Machine-Level Programming I: Basics

Intel x86 processors

- dominate laptop/desktop/server market
- evolutionary design
    - backwards compatible up to 8086
- **Complex Instruction Set Computer (CISC)**
    - large complexity in number and types of instructions
    - many different instructions with many different formats
        - but only small subset encountered with Linux programs
    - hard to match performance of **Reduced Instruction Set Computers (RISC)**
    - but, Intel has done that
        - in terms of speed, less so for low power

Architecture (ISA)

- instruction set architecture
- parts of a processor design that one needs to understand or write assembly/machine code
- Ex: instruction set specification, registers

- instruction for processor lives in the text segment of memory until needed

Microarchitecture

- implementation of the architecture
- Ex: cache sizes and core frequency

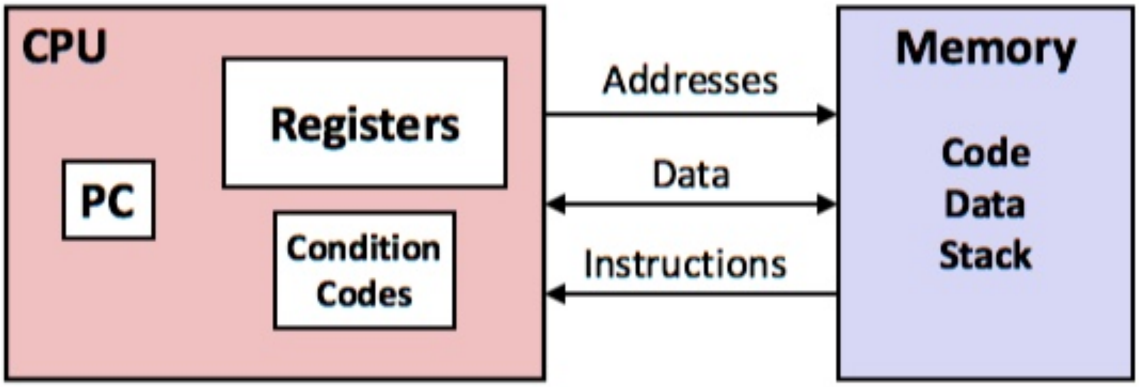
Code Forms:

- **Machine Code**: byte-level programs that a processor executes
- **Assembly Code**: A text representation of machine code

Example ISAs

- Intel: x86, IA32, Iteanium, x86-64
- ARM: Used in almost all mobile phones

Assembly/Machine Code View



PC: Program Counter

- address of next instruction
- called “RIP” (x86-64)

Register File

- heavily used program data

Condition Codes

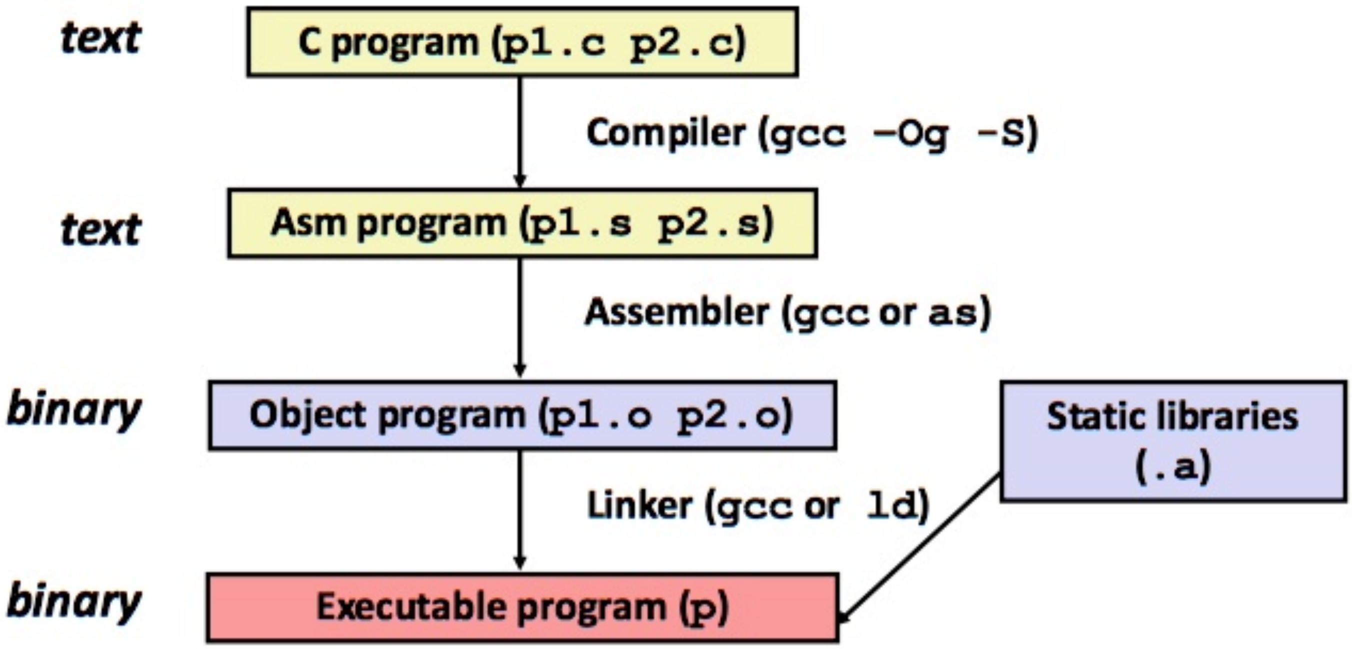
- store status info about most recent arithmetic or logical operation
- used for conditional branching

