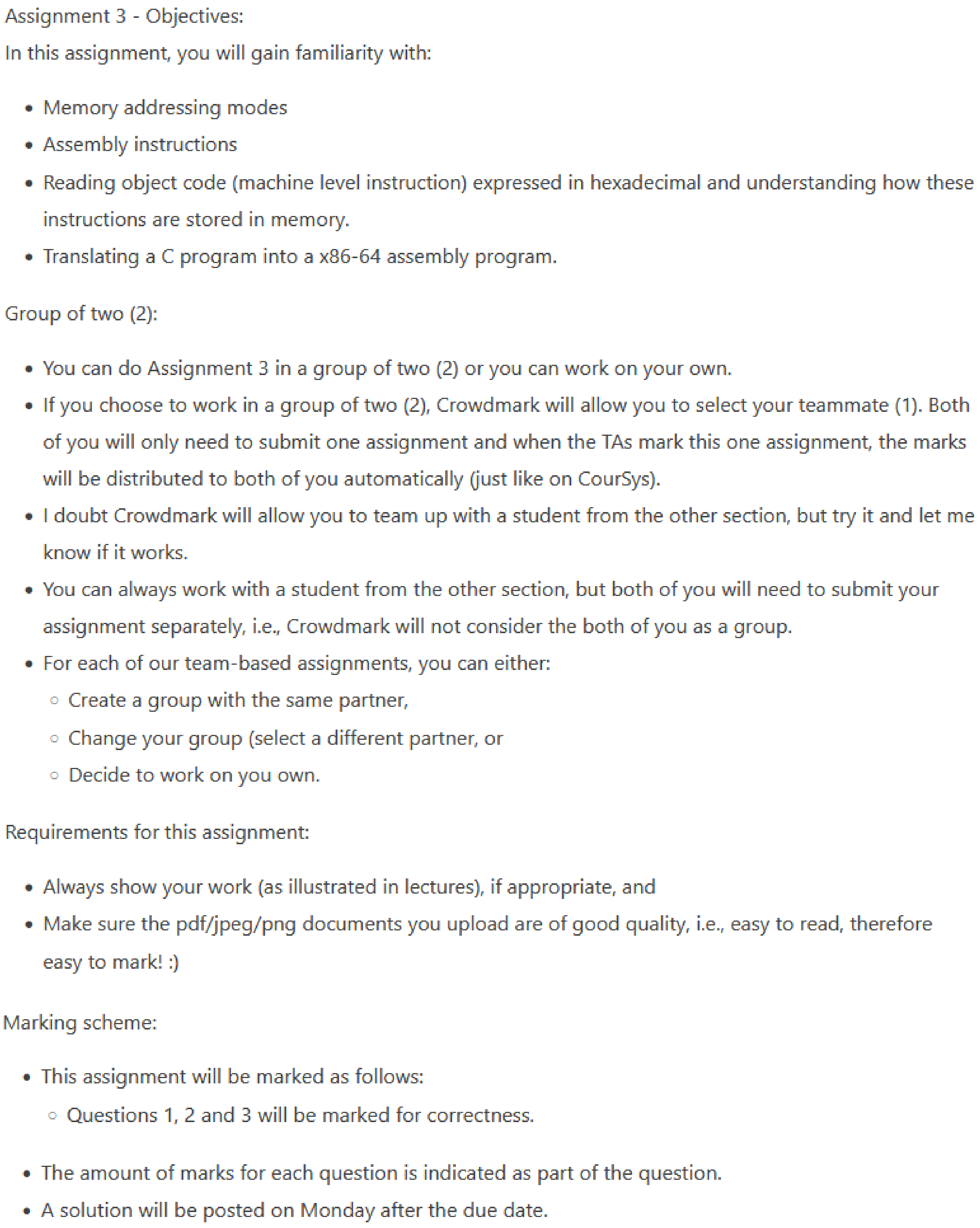
Assignment 3 - Solution



Assignment 3 - Objectives:

In this assignment, you will gain familiarity with:

- Memory addressing modes
- Assembly instructions
- Reading object code (machine level instruction) expressed in hexadecimal and understanding how these instructions are stored in memory.
- Translating a C program into a x86-64 assembly program.

