Roadmap

C:

```
car *c = malloc(sizeof(car));
c->miles = 100;
c->gals = 17;
float mpg = get_mpg(c);
free(c);
```

Java:

```
Car c = new Car();
c.setMiles(100);
c.setGals(17);
float mpg =
    c.getMPG();
```

Assembly language:

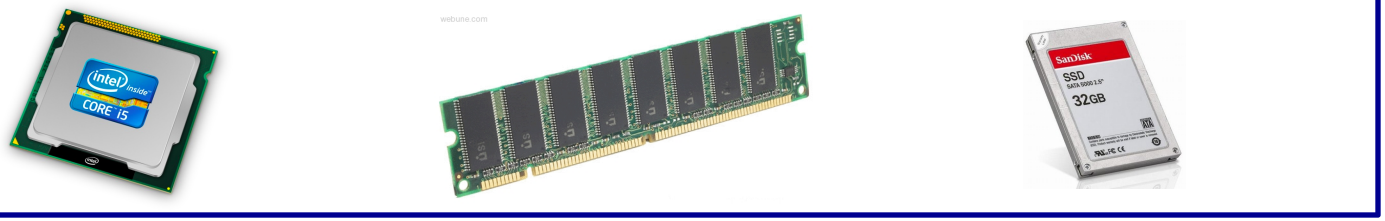
```
get_mpg:
    pushq   %rbp
    movq    %rsp, %rbp
    ...
    popq    %rbp
    ret
```

Machine code:

```
0111010000011000
100011010000010000000010
1000100111000010
11000001111110100001111l
```

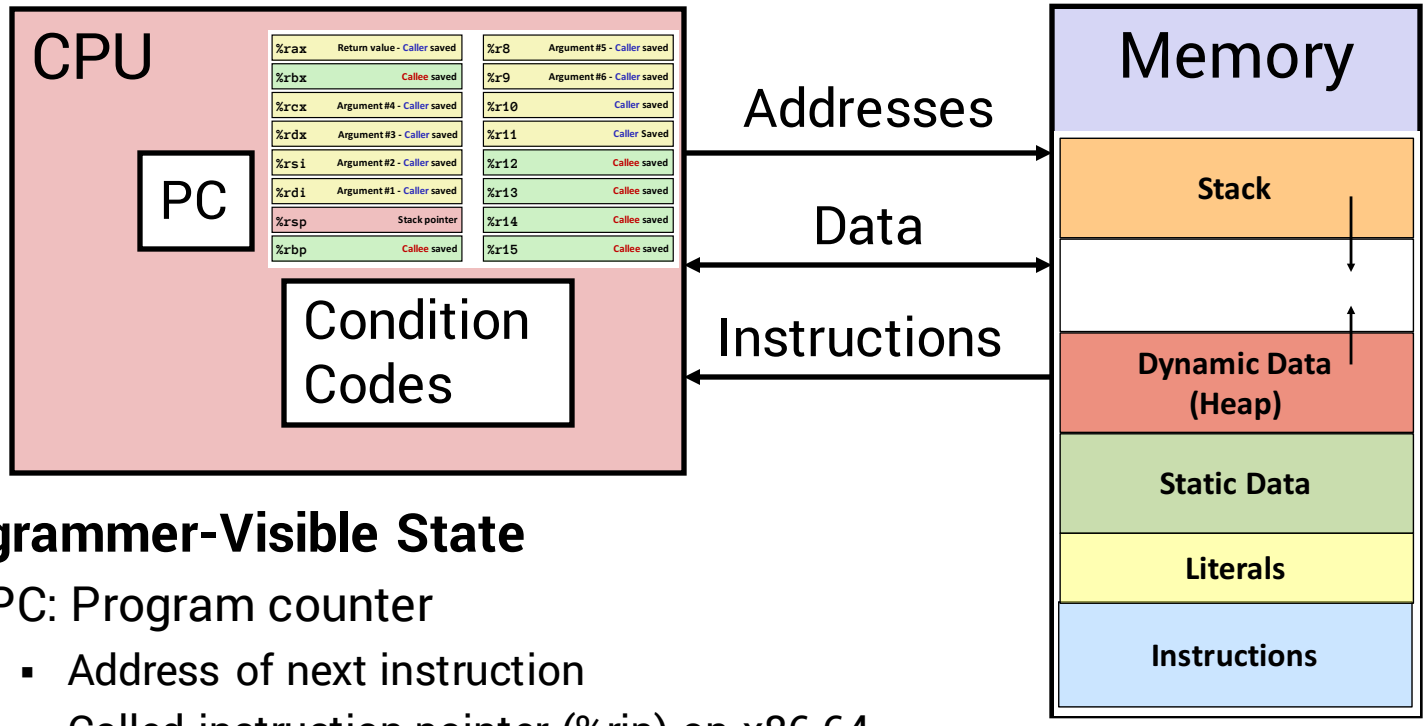
OS:

