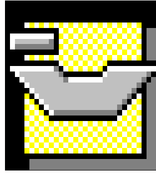


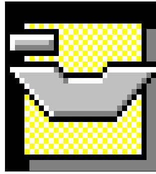
COMPUTERS ORGANIZATION

2ND YEAR COMPUTE SCIENCE MANAGEMENT ENGINEERING

UNIT 3 - ARITMETHIC-LOGIC UNIT

JOSÉ GARCÍA RODRÍGUEZ
JOSÉ ANTONIO SERRA PÉREZ

departamento de tecnología informática y computación
UNIVERSIDAD DE ALICANTE

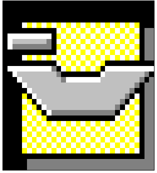


ALU

Arithmetic Logic Unit

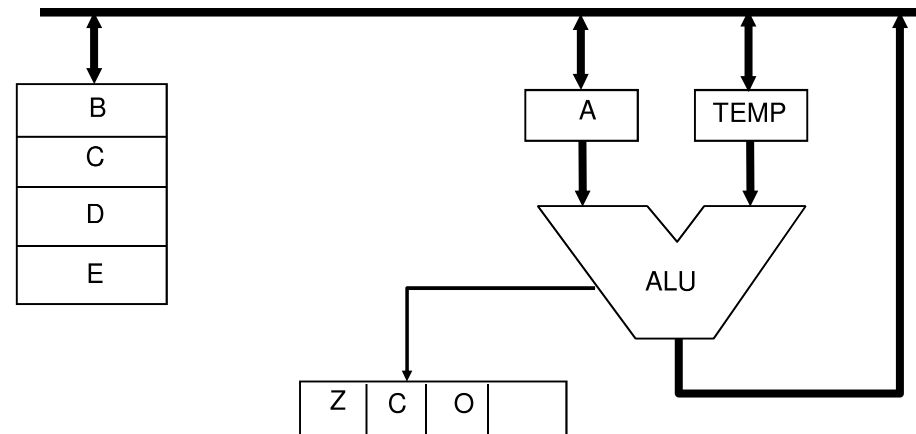
- Introduction
- Logic Operations
- Addition and subtraction
  - Carry propagated adder (CPA)
  - Adder-Subtractor circuit
  - Overflow
  - Carry Look-ahead Adder (CLA)
- Multiplication
  - Binary multiplication without sign
  - Binary multiplication with sign
  - Booth Algorithm
- Division
- Conclusions

departamento de tecnología informática y computación
UNIVERSIDAD DE ALICANTE



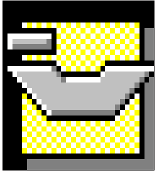
Introduction

Introduction



- Arithmetic and logic operator (one or more)
- Accumulator
- One or more temporal registers
- Result flags
  - Carry (C)
  - Negative (N)
  - Overflow (O or V)
  - Zero (Z)

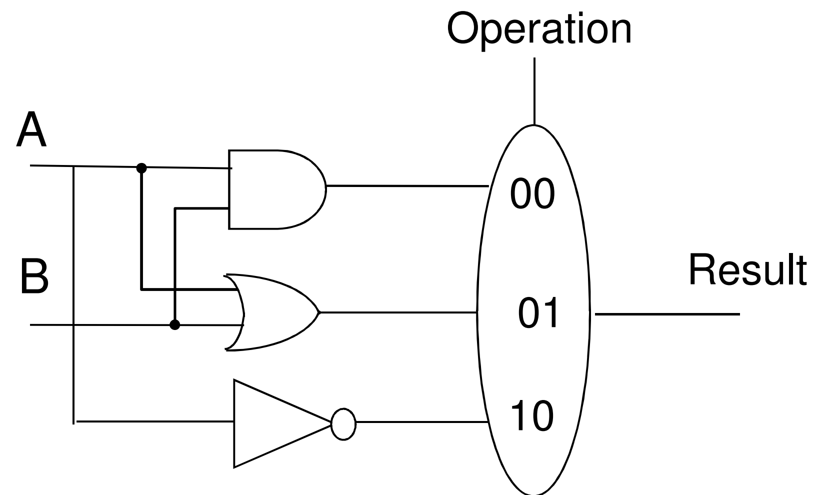
departamento de tecnología informática y computación
UNIVERSIDAD DE ALICANTE



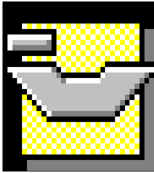
Logic Operations

Logic operations

- Easy to implement ⇒ Direct correspondence with the Hardware.
- Logic gates AND, OR, XOR, INVERTER,...



departamento de tecnología informática y computación
UNIVERSIDAD DE ALICANTE



Half-adder

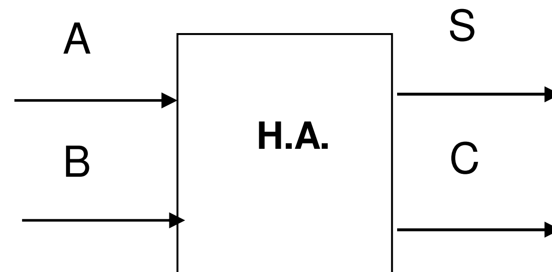
Addition and subtraction

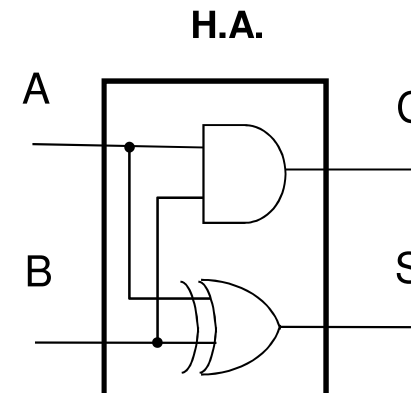
Binary half-adder (H.A.)

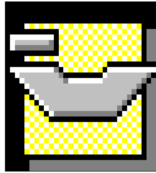
Inputs		Outputs	
X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S = \overline{X} \cdot Y + X \cdot \overline{Y} = X \oplus Y$$

$$C = X \cdot Y$$

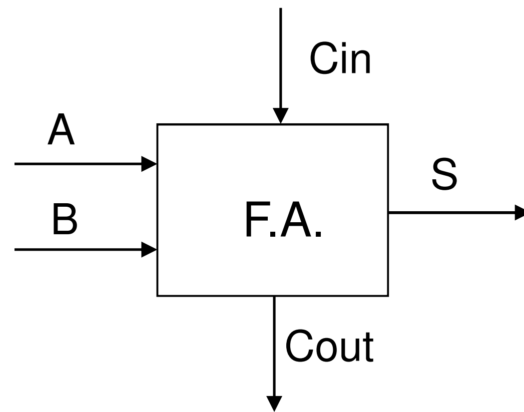






Addition and Subtraction

Full Adder (F.A.)



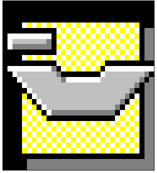
Inputs			Outputs	
A	**B**	**Cin**	**S**	**Cout**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = \overline{A} \cdot \overline{B} \cdot Cin + \overline{A} \cdot B \cdot \overline{Cin} + A \cdot \overline{B} \cdot \overline{Cin} + A \cdot B \cdot Cin$$

$$Cout = A \cdot B + A \cdot Cin + B \cdot Cin$$

$$S = (A \oplus B) \oplus Cin$$

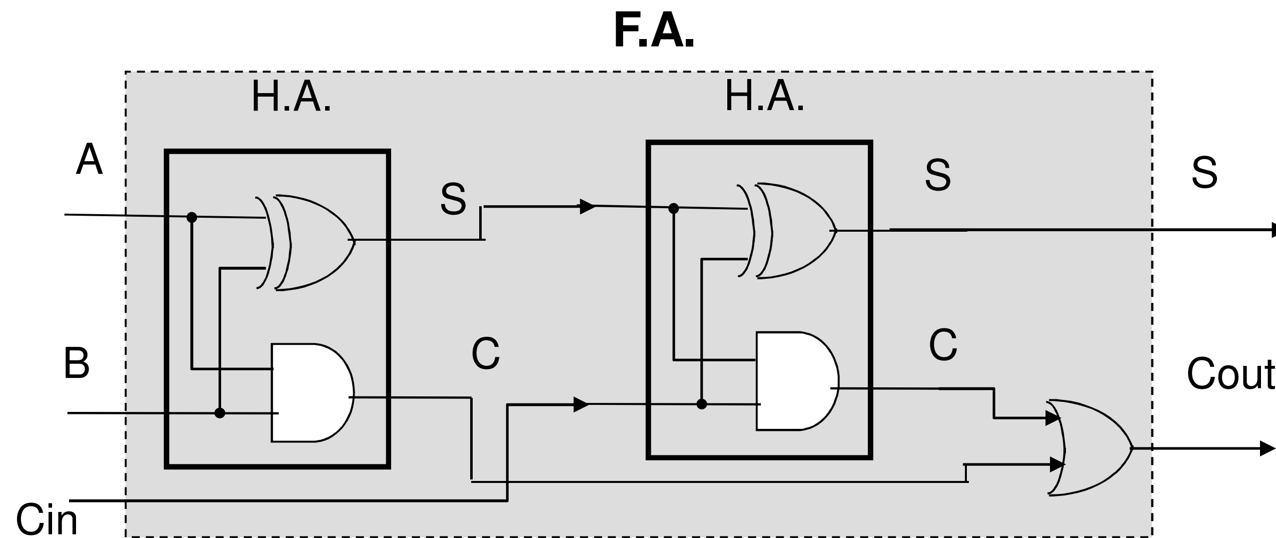
$$Cout = AB + Cin(A \oplus B)$$

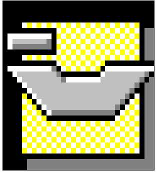


Addition and Subtraction

Full Adder (F.A.)

