howto.asm

Andrew Zonenberg / Alex Radocea
February 18, 2010

Compilation process

- Compile: C -> asm
- Assembly: asm -> object
- Link: object + libraries -> executable

demo

What is assembly?

- 1:1 mapping of instructions to machine code *(almost)*
- Human-readable mnemonics for native machine language

Overview of x86 registers

- Instruction pointer (eip)
- Stack pointer (esp)
- Base pointer (ebp)
- Four general purpose

General purpose registers

```
|---------------e?x---------------|
                |------?x------|
                |--?h--|--?l--|
```

eax, ax, ah, al
ebx, bx, bh, bl
ecx, cx, ch, cl
edx, dx, dh, dl

Basic instruction syntax (Intel)

```
blah dest, arg1, arg2 ...
```

(We will not be covering AT&T syntax in this class)

Intro to NASM

```
bits 32
section .text

global main
main:
  mov eax, 42
  ret
```

Compile: nasm -f elf file.asm
Link: gcc -m32 file.o

Using GDB with assembly

```
gdb ./a.out
run

set disassembly-flavor intel
disassemble main
```

Helpful GDB commands

break *abc* - sets breakpoint at *abc*
disassemble *xyz* - disassembles function *xyz*,
    if no argument is specified start at eip
print $*abc* - prints register *abc*
quit - obvious
run - runs to the next breakpoint
si - step one instruction
x *expr* - dump memory at address *expr*

Common x86 instructions

```
add dest, src => dest += src
call dest     => dest()
jmp dest      => goto dest
lea dest, src => dest  = &src
mov dest, src => dest  = src
sub dest, src => dest -= src
xor dest, src => dest ^= src
```

Pointers in x86

```
mov eax, [ebx]
          eax = *ebx
mov eax, [ebx+4]
          eax = *(ebx+4)
mov eax, [ebx + ecx]
          eax = *(ebx+ecx)
mov eax, [ebx + 2*ecx + 4]
          ...
```

C standard calling convention

(for 32 bit x86)

Return value in eax

Arguments on the stack, pushed last-to-first

Stack layout

Higher addresses

	arg3
	arg2
	arg1
ebp ->	return address
	saved ebp
	local1
esp ->	local2

Lower addresses

Flags

cmp a, b - compares A and B

je - jump if equal
jg/jge - jump if >, if >=
jl/jle - jump if < , if <=

jn* - jump if condition is false (jne, etc)

More flags

jc - jump if last operation overflowed (carry out)
jz - jump if result of last operation was zero

Print-string demo

```
bits 32
section .text

extern puts

mystring: db "hello world", 0

global main
main:
    mov     edi, mystring
    push    edi
    call    puts
    add     esp, 4


    xor     eax, eax
    ret
```