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

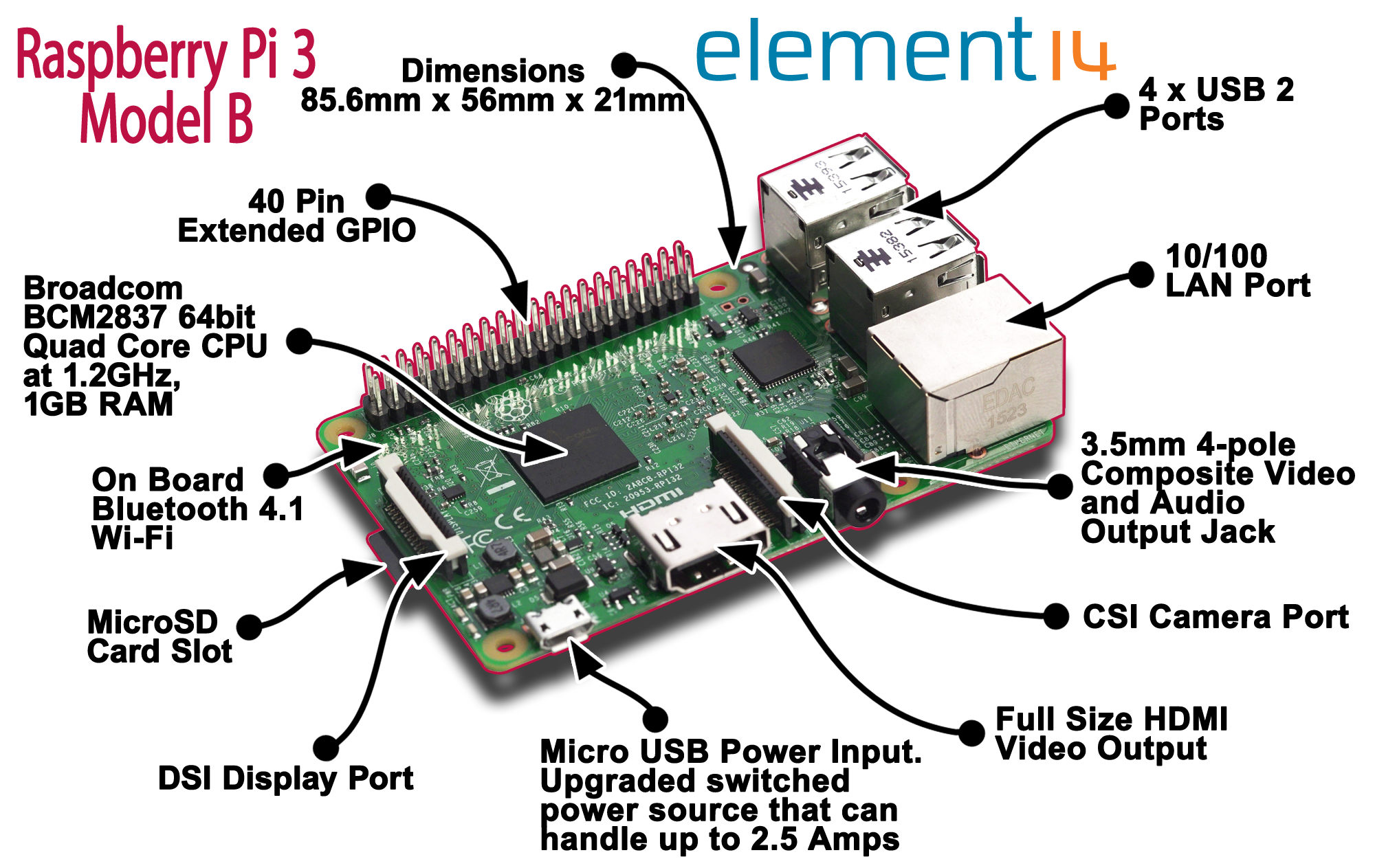
## **Introduction**

The **Raspberry Pi** is a small, low-cost, yet powerful computer intended for students and hobbyists to quickly learn and apply computer science and electronics skills. Unlike the desktop or laptop computers that are so widespread in the 21st century, the Raspberry Pi (or just Pi for short) is a **single-board** computer: all of its components are located on one credit-card sized board. These components, along with a large number of available Linux-based operating systems (known as distributions, or **distros**), allow the Pi to act as a full-fledged Linux-based computer, despite its small size and tiny power usage.1)



The RPi 3 Model B & 16GB NOOBS components include, but are not limited to:

* 1.2 GHz Quad-Core CPU and 1GB RAM;
* Onboard HDMI and DSI output;
* 4 USB ports
* Ethernet port
* Micro-USB power connector
* Micro-SD card slot
* 40 low-level pins which can be used to interface the Pi to other computers and electronic hardware
* WIFI
* Bluetooth
* 16GB micro-SD card with NOOBS (New Out-Of-the-Box Software)



## **Step 0: Safety for You and Your Pi**

Your Pi comes with a Safety Guide sheet. Take a minute and read through the Safety Instructions in the language of your choice. Most important is that your Pi should be placed on a stable, non-electrically-conductive surface. Your Pi has many exposed wires, and shorting out these wires or allowing them to come into contact with incompatible voltages often will damage a Pi.