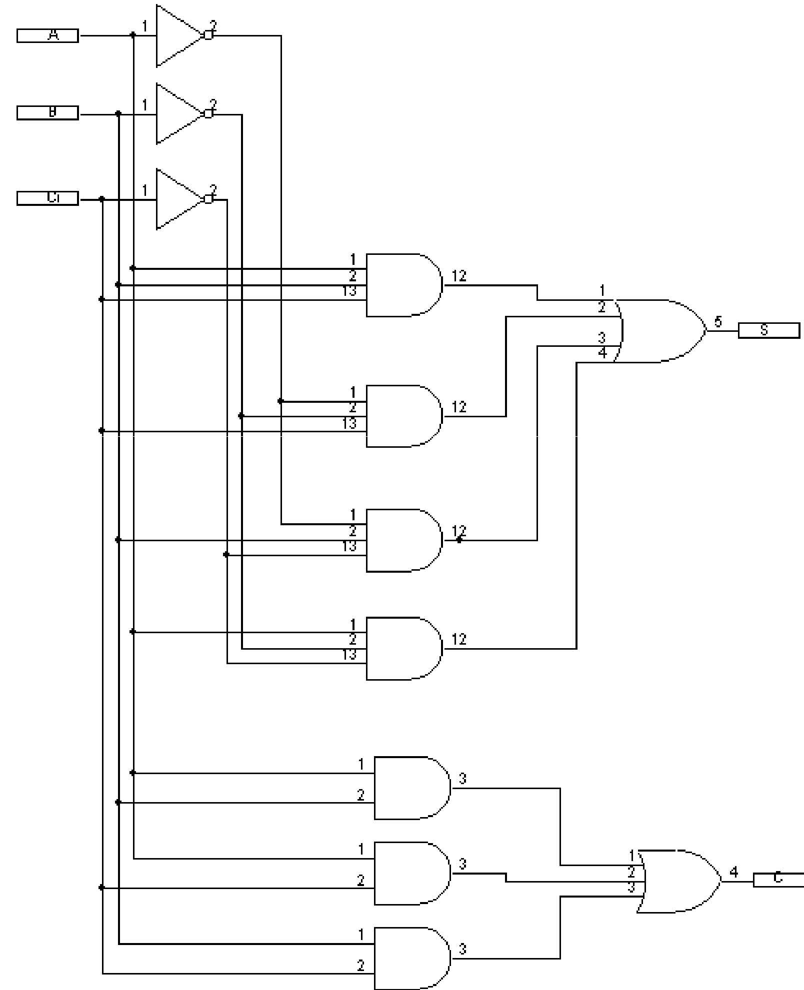
- Usign half-adders (H.A.)



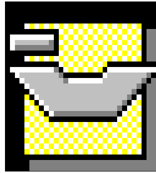


Full Adder (F.A.)

- Usign gates

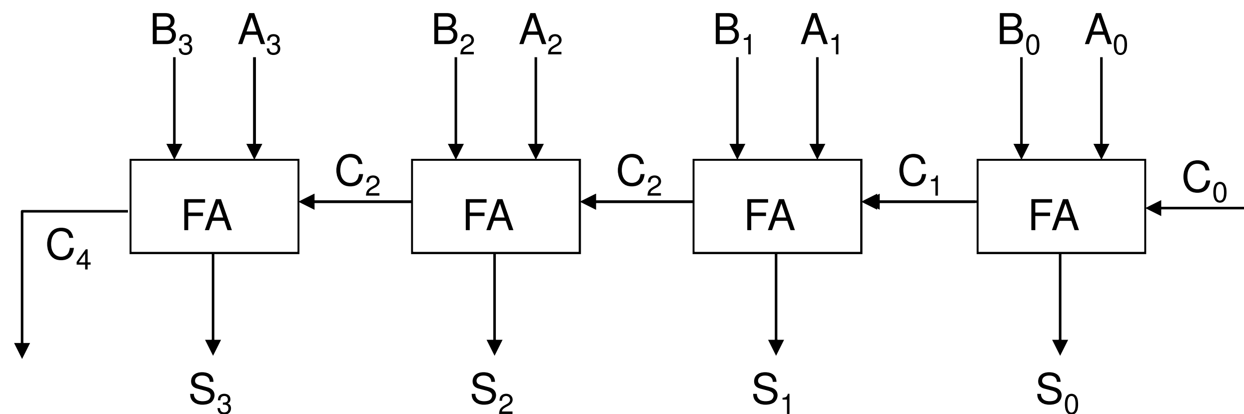


departamento de tecnología informática y computación
UNIVERSIDAD DE ALICANTE

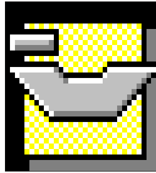


Carry Propagated Adder

- To add two numbers of n bits, n full adder should be placed one after another.
- Carry is propagated from one stage to the next one: Carry Propagated Adder.



departamento de tecnología informática y computación
UNIVERSIDAD DE ALICANTE



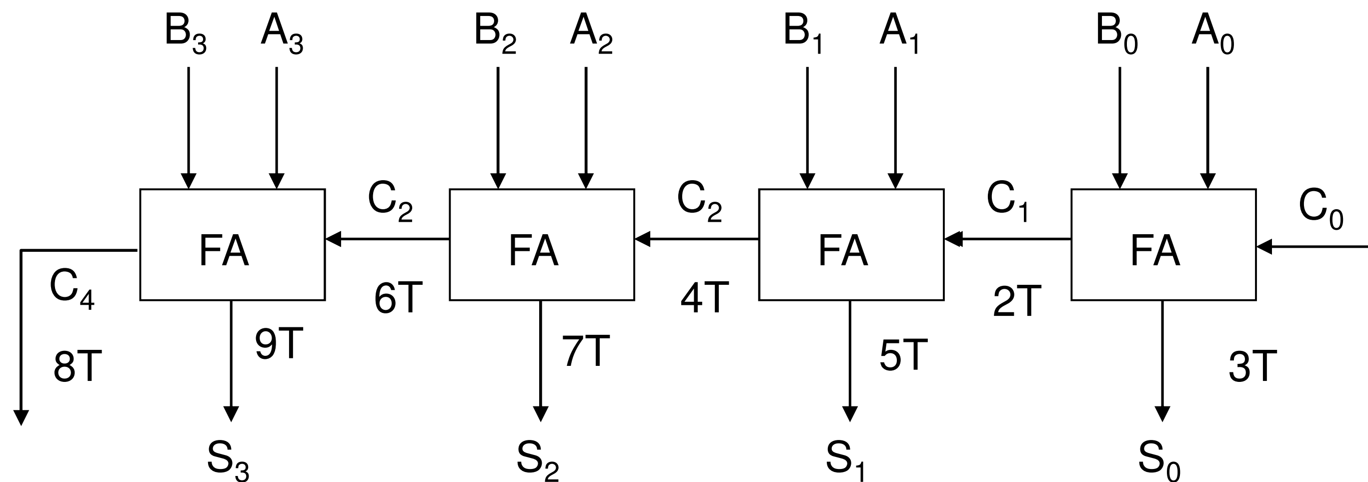
Addition and Subtraction

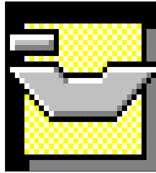
Carry Propagated Adder

- Adders built with logic gates using the expression:

$$S = \overline{A} \cdot \overline{B} \cdot Cin + \overline{A} \cdot B \cdot \overline{Cin} + A \cdot \overline{B} \cdot \overline{Cin} + A \cdot B \cdot Cin$$

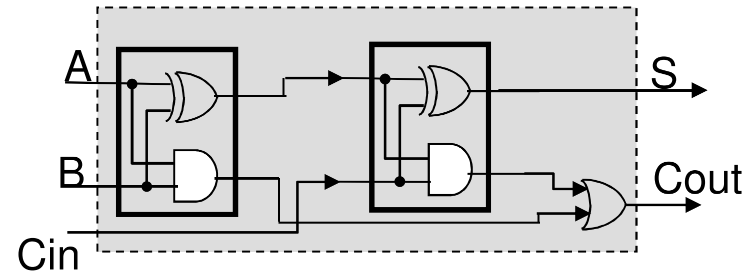
$$Cout = A \cdot B + A \cdot Cin + B \cdot Cin$$

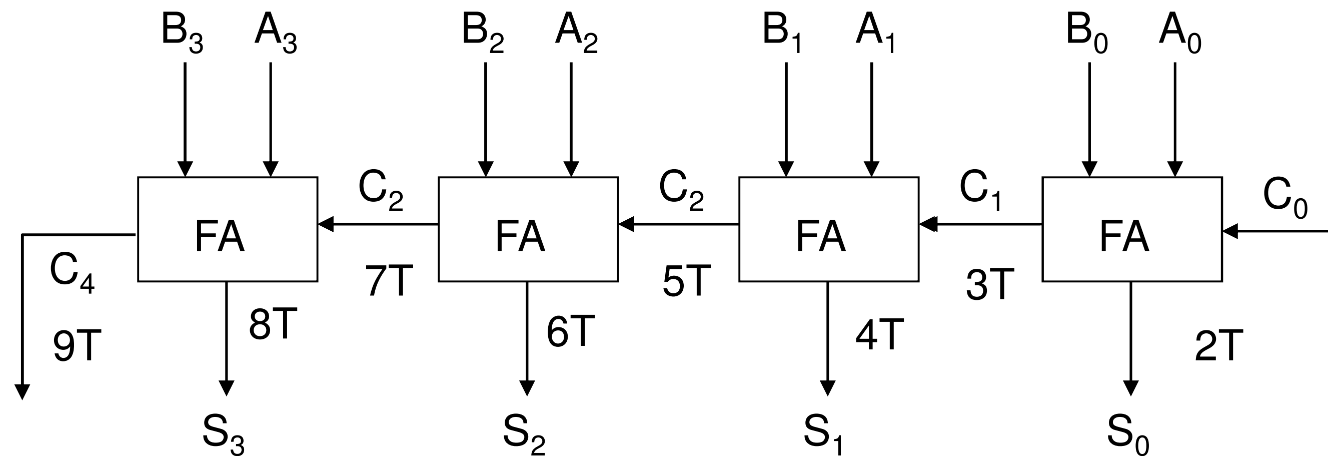




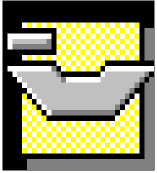
Carry Propagated Adder

- Full adders built with half-adders





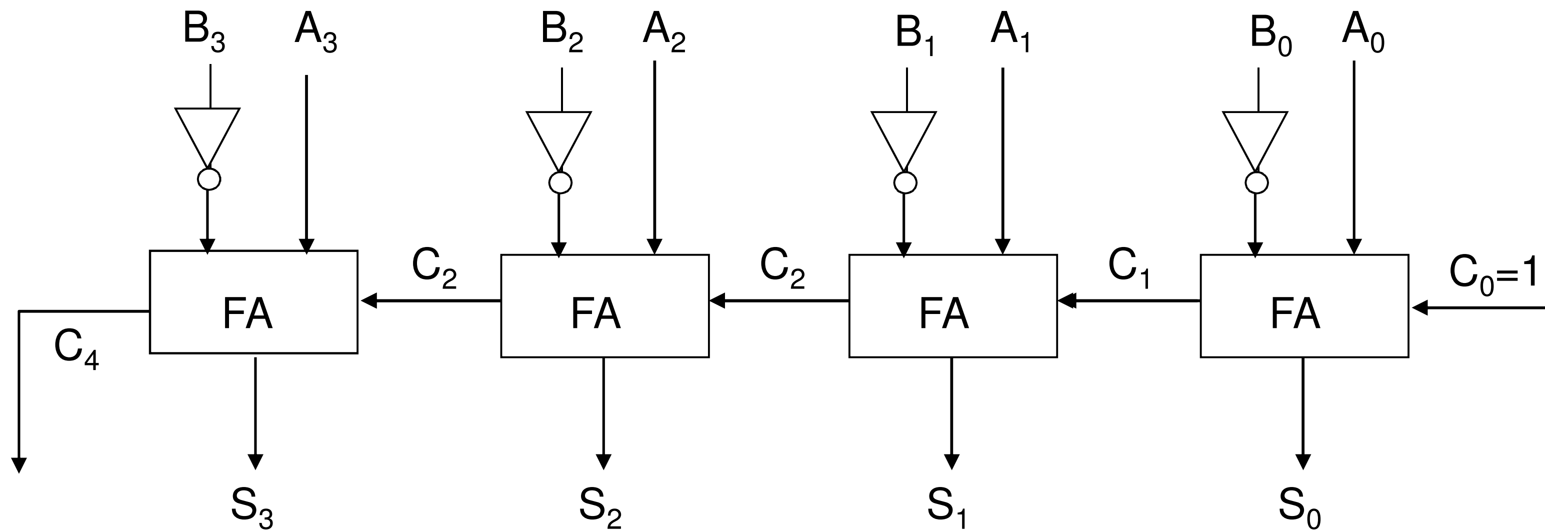
$$Total_Time = (2 \cdot n + 1)T$$

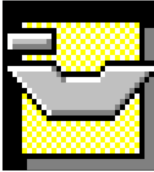


Addition and Subtraction

Subtractor circuit

- The circuit works with numbers in two's complement notation.
- A - B = A + (C1(B) + 1)



