CS 245 Assembly Language Programming

Intro to Computer Math

Text: *Computer Organization and Design, 4th Ed.*, D A Patterson, J L Hennessy
Section 2.4

Objectives: The Student shall be able to:

- Convert numbers between decimal, binary, hexadecimal
- Add binary and hexadecimal numbers.
- Perform logical operations: AND and OR on binary or hexadecimal numbers
- Determine the range of possible numbers given a number of bits for storage.
- Form or translate a negative number from a positive number and vice versa.

Class Time:

Binary, Octal, Hexadecimal	1 hour
Signed and Unsigned numbers	1 hour
Exercise	1 hour
Total	3 hours

Hello Binary!

Imagine a world of 1s and 0s – no other numbers exist. Welcome to the world of the computer. This is how information and instructions are stored in the computer.

Well, what happens when we add 1+1? We must get 10.
What happens if we add 10+1? We get 11.
What happens if we add 11+1? We get 100. Do you see the pattern? Try it for yourself below, by continually adding one to get the decimal number on the left:

1	1	11		21	
2	10	12		22	
3	11	13		23	
4	100	14		24	
5	101	15		25	
6		16		26	
7		17		27	
8		18		28	
9		19		29	
10		20		30	

Notice: what is the value of each digit? For example, if we have a binary number: B11111 what does each binary number stand for? For example, in decimal the 11111 number would be: 1+10+100+1000+10,000. Using the same idea, what do the binary values: 1, 10, 100, 1000, 10000 translate into in decimal?

Do you notice that each binary digit is basically a double of the digit to its right?

1	1	1	1	1	1	1	1
128	64	32	16	8	4	2	1

That is very important to remember. Always remember that each place is multiplied by 2! Translate the following binary numbers to decimal using this rule:

B 1010101 =

B 0101010 =

B 1110001 =

B 1100110 =

EXERCISE: BINARY ADDITION

Addition using Binary	Checking with Decimal
B 0101 + B 1010 = B 1111	5 + 10 = 15
B 1100 + B 0011 = B 1111	12 + 3 = 15
B 1001 + B 0011 = B 1100	9 + 3 = 12

Let's try something more complicated: B 1111 1111 + B 1001 1100 =

```
Carry:   1 1 1 1   1
          B 1 1 1 1   1 1 1 1
         +B 1 0 0 1   1 1 0 0
         B11 0 0 1   1 0 1 1
```

Add the following numbers:

Binary	Check your work with the Decimal Equivalent
0001 0110	
0011 0100	
1011 1001	
1001 1001 0110 0110	
1000 0000 0001 1111	
1010 1010 0101 0111	
1001 1001 1100 1100	

EXERCISE: AND & OR

Now let's play with AND and OR.

AND: If both bits are set, set the result: &
OR: If either bit is set, set the result: |

We can define truth tables for these operations. The bold italicized numbers IN the table are the answers. The column header and row header are the two numbers being operated on.

AND &	0	1
0	***0***	***0***
1	***0***	***1***

This table shows that:
0 & 0 = 0   1 & 0 = 0   0 & 1 = 0   1 & 1 = 1

<table>
<tr><td>OR |</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>

This table shows that:
0 | 0 = 0   1 | 0 = 1   0 | 1 = 1   1 | 1 = 1

I will show how these operations work with larger binary numbers:

<table>
<tr><td>B 1010101</td><td>B 1010101</td><td>B 1010101</td><td>B 1010101</td></tr>
<tr><td>& <u>B 0101010</u></td><td>| <u>B 0101010</u></td><td>AND <u>B 1110001</u></td><td>OR <u>B 1110001</u></td></tr>
<tr><td>B 0000000</td><td>B 1111111</td><td>B 1010001</td><td>B 1110101</td></tr>
</table>

Now you try some:

<table>
<tr><td>B 1100110</td><td>B 1100110</td><td>B 0111110</td><td>B 0111110</td></tr>
<tr><td>AND <u>B 1111000</u></td><td>OR <u>B 1111000</u></td><td>& <u>B 1001001</u></td><td>| <u>B 1001001</u></td></tr>
</table>

Below, show what binary value you would use to accomplish the operation. Then do the operation to verify that it works! Bits are ordered: 7 – 6 – 5 – 4 – 3 – 2 – 1 – 0

Using ORs to turn on bits:	Using ANDs to turn off bits:
B 000 0000 Turn on bits 0-3	B 1111 1111 Turn off bits 3-4
B 1111 0000 Turn on bit 0	B 1111 1111 Turn off bits 0-3
B 0000 1111 Turn on bits 3-4	B 1111 0000 Turn off bits 0-4

Let's build a table for each numbering system. You fill in the blanks…

Decimal = Base 10	Binary = Base 2	Octal = Base 8	Hexadecimal = Base 16
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16			10
17	10001		11
18	10010	22	12
19	10011	23	13
20	10100	24	14
21	10101	25	15
22	10110	26	16
23		27	17
24	11000	30	18
25	11001	31	19
26	11010	32	1A
27	11011	33	
28	11100	34	
29	11101	35	
30	11110	36	1E
31	11111	37	1F
32	100000	40	20

There is something very special about Base 8 and Base 16 – they are compatible with Base 2. So for example, let's take the binary number11000 = 24_{10}. Notice that Base 8 operates basically modulo 8, whereas base 16 operates modulo 16. It is not easy to convert between decimal and binary, but it is easy to convert between binary and octal or hexadecimal.
It is useful to know that the octal or base 8 number $324_8 = (3 \times 8^2) + (2 \times 8) + 4$
And the hexadecimal or base 16 number $324_{16} = (3 \times 16^2) + (2 \times 16) + 4$

Hello Octal!

