```
i-----------------------------------------------©
¦ RBT   ¦   Curso de Assembly   ¦   Aula Nº 05 ¦
È-----------------------------------------------¥
```

Por: Frederico Pissarra

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

Depois de algumas instruções de movimentação de dados vou mostrar a mecânica da lógica booleana, bem como algumas instruções.

A lógica booleana baseia-se nas seguintes operações: AND, OR, NOT. Para simplificar a minha digitação vou usar a notação simplificada: & (AND), | (OR) e ~ (NOT). Essa notação é usada na linguagem C e em muitos manuais relacionados a hardware da IBM.

¦ Operação AND:

A operação AND funciona de acordo com a seguinte tabela-verdade:

```
          +-----------+
          ¦ S = A & B ¦
          ã---Ð---Ð---Á
          ¦ A ¦ B ¦ S ¦
          +---+---+---¦
          ¦ 0 ¦ 0 ¦ 0 ¦
          ¦ 0 ¦ 1 ¦ 0 ¦
          ¦ 1 ¦ 0 ¦ 0 ¦
          ¦ 1 ¦ 1 ¦ 1 ¦
          +-----------+
```

Note que o resultado (S) será 1 apenas se A "E" B forem 1.
Aplicando esta lógica bit a bit em operações envolvendo dois bytes obteremos um terceiro byte que será o primeiro AND o segundo:

```
+-------------------------------------------------------------------+
¦              A = 01010111b       B = 00001111b                  ¦
¦                                                                  ¦
¦              S = A & B ->    01010111b                          ¦
¦                             & 00001111b                          ¦
¦                              -------------                       ¦
¦                                00000111b                         ¦
+-------------------------------------------------------------------+
```

Uma das utilidades de AND é resetar um determinado bit sem afetar os demais. Suponha que queira resetar o bit 3 de um determinado byte. Para tanto basta efetuar um AND do byte a ser trabalhado com o valor 11110111b (Apenas o bit 3 resetado).

Eis a sintaxe da instrução AND:

```
+-------------------------------------------------------------------+
¦     AND AL,11110111b                                              ¦
¦     AND BX,8000h                                                  ¦
¦     AND DL,CL                                                     ¦
¦     AND [DI],AH                                                   ¦
+-------------------------------------------------------------------+
```

Lembrando que o operando destino (o mais a esquerda) deve sempre ser um registrador ou uma referencia a memória. o operando a

direita (fonte) pode ser um registrador, uma referência a memória ou um valor imediato, com a restrição de que näo podemos usar referências a memória nos dois operandos.

A instrução AND afeta os FLAGS Z, S e P e zera os flags Cy (Carry) e O (veja os flags em alguma mensagem anterior a esta).

¦ Operação OR:

S = A | B

A	B	S
0	0	0
0	1	1
1	0	1
1	1	1

Note que S será 1 se A "OU" B forem 1.
Da mesma forma que AND, aplicamos essa lógica bit a bit envolvendo um byte ou word através de uma instrução em assembly. Vejamos um exemplo da utilidade de OR:

```
A = 01010111b       B = 10000000b

S = A | B ->    01010111b
                | 10000000b
                -----------
                  11010111b
```

A operação OR é ideal para setarmos um determinado bit sem afetar os demais. No exemplo acima B tem apenas o bit 7 setado... depois da operação OR com A o resultado final foi A com o bit 7 setado! :)

A sintaxe de OR é a mesma que a de AND (obviamente trocando-se AND por OR). Os flags afetados são os mesmos da instrução AND!

¦ Operação NOT:

NOT simplesmente inverte todos os bits de um byte ou word:

S = ~A

A	S
0	1
1	0

A sintaxe da instrução em assembly é a seguinte:

```
NOT AL
NOT DX
NOT [SI]
```

¦ Operação XOR:

A operação XOR é derivada das três acima. A equação booleana que descreve XOR é:

```
+-----------------------------------------------------------------+
¦  S = (A AND ~B) OR (~A AND B) = A ^ B                            ¦
+-----------------------------------------------------------------+
```

Que na tabela-verdade fica:

```
                          +-----------+
                          ¦ S = A ^ B ¦
                          ã---Ð---Ð---Á
                          ¦ A ¦ B ¦ S ¦
                          +---+---+---¦
                          ¦ 0 ¦ 0 ¦ 0 ¦
                          ¦ 0 ¦ 1 ¦ 1 ¦
                          ¦ 1 ¦ 0 ¦ 1 ¦
                          ¦ 1 ¦ 1 ¦ 0 ¦
                          +-----------+
```

Uso o simbolo ^ para o XOR aqui. XOR funciona da mesma forma que OR, só que o resultado será 1 se APENAS A ou APENAS B for 1, melhor dizendo: Se ambos forem diferentes.

XOR é muito útil quando se quer inverter um determinado bit de um byte ou word sem afetar os outros:

```
+-----------------------------------------------------------------+
¦              A = 01010111b       B = 00001111b                  ¦
¦                                                                 ¦
¦              S = A ^ B ->    01010111b                          ¦
¦                            ^ 00001111b                          ¦
¦                              -------------                      ¦
¦                              01011000b                          ¦
+-----------------------------------------------------------------+
```

No exemplo acima invertemos apenas os quatro bits menos significativos de A.

A sintaxe e os flags afetados são os mesmos que AND e OR.