Memory

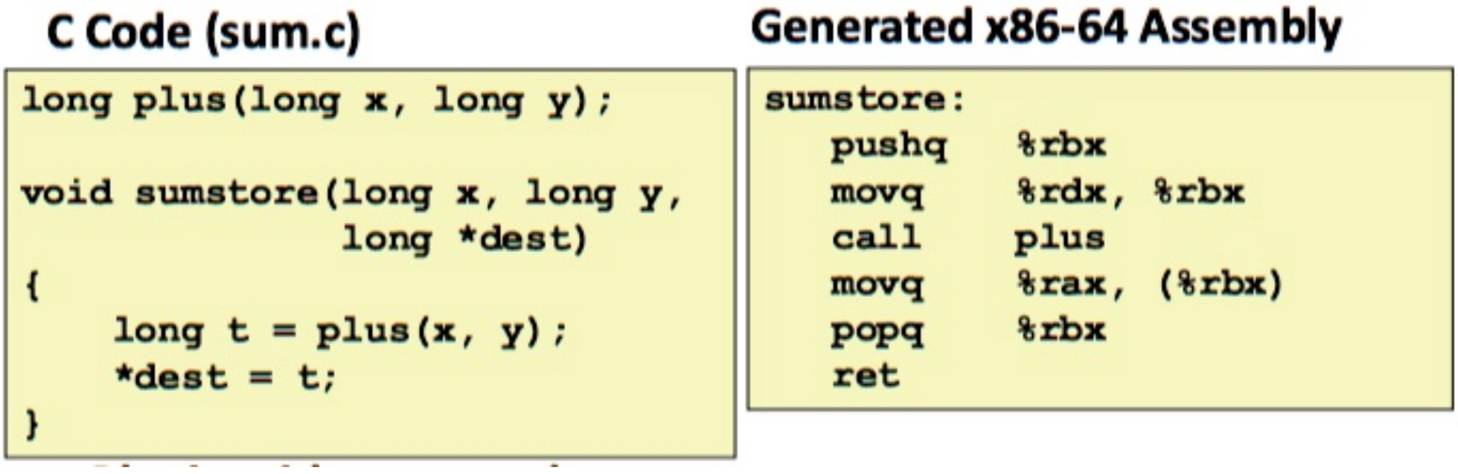
- byte addressable array
- code and user data
- stack to support procedures

Turning C into Object Code

- code in files `p1.c p2.c`
- compile with command:
    - `gcc -Og p1.c p2.c -o p`
    - use basic optimizations (-Og) [new to recent versions of GCC]
    - put resulting binary in file `p`



Compiling Into Assembly



C Code (sum.c)

```
long plus(long x, long y);

void sumstore(long x, long y,
              long *dest)
{
    long t = plus(x, y);
    *dest = t;
}
```