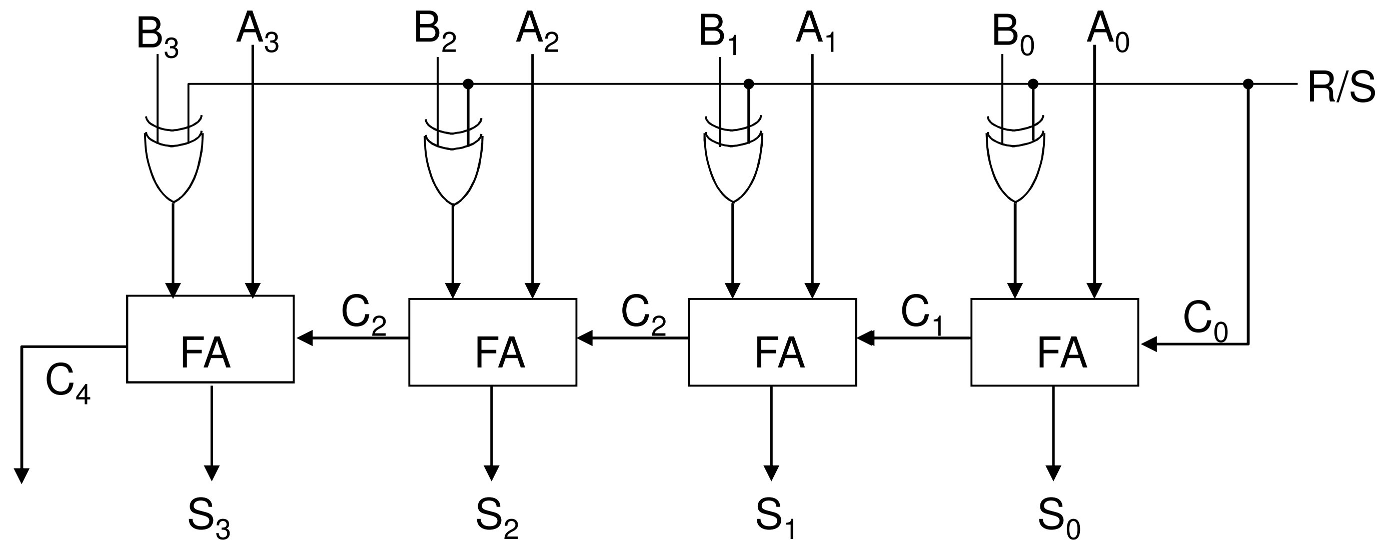


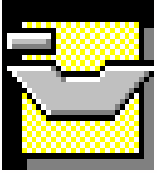
Addition and Subtraction

Adder-Subtractor Circuit

S/A	B_i	FA input
0	0	0
0	1	1
1	0	1
1	1	0

$$Total_Time = 2(n+1)T$$

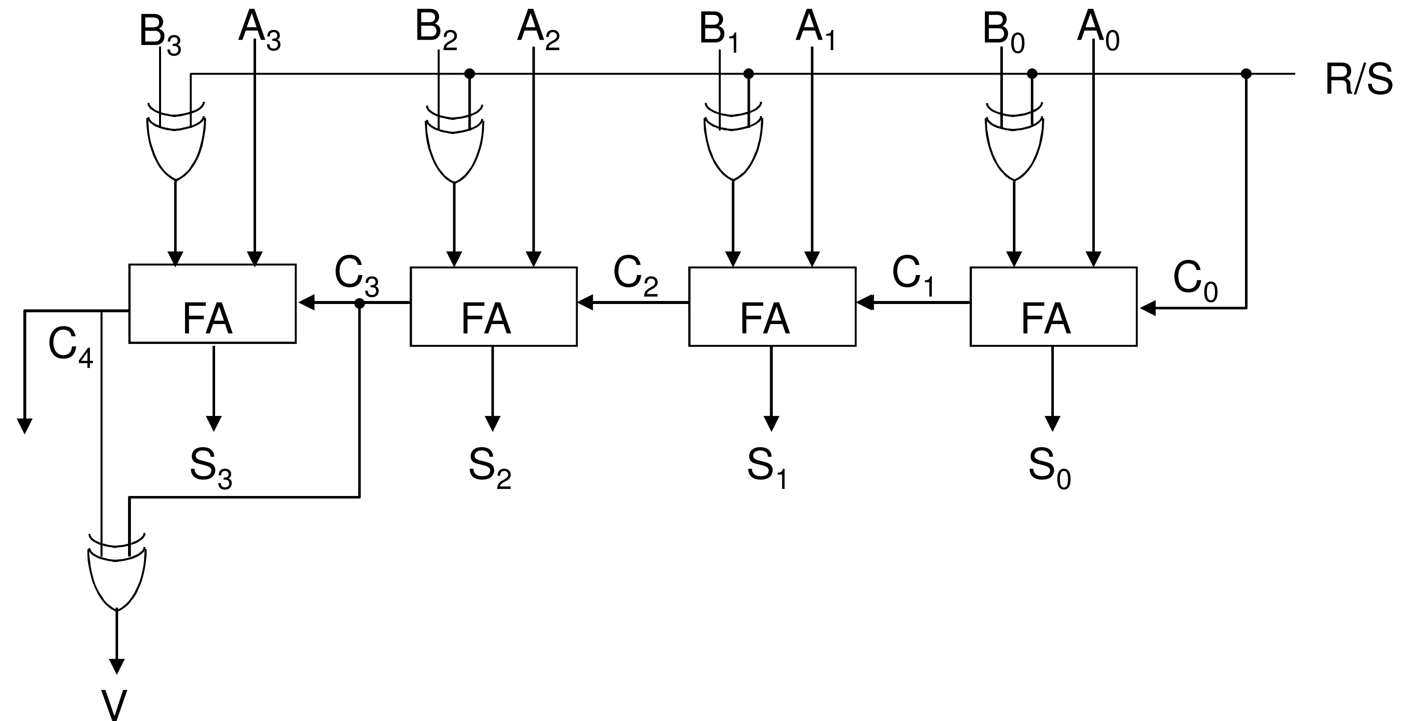




Addition and Subtraction

Overflow detection

Two's complement Adder-Substractor with overflow detection



Overflow Detection

Addition and Subtraction

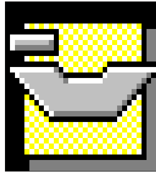
1. Case Addition of two positive numbers

	S				
C4	C3	C2	C1		
0	1	1	1		
	0	1	1	1	
	0	1	1	1	
	1	1	1	0	OV

2. Case Addition of two negative numbers

	S				
C4	C3	C2	C1		
1	0	1	1		
	1	0	0	1	
	1	0	1	1	
	0	1	0	0	OV

departamento de tecnología informática y computación
UNIVERSIDAD DE ALICANTE



Addition and Subtraction

Carry Look-ahead Adder

- Assume A and B to be 4 bits numbers
- Carry generating signal: $G_i = a_i \cdot b_i$
- Carry propagating signal: $\begin{cases} P_i = a_i \oplus b_i \\ P_i = a_i + b_i \end{cases}$
- Carry on stage i: $C_i = G_i + P_i \cdot C_{i-1}$
- Characterized for A and B:

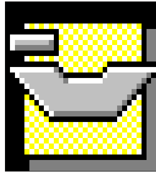
$$C_0 = G_0 + P_0 \cdot C_{-1}$$

$$C_1 = G_1 + P_1 \cdot C_0$$

$$C_2 = G_2 + P_2 \cdot C_1$$

$$C_3 = G_3 + P_3 \cdot C_2$$

departamento de tecnología informática y computación
UNIVERSIDAD DE ALICANTE



Addition and Subtraction

Carry Look-ahead Adder

- Expanding the expressions depending on C_{-1}:

$$C_0 = G_0 + P_0 \cdot C_{-1}$$

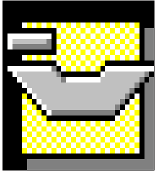
$$C_1 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_{-1}$$

$$C_2 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_{-1}$$

$$C_3 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_{-1}$$

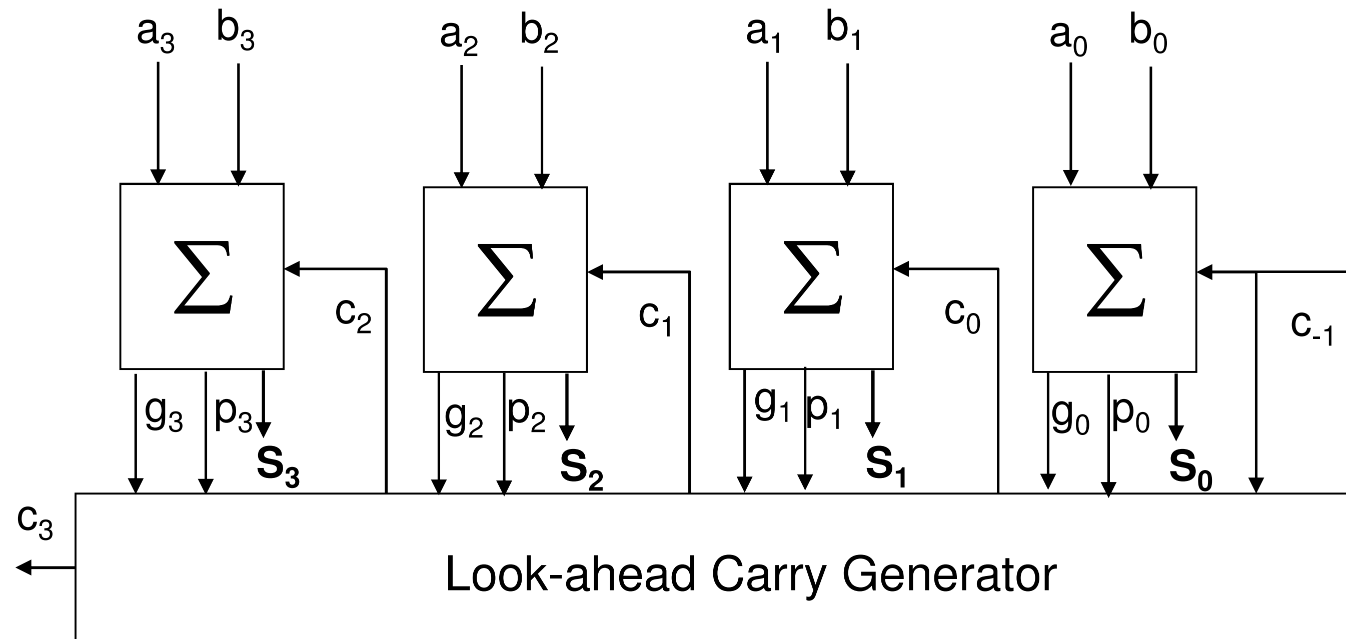
- Carries depend on a_i and b_i.
- These expressions are resolved as addition of products.
- Three leves of logic gates are needed to get each carry.