**Step 1: Starting Your Pi**

Find the Quick Start Guide in the box with your Pi. Follow the Plugging In Your Raspberry Pi instructions in order. The microSD card must be removed from its full-size SD card carrier before you insert it into the microSD card slot on the bottom side of your Raspberry Pi.

You may disconnect the keyboard and mouse from the computer at your LSWN B160 lab station and connect them to two of the USB ports on your Pi. Alternatively, find the bundled USB cables at your station labeled Keyboard & Mouse and connect them to your Pi. Find the small rectangular button on a wire labeled KEYBD & MS SWITCH. Clicking this switch toggles between the lab PC having the keyboard and mouse, and the external cables linking the keyboard and mouse to your Pi. The transfer of control to the Pi takes a second or two.

Find the blue HDMI cable labeled Monitor and connect that to your Pi; this will let you use the computer display at your station for your Pi.

The last connection you should make is to power. Finally, connect the USB-to-microUSB cable from your lab kit to the USB power source in the lab table and connect the microUSB end to your Pi. Your Pi should start to boot, showing a red LED and a green LED flashing occasionally. A detailed quick start guide with graphics is available at <http://www.raspberrypi.org/qsg>

The Pi runs its operating system from its SD card. While several operating system distros have been made available for the Pi, the one that we will be using is known as **Raspbian**. 2).

**Step 2: Recommended Configuration**

**Changing your password**

Your Pi will automatically log you in as user pi with password raspberry. It is a great idea to change your password from the default password. Having access to your Pi's password allows you to remotely access it via console cable or the Internet. For the sake of security, it is imperative that you change this password to something more secure immediately.

### **Via terminal**

While connected to the Pi's terminal, type the command passwd. This will prompt you for your current password (raspberry) and then ask you to enter a new password. For more information, consult man passwd.

Changing the keyboard layout

Raspbian defaults to the standard keyboard layout for United Kingdom, which is a little different from that of United States. For example, the key pressing “SHIFT+3” will be interpreted as the character ‘[£](https://en.wikipedia.org/wiki/Pound_sign" \t "_blank)’ instead of ‘#’. If you prefer the US keyboard layout, you can change it via the terminal. We don’t recommend using available GUI to change the default keyboard because the change may not hold across restarts of your Pi. Use your favorite text editor to edit this file, /etc/default/keyboard using the sudo command to gain privilege to make the change. The commands to a shell prompt in the Terminal application is you use the vim editor is:

$ sudo vim /etc/default/keyboard

then change the line  
 XKBLAYOUT=”gb”  
to  
 XKBLAYOUT=”us”

The keyboard change will go into effect after you restart your RPi because the Pi reads the file /etc/default/keyboard only during its boot up process.

Connecting to the internet and installing other software

Raspbian is a Debian-based Linux distro, so you can install software using the tool apt-get once you have access to the internet from RPi. You can connect to the internet via Ethernet cables or WiFi. Ethernet ports are available at each lab position in LWSN B160 and is the recommended way to connect to the internet in lab because you will have a high-speed, hassle-free connection, but you will need to bring your own Ethernet cable. Using WiFi requires tweaking the default Raspbian system configuration files.

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

As you we covered in class, integers can be stored in memory in two ways: Little-Endian or Big-Endian. In Little-Endian the least significant byte of the integer is stored the lowest numbered address in memory and each more significant byte of the integer is stored in the locations of sequentially increasing addresses. Big-Endian is the reverse of Little-Endian: the most significant byte is stored in the lowest address in memory.

Create a directory cs250/lab05-src in your RPI using the commands below. Don’t type the `$’ character because it stands for the prompt. A prompt is given by the system on the terminal and will look like pi@raspberrypi:~ $

$ cd  
$ mkdir -p cs250/lab04-src  
$ cd cs250/lab04-src  
$ nano endian.c

Then copy (type in) the following code:

-------------------------- endian.c ------------------------

#include <stdio.h>  
  
int isLittleEndian() {  
 int a = 0x05;  
 char \* p = (char \*) &a;  
 if (\*p==0x05) {  
 return 1;  
 }  
 return 0;  
}  
  
int main() {  
 if (isLittleEndian()) {  
 printf("Machine is Little Endian\n");  
 }  
 else {  
 printf("Machine is Big Endian\n");  
 }  
}

--------------------- end of endian.c --------------------

Then save the program using Ctrl-x and say yes (Y) to save it and type Return. Then compile it and run it:

$ gcc -o endian endian.c  
$ ./endian

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 |

Open a file sections.c and copy the following program into sections.c:

$ cd cs250/lab4-src  
$ nano sections.c

---------------------------- sections.c --------------------------

#include <stdio.h>  
#include <stdlib.h>  
  
int a=5;  
  
int buffer[1000000];  
  
int foo() {  
 int d;  
 static int e = 5;  
  
 printf("&d=0x%x &e=0x%x\n", &d, &e);  
}  
  
int main() {  
 int b;  
 static int c;  
 int \* p = (int \*) malloc(sizeof(int));  
 char \* str = "Hello World\n";  
  
 printf("&b=0x%x &c=0x%x\n", &b, &c);  
 printf("&p=0x%x p=0x%x\n", &p, p);  
 printf("&str=0x%x str=0x%x\n", &str, str);  
 foo();  
  
 printf("main=0x%x &foo=0x%x\n", main, &foo);  
}

--------------------------- end of sections.c ---------------------

Then save the program, compile it and run it.

$ gcc -o sections sections.c  
$ ./sections

Using the table above as a guide, paper 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**

Using your RPi, copy the following code into a file named memdump.c

$ cd cs250/lab04-src  
$ nano memdump.c

Complete the function memdump(char \*p, int len) in the following program that dumps in hexadecimal, byte by byte the contents of memory starting at location “p” and continuing for len bytes. An example output is given below.

--------------------------- memdump.c ------------------------

#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
  
void memdump(char \* p , int len) {  
 // Add your code here.  
   
   
}  
  
struct X{  
 char a;  
 int i;  
 char b;  
 int \*p;  
};  
  
struct List {  
 char \* str;  
 struct List \* next;  
};  
  
int main() {  
 char str[20];  
 int a = 5;  
 int b = -5;  
 double y = 12;  
 struct X x;  
 struct List \* head;  
  
 x.a = 'A';  
 x.i = 9;  
 x.b = '0';  
 x.p = &x.i;  
 strcpy(str, "Hello world\n");  
 printf("&x=0x%x\n", &x.a);  
 printf("&y=0x%x\n", &y);  
  
 memdump((char \*) &x.a, 64);  
 head = (struct List \*) malloc(sizeof(struct List));  
 head->str=strdup("Welcome ");  
 head->next = (struct List \*) malloc(sizeof(struct List));  
 head->next->str = strdup("to ");  
 head->next->next = (struct List \*) malloc(sizeof(struct List));  
 head->next->next->str = strdup("cs250");  
 head->next->next->next = NULL;  
 printf("head=0x%x\n", head);  
 memdump((char\*) head, 128);  
}

------------------------- end of memdump.c ---------------------

Here is an example of the output. For every line print the address, each byte in hexadecimal and the in the right column the ASCII value. The ASCII value will be printed only if the ASCII value is visible 32<=c<=127, otherwise it will print a “.”.

pi@raspberrypi:~/cs250/lab04-src$ ./memdump  
&x=0xbeab36e0  
&y=0xbeab36f0  
0xbeab36e0: 41 76 00 40 09 00 00 00 30 00 00 00 e4 36 ab be Av.@....0....6..  
0xbeab36f0: 00 00 00 00 00 00 28 40 48 65 6c 6c 6f 20 77 6f ......(@Hello wo  
0xbeab3700: 72 6c 64 0a 00 88 00 00 80 2b 2f 40 c0 87 00 00 rld......+/@....  
0xbeab3710: fb ff ff ff 05 00 00 00 00 00 00 00 00 .............  
head=0x84a008  
0x0084a008: 18 a0 84 00 28 a0 84 00 00 00 00 00 11 00 00 00 ....(...........  
0x0084a018: 57 65 6c 63 6f 6d 65 20 00 00 00 00 11 00 00 00 Welcome ........  
0x0084a028: 38 a0 84 00 48 a0 84 00 00 00 00 00 11 00 00 00 8...H...........  
0x0084a038: 74 6f 20 00 00 00 00 00 00 00 00 00 11 00 00 00 to .............  
0x0084a048: 58 a0 84 00 00 00 00 00 00 00 00 00 11 00 00 00 X...............  
0x0084a058: 63 73 32 35 30 00 00 00 00 00 00 00 a1 0f 02 00 cs250...........  
0x0084a068: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................  
0x0084a078: 00 00 00 00 00 00 00 00 00 .........  
pi@raspberrypi:~/cs250/lab04-src$

Run your version of memdump that prints the output above. The output may be different than the output above because the addresses will be different. However, the content will be the same. On your 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.

## **References and footnotes**

**1)** Adapted from [Wikipedia](http://en.wikipedia.org/wiki/Raspberry_Pi).

**2)** Raspbian is a Pi-specific version of the popular distro Debian, which was founded by Ian Murdock while he was a student at Purdue University.