Computer system:



Memory & data
Integers & floats
Machine code & C
x86 assembly
Procedures & stacks
Arrays & structs
Memory & caches
Processes
Virtual memory
Memory allocation
Java vs. C

Assembly Programmer's View



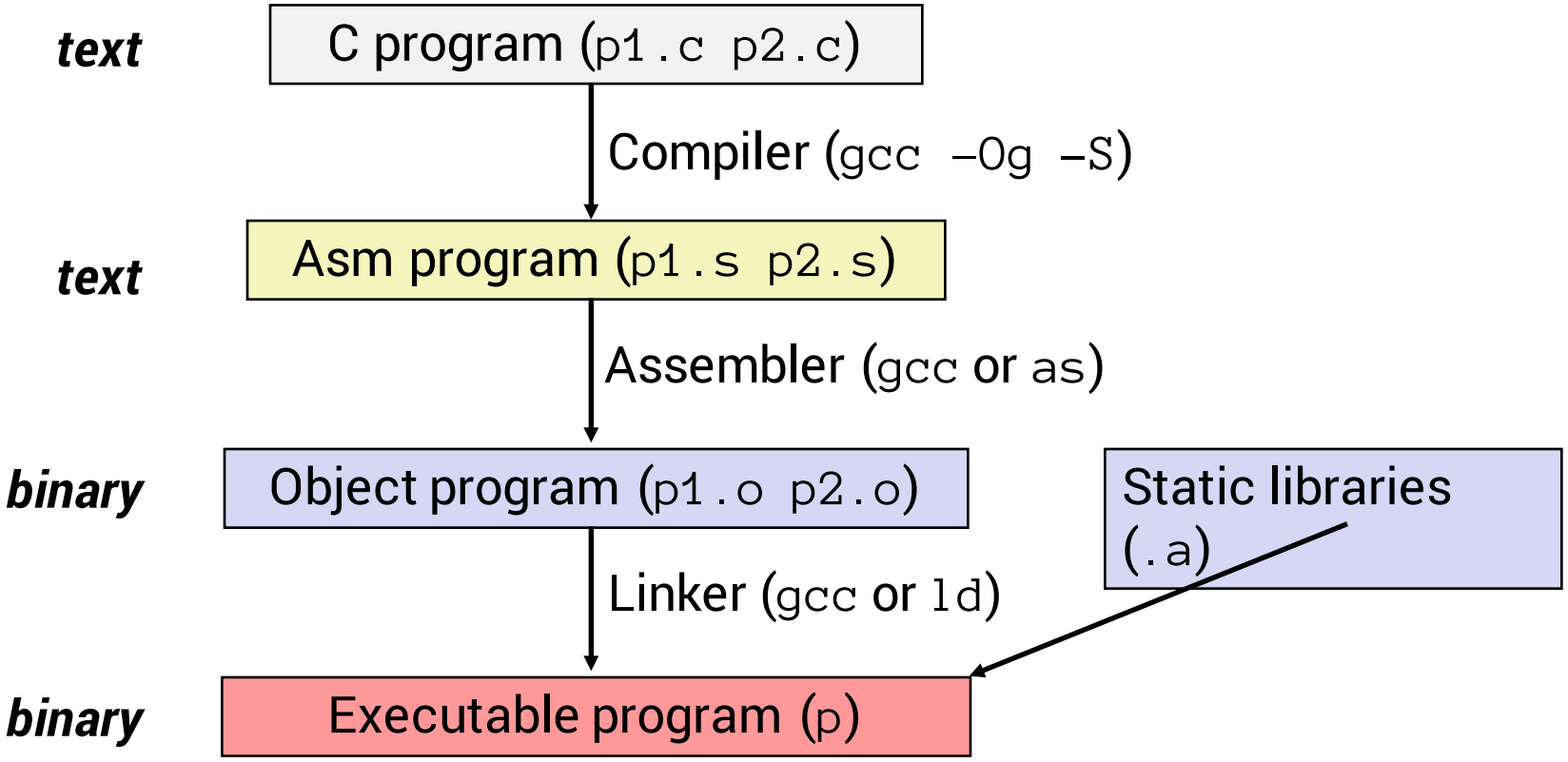
- **Programmer-Visible State**
  - PC: Program counter
    - Address of next instruction
    - Called instruction pointer (%rip) on x86-64
  - Named registers
    - Heavily used program data
    - Together, called "register file"
  - Condition codes
    - Store status information about most recent arithmetic operation
    - Used for conditional branching
- Memory
  - Byte addressable array
  - Code and user data
  - Includes *Stack* (for supporting procedures, we'll come back to that)

x86-64 Instructions

- **Data movement**
  - `mov`, `movs`, `movz`, …
- **Arithmetic**
  - `add`, `sub`, `shl`, `sar`, *`lea`*, …
- **Control flow**
  - `cmp`, `test`, `j_`, `set_`, …
- **Stack/procedures**
  - `push`, `pop`, `call`, `ret`, …

