**q1:**

noopt.asm:

; the string "A st" , including the null terminator, was previously

; stored in the variable $SG4294967291

; \_str$ is the offset of the beggining of str array (str array in the c program) from ebp

; copy the first 4 bytes of the variable $SG429496729

; which means it copied "A st" to ecx, without the null terminator  
mov ecx, DWORD PTR $SG4294967291

; copy the content of ecx to the beggining of str array

; which means copy "A st" to the beggining of str array

mov DWORD PTR \_str$[ebp], ecx

; copy the null terminator after "A st" to the lowest byte of edx

mov dl, BYTE PTR $SG4294967291+4

; copy the null terminator from dl to the fifth byte of str arry

; which means put a null terminator in str after "A st"

mov BYTE PTR \_str$[ebp+4], dl

opt.asm:

; Same as noopt.asm, except, it uses only eax without using ecx and edx registers.

ex1.s:

/NO\_APP

; notice that 1953701953 = 0x74732041

; in addition, in ASCII representation, 0x41 = 'A', 0x20 = ' ', 0x73 = 's', 0x74 = 't'

; that means that the constants 1953701953 represents the 4 chars - "A st"

lea eax, [ebp-208]

; calculates ebp-208 and store it in eax

mov DWORD PTR [eax], 1953701953

; copy "A st" (as previously explained) to the address stored in eax

; which means copy "A st" to the beggining of str array

mov BYTE PTR [eax+4], 0

; put null terminator at the fifth byte of str array => put null terminator after "A st"

**q2:**

noopt:

; put tv78[ebp] to point to the begging of the long string "A string..." and

; tv76[ebp] to point to the beggining of str array

mov DWORD PTR tv78[ebp], OFFSET $SG4294967289

lea edx, DWORD PTR \_str$[ebp]

mov DWORD PTR tv76[ebp], edx

$LL3@q2:

; compare the one char of the strings

mov eax, DWORD PTR tv76[ebp]

mov cl, BYTE PTR [eax]

mov BYTE PTR tv81[ebp], cl

mov edx, DWORD PTR tv78[ebp]

cmp cl, BYTE PTR [edx]

; different chars, put 1 or -1 according to carry flag

jne SHORT $LN4@q2

; see if we reached the null terminators (of both strings)

cmp BYTE PTR tv81[ebp], 0

; if so, return 0

je SHORT $LN5@q2

; otherwise, compare another char (same logic as the first char)

mov eax, DWORD PTR tv76[ebp]

mov cl, BYTE PTR [eax+1]

mov BYTE PTR tv88[ebp], cl

mov edx, DWORD PTR tv78[ebp]

cmp cl, BYTE PTR [edx+1]

jne SHORT $LN4@q2

add DWORD PTR tv76[ebp], 2

add DWORD PTR tv78[ebp], 2

cmp BYTE PTR tv88[ebp], 0

; if didn't reach null terminator, loop again

jne SHORT $LL3@q2

$LN5@q2:

; put 0 in the return value

mov DWORD PTR tv93[ebp], 0

jmp SHORT $LN6@q2

$LN4@q2:

; if the carry flag in on, the return value will be -1, otherwise 1

sbb eax, eax

or eax, 1

mov DWORD PTR tv93[ebp], eax

$LN6@q2:

; store the return value in the stack at offset \_rc$

mov ecx, DWORD PTR tv93[ebp]

mov DWORD PTR tv71[ebp], ecx

mov edx, DWORD PTR tv71[ebp]

mov DWORD PTR \_rc$[ebp], edx

opt:

; put the address of the variable ??\_C@... (which happens to store the "A string....")

; in eax

mov eax, OFFSET ??\_C@\_0DH@PFMIMKOJ@A?5string?5hjdhkjdhkdjhdkjhdkjhd?5d@

; put the address of str array in acx

lea ecx, DWORD PTR \_str$[ebp]

npad 6

$LL3@q2:

; compare the first byte of the strings

mov dl, BYTE PTR [ecx]

cmp dl, BYTE PTR [eax]

jne SHORT $LN4@q2

; see if we reached the null terminator, if so, return 0

test dl, dl

je SHORT $LN5@q2

; otherwise, compare the next char

mov dl, BYTE PTR [ecx+1]

cmp dl, BYTE PTR [eax+1]

jne SHORT $LN4@q2

; advance both pointers in 2 (becuase we already checked 2 chars)

add ecx, 2

add eax, 2

test dl, dl

jne SHORT $LL3@q2

$LN5@q2:

