



Level 0x0A

Binary Tricks



Topics

- Events
- Hacker History
- Ones and Zeros

Ongoing Events

- December hacking contests
 - Advent of Cyber - [TryHackMe.com](https://tryhackme.com)
 - Sans Institute [Holiday Hack Challenge](#) - Less CTF, more game, probably more incident responder and defender focused
 - [Advent of Code](#) - Programming challenges. Used to be 25 2-parters, now reducing down to 12 days of challenges
 - [Pwn.College](#) is going to do an Advent of Pwn



Cyber Quest and Code Quest

- Code Quest
 - Saturday, Feb 28th
 - Registration Nov 17th -
- Spring Break
 - March 23rd - 27th
- Cyber Quest
 - Saturday, March 28th
 - Registration Jan 5th -



Code Quest

- Details
 - Saturday, February 28th, 9:00 - 1:30-ish
 - Teams are 2-3 students each, 4 teams max per school
 - 1 computer / laptop per team (one keyboard and one mouse too 😜)
 - No cell phones inside, no cameras (not even outside)
 - No buses, must provide your own transportation (they can't stay on site during the competition)
 - Breakfast and Lunch provided
- What you need to provide me
 - Team members and teams (team names)
 - Birthday (must be 11 yrs and older, middle-school officially allowed)
 - Citizenship (not related to ICE current politics, standard procedure for military contractors)
 - Return 2 Lockheed forms (Liability and Photo Release)
 - Return 1 school permission slip



Cyber Quest

- Details:
 - Saturday, March 28th, 9:00 - 1:30 ish
 - Teams are 3-5 students each, 4 teams max per school
 - Very large monitors discouraged
 - No cell phones inside, no cameras (not even outside)
 - No buses, must provide your own transportation (they can't stay on site during the competition)
 - Breakfast and Lunch provided
- What you need to provide me
 - Team members and teams (team names)
 - Birthday (must be 14 yrs and older)
 - Citizenship (not related to ICE current politics, standard procedure for military contractors)
 - Return 2 Lockheed forms (Liability and Photo Release)
 - Return 1 school permission slip



Robert Morris

- Created the first internet worm (not the first worm)
 - Worm = malware that replicates and spreads itself
- Father (also Robert Morris) worked at Bell Labs and NSA
- Grad school student at Cornell
- Morris Worm spread by exploiting
 - Bug in `sendmail` debug mode
 - Buffer overflow in `fingerd` (gives you info about other users)
 - Weak user passwords
- First person to ever be prosecuted by Computer Fraud and Abuse Act
 - Govt: Tens of thousands of infections, \$200-\$50,000 per computer to fix
 - 3 yrs probation, 400 hrs community service, \$10,000 fine
 - Could have been a much harsher sentence per guidelines
 - Leniency because he never intended to cause damage



Binary Numbers

10110101	=	0xB5	=	181
01101101	=	0x6D	=	109

Addition

- Online 6502 Assembler and Emulator
 - <https://skilldrick.github.io/easy6502/>

A handwritten binary addition diagram. It shows two 8-bit binary numbers being added:

1	1	1	1	1	1	1	1
1	0	1	1	0	1	0	1
+	0	1	1	0	1	1	0
<hr/>							
1 0 0 1 0 0 0 1 0							

The result is shown as $0x122 = 290$. A curly brace under the last three bits of the sum is labeled "8-bit Carry Flag".

Screenshot of the online 6502 Assembler and Emulator interface. The top menu bar includes: Assemble, Run, Reset, Hexdump, Disassemble, and Notes.

The assembly code area contains:

```
LDA #$b5
ADC #$6d
```

The status register area shows:

Debugger

A=\$00 X=\$00 Y=\$00
SP=\$ff PC=\$0600
NV-BDIZC
00110000

Step Jump to...

The monitor area at the bottom left shows:

Monitor Start: \$0 Length: \$ff

The assembly log area at the bottom right shows:

```
Preprocessing ...
Indexing labels ...
Found 0 labels.
Assembling code ...
Code assembled successfully, 4 bytes.
```

Let's Try It Out

- Assemble program
- Reset CPU
- Toggle on Debugger
- Step instruction 1 instruction

LDA = LoaD Accumulator

- Accumulator = 0xB5

The screenshot shows a debugger interface with the following components:

- Top Bar:** Assemble, Run, Reset, Hexdump, Disassemble, Notes.
- Assembly View:** Displays the assembly code:

```
LDA #$b5
ADC #$6d
```
- Registers View:** Shows the state of registers:

<input checked="" type="checkbox"/> Debugger
A=\$b5 X=\$00 Y=\$00
SP=\$ff PC=\$0602
NV-BDIZC
10110000
- Control Buttons:** Step, Jump to...
- Monitor Bar:** Monitor Start: \$0 Length: \$ff
- Message Log:** Displays the assembly process:

```
Preprocessing ...
Indexing labels ...
Found 0 labels.
Assembling code ...
Code assembled successfully, 4 bytes.
```

Two green arrows point from the text "Accumulator = 0xB5" to the "A=\$b5" entry in the Registers view, and from the text "LDA = LoaD Accumulator" to the first instruction in the Assembly view.

