CPSC 185
Introduction to Computing

The course home page

http://cs.calvin.edu/CS/185

1

Sec. 1.2

What is Programming?

- *Programming:* designing, writing and maintaining a program.
- What is a *program*?
  - — a collection of statements that
  - — specify the steps taken to solve a problem, and are
  - — written in a programming language — a language that the computer can understand.

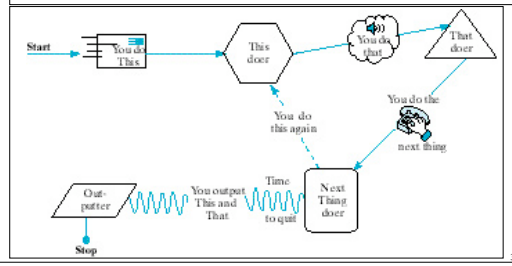
2

What About OOP (Object-Oriented Programming)?

Object-oriented program:

a collection of object interactions that solve a problem

```
Start
Repeatedly do the following:
   1. This
   2. That
   3. The Next Thing
   4. If it's time to quit, stop this repetition
Output the results of This and That
Stop
```



3

Sec. 1.1: Brief History of OOP & Java

- Java is an OOP language
  - — Uses objects to carry out the tasks
  - — Sends messages to the objects to perform the tasks
  - — Objects interact with each other to do the tasks
  - — An actual object is called an instance of a class (the declaration of or blueprint for the object)

Sec. 0.3

What kinds of statements do computers understand?

- A computer only understands a specially designed **machine language**.
- Machine language statements are:
  - Stored in a computer's memory, which is a sequence of binary digits (**bits**):
    - 0 (a switch in the "off" position)
    - 1 (a switch in the "on" position)
  - Retrieved from memory and executed one at a time

5

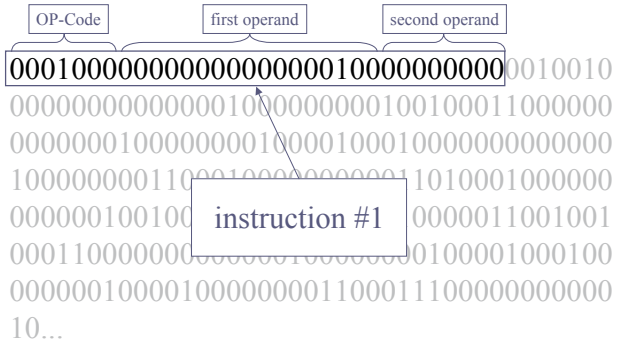
Machine Language (ML) Example

(hypothetical)

```
00010000000000000000010000000000001
00100000000000000010000000001001000
11000000000000010000000010000100010
00000000000010000000011000100000000
00110100010000000000001001000000000
00000010000011001001000110000000000
00010000000100001000100000000010000
10000000011000111000000000010...
```

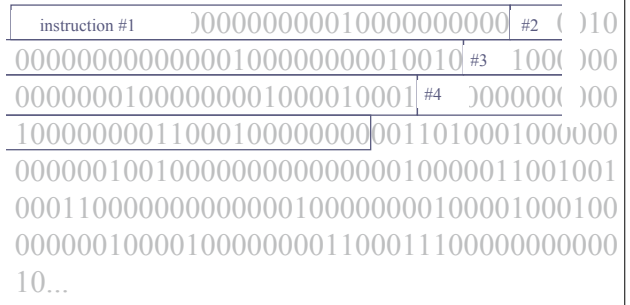
6

ML Example (hypothetical)



7

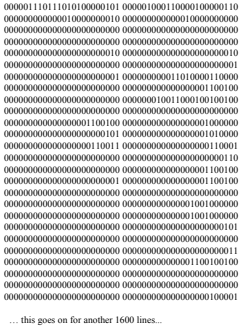
ML Example (hypothetical)



8

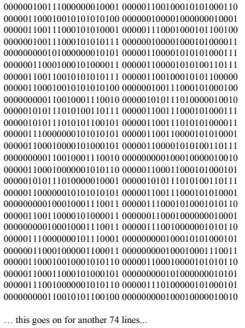
A Real ML Example

SPARC executable machine code



… this goes on for another 1600 lines…

Intel Pentium executable machine code



… this goes on for another 74 lines…

```
int main()
{
  int x, y;
  x = 1;
  y = x + 2;
  return 0;
}
```

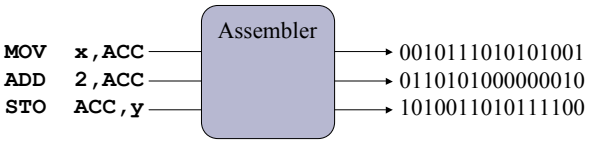
9

- Early Computers required a programmer to write in machine language:
  - It was easy to make mistakes.
  - The mistakes were hard to find.
  - The resulting program was not *portable.*
- As a result, programming in machine language was very difficult.

Assembly Language

To make programming easier, you can:

- Devise a set of abbreviations (mnemonics) corresponding to the machine language instructions.
- Create a program to translate them into machine language.



