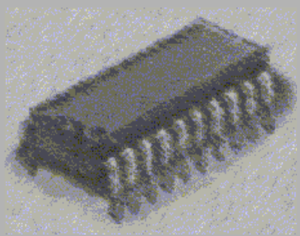
# Lesson 1. The datapath

*Computers Structure and Organization*

Graduate in Computer Sciences /
Graduate in Computers Engnieering







## Contents

- ALU structure and implementation
- ALUs circuits and algorithms
  - Logic and shifts operators
  - Signed operations
  - Fixed point add
  - Floating point add
  - Guard digits
  - Rounding methods
  - Integer multiply and divide operations
  - Floating point multiply and divide operations
- ALU and von Neumann architecure: datapath and functional units
- MC68000 execution unit
- Bibliography

# ALU STRUCTURE AND IMPLEMENTATION (I)

- **ARITHMETIC-LOGIC UNIT (ALU)**: is a set of avaiable operators in the computer
- Made of:
  - Arithmetic, logic and shift operators
  - Registers for storing temporal data
  - Flag register
  - Program counter register
  - Interrupt addresses register
- ALU types:
  - Fixed Point
  - Floating Point





# Estructura e implementación de la ALU (y II)

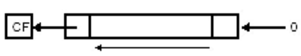
**Operators classification:**

| | |
|---|---|
| Type of operato | General<br>Specialized |
| Build | Combinacional<br>Secuetcial |
| Number of operands | One<br>Two or more |
| Paralelism | Serial or digit operator<br>Paralel or array operator |
| Operation | Shift<br>Lógic<br>Aritmethic |
| Technology | MOS<br>Bipolar |

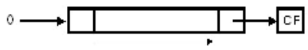
# ALU circuits and algorithm (I) Shifts operations(I)

- **Logic shifts**:
  - Right / Left zero introduction. It doesn't depend on the representation system
  - Last bit of the shift operation is copied on the carry flag

Left shift




Right shift



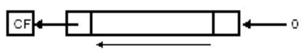




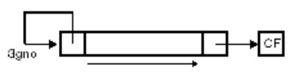

# ALU circuits and algorithm (II) Shifts operations (II)

- **Arithmethic shifts**:
  - These operations are equivalent at multiply or divide by powers of two.
  - Representation system must be taken into account
  - Last bit of the shift operation is copied to the carry flag

Complement 2
left arithmetic shift



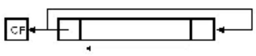

Complement 2
Right arithmetic shift



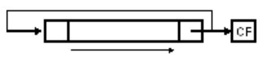
# ALU circuits and algorithm (III) Shifts operations(III)

- **Rotation operation / Rotation throught carry flag operation**:
  - Bits of the end / beginning come into the beginning / end of the register in a circular way.
  - If the rotation is throught the crry flag, this bit is also taken into account to perform the operation
  - Last bit of the rotation operation is copied in the carry flag

Left rotation




Right rotation



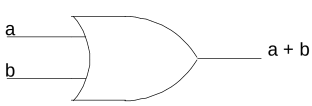





# ALU circuits and algorithms (V) Logic operations (I)

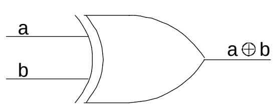
- Logic operations are performed on each individual bit of the operands

OR




| a | b | a OR b |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

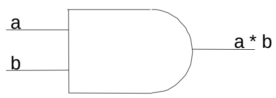
XOR




| a | b | a XOR b |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

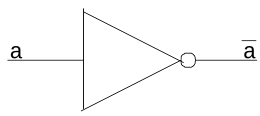
# ALU circuits and algorithms (VI) Logic operations (and II)

## AND




| a | b | a AND b |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

## NOT




| a | $\overline{a}$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

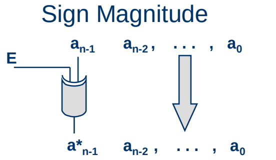




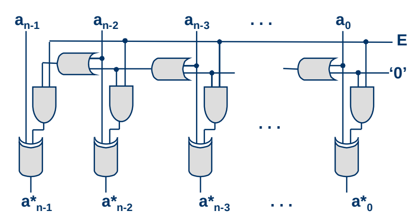
# ALU circuits and algorithms (VII) Atirhmetic operations (I)

- **Signed operation. Change**

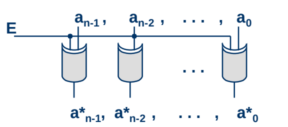
## Sign Magnitude




## Complement 2




## Complement 1

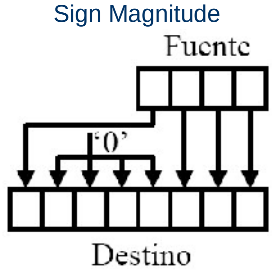


# ALU circuits and algorithms (VIII) Atirhmetic operations (II)

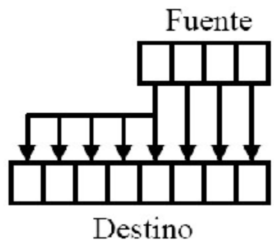
- **Signed operatinon. Sign extention**

To fill empty bits when data pass from less bits to more bits elements.