Binary is rather tedious isn't it? It is hard to keep track of all those 1s and 0s. So someone invented base 8 and base 16. These are also known as octal and hexadecimal systems, respectively. The octal (base 8) numbering system works as follows:

Base 8: 1 2 3 4 5 6 7 10 11 12 13 14 15 16 17 20 21 22 23 24 25 26 27 30

To convert between binary and octal:

Step 1: group the binary digits by threes, similar to how we use commas with large numbers:

B 110011001100 becomes B 110 011 001 100
B 11110000 becomes B 11 110 000

Step 2: Add zeros to the left (most significant digits) to make all numbers 3 bit numbers:

B 11 110 000 becomes B 011 110 000

Step 3: Now convert each three bit number into a octal number: 0..7

B 110 011 001 100 becomes 6314_8
B 011 110 000 becomes 360_8

Likewise we can convert from Octal to Binary:

577_8 = 101 111 111
1234_8 = 001 010 011 100

Now you try!

Binary -> Octal	Octal -> Binary
B 01101001=	264_8=
B 10101010=	701_8=
B 11000011=	076_8=
B 10100101=	567_8=

If we want to convert from octal to decimal, we do:

$893_8 = (8x8^2) + (9x8^1) + (3x8^0) = 8x64 + 9x8 + 3 = 587$

Now you try! 127_8 = 1000_8=

64_8= 212_8=

Hello Hexadecimal!

The hexadecimal (base 16) number systems work as follows:

Base 16: 1 2 3 4 5 6 7 8 9 A B C D E F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30

Since base 16 needs more digits (after 9) we add A B C D E F. Therefore, A=10, B=11, C=12, etc. It helps to be able to memorize some hexadecimal digits. For example I remember that:

0xA = 10_{10} = 1010_2
0xC = 12_{10} = 1100_2
0xF = 15_{10} = 1111_2

and then I simply remember that B1101 = B1100 (or 12) +1 = 13.

To convert from binary to hexadecimal:
Step 1: group the binary digits by fours. A 4-bit number is called a **nibble**:

B 110011001100	becomes	B 1100 1100 1100
B 11110000	becomes	B 1111 0000
B 111000111	becomes	B 1 1100 0111

You may use commas, instead of or in addition to spaces, to separate digits.

Step 2: Add zeros to the left (most significant digits) to make all numbers 4 bit numbers:

B 1 1100 0111	becomes	B 0001 1100 0111

Step 3: Now convert each three bit number into a hexadecimal number: 0..7

B 1100 1100 1100	becomes	0x ccc
B 1111 0000	becomes	0x f0
B 0001 1100 0111	becomes	0x 1c7

It is good to get some practice! Try translating the following numbers to hexadecimal:

Binary	Hexadecimal	Binary	Hexadecimal
10011001 = 1001 1001	0x99	0011 1100	
11001110 = 1100 1110	0xCE	0001 1110	
101111 = 10 1111	0x2F	1110 0001	
0011 0011		1011 0100	
1100 0011		0110 1001	
1010 0101		0101 1010	
1001 1001		1100 0011	

Now convert from hexadecimal back to binary:

Hexadecimal	Binary	Hexadecimal	Binary
0x23		0x31	
0x4a		0x58	
0x18A	0001 1000 1010	0xAB1	
0x23F		0xC0D	
0x44		0xF00	1111 0000 0000
0x3C		0x28	
0x58		0x49	

Okay, now we can convert between binary and hexadecimal and we can do ANDs and ORs. Let's try doing these together. Let's AND hexadecimal numbers together:

```
0x254 AND 0x0f0 =      0010,  0101,  0100
                    &  0000, 1111,  0000
                       0000, 0101,  0000  =  0x 050
```

Notice that what we are doing is that we convert the hexadecimal to binary, and do the AND, and convert the resulting binary digits back to hexadecimal. Let's try an OR:

```
0x254 OR 0x0f0 =       0010,  0101,  0100
                    |  0000,  1111,  0000
                       0010,  1111,  0100  =  0x 2f4
```

ANDs and ORs are useful to turn on and off specific bits. Now you do some:

0x1a3 & 0x111 =

0x273 & 0x032 =

0x1a3 | 0x111=

0x273 | 0x032 =

Conversions: Hexadecimal ←→ Binary ←→ Decimal

There are two ways to convert between Base 16 or Base 8 … and Decimal.

