Registers

- Only variables available in assembly
- General Purpose Registers:
  - EAX, EBX, ECX, EDX (32 bit)
  - Can be addressed by 8 and 16 bit subsets

Registers (cont.)

- Segment Registers
  - Used to implement segmentation
  - CS – Code Segment
  - DS – Data Segment
  - ES, FS, GS – misc extras
  - SS – Stack Segment

Registers (cont.)

- Index and Pointer Registers
  - EBP – Stack Base
  - ESP – Stack “Top”
  - EIP – Instruction Pointer
  - ESI& EDI
- EFLAGS – holds processor state
  - Bitwise interpretation
Basic Instruction Layout

- Opcode Dest, Src1, Src2
  - ADD %EAX, %EBX == EAX = EAX + EBX
- Operation Suffix indicates operand size:
  - l (long) = 32 bits
    - ex: addl %eax, %ebx
  - w (word) = 16 bits

Basic Instructions

- Simple Instructions:
  - ADD, SUB, MUL, DIV
- Stack Manipulation - PUSH, POP
  - PUSHAL, POPAL – push/pop “big 7” registers at once
  - PUSHF, POPF - push/pop eflags register
- Call a function with CALL
- Return from a function with RET
- Copy a register value with MOV

Addressing Memory

- Address stored in a register: (%eax)
- Address in register + offset: 4(%eax)
- C variable foo becomes: _foo

Inlined Assembly

```c
... // c code
asm ( "assembly code" \
  output registers : \n  input registers : \n  clobbered registers );
```
A Concrete Example

asm volatile("movl %0, %%edx; \
  movl %1, %%ecx; \
  movl %2, %%ebx; \
  movl %3, %%eax; \
  xchg %%bx, %%bx " \
  : /* no output*/ \
  : "g"(addr), "g"(name), \n  "g"(len), "g"(105) \n  : "eax", "ebx", "ecx", "edx");