PLTW COMPUTER SCIENCE

Activity 1.3.1

Programs are Data

Introduction

Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

We have seen different kinds of data: numbers in a bank statement, names in a student roster, colors in a list of filters to be used in lighting a school play, even the notes on a musical score can all be viewed as data that is collected, sorted, analyzed, and organized with the aid of computers.

One of the odd facts of computing is that while we program computers to deal with data, the programs that we create are themselves data!

Background

Each block in a Scratch™ or App Inventor program is recorded whenever you save the program. That data is in turn analyzed by other programs called interpreters, compilers, or assemblers. These programs translate your instructions into **machine code**. Machine code is made only with the simple instructions that can be executed by the computer's processor. Machine code is represented as 0s and 1s.

In this lesson we'll look at a text-based language, *Python*®, that looks and feels closer to the machine code than anything we've seen yet. *Python* is still a relatively abstract language, so to give you a feel for the languages that early pioneers of computing like Ada Lovelace used and for the machine language that Scratch, App Inventor, or *Python* are translated into, we will briefly work with a simplified assembly language.

For an assembly language to work, it needs a hardware element called a **register**. The register is a tiny amount of storage space used for holding a number to be added or a memory address. The original 8-bit Nintendo® Entertainment System was called 8-bit because that was the size of its

Copyright © 2017 Project Lead The Way, Inc. All rights reserved.

registers. The difference between a 32-bit computer and a 64-bit computer is also a difference in register size.

The imaginary language we're inventing for this activity is much smaller than real assembly languages. Like real assembly languages, the instructions are made from op codes and data, and additional data can be provided as input or written as output from/to the registers. The imaginary assembly language has just two registers and four op codes. The op codes are shown below, along with what they tell the processor to do.

Each instruction in this language will consist of one byte of data. The two bits furthest to the right (the least significant bits, in the 1 and 2 place values) will represent the op code of the instruction; the six leftmost bits (in the 4s through 128s place values) will be data used by the operation.

Table of OP Codes and Processor Tasks:

OP CODE Mnemonic	Op-code in Binary	What the Processor Will Do
ADD	00	Add the contents of register 1 to the 6 data bits in this instruction, ignore overflow. Put the result in register 2.
MUL	01	Multiply the contents of register 1 by the 6 data bits in this instruction, ignore overflow. Put the result in register 2.
STO	10	Store the 6 data bits in this instruction in register 1. Overwrite only the 6 least significant bits.
REA	11	Read the bits in register 2 and store them in register 1.

Procedure

1. Here is a sample program, written in binary:
   01010000
   01001011
   00000011

   When a program starts we will assume that the registers both have only 0s in them. The beginning state of this program looks like this:

Register 1	Register 2
00000000	00000000

Current Instruction in Binary	Current Instruction Separated		Current Instruction Written with Mnemonic
	OP Code Bits	Data Bits	
01010000	00	010100	ADD 010100

   The first instruction 01010000 is broken down into two parts: the first six bits of data, 010100, and the last two bits of op code, 00. The mnemonic for this is written as ADD 010100.

Copyright © 2017 Project Lead The Way, Inc. All rights reserved.

Observe the registers after ADD 010100 executes. According to the "Table of Op Codes and Processor Tasks" above, the data bits 010100 were ADDed to Register 1 and the result was placed in Register 2.

Register 1	Register 2
00000000	00010100

Fill in the following tables for the execution of the rest of the program:

Instruction 2

Current Instruction in Binary	Current Instruction Separated		Current Instruction Written with Mnemonic
	OP Code Bits	Data Bits	
01001011			

Registers After Instruction 2

Register 1	Register 2

Instruction 3

Current Instruction in Binary	Current Instruction Separated		Current Instruction Written with Mnemonic
	OP Code Bits	Data Bits	
00000011			

Registers After Instruction 3

Register 1	Register 2

2. The key concept for this activity is that 0s and 1s can represent any digital data, including programs themselves. Binary data might be an image of a beautiful sunset, or it might be a malicious virus. A computer does exactly what the operating system and the applications it is running tell it to do with the 0s and 1s. That binary data might be intended to be an image, but the computer could just as easily be told to execute that data as assembly instructions.

   As an example, the following program is composed of the three bytes of data that represent

Copyright © 2017 Project Lead The Way, Inc. All rights reserved.

the RGB values of a shade of cyan:

- Byte 1: 00000000
- Byte 2: 11111110
- Byte 3: 11111100

Explain why these three bytes represent a cyan pixel.

3. Now suppose the three bytes representing this pixel are instead executed by the processor using the instruction set described in the "Table of Op Codes and Processor Tasks". What are the values stored in the two registers after this program has executed? The registers will again start out containing all 0s.

Register 1	Register 2
00000000	00000000

Instruction 1

Current Instruction in Binary	Current Instruction Separated		Current Instruction Written with Mnemonic
	OP Code Bits	Data Bits	

Registers After Instruction 1

Register 1	Register 2

Instruction 2

Current Instruction in Binary	Current Instruction Separated		Current Instruction Written with Mnemonic
	OP Code Bits	Data Bits	

Registers After Instruction 2

Register 1	Register 2

Copyright © 2017 Project Lead The Way, Inc. All rights reserved.

Instruction 3

Current Instruction in Binary	Current Instruction Separated		Current Instruction Written with Mnemonic
	OP Code Bits	Data Bits	

Registers After Instruction 3

Register 1	Register 2

4. The first program that we examined in this activity was composed of the first three bytes of this Word® document. What other types of files contain data that could make a program in this language?

Conclusion

1. “The first program in this exercise was actually pulled directly from the bit representation of this Word® document.”
    - How did this activity change your perception of what programs are?
    - How did it change your perception of what data are?
2. Real assembly languages are similar to the simplified one used in this activity, though the way they process instructions and their instruction sets may vary greatly. For example, most assembly languages give programmers the ability to manipulate multiple registers, specifying which registers to work with, or they grant access to data outside of the CPU. What do you think the challenges of programming in an assembly language instead of a high level language like Scratch would be?
3. Run a shade of yellow represented by decimal RGB values of 255, 255, and 102. What are the end contents of the registers?

Copyright © 2017 Project Lead The Way, Inc. All rights reserved.