Z80 Assembler – Part 2

Introducation

Welcome to the second part of these tutorials, part 2 is slightly different to what I originally had in mind due to feedback from yourselves. We will return to what I was going to cover, maybe later in part 2 or if not, in part 3.

So, what are we going to look at? Number Bases. I had incorrectly assumed everyone was happy with number bases and binary logic. While I think most people know the difference, being happy with them is not a term several people would use so in response we are going to use our knowledge from Part 1 and look into this area.

Like before if you have any comments etc, then please email myself at adrian@apbcomputerservices.co.uk or post to the facebook page. I will endeavour to answer any questions and comment on any posts.

Getting Started

Number bases such as hexadecimal (hex) and binary are not nearly as scary as people think. Often using the correct number base will make your life much easier. As many of you will have noticed there is a lot of discussion on which number base is the correct one to work in, well im going to say all of them! Each one is useful in certain situations and you will make your life harder if you try and use one exclusively.

So what is a number base, well the base (or radix) is the number of digits used to represent a value in a given number column. Hopefully you are all familiar with base 10 (You may not have heard it called that) as it's the decimal system we use for most every day counting. When adding values we talk about the units column, the 10's column etc, we will often align the columns and add them that way a)

$$\begin{array}{cccc} & 5 & 7 & 3 \\ + & & 6 & 9 \\ \hline & & & \\ \hline \end{array}$$

b)

$$\begin{array}{cccc} & 5 & 7 & 3 \\ + & & 6 & 9 \\ \hline & & & \mathbf{2} \\ \hline & & 1 & \end{array}$$

c)

$$\begin{array}{cccc} & 5 & 7 & 3 \\ + & & 6 & 9 \\ \hline & & \mathbf{4} & \mathbf{2} \\ \hline & 1 & 1 & \end{array}$$

d)

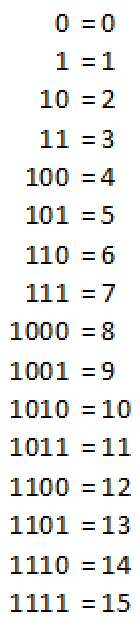
$$\begin{array}{cccc} & 5 & 7 & 3 \\ + & & 6 & 9 \\ \hline & \mathbf{6} & \mathbf{4} & \mathbf{2} \\ \hline & 1 & 1 & \end{array}$$

I would imagine many of us are taught this way of adding up numbers. Align the columns (a), then add the units 3 + 9 = 12, or as we often say here, 2 carry the 1. What do we mean by carry the 1? When counting up, when we get to 9, the next value is 10, that is 1 in the next column (the 10's column) and go back to 0 in the units column, we carried a value from the units over to the tens column to say we have 1 lot of ten and 0 units. Next we go to 11, (one ten and 1 unit). This can be seen in (b) where the 1 carried over into the 10's column is in red. Now when adding the tens column we do 7 + 6 + 1 (as we need to add in that 1 we carried over before). Doing so we get the answer 14, that is one carried over and 4 in the tens column (c). Lastly we add the hundreds column 5 + 1 = 6 (d). There is nothing carried over and no other value to add so we have the final answer – 642. What do we mean by 642, well is 6 in the hundreds column, 4 in the tens column and 2 in the units column, that is to say 6 lots of 100, 4 lots of 10 and 2 units. Or (6 * 100) + (4 * 10) + (2 * 1). This method of adding up the columns will reappear in all bases.

In this example we talked about the units column, the tens column and the hundreds column because each column is worth 10 times as much as the column before, this is why it is called base 10. This theory carries on with all other bases and we use them in certain ways in every day life. Don't believe me, think about a clock, that uses base 60 and base 24. When you get to 59 seconds you have 1 minute and 0 seconds, we have carried the 1 over into the minutes column and reset the seconds column back to 0. 59 minutes goes to 1 hour and 0 minutes. The only odd thing with a clock is the hours is in base 24.

Ok, so how does that relate?

Onto binary first, base 2. Like decimal it has columns and you can align them to do addition etc. The only difference is as its base 2 each column is worth 2 times the pervious column, when we get to the base we carry 1 over to the next column and reset, so in this case we count 0 then 1 then carry 1 over and reset to 0 giving us 10 (usually pronounced one zero rather than ten), next comes 11 (one one) then again we have to carry the 1 over from the units to the 2's column and reset, but now we have 1+1 in the 2's column so we carry it over to the 4's column (remember each column is worth 2 times – the bases value – the previous column) giving 100 (one zero zero) which is 1 lot of 4, 0 lots of 2 and 0 lots of units, i.e 4. So 100 in binary is worth 4 in decimal. If we carry on counting up to 15 you get



0	=0
1	=1
10	=2
11	=3
100	=4
101	=5
110	=6
111	=7
1000	=8
1001	=9
1010	=10
1011	=11
1100	=12
1101	=13
1110	=14
1111	=15

As with decimal we are happy the first column is the units (going from right to left), then tens, hundreds, thousands and so on. In binary we have similar just each being 2 times the previous so units, 2's, 4's, 8's, 16's. Below are the values of the first 8 columns

Value	128	64	32	16	8	4	2	1

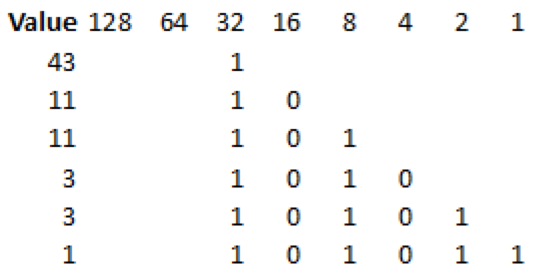
Why 8 columns…. Well each column holds what we call a bit (a Binary Digit), that is to say a 0 or a 1. 8 of these bits we refer to in programming as a byte (yes its also called a char and other things, but in assembly we call it a byte). The lowest value we can store in 8 bits is when they are all 0, that is a value of 0. The highest is when they are all 1's. so 1 lot of 128 add 1 lot of 64…. Giving 128+64+32+16+8+4+2+1 = 255. This is why the maximum value a byte can store is the value 255 (and why the next column is worth 256 or 2 * 128).

