CS240 Laboratory 6
Disassembly and Reverse Engineering

Memory Layout

Kernel
above 0x7fffffff

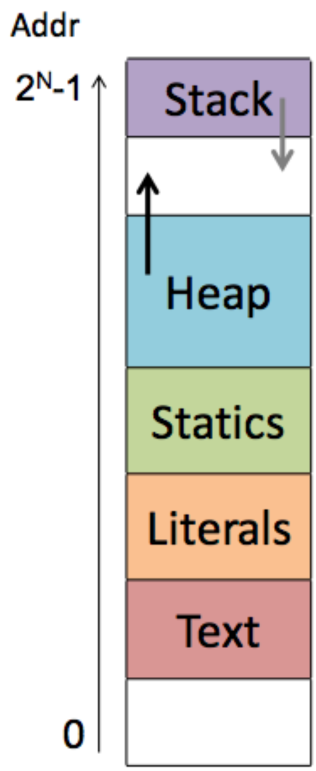
Stack
below Ox7fffffff
grows down

Heap
above Data segment

Data segment
statics and literals

Text segment
starts at 0x400000

addresses below 0x400000 reserved for operating system



Perm	Contents	Managed by	Initialized
RW	Procedure context	Compiler	Run-time
RW	Dynamic data structures	Programmer, malloc/free, new/GC	Run-time
RW	Global variables/ static data structures	Compiler/ Assembler/Linker	Startup
R	String literals	Compiler/ Assembler/Linker	Startup
X	Instructions	Compiler/ Assembler/Linker	Startup

Instructions

The size of the data being referenced is often specified with an additional character:

b (byte)
w (2 bytes)
l (4 bytes), or
q (8 bytes).

Operand Types

Immediate $0x400, $-533

Register: %rax,%rbx,%rcx,%rdx,%rsi,%rdi,%rbp,%rsp,
%r8,%r9,%r10,%r11,%r12,%r13,%r14,%r15

some have special purpose: **%rsp** is stack pointer, **%rax** always used to return value from functions

Memory -0x18(%rsp)

Most General Form:

	D(Rb,Ri,S)	**Mem[Reg[Rb] + S*Reg[Ri] + D]**
D:	Constant "displacement" value represented in 1, 2, or 4 bytes	
Rb:	Base register: Any register	
Ri:	Index register: Any except %esp (or %rsp if 64-bit) ; %ebp unlikely	
S:	Scale: 1, 2, 4, or 8 (*why these numbers?*)	

Special Cases: can use any combination of D, Rb, Ri and S

(Rb,Ri)	**Mem[Reg[Rb]+Reg[Ri]]**	(S=1, D=0)
D(Rb,Ri)	**Mem[Reg[Rb]+Reg[Ri]+D]**	(S=1)
(Rb,Ri,S)	**Mem[Reg[Rb]+S*Reg[Ri]]**	(D=0)

Types of Instruction

Moving Data

movl Src,Dest // copy 4 bytes from source to destination

Arithmetic/Logical operations – 2 operands

add *Src,Dest*
sub *Src,Dest*
imul *Src,Dest*

shr *Src,Dest*
sar *Src, Dest*
shl *Src,Dest*
sal *Src, Dest*
shr *Src,Dest*

xor *Src,Dest*
and *Src,Dest*
or *Src,Dest*

mul *Src,Dest*
imul *Src,Dest*
div *Src,Dest*
idiv *Src,Dest*

Arithmetic/Logical operations – 1 operand

inc Dest
del Dest
neg Dest
not Dest

Setting Condition Codes Explicitly – used for control flow

cmp Src2,Src1	sets flags based on value of Src2 – Src1, discards result
test Src2,Src1	sets flags based on a & b, discards result

Control Flow

Conditional jump instructions in X86 implement the following high-level constructs:

