Section 04 Part 07 – The ROL & ROR Instructions

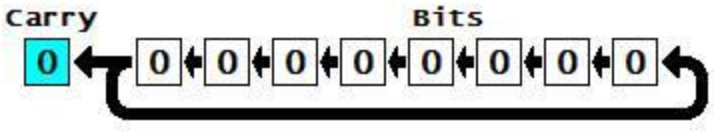
"I've learned over the years that it doesn't matter where you pitch in the rotation. For me, preparation is everything." ~Cory Lidle

Introduction

Now that we've looked into bit shifting, we can now look at bit rotation. We'll start by using the byte C4 as an example, and here it is in binary:

1100 0100

When rotating left, all bits are shifted to the left. Any bits that go outside the byte on the left come back in on the right. As shown by this graphical example:



So if we rotate the byte C4 a bit to the left, we'll get:

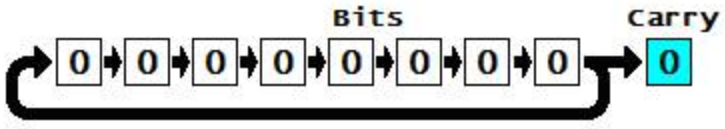
v 1000 1001 <
> > > > > > ^

Which is 89 in hex. As you can see, all of the bits have shifted to the left, and the MSB was taken out and put on the far right as the LSB. If we rotate it an extra 2 times to the left, we'll get:

v 0010 0110 <
> > > > > > ^

Where the bits 10 on the far left have shifted out of the byte, and back round to the right side.

For rotating right, the same rules apply. Any bits that go outside the byte on the right come back in on the left. As shown by this graphical example:



So we'll use the byte C4 once again:

1100 0100

And we'll rotate it to the right by 2 bits:

> 0011 0001 v
^ < < < < < <

All of the bits have shifted to the right, and the LSB bits were taken out and put on the far left as the MSB bits.

And that is the basic idea of bit rotation.

The ROL Instruction

ROL – ROtate Left (without extended)

This instruction will rotate the bits of the destination operand to the left. The number of times that the bits are rotated is decided by the source operand.

Examples

```
rol.b   #$02,d0
```

This will rotate the byte of d0, 02 bits to the left. We'll pretend that d0 contains 009F00B2 in this example, and since it's a byte, only the B2 is change. B2 in binary is:

1011 0010

After rotating the bits left by 02, we get:

```
v 1100 1010 <
> > > > > > ^
```

1100 1010 in hex is CA, which is saved back into d0. Now d0 contains 009F00CA.

Here's another example using long-word:

```
rol.l   #$04,d2
```

This will rotate the long-word of d2, 04 bits to the left. We'll pretend that d2 contains 76543210, in binary that's:

0111 0110 0101 0100 0011 0010 0001 0000

Rotating left 04 bits results in:

```
v 0110 0101 0100 0011 0010 0001 0000 0111 <
> > > > > > > > > > > > > > > > > > > > > > ^
```

Now d2 contains 65432107.

There is a small different between the shift and rotation instructions when it comes to using a data register, for example:

```
rol.l   d0,d1
```

Now as I mentioned before; for shifting, the maximum is the number of bits that can be shifted over depends on the size you've chosen.

- .b for byte gives a maximum shift of $08 bits.
- .w for word gives a maximum shift of $10 bits.
- .l for long-word gives a maximum shift of $1F bits.

For rotation, there is no maximum (technically speaking):

- Going any higher than $07 bits for byte will wrap around to $00
- Going any higher than $0F bits for word will wrap around to $00
- Going any higher than $1F bits for long-word will wrap around to $00

So, rotating a byte 09 bits is the same as rotating a byte 01 bit. 0A is the same as 02. And so on...

The rules for the shift instructions used on memory apply here for the rotation instructions too; see the previous part for details.

The ROR Instruction

ROR – ROtate Right (without extended)

This instruction will rotate the bits of the destination operand to the right. The number of times that the bits are rotated is decided by the source operand.

Examples

```
ror.b    #$02,d0
```

This will rotate the byte of d0, 02 bits to the right. We'll pretend that d0 contains 009F00B2 in this example, and since it's a byte, only the B2 is change. B2 in binary is:

1011 0010

After rotating the bits right by 02, we get:

```
> 1010 1100 v
^ < < < < < < <
```

1010 1100 in hex is AC, which is saved back into d0. Now d0 contains 009F00AC.

Here's another example using word:

```
ror.w    #$04,d2
```

This will rotate the word of d2, 04 bits to the right. We'll pretend that d0 contains 0000BA98. Only a word is changed, so BA98 in binary is:

1011 1010 1001 1000

Rotating right 04 bits results in:

```
> 1000 1011 1010 1001 v
^ < < < < < < < < < < <
```

Now d2 contains 00008BA9.

The rules for the shift instructions used on memory apply here for the rotation instructions too; see the previous part for details.

Previous Part Main Page Next Part