Assembly Language – Addressing Modes

• Learning Objectives
  • Interpret different modes of addressing in x86 assembly
Accessing Memory

• So far we’ve seen that you can move data to and from memory using expressions that look like:
  • (\%reg)
  • N(\%reg)

• Now it’s time to dig into that in a bit more detail and find out what other ways you can access memory.

• Fundamentally there are three ways to access memory:
  • Via labels (names)
  • Via registers
  • Via constants
Register based addressing

• Recall that simply using $%\text{reg}$ (e.g., $%\text{eax}$) accesses the value stored in the register.

• **Indirect addressing**: Addressing that uses the contents of a register to produce an address:
  • $(%\text{reg})$ treats the value in $\text{reg}$ as an address and refers to the value stored at that address.
    • Example: $(%\text{esp})$ Refers to the value stored on the stack at the location referenced by the stack pointer.
  • $N(\text{reg})$ refers to the data at the address produced by adding $N$ to the contents of $\text{reg}$.
    • Example $4(%\text{esp})$ refers to the value stored at the address 4 greater than the stack pointer.
Indirect Addressing Continued

• **Indirect addressing**: Addressing that uses the contents of a register to produce an address:
  - \((%\text{reg1}, %\text{reg2})\) Refers to the value stored at the address formed by adding the contents of \text{reg1} and \text{reg2}.
  - Example: \((%\text{eax}, %\text{edx})\): If \text{eax} contains the starting address of an array of chars, and \text{edx} is the index, this accesses the value at that index.

• **Extended Indirect addressing**:
  - \((%\text{reg1}, %\text{reg2}, S)\) refers to the value stored at the address formed by adding \text{reg2} \times S to \text{reg1}. S must be one of 1, 2, 4, 8 (when missing, 1 is implied).
  - Example: \((%\text{eax}, %\text{edx}, 4)\) Now the example above works on an array of integers.
Complete form of Indirect Addressing

\[ N(\%reg1, \%reg2, S) \]

- Compute address as follows:
  \[ N + \text{reg1} + (\text{reg2} \times S) \]

- Another handy instruction: leal = load effective address: computes an address and places the result in a register.
Other ways to refer to memory

• Via labels/names:
  • If we have a global variable foo, the label “foo” can be used in addressing (from 10.s):
    • movl foo, %ecx
  • Moves the value stored in the address referenced by the label foo into the register eax.
  • (Fun note: it is quite interesting to see what the compiler does with various levels of optimization – you might experiment with 10.s trying different optimization levels as well as different sized arrays.)
  • The assembly also create local labels (.LBB0_1) and uses those to reference addresses:
    • jne .LBB0_1

• Via constants
  • movl 0xbfff1000, %eax
  • Treat 0xbfff1000 as an address, dereference and move it into %eax.