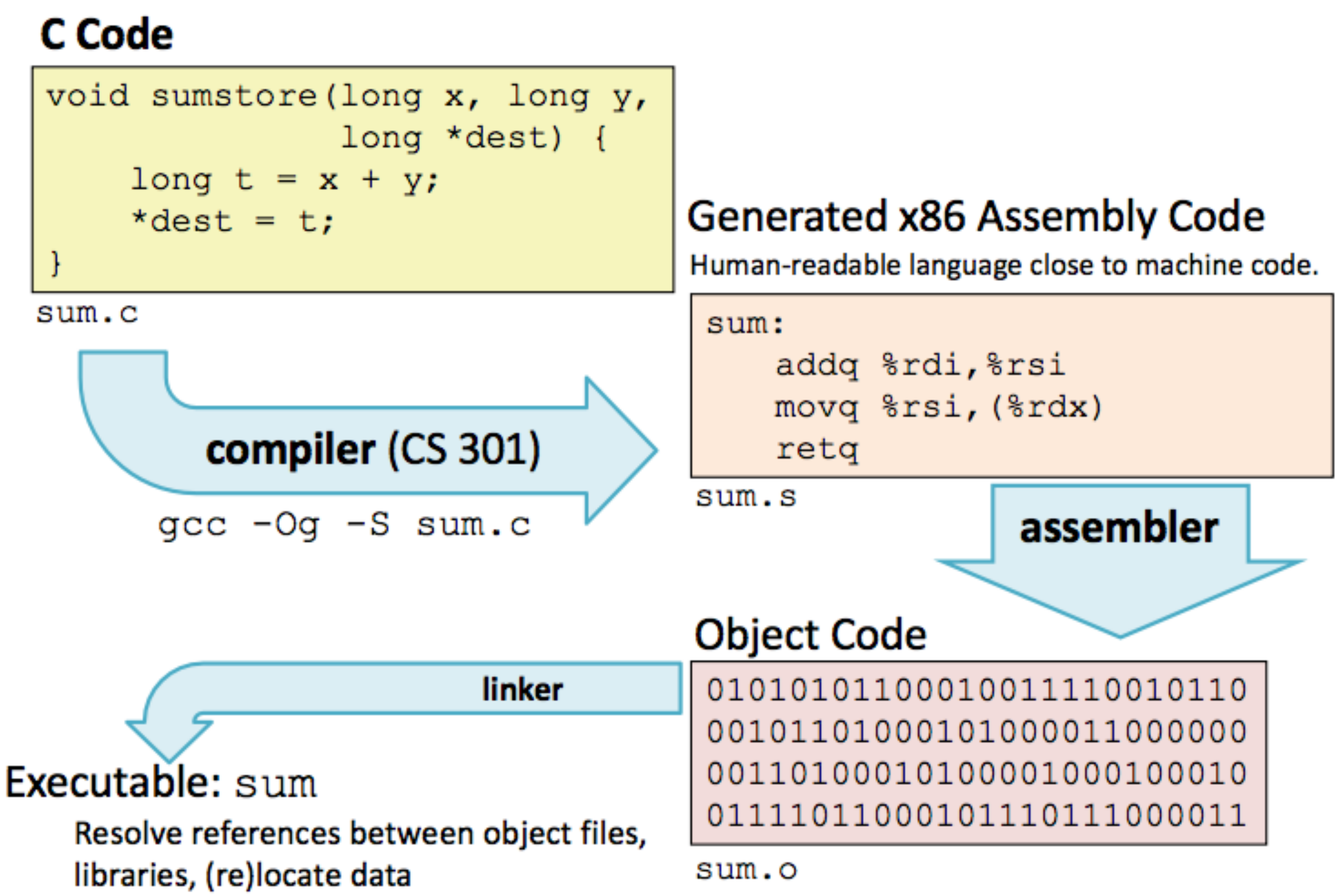
- if (condition) then {...} else {…}
- while (condition) {…}
- do {…} while (condition)
- for (initialization; condition; iterative) {...}

Unconditional jumps are used for high-level constructs such as:

- break
- continue

jX	Condition	Description
jmp	1	Unconditional
je	ZF	Equal / Zero
jne	~ZF	Not Equal / Not Zero
js	SF	Negative
jns	~SF	Nonnegative
jg	~(SF^OF)&~ZF	Greater (Signed)
jge	~(SF^OF)	Greater or Equal (Signed)
jl	(SF^OF)	Less (Signed)
jle	(SF^OF)|ZF	Less or Equal (Signed)
ja	~CF&~ZF	Above (unsigned)
jb	CF	Below (unsigned)

Turning C into Machine Code



- X86 instructions can be in different order from C code
- Some C expressions require multiple X86 instructions
- Some X86 instructions can cover multiple C expressions
- Compiler optimization can do some surprising things!
- Local or temporary variables can be stored in registers or on the stack

Function Calling Conventions

- Arguments for functions are stored in registers, in the following order: arg1 – arg6: %rdi,%rsi,%rdx,%rcx,%r8,%r9
- If there are more than 6 parameters for a function, the rest of the arguments are stored on the stack before the function is called
- Return value from function is always in %rax

The compiler will use only part of a register if the value stored there will fit in less than 64 bits (8 bytes). This is an optimization that makes instructions a bit shorter.