Let's Try It Out

- Step instruction 1 more

ADC = ADd with Carry

- Accumulator = 0x22
- Carry Flag is set to 1

The screenshot shows a debugger interface with the following components:

- Top Bar:** Assemble, Run, Reset, Hexdump, Disassemble, Notes.
- Assembly View:** LDA #\$b5
ADC #\$6d
- Registers View:** Debugger checked. Registers: A=\$22, X=\$00, Y=\$00, SP=\$ff, PC=\$0604, NV-BDIZC, 00110001. Buttons: Step, Jump to...
- Monitor View:** Preprocessing ..., Indexing labels ..., Found 0 labels., Assembling code ..., Code assembled successfully, 4 bytes.

Green arrows point from the list items to the corresponding parts of the debugger interface: one arrow points from "Step instruction 1 more" to the assembly code, another from "Accumulator = 0x22" to the register values, and a third from "Carry Flag is set to 1" to the NV-BDIZC status in the registers view.

What does “with Carry” mean?

- Added a new instruction

ADC #\$00 = ADd with Carry

- We add 0 to the accumulator...
- Plus 1 for the Carry Flag

- Accumulator is now 0x23
- Carry flag is now 0

The screenshot shows a debugger interface with the following components:

- Assembly View:** Contains the assembly code:

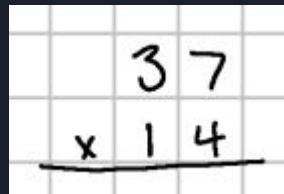
```
LDA #$b5
ADC #$6d
ADC #$00
```
- Registers View:** Shows the state of registers:

A=\$23	X=\$00	Y=\$00
SP=\$ff	PC=\$0606	NV-BDIZC
00110000		
- Buttons:** Includes "Step" and "Jump to..." buttons.
- Monitor View:** Displays the monitor command input fields: "Monitor" checkbox, "Start: \$0", and "Length: \$ff". Below it, the monitor output shows:

```
Preprocessing ...
Indexing labels ...
Found 0 labels.
Assembling code ...
Code assembled successfully, 6 bytes.
```

Multiplication

- How do we do multiplication in base-10 number system?



A handwritten multiplication problem on a grid background. The top row contains the number 37. Below it, a line with an 'x' and a line with 14 are aligned under the 7. A horizontal line is drawn below the 14.

$$\begin{array}{r} 37 \\ \times 14 \\ \hline \end{array}$$

Multiplication

- How do we do multiplication in base-10 number system?

$$\begin{array}{r} 37 \\ \times 14 \\ \hline \end{array}$$

$$\begin{array}{r} ^2\overline{)37} \\ \times 14 \\ \hline 148 \\ 37 \\ \hline 518 \end{array}$$

$$\begin{array}{r} A B C D \\ \times 1234 \\ \hline \end{array}$$

- What about binary or base-16?

It works in binary / hex too!



A handwritten multiplication diagram on grid paper. The top row shows $0x29$ above two columns of letters: A B and C D. Below this is a multiplication sign (\times). The next row shows a horizontal line with three columns of digits below it: 2 2 B C, 3 4, and A 4. To the left of this row is a plus sign (+). The bottom row shows another horizontal line with two columns of digits: 2 2 E 5 and A 4.

$0x29$

A B C D

\times

2 2 B C 3 4 A 4

+

2 2 E 5 A 4

$CD \times 34 = 29A4$

$AB \times 34 = 22BC$

Sign Bit?

Handwritten multiplication diagram:

$$\begin{array}{r} \text{A B C D} \\ \times 1 2 \\ \hline \text{E 5 A 4} \\ \hline \text{0 C 0 6 6 A} \\ \hline \text{C 3 7 4 F A 4} \end{array}$$

Calculation details:

$$C D \times 1 2 = 0 E 6 A$$
$$A B \times 1 2 = C 0 6$$

```
>>> hex(0xabcd * 0x1234)
'0xc\u003374fa4'
```

Negative integers

- Let's propose using the highest / top bit as a sign bit
 - 1 = negative
 - 0 = positive
- Could represent -127 to 127
- What is -0 ???

