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	ARM	Little Endian
lore.cs.purdue.edu	SPARC	Big Endian
data.cs.purdue.edu	x86	Little Endian

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

Section Output

&b=0x7e871624
&c=0x2112c
&p=0x7e871620
p=0x15d5008
&str=0x7e87161c
str=0x105b4
&d=0x7e87160c
&e=0x21124
main=0x10484
&foo=0x10450

Highest Address
Stack
&b=0x7e871624
&p=0x7e871620
&str=0x7e87161c
&d=0x7e87160c
Heap
p=0x15d5008
Data
&c=0x2112c
&e=0x21124
Text
str=0x105b4
main=0x10484
&foo=0x10450
Lowest Address

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

Head
Head->str
Head->next
Head->next->str
Head->next->next
Head->next->next->str
Head->next->next->next

```
&x=0x7eeda5e8
&y=0x7eeda5f8
0x7eeda5e8: 41 00 00 00 09 00 00 00 30 00 00 00 ec a5 ed 7e A.......0......~
0x7eeda5f8: 00 00 00 00 00 00 28 40 48 65 6c 6c 6f 20 77 6f ......(@Hello wo
0x7eeda608: 72 6c 64 0a 00 07 01 00 a0 5b ec 76 a4 07 01 00 rld......[.v....
0x7eeda618: fb ff ff ff 05 00 00 00 00 00 00 00 00          .............
head=0x1cb9008
0x01cb9008: 18 90 cb 01 28 90 cb 01 00 00 00 00 11 00 00 00 ....(...........
0x01cb9018: 57 65 6c 63 6f 6d 65 20 00 00 00 00 11 00 00 00 Welcome ........
0x01cb9028: 38 90 cb 01 48 90 cb 01 00 00 00 00 11 00 00 00 8...H...........
0x01cb9038: 74 6f 20 00 00 00 00 00 00 00 00 00 11 00 00 00 to ..............
0x01cb9048: 58 90 cb 01 00 00 00 00 00 00 00 00 11 00 00 00 X...............
0x01cb9058: 63 73 32 35 30 00 00 00 00 00 00 00 a1 0f 02 00 cs250...........
0x01cb9068: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0x01cb9078: 00 00 00 00 00 00 00 00                         ........
```

The hex of ff ff ff fb is -5. In binary it would be 1011 with 28 sign extended 1s in front of it.

For y, the sign is the first bit of the 4 in the hex 4028. The exponent takes the remaining 3 bits of the 4 and the 28. The mantissa I the 8 in the hex 4028. A binary breakdown would be the sign is “0” for positive, the exponent is **“10000000010” and the mantissa is “1000000000000000000000000000000000000000000000000000”.** If the double was -12, rather than 12, the hex would have been c028.