ETGG1801 – Game Programming Foundations I

Andrew Holbrook

Fall 2013

Introduction to Computers

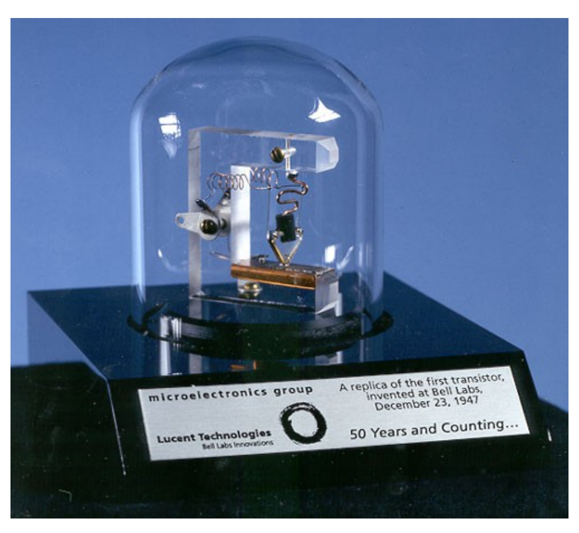
- Vacuum Tubes and Transistors
  - Electrically-controlled switches
- Logic Gates
  - AND, OR, NAND, NOR, XOR, etc.
- Binary Numeral System
  - Base-2 number system
  - Symbols: 0 or 1
  - Addition in binary
  - Compact representation using hexadecimal
- Converting Between Numeral Systems

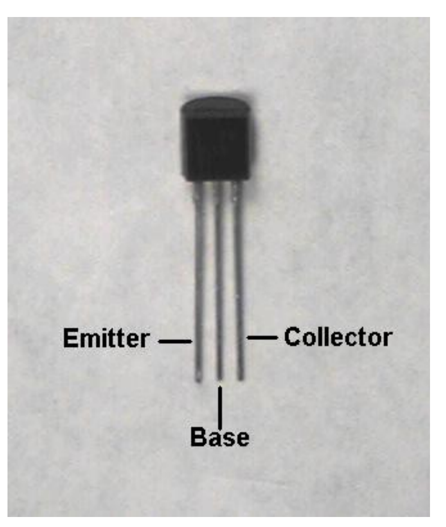
Vacuum Tubes

- Control the flow of electrons from one electrode to another.
- Can be used as an electronic switch.
- Bulky and inefficient when compared to transistors.