Sign Magnitude




Complement 1 and 2



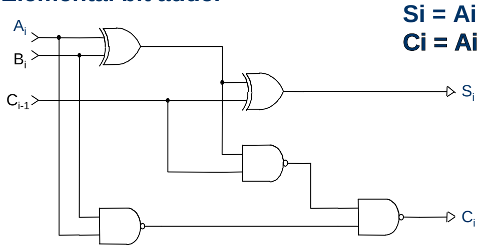





# ALU circuits and algorithms (IX) Atirhmetic operations (III)

- **ADD:** the add operation is the most important operation. It's used to:
  - To calculate next instruction addresses
  - To calculate operands addresses
  - It's employed by multiply and divide operations

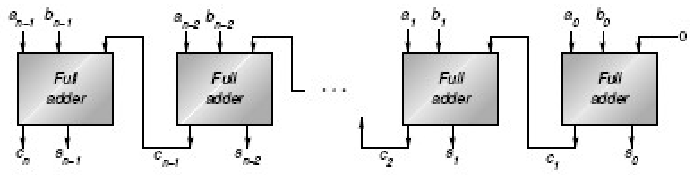
**Elemental bit adder**




**Si = Ai ⊕ Bi ⊕ Ci-1**
**Ci = Ai Bi + Bi Ci-1 + Ai Ci-1**

## ALU circuits and algorithms (X)
## Atirhmetic operations (IV)

- **Carry propagation adder of *n* bits by using 1 bit *full adders***



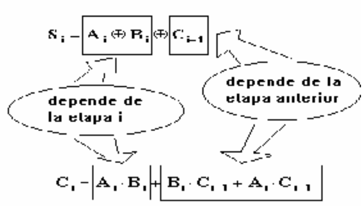

- **Drawbacks:**
  - Very slow adder. Carry must be propagate from the firs adder to the final one to get the result of the operation.
  - Delay of *2·n·r* (maximun level gates to pass throught)
  - *N is the number of adders*
  - R is the delay of each gate

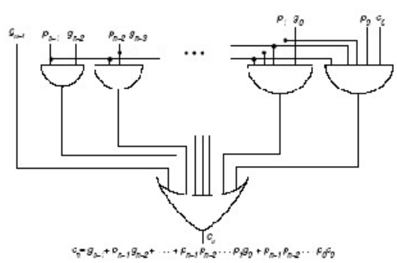




## ALU circuits and algorithms (XI)
## Atirhmetic operations (V)

- **Integer add aceleration:**
  - Additional circuits are added to anticipate to end carry calculus



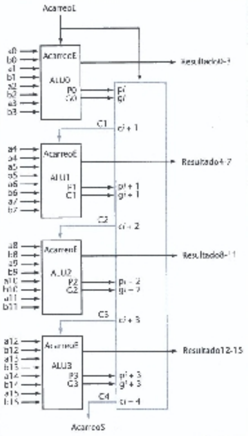



Generation function ($g_i = a_i \cdot b_i$)
Propagation function ($p_i = a_i \oplus b_i$)
Output carry ($c_{i+1} = g_i + p_i \cdot c_{i-1}$)

- **Problem: variable *fan-in***

# ALU circuits and algorithms (XII) Atirhmetic operations (VI)

- **Integer add acceleration:**
  - To homogenise *fan-in* by using 4 bits full adders such as basic component
  - Block Generation and Propagation functions must be defined
  - E.g. 16 bits Look-ahead adder
- **Block level functions**
  - $P_0 = p_3 \cdot p_2 \cdot p_1 \cdot p_0$
  - $G_0 = g_3+(g_2 p_3)+(g_1 p_3 p_2)+(g_0 p_3 p_2 p_1)$
  - $C_1 = G_0 + P_0 \cdot c_e$
  - $C_2 = G_1 + (G_0 \cdot P_1) + (c_e \cdot P_1 \cdot P_0)$

