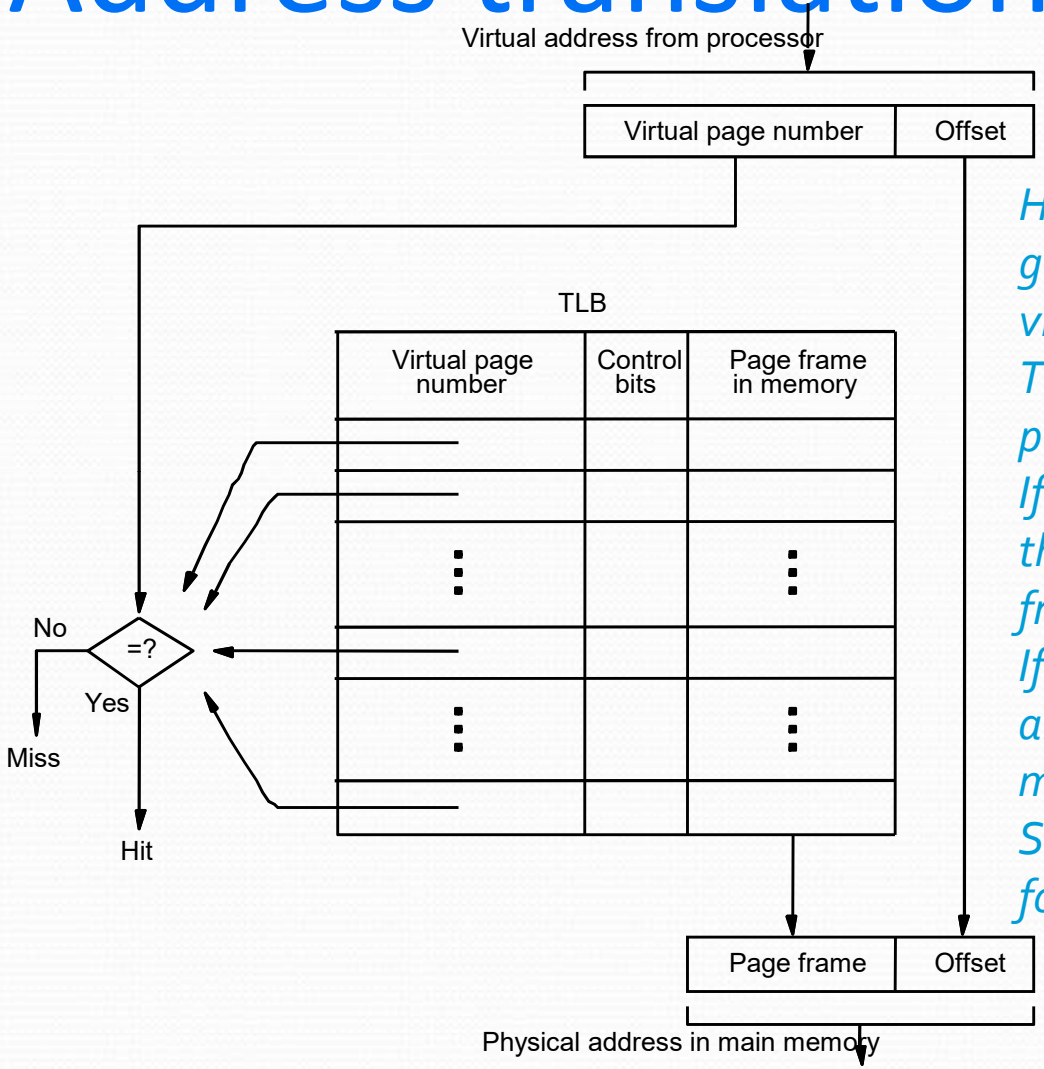
Address translation (contd..)

- Where should the page table be located?
- Recall that the page table is used by the MMU for every read and write access to the memory.
  - Ideal location for the page table is within the MMU.
- Page table is quite large.
- MMU is implemented as part of the processor chip.
- Impossible to include a complete page table on the chip.
- Page table is kept in the main memory.
- A copy of a small portion of the page table can be accommodated within the MMU.
  - Portion consists of page table entries that correspond to the most recently accessed pages.

Address translation (contd..)