; put 0 in eax, which tells us the here calling convension uses eax

; to return a value (at least in this function)

xor eax, eax

jmp SHORT $LN6@q2

$LN4@q2:

; same logic as in noopt, put 1 or -1

sbb eax, eax

or eax, 1

$LN6@q2:

ex1.s:

; store the address of the hard coded string in the stack

mov DWORD PTR [esp+4], OFFSET FLAT:LC0

; store the address of str array in the stack

lea eax, [ebp-212]

mov DWORD PTR [esp], eax

; call strcmp

call \_strcmp

; move the return value of strcmp to eax

mov DWORD PTR [ebp-12], eax

**q3:**

noopt:

; loop that runs 50 times, and copies 4 bytes on each iteration from src to dst.

; it is done by decrementing ecx by 1 each time and repeating the movsd command.

; this command also increases esi and edi by 4 each iteration

mov ecx, 50 ; 00000032H

lea esi, DWORD PTR \_src$[ebp]

lea edi, DWORD PTR \_str$[ebp]

rep movsd

opt:

; same as noopt.asm

ex1.s:

; same as noopt, only the compiler copies addresses to edi and esi via some

; other registers, and initialize ecx via eax

**q4:**

noopt.asm:

; initialize i to 0

mov DWORD PTR \_i$[ebp], 0

jmp SHORT $LN3@q4

$LN2@q4:

; increment i by 1

mov eax, DWORD PTR \_i$[ebp]

add eax, 1

mov DWORD PTR \_i$[ebp], eax

$LN3@q4:

; compare i to 100, if equal or greater, finish iterating

cmp DWORD PTR \_i$[ebp], 100 ; 00000064H

jge SHORT $LN1@q4

; otherwise, add i to rc

mov ecx, DWORD PTR \_rc$[ebp]

add ecx, DWORD PTR \_i$[ebp]

mov DWORD PTR \_rc$[ebp], ecx

jmp SHORT $LN2@q4

$LN1@q4:

Opt.asm:

; here we can see the compiler does some magic

; and calculates 0+1+...+99 in 25 iterations instead of 100

; set eax, edx, esi, edi to 0

xor eax, eax

xor edx, edx

xor esi, esi

xor edi, edi

npad 2

$LL3@q4:

; edi(i) = i + 2\*(i-1)\*i

; esi(i) = 2\*i + 2\*(i-1)\*i

; edx(i) = 3\*i + 2\*(i-1)\*i

; ecx(i) = 2\*(i-1)\*i

inc edi

add esi, 2

add edx, 3

add ecx, eax

add edi, eax

add esi, eax

add edx, eax

add eax, 4

cmp eax, 100 ; 00000064H

jl SHORT $LL3@q4

; ecx = edi(25)+esi(25)+edx(25)+ecx(25)

; ecx = 1225 + 1250 + 1275 + 1200 = 4950 = 0+1+2+...+99

lea eax, DWORD PTR [edx+esi]

add eax, edi

add ecx, eax

ex1.s:

; iterates 100 times, and each iteration, adds the current number

; to a variable on a stack

mov DWORD PTR [ebp-12], 0

jmp L5

L6:

mov eax, DWORD PTR [ebp-12]

add DWORD PTR [ebp-16], eax

inc DWORD PTR [ebp-12]

L5:

cmp DWORD PTR [ebp-12], 99

jle L6

**q5:**

noopt.asm:

; 80 : switch (i)

mov eax, DWORD PTR \_i$[ebp]

mov DWORD PTR tv65[ebp], eax

mov ecx, DWORD PTR tv65[ebp]

sub ecx, 48 ; 00000030H

mov DWORD PTR tv65[ebp], ecx

; case i='u'

cmp DWORD PTR tv65[ebp], 69 ; 00000045H

; if tv65[ebp] is bigger than 69, it means i value is bigger

; than 'u' ascii value => which means none of the options

; are met, and the switch goes to deafult case

ja SHORT $LN1@q5

mov edx, DWORD PTR tv65[ebp]

; $LN18@q5 is a kind map array

; index i in the LN18@q5 array, contains index for the $LN19@q5 array,

; which in its turn, contains the address of

; the instructions handling the case of ascii char represented by i+48

; for example - 'C' char is represented by 67, which means index 67-48=19 in

; $LN18@q5 contains 4, LN19@5 at index 4 contains $LN11@q5

; we can see that the instructions starting at $LN11@q5 are handling case 'C'.