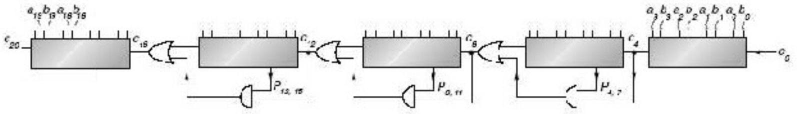






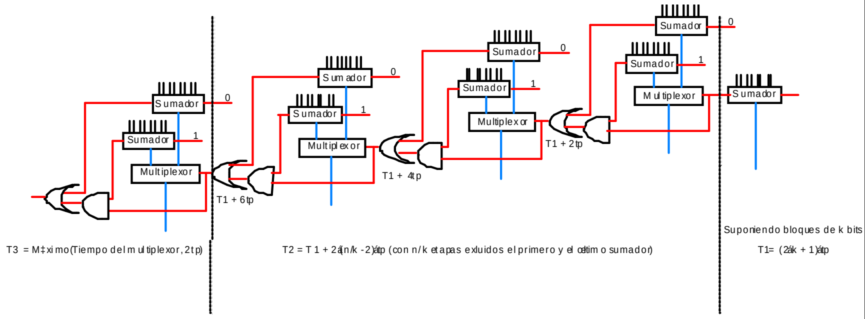
# ALU circuits and algorithms (XIII) Atirhmetic operations (VII)

- **Integer add acceleration. Carry Skip Adder (CSK)**
  - It's betwenn a full adder and a look ahead one.
  - Only calculate Pi (it's easyer).
- **K bits Carry Skip Adder:**
  - First full adder: *2k +1 (plus one level)*
  - Number of gates: *2(n/k-2)*
  - Last full adder: *2k*
  - Total: levels number= *4k + 2n/k -3*
  - E.g.*n = 20, K = 4* → num. Niveles = 23



# ALU circuits and algorithms (XIV) Atirhmetic operations (VIII)

- **Integer add acceleration. Carry select**
  - It's needed to duplicate hardware items. Add with 0 and 1 carry-in is paralel performed. When the correct carry is known, right result is selected via multipexors.








# ALU circuits and algorithms (XV) Atirhmetic operations (IX)

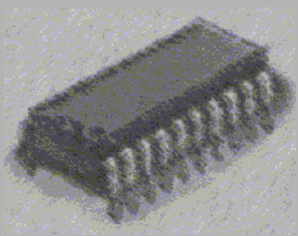
- **Different Adders time and space required**

| ADDER | TIME | SPACE |
|---|---|---|
| Full / Propagation Adder | $O(n)$ | $O(n)$ |
| Carry Look Ahead Adder | $O(\log n)$ | $O(n \cdot \log n)$ |
| Carry Skip Adder | $O(\sqrt{n})$ | $O(n)$ |
| Carry Select Adder | $O(\sqrt{n})$ | $O(n)$ |

# Lesson 1. The datapath

***Computers Structure and Organization***

Graduate in Computer Sciences /
Graduate in Computers Engnieering





Achademic course 2011-2012

Automatic Department

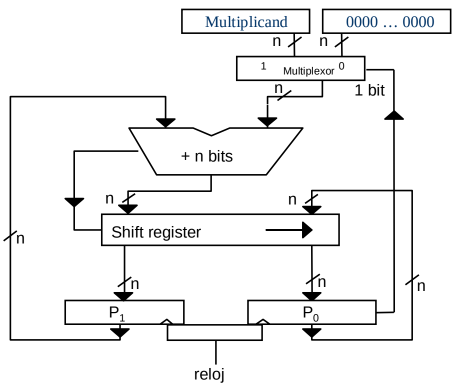


## Contents

- ALU structure and implementation
- ALUs circuits and algorithms
  - Logic and shifts operators
  - Signed operations
  - Fixed point add
  - Floating point add
  - Guard digits
  - Rounding methods
  - Integer multiply and divide operations
  - Floating point multiply and divide operations
- ALU and von Neumann architecure: datapath and functional units
- MC68000 execution unit
- Bibliography

# ALU circuits and algorithms (XVI) Atirhmetic operations (X)

- **Multiplication. Add-Shift algorithm (A x B)**
- Initiate $P_0$ = B
- Only unsigned numbers

