

(S\$\$)

ixia



Hexcellents

Session 2 Assembly Language

Security Summer School

ACS/Ixia/Hexcellents

Outline

- Instruction Set Architecture
- Hello (Assembly) World
- Assembly Basics
- Data Transfer
- Control Flow
- Arithmetic/Logic
- Function Calls
- System Calls
- Compiler Patterns
- Resources

Instruction Set Architecture

- Microprocessor Operations:
 - logical
 - arithmetic
 - control
 - input/output (I/O)
- Structure of an x86 instruction:

```
NASM syntax: add dword [0xdeadbeef], 42
hex: 8 3 0 5 e f b e a d d e 2 a
binary: [1000 0011][0000 0101][1110 1111 1011 1110 1010 1101 1101 1110][0010 1010]
        |           |           |                                           \- immediate: 42
        |           |           |                                           \- memory address: 0xdeadbeef (note the endianness)
        |           |           |
        |           |           | \- opcode modifiers:
        |           |           |   2 bits = addressing mode
        |           |           |   3 bits = register/opcode modifier
        |           |           |   3 bits = r/m field
        \- opcode: add sign-extended 8-bits immediate to register, or 32-bits memory addr
```

Hello (Assembly) World

- Compiling the Hello World C code

```
gcc -m32 -O0 hello.c -o hello
```

- Dumping the object

```
objdump -M intel -d hello
```

- Compiling the Assembly code

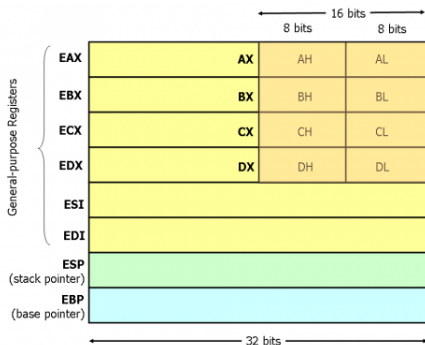
```
nasm -f elf32 hello.asm
```

- Linking the Assembly object

```
ld -s -lc -m elf_i386 -dynamic-linker \
/lib/ld-linux.so.2 -e main hello.o -o h
```

Assembly Basics

- CPU Operation Modes:
 - protected mode
 - real mode
 - virtual 8086 mode
 - long mode
- x86 Registers:



Assembly Basics - Register Usage

- eax: accumulator, used in arithmetic operations
- ebx: base pointer in memory operations (e.g., arrays)
- ecx: loop counters
- edx: also used in arithmetic operations
- esi: source addresses in memory operations
- edi: destination addresses in memory operations
- ebp: frame base pointer
- esp: stack pointer

Assembly Basics - Addressing Modes

- Addressing Formula:

$$\left\{ \begin{array}{l} CS: \\ DS: \\ SS: \\ ES: \\ FS: \\ GS: \end{array} \right\} \left[\begin{array}{l} EAX \\ EBX \\ ECX \\ EDX \\ ESP \\ EBP \\ ESI \\ EDI \end{array} \right] + \left[\begin{array}{l} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESI \\ EDI \end{array} \right] * \left\{ \begin{array}{l} 1 \\ 2 \\ 4 \\ 8 \end{array} \right\} + [\text{displacement}]$$

- Addressing Modes:

```
; direct (displacement)
```

```
mov eax, [0xcafebab3]
```

```
; register indirect (base)
```

```
mov eax, [esi]
```

```
; based (base + displacement)
```

```
mov eax, [ebp-8]
```

```
; indexed (index*scale + displacement)
```

```
mov eax, [ebx*4 + 0xdeadbeef]
```

```
; based-indexed w/o scale (base + index + displacement)
```

```
mov eax, [edx + ebx + 12]
```

```
; based-indexed w/ scale (base + index*scale + displacement)
```

```
mov eax, [edx + ebx*4 + 42]
```

Data Transfer

- `mov <dest>, <src>`: move
- `xchg <dest>, <src>`: exchange (swap)
- `movzx <dest>, <src>`: move with zero extend
- `movsx <dest>, <src>`: move with sign extend
- `movsb`: move byte from location pointed to by `esi` to `edi`
- `movsw`: similar, move word (2 bytes)
- `lea <dest>, <src>`: load effective address (calculate address of `<src>` and load it to `<dest>`)