; we find that in all the chars which doesn't match any case we find index 12

; and in $LN19@q5 at index 12 leads to the default case

movzx eax, BYTE PTR $LN18@q5[edx]

jmp DWORD PTR $LN19@q5[eax\*4]

; all the cases

$LN13@q5:

mov DWORD PTR \_rc$[ebp], 1

jmp SHORT $LN14@q5

$LN12@q5:

mov DWORD PTR \_rc$[ebp], 9

jmp SHORT $LN14@q5

$LN11@q5:

mov DWORD PTR \_rc$[ebp], 8

jmp SHORT $LN14@q5

$LN10@q5:

mov DWORD PTR \_rc$[ebp], 7

jmp SHORT $LN14@q5

$LN9@q5:

mov DWORD PTR \_rc$[ebp], 6

jmp SHORT $LN14@q5

$LN8@q5:

mov DWORD PTR \_rc$[ebp], 5

jmp SHORT $LN14@q5

$LN7@q5:

mov DWORD PTR \_rc$[ebp], 4

jmp SHORT $LN14@q5

$LN6@q5:

mov DWORD PTR \_rc$[ebp], 11 ; 0000000bH

jmp SHORT $LN14@q5

$LN5@q5:

mov DWORD PTR \_rc$[ebp], 9

jmp SHORT $LN14@q5

$LN4@q5:

mov DWORD PTR \_rc$[ebp], 9

jmp SHORT $LN14@q5

$LN3@q5:

mov DWORD PTR \_rc$[ebp], 9

jmp SHORT $LN14@q5

$LN2@q5:

mov DWORD PTR \_rc$[ebp], 2

jmp SHORT $LN14@q5

$LN1@q5:

mov DWORD PTR \_rc$[ebp], 30 ; 0000001eH

$LN14@q5:

opt.asm:

; same as noopt.asm, except here the compiler joins the cases where

; the code puts the same value in rc, and some more minor diffrences

; such as the index to the addresses table is saved actually as an index

; and multiplied by 4 the get an offset

ex1.s:

; here we can see similar pattern to noopt.asm, with decreasing 48 from i

; and mapping all the possibilities between '0'-48 to 'u'-48, where chars that are

; not part of the switch leads to the default case

**Summary**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Noopt.asm** | **Opt.asm** | **Ex1.s** | **Differences** |
| **Q1** | Uses 2 registers  Stores string in memory | Uses only eax  Stores string in memory | Uses only eax, stores the string in hard coded hexa as immediate | Memory usage.  Registers usage |
| **Q2** | Uses temp values and registers | Uses registers only | Calls the \_strcmp function | Hard coded comparison vs function call |
| **Q3** | 50 iterations of loop. Each copies 4 bytes | Same as noopt | Same as other, minor changes in register usage | Minor changes |
| **Q4** | Naively calculates 1+2+3+…  Until 100 | Uses 4 registers to calculate the sum of 1..100 in 25 iterations, using 4 parallel calculations | Similar to noopt, but uses the stack instead of pointer to \_rc | Mathematical logic is different, adds parallelism. |
| **Q5** | Creates a memory address for each option and looks for match | Compiler joins similar cases to same code, preserves memory accesses | Similar to noopt. Minor changes in target calculation | Combines same logic to single code snippet, instead of code repetition |

חלק ג':

קובץ הנוצר ע"י GCC:

*LC0:*

*.ascii "OK\0" //****string for printing***

*LC1:*

*.ascii "Wrong ID\0" //* ***const string for printing***

*.text*

*.globl \_main*

*.def \_main; .scl 2; .type 32; .endef*

*\_main:*

*push ebp*

*mov ebp, esp*

*and esp, -16*

*sub esp, 48*

*call \_\_\_main*

*mov BYTE PTR [esp+47], 0*

*mov DWORD PTR [esp+26], 1835103337 //****decimal value for our “names string”***

*mov DWORD PTR [esp+30], 1718579809 //* ***continues here***

*mov WORD PTR [esp+34], 26994 //* ***finishes here***

*mov BYTE PTR [esp+36], 0 //* ***null terminator of “itamarofri” const string***

*mov eax, DWORD PTR \_\_imp\_\_\_iob //****input of stdin to fgets***

*mov DWORD PTR [esp+8], eax //****arguments for fgets***

*mov DWORD PTR [esp+4], 11 //****length of input as argument to fgets***

*lea eax, [esp+37]*

*mov DWORD PTR [esp], eax*

*call \_fgets*

*lea eax, [esp+26] //* ***calculates names array address***

*mov DWORD PTR [esp+4], eax //****push arguments to stack – “itamarofri” string address***