ENIAC 1

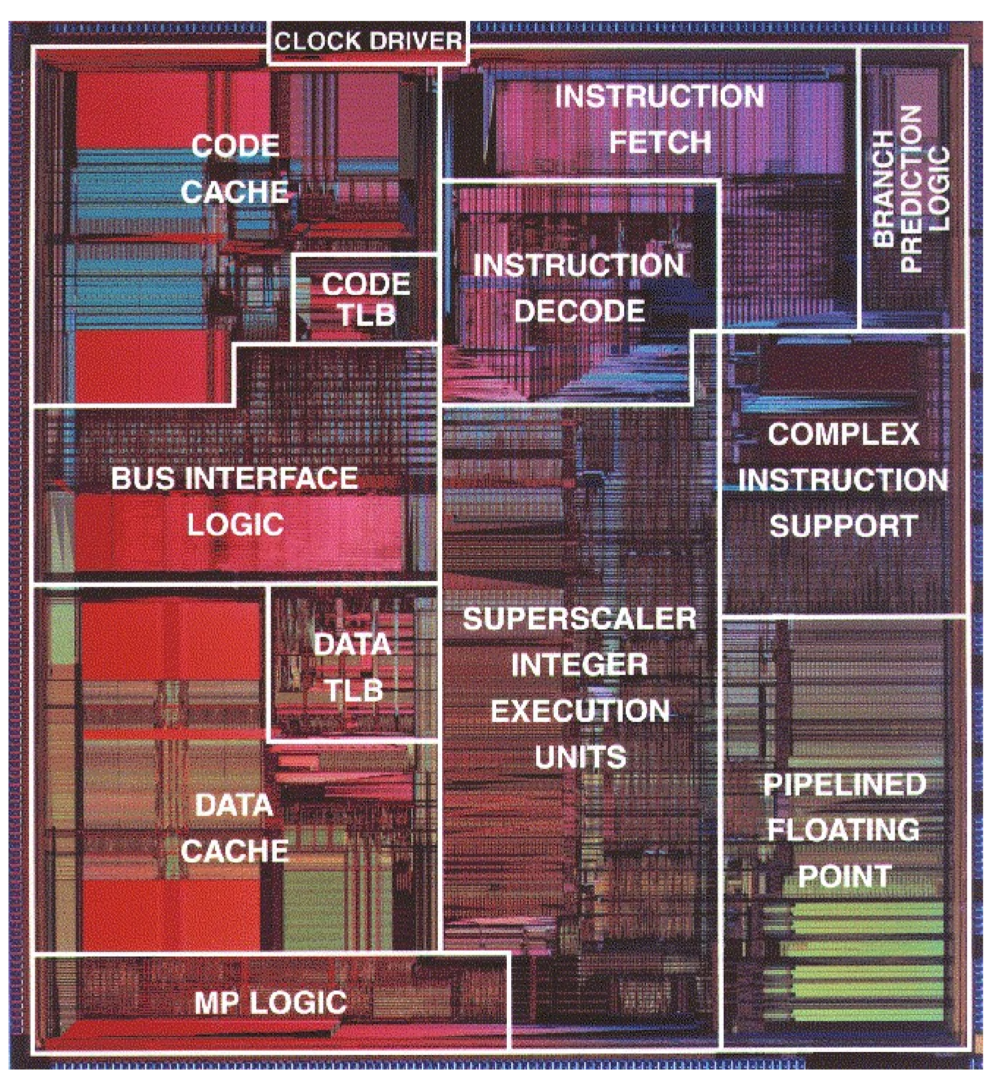
Transistors





- First silicon transistor created by Texas Instruments in 1954.
- Smaller and more efficient than vacuum tubes.
- Used in electronics as amplifiers, switches (on/off), etc.

Intel Pentium



Logic Gates

- Transistors are used to build circuits called 'logic gates.'
- Logic gates take an input (0-off / 1-on) and produce an output (0/1).
- Implement Boolean logic (True-1 / False-0).

Name	Graphic Symbol	Algebraic Function	Truth Table
AND	A B F	$F = A \cdot B$ or $F = AB$	A B | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR	A B F	$F = A + B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT	A F	$F = \bar{A}$ or $F = A'$	A | F 0 | 1 1 | 0
NAND	A B F	$F = \overline{(AB)}$	A B | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR	A B F	$F = \overline{(A + B)}$	A B | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0

Binary Numeral System

- Base-2 (decimal is base-10).
- Each digit corresponds to a place value of 2^n
  - Each digit in decimal corresponds to a value of 10^n
- Decimal:

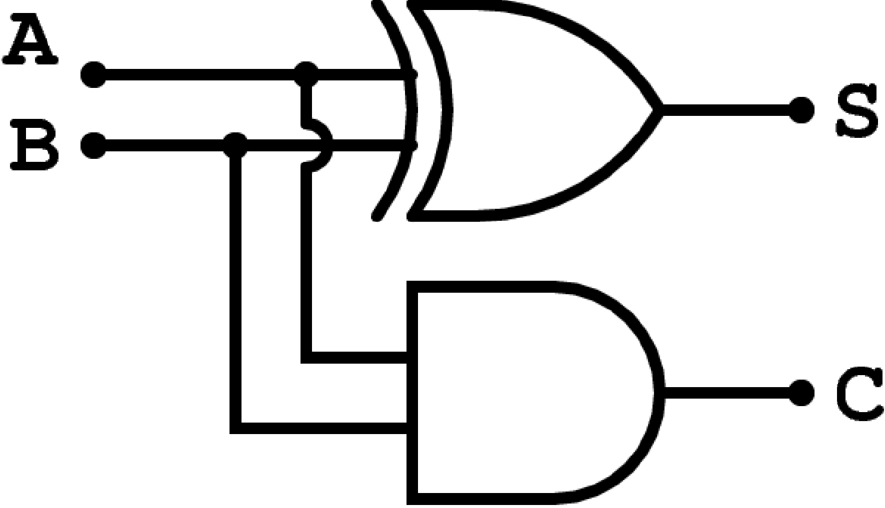
n	4	3	2	1	0
value (decimal)	10000	1000	100	10	1

- Binary:

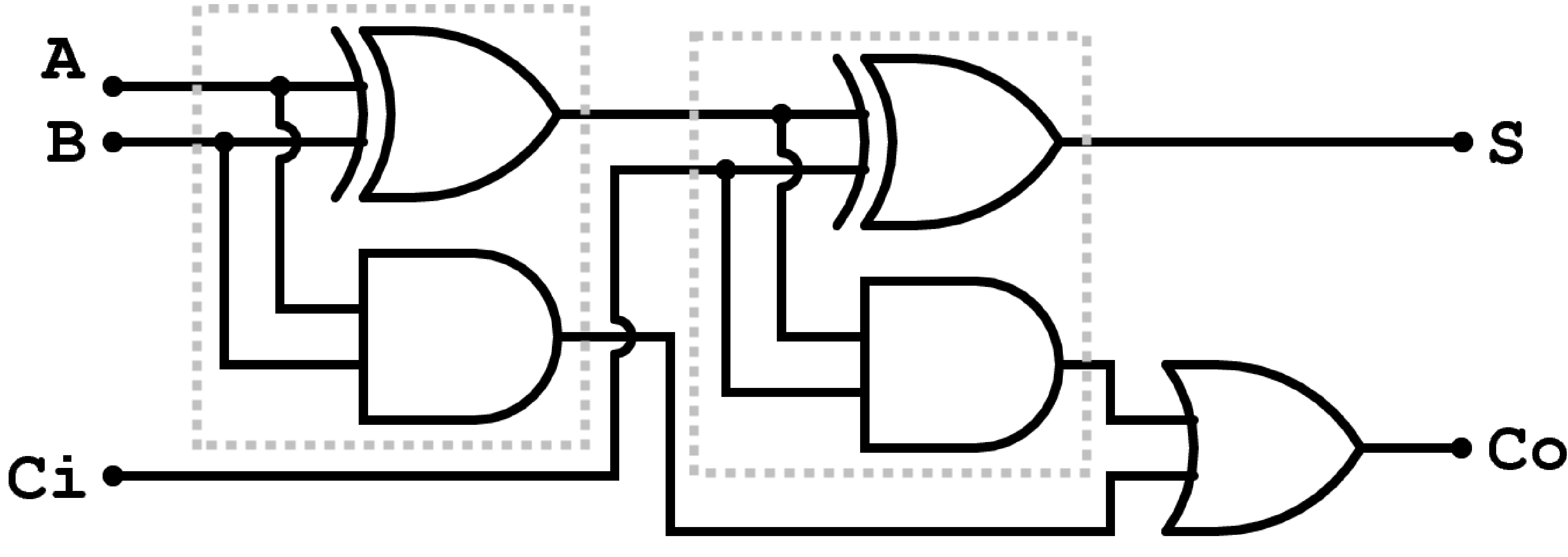
n	4	3	2	1	0
Value (decimal)	16	8	4	2	1

Addition in Binary

- Similar to addition in decimal.
- A and B are inputs.
- S is the result for a digit.
- C is the carry.



Addition in Binary



Hexadecimal (hex)

- Binary numbers can have many digits.
- 1437 (decimal) = 10110100111 (binary)
- Hex is a more compact representation.
- Binary can be easily converted to hex.
- 10110100111 (binary) = 5A7 (hex)
  - Hex is base-16
- Since decimal and hex share some of the same symbols (0-9), '0x' is prepended to hex values. (e.g. 0x5A7)

Hexadecimal (hex)

- Being base-16, hex digits are represented with 16 symbols: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
  - Decimal has 10 symbols: 0-9
  - Binary has 2: 0 and 1
- A in hex is 10 in decimal, B is 11, C is 12, etc.
- Each digit in hex represents four binary digits.

Hexadecimal (hex)

Hexadecimal	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Binary to/from Hex

- 11101001100101010

0001	1101	0011	0010	1010
1	D	3	2	A

- 0x1D32A

- 0xA4B2

A	4	B	2
1010	0100	1011	0010

- 1010010010110010

Binary to Decimal

- 101011

2^5 (32)	2^4 (16)	2^3 (8)	2^2 (4)	2^1 (2)	2^0 (1)
1	0	1	0	1	1

  - Multiply binary digit by its value and add.
- From right to left:

Binary digit * (place value)
1 * (1)
1 * (2)
0 * (4)
1 * (8)
0 * (16)
+ 1 * (32)
43

Decimal to Binary

- 107
- Find the largest binary place value that is less than the decimal value.