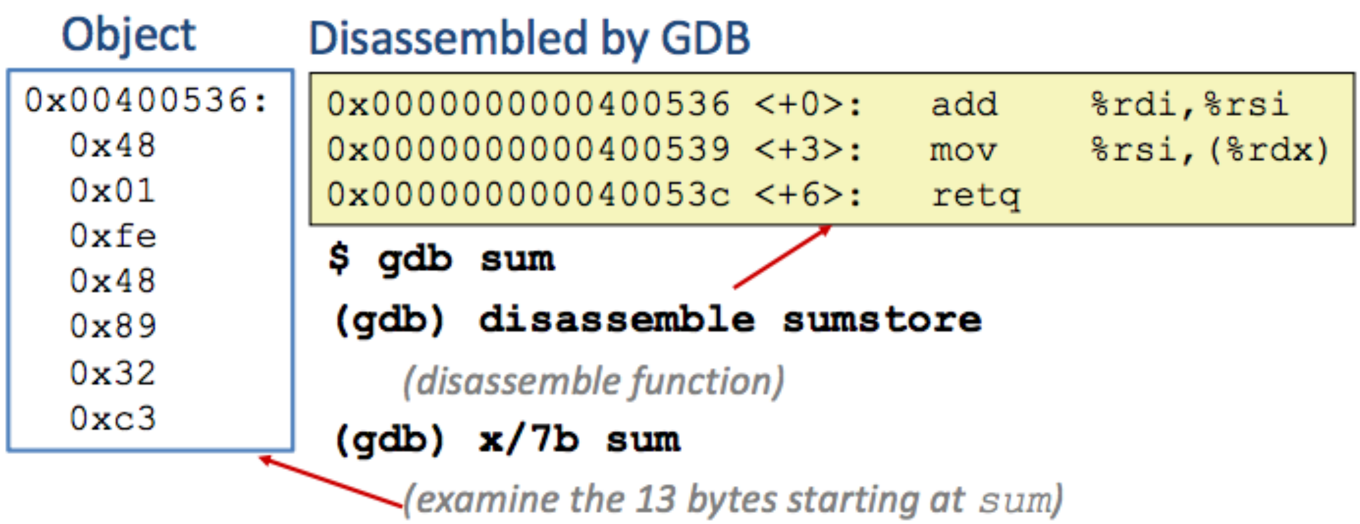
So, in the code, you may see register names of the following form, all of which refer to %rax:

%**r**ax = 8 byte value
%**e**ax = 4 byte value
%ax = 2 byte value
%al = 1 byte value

Tools

Tools can be used to examine bytes of object code (executable program) and reconstruct (reverse engineer) the assembly source.

gdb – disassembles an executable file into the associated assembly language representation, and provides tools for memory and register examination, single step execution, breakpoints, etc.



objdump

can also be used to disassemble and display information

$ *objdump –t p*

Prints out the program's symbol table. The symbol table includes the names of all functions and global variables, the names of all the functions the called, and their addresses.

$ *objdump -d p*

Object Code	**Disassembled version**
0x401040 <sum>:	00401040 <_sum>:
0x55	0: 55 push %ebp
0x89	1: 89 e5 mov %esp,%ebp
0xe5	3: 8b 45 0c mov 0xc(%ebp),%eax
0x8b	6: 03 45 08 add 0x8(%ebp),%eax
0x45	9: 89 ec mov %ebp,%esp
0x0c	b: 5d pop %ebp
0x03	c: c3 ret
0x45	
0x08	
0x89	
0xec	
0x5d	
0xc3	

strings

$ *strings –t x p*

Displays the printable strings in your program.