METRO STATE
UNIVERSITY

ICS 232 Computer Organization & Architecture
Project 2 - 100 points + 20 Bonus Points
Due Date: 7/26/2023

Name:

Note: Post your submission to ICS232 D2L site on or before the due date.

Programming Assignment 2:

The purpose of this project is to gain a greater understanding of the Intel 32-bit instruction set and understand how a compiler translates C code into assembly language. By compiling the program in unoptimized mode you will hopefully see a fairly clear translation. When running in optimization mode, you will see how well compilers can optimize your code.

To compile an unoptimized version use:

```
gcc -Wa,-adhln -g -masm=intel -m32 "Project 2.c" > "Project 2-g.asm"
```

This will write the unoptimized inter-mixed source and assembly code Project 2-g.asm.

To compile an optimized version use:

```
gcc -Wa,-adhln -O -masm=intel -m32 "Project 2.c" > "Project 2-o.asm"
```

This will write the optimized inter-mixed source and assembly code Project 2-o.asm.

The `-Wa,-adhln` option causes gcc to generate intermixed source and assembly code.

The `-masm=intel` option causes gcc to generate assembly code in intel format.

The `-m32` generates 32-bit code.

The `-g` option generates unoptimized code while `-O` generates optimized code.

The web site Godbolt.org can also be used to compile the program. Make sure you use the correct compiler options as shown above.

The generated code includes quite a few directives that can be ignored. Most of the directives begin with a period (.). There are also a few `call` instructions to procedures that you may not know what they mean that can be ignored. For example:

```
call  __x86.get_pc_thunk.bx
add   ebx, OFFSET FLAT:_GLOBAL_OFFSET_TABLE_
```

METRO STATE
UNIVERSITY

ICS 232 Computer Organization & Architecture
Project 2 - 100 points + 20 Bonus Points
Due Date: 7/26/2023

In all of the following questions, the line numbers refer the line numbers in the C code.

The following questions should be answered using the unoptimized code.

1. Describe the stack at line 25 `(return (sum);)`. The bottom of the stack is the first row below. The top of the stack will be the last item. The first few items are provided. You will probably need to add some additional rows to the table below.

Description	**Value**
`First argument to main`	`argv`
`Second argument to main`	`argc`
`Return address of main`	`Caller's return address`
`C runtime value of bp`	`Caller's bp register`
`Local variable i in main [bp-12]`	`5`
`Local variable sum in function1[bp-16]`	`sum`
`Array values in function1[bp-36]`	`{0,0,0,0,0,0,0,0,0,0}`
`Local variable I in function1[bp-40]`	`0`

2. In detail explain the code generated for line 105 `(k = function1(i, j);)`. For this and all further questions in which you are asked to explain the code generated in detail, I expect you to copy the generated code and add a comment for each line. For example:

309 01e4 83EC08 sub esp, 8 //subtracts 8 from the esp stack
310 01e7 FF75C4 push DWORD PTR -60[ebp] //pushes -60[ebp] onto the stack, adds the -60 to the base pointer
311 01ea FF75C0 push DWORD PTR -64[ebp] //pushes -64[ebp] onto the stack

METRO STATE
UNIVERSITY

ICS 232 Computer Organization & Architecture
Project 2 - 100 points + 20 Bonus Points
Due Date: 7/26/2023

312 01ed E80EFEFF call function1 //address of the next instruction after the call instruction is pushed onto the stack as the return address. The program will return the current execution point after the function completes

313 01f2 83C410 add esp, 16 //adds 16 to the stack pointer esp
314 01f5 8945C8 mov DWORD PTR -56[ebp], eax //moves the value of the tax register into the memory location -56[ebp], the instruction will store the result of function1 into the memory location

3. In detail explain the code generated for function2 (line 29 through 49).

32:Project 2.c **** int i; // below lines are for variable declaration
33:Project 2.c **** int sum1;
34:Project 2.c **** int sum2;
35:Project 2.c **** int v;
36:Project 2.c ****
37:Project 2.c **** sum1 = 0; // assigned the value 0 to the variable sum1, moves the value 0 into the memory location -12[ebp]
83 .loc 1 37 7
84 0085 C745F400 mov DWORD PTR -12[ebp], 0
84 000000
38:Project 2.c **** sum2 = 0; //assigns the value 0 into the variable sum2, moves value 0 into the memory location -8[ebp]
85 .loc 1 38 7
86 008c C745F800 mov DWORD PTR -8[ebp], 0
86 000000
39:Project 2.c **** for (i = 0; i < valuesLen; i++) { //loop initialization and condition
87 .loc 1 39 9
88 0093 C745F000 mov DWORD PTR -16[ebp], 0 //moves the value 0 into the memory location -16[ebp]
88 000000
89 .loc 1 39 2
90 009a EB2C jmp .L7
91 .L10: //labels the beginning of the loop body