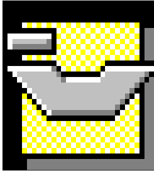
departamento de tecnología informática y computación
UNIVERSIDAD DE ALICANTE



Addition and Subtraction

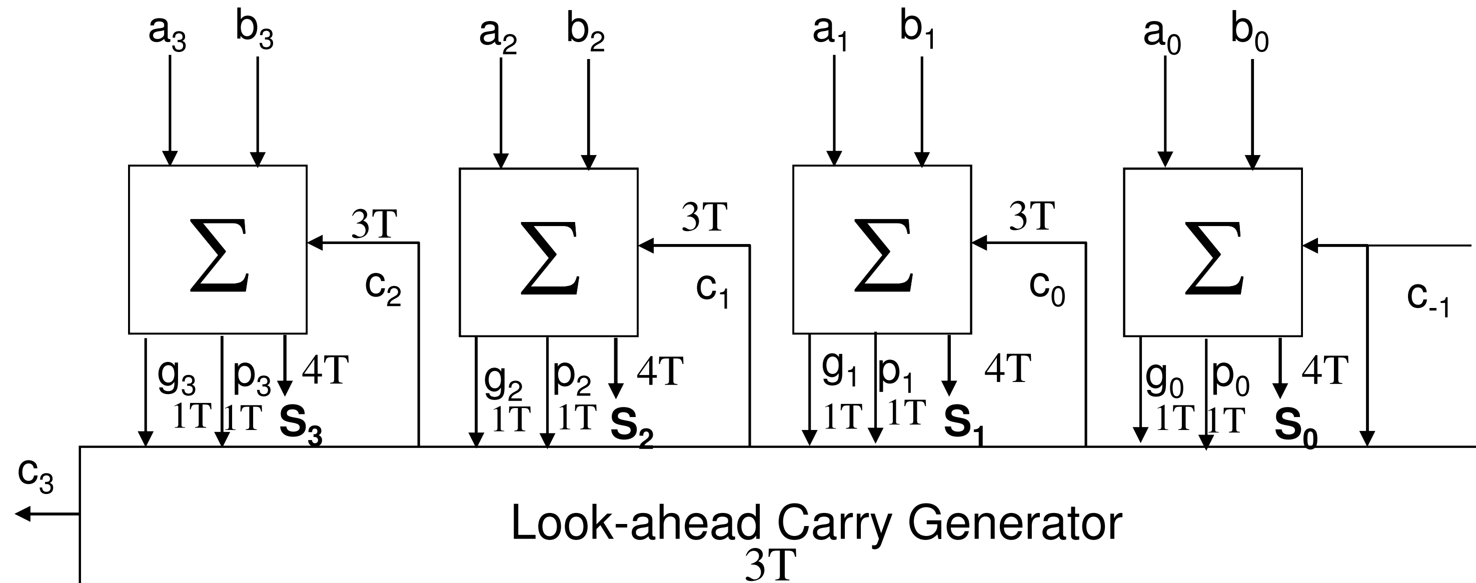
Carry Look-ahead Adder

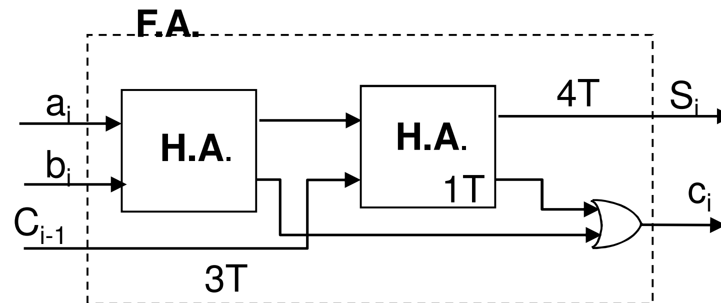




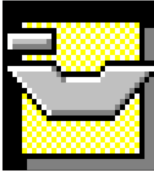
Addition and Subtraction

Carry Look-ahead Adder



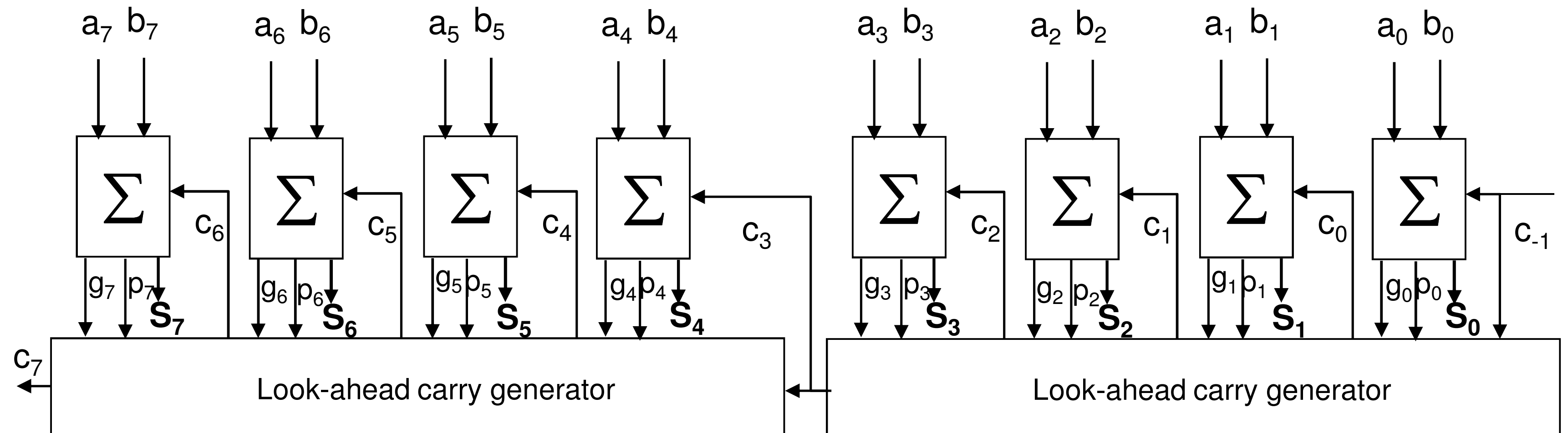


Adders built with half-adders



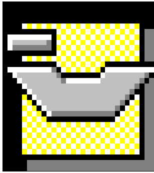
Addition and Subtraction

Example (8 bits CLA)



Calculate the delay in this CLA supposing that the adders are built with half-adders.

Compare the result against a 8 bits CPA.

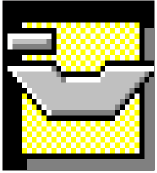


Addition and Subtraction

Example (4 bits CLA)

departamento de tecnología informática y computación

UNIVERSIDAD DE ALICANTE

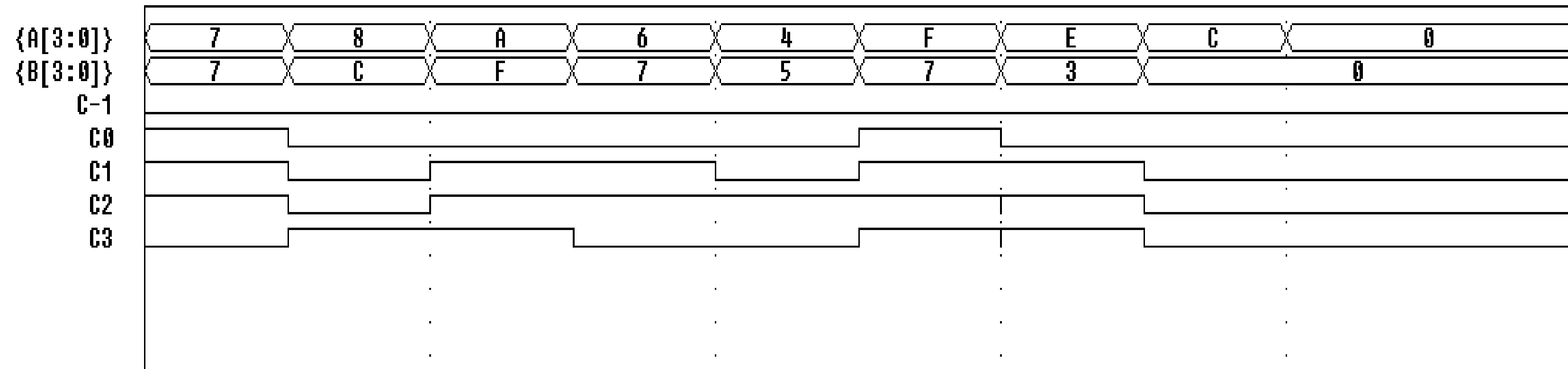


Addition and Subtraction

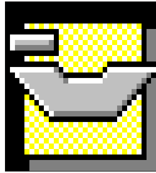
Example (4 bits CLA)

C3	C2	C1	C0	
0	1	1	1	
	0	1	1	1
	0	1	1	1
	1	1	1	0

C3	C2	C1	C0	
1	1	1		
	1	0	1	0
	1	1	1	1
	1	0	0	1



departamento de tecnología informática y computación
UNIVERSIDAD DE ALICANTE



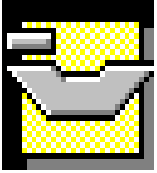
Multiplication

Multiplication

- Sums and shifts algorithm
- If multiplicand has n bits and multiplier has m bits, them the product will have n+m bits.
- Binary multiplication: simple as you only multiply by 1 or by 0.

