```
i-----------------------------------------------©
¦ RBT   ¦   Curso de Assembly   ¦   Aula Nº 02 ¦
È-----------------------------------------------¥
```

Por: Frederico Pissarra

```
i-------------©
¦ ASSEMBLY II ¦
È-------------¥
```

Mais alguns conceitos säo necessários para que o pretenso programador ASSEMBLY saiba o que está fazendo. Em eletrônica digital estuda-se a algebra booleana e aritimética com números binários. Aqui esses conceitos também são importantes... Vamos começar pela aritimética binária:

A primeira operação básica - a soma - não tem muitos mistérios... basta recorrer ao equivalente decimal. Quando somamos dois números decimais, efetuamos a soma de cada algarismo em separado, prestando atenção aos "vai um" que ocorrem entre um algarismo e outro. Em binário fazemos o mesmo:

```
+-------------------------------------------------------------+
¦ 1010b + 0110b = ?                                           ¦
¦                                                             ¦
¦    111         <- "Vai uns"                                 ¦
¦     1010b                                                   ¦
¦   + 0110b                                                   ¦
¦   ---------                                                 ¦
¦    10000b                                                   ¦
+-------------------------------------------------------------+
```

Ora, na base decimal, quando se soma - por exemplo - 9 e 2, fica 1 e "vai um"... Tomemos o exemplo do odômetro (aquele indicador de quilometragem do carro!): 09 -> 10 -> 11

Enquanto na base decimal existem 10 algarismos (0 até 9), na base binária temos 2 (0 e 1). O odômetro ficaria assim:
00b -> 01b -> 10b -> 11b

Portanto, 1b + 1b = 10b ou, ainda, 0b e "vai um".

A subtração é mais complicada de entender... Na base decimal existem os números negativos... em binário nao! (Veremos depois como "representar" um número negativo em binário!). Assim, 1b - 1b = 0b (lógico), 1b - 0b = 1b (outra vez, evidente!), 0b - 0b = 0b (hehe... você deve estar achando que eu estou te sacaneando, né?), mas e 0b - 1b = ?????

A solução é a seguinte: Na base decimal quando subtraimos um algarismo menor de outro maior costumamos "tomar um emprestado" para que a conta fique correta. Em binário a coisa funciona do mesmo jeito, mas se não tivermos de onde "tomar um emprestado" devemos indicar que foi tomado um de qualquer forma:

```
+-------------------------------------------------------------+
¦ 0b - 1b = ?                                                 ¦
¦                                                             ¦
¦     1         <- Tomamos esse um emprestado de algum lugar! ¦
¦      0b                          (não importa de onde!)     ¦
¦   -  1b                                                     ¦
¦   ------                                                    ¦
¦      1b                                                     ¦
```

```
+-------------------------------------------------------------+
```

Esse "1" que apareceu por mágica é conhecido como BORROW. Em um número binário maior basta usar o mesmo artificio:

```
+-------------------------------------------------------------+
¦ 1010b - 0101b = ?                                           ¦
¦                                                             ¦
¦      1 1         <- Os "1"s que foram tomados emprestados säo ¦
¦      1010b          subtraídos no proximo digito.          ¦
¦    - 0101b                                                  ¦
¦   ---------                                                 ¦
¦      0101b                                                  ¦
+-------------------------------------------------------------+
```

Faça a conta: 0000b - 0001b, vai acontecer uma coisa interessante! Faça a mesma conta usando um programa, ou calculadora cientifica, que manipule números binários... O resultado vai ser ligairamente diferente por causa da limitação dos digitos suportados pelo software (ou calculadora). Deixo a conclusäo do "por que" desta diferença para você... (Uma dica, faça a conta com os "n" digitos suportados pela calculadora e terá a explicação!).

```
i-------------------------------------------------------------©
¦ Representando números negativos em binário                  ¦
È-------------------------------------------------------------¥
```

Um artificio da algebra booleana para representar um número interiro negativo é usar o último bit como indicador do sinal do número. Mas, esse artificio gera uma segunda complicação...

Limitemos esse estudo ao tamanho de um byte (8 bits)... Se o bit 7 (a contagem começa pelo bit 0 - mais a direita) for 0 o número representado é positivo, se for 1, é negativo. Essa é a diferença entre um "char" e um "unsigned char" na linguagem C - ou um "char" e um "byte" em PASCAL (Note que um "unsigned char" pode variar de 0 até 255 - 00000000b até 11111111b - e um "signed char" pode variar de -128 até 127 - exatamenta a mesma faixa, porém um tem sinal e o outro näo!).

A complicação que falei acima é com relação à representação dos números negativos. Quando um número näo é nagativo, basta convertê-lo para base decimal que você saberá qual é esse número, no entanto, números negativos precisam ser "complementados" para que saibamos o número que está sendo representado. A coisa NÄO funciona da seguinte forma:

```
+-------------------------------------------------------------+
¦ 00001010b   =   10                                          ¦
¦ 10001010b   =  -10     (ERRADO)                             ¦
+-------------------------------------------------------------+
```

Näo basta "esquecermos" o bit 7 e lermos o restante do byte. O procedimento correto para sabermos que número está sendo representado negativamente no segundo exemplo é:

```
¦ Inverte-se todos os bits
¦ Soma-se 1 ao resultado
```

```
+-------------------------------------------------------------+
¦ 10001010b   ->  01110101b + 00000001b   ->  01110110b       ¦
¦ 01110110b   =   118                                         ¦
¦ Logo:                                                       ¦
¦ 10001010b   =  -118                                         ¦
```

+--+

Com isso podemos explicar a diferença entre os extremos da faixa de um "signed char":

¦ Os números positivos contam de 00000000b até 01111111b, isto é, de 0 até 127.
¦ Os números negativos contam de 10000000b até 11111111b, isto é, de -128 até -1.

Em "C" (ou PASCAL), a mesma lógica pode ser aplicada aos "int" e "long" (ou INTEGER e LONGINT), só que a quantidade de bits será maior ("int" tem 16 bits de tamanho e "long" tem 32).

Não se preocupe MUITO com a representação de números negativos em binário... A CPU toma conta de tudo isso sozinha... mas, as vezes, você tem que saber que resultado poderá ser obtido de uma operação aritimética em seus programas, ok?

As outras duas operações matemáticas básicas (multiplicação e divisão) tanbém estão presentes nos processadores 80x86... Mas, não necessitamos ver como o processo é feito a nível binário. Confie na CPU! :)