- A small cache called as Translation Lookaside Buffer (TLB) is included in the MMU.
  - TLB holds page table entries of the most recently accessed pages.
- Recall that cache memory holds most recently accessed blocks from the main memory.
  - Operation of the TLB and page table in the main memory is similar to the operation of the cache and main memory.
- Page table entry for a page includes:
  - Address of the page frame where the page resides in the main memory.
  - Some control bits.
- In addition to the above for each page, TLB must hold the virtual page number for each page.

Address translation (contd..)



Associative-mapped TLB

High-order bits of the virtual address generated by the processor select the virtual page.
These bits are compared to the virtual page numbers in the TLB.
If there is a match, a hit occurs and the corresponding address of the page frame is read.
If there is no match, a miss occurs and the page table within the main memory must be consulted.
Set-associative mapped TLBs are found in commercial processors.

Address translation (contd..)

- How to keep the entries of the TLB coherent with the contents of the page table in the main memory?
- Operating system may change the contents of the page table in the main memory.
  - Simultaneously it must also invalidate the corresponding entries in the TLB.
- A control bit is provided in the TLB to invalidate an entry.
- If an entry is invalidated, then the TLB gets the information for that entry from the page table.
  - Follows the same process that it would follow if the entry is not found in the TLB or if a "miss" occurs.

Address translation (contd..)

- What happens if a program generates an access to a page that is not in the main memory?
- In this case, a page fault is said to occur.
  - Whole page must be brought into the main memory from the disk, before the execution can proceed.
- Upon detecting a page fault by the MMU, following actions occur:
  - MMU asks the operating system to intervene by raising an exception.
  - Processing of the active task which caused the page fault is interrupted.
  - Control is transferred to the operating system.
  - Operating system copies the requested page from secondary storage to the main memory.
  - Once the page is copied, control is returned to the task which was interrupted.

Address translation (contd..)

- Servicing of a page fault requires transferring the requested page from secondary storage to the main memory.
- This transfer may incur a long delay.
- While the page is being transferred the operating system may:
  - Suspend the execution of the task that caused the page fault.
  - Begin execution of another task whose pages are in the main memory.
- Enables efficient use of the processor.

Address translation (contd..)

- How to ensure that the interrupted task can continue correctly when it resumes execution?
- There are two possibilities:
  - Execution of the interrupted task must continue from the point where it was interrupted.
  - The instruction must be restarted.
- Which specific option is followed depends on the design of the processor.

Address translation (contd..)

- When a new page is to be brought into the main memory from secondary storage, the main memory may be full.
  - Some page from the main memory must be replaced with this new page.
- How to choose which page to replace?
  - This is similar to the replacement that occurs when the cache is full.
  - The principle of locality of reference (?) can also be applied here.
  - A replacement strategy similar to LRU can be applied.
- Since the size of the main memory is relatively larger compared to cache, a relatively large amount of programs and data can be held in the main memory.
  - Minimizes the frequency of transfers between secondary storage and main memory.

Address translation (contd..)

- A page may be modified during its residency in the main memory.
- When should the page be written back to the secondary storage?
- Recall that we encountered a similar problem in the context of cache and main memory:
  - Write-through protocol(?)
  - Write-back protocol(?)
- Write-through protocol cannot be used, since it will incur a long delay each time a small amount of data is written to the disk.

Memory Management

The Memory System

Memory management

- Operating system is concerned with transferring programs and data between secondary storage and main memory.
- Operating system needs memory routines in addition to the other routines.
- Operating system routines are assembled into a virtual address space called system space.
- System space is separate from the space in which user application programs reside.
  - This is user space.
- Virtual address space is divided into one

  system space + several user spaces.

Memory management (contd..)

- Recall that the Memory Management Unit (MMU) translates logical or virtual addresses into physical addresses.
- MMU uses the contents of the page table base register to determine the address of the page table to be used in the translation.
  - Changing the contents of the page table base register can enable us to use a different page table, and switch from one space to another.
- At any given time, the page table base register can point to one page table.
  - Thus, only one page table can be used in the translation process at a given time.
  - Pages belonging to only one space are accessible at any given time.