Multiplicand				5	3	2
Multiplier				4	3	1
				5	3	2
		1	5	9	6	
	2	1	2	8		
Product	2	2	9	2	9	2

departamento de tecnología informática y computación
UNIVERSIDAD DE ALICANTE



Multiplication

Binary multipication without sign

Repeat n times

    If bit 0 of multiplier=1 then

        Sum the multiplicand to the left half of the product and place the result int the left half of the product

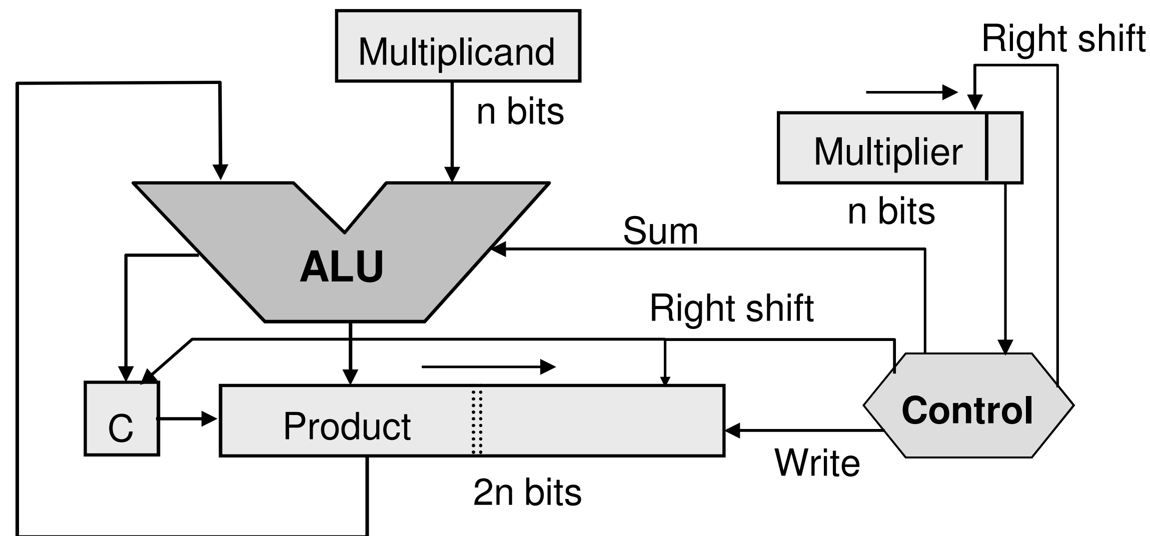
    End if

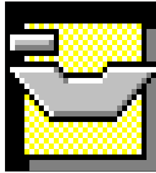
    Shift the product register 1 bit to the right

    Shift the multiplier register 1 bit to the right

End repeat

Preliminary Version



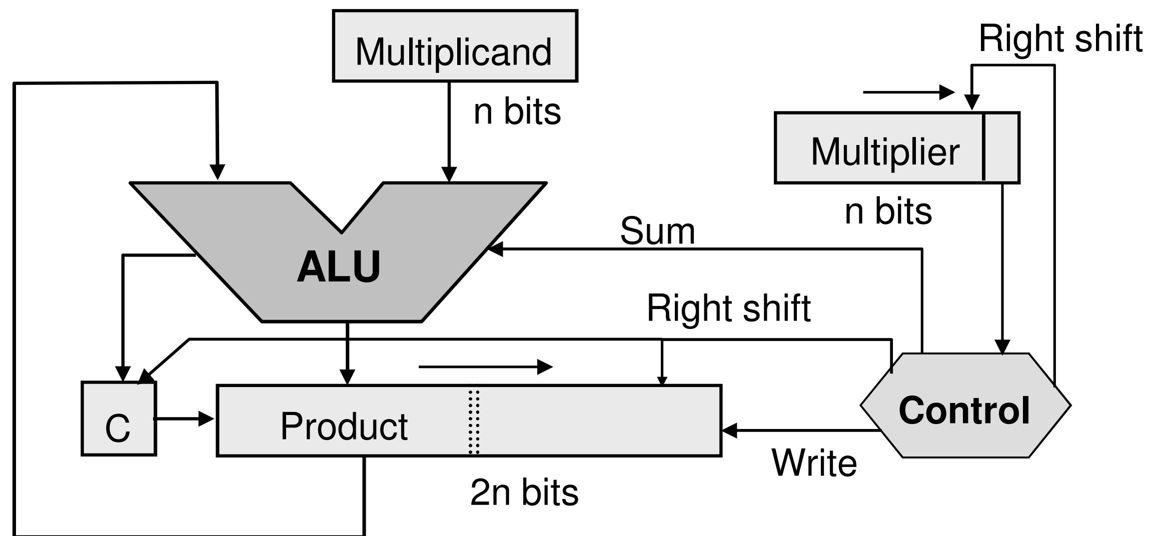


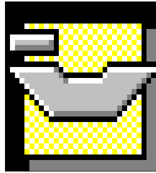
Multiplication

Binary multiplication without sign

Multiplicand					1	0	1	1
Multiplier					1	1	0	1
					1	0	1	1
				0	0	0	0	
			1	0	1	1		
		1	0	1	1			
Product	1	0	0	0	1	1	1	1

Preliminary Version





Multiplication

Binary multiplication without sign

Repeat n times

  If bit 0 of product register=1 then

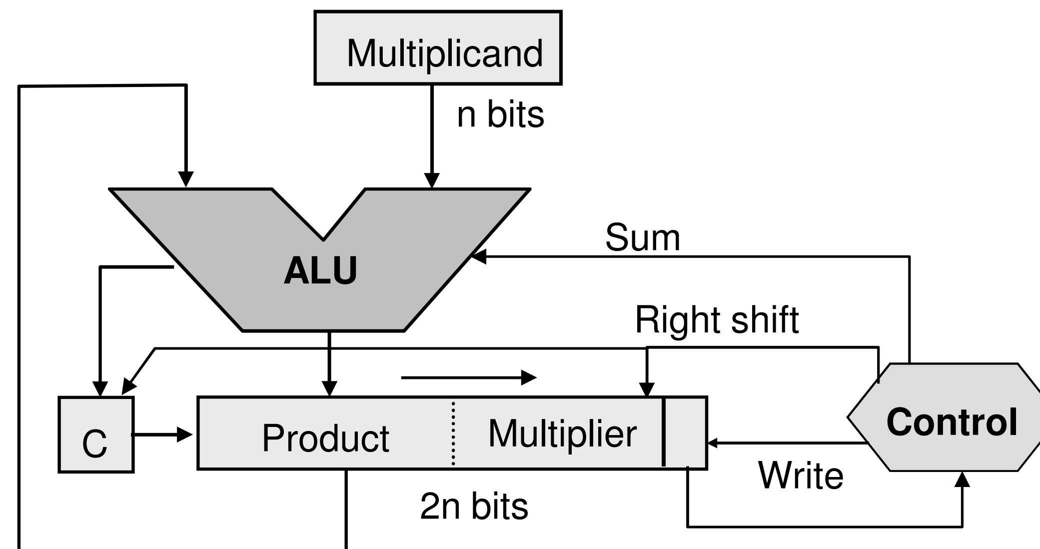
    Sum the multiplicand to the left half of the product and place the result in the left half of the product.

