CSE 127 Week 2 Discussion

PA1: GDB + x86

- Friday, October 7th at 11:59 pm
- Group submissions - Groups of 2
- Goal is to prepare you for the next assignment

Virtual Machine

- Weird stack trace on startup and system doesn't start
  - In advanced boot options, try booting using sysvinit or switch to an older kernel
- VirtualBox throws an error on startup
  - This varies, but on windows it is most likely because you haven't enabled Hyper-V, which there are resources to do here.
- To make SSH work within WSL or other OS you can enable port-forwarding.
- Use QEMU for M1 macs.
- SSH is not required, but you need a way to transfer your solution out of the VM.

GDB

- GNU Debugger
- Allows you to "see" inside your program
  - See registers, memory access, instructions
  - Breakpoints allow you to pause execution at any point

GDB Demo

GDB

- Layout next → shows the code
- b main → add breakpoint
- info frame → print info about the current stack frame
- info registers → registers in both hex and natural format
- x/[count][format][unit] → defaults: count = 1, format = x, unit = w
- x/10x $ebp+4 → show as hex
- x/10i $eip → show as instructions
- x/5c name → show as char
- More uses of x here

GDB

- disass main → disassemble a function
- tui enable → enable text user interface (gdb -tui)
- layout src/asm → show source code/assembly
- tui reg general → show registers
- print i → prints variable value
- print i=10 → sets variable value
- set $ebp = 123 → set value for a register
- set {int}0xfff12345 = 123 → set value for a memory region

More resources here

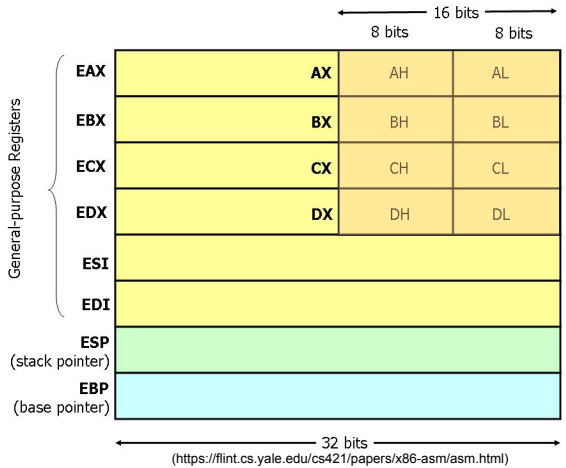
echo

- Write a simplified version of the echo utility using the example code provided
- Use only raw x86 assembly code
- Hints:
  - Strings are terminated by a null byte (a null byte has value 0x0)
  - You might need to write a loop
  - You can make more than one system call
  - You can append a `-g` flag to the `ASFLAGS` in `Makefile` to get debugging information generated, but you need to make sure your program also work without the flag

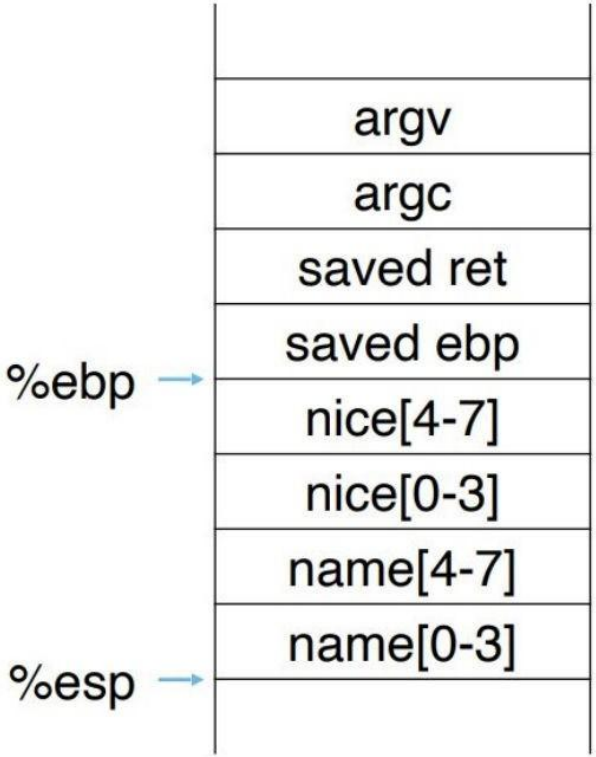
x86 Registers

x86 Registers

- %esp, or the Stack Pointer
  - Designates the top of the stack
  - Grows from high to low memory addresses
- %ebp, or the Frame Pointer/Base Pointer
  - Points to middle of stack frame(to the saved base pointer)
  - Doesn't move as function calls are made



(https://flint.cs.yale.edu/cs421/papers/x86-asm/asm.html)



x86 Registers

- %eip, or the Instruction Pointer
  - Holds the address of the next instruction to be executed

%eip →

```
pushl   %ebp
movl    %esp, %ebp
subl    $40, %esp
movl    16(%ebp), %eax
movl    %eax, -28(%ebp)
movl    %gs:20, %eax
movl    %eax, -12(%ebp)
xorl    %eax, %eax
```

x86 Registers

- inc %eax → eax
- inc (%eax) → *eax
- inc 4(%eax) → *(eax + 4)
- inc 4(%eax, %ebx, 2) → *(eax + 4 + %ebx * 2)

https://patshaughnessy.net/2017/1/20/pointers-in-c-and-x86-assembly-language

x86 Instructions

- movl
- cmpb
- je, jne, jmp
- add, sub, inc, dec
- int 0x80 - ?

x86 Instructions

- Byte (B)
  - 8-bits
- Word (W)
  - 16-bits = 2 bytes
- Double word (L)
  - 32-bits = 4 bytes
- Quad word (Q)
  - 64-bits = 8 bytes

SSH FS with VS Code

get syntax highlighting, File system GUI and terminal with VSCode