Name: Jared Dyreson
Class: CPSC-240-09; Assembler
Professor: John Overton TR @ 11:30 - 13:20
Date: 02/01/2019

Advertisements

- Pop quiz on 02/05
- We will only be using 64 bit mode

Definition:

- Little Endian: the order of the memory address is order in LSB (Least Significant Byte)
  - Example: Given 0x0000004A, the little Endian version is 0x4A000000

ASM Program Format

```
label:    instruction  operands  ; optional comment
```

Pseudo instructions:
  - Not "real" machine instructions but are used anyways because it is the most convenient pla
ce to put them
  - Link for more information --> https://www.tortall.net/projects/yasm/manual/html/nasm-pseud
op.html

Directives: are commands that are part of the assembler syntax but are not related to the x86 processor instruction set. All assembler directives begin with a period (.)
  - Link --> https://docs.oracle.com/cd/E26502_01/html/E28388/eoiyg.html

Macros: text that is to be replaced with other text
  - #define statement in C/C++
  - "equ" is used to define macros

’;’ -> a literal comma indicates a comment

Sections: different regions in the program to distinguish what each chunk does

  - .text: for code sections (instructions defined here)

  - .data: where data is stored

  - .bss: where uninitialized data is stored (also static vars in C/C++ programs)

    - reserve space with these keywords:
      - resb: allocates 1 byte
      - resw: allocates 2 bytes (word)
      - resd: allocates 4 bytes (double word)
      - resq: allocates 8 bytes (quad word)
      - more information here: https://stackoverflow.com/questions/44860003/how-much-bytes-doe
s-resb-resw-resd-resq-allocates-in-nasm

Registers: things we can put information into
  - We are focused on the 64-bit ones and those are as follows:
    - rax: register a extended
    - rbx: register b extended
    - rcx: register c extended
    - rsi: register source index (source for data copies)
    - rdi: register destination index (destination for data copies)
    - rbp: register base pointer (start of the stack)
    - rsp: current location in stack, growing downwards

    - r{8-15}: put whatever the hell you want into them
    - more information: https://wiki.cdot.senecacollege.ca/wiki/X86_64_Register_and_Instruction_Quick_Start

Getting into more instructions

- mov: the process of moving data from one region of memory into another
  - notation: mov <dest> <src>
  - if the destination is anything lower than the source register, then whatever preceding the start of the destination register will be ignore (as it would not fit in there anyways)
  - example: mov eax, dword[myVar]
    - dword is essentially a type cast to tell the compiler how much memory is needed for a given variable
  - src and dest cannot both be memory locations
    - you need a two step process for moving a variable into another variable
  - if you want to access memory locations, you need to specify '[]'

- lea: load the address of a variable (a pointer)
  - example
    - lea   rax,[varName]
    - mov   eax,dword[rax]
  - load the memory address of varName, then move it's memory address into the eax register

Narrowing Conversions

- example: going from a dword to a word
- put the data in a lower memory section of the register
- example code:
  - mov   rax, dword[dVal]
  - mov   byte[bVal], al

Widening Conversions: Unsigned

- difference: signed and unsigned
  - unsigned: make sure the higher order bits are zero (would result in an always positive state)
    - example code:
      - mov   rbx,0 ; clear rbx
      - mov   bl, byte[bVar] ; load byte value that you want to widen
      - mov   dword[dVar], rbx ; from rbx to dVar variable with type dword

  - signed: extend the sign
    - if zero (+ number), all upper bit positions will then be set to zero as well
    - if one (- negative), all upper bit positions will then be set to one as well
  - we have registers that do this, found on pages 18 and 19 of the powerpoint