## **Lab 05 - Introduction to the Raspberry Pi 3 Model B & 16GB NOOBS – ASSIGNMENT**

## **Step 3: Checking the Endian-ness of the RPi**

What is the endian-ness of the Raspberry Pi? Try also the same program in lore.cs.purdue.edu and in data.cs.purdue.edu (lore is not equipped with gcc, use c99 instead.). Type “uname -a” to know what processor each machine uses. Fill in the following table and turn it in during lab next week.

|  |  |  |
| --- | --- | --- |
| Host Name | Architecture (x86, ARM, SPARC) | Endian-ness |
| RPI |  |  |
| lore.cs.purdue.edu |  |  |
| data.cs.purdue.edu |  |  |

## **Step 4. To Do at Home: Program Memory Sections**

The memory usage (footprint) of a program is comprised of the following memory sections:

|  |  |  |
| --- | --- | --- |
| Memory Section Name | Description | Allowed Access Modes |
| text (or code segment) | This area of memory contains the machine instructions that correspond to the compiled program and also contains constants such as string literals and variables defined using the const keyword. If there are multiple instances of a running program then typically all instances share this memory area. | Read, Execute |
| data | This region of memory for a running program contains storage for initialized global variables and static variables that are explicitly initialized to a non-zero value. There must be a separate data segment for each running instance of a program. | Read, Write |
| bss | This memory area contains storage for uninitialized global variables and static variables that are not explicitly initialized or initialized to zero. It is also separate for each running instance of a program. | Read, Write |
| stack | This region of the memory image of a running program contains storage for the automatic (non-static, local) variables of the program. It also stores context-specific information before a function call, e.g. the value of the Instruction Pointer (Program Counter) register before a function call is made. For most architectures the stack grows from higher memory addresses to lower memory addresses. A running instance of a program may have multiple stacks (as in a multi-threaded program) | Read, Write |
| heap | This memory region is reserved for dynamically allocating memory for variables at run time. Dynamic memory allocation is done by using the malloc() or calloc() functions. | Read, Write |
| shared libraries | This region contains the executable image of shared libraries being used by the program. | Read, Execute |

Using the table above as a guide, draw an approximate map of the memory of the program indicating the text, data, bss, stack, heap of the program. Also draw where each variable is located as well as the address as indicated by the program.

## **Step 5. To Do at Home: Memory Dump**

Run your version of memdump. On the output, indicate where the following items are located:

* str
* a
* b
* y
* x.a
* x.i
* x.b
* x.p
* head
* head->str
* head->next
* head->next->str
* head->next->next
* head->next->next->str
* head->next->next->next

Also, show the binary value of -5 (two’s complements of 5).

Also show the value of y for the sign, mantissa, and exponent. Verify that the value stored in memory is correct.

## **To turn in next week**

Upload your answers in a PDF file to Blackboard.

1. The endian-ness table of Step 3.
2. The map of the sections in memory in Step 4.
3. A printout of your memdump program.
4. The output of your memdump program in Section 5 indicating where the items indicated are found in the output.
5. Two’s complement of -5 as stored in memory on the RPi.
6. Sign, mantissa, exponent of “y” in memdump program and verification that the value stored in memory is correct.