Group of two (2):

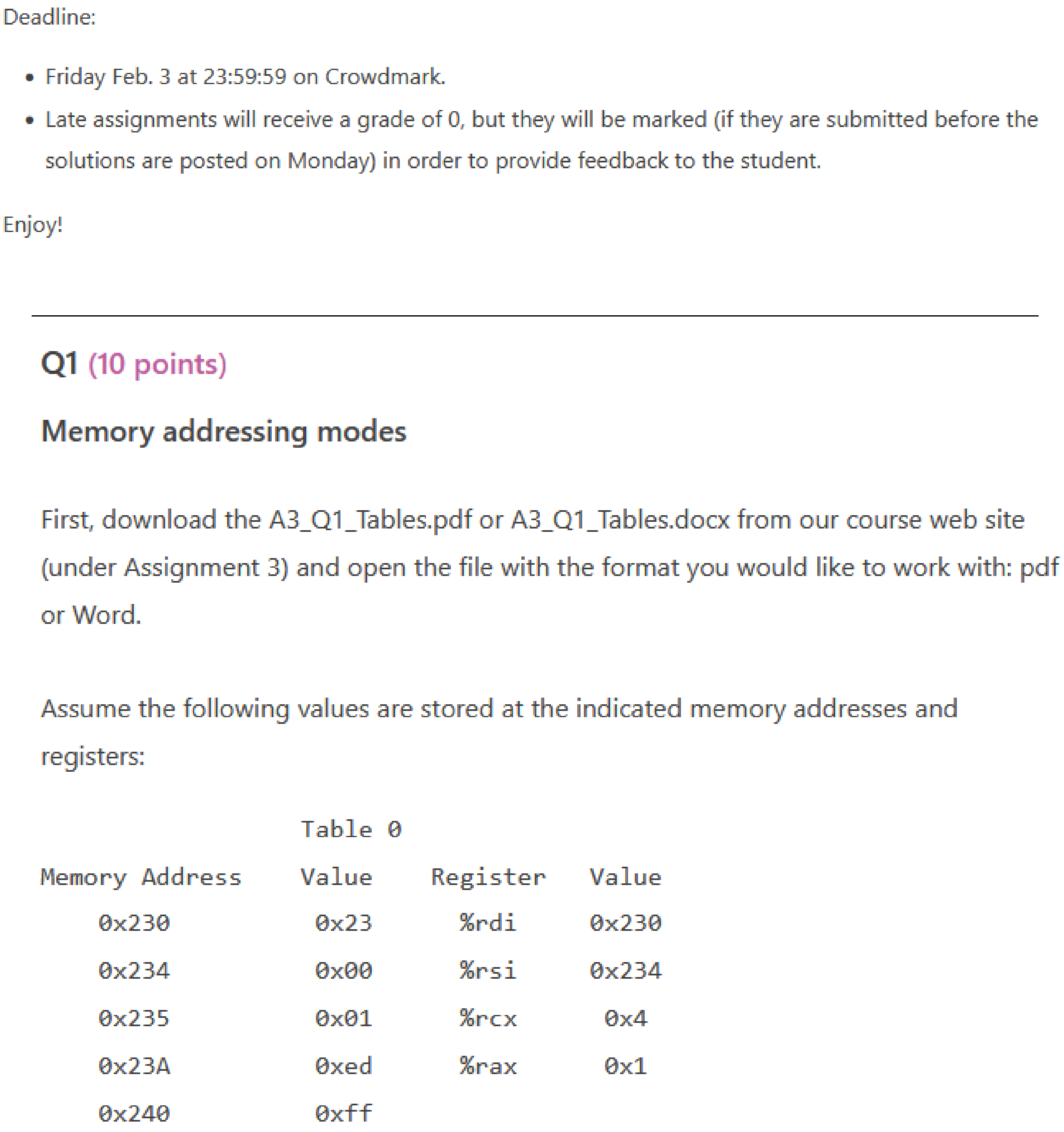
- You can do Assignment 3 in a group of two (2) or you can work on your own.
- If you choose to work in a group of two (2), Crowdmark will allow you to select your teammate (1). Both of you will only need to submit one assignment and when the TAs mark this one assignment, the marks will be distributed to both of you automatically (just like on CourSys).
- I doubt Crowdmark will allow you to team up with a student from the other section, but try it and let me know if it works.
- You can always work with a student from the other section, but both of you will need to submit your assignment separately, i.e., Crowdmark will not consider the both of you as a group.
- For each of our team-based assignments, you can either:
  - Create a group with the same partner,
  - Change your group (select a different partner, or
  - Decide to work on you own.

