Section Outline

- x86
- C → x86 Exercise
- Debugging w/ GDB
- HW 1 Questions
x86

- x86 is a family of ISAs based on the architecture of the Intel 8086 CPU
- Provides abstractions for programmers
  - Instructions
  - CPU register access
- Accumulator styled ISA
  - addl %eax, %edx  # EDX += EAX
x86 - Registers

• 8 addressable registers
  ● eax – gen. purpose register also used for function return values
  ● ebx – gen. purpose register also used to hold array/string base address
  ● ecx – gen. purpose register also used for counting
  ● edx – gen. purpose register also used for array data
  ● edi – gen. purpose register also used for src array/string index
  ● esi – gen. Purpose register also used for dest array/string index
  ● esp – contains stack pointer... more later
  ● ebp – contains frame pointer... more later

• Inaccessible registers, managed via instructions
  ● eip – instruction pointer
  ● flags – set for performing value comparison
  ● cs, ds, es, ss – Segment registers for memory addressing
x86 Basics – Register Structure

- Can access different bit ranges of the registers
- Use special names
  - **Ex:** least significant byte of eax is “al”
X86 Basics - Instructions

- **Arithmetic**
  - add, sub, mul, idiv

- **Logical / Bitwise**
  - and, or, xor, neg, sal/shl, sar/shr

- **Control**
  - jmp, je, jne, jg, jl, jle, jge
  - Use after test or cmp instruction
    - test – bitwise AND which sets flags
    - cmp – subtraction which sets flags
  - ret – used to return from a function

- **Other**
  - Stack insns: push, pop
  - Data manipulating: mov, enter, leave
Instructions take a data size specifier as their last character

- **L** – operate on 4 bytes
  - Ex: `addl`, `pushl`, `movl`, `cmpl`
- **B** – operate on least significant byte
  - Ex: `movb`, `cmpb`, `testb`

Need to be combined with appropriately named operands!

- Ex: `addl %edx, %eax` → valid!
  - `cmpb %eax, %cl` → invalid!
C → x86 Exercise

• Implement body of `strcmp()`, a standard C function for comparing two ASCII-encoded strings in x86

• Work in groups of 2 - 4

• C Implementation:

```c
int strcmp(char* a, char* b) {
    while( *a && *b ) {
        if( *a != *b )
            break;
        a++;
        b++;
    }
    return *a - *b;
}
```