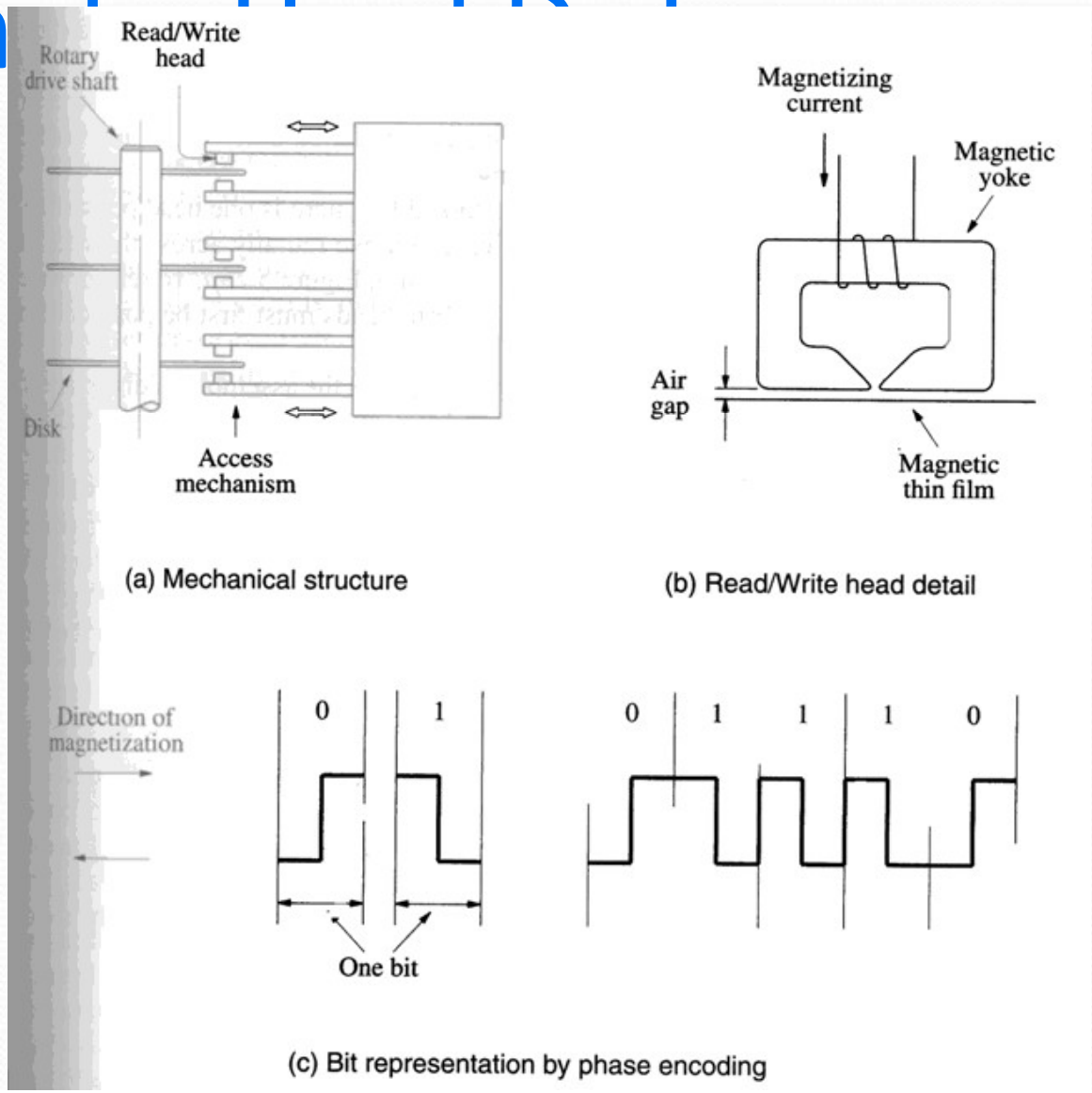
Memory management (contd..)

- When multiple, independent user programs coexist in the main memory, how to ensure that one program does not modify/destroy the contents of the other?
- Processor usually has two states of operation:
  - Supervisor state.
  - User state.
- Supervisor state:
  - Operating system routines are executed.
- User state:
  - User programs are executed.
  - Certain privileged instructions cannot be executed in user state.
  - These privileged instructions include the ones which change page table base register.
  - Prevents one user from accessing the space of other users.

