Memory organization of programs

Why?

We often take for granted that there are several kinds of data that make up a running program. You are already familiar in some way with all of these data: code, pre-allocated data, dynamically allocated data, and local data. Now you will see how they are organized; we are getting closer to explaining how a program runs on computer hardware.

Materials checklist

- Quality controller: be sure you team has
  - printed copy of this activity for the Recorder
  - 2 printed copies of the memory diagram
  - at least one computer with MARS installed and the file mem.s downloaded

Model 1: Kinds of storage, lifetime, and scope

```
1  class Example {
2      static int x;
3      int y;
4
5      public int doSomething() {
6          int z = this.y + x;
7          return z;
8      }
9      public String getString() {
10         return "Hello world " + y;
11     }
12     public static void main(String[] args) {
13         Example e = new Example();
14         System.out.println(e.getString());
15         int a = e.doSomething();
16     }
17 }
```

Kinds of storage: code (CODE), class variable (CV), instance variable (IV), object (OBJ), local variable (LV)
Times when memory is allocated: before the program starts running (BP), when a function is called (FC), when the `new` operator is called (NEW).
Times when memory is deallocated: after the program finishes running (AP), when a function returns (FR), when it is garbage-collected (GC).

1. The following table names all the data in the program. For each row, use Model 1 to determine what that data's attributes are. Note that the Model includes the possible answers for each attribute. We've completed the first row for you.

Data	**Kind of storage**	**When is memory allocated?**	**When is memory deallocated?**	**What lines of the code are allowed to read or write this data?**
`doSomething`, `getString`, and `main`	code	before program starts	when program ends	any lines of code
x	int			
y				
z				
e				
a				
object created by `new Example()`				

2. Send your presenter to another team to compare your answers to #1. Circle the cells where your teams disagreed and let your team know.

Compared with team #:_____

Number of cells that disagreed: _____

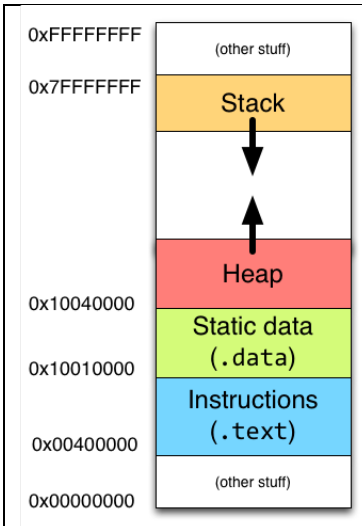
Read This!

The ***lifetime*** of storage is when its memory is allocated and deallocated. The ***scope*** of storage is what parts of the program it is visible/accessible from. The lifetime and scope together partly characterize a kind of storage.

3. What is the purpose of having multiple kinds of storage, each with different lifetime and scope? Why don't we just have one kind of storage?

Model 2: Memory organization

A MIPS program running on a computer has a 32-bit address space, with bytes labeled 0x00000000 to 0xFFFFFFFF. Different kinds of storage are allocated to different regions of this address space. Below is the allocation we will assume.



Stack: local variables, return addresses; grows down

Heap: dynamically allocated memory such as objects; grows up

.**data**: global data such as class variables

.**text**: the assembled code

1. What address range may the .text consume? The .data?

2. Integrating data from Model 1's answers and Model 2: what is the lifetime of data in each region?

Region	**Lifetime**
Stack	
Heap	
.data	
.text	

Exercises

Suppose the following MIPS program is running.

Address	Program
	.data
0x10010000	values: .word 2 6 3 0
	.text
0x00400000	j main
	foo:
0x00400004	addiu $sp, $sp, -4
0x00400008	sw $ra, 0($sp)
0x0040000C	addiu $a0, $a0, -1
0x00400010	beq $a0, $zero, fooend
0x00400014	jal foo
	fooend:
0x00400018	lw $ra, 0($sp)
0x0040001C	addiu $sp, $sp, 4
0x00400020	jr $ra
	main:
0x00400024	la $t0, values
0x00400028	#la expands to lui/ori
0x0040002C	lw $a0, 0($t0)
0x00400030	jal foo

Manager: break up the team into Group Paper: those looking at this version of the code and Group MARS: those looking at mem.s in MARS. You will both work to understand the memory organization but with different approaches.

Assume that the running program has just gotten to line `0x00400010` **for the second time.**

Group Paper:

3. Trace through the program on paper up to when 0x00400010 is reached for the second time. In the memory diagram, draw in the data. Be precise about the ***address*** (or address range) where the data is and its ***value***. When there is a lot of data (e.g., Code), you can summarize your answer as "machine code at address range _____".

Group MARS:

4. Follow these steps to analyze the program:
    a. Assemble mem.s and set a breakpoint (*Bkpt* column) on line 0x00400010.
    b. Click Run. The program should stop at the line. That is the first time the program reaches 0x00400010.

c. Now click Run again. The program should stop at the line for the second time.
d. Now use the Data Segment window to look for what values there are in the .text, .data, and stack. Write down the **address** and **value**. When there is a lot of data (e.g., Code), you can summarize your answer as "machine code at address range _____".
    i. If you are unsure of when a particular memory location was written to, then use the step backwards button until you see the value in that memory location change.

5. **Check each other**. Resolve the conclusions of Group Paper and Group MARS to fill out your memory diagram. Remember to include addresses and values of all the data.

Read This!

You should have found that the Heap region of memory was empty. To dynamically allocate memory from the Heap, we need to use the special procedure sbrk ("s-break"). Sbrk asks the operating system for a *contiguous* chunk of bytes. The incantation for calling sbrk is...

```
li $v0, 9    # syscall code for sbrk is 9
li $a0, 12   # the number of bytes you want to allocate, e.g., 12
syscall
# when syscall finishes, the address of the first byte is in $v0
```

For more information see MARS documentation on syscall: https://courses.missouristate.edu/KenVollmar/mars/Help/SyscallHelp.html.