2^7 (128)	2^6 (64)	2^5 (32)	2^4 (16)	2^3 (8)	2^2 (4)	2^1 (2)	2^0 (1)
	1						

  - 128 is greater than 107, so pick 64.
- Put a '1' in that digit and subtract that value from the decimal value.
  - 107 – 64 = 43
- Repeat until you have 0.

Decimal to Binary

- 107

2^7 (128)	2^6 (64)	2^5 (32)	2^4 (16)	2^3 (8)	2^2 (4)	2^1 (2)	2^0 (1)
	1						

- 107 – 64 = 43

2^7 (128)	2^6 (64)	2^5 (32)	2^4 (16)	2^3 (8)	2^2 (4)	2^1 (2)	2^0 (1)
	1	1					

- 43 – 32 = 11

2^7 (128)	2^6 (64)	2^5 (32)	2^4 (16)	2^3 (8)	2^2 (4)	2^1 (2)	2^0 (1)
	1	1	0				

  - 16 is greater than 11, so put a '0'

2^7 (128)	2^6 (64)	2^5 (32)	2^4 (16)	2^3 (8)	2^2 (4)	2^1 (2)	2^0 (1)
	1	1	0	1	0	1	1

Hex to/from Decimal

- Hex to Decimal
  - 1) Convert hex value to binary
  - 2) Convert binary value to decimal
- Decimal to Hex
  - 1) Convert decimal value to binary
  - 2) Convert binary value to hex

Intro to Computers, cont'd

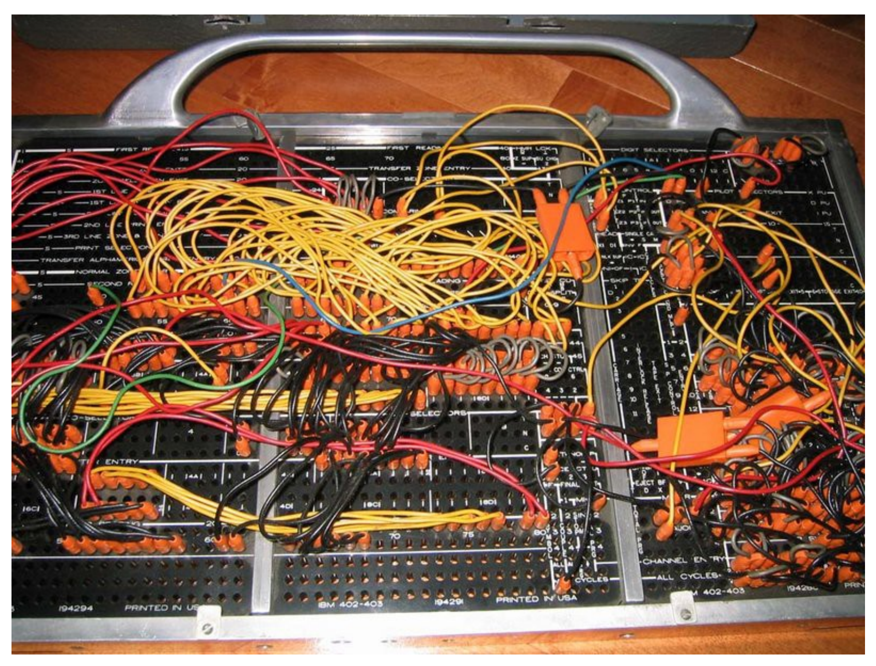
- Central Processing Unit (CPU)
  - Instructions/Operations
  - Inputting Instructions: Plugboard and punch cards
  - Memory Devices
- Interacting with Devices
  - Address and Data Buses
  - Memory Map
- How Data is Stored in Memory
  - Images and sound