The Real Example

Intel Pentium assembly language:

```
_main:
        pushl %ebp
        movl %esp,%ebp
        subl $24,%esp
        call ___main
        movl $1,-4(%ebp)
        movl -4(%ebp),%eax
        addl $2,%eax
        movl %eax,-8(%ebp)
        xorl %eax,%eax
        jmp L2
        .align 4
L2:
        movl %ebp,%esp
        popl %ebp
        ret
```

SPARC assembly language:

```
main:
        save    %sp, -120, %sp
        mov     1, %o0
        st      %o0, [%fp-20]
        ld      [%fp-20], %o0
        add     %o0, 2, %o1
        st      %o1, [%fp-24]
        mov     0, %i0
        b       .LL2
         nop
        mov     0, %i0
        b       .LL2
         nop
.LL2:
        ret
        restore
```

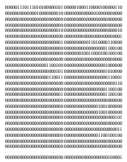
12

The Real Example

Intel Pentium assembly language:

```
_main:
        pushl %ebp
        movl %esp,%ebp
        subl $24,%esp
        call ___main
        movl $1,-4(%ebp)
        movl -4(%ebp),%eax
        addl $2,%eax
        movl %eax,-8(%ebp)
        xorl %eax,%eax
        jmp L2
        .align 4
L2:
        movl %ebp,%esp
        popl %ebp
        ret
```

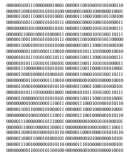
Intel Assembler



SPARC assembly language:

```
main:
        save    %sp, -120, %sp
        mov     1, %o0
        st      %o0, [%fp-20]
        ld      [%fp-20], %o0
        add     %o0, 2, %o1
        st      %o1, [%fp-24]
        mov     0, %i0
        b       .LL2
         nop
        mov     0, %i0
        b       .LL2
         nop
.LL2:
        ret
        restore
```

Sun Assembler



13

Using mnemonics is more natural than binary.

+ Much easier to read programs
+ Much easier to find and fix mistakes
– Still not portable to different machines
– Still hard to write, read, and debug programs

14

High Level Languages

To improve on assembly language:

- Devise a set of statements that are closer to human language and methods of writing expressions called a *high level language* (HLL)

and

- Create a program to translate them into ML:
  - Compiler
  - Interpreter

15

Compilers

A *compiler* translates an entire program into machine language.

```
x = 1;
y = x+2;
```

→ Compiler →

```
0010111000000001
1010011010101001
0010111010101001
0110101000000010
1010011010111100
```

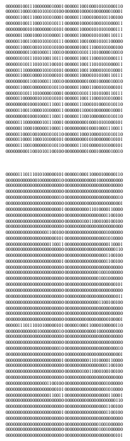
(Note that an assembler translates one AL seatement into one ML statement; a HLL compiler translates one HLL statement into multiple ML statements.)

16

The Real Example

```
int main()
{
   int x, y;
   x = 1;
   y = x + 2;
   return 0;
}
```

Intel C++ compiler

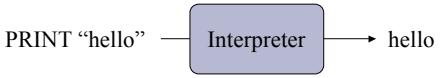


Sun C++ compiler

17

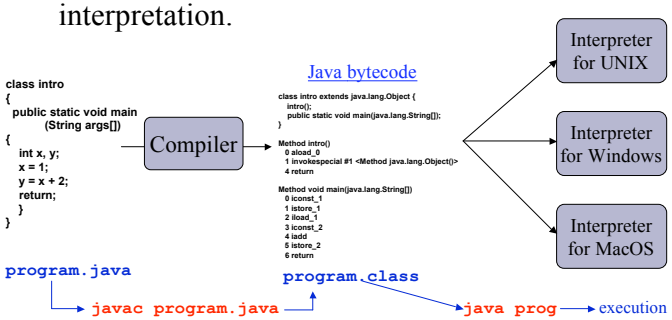
Interpreters

- An *interpreter* decodes each individual instruction and then executes it:
  - development is easier
  - execution is slower



Compilation + Interpretation

Languages can combine compilation and interpretation.



History & Advantages of HLLs

- History of high-level languages:
  - Fortran (1957) - the first compiled language
  - Lisp (1957), BASIC (1964) - interpreted languages
  - Java (1991) - a compiled/interpreted hybrid
- With programming in high-level languages:
  - \+ The programs were much easier to read.
  - \+ Mistakes were much easier to find and fix.
  - \+ The programs were (or could be) portable.
  - – Not so simple that just anyone can use them (otherwise this course wouldn't exist)!

Virtual Machines

- Levels:
  - machine language
  - assembly language
  - high-level language
- Each level is built upon the previous level.
- You can work at one level of abstraction without understanding the implementation of the lower levels.

21

Objectives in Programming

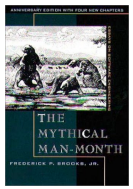
A program should solve a problem:

- *correctly* (it actually solves the problem)
- *efficiently* (without wasting time or space)
- *readably* (understandable by another person)
- in a *user-friendly* fashion (in a way that is easy for its user to use).

22

Fredrick P. Brooks, Jr.

(1931-)



The joys of programming:

We enjoy designing things because we are created in the image of God.

The computer is a powerful and rewarding tool to use.

The woes of programming:

The "mindless" details can be excessively tedious.

Products become obsolete too quickly.

23