ICS 232 Computer Organization & Architecture
Project 2 - 100 points + 20 Bonus Points
Due Date: 7/26/2023

```
40:Project 2.c   ****       v = values[i];       //loads the value of the array values at index I toto the variable v
92                  .loc 1 40 13
93 009c 8B45F0       mov   eax, DWORD PTR -16[ebp]   //moves the value of I into the eax register
94 009f 8D148500     lea   edx, 0[0+eax*4]          //multiplies the address offset based on tax by 4 and store into the edx register, occupies 4 bytes
94      000000
95 00a6 8B4508       mov   eax, DWORD PTR 8[ebp]       // retrieves the value of valuesLen and moves it into eax register
96 00a9 01D0         add   eax, edx   //adds the offset edx into eax, and store its value into the eax register
97                  .loc 1 40 5
98 00ab 8B00         mov   eax, DWORD PTR [eax]          //loads the value of eax register into eax register, stores the value of eax into the memory location -4[ebp]
99 00ad 8945FC       mov   DWORD PTR -4[ebp], eax     //compares the value stored inside -4[ebp] with 0
41:Project 2.c   ****       if (v > 0)
100                 .loc 1 41 6
101 00b0 837DFC00    cmp   DWORD PTR -4[ebp], 0
102 00b4 7E08        jle   .L8          //conditional jump instruction, if the previous comparison result is less than or equal to zero. Checks
42:Project 2.c   ****         sum1 += v;
103                 .loc 1 42 9
104 00b6 8B45FC      mov   eax, DWORD PTR -4[ebp]   //loads the value stored at the memory location -4[ebp] into the eax register. 105 00b9 0145F4
add   DWORD PTR -12[ebp], eax   //The value of eax is added to the value at the memory location -12[ebp]. Updates the value of sum1 by adding v to it

106 00bc EB06        jmp   .L9        //unconditional jump instruction, skips the code in the else block and continues with the next iteration of the loop
107               .L8:
43:Project 2.c   ****       else
44:Project 2.c   ****         sum2 += v;
```

```
 108                 .loc 1 44 9
 109 00be 8B45FC        mov   eax, DWORD PTR -4[ebp]    //loads the value stored at the memory location -4[ebp] into the eax register
 110 00c1 0145F8        add   DWORD PTR -8[ebp], eax        //add the value of eax register with memory location -4[ebp], and store the value at the memory location -8[ebp]. Updates the value of sum2 by adding v to it
 111               .L9:
  39:Project 2.c   ****       v = values[i];
 112                 .loc 1 39 30 discriminator 2
 113 00c4 8345F001      add   DWORD PTR -16[ebp], 1       //increments the value stored at memory location -16[ebp] by 1
 114               .L7:     //labels the beginning of the loop iteration
  39:Project 2.c   ****       v = values[i];
 115                 .loc 1 39 2 discriminator 1   //indicates the source code and debug information
 116 00c8 8B45F0        mov   eax, DWORD PTR -16[ebp]   //retrieve the variable stored at -16[ebp] and compare it with the memory location 12[ebp]
 117 00cb 3B450C        cmp   eax, DWORD PTR 12[ebp]    //compare the processor's flags based on the comparison
 118 00ce 7CCC          jl .L10          //conditional jump instruction
  45:Project 2.c   ****     }
  46:Project 2.c   ****
  47:Project 2.c   ****     return (sum1 + sum2);
 119                 .loc 1 47 15
 120 00d0 8B55F4        mov   edx, DWORD PTR -12[ebp]  //moves the value of sum1 into the edx register
 121 00d3 8B45F8        mov   eax, DWORD PTR -8[ebp]     //value of sum2 into the eax register
 122 00d6 01D0          add   eax, edx  //adds the value of edx into the eax register, storing the result into the eax register
  48:Project 2.c   ****
  49:Project 2.c   **** }
```

4. In detail explain the code generated for function5 (line 80 through 93).

METRO STATE UNIVERSITY

ICS 232 Computer Organization & Architecture
Project 2 - 100 points + 20 Bonus Points
Due Date: 7/26/2023

The following questions should be answered using the optimized code.

112 007f 89C2 mov edx, eax //moves the values of the eax register into the edx register
113 0081 C1E204 sal edx, 4 //performs left shift operations on the edx register value, shifts the value by 4 bits, result stored in the edx register.
114 0084 89C1 mov ecx, eax //the instruction moves the values of the eax register into the ecx register
115 0086 C1E903 shr ecx, 3 //perform the right shift operation on the value of the ecx register, the value shifts right by 3 bits and store value in ecx register.
116 0089 01CA add edx, ecx //adds the value in the ecx register to the edx register, storing the result in edx register
117 008b 83E007 and eax, 7 //performs bitwise and operation between the value in the eax register and value 7, put the value into at least three significant bits. Storing results in the eax register.
118 008e 01D0 add eax, edx //instruction adds the value in the edx register to the eax register value, storing the value to the eax register.

5. In detail explain the code generated for function1 (line 12 through 27). Why does this work? It has two parameters x and y which is the integer and returns the same datatype of datatype. While the loop is running of I from 0 to 10, in which I is incremented within every iteration. For each iteration, multiply the I value with 10 and multiply x with y, finally assigning the value to the array value[I]. The adding the values to the total sum, returning the sum after the loop terminates.

