Welcome to CSCI-110

Imperative Problem Solving

What will we do in this class?

- Learning to solve problems with a programming language
- Develop algorithms to solve problems
- Use functions to modularize our programs and promote “code reuse”
- Debug issues with programs
- Examine data representations for types in a programming language
- Learn good coding style/practices
- Introduce some software engineering principles

What technologies will we use?

- Editor (VSCode)
- Linux/Unix Development Environment
- Version Control (Git)
- Compiler (GCC)
  - We will write programs in C
- Build System (Make)

How to be successful in this class

- Come to each class and **arrive on time**
- Read every assignment completely the day it is assigned
  - Even if you don't intend to work on it just yet
- Work on assignments early
  - There is always unforeseen issues and/or nuance that will leave you stressed if you procrastinate too long
- Do all the assignments
  - We get better at programming by practicing
- ASK ME, TA, CA, and ZIs QUESTIONS!
  - Come to us before Google or AI…we're better and we don't have ads! :D

Why C?

- Developed to provide an alternative to assembly languages
- High Performance and Efficiency
- Basic abstractions to simplify code
- Low level features (manual memory management)

What is C used for?

- Real-time systems
  - Avionics, Cars
- Embedded systems
  - IOT/Wearables
- Operating System Kernels
  - Unix/Linux/macOS and Windows
- Drivers
  - Video Cards, Wifi, Bluetooth, etc.
- Other programming language tools
  - Python is written in C!
  - Some libraries

What’s wrong with assembly…

AVR Microcontroller Assembly

```
.INCLUDE <m328pdef.inc>

.DSEG
A:     .BYTE 1
B:     .BYTE 1
C:     .BYTE 1

.CSEG
Adder88:
lds  r0,A      ; load
lds  r2,B
add  r0,r2   ; add
sts  C,r0      ; store
rjmp Adder88
```

What’s wrong with assembly…nothing

AVR Microcontroller Assembly

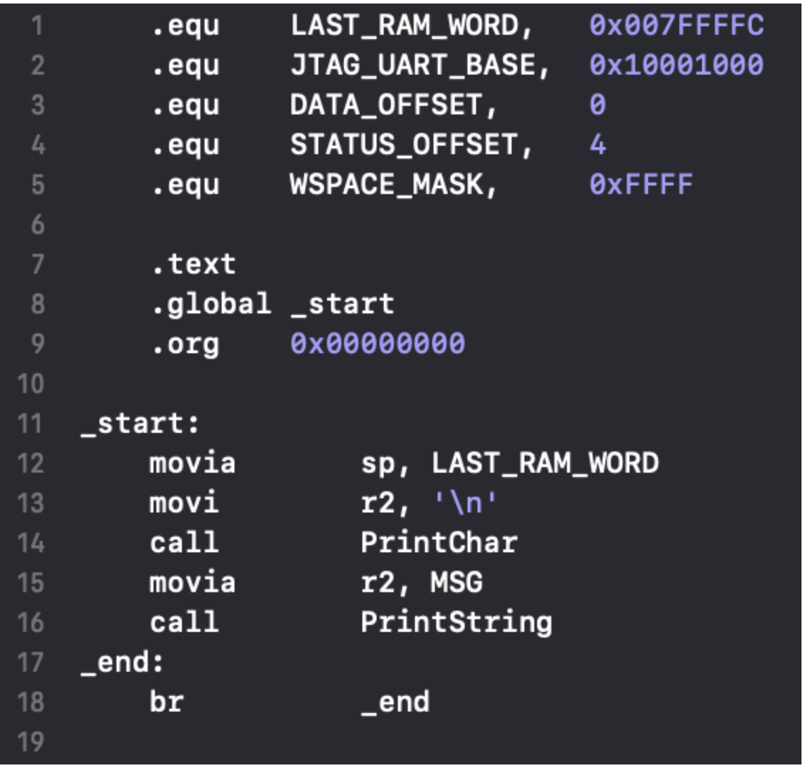
```
.INCLUDE <m328pdef.inc>

.DSEG
A:    .BYTE 1
B:    .BYTE 1
C:    .BYTE 1

.CSEG
Adder88:
lds  r0,A      ; load
lds  r2,B
add  r0,r2   ; add
sts  C,r0      ; store
rjmp Adder88
```

C

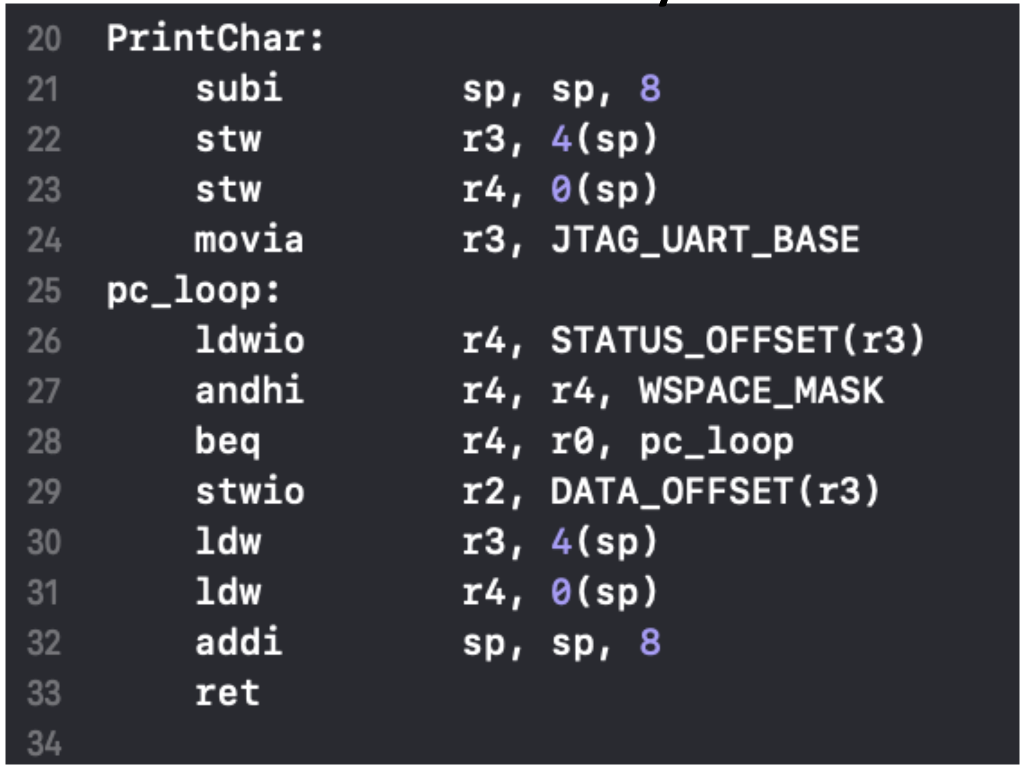
```
int main() {
    int a = 5;
    int b = 5;
    int c = a + b;
    return 0;
}
```

What's wrong with assembly…well…

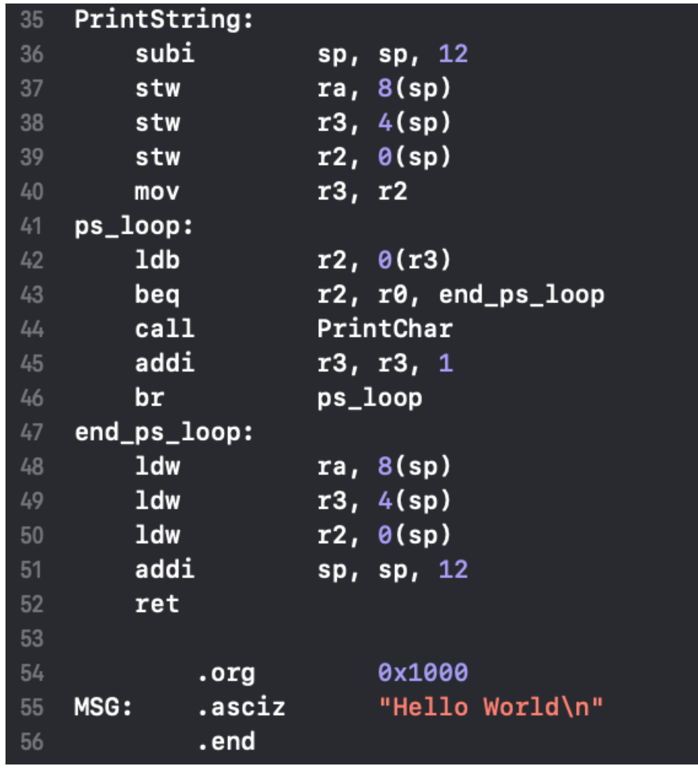
Intel Assembly



```
        .equ    LAST_RAM_WORD,    0x007FFFFC
        .equ    JTAG_UART_BASE,   0x10001000
        .equ    DATA_OFFSET,      0
        .equ    STATUS_OFFSET,    4
        .equ    WSPACE_MASK,      0xFFFF

        .text
        .global _start
        .org    0x00000000

_start:
        movia         sp, LAST_RAM_WORD
        movi          r2, '\n'
        call          PrintChar
        movia         r2, MSG
        call          PrintString
_end:
        br            _end
```



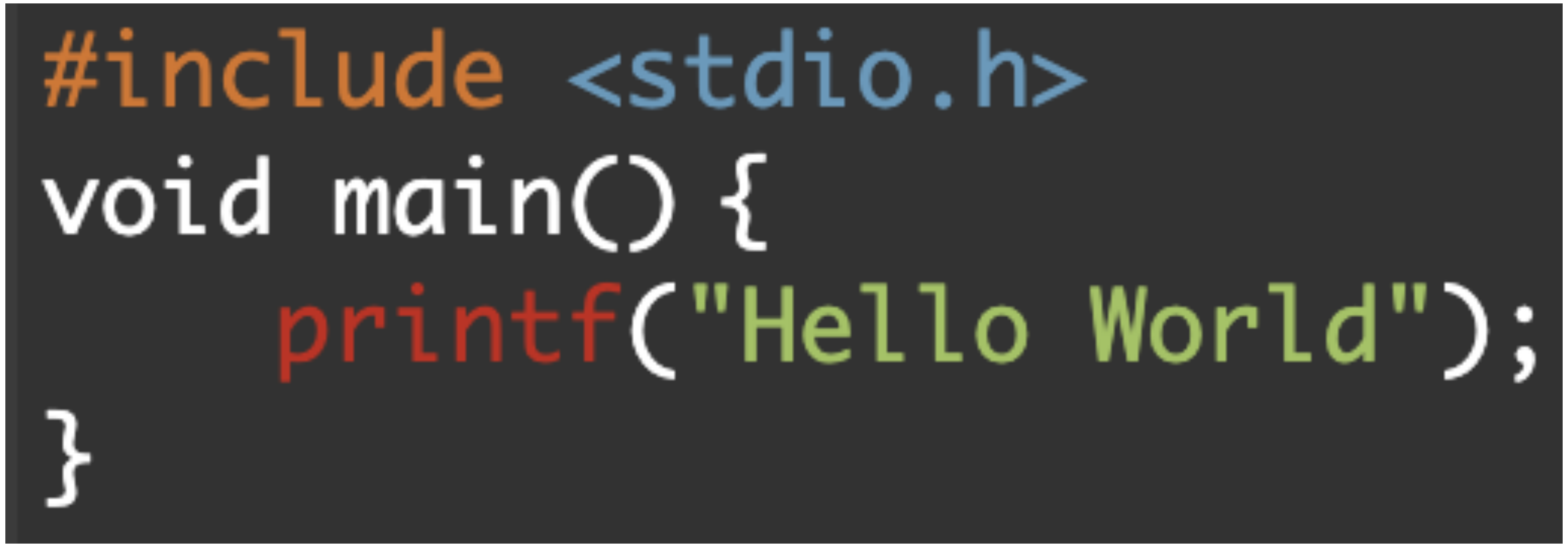
```
PrintChar:
        subi          sp, sp, 8
        stw           r3, 4(sp)
        stw           r4, 0(sp)
        movia         r3, JTAG_UART_BASE
pc_loop:
        ldwio         r4, STATUS_OFFSET(r3)
        andhi         r4, r4, WSPACE_MASK
        beq           r4, r0, pc_loop
        stwio         r2, DATA_OFFSET(r3)
        ldw           r3, 4(sp)
        ldw           r4, 0(sp)
        addi          sp, sp, 8
        ret
```



```
PrintString:
        subi          sp, sp, 12
        stw           ra, 8(sp)
        stw           r3, 4(sp)
        stw           r2, 0(sp)
        mov           r3, r2
ps_loop:
        ldb           r2, 0(r3)
        beq           r2, r0, end_ps_loop
        call          PrintChar
        addi          r3, r3, 1
        br            ps_loop
end_ps_loop:
        ldw           ra, 8(sp)
        ldw           r3, 4(sp)
        ldw           r2, 0(sp)
        addi          sp, sp, 12
        ret

                .org      0x1000
MSG:            .asciz    "Hello World\n"
                .end
```

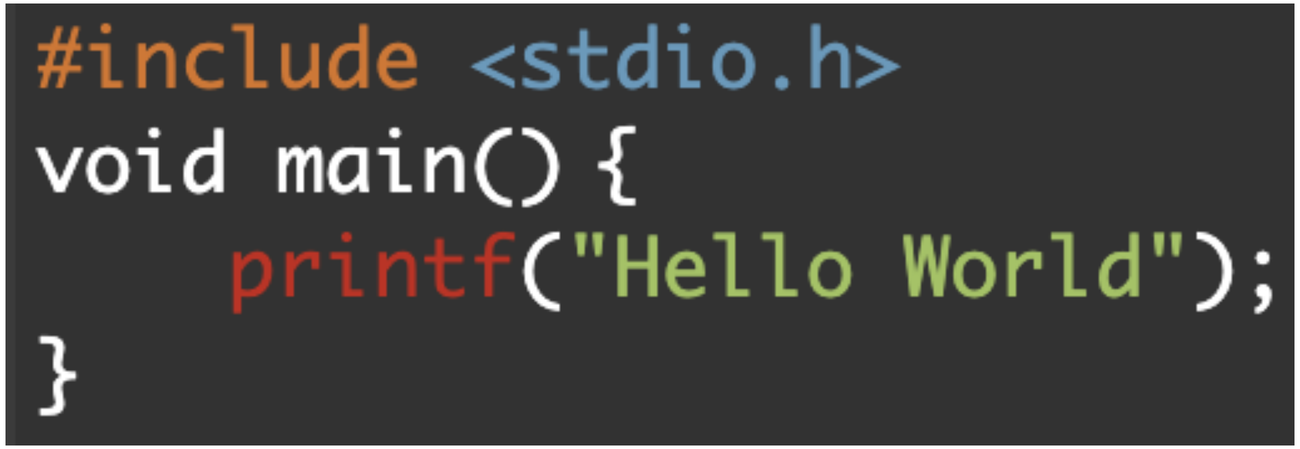
We’d like to use something that is easier to comprehend for most of our programming needs

C



```c
#include <stdio.h>
void main() {
    printf("Hello World");
}
```

AND IT WORKS NEARLY EVERYWHERE!!!



```
#include <stdio.h>
void main() {
    printf("Hello World");
}
```

Everywhere! How?

- The C language isn’t for computers...it’s for people!
- A special program called a compiler takes the C source code and translates it so your processor can understand what it means
- With a compiler you can write the code once and then compile it for a variety of platforms

What’s the catch?

- Can be more challenging than higher-level languages like Python
    - Manually managing memory
    - Runtime errors can be hard to debug (Crashes)