Computer Systems Org.

What is it about?

- Getting "under the hood"
- Learning how computers actually work.

Start by programming in C.

```c
#include <stdio.h>

typedef int mynum;

mynum x = 3;

typedef int *mypointer;

mypointer p = &x;

typedef struct cell {
  int data;
  struct cell *next;
} CELL;

void build_list(int n)
{
  CELL *head = NULL;
  for(int i = 1; i <= n; i++) {
    CELL *p = (CELL *)  malloc(sizeof(CELL));
    p->data = i;
    p->next = head;
    head = p;
  }
  CELL *q;
  for (q = head; q != NULL; q = q->next)
    printf("%d ", q->data);
  printf("\n");
}

int main()
{
  build_list(10);
}
```

The C compiler translates your C program into Assembly code.

- Human readable version of machine code.

```
        .text
        .globl  allocate_cell
        .def    allocate_cell;  .scl    2;      .type   32;     .endef
allocate_cell:
        pushq   %rbp
        movq    %rsp,%rbp
        movq    $16,%rcx
        subq    $32,%rsp
        call    malloc
        addq    $32,%rsp
        popq    %rbp
        retq

        .globl  create_list
        .def    create_list;    .scl    2;      .type   32;     .endef
create_list:
        pushq   %rbp
        movq    %rsp,%rbp
        movq    $0,%rax
        movl    $10,%ecx
TOP:
        cmpl    $0,%ecx
        jle     DONE

        pushq   %rax
        pushq   %rcx
        subq    $32,%rsp
        call    allocate_cell
        addq    $32,%rsp
        movq    %rax,%rdx

        popq    %rcx
        popq    %rax
        movl    %ecx,0(%rdx)
        movq    %rax,8(%rdx)
        movq    %rdx,%rax
        decl    %ecx
```

You will be writing assembly code, too.

The assembly code is translated to machine code by an assembler.

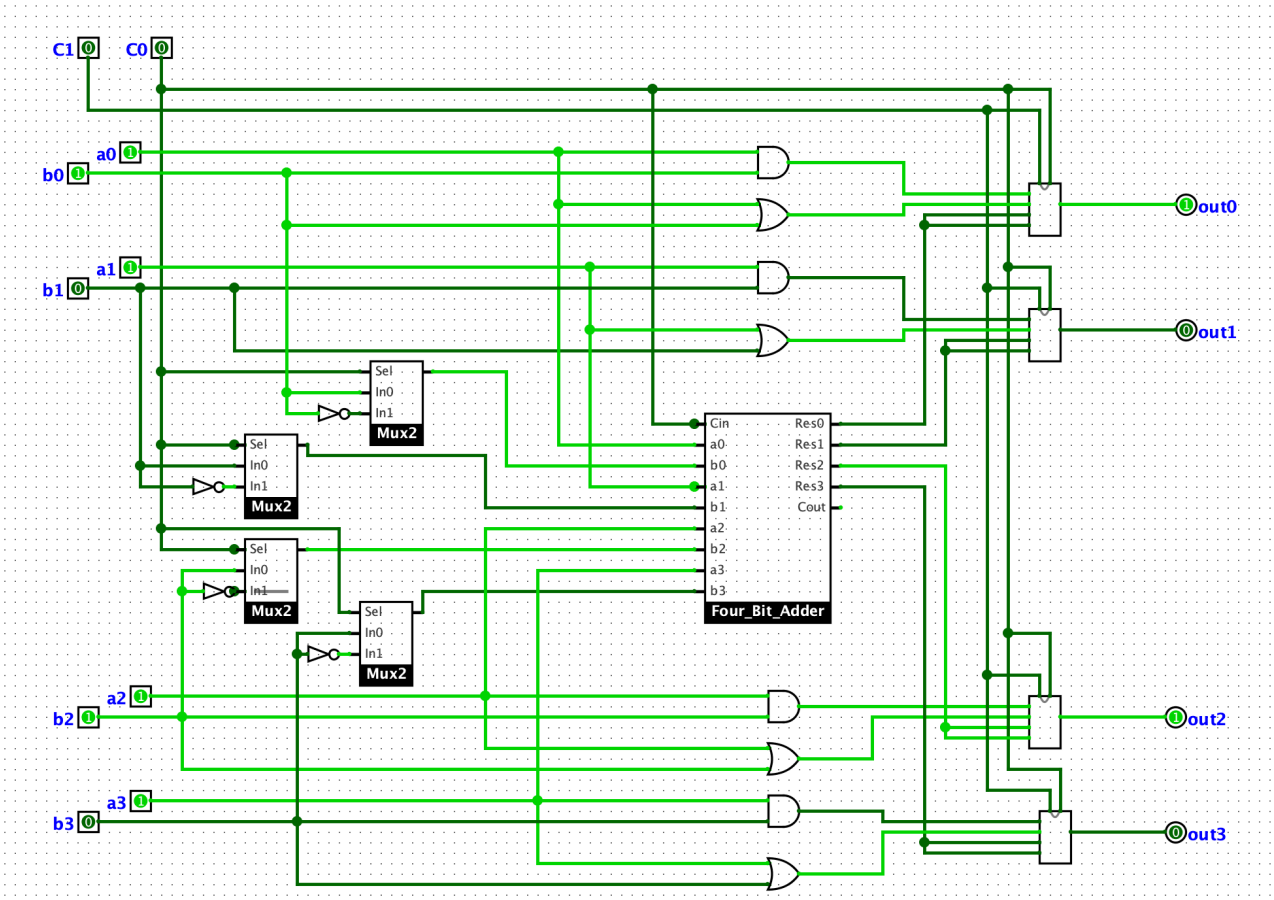
machine code:

```
01101010101...
10110110101...
11100010111...
```

