

# Background

Computers, Languages, Compilers, ...

# Computers and Programs

- A simplified model of a computer consists of a processing unit (CPU) and a memory.
- CPU can understand simple instructions:
  - read/write a memory location
  - add two numbers
  - compare numbers
  - etc.

# Machine Code

- An executable program is a sequence of these simple instructions.
- The sequence is stored in memory.
- The CPU processes the simple instructions sequentially.
  - Some instructions can tell the CPU to jump to a new place in memory to get the next instruction.

# Instructions

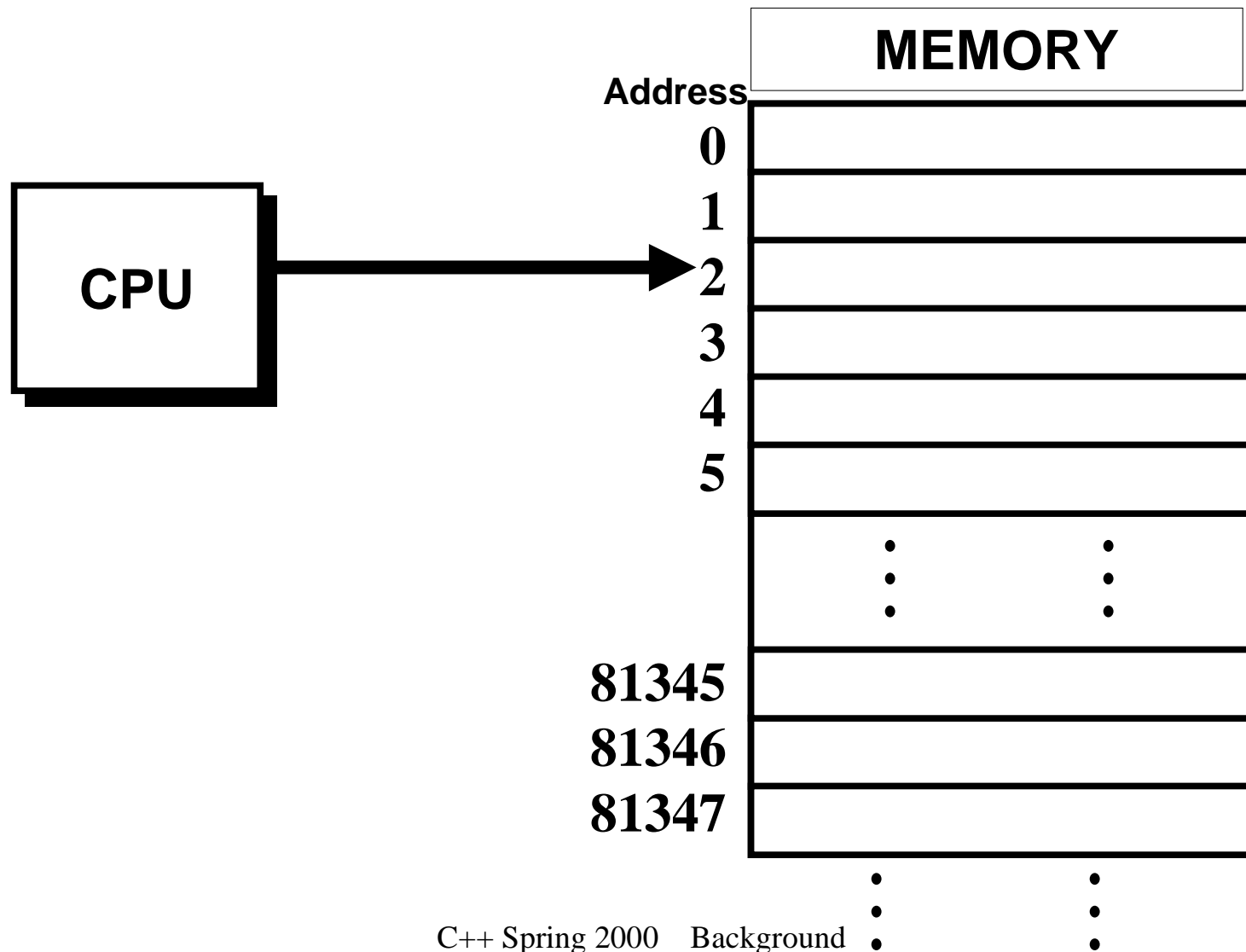
- Each instruction is stored in memory as a bunch of *bits*.
- The CPU *decodes* the bits to determine what should happen.
- For example, the instruction to add 2 numbers might look like this:

**10100110**

# Memory

- The Memory can hold programs *and* data.
- Each piece of data is also just a bunch of bits.
- Typically the memory is organized in chunks of 8 bits (called a byte).
- Each chunk (byte) has an *address*.