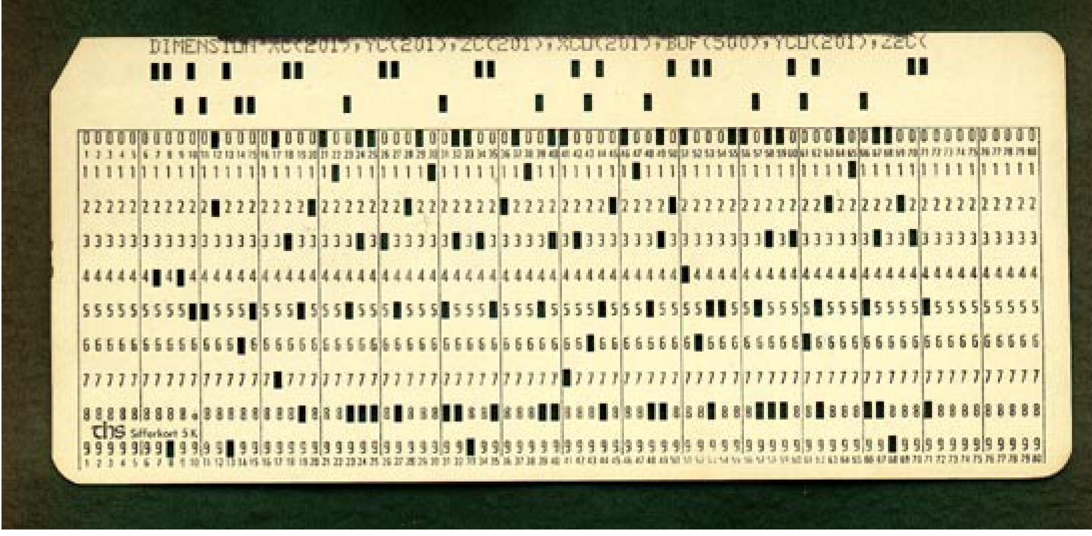
CPU

- The "brain" of the computer
- Types of instructions include:
  - Addition, subtraction, multiplication, division
  - Compare
    - Compare two values
  - Jump/Branch
    - Tell CPU to execute instructions at a specified location

Inputting Instructions

- The CPU must be able to access instructions
- Historically, plugboards or punch cards were used





Inputting Instructions, cont'd

- Memory Devices
  - Hard disk drives (HDD), RAM, ROM
  - Instructions can be stored in files (programs/applications) on the HDD, or in ROM (BIOS)
  - RAM takes less time to access than a HDD, so it's typically used to store currently running programs (the instructions)
  - The CPU will fetch and execute instructions located in memory.

Interacting with Devices

- The CPU communicates with devices via the address and data bus.
- When the CPU wants data from a device, it will post the address of the device on the address bus and wait for the device to send the data over the data bus.
- A map of all accessible addresses, and devices at those address, is called a memory map.