Generated x86-64 Assembly

```
sumstore:
   pushq   %rbx
   movq    %rdx, %rbx
   call    plus
   movq    %rax, (%rbx)
   popq    %rbx
   ret
```

- obtain with command
  - `gcc –Og –S sum.c`
- produces file `sum.s`
- WARNING: Can get very different results on different machines due to different versions of gcc and different compiler settings

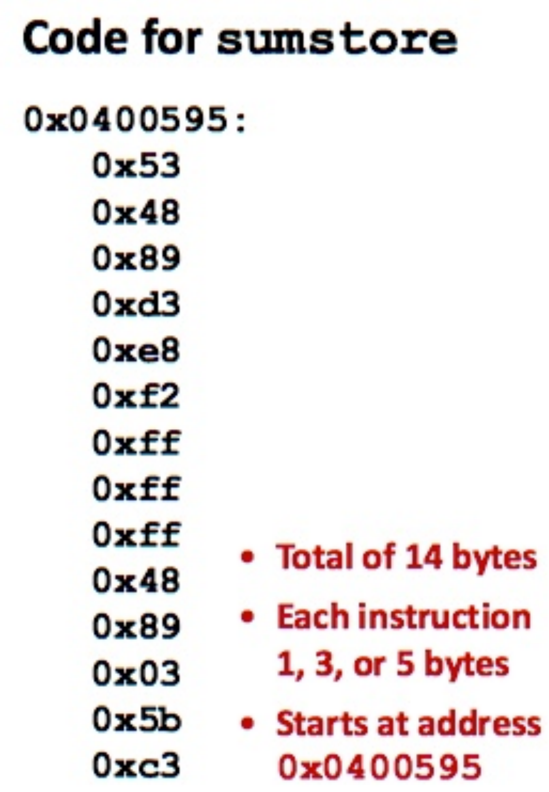
Assembly Characteristics: Data Types

- **integer** data of 1, 2, 4, or 8 bytes
  - data values
  - addresses (untyped pointers)
- **floating point** data of 4, 8, or 10 bytes
- code: byte sequences encoding series of instructions

- no aggregate types such as arrays or structures
    - *just contiguously allocated bytes in memory*

Assembly Characteristics: Operations

- perform arithmetic function on register or memory data
- transfer data between memory and register
    - load data from memory into register
    - store register data into memory
- transfer control
    - uncoditional jumpes to/from procedures
    - conditional brances



Code for sumstore

```
0x0400595:
    0x53
    0x48
    0x89
    0xd3
    0xe8
    0xf2
    0xff
    0xff
    0xff
    0x48
    0x89
    0x03
    0x5b
    0xc3
```

- Total of 14 bytes
- Each instruction 1, 3, or 5 bytes
- Starts at address 0x0400595

Assembler

- translates .s into .o
- binary encoding of each instruction
- nearly-complete image of executable code
- missing linkages between cod ein different files

Linker

- resolves references between files
- combines with static run-time libraries

    - e.g. code for `malloc`, `printf`
- some libraries are *dynamically linked*
    - linking occurs when program begins execution

Machine Instruction Example

C Code

- store value t where designated by `dest`

Assembly

- move 8-byte value to memory
- operands
    - `t`: Register `%rax`
    - `dest`: Register `%rbx`
    - `*dest`: Memory `M[%rbx]`

Object Code

- 3-byte instruction
- stored at address `0x40059e`

```
// Dereferencing a destination
* dest = t;             // code
-- movq %rax, (%rbx)    // assembly
0x40059e: 48 89 03      // binary
```