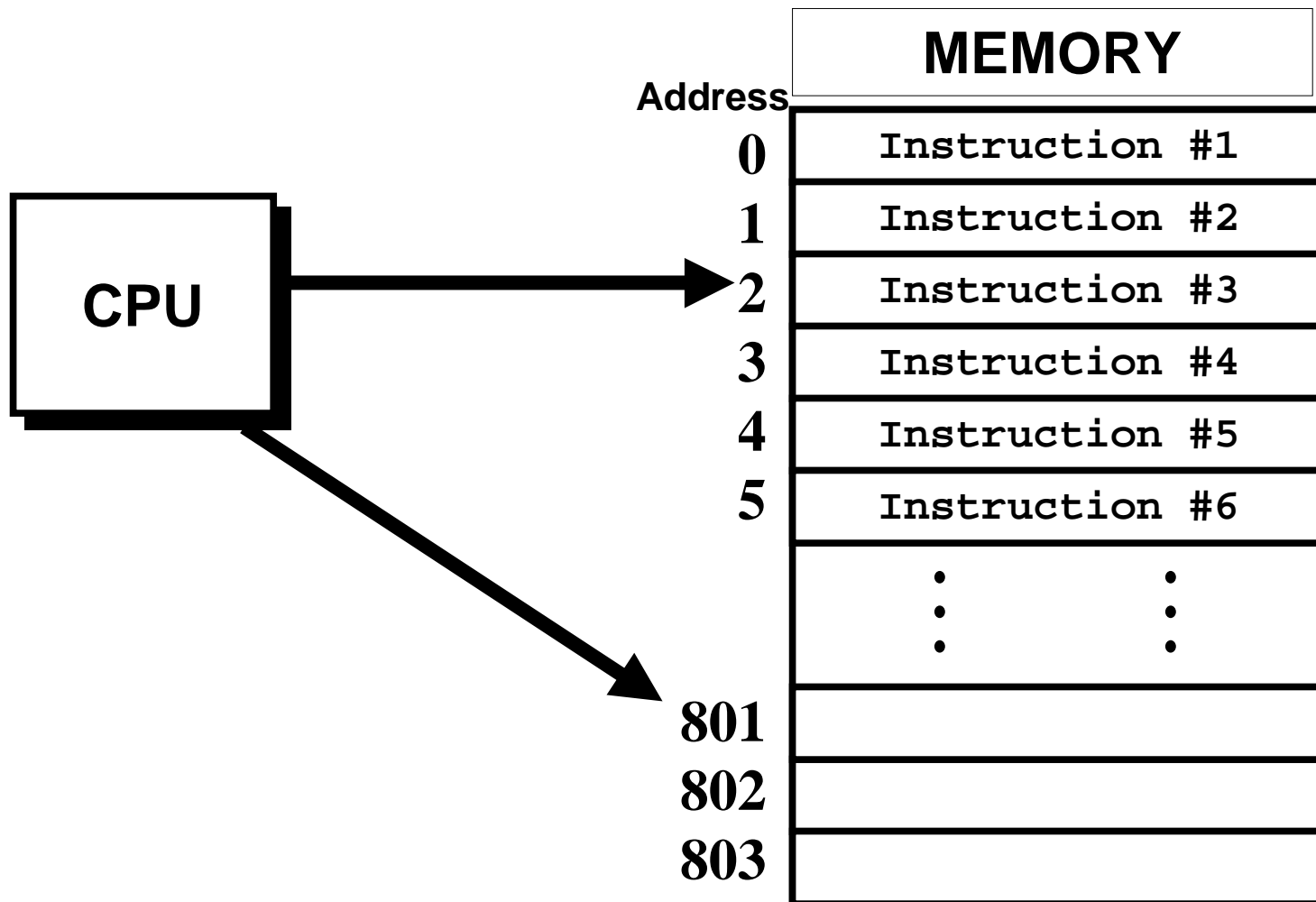
# A Picture





# Sample Program

```
# Instruction
1 Set memory[801] to hold 00000001
2 Set memory[802] to hold 00000000
3 If memory[802] = 10 jump to instruction #8
4 increment memory[802]
5 set memory[803] to 2 times memory[801]
6 put memory[803] in to memory[801]
7 jump to instruction #3
8 print memory[801]
```