Secondary Storage

The Memory System

Magn



Disk

Disk drive

Disk controller

Organization of Data on a Disk

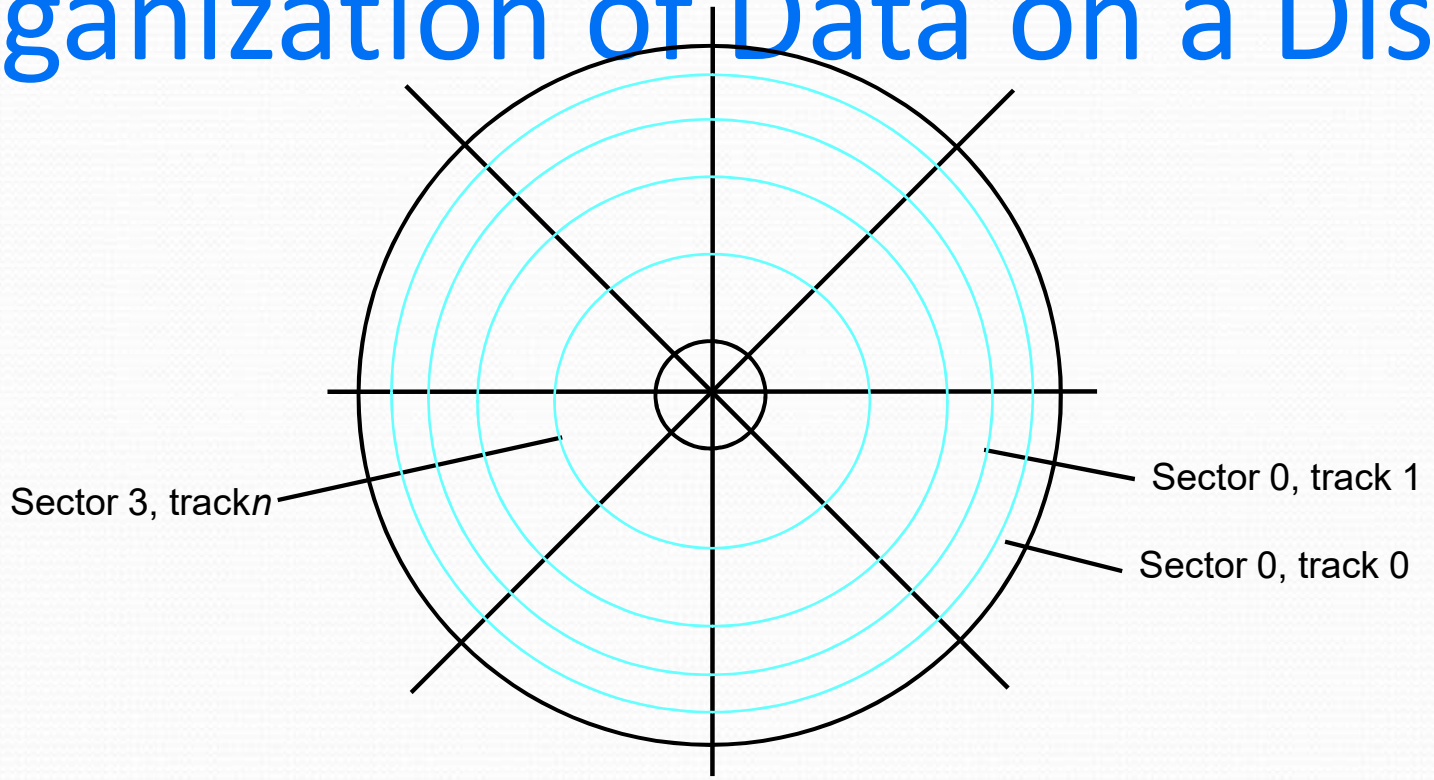


Figure 5.30. Organization of one surface of a disk.