Requirements for this assignment:

- Always show your work (as illustrated in lectures), if appropriate, and
- Make sure the pdf/jpeg/png documents you upload are of good quality, i.e., easy to read, therefore easy to mark! :)

Marking scheme:

- This assignment will be marked as follows:
  - Questions 1, 2 and 3 will be marked for correctness.
- The amount of marks for each question is indicated as part of the question.
- A solution will be posted on Monday after the due date.



Deadline:

- Friday Feb. 3 at 23:59:59 on Crowdmark.
- Late assignments will receive a grade of 0, but they will be marked (if they are submitted before the solutions are posted on Monday) in order to provide feedback to the student.

Enjoy!

Q1 (10 points)

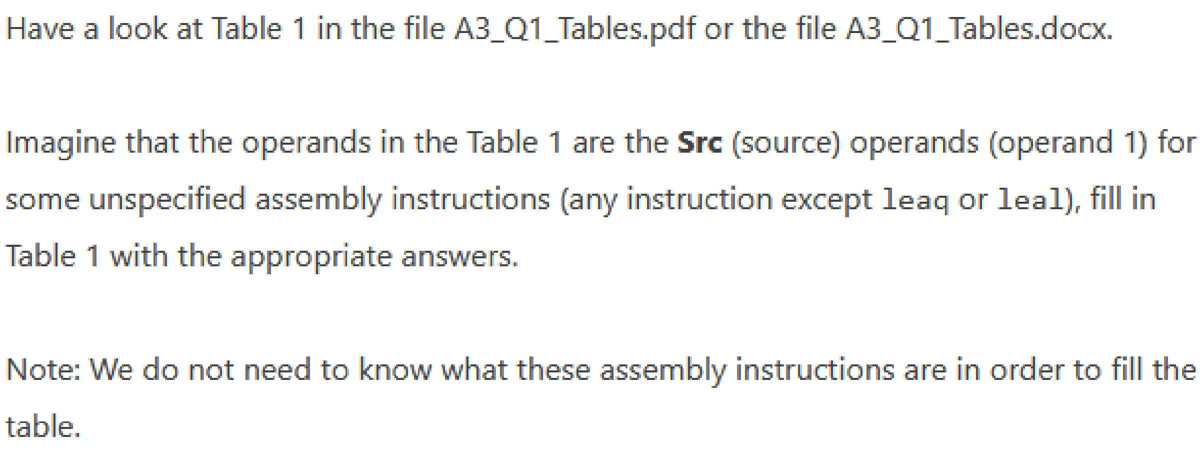
Memory addressing modes

First, download the A3_Q1_Tables.pdf or A3_Q1_Tables.docx from our course web site (under Assignment 3) and open the file with the format you would like to work with: pdf or Word.

Assume the following values are stored at the indicated memory addresses and registers:

Table 0

Memory Address	Value	Register	Value
0x230	0x23	%rdi	0x230
0x234	0x00	%rsi	0x234
0x235	0x01	%rcx	0x4
0x23A	0xed	%rax	0x1
0x240	0xff		



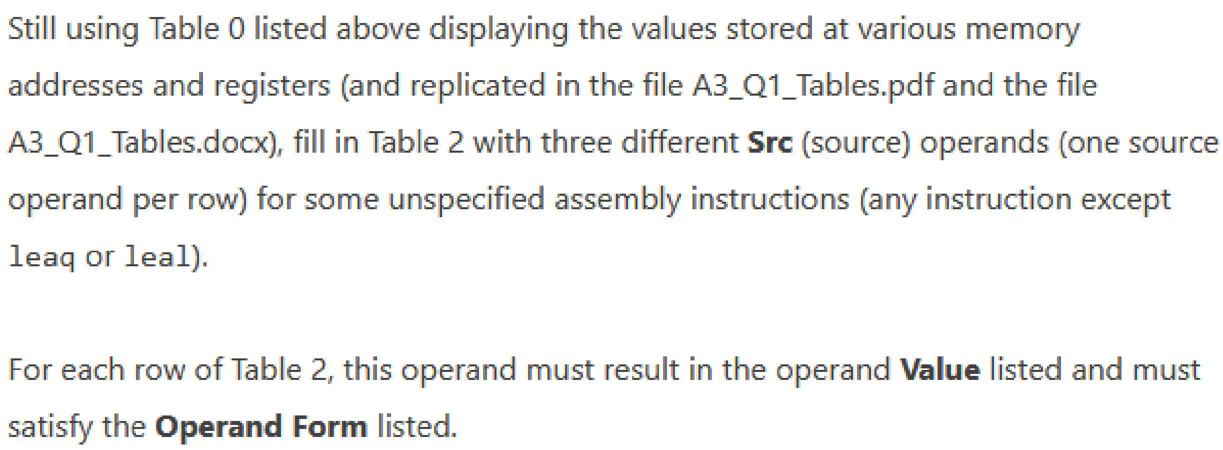
Have a look at Table 1 in the file A3_Q1_Tables.pdf or the file A3_Q1_Tables.docx.