# Another Picture



# Human vs Machine Programs

- The computer can only understand the *bits* (the encoded program)  
 ***Machine Language***
- Humans don't like to deal with bits, so they developed english-like abbreviations for programs.  
 ***Assembly Language***

# Assembly & Machine Language

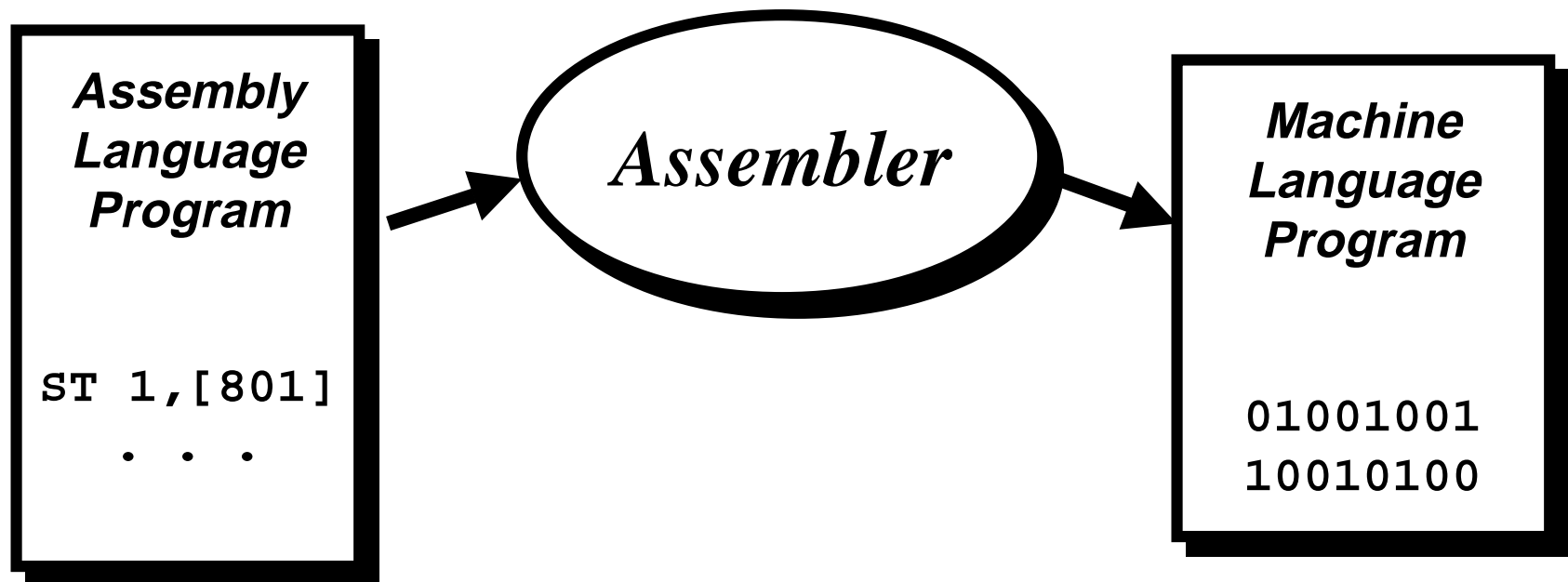
## *Assembly Language*

```
        ST 1,[801]
        ST 0,[802]
TOP:    BEQ [802],10,BOT
        INCR [802]
        MUL [801],2,[803]
        ST [803],[801]
        JMP  TOP
BOT:    LD  A,[801]
        CALL PRINT
```

## *Machine Language*

```
00100101 11010011
00100100 11010100
10001010 01001001 11110000
01000100 01010100
01001000 10100111 10100011
11100101 10101011 00000010
00101001
11010101
11010100 10101000
10010001 01000100
```

# An Assembler



# Higher-Level Languages

- Assembly Language is much easier to deal with than Machine Language, but you need to know about all the instructions.
- People developed languages that were independent of the specific machine language.
  - More abstract representation of instructions.