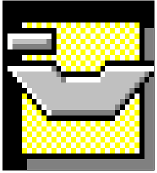
  End if

  Shift the product register 1 bit to the right

End repeat

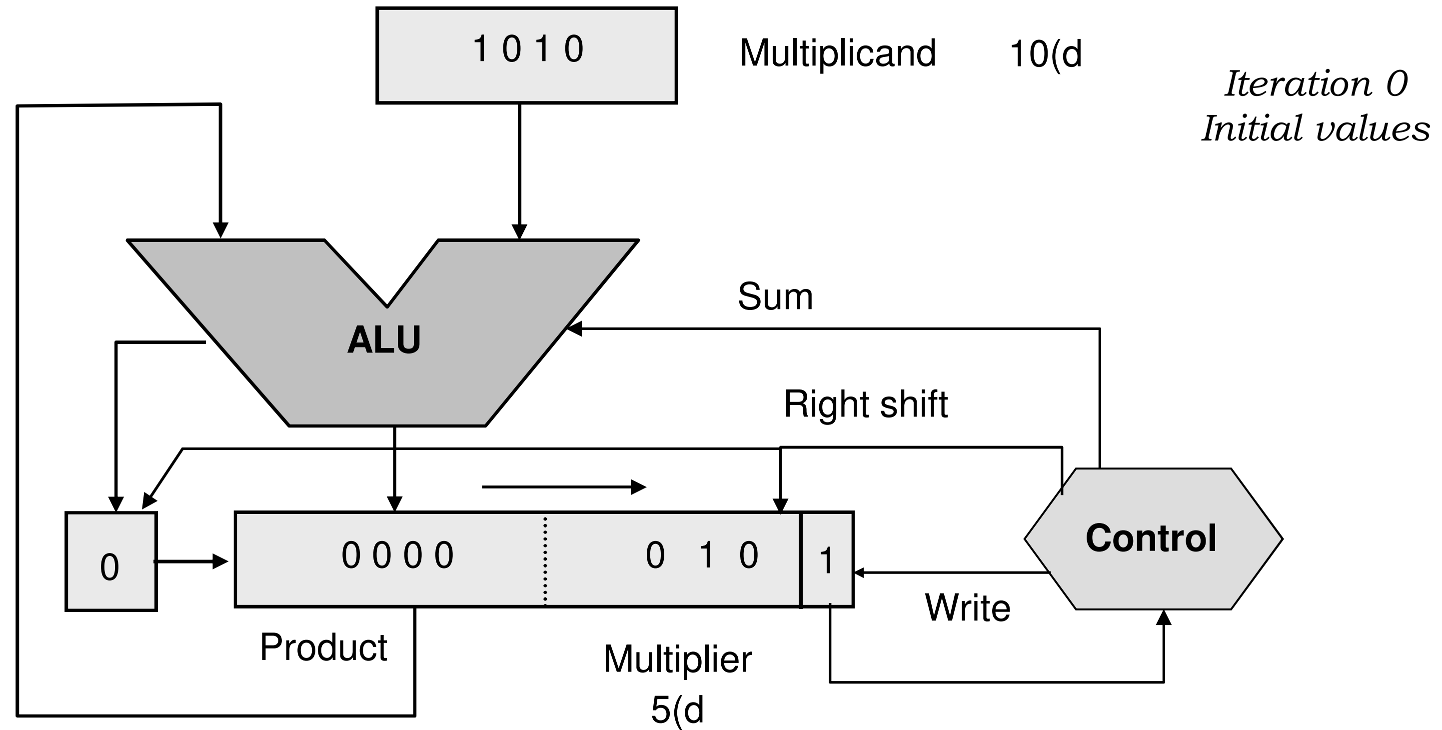
Final Version

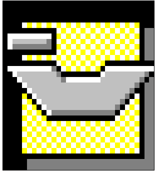




Multiplication

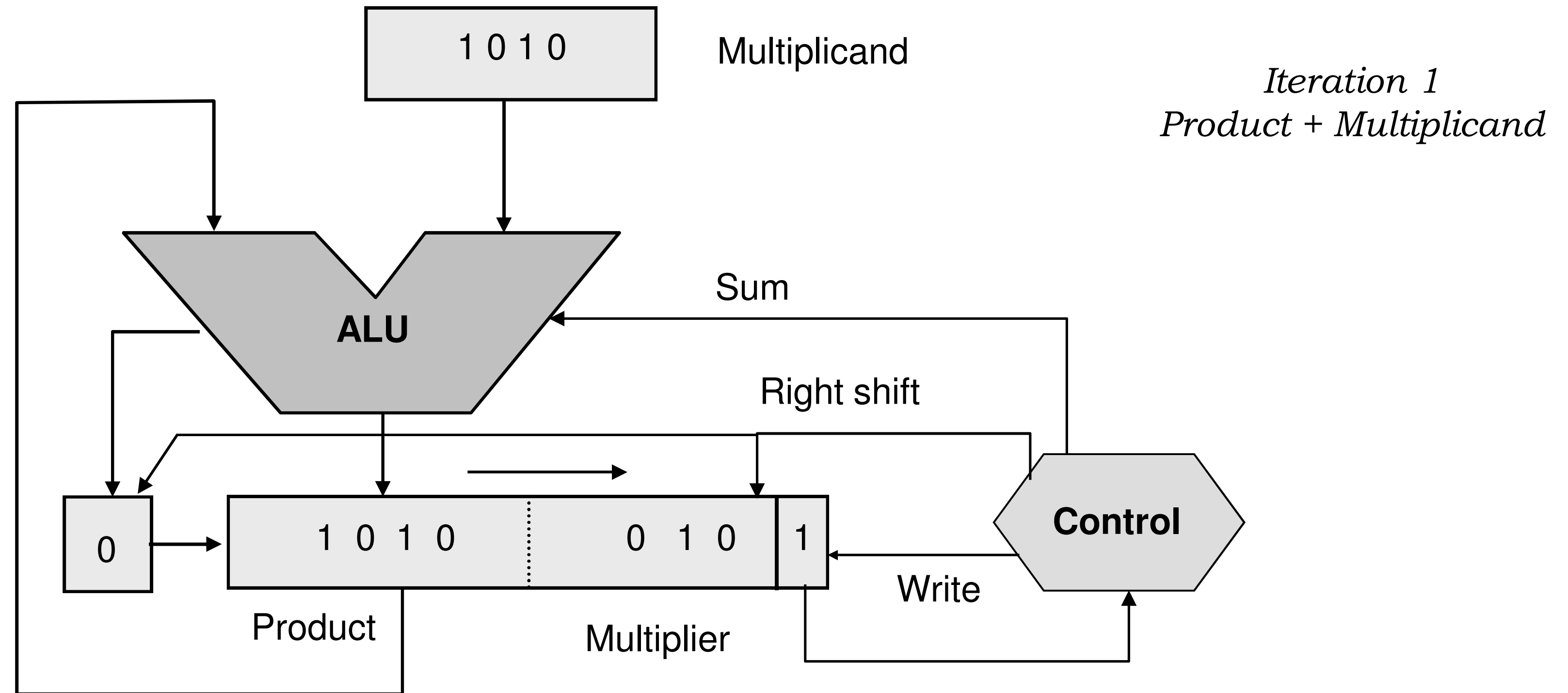
Binary multiplication without sign



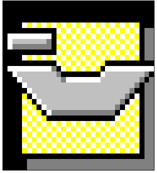


Multiplication

Binary multiplication without sign

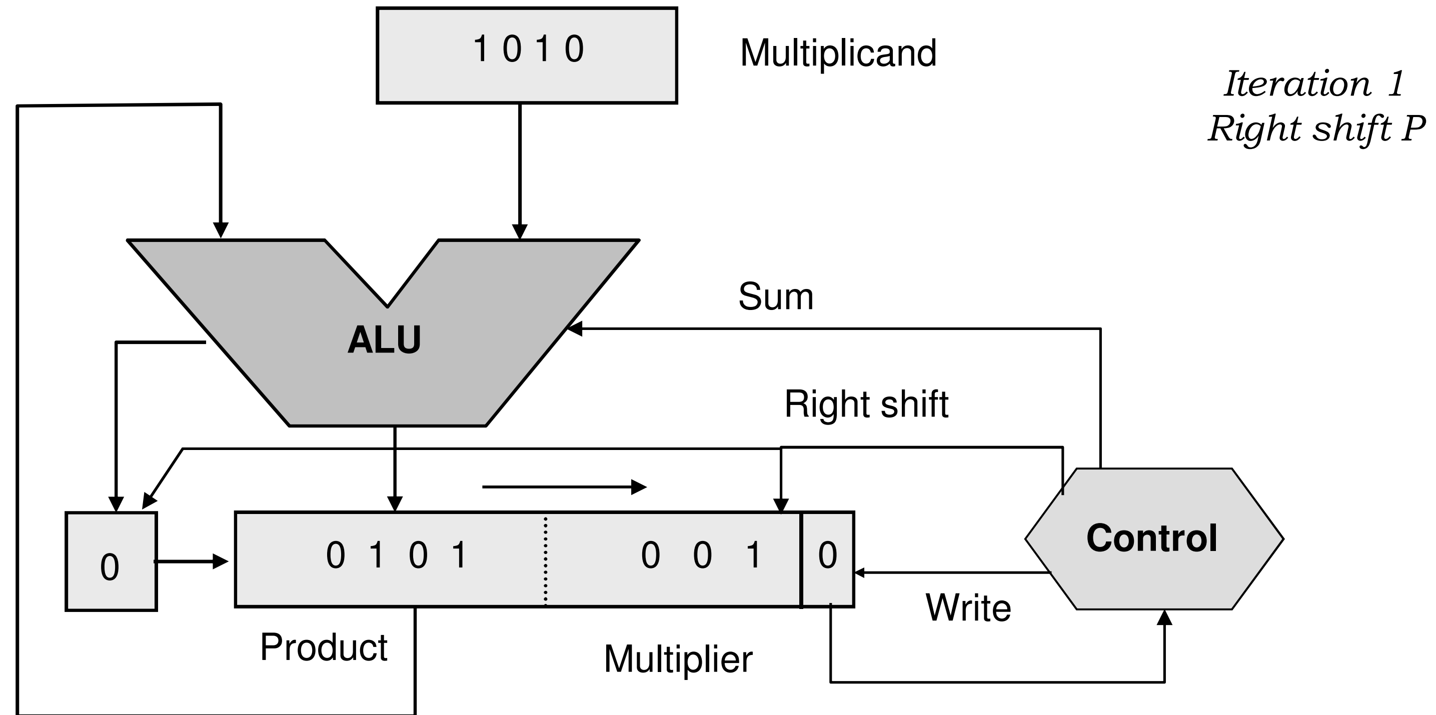


Iteration 1
Product + Multiplicand

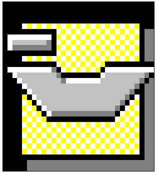


Multiplication

Binary multiplication without sign



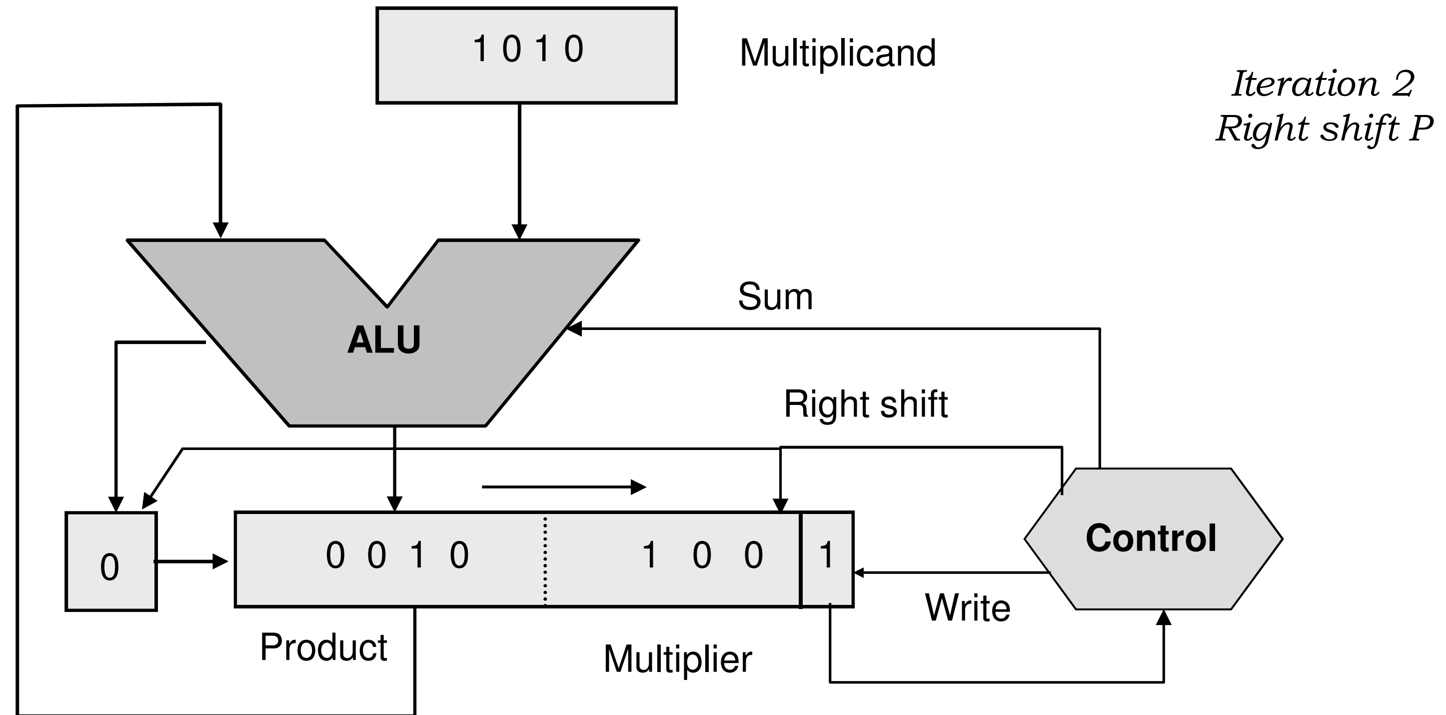
Iteration 1
Right shift P



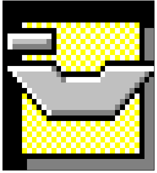
Multiplication

Binary multiplication without sign

Iteration 2
Right shift P



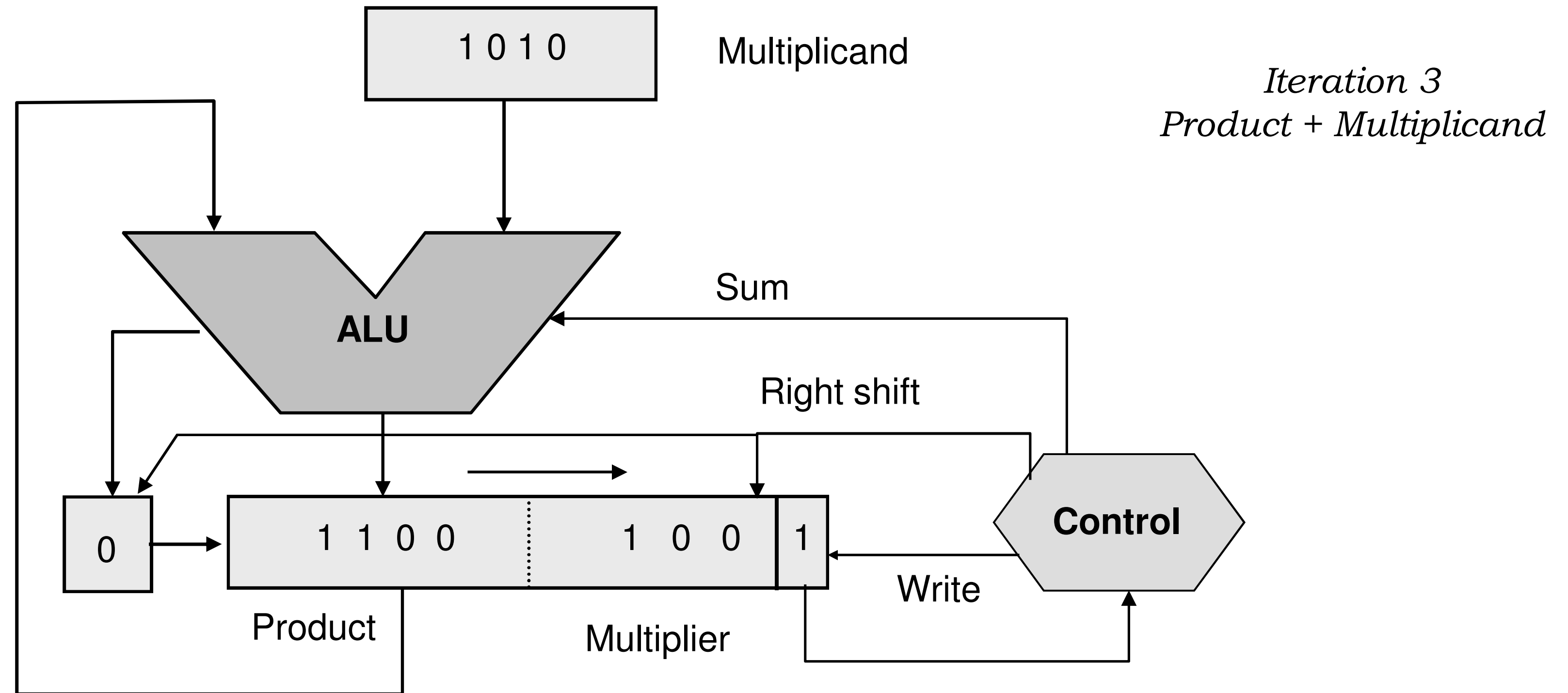
departamento de tecnología informática y computación
UNIVERSIDAD DE ALICANTE



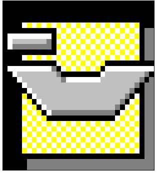
Multiplication

Binary multiplication without sign

Iteration 3
Product + Multiplicand

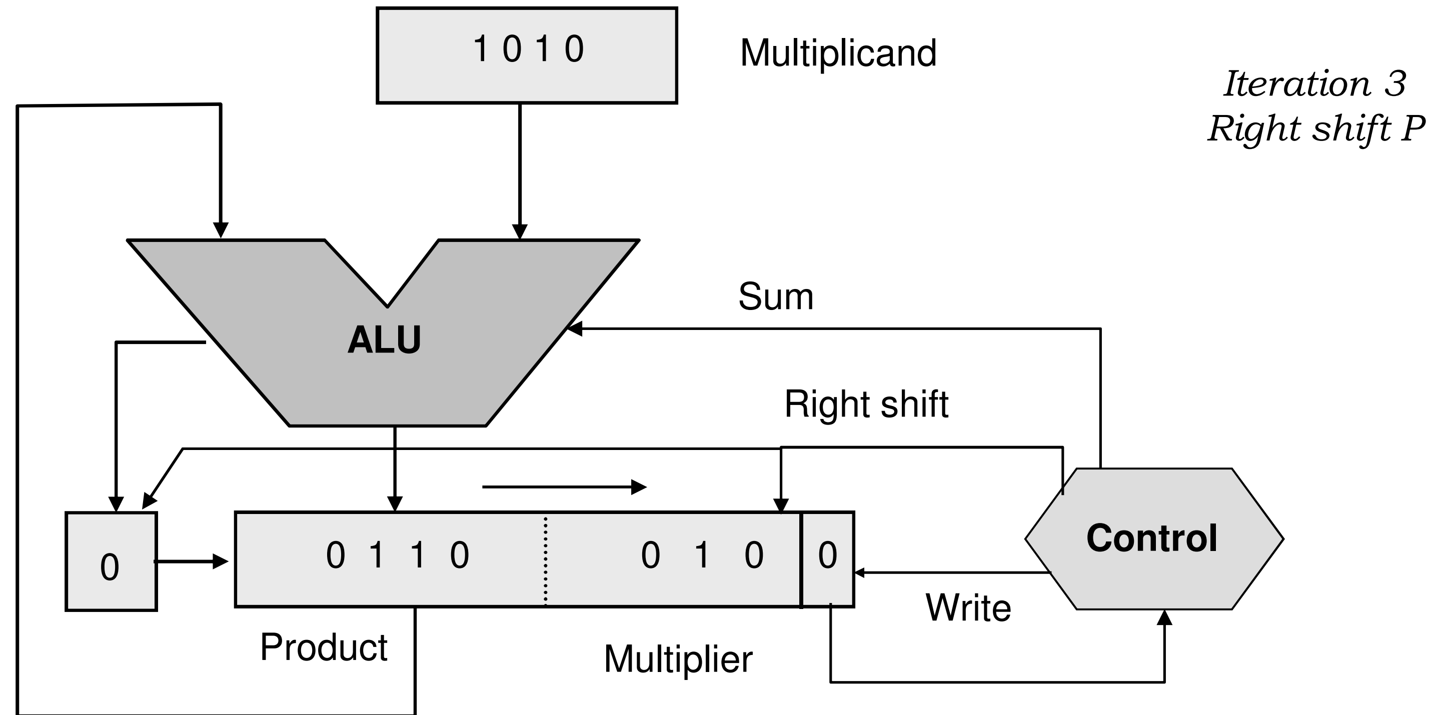


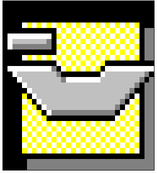