S	X X X	X X X X
↑ sign bit		7-bits
0	0 0 0 0 0 0 0	= 0x00 = 0
0	1 1 1 1 1 1 1	= 0x7F = 127
1	0 0 1 0 0 1 1	= -0x13 = -35
1	1 1 1 1 1 1 1	= -0x7F = -127
1	0 0 0 0 0 0 0	= -0x0 = ???

Negative integers

- Let's propose using the highest / top bit as a sign bit



~~S X X X X X X X~~
~~↑ 7-bits~~

	0 0 0 0 0 0 0	= 0x00	= 0
0	1 1 1 1 1 1 1	= 0x7F	= 127
1	0 0 1 0 0 1 1	= 0x13	= -35
1	1 1 1 1 1 1 1	= -0x7F	= -127
1	0 0 0 0 0 0 0	= -0x0	= ???

Two's Complement

- To invert the sign of a number
 - Invert all the bits
 - Add 1
- That top bit is kind-of a sign bit
 - If it's a 1, it's a negative number
 - But we don't have negative 0
- -2^N to $2^N - 1$ -128 to 127 for 8-bit



$$\begin{array}{r} 00000101 = 5 \\ \\ \begin{array}{r} 11111010 \\ + 1 \\ \hline 11111011 \end{array} = 0x\text{FB} = -5 \end{array}$$

$$\begin{array}{r} 01111111 = 127 \\ \\ \begin{array}{r} 10000000 \\ + 1 \\ \hline 10000001 \end{array} = -127 \end{array}$$

Two's Complement

- Negative 1 is always all 1's / all F's
 - 8 bit => 0b11111111 = 0xFF
 - 16-bit -> 0b1111111111111111 = 0xFFFF
 - And so on...
- If you forget how to do a 2's complement
 - Remember -1 is all F's
 - Remember invert all the bits

0000 0001 1
+ 1111 1110 ← what do I need to do?

Example Math

Subtraction is essentially adding a negative number...

Handwritten subtraction example:

85 - 42 = 0x55 - 42 = 0x2A

Binary representation:

00101010 - 00101010 = -42

Binary addition:

11010101 + 1 = 11010110 = -42

Final result:

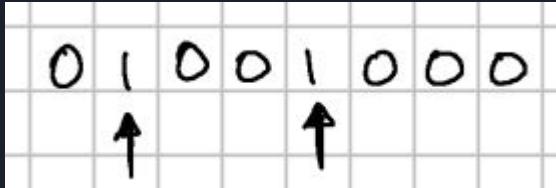
$$\begin{array}{r} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ + & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ \hline 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{array}$$

throw away

$= 0x55 = 81$
 $= -42$
 $\Rightarrow 0x2B = 43$

Bit Counter

- How can I count the bits that are set in a number?
 - POPCNT on x86
- You could loop through all bits (shifting a 1) and test each bit
 - Takes a lot of loops for 64-bit values



```
unsigned int popcount_shift(uint32_t x)
{
    unsigned int count = 0;

    while (x != 0) {
        // Check lowest bit (LSB)
        if (x & 1u) {
            count++;
        }

        // Shift right by 1 bit, discarding LSB
        x >>= 1;
    }

    return count;
}
```

Try This...

- Subtract 1 from the number
- Bitwise-AND with the original number

01001000 Subtract 1

$\begin{array}{r} 01001000 \\ + 11111111 \\ \hline 01000111 \end{array}$

$\Leftarrow -1$

01001000 And with original
8 $\underline{- 01000111}$ number
 $\hline 01000000$

Counted 1 bit

01000000 Sub 1

$\begin{array}{r} 01000000 \\ + 11111111 \\ \hline 00111111 \end{array}$

and with orig

01000000
8 $\underline{- 00111111}$
 $\hline 00000000$

2 loops
0 bits set now



Bit Twiddling Hacks

- <https://graphics.stanford.edu/~seander/bithacks.html>
 - Full of the most clever (evil) math / algorithms you will ever see

Counting bits set, Brian Kernighan's way

```
unsigned int v; // count the number of bits set in v
unsigned int c; // c accumulates the total bits set in v
for (c = 0; v; c++)
{
    v &= v - 1; // clear the least significant bit set
}
```



Links

- <https://risky.biz/HTWGO1/> - About Robert Morris case
- <https://skilldrick.github.io/easy6502/>
- <https://graphics.stanford.edu/~seander/bithacks.html>