6. How are function calls optimized? For example, see lines 114 and 117. By many ways, innings, specialization, compiler optimization. When a function is inlined, the compiler replaces the function call with the actual code of the function, eliminating the function calling itself. If a function is called with constant arguments, the compiler may specialize the function call by generating versions of the function optimized for those specific arguments. Allowing the compiler to further optimize based on the known argument values. The compiler identifies opportunities for optimization. Using special techniques.

The following questions should be answered using by comparing the unoptimized code and the optimized code.

7. Compare the code generated by function4 in the unoptimized and optimized versions. Explain the optimizations.

Inlining: the unoptimized version suggests that the function call shouldn't be inclined. Inlining removes the overhead of the function call itself, improve performance.

Code simplification: the optimized version simplifies the code by removing the unnecessary variables. Optimized version directly returns the result of the condition, eliminating the need for temporary variable and simplifies the code structure.

Control flow optimization: optimized version has advantage of the "if-else" nature of each condition. Improve performance by reducing pipeline stalls and branch predictions.

Constant folding: in optimized version, the multiplication performs at compile time instead of runtime. The compiler evaluates the constant expressions and replace them with their respective results. Eliminating the need for the multiplication at runtime, resulting in faster execution.

Dead code elimination: the optimized version applys the dead code by removing unnecessary branches or statements unreachable. It eliminates the unreachable variables. The optimization reduces code size, eliminating unnecessary operations, improve speed.

Bonus – 10 points

In detail explain the code generated for function3. This a hard. Try your best. Why is the compiler doing this? You need to explain the purpose of each line of the generated assembly code.

```
 51:Project 2.c   **** static NOINLINE int function3(int x)
 52:Project 2.c   **** {
134                .loc 1 52 1
135                .cfi_startproc
136 00da F30F1EFB      endbr32
137 00de 55            push  ebp          //saves the pointer value onto the stack
138                .cfi_def_cfa_offset 8
139                .cfi_offset 5, -8
```

ICS 232 Computer Organization & Architecture
Project 2 - 100 points + 20 Bonus Points
Due Date: 7/26/2023

```
140 00df 89E5         mov   ebp, esp  //sets ebp to the current stack pointer, establishes access function parameter and local variables
141                   .cfi_def_cfa_register 5
142 00e1 83EC10       sub   esp, 16  //reserves 16 bytes of byte space on th stack for the local variables, providing storage room for variable y
143 00e4 E8FCFFFF     call  __x86.get_pc_thunk.ax
143      FF
144 00e9 05010000     add   eax, OFFSET FLAT:_GLOBAL_OFFSET_TABLE_       //calculates the address of the global offset table, need access global variables or function pointers
144      00
 53:Project 2.c   ****
 54:Project 2.c   ****    int y;
 55:Project 2.c   ****
 56:Project 2.c   ****    y = x / 20;
145                   .loc 1 56 4
146 00ee 8B4D08       mov   ecx, DWORD PTR 8[ebp]     //moves the values of x into the ecx register
147 00f1 BA676666     mov   edx, 1717986919     //initializaes the edx register with the constant value 1717986919
147      66
148 00f6 89C8         mov   eax, ecx  //copies the value of x from ecx to eax
149 00f8 F7EA         imul  edx       //performs signed multiplication of eax and edx
150 00fa C1FA03       sar   edx, 3    //performs arithmetic right shift of the value in edx by 3 bits
151 00fd 89C8         mov   eax, ecx  //stores the value x from ecx to eax
152 00ff C1F81F       sar   eax, 31   //shift value of eax register by 31 bits
153 0102 29C2         sub   edx, eax  //subtracts the value in eax from edx
154 0104 89D0         mov   eax, edx //copies the division result from edx to eax
155 0106 8945FC       mov   DWORD PTR -4[ebp], eax    //stores the value of y onto the stack at the memory location -4[ebp]
 57:Project 2.c   ****
 58:Project 2.c   ****    return (y);
```

METRO STATE UNIVERSITY

ICS 232 Computer Organization & Architecture
Project 2 - 100 points + 20 Bonus Points
Due Date: 7/26/2023

Bonus – 10 points

The file `strlen.asm` contains the Microsoft Visual Studio source code for the `strlen()` function. You need to explain the purpose of each line of the assembly code particularly lines 81 through 88.

Grading Criteria By Project Items:

1. Stack contents not defined well enough (-5 -> -10 points).
2. Call functon1 description (-5 -> -10 points).
3. Code description for function2 not complete enough (-5 -> -10 points).
4. Code description for function5 not complete enough (-5 -> -10 points).
5. Optimized function1 description incomplete (-5 -> -10 points).
6. Optimized function call description incomplete (-5 -> -10 points).
7. Function4 optimized description incomplete (-5 -> -10 points).

Late Submission Policy:

-10 points if submitted after the due date and an extension has been granted.

-20 points if submitted after the due date and an extension has not been granted.