departamento de tecnología informática y computación
UNIVERSIDAD DE ALICANTE



Multiplication

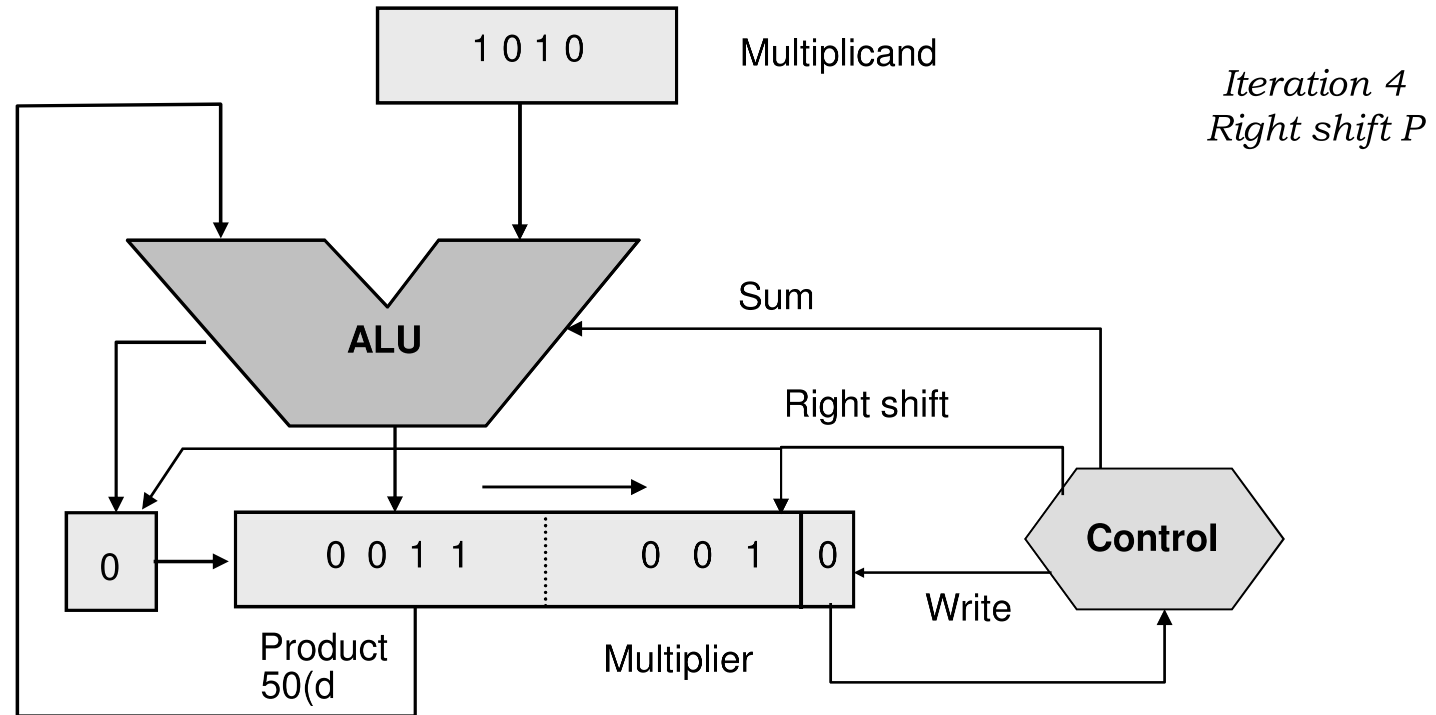
Binary multiplication without sign



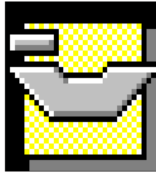


Multiplication

Binary multiplication without sign



Iteration 4
Right shift P

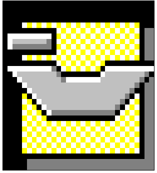


Multiplication

Binary multiplication without sign

Multiplicand = 1010
Multiplier = 0101

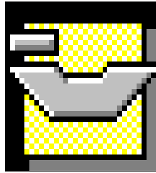
Product	Multiplicand	Action	Iteration
0000 010**1**	1010	Initial values	0
1010 0101	1010	Sum prod. and multiplicand	1
0101 001**0**	1010	Right shift prod. 1 bit	1
0010 100**1**	1010	Right shift prod. 1 bit	2
1100 1001	1010	Sum prod. and multiplicand	3
0110 010**0**	1010	Right shift prod. 1 bit	3
0011 0010	1010	Right shift prod. 1 bit	4



Multiplication

Fast multiplication

departamento de tecnología informática y computación
UNIVERSIDAD DE ALICANTE



Multiplication

Binary multiplication with sign

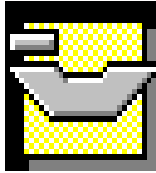
- Assume two's complement numbers
- A = 1010 y B = 0011
- Apply sums and shifts algorithm