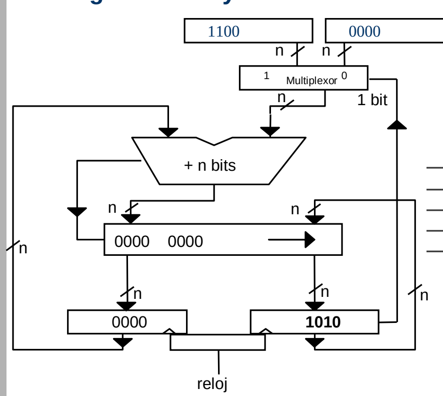







# ALU circuits and algorithms (XVII) Atirhmetic operations (XI)

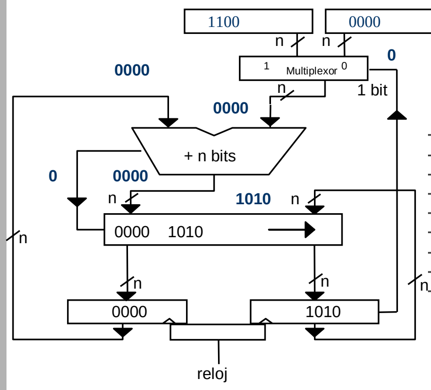
- **E..g. A = 1100 y B = 1010**




| SHIFT Register | P: $P_1$ | P: $P_0$ | Operation |
|---|---|---|---|
| 0000 0000 | 0000 | 1010 | Initial State |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# ALU circuits and algorithms (XVIII) Atirhmetic operations (XII)

- **E.g. A = 1100 y B = 1010**



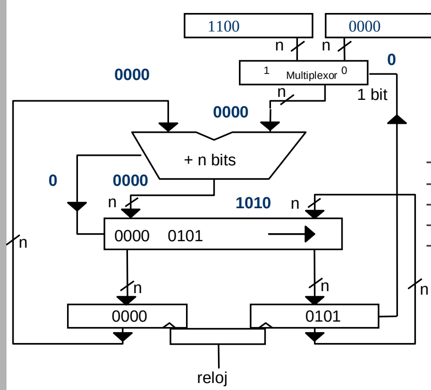

| SHIFT Register | P: $P_1$ | P: $P_0$ | Operation |
|---|---|---|---|
| 0000 0000 | 0000 | 1010 | Inital state |
| **0000 1010** | **0000** | **1010** | **ADD** |





# ALU circuits and algorithms (XIX) Atirhmetic operations (XIII)

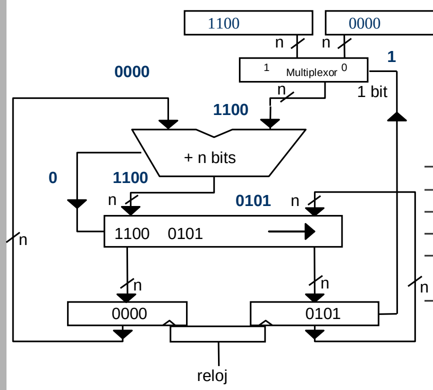
- **E.g. A = 1100 y B = 1010**




| SHIFT Register | P: $P_1$ | P: $P_0$ | Operation |
|---|---|---|---|
| 0000 0000 | 0000 | 1010 | Initial State |
| 0000 1010 | 0000 | 1010 | ADD |
| **0000 0101** | **0000** | **0101** | **SHIFT** |

# ALU circuits and algorithms (XX) Atirhmetic operations (XIV)

- **E.g. A = 1100 y B = 1010**



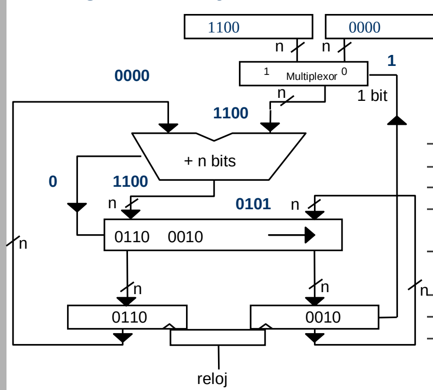

| SHIFT Register | P $P_1$ | $P_0$ | Operation |
|---|---|---|---|
| 0000 0000 | 0000 | 1010 | Initla State |
| 0000 1010 | 0000 | 1010 | ADD |
| 0000 0101 | 0000 | 0101 | SHIFT |
| **1100 0101** | **0000** | **0101** | **ADD** |





# ALU circuits and algorithms (XXI) Atirhmetic operations (XV)

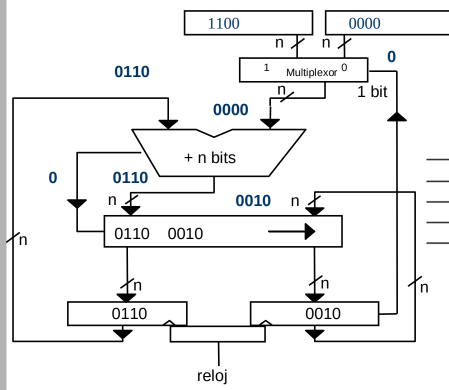
- **E.g. A = 1100 y B = 1010**




| SHIFT Register | P $P_1$ | $P_0$ | Operation |
|---|---|---|---|
| 0000 0000 | 0000 | 1010 | Initial State |
| 0000 1010 | 0000 | 1010 | ADD |
| 0000 0101 | 0000 | 0101 | SHIFT |
| 1100 0101 | 0000 | 0101 | ADD |
| **0110 0010** | **0110** | **0010** | **SHIFT** |