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 machine code in file `p`



Machine Instruction Example

```
*dest = t;
```

```
movq %rsi, (%rdx)
```

```
0x400539:  48 89 32
```

- **C Code**
  - Store value `t` where `dest` points
- **Assembly**
  - Move 8-byte value to memory
    - Quad words in x86-64 parlance
  - Operands:

    `t:` Register `%rsi`

    `dest:` Register `%rdx`

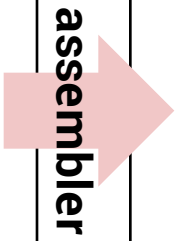
    `*dest:` Memory **M**`[%rdx]`
- **Object Code**
  - 3-byte instruction
  - Stored at address `0x40059e`

Assembling

Assembly has **labels**.

```
pcount_r:
    movl    $0, %eax
    testq   %rdi, %rdi
    je      .L6
    pushq   %rbx
    movq    %rdi, %rbx
    shrq    %rdi
    call    pcount_r
    andl    $1, %ebx
    addq    %rbx, %rax
    popq    %rbx
.L6:
    rep ret
```

gcc –S pcount.c



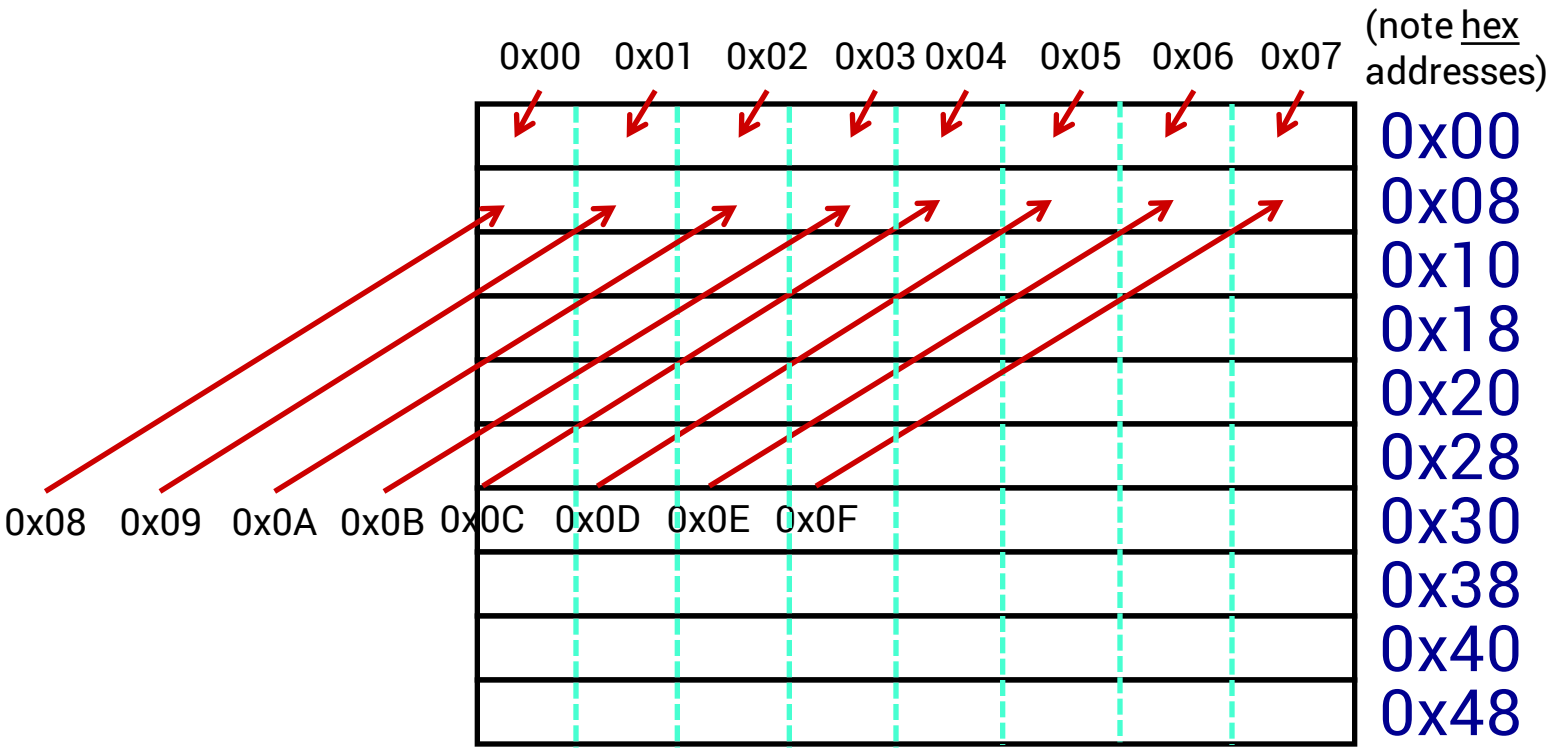
Executable has **addresses**.

```
00000000004004f6 <pcount_r>:
  4004f6:  b8 00 00 00 00   mov    $0x0,%eax
  4004fb:  48 85 ff         test   %rdi,%rdi
  4004fe:  74 13            je     400513 <pcount_r+0x1d>
  400500:  53               push   %rbx
  400501:  48 89 fb         mov    %rdi,%rbx
  400504:  48 d1 ef         shr    %rdi
  400507:  e8 ea ff ff ff   callq  4004f6 <pcount_r>
  40050c:  83 e3 01         and    $0x1,%ebx
  40050f:  48 01 d8         add    %rbx,%rax
  400512:  5b               pop    %rbx
  400513:  f3 c3            repz retq
```