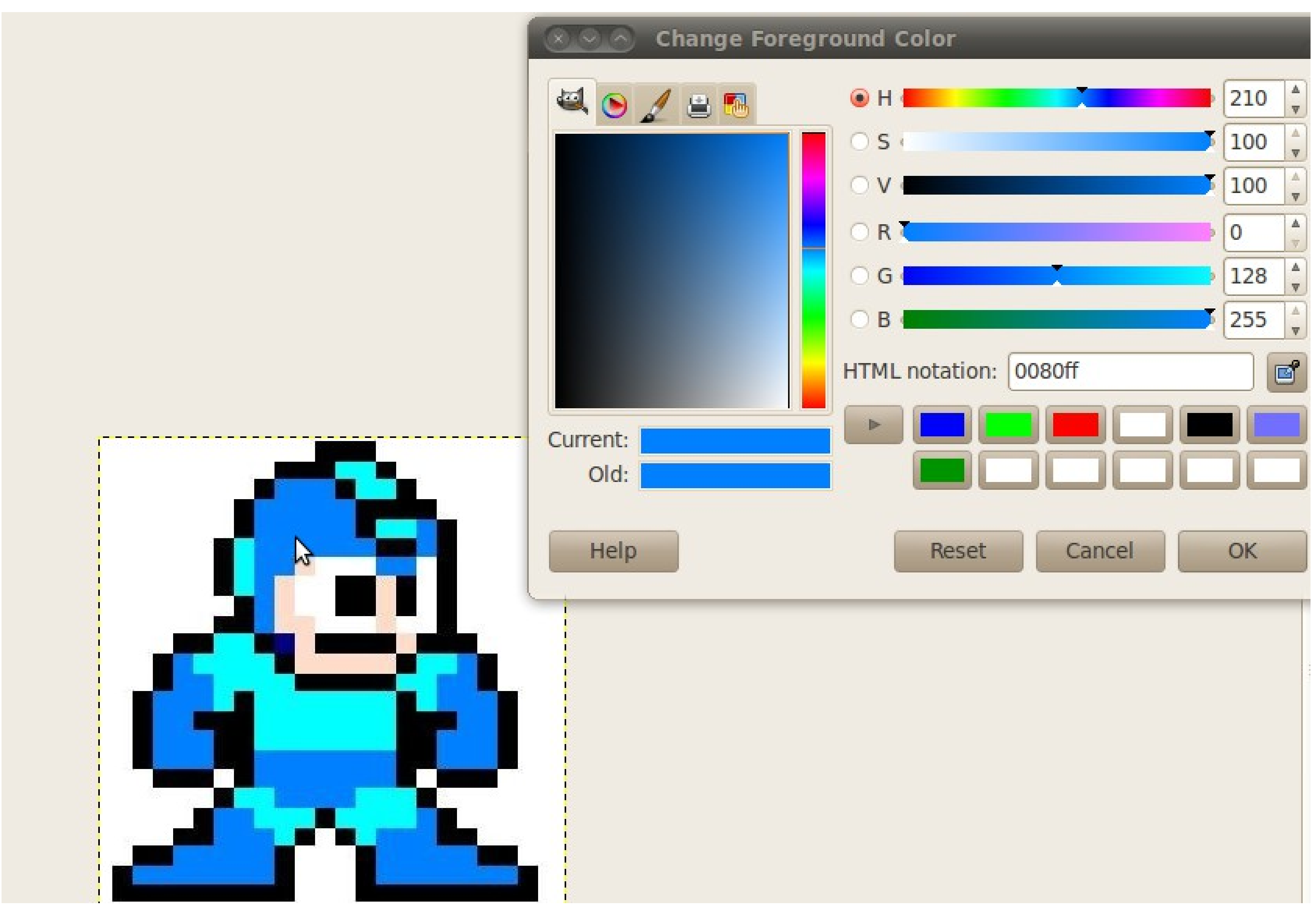
Memory Map

- (Draw map here)

How Images are Stored in Memory

- Storing an image in memory requires converting it into numeric values (memory devices store data as 0s and 1s).
- This means choosing the number of bits to use for each pixel (8,16,24,32,etc), and also how the bits are formatted (how many bits for red, green, blue, and alpha color components).

Example Image



How Sounds are Stored

- Analog sound waves are **sampled** at a configurable rate (sample rate).
- The number of bits to use per sample must also be determined.
- For example, a sound could be sampled at 44.1 kHz using 8 bits (1 byte) per sample.

Introduction to Programming

- A program is a set of instructions for the CPU to execute.
- A programmer must write these instructions in such a way that the CPU does what he/she wants.
- Writing each individual instruction would be impractical for most application, including gaming and simulations (even simple games can have millions of instructions)!

Low-Level Programming

- Low-Level programming languages require knowledge of the CPU's instruction set.
- Assembly language is an example of a low-level programming language.
- After writing assembly code, an **assembler** will convert the instruction to **machine code** (the numeric values that represent each instruction).

Assembly Code Example

```
_start:
        ldr r0, =REG_DISPCNT
        ldr r1, =MODE_3
        orr r1, r1, #BG2
        strh r1, [r0]

main:
        ldr r0, =REG_KEYINPUT
        ldr r0, [r0]
        ands r0, r0, #1

        ldr r0, =120
        ldr r1, =80
        bleq draw_pixel

        b main

draw_pixel:
        @ Parameters: r0 - x coordinate
        @             r1 - y coordinate

        stmfd r13!, {r0-r3}

        ldr r2, =WIDTH
        ldr r3, =SCREEN

        mla r0, r1, r2, r0
        add r0, r3, r0, LSL #1

        mvn r3, #0

        strh r3, [r0]

        ldmfd r13!, {r0-r3}
        mov r15, r14
```

High-Level Programming

- High-level programming languages allow programmers to write software for a CPU without knowing the instruction set.
  - Some high-level languages include C/C++, Objective-C, Java, Python, LISP, PHP, etc.
- When a programmer writes code in a high-level language, he/she must then **compile** the code using a **compiler**. The compiler generates **assembly code**, which can then be run through the **assembler** to generate **machine code**.

High-Level Programming, cont'd