Converting between decimal and binary

Converting from binary to decimal is simply a case of adding up the column values as with decimal, remembering the value of each column, so 1101 in binary is 1 lot of 8 add 1 lot of 4 add 0 lot of 2 add 1 unit or (1 * 8) + (1 * 4) + (0* 2) + (1 * 1) so in decimal that would be 13. 1101 binary is 13 in decimal. Going from decimal to binary is in my opinion the worst conversion to do, the process is as follows.

1) Start with the largest column value that isnt great than the number and put a 1 in this column.
2) Subtract the column value from the number
3) Step to a column down (to the right) until we have done all the columns
4) If the new number is greater than the column value write a 1 in this next column and goto step 2, else write a 0 and goto step 3

Lets say we want to covert 43 decimal into binary, step 1 says we start with the column worth 32 (as the next column is 64 and 43 is less than 64).



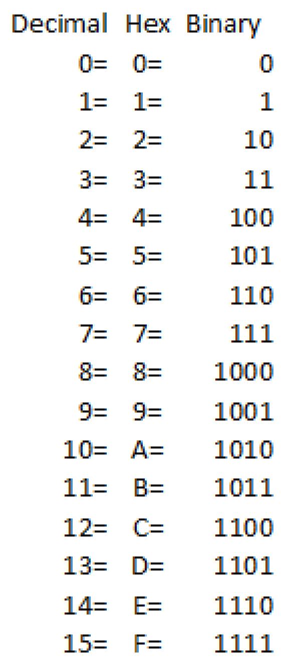
Value	128	64	32	16	8	4	2	1
43			1					
11			1	0				
11			1	0	1			
3			1	0	1	0		
3			1	0	1	0	1	
1			1	0	1	0	1	1

So 43 in decimal is 101011 in binary. If you got lost, re-read that last bit, as I say this is the worst conversion you have to deal with and often you wont do this way, but it is worth looking at.

Binary has a lot of uses in computers as all digital devices at their lowest level only deal with on/off, 1 or 0. Nothing else, every complex thing you see on a computer comes back to that. We will come back to binary soon… But lets look at another base, base 16 (hexadecimal).

Base 16 – Hexadecimal (hex)

This is the base a lot of people will say is the best base for everything, most things yes, but I wouldn't go with everything. This base is VERY closely linked to binary, why do I say that? Well hex has values from 0 to 15 in each column before it carries over (hence base 16), as you saw at the start of the binary section, 4 binary digits are required to hold the value 15, 16 would need the next binary column and we would end up with the binary 10000. All the hex digits can be represented with 4 binary digits (bits). The problem is we can use 0-9 decimal values to represent 0-9 in hex, but how do we represent 10 decimal in hex? To save confusion where 10 could be ten lots of the units column OR one lot of the second column and no units hex moves onto the letters A-F to represent 10,11,12,13,14,15 decimal respectively, so.



Decimal	Hex	Binary
0=	0=	0
1=	1=	1
2=	2=	10
3=	3=	11
4=	4=	100
5=	5=	101
6=	6=	110
7=	7=	111
8=	8=	1000
9=	9=	1001
10=	A=	1010
11=	B=	1011
12=	C=	1100
13=	D=	1101
14=	E=	1110
15=	F=	1111

Like in binary we would say 10 as one zero, in hex we would usually say 10 in hex as one zero to save confusion.

Converting between hex and binary.

When you need to convert between hex and binary (and you will do this more than you realise) its actually a simple process. As mentioned before, each hex digit is four bits (Believe it or not four bits is called a nibble – or nybble – I prefer nibble myself. Why? A nibble is a small bite (byte) 😉).

Converting to binary is just a case of writing down the 4 bits relating to each hex digit in turn. To start with you may find yourself referring to the chart shown previously but you will soon find it just sticks in your mind. Lets look at A3D hex in binary, first we get the four bits for the A – 1010, next 3 which is 0011 and finally D 1101. So A3D hex is 1010 0011 1101 binary.

Going from binary to hex is just the reverse process, from lowest bit upwards split the binary into groups of 4 bits (if you have less than 4 bits in the last group just add 0's to make it up to 4 bits). So 111101011101 binary becomes…

1111 0101 1101. Take each group in turn and work out the hex using the table again.

1111 = F
0101 = 5
1101 = D

So 111101011101 in binary is F5D in hex.

Which base is it??

How can you tell what base a number is written in? Well if you saw 101 written down it would be hard to say for sure. There are many different conventions that have appeared over the years, when a number is just presented on its own it is usually said to be in decimal. Sometimes a lowercase 'd' is added to the end, 101d. If it was in binary a lowercase 'b' is added to the end or a % is added to the start %101 or 101b. A hex number is written with a lowercase 'h' at the end or a $ or & or 0x at the start! So &101, $101, 0x101 or 101h. Ive also seen the lowercase letter put at the start of the number. It will come down to which assembler you use, although most will support more than one format for the numbers.

Part 3..

I was going to start on binary logic and bit shifting in this part, but its fairly long as it is, so to keep these bite size and easy to reference ill leave this here. Part 3 we will get onto those and there will be more challenges to get your teeth into. This was a fairly important topic to make sure everyone knows about the main number bases. If you have any questions, shout 😊