Disassembling Object Code

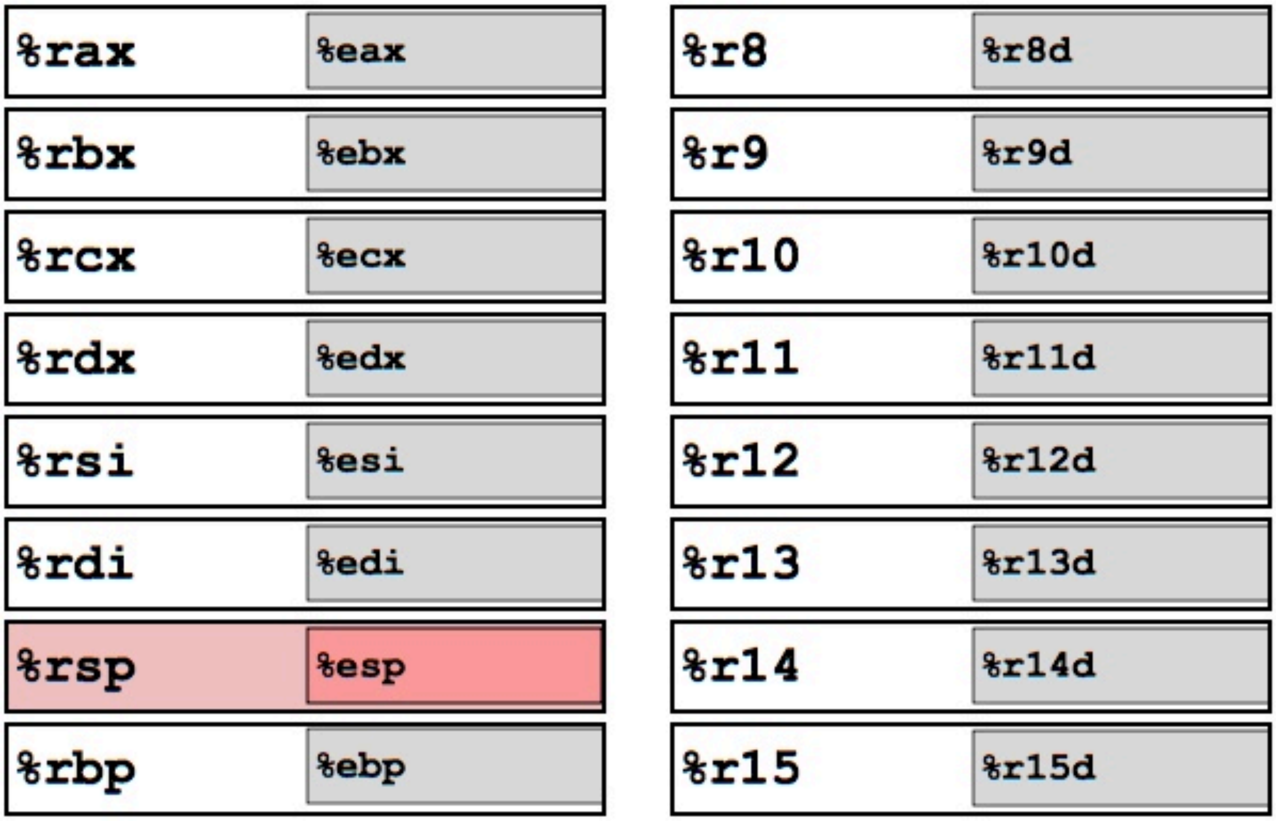
```
0000000000400595 <sumstore>:
400595:  53                 push %rbx
400596:  48 89 d3           mov %rdx, %rbx
400599:  e8 f2 ff ff ff     callq 400590 <plux>
40059e:  48 89 03           mov %rax, (%rbx)
4005a1:  5b                 pop %rbx
4005a2:  c3                 retq
```

Disassembler

`objdump -d sum`

- useful tool for examining object code
- analyzes bit patter of series of instructions
- produces approximate rendition of assembly code
- can be run on either a.out (complete executable) or .o file

x86-64 Integer Registers



%rax	%eax	%r8	%r8d
%rbx	%ebx	%r9	%r9d
%rcx	%ecx	%r10	%r10d
%rdx	%edx	%r11	%r11d
%rsi	%esi	%r12	%r12d
%rdi	%edi	%r13	%r13d
%rsp	%esp	%r14	%r14d
%rbp	%ebp	%r15	%r15d

Moving Data

`movq source, dest:`

Operand Types

- **immediate**: constant integer data
  - Ex: $0x400, $-533
  - like C constant, but prefixed with ‘$’
  - encoded with 1, 2, or 4 bytes
- **register**: one of 16 integer registers
  - Ex: %rax, %r13
  - but %rsp reserved for special use
  - others have special uses for particular instructions
- **memory**: 8 consecutive bytes of memory at address given by register
  - simplest example: (%rax)
  - various other “address modes”