# ALU circuits and algorithms (XXII) Atirhmetic operations (XVI)

- **E.g. A = 1100 y B = 1010**



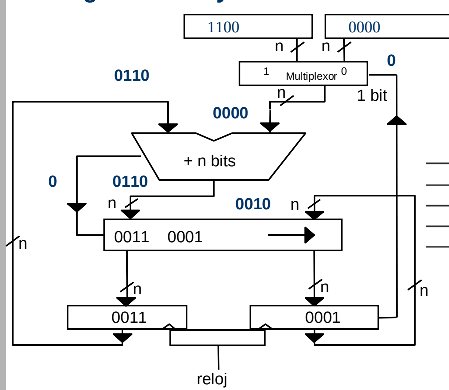

| SHIFT Register | P: $P_1$ | P: $P_0$ | Operation |
|---|---|---|---|
| 0000 0000 | 0000 | 1010 | Initial Stae |
| 0000 1010 | 0000 | 1010 | ADD |
| 0000 0101 | 0000 | 0101 | SHIFT |
| 1100 0101 | 0000 | 0101 | ADD |
| 0110 0010 | 0110 | 0010 | SHIFT |
| **0110 0010** | **0110** | **0010** | **ADD** |





# ALU circuits and algorithms (XXIII) Atirhmetic operations (XVII)

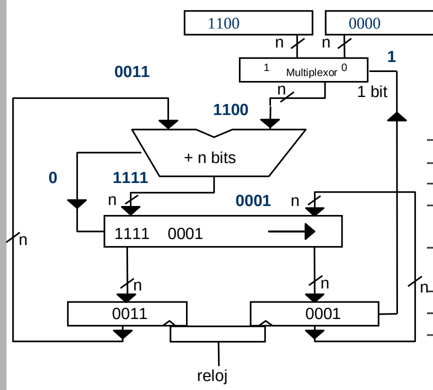
- **E.g. A = 1100 y B = 1010**




| SHIFT Register | P: $P_1$ | P: $P_0$ | Operation |
|---|---|---|---|
| 0000 0000 | 0000 | 1010 | Initial State |
| 0000 1010 | 0000 | 1010 | ADD |
| 0000 0101 | 0000 | 0101 | SHIFT |
| 1100 0101 | 0000 | 0101 | ADD |
| 0110 0010 | 0110 | 0010 | SHIFT |
| 0110 0010 | 0110 | 0010 | ADD |
| **0011 0001** | **0011** | **0001** | **SHIFT** |

# ALU circuits and algorithms (XXIV)
# Atirhmetic operations (XVIII)

- **E.g. A = 1100 y B = 1010**



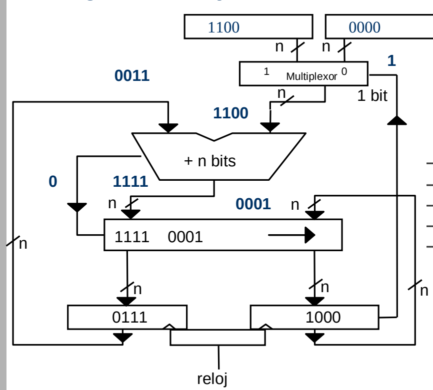

| SHIFT Register | P: $P_1$ | P: $P_0$ | Operation |
|---|---|---|---|
| 0000 0000 | 0000 | 1010 | Initial State |
| 0000 1010 | 0000 | 1010 | ADD |
| 0000 0101 | 0000 | 0101 | SHIFT |
| 1100 0101 | 0000 | 0101 | ADD |
| 0110 0010 | 0110 | 0010 | SHIFT |
| 0110 0010 | 0110 | 0010 | ADD |
| 0011 0001 | 0011 | 0001 | SHIFT |
| **1111 0001** | **0011** | **0001** | **ADD** |





