EC310 Memory Storage Example for the Stack

Suppose I have the following variable declarations in a C program:

```
int  zz = 206578;
char letter = 'v';
char word[ ] = "Bird";
int  y = 154;
```

Note:

- decimal value 206578_{10} is 0x326F2 in hexadecimal
- character 'v' is 0x76 in hexadecimal (from ASCII table)
- the characters in the string "Bird" are 0x42, 0x69, 0x72, 0x64, 0x00 (NULL) in hexadecimal (from ASCII table)
- decimal 154_{10} is 0x9A in hexadecimal

When the program is compiled and run, the operating system sets aside storage space in the main memory (RAM) for the program's instructions (in the *text segment*) and its variables (on the *stack*). Suppose the table below shows a portion of RAM where the **stack** is located.

Memory Address	Data at that Memory Address (Hex)
0x08048370	
0x08048371	
0x08048372	
0x08048373	
0x08048374	
0x08048375	
0x08048376	
0x08048377	
0x08048378	
0x08048379	
0x0804837A	
0x0804837B	
0x0804837C	
0x0804837D	
0x0804837E	
0x0804837F	
0x08048380	
0x08048381	

1. How many total bytes are used to store the declared variables in memory?

Answer:
4 bytes for zz, 1 byte for letter, 5 bytes for word, 4 bytes for y: 14 bytes total

2. What are the actual bit values that will be stored in memory? Give your answer as hexadecimal values.

Answer:
Variable *zz* is an integer, so is stored in 4 bytes (which is 8 hexadecimal digits). In memory, its value is: 0x000326F2, but the bytes will be stored in little-endian order: 0xF2, 0x26, 0x03, 0x00

Variable *letter* is stored in one byte (which is 2 hexadecimal digits). In memory, its value looks like: 0x76

Variable *word* (a string) is also stored in one byte per character including the NULL character, and in memory its value looks like: 0x42, 0x69, 0x72, 0x64, 0x00 (in that order).

Variable *y* is an integer, so it is stored in 4 bytes (8 hexadecimal digits). In memory, its value is: 0x0000009A, but the bytes will be stored in little-endian order: 0x9A, 0x00, 0x00, 0x00

3. How/where will the values be stored in memory if ebp = 0x0804837E, and esp = 0x08048370?

Answer:

The *ebp* value is the address right AFTER the stack. On the stack, variables are stored in the order they are declared, from the bottom of the stack up.

int values are stored in "little endian" format, so the least significant byte is stored FIRST in the memory location, and the most significant byte is stored LAST (this is the reverse order of what you'd think it should be).

char values are stored in one byte, so they look as is.

Strings are made up of chars (one byte), and are stored in the correct order.

When the program runs, the memory values will look as follows:

Variable	Memory Address	Data at that Memory Address (in Hex)
y	0x08048370	**9A**
	0x08048371	**00**
	0x08048372	**00**
	0x08048373	**00**
word	0x08048374	42 'B'
	0x08048375	69 'i'
	0x08048376	72 'r'
	0x08048377	64 'd'
	0x08048378	00 <Null>
letter	0x08048379	**76 'v'**
zz	0x0804837A	F2
	0x0804837B	26
	0x0804837C	03
	0x0804837D	00
Garbage bits	0x0804837E	10
	0x0804837F	00
	0x08048380	00
	0x08048381	A3

4. What are the values and addresses of the variables?

Answer:

Recall: the & symbol means "the address of"

zz = 206578_{10} (which is 0x000326F2 in hex), and &zz = 0x0804837A

letter = 'v' (which is 76 in hex), and &letter = 0x08048379

word = "Bird" (which is 0x42, 0x69, 0x72, 0x64, 0x00 (in that order), and &word[0] = 0x08048374, which is the address of the string.

y = 154_{10} (which is 0x0000009A in hex), and &y = 0x08048370