Today

- Preparing for Project 1
  - C for gcc vs bcc
  - Hex in C
  - Git

Assign 2 Review

- Code from main.c

```
char buffer[30];
…
if (*buffer == '.')
    break;
```

buffer is an array but it's being compared to a character.
Why does this work?
What is the *real* condition for when our program stops reading in input?

Example of a Level of Indirection

- Output to stdout OR a file

```
FILE* output = stdout;
// default behavior

…
case ‘f’:
    char* filename = *argv + 2
    output = fopen( filename, “w”);
    break;
…
fprint(output, “output”);
```

output points to the desired output stream

Use Piazza

Review

- What are the two modes that the OS can run in?
- Why do these two modes exist?
- How can we switch between the two modes?
- How does a computer boot?

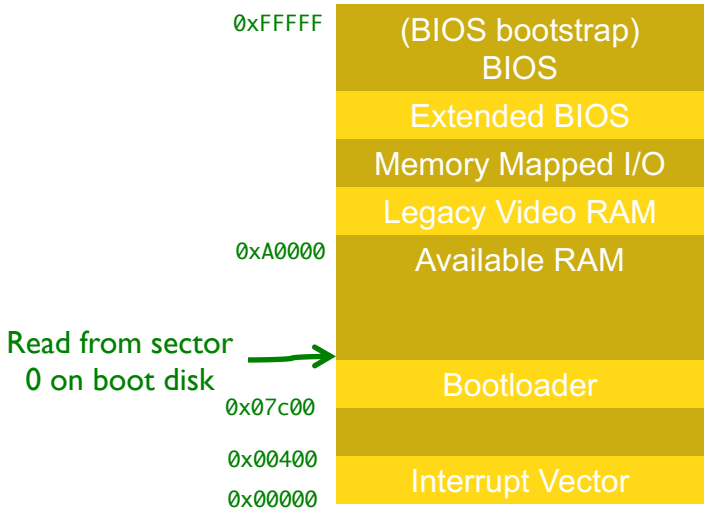
Our OS Project

- "**Build an operating system from scratch: a project for an introductory operating systems course**" by Michael Black
- 6 Projects: Build on each other
  - Project #1 – Introduction and Booting
  - Project #2 – System calls
  - Project #3 – Loading & Executing Programs + Command Line Shell
  - Project #4 – Writing Files + Improved Shell
  - Project #5 – Processes and Multiprogramming

Intel Architecture Bootstrap Process

- Machine starts in 16-bit real mode
  - 16-bit registers, 20-bit memory addresses
    - Extend range of addressable memory locations beyond what was possible using 16-bit addresses
- Instruction Pointer (IP) initialized to address of BIOS bootstrap
  - 0xFFFF0
- BIOS bootstrap program runs
  - Loads sector 0 from boot disk at 0x07C00
    - Sector 0 C:H:S = 0:0:1 ← *starts at 1!*
    - C:H:S = Cylinder:Head:Section addressing
  - Jumps to 0x07C00

16-bit Real Mode Memory Map

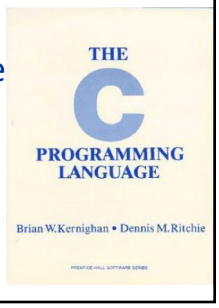


Segmented Memory Access in 16-bit Real Mode

- Registers hold 16 bits but memory addresses are 20 bits (?!??!!)
- All addresses have 2 parts:
  - Segment – 16 bits
  - Offset – 16 bits
    - E.g. `0x1000` : `0xABCD`
      (`0x1000` = segment, `0xABCD` = offset)
- Computing the actual address:
  - `address = segment*0x10 + offset`
    - Add extra 0 to right of segment and add offset.
    - E.g. `0x1ABCD`

bcc

- We'll be using bcc to compile our programs
  - bcc – Bruce's C Compiler
- Produces 8086 executables that can run in 16-bit real mode
- Understands original K&R C Syntax + a few extensions if the –ansi flag is used
  - K&R = Brian Kernighan and Dennis Ritchie
  - 1978 – *The C Programming Language*



My First (bcc) C Program

```
#include "stdio.h"

int main() {
  printf("Hello World!");
}
```

- Execution begins in `main` function.
- *Don't get used to using printf or stdio.h!*

Variables

```
int main() {
  int i=0;
  int x=2;

  printf("Numbers Divisible by %d\n", x);
  for (i=0; i<10; i++) {
    if (i % x == 0) {
      printf("%d ", i);
    }
  }
  printf("\n");
}
```

- All local variables must be declared at start of block.
  - Immediately following {

Integer Data

```
int main() {
  int a=171;
  int b=0x00AB;

  char c = 15;
  char d = 0x0F;

  if (a == b && c == d) {
    printf ("OK!\n");
  }
}
```

- Integer Types:
  - int 16-bit 2's complement
  - char 8-bit 2's complement
- Can mix decimal and hex values.

Character Data

```
int main() {
  char ch1 = 'A';
  char ch2 = 'B';

  if (ch1 == 'A' && ch2 == 66) {
    printf ("OK!\n");
  }
}
```

- char values can be specified:
  - Using single quotes ('A')
  - Using ASCII values

Function Prototypes

```
int indexOf(char *str, char ch);

main() {
  char *str = "Abc123!\0";
  int index = indexOf(str, '2');
  printf("%d\n",index);
}

int indexOf(char *str, char ch) {
  int i=0;
  while(str[i] != ch) {
    if (str[i] == '\0') {
      return -1;
    }
    i++;
  }
  return i;
}
```

- Functions must be declared before they are called.
  - A prototype is included before main.
  - Function names may not be overloaded.

and conversions

BITS & BYTES

Data Storage with Bits

- All data stored in bits (binary digits): 0/1, T/F
- Byte = 8 bits

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

=19

- A byte can store 0 – 255
- 4 bytes (a word) can store 0 – 2^{32}-1 (≈4.3M)
- 32-bit architecture = max 4 GB memory where each memory location is a byte (8 bits)

Endian Order

- We've been writing the most significant bits to the left, following the mathematic convention
- But where are the most significant bits/bytes stored? Left to right or right to left?

Byte 3	Byte 2	Byte 1	Byte 0

Little Endian

Byte 0	Byte 1	Byte 2	Byte 3

Big Endian

- Important for parsing & memory layout

Counting in Number Systems

Decimal	Binary	Octal	Hexadecimal
0	00000	00	0x0
1	00001	01	0x1
2	00010	02	0x2
3	00011	03	0x3
4	00100	04	0x4
5	00101	05	0x5
6	00110	06	0x6
7	00111	07	0x7
8	01000	010	0x8
9	01001	011	0x9
10	01010	012	0xA
11	01011	013	0xB
12	01100	014	0xC
13	01101	015	0xD
14	01110	016	0xE
15	01111	017	0xF
16	10000	020	0x10
17	10001	021	0x11
18	10010	022	0x12
19	10011	023	0x13

In C

- Octal begins with a 0
- Hexademical begins with 0x
- Addresses usually in hex

Number Conversion Self-Assessment

- Convert 1100 binary to hexidecimal
- Convert 11001100 binary to hexadecimal

Number Conversion Self-Assessment

- Convert 1100 binary to hexadecimal
  - 1100 → 12 → C → 0xC
- Convert 11001100 binary to hexadecimal
  - 0xCC
  - Note that the above number is a ***byte***
    - A byte has a higher ”nibble” and a lower “nibble”
    - Bytes are made up for two hexadecimal numbers

Version control

GIT

Git Review

- What is version control for?
- What are the basic operations for version control?
- What are the operations for git specifically?

Common Git Commands

Command	What it does
add [file]	Adds the file to the staging area
commit	Commits all the staged files (locally)
push	Push all your changes to the remote → You need your code to be pushed so that I can see it.
branch	List all local branches
branch [name]	Creates a new branch with that name

Typical, Simple Workflow

- Clone the project
- Update files
- When you've hit a good checkpoint, add the changed files to the "staging area" and then commit those files
  - Add a descriptive comment about what you've done.
- If you are ready to put your code on GitHub (doesn't need to be complete), push
  - I recommend doing this at least at the end of every "work session"

Looking Ahead

- Project 1: Due Tuesday, October 2 at 11:59:59
  - By **Wednesday**, you should be able to display an 'A' in the top left of your emulated screen.
- Approximation of learning curve
  - Start early!