Imagine that the operands in the Table 1 are the **Src** (source) operands (operand 1) for some unspecified assembly instructions (any instruction except `leaq` or `leal`), fill in Table 1 with the appropriate answers.

Note: We do not need to know what these assembly instructions are in order to fill the table.

Solution:

Operand	**Operand Value** (expressed in hexadecimal)	**Operand Form** (Choices are: Immediate, Register or one of the 9 Memory Addressing Modes)
`%rsi`	**0x234**	Register
`(%rdi)`	**0x23**	Indirect memory addressing mode
`$0x23A`	**0x23A**	Immediate value
`0x240`	`0xff`	Absolute memory addressing mode (this answer is preferable to “Imm” as it is more specific than “Imm” and highlights the fact that it does not require a “$” – see first row of table below)
`10(%rdi)`	**0xed**	“Base + displacement” memory addressing mode
`560(%rcx,%rax)`	**0x01**	Indexed memory addressing mode
`-550(, %rdi, 2)`	**0xed**	Scaled indexed memory addressing mode
`0x6(%rdi, %rax, 4)`	**0xed**	Scaled indexed memory addressing mode



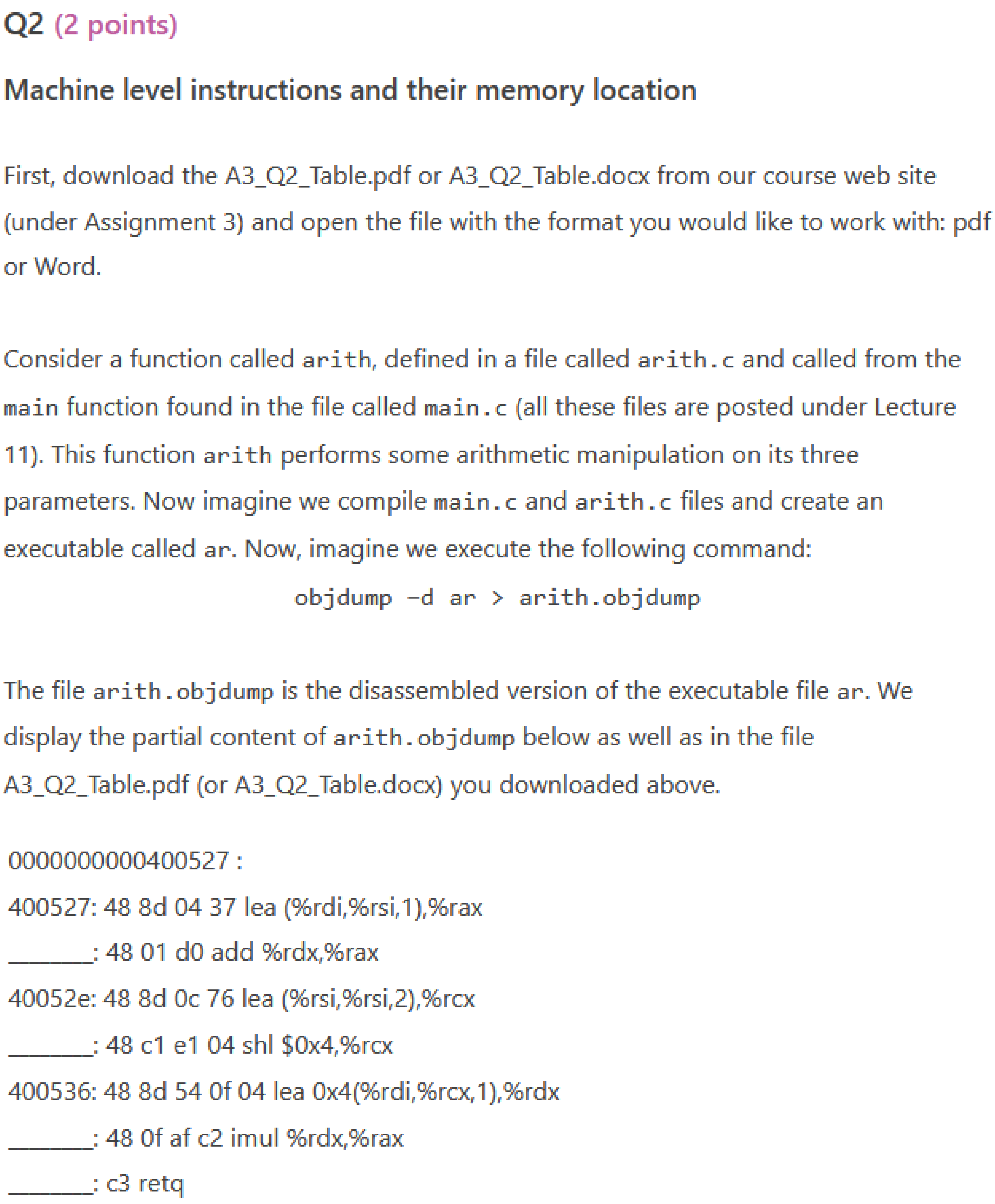
Still using Table 0 listed above displaying the values stored at various memory addresses and registers (and replicated in the file A3_Q1_Tables.pdf and the file A3_Q1_Tables.docx), fill in Table 2 with three different **Src** (source) operands (one source operand per row) for some unspecified assembly instructions (any instruction except `leaq` or `leal`).

For each row of Table 2, this operand must result in the operand **Value** listed and must satisfy the **Operand Form** listed.

Solution:

Operand	**Value**	**Operand Form** (Choices are: Immediate, Register or one of the 9 Memory Addressing Modes)
`0x234`	0x00	Absolute memory addressing mode
`(%rdi, %rax, 4)`	0x00	Scaled indexed memory addressing mode
`(%rdi, %rcx)`	0x00	Indexed memory addressing mode

Other answers are possible!



Q2 (2 points)

Machine level instructions and their memory location

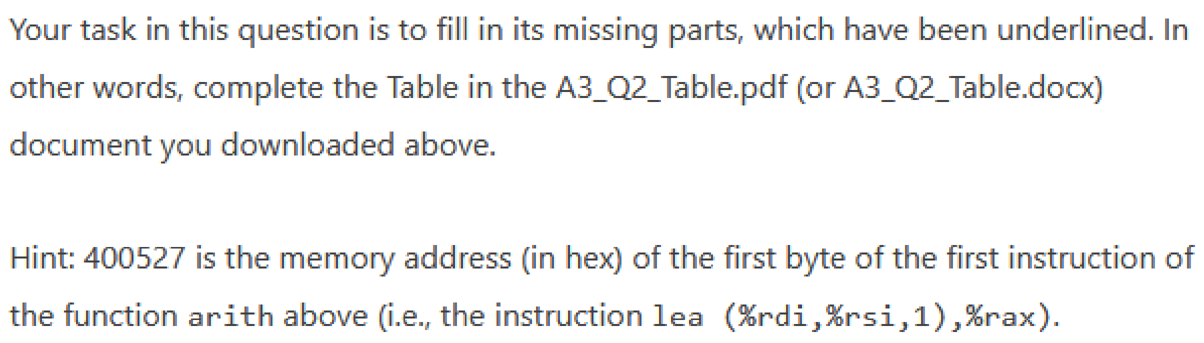
First, download the A3_Q2_Table.pdf or A3_Q2_Table.docx from our course web site (under Assignment 3) and open the file with the format you would like to work with: pdf or Word.

Consider a function called `arith`, defined in a file called `arith.c` and called from the `main` function found in the file called `main.c` (all these files are posted under Lecture 11). This function `arith` performs some arithmetic manipulation on its three parameters. Now imagine we compile `main.c` and `arith.c` files and create an executable called `ar`. Now, imagine we execute the following command:

```
objdump -d ar > arith.objdump
```

The file `arith.objdump` is the disassembled version of the executable file `ar`. We display the partial content of `arith.objdump` below as well as in the file A3_Q2_Table.pdf (or A3_Q2_Table.docx) you downloaded above.

```
0000000000400527 :
400527: 48 8d 04 37 lea (%rdi,%rsi,1),%rax
_______: 48 01 d0 add %rdx,%rax
40052e: 48 8d 0c 76 lea (%rsi,%rsi,2),%rcx
_______: 48 c1 e1 04 shl $0x4,%rcx
400536: 48 8d 54 0f 04 lea 0x4(%rdi,%rcx,1),%rdx
_______: 48 0f af c2 imul %rdx,%rax
_______: c3 retq
```