Method 1: Convert to Binary, then Decimal:

0x1af = 0001 1010 1111
$= 2^0 + 2^1 + 2^2 + 2^3 + 2^5 + 2^7 + 2^8$
$= 1 + 2 + 4 + 8 + 32 + 128 + 256$
$= 431_{10}$

0x456 = 0100 0101 0110 = $2^{10}+2^6+2^4+2^2+2^1 = 1024 + 64 + 16 + 4 + 2 = 1110_{10}$

Method 2: Use division remainders:

Convert from base 10 to base N (Example base 2):

Number / 2 -> remainder is digit_0
-> quotient / 2 -> remainder is digit_1
-> quotient / 2 -> remainder is digit_2
…

Example 1: Convert 36_{10} into binary:

Quotient/2	->Remainder
36/2	->0
18/2	->0
9/2	->1
4/2	->0
2/2	->0
1/2	->1

$36_{10} = 100100_2$

Example 2: Convert 36_{10} into base 16:

36/16 ->4
2/16 ->2
$36_{10} = 24_{16}$

Example 3: Convert 0x1af to base 10:

0x1af = 1×16^2 =	=	256
a x 16 = 10 x 16	=	160
f	=	<u>+15</u>
Total	=	431_{10}

Now you try some conversions between base 16 and base 10 (Your choice of method!)

0x AC4=

0x C4A=

43_{10}=

162_{10}=

Signed & Unsigned Numbers

Assuming 1 byte:

Binary	Signed	Unsigned
00000000	0	0
00000001	1	1
00000010	2	2
01111110	+126	+126
01111111	+127	+127
10000000	-128	+128
10000001	-127	+129
10000010	-126	+130
11111110	-2	+254
11111111	-1	+255

Notice that you get HALF of the total positive numbers with signed integers!!!

Example: Convert 10101010 to a signed 8-bit integer:

Converting to Decimal: Powers of Two

The sign bit (bit 7) indicates both sign and value:
If top N bit is ‘0’, sign & all values are positive: top set value: 2^N
If top N bit is ‘1’, sign is negative: -2^N
Remaining bits are calculated as positive values:

$10101010 = -2^7 + 2^5 + 2^3 + 2^1 = -128 + 32 + 8 + 2 = -86$

$01010101 = 2^6 + 2^4 + 2^2 + 2^0 = 64 + 16 + 4 + 1 = 85$

Changing Signs: Two’s Compliment

A positive number may be made negative and vice versa using this technique
Method: Take the inverse of the original number and add 1.

Original:	01010101 = 85	10101011 = -85
invert:	10101010	01010100
add 1:	+1	+1
sum:	10101011 = -85	01010101 = 85

First we determine what the range of numbers is for signed versus unsigned numbers:

	4 bits		8 bits		12 bits	
	Low	High	Low	High	Low	High
Unsigned	0000	1111				
Signed	1000	0111				
Total number of possible numbers	Unsigned: Low = 0 High = 15 Set of 16 numbers					

Now you try some conversions between positive and negative numbers. Assume 8-bit signed numbers (and top bit is signed bit).

Hexadecimal value:	Actual Signed Decimal Value:	Change sign:
0x 87	$-2^7+2^2+2^1+2^0 =$ -128 + 7 = -121	In binary: 1000 0111 Invert: 0111 1000 Add 1: 0111 1001 = 0x 79 Translate: 64+32+16+8+1=121
0x ba		
0x fc		
0x 03		
0x 81		
0x 33		

Real-World Exercise: Conversion

Let's do something USEFUL! Below is a table to show how IP headers are formatted. In yellow is shown the formatting for an ICMP header for a PING message.

<table>
<tr><td>0</td><td>4</td><td>8</td><td>16 17 18</td><td>19 … 31</td></tr>
<tr><td>Version</td><td>HLenth</td><td>Service Type</td><td colspan="2">Total Length</td></tr>
<tr><td colspan="3">Datagram Identification</td><td>Flags</td><td>Fragment Offset</td></tr>
<tr><td colspan="2">Time to Live</td><td>Protocol</td><td colspan="2">Header Checksum</td></tr>
<tr><td colspan="5">Source IP Address</td></tr>
<tr><td colspan="5">Destination IP Address</td></tr>
<tr><td colspan="2">Type</td><td>Code</td><td colspan="2">Checksum</td></tr>
<tr><td colspan="3">Identifier</td><td colspan="2">Sequence Number</td></tr>
<tr><td colspan="5">Data</td></tr>
</table>

You are writing logic to decode this hexadecimal sequence and now you want to verify that the interpreted packet is correct – you must convert it manually to verify!

4500 05dc 039c 2000 8001 902b c0a8 0004
c0a8 0005 0800 2859 0200 1c00 6162 6364
6566 6768 696a 6b6c 6d6e 6f70 7172 7374

What are the **decimal** values for the following fields:

Word 1: Version: HLenth: Total Length:

Word 2: Datagram Id: Fragment Offset:

Word 2: A flag is a one-bit field. Flags include bits 16-18:
Don't Fragment (Bit 17): More Fragment (Bit 18):

Word 3: Time to Live: Protocol:

For the two addresses below, convert each byte in word to decimal and separate by periods (e.g., 12.240.32.64):

Word 4: Source IP Address:

Word 5: Destination IP Address:

ICMP: Type: Code: Sequence Number: