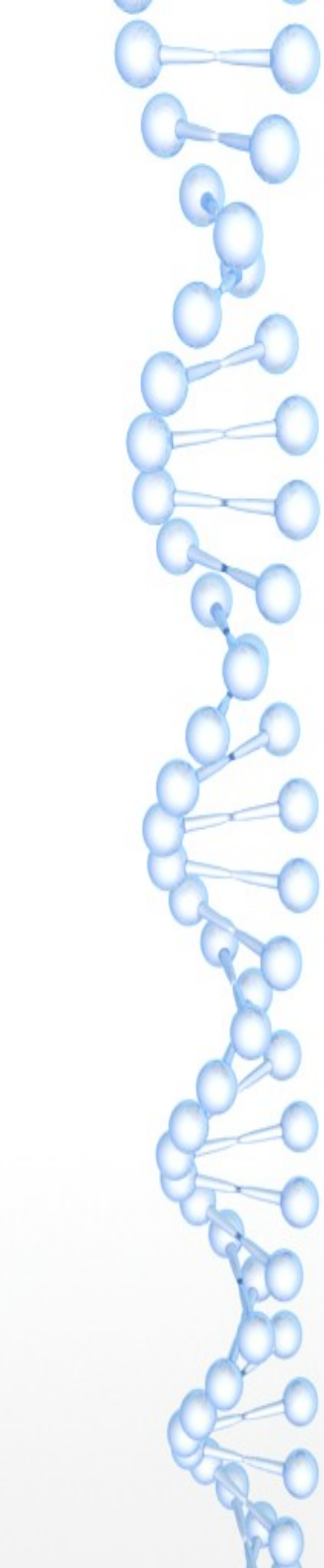
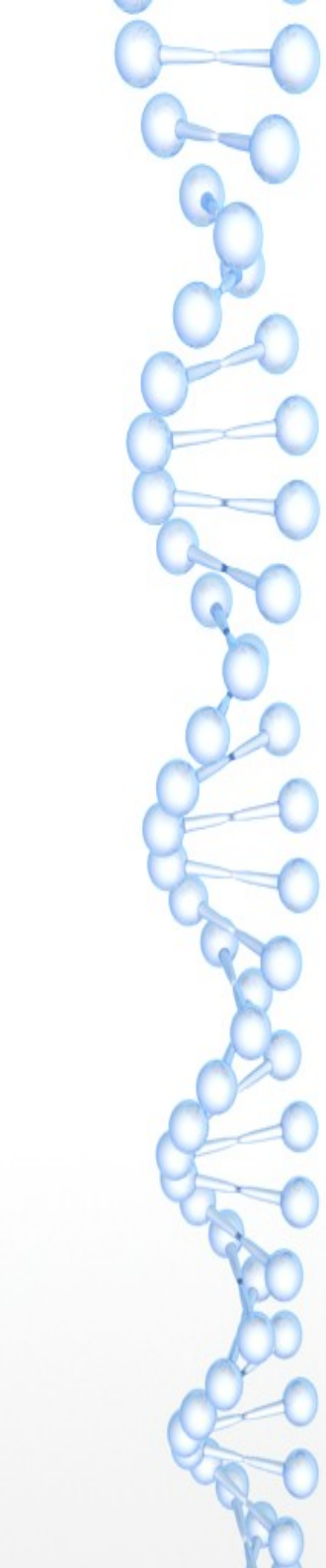


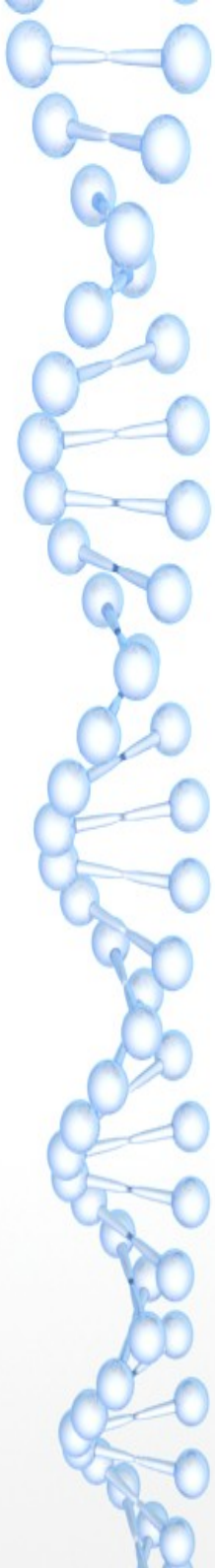
Introduction to Assembly Code

Nels “Chip” Pearson
Pearson Consulting
pc.gameactive.org



Background

- Started coding in assembly in 1977
- Have coded 1, 4, 8, 16, & 32 bit MPUs
- Currently working with AVR and PIC
- Code sensors and platforms in ASM



Outline

- Why program in assembly?
- What makes up a computer?
- How the computer uses data.
- First cut at an example.
- General program structure.
- Assembly coding the example with AVR.
- Extending the example.

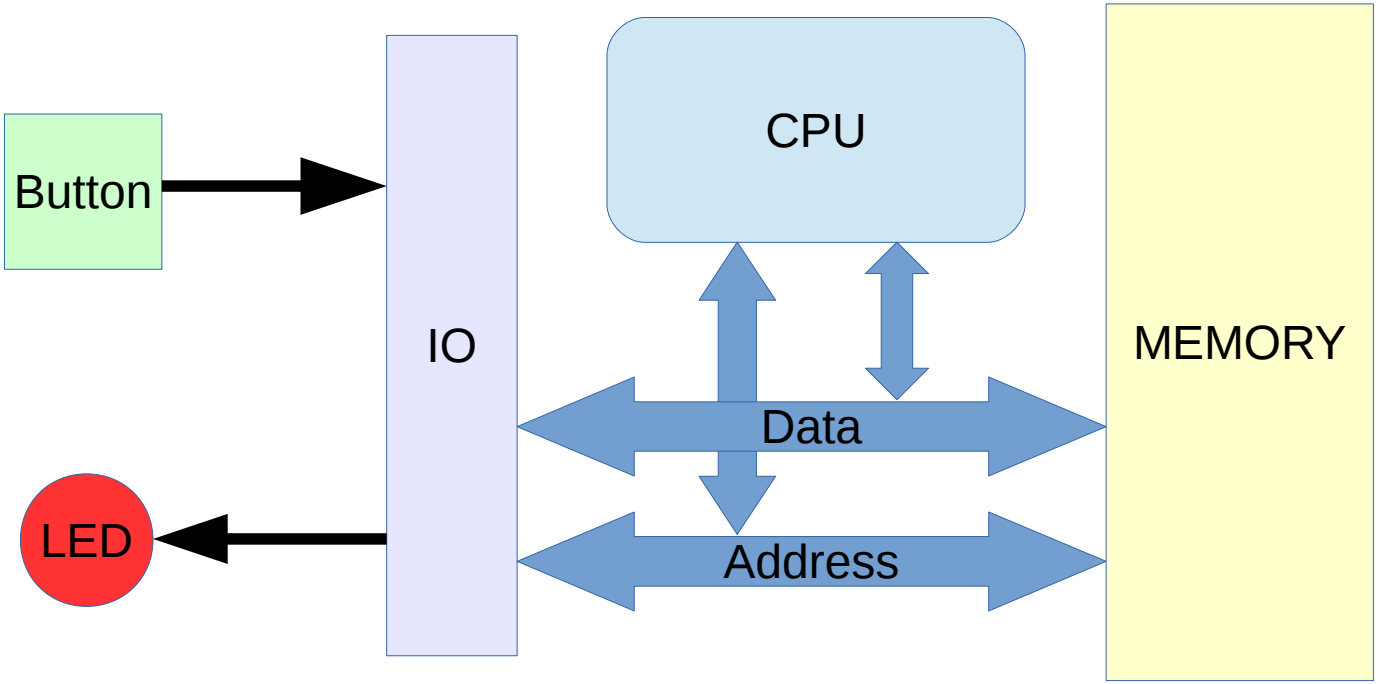
Why program in assembly?

- Mimic hardware
  - Digital logic operations
  - Digital registers
- Manipulate or test data bits
  - Shifts
  - Rotate
  - Test individual or groups of bits

What makes up a computer?

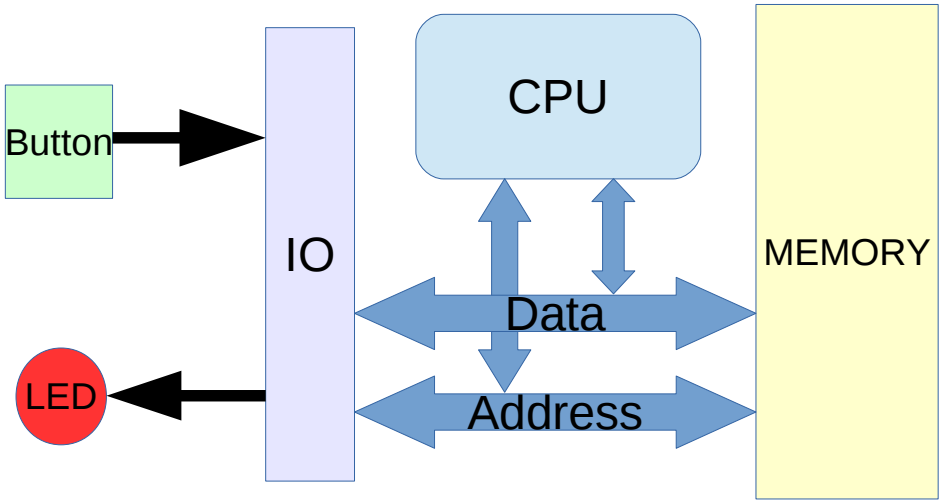
- CPU (Processing)
- + Memory (Storage)
- + Input/Output (Interface to outside world)
  - IO ↔ CPU ↔ Memory
- Terminology:
  - REGISTER (CPU or IO)
  - ADDRESS LOCATION (Memory or IO)
    - How to find data
    - How to store data