# ALU circuits and algorithms (XXV)
# Atirhmetic operations (XIX)

- **E.g. A = 1100 y B = 1010**




| SHIFT Register | P: $P_1$ | P: $P_0$ | Operation |
|---|---|---|---|
| 0000 0000 | 0000 | 1010 | Initial State |
| 0000 1010 | 0000 | 1010 | ADD |
| 0000 0101 | 0000 | 0101 | SHIFT |
| 1100 0101 | 0000 | 0101 | ADD |
| 0110 0010 | 0110 | 0010 | SHIFT |
| 0110 0010 | 0110 | 0010 | ADD |
| 0011 0001 | 0011 | 0001 | SHIFT |
| 1111 0001 | 0011 | 0001 | ADD |
| **0111 1000** | **0111** | **1000** | **SHIFT** |

# ALU circuits and algorithms (XXVI) Atirhmetic operations (XX)

- **Complement 1 and complement 2 adjusts for the algorithm**
  - **Complement 2**: if B is a negative number, Subtract A when last 1 bit arrives to multiplexor.
  - **Complement 1**:if B is a negative number, Subtract A when last 1 bit arrives to multiplexor. Moreover, P1 must be initiated to A instead of 0.





# ALU circuits and algorithms (XXVII) Atirhmetic operations (XXI)

- **Multiplication. Booth Algorithm (C2)**
  - For avoiding zero additions. These adds operations only consume clock cycles because operand is only shifted
  - Search 1's and 0's strings to code the multiplier and powering by the power of them

- **A = 1011, B = 0110**
- **A x B = +Ax2$^3$ –Ax2$^1$**

0001010

1011**000 +**

**1100010**

| $X_i$ | $X_{i-1}$ | Meaning | Action to perform | $Y_i$ |
|---|---|---|---|---|
| 0 | 0 | Zeros string | Shift | 0 |
| 1 | 1 | Ones string | Shift | 0 |
| 1 | 0 | Starting 1's string | Subtract and Shift | -1 |
| 0 | 1 | Ending 0's string | Add and Shift | 1 |

# ALU circuits and algorithms (XXVIII) Atirhmetic operations (XXII)

- **Division. Unsigned "With Remainder Restoration" Division Algorithm**

- **Starting point**: Take as many bits of the dividend as the divisor has.
- Dividend and divisor must be unsigned
- Add complement 2 of the divisor to the dividend
  - If positive:
    - Take a new bit of the dividend
    - Write an 1 in the quotient
  - If negative:
    - Add  divisor to dividend again or restore previous value from a register
    - Take a new bit of the dividend
    - Write a 0 in the quotient
- Repeat the process until we have no more dividend bits to take
- If final quotient is negative then restore it





# ALU circuits and algorithms (XXIX) Atirhmetic operations (XXIII)

- **Division. Unsigned "With Remainder" Restoration Division Algorithm**
  - A = 0100011, B = 0011, Quotient = 1011, Remainder = 0010

```
 0 1 0 0 0 1 1   | 0011
+1 1 0 1           1 0 1 1
1 0 0 0 1 0
 + 1 1 0 1
   1 1 1 1
   0 0 1 0 1   ← Restauración
   + 1 1 0 1
   1 0 0 1 0 1
     +1 1 0 1
     1 0 0 1 0
```

# ALU circuits and algorithms (XXX)
# Atirhmetic operations (XXIV)

**Division. Unsigned "Without Remainder Restoration" Division Algorithm**

- **Starting point**: Take as many bits of the dividend as the divisor has
- Dividend and divisor must be unsigned
- Add complement 2 of the divisor to dividend
  - If positive:
    - Take a new bit of the dividend
    - Write an 1 in the quotient
  - If negative:
    - Add the divisor to the last result
    - Take a new bit of the dividend
    - Write a 0 in the quotient
- Repeat until we have no more dividend bits
- If the final quotient is negative then restore it.





# ALU circuits and algorithms (XXXI)
# Atirhmetic operations (XXV)

**Division. Unsigned "Without Remainder Restoration" Division Algorithm**

- A = 0100011, B = 0011, Quotient= 1011, Remainder= 0010

```
 0 1 0 0 0 1 1  | 0011
+1 1 0 1          1 0 1 1
 0 0 0 1 0
 + 1 1 0 1
    1 1 1 1 1
    + 0 0 1 1
     1 0 0 1 0 1
       + 1 1 0 1
        1 0 0 1 0
```

# ALU circuits and algorithms (XXXII) Atirhmetic operations (and XXVI)

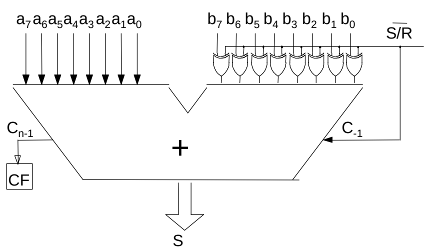
- **Floating point multiplication and division**
  - Use fixed point algorithms
  - Algorithms are applied to mantissas
  - Exponent are added (multiplication) or subtract (division)
  - Result of the operation must be normalized and rounded
  - It's possible to employ guard digits to improve the accurate of the result