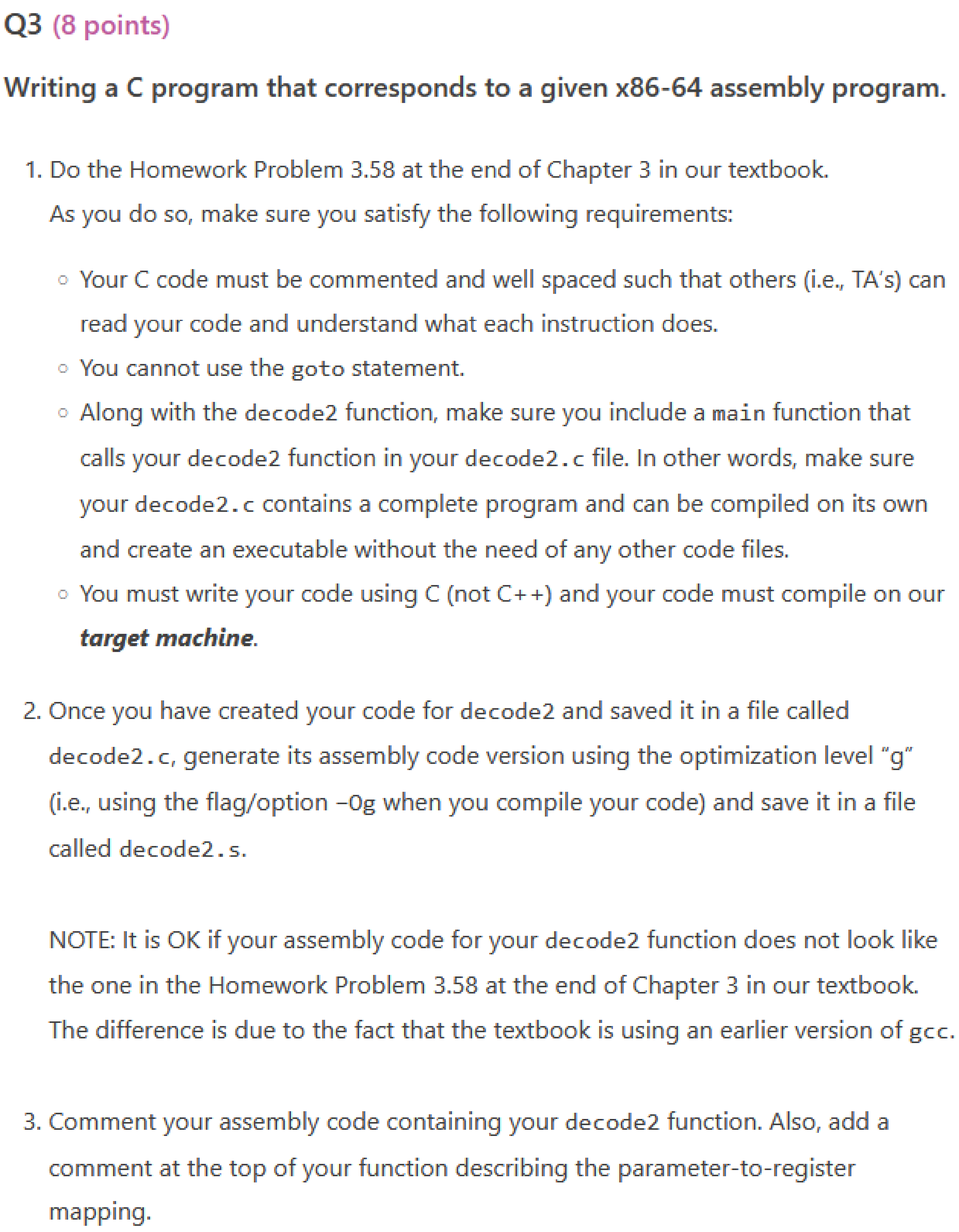


Your task in this question is to fill in its missing parts, which have been underlined. In other words, complete the Table in the A3_Q2_Table.pdf (or A3_Q2_Table.docx) document you downloaded above.

Hint: 400527 is the memory address (in hex) of the first byte of the first instruction of the function `arith` above (i.e., the instruction `lea (%rdi,%rsi,1),%rax`).

Solution:

```
0000000000400527 <arith>:
  400527:    48 8d 04 37          lea    (%rdi,%rsi,1),%rax
  40052b:    48 01 d0             add    %rdx,%rax
  40052e:    48 8d 0c 76          lea    (%rsi,%rsi,2),%rcx
  400532:    48 c1 e1 04          shl    $0x4,%rcx
  400536:    48 8d 54 0f 04       lea    0x4(%rdi,%rcx,1),%rdx
  40053b:    48 0f af c2          imul   %rdx,%rax
  40053f:    c3                   retq
```



Q3 (8 points)

Writing a C program that corresponds to a given x86-64 assembly program.