How the computer uses data.



- Move data from Memory/IO to CPU to Memory/IO
- May modify the data as it passes through the CPU
- CPU can also generate data

First cut at an example.

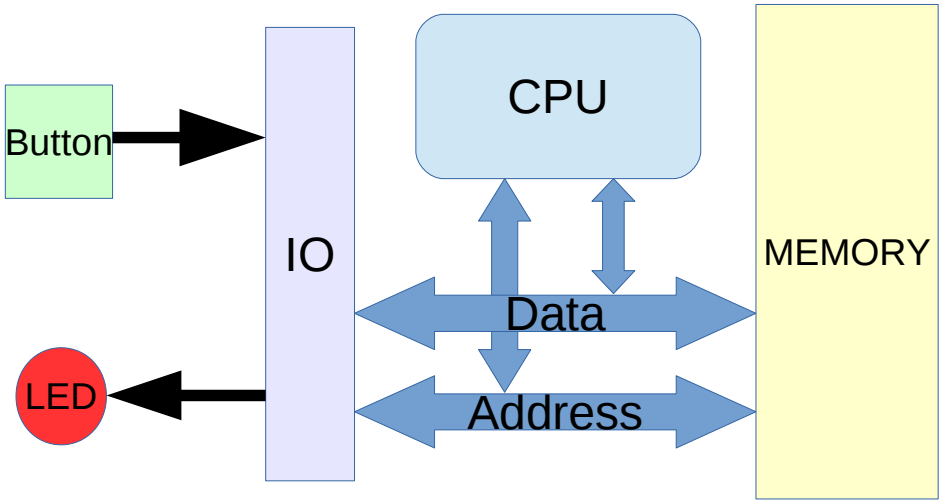


- Read the data at the address of the Button line
- Write that data to the address of the LED line
- Repeat the operation

General program structure.

- Linear sequence of instructions
  - Walk through the instructions
  - Stop at the end
- Linear plus Branch
  - Walk through the instructions
  - Decide whether or not to do the next one or jump to another instruction in the list (JUMP)
  - Possibly go to another list, do those, and return to do the next instruction (CALL)

Assembly coding with AVR.

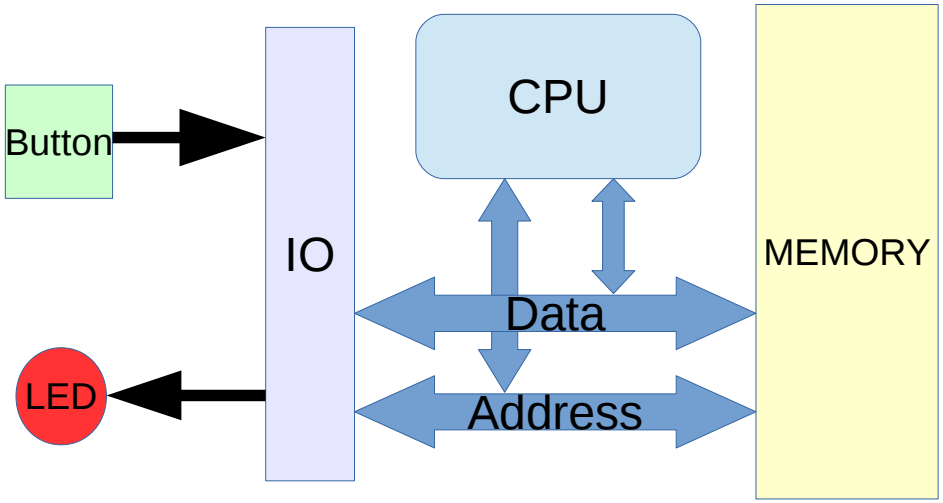



```
    rjmp    main
; Interrupt Vector table
main:
    mov     r16, PORTC          ; read button
    mov     PORTD,r16           ; copy to LED
    rjmp    main
```

Actual Code

```
    rjmp    main
; Pre-allocated Interrupt Vector table
main:
    ser     r16             ; set all bits = 1
    out     DDRD, r16       ; set PORTD lines
                            ; as OUTPUTs
main_loop:
    in      r16, PINC       ; read BUTTON
    out     PORTD,r16       ; copy to LED
    rjmp    main_loop       ; repeat
```

Assembly coding with AVR.