Access Data on a Disk

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

Disk Controller

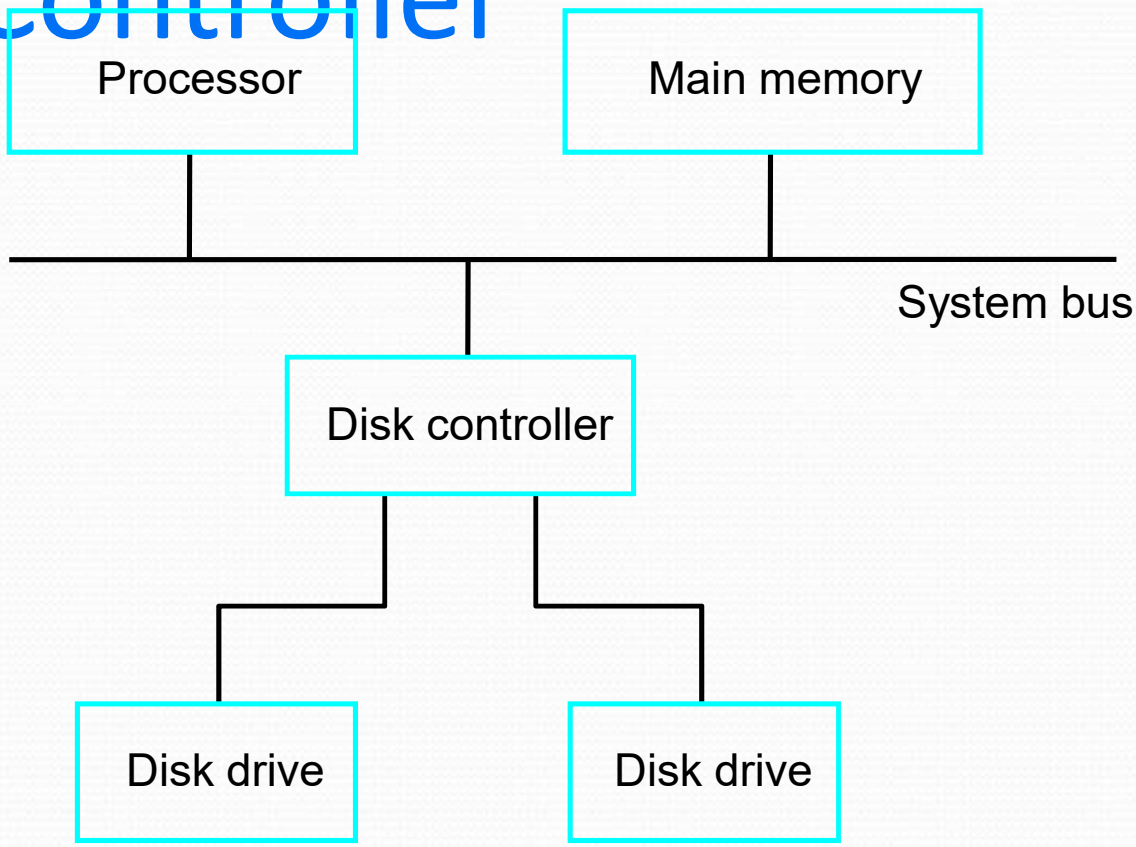


Figure 5.31. Disks connected to the system bus.