Control Flow

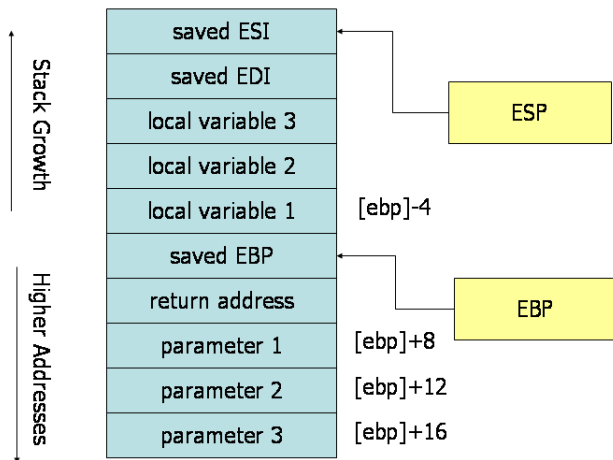
- Control Instructions:
 - `jmp <addr>`: loads `<addr>` into `eip`
 - `call <addr>`: pushes current `eip` on stack, and loads `<addr>` into `eip`
 - `ret <val>`: loads head of stack into `eip`, and pops `<val>` bytes off the stack
 - `loop <addr>`: decrements `ecx`, and jumps to `<addr>` if `ecx != 0`
- Conditional Jump Flags:
 - ZF (zero flag): previous arithmetic operation resulted in zero
 - SF (sign flag): previous result's most significant bit
 - CF (carry flag): previous result requires a carry
 - OF (overflow flag): previous result overflows the maximum value that fits a register

Arithmetic/Logic

- Arithmetic Instructions:
 - add $\langle \text{dest} \rangle, \langle \text{src} \rangle$: addition
 - sub $\langle \text{dest} \rangle, \langle \text{src} \rangle$: subtraction
 - mul $\langle \text{arg} \rangle$: multiplication with corresponding byte-wise eax (i.e., $\langle \text{arg} \rangle = \text{"dh"} \ ? \ \text{dh} * \text{ah}$)
 - imul $\langle \text{arg} \rangle$: signed multiplication
 - imul $\langle \text{dest} \rangle, \langle \text{src} \rangle$: signed multiplication ($\text{dest} = \text{dest} * \text{src}$)
 - imul $\langle \text{dest} \rangle, \langle \text{src} \rangle, \langle \text{aux} \rangle$: signed multiplication ($\text{dest} = \text{src} * \text{aux}$)
 - div $\langle \text{arg} \rangle$: division
 - idiv $\langle \text{arg} \rangle$: signed division
 - neg $\langle \text{arg} \rangle$: 2's complement negation
- Shifts and Rotations:
 - shr, shl (logical shift right/left)
 - sar, sal (arithmetic shift right/left)
 - shld, shrd (double-shift)
 - ror, rol (rotate)
 - rcr, rcl (rotate with carry)
- Logical Instructions:
 - and, or, xor, not

Function Calls

- Calling conventions:
 - cdecl, stdcall, fastcall, thiscall
- Call Stack:



System Calls

- Syscalls are the interface that allows user applications to request services from the OS kernel
- The mechanism is invoked by triggering an interrupt (int 0x80)
- The conventions for invoking a syscall on Linux are:
 - eax contains the syscall ID
 - parameters are passed in ebx, ecx, edx, esi, edi, ebp (in this order)
 - the syscall is responsible of saving and restoring all registers

Compiler Patterns

- function prologue
- function epilogue
- for loop
- while loop
- nested fors with break and continue

Resources

- 1 ref.x86asm.net/index.html
- 2 intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html
- 3 net.cs.uni-bonn.de/fileadmin/user_upload/plohmann/x86_opcode_structure_and_instruction_overview.pdf
- 4 nasm.us/xdoc/2.11.05/html/nasmdoc0.html
- 5 en.wikipedia.org/wiki/Linux_Standard_Base
- 6 muppetlabs.com/~breadbox/software/tiny/teensy.html
- 7 timelessname.com/elfbin/
- 8 codegolf.stackexchange.com/questions/5696/shortest-elf-for-hello-world-n
- 9 unixwiz.net/techtips/x86-jumps.html
- 10 gcc.gnu.org/onlinedocs/gcc-4.7.2/gcc/Function-Attributes.html
- 11 docs.cs.up.ac.za/programming/asm/derick_tut/syscalls.html
- 12 linuxjournal.com/article/3326
- 13 users.ece.cmu.edu/~sangkilc/papers/ccs10-cha.pdf