1. Explain what each of the assembly language instructions in exampleIML-1b.s does *and why*. (A couple of the "whys" aren't obvious, so don't hesitate to ask for help.) Two hints:

* Remember, this code is un-optimized, so not all operations may have been completed in as few instructions as possible.
* Review the section above discussing how the "r" registers are related to the "e" registers.

.file "exampleIML-1b.c"

.section .rodata

.LC0:

.string "%d + %d = %d\n"

.text

.globl main

.type main, @function

main:

.LFB0:

.cfi\_startproc

pushq %rbp // set up stack pointer = %rbp

.cfi\_def\_cfa\_offset 16

.cfi\_offset 6, -16

movq %rsp, %rbp // set up frame pointers

.cfi\_def\_cfa\_register

subq $16, %rsp // moves rsp down 16 registers for space

movl $1324, -4(%rbp) // initialize a

movl $5657, -8(%rbp) // initialize b

movl $9876, -12(%rbp) // initialize difference

movl $2221, -16(%rbp) // initialize prinf\_answer

movl -8(%rbp), %eax // access the b variable into %eax

movl -4(%rbp), %edx // access the a variable into %edx

subl %eax, %edx // the actual subtraction (subtracting b from a)

movl %edx, %eax // subtracted value placed in %edx, moving it to %eax

movl %eax, -12(%rbp) // move the value into the register local difference

movl -12(%rbp), %ecx // moving the differnce into %ecx

movl -8(%rbp), %edx // reload b with original value

movl -4(%rbp), %eax // reload a with original value

movl %eax, %esi // move a into %esi

movl $.LC0, %edi // special for printf function

movl $0, %eax // move ZERO value into %eax to clear

call printf // system call for printf

movl %eax, -16(%rbp) // pulling value for %d in printf

movl -12(%rbp), %eax // pulling value for %d in printf

leave

.cfi\_def\_cfa 7, 8

ret

.cfi\_endproc

.LFE0:

.size main, .-main

.ident "GCC: (GNU) 4.8.5 20150623 (Red Hat 4.8.5-4)"

.section .note.GNU-stack,"",@progbits

1. Using Table 2-2, identify the addressing mode that corresponds to each of the four possible values of Mod. Hint: Look at the operands in the "Effective Address" column. If you saw EBP or %ebp in an assembly instruction, where would the data for that instruction come from? What addressing mode does that correspond do? How about if you saw either [EBP]-16 (which gcc writes as -16(%ebp))?

00 - Register indirect addressing mode with no displacement

01 - one-byte signed displacement

10 - four-byte signed displacement

11 - register addressing mode

%epb would be direct register access to the value inside the register.

[EBP] would be memory location of the register. So [EBP] -16 would be a 16 bit offset of the register memory location.

Dump of assembler code for function main:

(7)0x0000000000400530 <+0>: push %rbp

(6)0x0000000000400531 <+1>: mov %rsp,%rbp

0x0000000000400534 <+4>: sub $0x10,%rsp

0x0000000000400538 <+8>: movl $0x52c,-0x4(%rbp)

(1)0x000000000040053f <+15>: movl $0x1619,-0x8(%rbp)

0x0000000000400546 <+22>: movl $0x2694,-0xc(%rbp)

0x000000000040054d <+29>: movl $0x8ad,-0x10(%rbp)

(2)0x0000000000400554 <+36>: mov -0x8(%rbp),%eax

0x0000000000400557 <+39>: mov -0x4(%rbp),%edx

0x000000000040055a <+42>: sub %eax,%edx

0x000000000040055c <+44>: mov %edx,%eax

(3)0x000000000040055e <+46>: mov %eax,-0xc(%rbp)

0x0000000000400561 <+49>: mov -0xc(%rbp),%ecx

0x0000000000400564 <+52>: mov -0x8(%rbp),%edx

0x0000000000400567 <+55>: mov -0x4(%rbp),%eax

0x000000000040056a <+58>: mov %eax,%esi

0x000000000040056c <+60>: mov $0x400620,%edi

(4)0x0000000000400571 <+65>: mov $0x0,%eax

0x0000000000400576 <+70>: callq 0x400410 <printf@plt>

0x000000000040057b <+75>: mov %eax,-0x10(%rbp)

0x000000000040057e <+78>: mov -0xc(%rbp),%eax

(5)0x0000000000400581 <+81>: leaveq

0x0000000000400582 <+82>: retq

End of assembler dump.

4. Notice that the push instruction is only one byte long. How did the designers squeeze both the opcode and the operator into one byte?

The designers were able to incorporate the opcode and the operator into one byte by including the rbp in the op, to be able to utilize the stack.

5. When using Table 2-2, sometimes the y-axis refers to the source operand, and sometimes it refers to the destination. How can you tell whether the y-axis refers to the first or second operator? Hint: Compare instructions main+36 and main+46.

You can tell by the differences between the two instructions between loading into a register or storing a value into memory. The op code is what determins this.

6. How/where does instruction main+15 encode that one of the parameters is an immediate value? How is the R/M byte for this instruction used?

The op code tells the machine that it is an immeditate value. And the aR/M byte tells the machine the offset.

7. Extra Credit: Explain how the IA64 machine language encodes the "r" registers (r8d, r9d, ..., r15d). Your explanation should include a table for this instruction: sub %r11d, %r14d