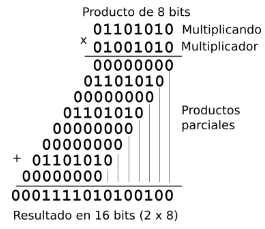
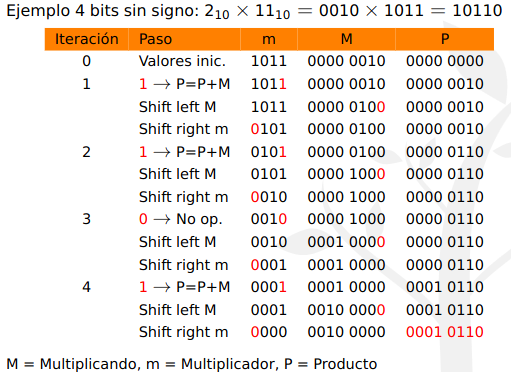
**Aritmética del computador**

**Suma y resta en punto fijo**

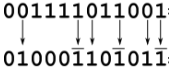
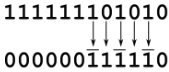
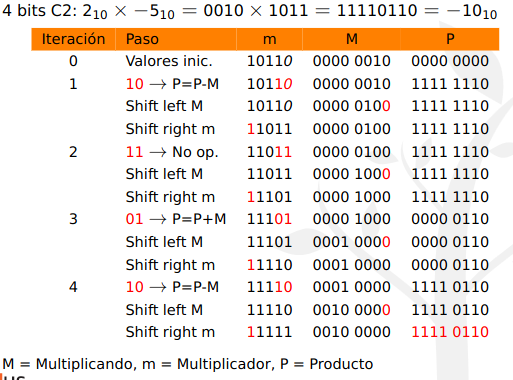
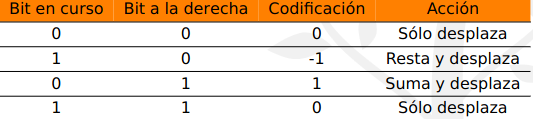
* Los dígitos se suman bit a bit de derecha a izquierda, pasando acarreo
* Para **restar**, el sustraendo se complementa a 2 y luego se suma.
* Para sumar N bits, se utilizan N full adders, conectados formando un sumador de acarreo enlazado.
* Se detecta desbordamiento si el signo del resultado no es correcto
  + En este caso, se puede hacer wrap-around (C, Java), saturar al extremo o convertir a un tipo más grande (python)

**Multiplicación en punto fijo (sin signo)**



* Se adjudican al multiplciando tantos bits como sean necesarios para el resultado, y al multiplicador los bits necesarios para el multiplicador.
* Por cada bit del multiplicador, empezando por el de la derecha, se suma al producto el producto parcial:
  + Si el bit es un 1, se suma el multiplicando al producto (1 \* M)
  + Si el bit es un 0, se suma 0\*M, por lo que el producto no cambia.
* Se desplaza el multiplicador 1 bit a la derecha y el multiplicando a la izquierda. Se repite el proceso hasta que el multiplicador se haya desplazado por completo.
* Para multiplicar sin signo 32 bits, son necesarios:
  + Un registro con desplazamiento de 64 bit para el multiplicando, y uno de 32 bits para el multiplicador
  + Un registro de 64 bits para el resultado
  + Una unidad de control para las condiciones y una ALU para las sumas

**Multiplicación en punto fijo (con signo)**

* Se utiliza el **algoritmo de Booth:**
* Recodificamos el multiplicador para aprovechar secuencias de 1s:
  + Se lee del LSB al MSB, dejando los 0 sin cambiar
  + Al encontrar un 1, se cambia por un -1 (1 negado). Si el LSB inicial es un 1, se cambia por -1 igualmente.
  + Luego, todos los 1s consecutivos se cambian por 0s
  + Al llegar a otro 0, este se cambia por un 1, y se reanuda el proceso.
* Ejemplo:
* En circuito:
  + Se añade un bit 0 a la derecha del LSB
  + Se recorre de LSB a MSB siguiendo el criterio de la tabla.
  + La implementación se realiza con **ALU** y **desplazadores**.
  + En el MIPS existen registros especiales de **64 bits** (HI, LO) para almacenar el resultado. (ver instrucciones mult, multi, mflo, mfhi)

**Suma en punto flotante**

* **Alineamiento:** Se pone el mismo exponente en ambos operandos
* Se **suma/resta** las mantisas, si es resta se complementa un operando
  + Si hay overflow, el resultado es +-inf
* **Normalización:** Se transforma el resultado para dejar un bit en la parte entera
* Si es necesario, se **redondea** al nº de bits de salida
  + Operaciones no asociativas, debido al error cometido al aproximar

**Multiplicación en punto flotante**

* El exponente del resultado es la **suma** de los exponentes de los productos
* Se **multiplican** las mantisas y se **normaliza** el resultado
* Se redondea al número de bits necesarios

**Modos de redondeo**

* **Truncamiento:** Se descartan los bits adicionales
* **Redondeo al más próximo, empate a par**: Se redondea al número más proximo y en empate se redondea al número par
* **Redondeo a más/menos infinito:** siempre al número mayor/menor