# High-Level Languages

- Many high-level languages have been developed.
  - Different ways of representing computations.
  - Different languages for different needs:
    - symbolic vs. numeric computation
    - human efficiency vs. program efficiency
    - portability
    - extensibility

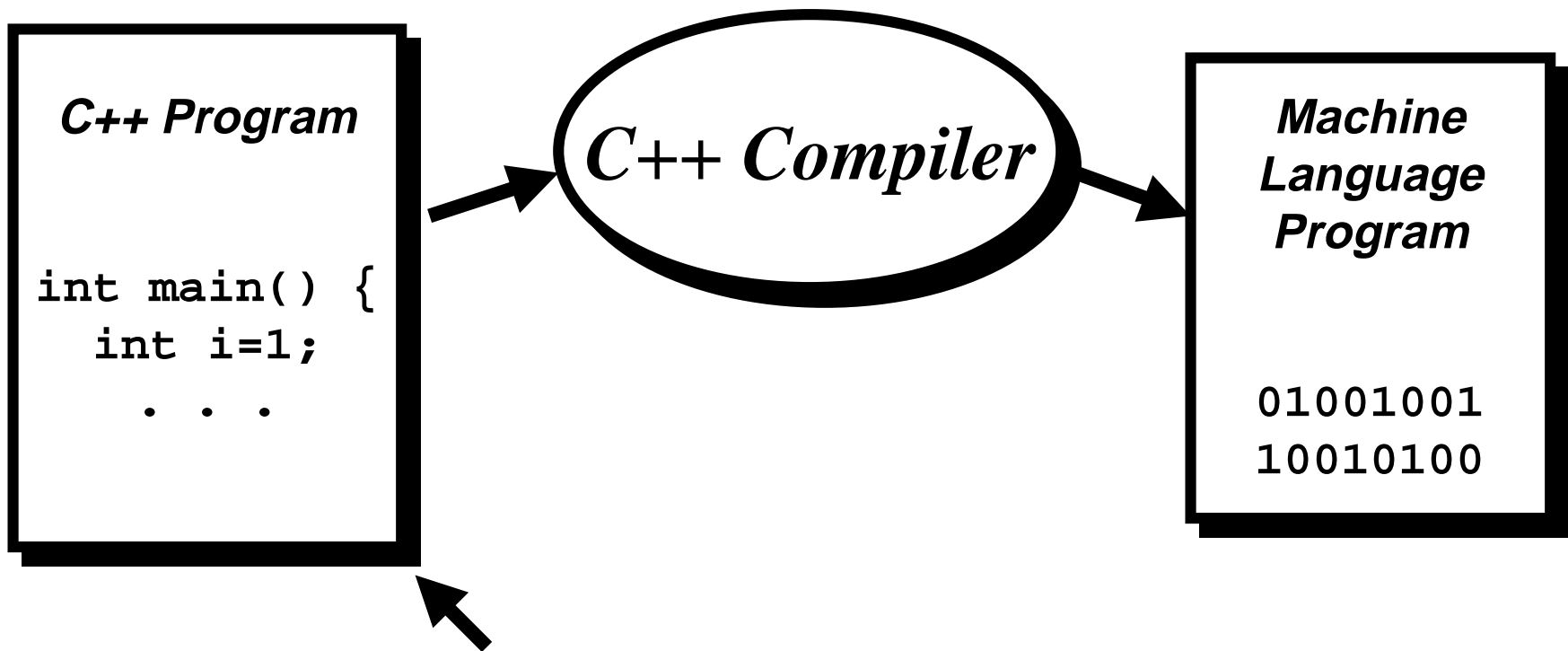
# C++

- C++ is an extension of C.
- C++ was first developed in the early 1980s (back in the last century!).
- Focus was on Object Oriented Programming
  - view computer programs as collection of objects.
  - Objects have attributes and actions.

# As a C/C++ program

set memory[801] to hold 00000001	←.....	x=1;
set memory[802] to hold 00000000	←.....	i=0;
if memory[802] = 10 jump to instruction #8	←...	while (i!=10) {
increment memory[802]	←.....	i++;
set memory[803] to 2 times memory[801]	}.....	x=x*2;
put memory[803] in to memory[801]		.....
jump to instruction #3	←.....	
print memory[801]	←.....	printf("%d",x);

# Compiler



Created with text editor or  
development environment

# Many Different Compilers

- There are many different C++ Compilers:
  - Microsoft Visual C++
  - Borland C++
  - GNU g++
  - IBM xlc
  - Sun CC

# C++ Compiler for Projects

- For this course I don't care what compiler/development environment you use as long as we can compile and run your programs.
- In class we will use both Visual C++ and GNU g++ (possibly other Unix based compilers).