Disk Controller

- Seek
- Read
- Write
- Error checking

RAID Disk Arrays

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

Optical Disks

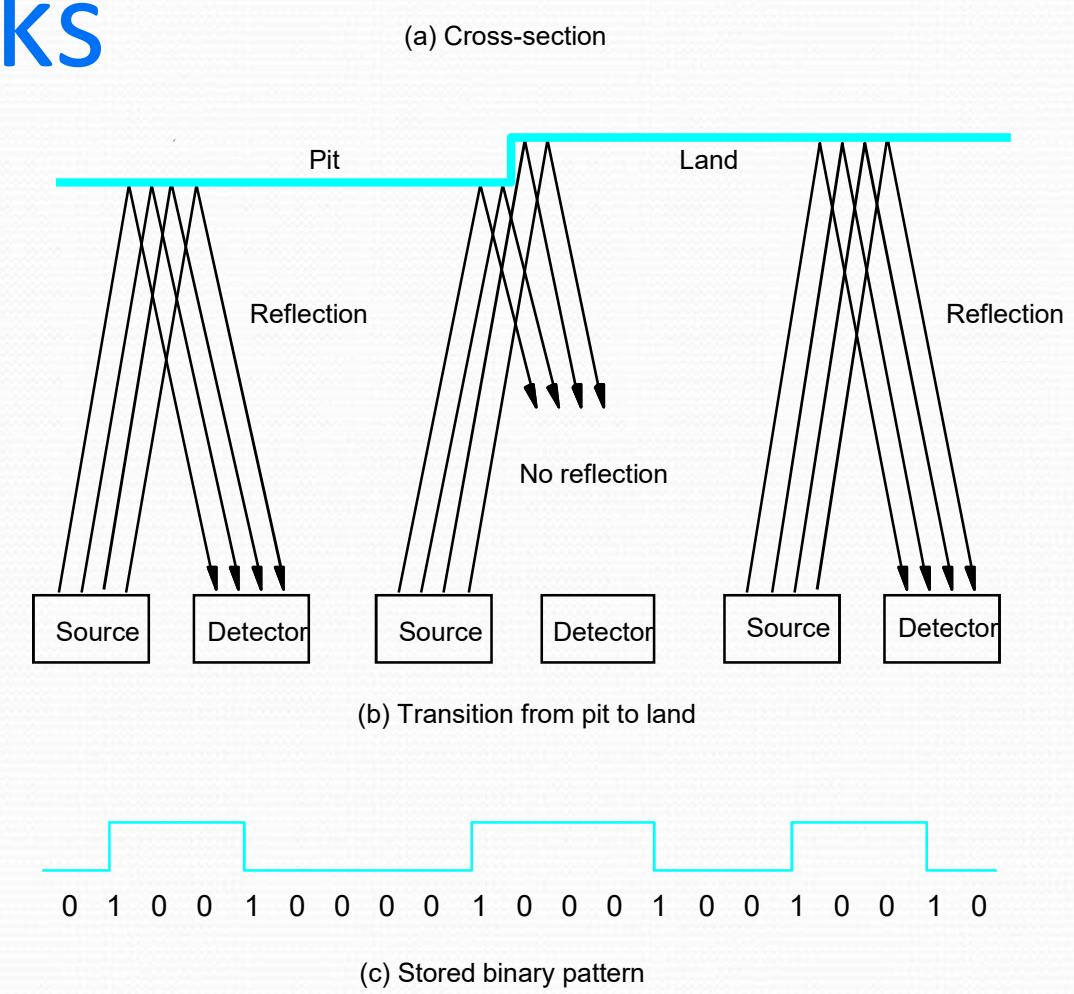


Figure 5.32. Optical disk.

Optical Disks

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

Magnetic Tape Systems

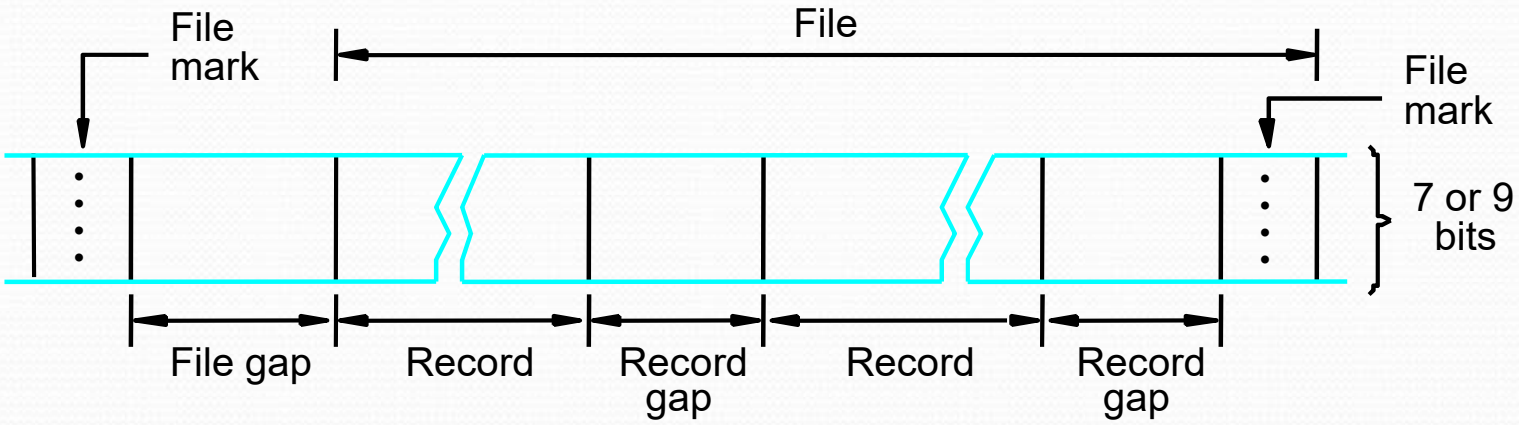


Figure 5.33. Organization of data on magnetic tape.