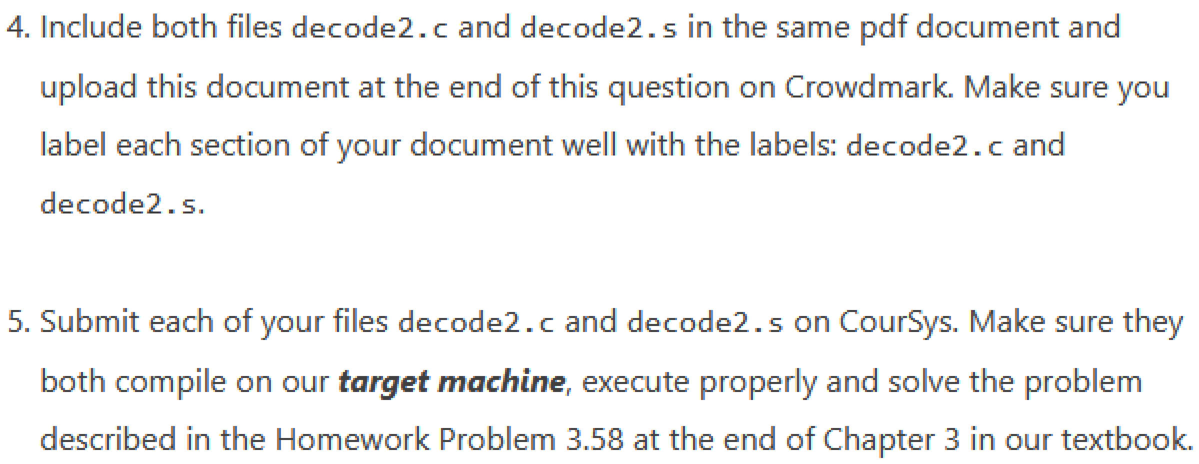
1. Do the Homework Problem 3.58 at the end of Chapter 3 in our textbook.
   As you do so, make sure you satisfy the following requirements:

   - Your C code must be commented and well spaced such that others (i.e., TA's) can read your code and understand what each instruction does.
   - You cannot use the `goto` statement.
   - Along with the `decode2` function, make sure you include a `main` function that calls your `decode2` function in your `decode2.c` file. In other words, make sure your `decode2.c` contains a complete program and can be compiled on its own and create an executable without the need of any other code files.
   - You must write your code using C (not C++) and your code must compile on our ***target machine***.

2. Once you have created your code for `decode2` and saved it in a file called `decode2.c`, generate its assembly code version using the optimization level "g" (i.e., using the flag/option –Og when you compile your code) and save it in a file called `decode2.s`.

   NOTE: It is OK if your assembly code for your `decode2` function does not look like the one in the Homework Problem 3.58 at the end of Chapter 3 in our textbook. The difference is due to the fact that the textbook is using an earlier version of `gcc`.

3. Comment your assembly code containing your `decode2` function. Also, add a comment at the top of your function describing the parameter-to-register mapping.



4. Include both files `decode2.c` and `decode2.s` in the same pdf document and upload this document at the end of this question on Crowdmark. Make sure you label each section of your document well with the labels: `decode2.c` and `decode2.s`.

5. Submit each of your files `decode2.c` and `decode2.s` on CourSys. Make sure they both compile on our ***target machine***, execute properly and solve the problem described in the Homework Problem 3.58 at the end of Chapter 3 in our textbook.

Possible Solution:

decode2.c

```
/*
 * Filename: decode2.c
 *
 * Description: Function decode2 along with its
 *              test driver for our A3 Q3.
 *
 * Auhtor: AL
 * Modification date: Feb. 2023
 */

#include <stdlib.h>  // atoi()
#include <stdio.h>   // printf()

long decode2(long x, long y, long z) {

  long t1 = y - z;
  long t2 = x * t1;
  // t3 is either 0 (if t1 is even) or -1 (if t1 is odd):
  long t3 = (t1 << 63) >> 63;
  // if t3 is -1, t2 is complemented (i.e., bits are flipped)
  // otherwise, t2 remains unchanged. In both cases, t2 -> t4
  long t4 = t3 ^ t2;
  return t4;
}

int main(int argc, char *argv[]) {
  long a = 0;
  long b = 0;
  long c = 0;
  long answer = 0;

  // Test case data entered at command line - expecting 3
numbers
  if (argc == 4) {
    a = atoi(argv[1]);
    b = atoi(argv[2]);
    c = atoi(argv[3]);

  // Call decode2
    answer = decode2(a, b, c);

  // Print test data and actual result
    printf("decode2(%ld, %ld, %ld) produces %ld as a
result.\n", a, b, c, answer);
  }
  else {
    printf("Must supply 3 long integers.\n");
    return 1;
  }
  return 0;
}
```

