Short Notes Chapter 4 [Lecture 10-12]

Shift Logical Right [SHR]:

The shift logical right operation inserts a zero from the left and moves every bit one position to the right and copies the rightmost bit in the carry flag.

|0|→|1|0|1|1|0|1|0|0|→|C|

Shift Logical Left [SHL]/Shift Arithmetic Left [SAL]:

In this operation the zero bit is inserted from the right and every bit moves one position to its left with the most significant bit dropping into the carry flag.

|C|←|1|0|1|1|0|1|0|0|←|0|

Shift Arithmetic Right [SAR]:

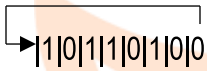
The operation of shift arithmetic right is to shift every bit one place to the right with a copy of the most significant bit left at the most significant place. The bit dropped from the right is caught in the carry basket. The sign bit is retained in this operation



|1|0|1|1|0|1|0|0|→|C|

Rotate Right [ROR]:

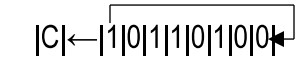
In the rotate right operation every bit moves one position to the right and the bit dropped from the right is inserted at the left. This bit is also copied into the carry flag.



|1|0|1|1|0|1|0|0|→|C|

Rotate Left [ROL]:

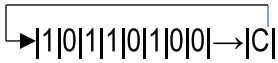
In the operation of rotate left instruction, the most significant bit is copied to the carry flag and is inserted from the right, causing every bit to move one position to the left.



|C|←|1|0|1|1|0|1|0|0|

Rotate Through Carry Right [RCR]:

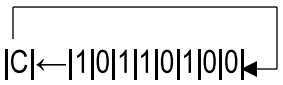
In the rotate through carry right instruction, the carry flag is inserted from the left, every bit moves one position to the right, and the right most bit is dropped in the carry flag.



|1|0|1|1|0|1|0|0|→|C|

Rotate Through Carry Left [RCL]:

In its operation the carry flag is inserted from the right causing every bit to move one location to its left and the most significant bit occupying the carry flag.



AND operation:

AND performs the logical bitwise and of the two operands (byte or word) and returns the result to the destination operand. A bit in the result is set if both corresponding bits of the original operands are set; otherwise the bit is cleared as shown in the truth table.

Examples are "and ax, bx" and "and byte [mem], 5." All possibilities that are legal for addition are also legal for the AND operation.

X	Y	X and Y
0	0	0
0	1	0
1	0	0
1	1	1

OR operation:

OR performs the logical bitwise "inclusive or" of the two operands (byte or word) and returns the result to the destination operand. A bit in the result is set if either or both corresponding bits in the original operands are set otherwise the result bit is cleared as shown in the truth table. Examples are "or ax, bx" and "or byte [mem], 5."

X	Y	X or Y
0	0	0
0	1	1
1	0	1
1	1	1

XOR operation:

XOR (Exclusive Or) performs the logical bitwise "exclusive or" of the two operands and returns the result to the destination operand. A bit in the result is set if the corresponding bits of the original operands contain opposite values (one is set, the other is cleared) otherwise the result bit is cleared as shown in the truth table. XOR is a very important operation due to the property that it is a reversible operation. It is used in many cryptography algorithms, image processing, and in drawing operations. Examples are "xor ax, bx" and "xor byte [mem], 5."

X	Y	X xor Y
0	0	0
0	1	1
1	0	1
1	1	0

NOT operation:
NOT inverts the bits (forms the one's complement) of the byte or word operand. Unlike the other logical operations, this is a single operand instruction, and is not purely a logical operation in the sense the others are, but it is still traditionally counted in the same set.

Examples are "not ax" and "not byte [mem]".

Masking Operations:

Selective Bit Clearing:

Another use of AND is to make selective bits zero in its destination operand. The source operand is loaded with a mask containing one at positions which are retain their old value and zero at positions which are to be zeroed. The effect of applying this operation on the destination with mask in the source is to clear the desired bits. This operation is called masking. For example, if the lower nibble is to be cleared then the operation can be applied with F0 in the source. The upper nibble will retain its old value and the lower nibble will be cleared.

Selective Bit Setting:

The OR operation can be used as a masking operation to set selective bits. The bits in the mask are cleared at positions which are to retain their values, and are set at positions which are to be set. For example, to set the lower nibble of the destination operand, the operation should be applied with a mask of 0F in the source. The upper nibble will retain its value and the lower nibble will be set as a result.

Selective Bit Inversion:

XOR can also be used as a masking operation to invert selective bits. The bits in the mask are cleared at positions, which are to retain their values, and are set at positions, which are to be inverted.

For example, to invert the lower nibble of the destination operand, the operand should be applied with a mask of 0F in the source. The upper nibble will retain its value and the lower nibble will be set as a result. Compare this with NOT which inverts everything. XOR on the other hand allows inverting selective bits.

Selective Bit Testing:

AND can be used to check whether particular bits of a number are set or not. AND can be applied on a destination with a 1-bit in the desired position and a source, which is to be checked. If the destination is zero as a result, which can be checked with a JZ instruction, the bit at the desired position in the source was clear.

TEST instruction:

This is the TEST instruction and is a non-destructive AND operation. It doesn't change the destination and only sets the flags according to the AND operation. By checking the flags, we can see if the desired bit was set or cleared.