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

Wrong version

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

Right version



Multiplication

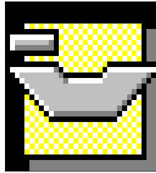
Booth Algorithm

- Assume Multiplicand = 2 and Multiplier = 7 (binaries 0010 x 0111)
- Booth expressed 7 = 8 - 1 and replaced the multiplier by this decomposition:

$$0111 = 1000 - 0001 = +100\text{-}1$$

				0	0	1	0	Multiplicand
			x	+1	0	0	-1	Multiplier according to A. Booth
1	1	1	1	1	1	1	0	Multiplicand subtraction
0	0	0	0	0	0			2 shifts (2 zeros in the multiplier)
0	0	0	1	0				Multiplicand addition
0	0	0	0	1	1	1	0	

Departamento de tecnología informática y computación
UNIVERSIDAD DE ALICANTE



Multiplication

Booth Algorithm

Current Bit	Bit on the left	Replacement
0	0	0 (no transition)
0	1	-1 (transition to negative)
1	0	+1 (transition to positive)
1	1	0 (no transition)

Example: Multiplicand = 11101110 y Multiplier = 01111010

Multiplier according to Booth = +1000-1+1-10

								1	1	1	0	1	1	1	0
							x	+1	0	0	0	-1	+1	-1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	
1	1	1	1	1	1	1	1	1	0	1	1	1	0		
0	0	0	0	0	0	0	0	1	0	0	1	0			
1	1	1	1	0	1	1	1	0	0	0	0				
1	1	1	1	0	1	1	1	0	1	1	0	1	1	0	0

departamento de tecnología informática y computación
UNIVERSIDAD DE ALICANTE

Multiplication

Booth Algorithm

$q_{-1}=0$

Repeat n times

If $q_0 = 1$ and $q_{-1} = 0$ then

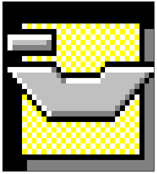
Product$_h$ = product$_h$ - Multiplicand

If $q_0 = 0$ and $q_{-1}=1$ then

Product$_h$ = Product$_h$ + Multiplicand

Arithmetic shift to the right of Product and q_{-1}

End repeat.

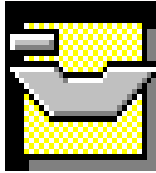


Multiplication

Booth Algorithm

Multiplicand = 1010
Multiplier = 1110

Multiplicand	Product	q_{-1}	Action	Itertion
1010	0000 111**0**	**0**	Initial values	0
1010	0000 1110	0	00 → No operation	1
1010	0000 011**1**	**0**	Right shift	1
1010	0110 0111	0	10 → Subtraction	2
1010	0011 001**1**	**1**	Right shift	2
1010	0011 0011	1	11 → No operation	3
1010	0001 100**1**	**1**	Right shift	3
1010	0001 1001	1	11 → No operation	4
1010	**0000 1100**	1	Right shift	4



Division

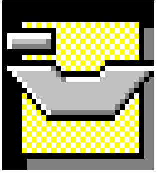
Division

- Division can be expressed as:

  Dividend = Quotient x Divisor + Remainder

- The remainder is smaller than the divisor. Double of space should be reserved for the dividend.
- Assume positive operands.

```
Dividend →   10010011 | 1011     ←Divisor
                      |--------
             10010      01101    ←Quotient
              1011
             ---------
             001110
               1011
             ---------
               001111
                 1011
             ---------
                 0100  ←Remainder
```

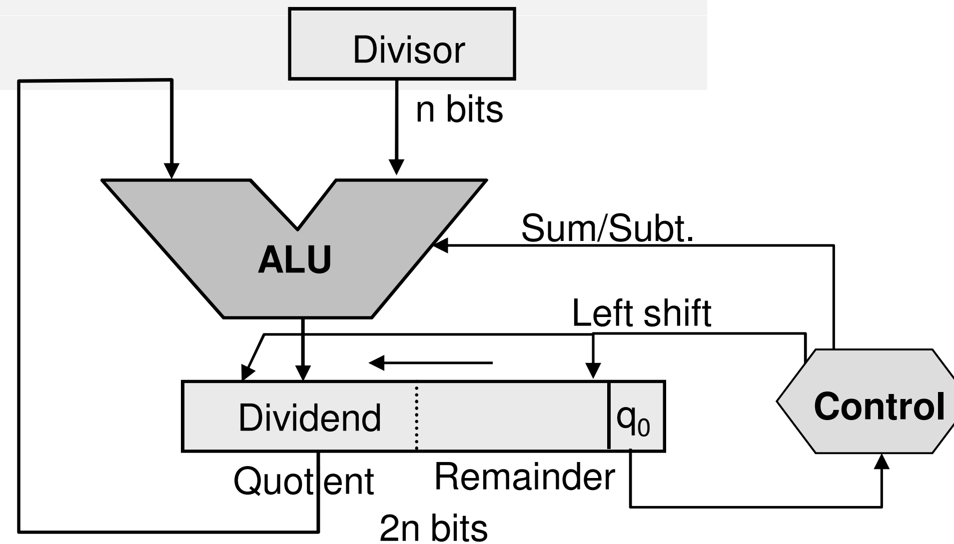


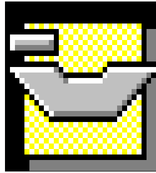
Division

Algorithm with restoration

Repeat n times
- Shift dividend to the left
- $Dividend_h = Dividend_h - Divisor$
- If $Dividend_h < 0$ then (does not fit)
  - $q_0=0$
  - $Dividend_h = Dividend_h + Divisor$ (restore)
- else
  - $q_0=1$
- end if

end repeat





Division

Algorithm with restoration

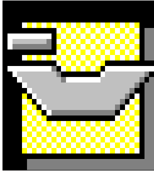
Dividend	Divisor	Action	Iteration
0101 0011	0110	Initial values	0
1010 011_	0110	Shift 1 bit to the left	1
0100 011_	0110	Restar	1
0100 0111	0110	$\text{Dividend}_h > 0 \Rightarrow q_0 = 1$	1
1000 111_	0110	Shift 1 bit to the left	2
0010 111_	0110	Dividend_h - Divisor (Subtract)	2
0010 1111	0110	$\text{Dividend}_h > 0 \Rightarrow q_0 = 1$	2
0101 111_	0110	Shift 1 bit to the left	3
1111 111_	0110	Dividend_h - Divisor (Subtract)	3
1111 1110	0110	$\text{Dividend}_h <= 0 \Rightarrow q_0 = 0$	3
0101 1110	0110	Dividend_h + Divisor (Restore)	3
1011 110_	0110	Shift 1 bit to the left	4
0101 110_	0110	Dividend_h - Divisor (Subtract)	4
0101 1101	0110	$\text{Dividend}_h > 0 \Rightarrow q_0 = 1$	4

↑ ↑

Rem. Quot.

departamento de tecnología informática y computación

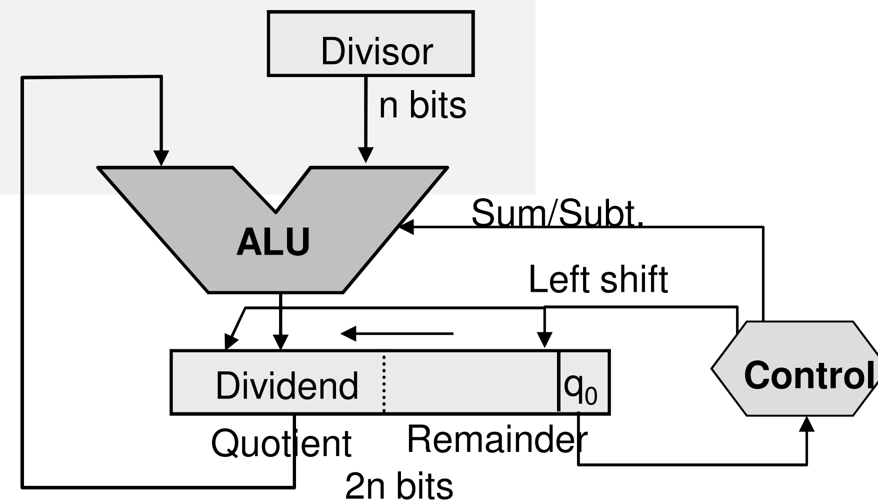
UNIVERSIDAD DE ALICANTE

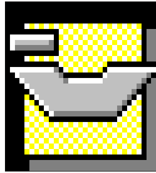


Division

Algorithm without restoration

Dividend$_h$ = Dividend$_h$ - Divisor
Repeat n times
    If Dividend$_h$ < 0 then
    Shift the Dividend to the left
    Dividend$_h$ = Dividend$_h$ + Divisor
    Else
        Shift the Dividend to the left
        Dividend$_h$ = Dividend$_h$ - Divisor
    End if
    If Dividend$_h$ < 0 then
        q_0=0
    Else
        q_0=1
    End if
End repeat





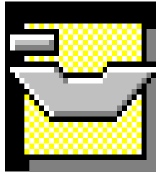
Algorithm without restoration

Dividend	Divisor	Action	Iteration
0000 0111	0010	Initial values	0
1110 0111	0010	Dividend$_h$ - Divisor	0
1100 111_	0010	Dividend$_h$ < 0 $\Rightarrow$ Shift left	1
1110 111_	0010	Dividend$_h$ + Divisor	1
1110 111**0**	0010	Dividend$_h$ < 0 $\Rightarrow$ q_0 = 0	1
1101 11**0**_	0010	Dividend$_h$ < 0 $\Rightarrow$ Shift left	2
1111 11**0**_	0010	Dividend$_h$ + Divisor	2
1111 1**100**	0010	Dividend$_h$ < 0 $\Rightarrow$ q_0 = 0	2
1111 **100**_	0010	Dividend$_h$ < 0 $\Rightarrow$ Shift left	3
0001 **100**_	0010	Dividend$_h$ + Divisor	3
0001 **1001**	0010	Dividend$_h$ >= 0 q_0 = 1	3
0011 **001**_	0010	Dividend$_h$ > 0 $\Rightarrow$ Shift left	4
0001 **001**_	0010	Dividend$_h$ - Divisor	4
0001 **0011**	0010	Dividend$_h$ > 0 $\Rightarrow$ q_0 = 1	4

↑ ↑

Remain. Quotient

departamento de tecnología informática y computación
UNIVERSIDAD DE ALICANTE



Conclusions

Conclusions

- Adders
  - Temporal problems with Carry Propagated Adder, specially if n high.
  - Carry Look-ahead Adders improve response time of the adders.
- Multiplication
  - Problems with the multiplication of signed numbers.
  - Booth algorithm allows to multiply two's complement numbers and sometimes it reduces the number of operations if there is 1's or 0's chains in the multiplier.
- Division
  - Algorithm with restoration for positive numbers. For negative numbers, the sign must be preprocessed. The result's sign will depend on that.

departamento de tecnología informática y computación
UNIVERSIDAD DE ALICANTE