# ALU circuits and algorithms (XXXIII) Circuits (I)

- **Binary Subtract-Adder**



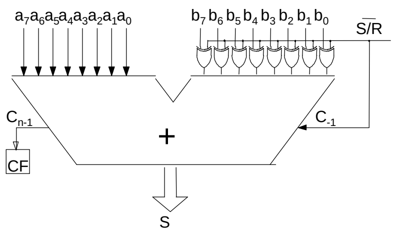

$$Overflow = c_{n-1} \oplus \bar{S}/R$$

- **Sign-Magnitude Subtract-Adder**

Uses a binary subtract-adder and a signed circuit is needed to determine operation

# ALU circuits and algorithms (XXXIV) Circuits (II)

- **Complement 2 Subtract-Adder**



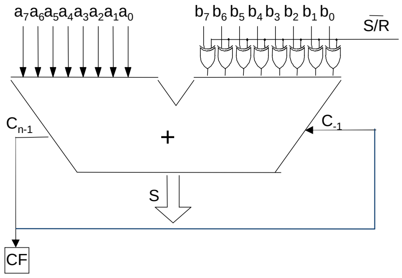

$$Overflow = c_{n-1} \oplus c_{n-2}$$





# ALU circuits and algorithms (XXXV) Circuits (III)

- **Complement 1 Subtract-Adder**




$$Overflow = c_{n-1} \oplus c_{n-2}$$

# ALU circuits and algorithms (XXXVI) Circuits (IV)

- **Excess Subtract-Adder**

Result operation adjust is needed:
- If ADD operation → subtract M
- If SUBTRACT operation → add M

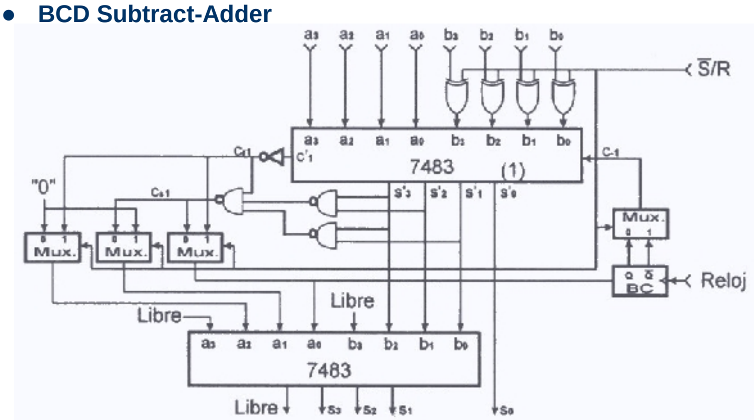
- IF **M = $2^{n-1}$** is easy:
  1º Most Significant Bit must be inverted for each operand
  2º ADD or SUBTRACT as Complement 2 represented operands
  3º Invert Most Significant Bit of the result





# ALU circuits and algorithms (XXXVII) Circuits (and V)

- **BCD Subtract-Adder**



## ALU circuits and algorithms (XXXIX)
## FLOATING POINT ADDITION-SUBTRACTION (I)

- **How to Add-Subtract Floating Point Number Algorithm:**
  1. Split mantissas and exponents
  2. Compare exponents and:
     - Keep greatest exponent. It will be the result exponent except if result must be normalized
     - Subtract both exponents and take the absolute value. It will be the number of lowest mantissa right shift required
  3. mantissas align is required by right shifting the lower one
  4. Add or subtract aligned mantissas
  5. Check if result is normalized. If not, normalize it.
  6. Round the result if required





## ALU circuits and algorithms (XXXX)
## FLOATING POINT ADDITION-SUBTRACTION (and II)

- **Floating point addition example:**
  - Lets A and B floating point represented numbers. Excess $2^{n-1}$ exponent (8 bits) mantissa (8 bits to), complement 2, normalized and without implicit bit expressed
  - EA = 1000 0100 MA = 0100 0100
  - EB = 1000 0011 MB = 0111 0000
  - Exponents comparation EA(4) > EB(3). Result exponent EA
  - Shift MB EA-EB times, 4 – 3 = 1
  - 0100 0100
    0011 1000 +
    0111 1100
  - ER = 1000 0100 MR = 0111 1100