decode2.s

```
     .file          "decode2.c"
     .text
     .p2align 4
     .globl         decode2
     .type          decode2, @function
decode2:
     # x in %rdi, y in %rsi, z in %rdx
.LFB39:
     .cfi_startproc
     endbr64
     subq %rdx, %rsi            # t1 = y - z; t1 -> %rsi
     movq %rsi, %rax            # t1 -> %rax
     imulq %rdi, %rsi           # t2 = t1 * x; t2 -> %rsi
     salq $63, %rax             # LSb of t1 becomes its MSb + 63 0's
     sarq $63, %rax           # MSb of t1 fills t1; t3 -> %rax
     # if t1 was initially even, then t3 all 0’s (value = 0)
     # if t1 was initially odd, then t3 all 1’s (value = -1)
     xorq %rsi, %rax          # if t3 -1, t2 is complemented -> %rax
                           # otherwise, t2 remains unchanged -> %rax
     ret
     .cfi_endproc
.LFE39:
     .size     decode2, .-decode2
     .section  .rodata.str1.8,"aMS",@progbits,1
     .align 8
.LC0:
     .string   "decode2(%ld, %ld, %ld) produces %ld as a
result.\n"
     .section  .rodata.str1.1,"aMS",@progbits,1
.LC1:
     .string   "Must supply 3 long integers."
     .section  .text.startup,"ax",@progbits
     .p2align 4
     .globl    main
     .type     main, @function
main:
.LFB40:
     .cfi_startproc
     endbr64
     pushq     %r13
     .cfi_def_cfa_offset 16
     .cfi_offset 13, -16
     pushq     %r12
     .cfi_def_cfa_offset 24
     .cfi_offset 12, -24
     pushq     %rbx
     .cfi_def_cfa_offset 32
     .cfi_offset 3, -32
     cmpl $4, %edi
     jne  .L4
     movq 8(%rsi), %rdi
     movq %rsi, %rbx
     movl $10, %edx
     xorl %esi, %esi
     call strtol@PLT
     movq 16(%rbx), %rdi
     movl $10, %edx
     xorl %esi, %esi
     movslq    %eax, %r12
     call strtol@PLT
     movq 24(%rbx), %rdi
     movl $10, %edx
     xorl %esi, %esi
     movslq    %eax, %r13
     call strtol@PLT
     movq %r13, %rcx
     movq %r12, %rdx
     movl $1, %edi
     movslq    %eax, %r8
     movq %r13, %rax
     leaq .LC0(%rip), %rsi
     subq %r8, %rax
     movq %rax, %r9
     imulq     %r12, %rax
     salq $63, %r9
     sarq $63, %r9
     xorq %rax, %r9
     xorl %eax, %eax
     call __printf_chk@PLT
     xorl %eax, %eax
.L3:
     popq %rbx
     .cfi_remember_state
     .cfi_def_cfa_offset 24
     popq %r12
     .cfi_def_cfa_offset 16
     popq %r13
     .cfi_def_cfa_offset 8
     ret
.L4:
     .cfi_restore_state
     leaq .LC1(%rip), %rdi
     call puts@PLT
     movl $1, %eax
     jmp  .L3
     .cfi_endproc
.LFE40:
     .size     main, .-main
     .ident    "GCC: (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0"
     .section  .note.GNU-stack,"",@progbits
     .section  .note.gnu.property,"a"
     .align 8
     .long      1f - 0f
     .long      4f - 1f
     .long      5
0:
     .string    "GNU"
1:
     .align 8
     .long      0xc0000002
     .long      3f - 2f
2:
     .long      0x3
3:
     .align 8
4:
```

Other (earlier) versions of gcc/Ubuntu such as:
gcc 7.5.0, Ubuntu 7.5.0-3
OR
gcc: 9.3.0 Ubuntu 9.3.0-17ubuntu1~20.04)
not accepted! ☹