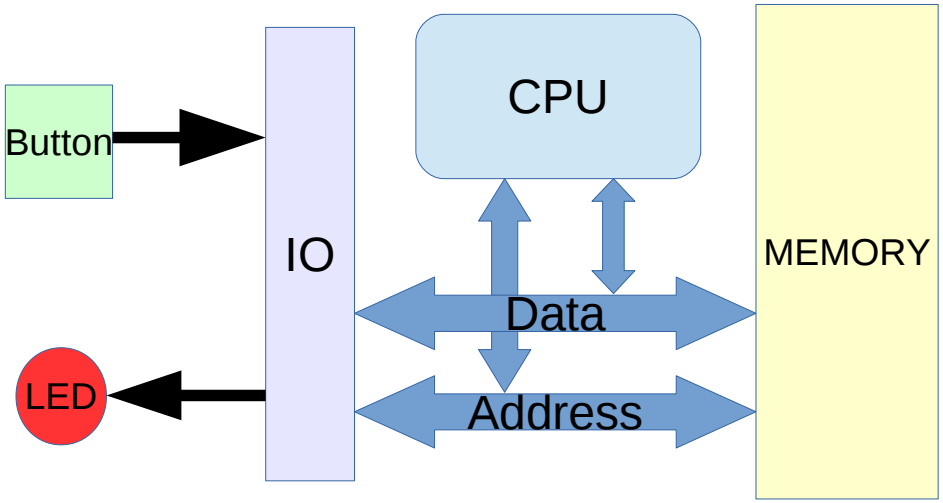



```
    rjmp    main
; Interrupt Vector table
main:
    sbi     PORTD, PORTD0          ; turn ON LED
    rjmp    main
```

Actual Code

```
    rjmp    main
; Pre-allocated Interrupt Vector table
main:
    sbi    DDRD, PORTD0  ; set PORTD0 as OUTPUT
main_loop:
    sbi    PORTD, PORTD0 ; turn ON LED that's
                         ; on pin PD0
    rjmp   main_loop     ; repeat
```

Extending the example.




```
    rjmp    main
; Interrupt Vector table
main:
    sbi     PORTD, PORTD0          ; turn ON LED
    cbi     PORTD, PORTD0          ; turn OFF LED
    rjmp    main
```

Actual Code

```
    rjmp    main
; Pre-allocated Interrupt Vector table
main:
    sbi    DDRD, PORTD0  ; set PORTD0 as OUTPUT
main_loop:
    sbi    PORTD, PORTD0 ; turn ON LED that's
                         ; on pin PD0
    cbi    PORTD, PORTD0 ; turn OFF LED that's
                         ; on pin PD0
    rjmp   main_loop     ; repeat
```

Extending the example.

```
    rjmp    main
; Interrupt Vector table
main:
    sbic    PORTC, PORTC3           ; check switch
    rjmp    skip00                  ; not pressed
    sbi     PORTD, PORTD0           ; turn ON LED
    rjmp    main_end
skip00:
    cbi     PORTD, PORTD0           ; turn OFF LED
main_end:                           ; single exit
    rjmp    main
```

Actual Code

```
    rjmp    main
; Pre-allocated Interrupt Vector table
main:
    sbi     DDRD, PORTD0            ; set PORTD0 as
OUTPUT
main_loop:
    sbic    PORTC, PORTC3           ; check switch
on pin PC3
    rjmp    skip00                  ; not pressed
    sbi     PORTD, PORTD0           ; turn ON LED on
pin PD0
    rjmp    main_end
skip00:
```

What else is there?

- Logic
  - AND, OR, XOR, NOT
- Number systems for data
  - Binary, Hex
- Data formats for groups of bits
  - Binary Coded Decimal (BCD), Floating point
  - ASCII, Character Strings
  - Images, Audio
- Other data structures
  - Arrays, tables, databases

More Information

- Assembly Language Step by Step, Jef Dunteman, Wiley, 2009
- Search: Assembly programming
- Atmel Studio 6.2 Help
  - Rev 4.19 is much more friendly for ASM coding
- https://github.com/CmdrZin/chip_avr_asm_lib/
  - Lots of examples