## ALU circuits and algorithms (XXXXI) Guard Digits

- Guard Digits are only added and used inside Arithmetic-Logic Unit
- Guard Digits are used for helping in rounding and normalizing numbers and increasing the accurate of operations.
- Usually, two guard digits and one sticky bit are added to the end of the operands.

$$b_8b_7b_6b_5b_4b_3b_2b_1b_0\ b_{g1}\ b_{g2}\ b_r$$

- Bits $b_8$ to $b_0$ are 8 bits data bits
- $b_{g1}$ first guard digit used for normalization purposes
- $b_{g2}$ second guar digit employed in rounding techniques
- $b_r$ the sticky bit. Keep an one if during shift phases an one pass through it in order of avoid accurate losing in subtract operations





## ALU circuits and algorithms (XXXXII) Rounding Methods

- Guard digits and sticky bit must be removed when passing from ALU to computer register or memory positions. These information must be taken into account before removing.
- Most frequent rounding techniques are the following:
  - **Truncation**: guard digits and sticky bits are removed.
  - **LSB forced to 1**: guard digits and sticky bit are removed and the least significant bit of the operand is forced to 1.
  - **Nearest rounding**: It's the more difficult to implement but the one which better results obtain. If digits are above the half of combination then truncate and add one. If digits are bellow the half of combination truncate. If digits are exactly in the half truncate and add the LSB to the LSB
  - 3 guar digits: 000, 001, 010, 011 truncate. 101, 110 y 111 truncate and add 1. Finally 100 add 1 if LSB 1s 1 or truncate if LSB is 0

# ALU and Von Neumann's Architecure (I) The Datapath

- von Neumann arithmetic unit is called the *data path*
- It's composed of arithmetic-logic units, shifters, registers and buses to communicate them
- Flags register belongs to the data path
- Program counter register (PC) and Interrupt Addresses register also belongs to the data path
- Data path determines the cost of the processor. It requires the half of the number of transistors and the half in the silicon area of the processor
- Data path is the bottleneck of the processor because the slowest circuits determine the time clock cycle
- Keys to design the data path:
  - Choose the number and port numbers for the register bank
  - Choose the type of ALU or ALUs tu be included

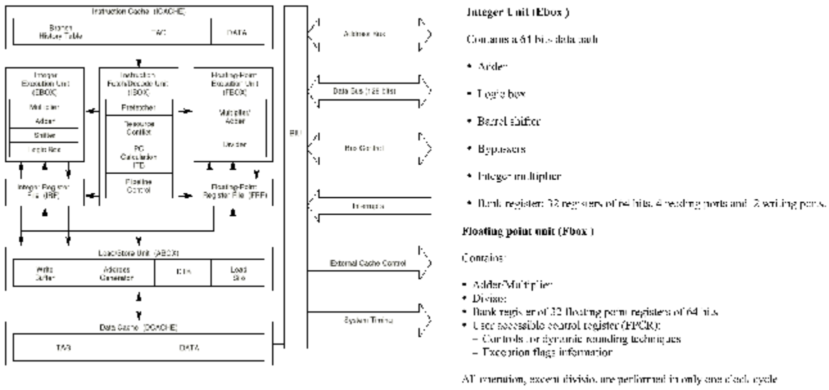




# ALU and Von Neumann's Architecure (II) The Flag Register.

- There are flip-flop related to the result of the arithmetic and logic operation performed in the computer. All of them are collected in one register called the Flags Register
- The aim of the flag register is to show some important characteristics of the operation results.
- Most frequent flags are:
  - Zero, Signed, Overflow and Carry
  - Parity, Auxiliar Carry, Trap, Interrupt Enable, …
- Exception conditions. Some of them are based on the content of the flags register. E.g. overflow, parity memory error,…

# Ejemplo de hardware real (I)
# Alpha 21064. Aceleración de la suma (I)








# Ejemplo de hardware real (II)
# Alpha 21064. Aceleración de la suma (y II)

- Alpha 21064 integer and floating point adder is a 64 bits adder which performs to add per clock cycle. It’s two segmented stages.
- Three methods are combined:
  - 8 bits full adders
  - Carryselct in 2 blocks of 32 bits
  - 32 bits blocks carry look ahead
- Latches are used to keep temporal results between phases

# Bibliography


- Estructura y diseño de computadores
  David A. Patterson y John L. Hennessy. Reverté, 2000
  Capítulo 4

- Arquitectura de computadores. Un enfoque cuantitativo
  John L. Hennessy y David A. Patterson. Mc Graw Hill, 3ª ed, 2002
  Apéndices A: aritmética de computadores

- Organización y arquitectura de computadores
  William Stallings. Prentice Hall. 1996
  Capítulo 8