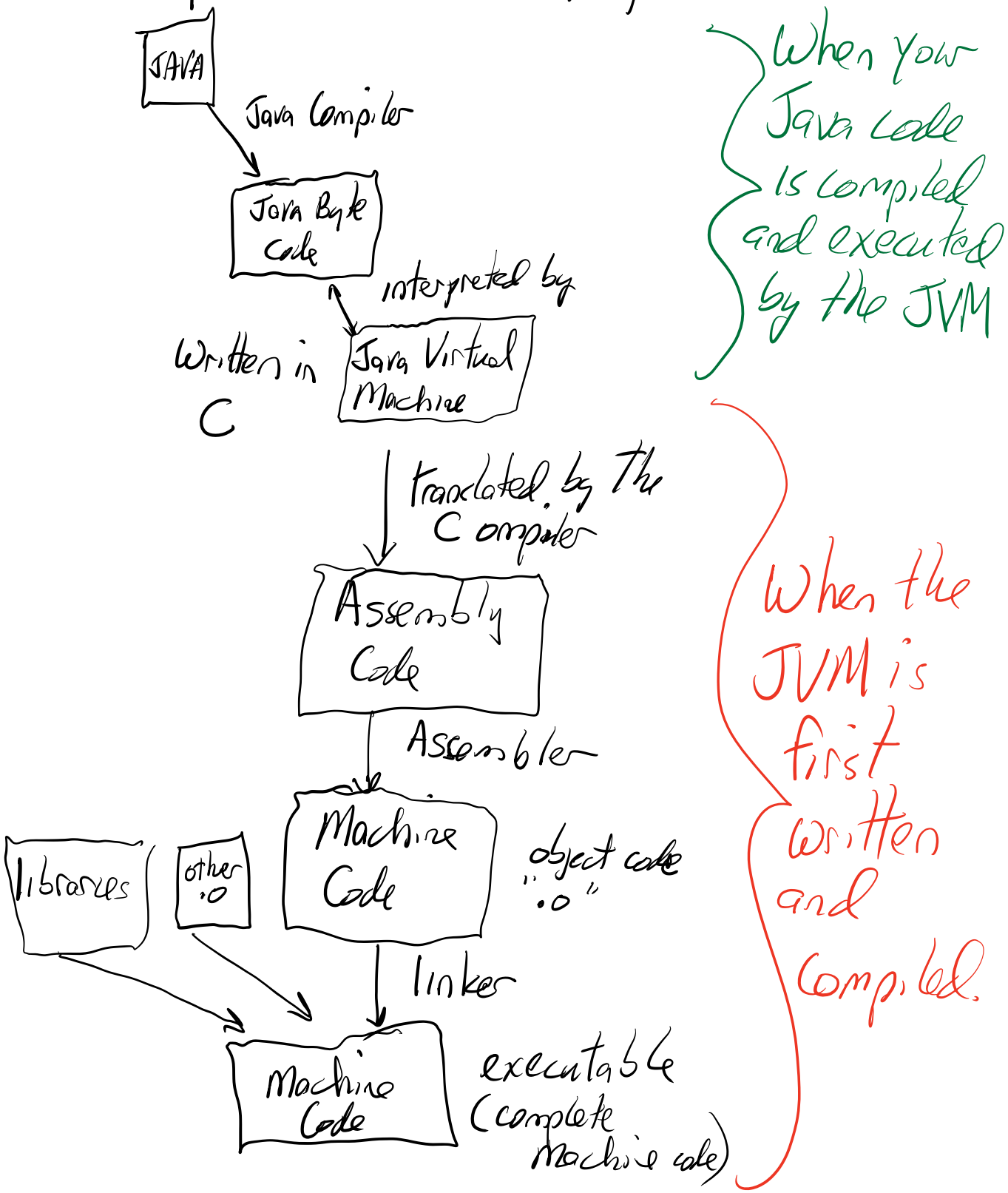
The hardware can then execute the machine code.

- You won't be writing machine code!

In this class, you will use a circuit simulator to "build" the processor hardware.



When you write a Java program:



When you invoke the GCC C compiler it will also call the assembler and linker for you.

"gcc ben.c"
will generate an executable directly, called "a.out" on MacOS & Linux, "a.exe" on windows.

Some terminology:
"bit": - comes from "binary digit"
- either a 0 or a 1.

"byte": 8 bits

e.g. 01101100

- enough to represent one character: A B .. Z a b ... z, 0 1 2 .. 9

"word": (Depends on the machine). This is typically the number of bits used to represent an integer or a pointer.

- Historically, generally 32 bits (4 bytes)
- Modern machines tend to have 64-bit (8-byte) words.

I will try to be explicit about the word size when I'm discussing words, but generally I mean 32-bits (4 bytes).

Material for the quiz on Wednesday 2/2:

- See the handout on "Powers of Two and Hexadecimal Numbers" on Brightspace under "Resources".

Powers of 2

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$

$2^9 = 512$
$2^{10} = 1024 = 1K$
$2^{20} = 1M$ ("Meg")
$2^{30} = 1G$ ("Gig")
$2^{40} = 1T$ ("Tera")

Memorize all these.

So, what number is 2^{32}?

$2^{32} = 2^{(30+2)} = 2^{30} \times 2^2 = 1G \times 4 = 4G$

Based on the formula,

$2^{a+b} = 2^a \times 2^b$

Log base 2

Log base 2 is just the reverse process:

What is $\log 64M$?

$\log 64M = \log(64 * 1M) =$

$\log 64 + \log 1M = 6 + 20 = 26$

Based on the formula

$\log(a*b) = \log a + \log b$