gcc –g pcount.c –o pcount
objdump –d pcount

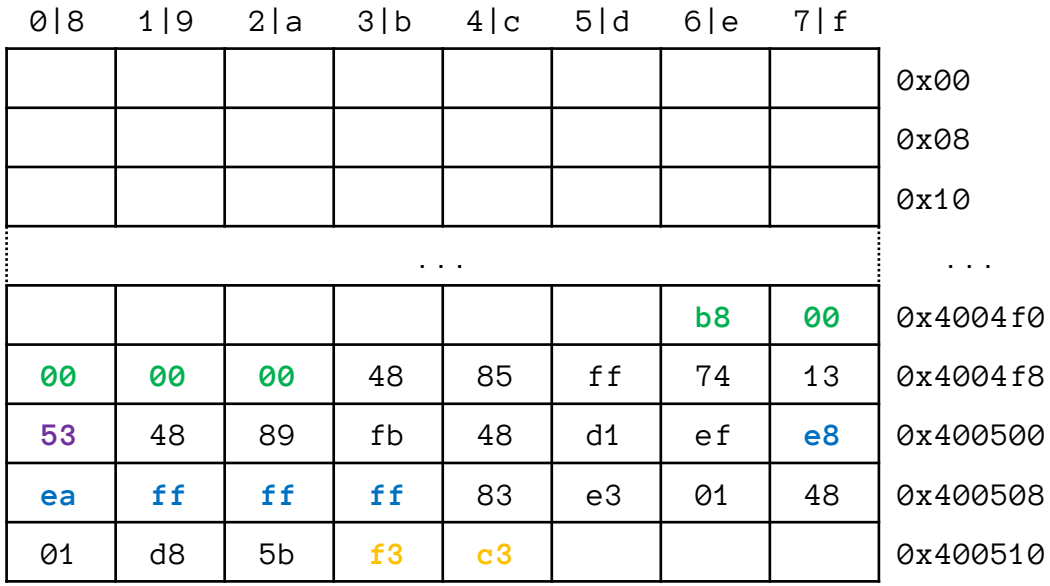
A Picture of Memory (64-bit view)

- **A "64-bit (8-byte) word-aligned" view of memory:**
  - In this type of picture, each row is composed of 8 bytes
  - Each cell is a byte
  - A 64-bit pointer will fit on one row



A Picture of Memory (64-bit view)

```
0000000004004f6 <pcount_r>:
  4004f6:  b8 00 00 00 00    mov    $0x0,%eax
  4004fb:  48 85 ff          test   %rdi,%rdi
  4004fe:  74 13             je     400513 <pcount_r+0x1d>
  400500:  53                push   %rbx
  400501:  48 89 fb          mov    %rdi,%rbx
  400504:  48 d1 ef          shr    %rdi
  400507:  e8 ea ff ff ff    callq  4004f6 <pcount_r>
  40050c:  83 e3 01          and    $0x1,%ebx
  40050f:  48 01 d8          add    %rbx,%rax
  400512:  5b                pop    %rbx
  400513:  f3 c3             repz retq
```



Just For Fun

This `call` actually uses a *relative* address:

`ea ff ff ff` = -22 (little endian)