*lea eax, [esp+37]*

*mov DWORD PTR [esp], eax //****push input string address to stack***

*call \_strcmp*

*test eax, eax //****check result of strcmp***

*jne L2 //* ***if its not zero, jump to “wrong id” at L2***

*mov DWORD PTR [esp], OFFSET FLAT:LC0 //****gets here if the string are equal, pushes “ok” string address***

*call \_puts //****prints it***

*mov eax, 0 //* ***return value of “main”***

*jmp L4*

*L2:*

*mov DWORD PTR [esp], OFFSET FLAT:LC1 //****printing of “Wrong ID”***

*call \_puts*

*mov eax, 0*

*L4:*

*leave*

*ret*

קובץ הנוצר ע"י Visual Studio:

; 5 : int main() {

push ebp

mov ebp, esp

sub esp, 44 ; 0000002cH // **allocates stack memory for program**

; 6 : char input[11];

; 7 : input[10] = 0;

mov eax, 1

imul ecx, eax, 10

mov BYTE PTR \_input$[ebp+ecx], 0 // **null terminator for input string**

; 8 : char names[] = "itamarofri"; // **next command until section 9 does the insertion of the const string**

**“itamarofri” to names array**

mov edx, DWORD PTR $SG4294967291

mov DWORD PTR \_names$[ebp], edx

mov eax, DWORD PTR $SG4294967291+4

mov DWORD PTR \_names$[ebp+4], eax

mov cx, WORD PTR $SG4294967291+8

mov WORD PTR \_names$[ebp+8], cx

mov dl, BYTE PTR $SG4294967291+10

mov BYTE PTR \_names$[ebp+10], dl

; 9 : fgets(input, 11, stdin);

push 0

call DWORD PTR \_\_imp\_\_\_\_acrt\_iob\_func //**calling some function for stdin value**

add esp, 4

push eax

push 11 ; 0000000bH

lea eax, DWORD PTR \_input$[ebp] //**these commands prepare parameters for function fgets**

push eax

call DWORD PTR \_\_imp\_\_fgets

add esp, 12 ; 0000000cH

; 10 : if (!strcmp(input, names)) {

lea ecx, DWORD PTR \_names$[ebp]

mov DWORD PTR tv95[ebp], ecx

lea edx, DWORD PTR \_input$[ebp]

mov DWORD PTR tv93[ebp], edx

$LL4@main:

mov eax, DWORD PTR tv93[ebp]

mov cl, BYTE PTR [eax]

mov BYTE PTR tv131[ebp], cl

mov edx, DWORD PTR tv95[ebp]

cmp cl, BYTE PTR [edx]

jne SHORT $LN5@main

cmp BYTE PTR tv131[ebp], 0

je SHORT $LN6@main // **in case the null-terminator has been reached, jump to end (success)**

mov eax, DWORD PTR tv93[ebp]

mov cl, BYTE PTR [eax+1]

mov BYTE PTR tv138[ebp], cl

mov edx, DWORD PTR tv95[ebp]

cmp cl, BYTE PTR [edx+1]

jne SHORT $LN5@main // **found that string are not equal, jumps to LN5 then LN2, which leads to**

**“wrong” message, and exits program**

add DWORD PTR tv93[ebp], 2

add DWORD PTR tv95[ebp], 2

cmp BYTE PTR tv138[ebp], 0

jne SHORT $LL4@main

$LN6@main:

mov DWORD PTR tv143[ebp], 0 // **strings are equal, return 0 through stack, and jump to LN7 which is**

**success**

jmp SHORT $LN7@main

$LN5@main:

sbb eax, eax

or eax, 1

mov DWORD PTR tv143[ebp], eax

$LN7@main:

mov ecx, DWORD PTR tv143[ebp]

mov DWORD PTR tv79[ebp], ecx

cmp DWORD PTR tv79[ebp], 0

jne SHORT $LN2@main

; 11 : printf("OK\n");

push OFFSET $SG4294967290

call \_printf

add esp, 4

; 12 : return 0;

xor eax, eax

jmp SHORT $LN1@main

$LN2@main:

; 13 : }

; 14 : printf("Wrong ID\n");

push OFFSET $SG4294967289

call \_printf

add esp, 4

; 15 : return 0;

xor eax, eax

$LN1@main:

; 16 : }

לסיכום,

הקומפיילר יצר קובץ אסמבלי שבו מתבצעת השוואת מחרוזות ע"י לולאה והשוואת בית-בית עם מחרוזת הקלט.

מחרוזת המפתח (איחוד השמות) שמורה כ-"קבוע" (data) בזיכרון התוכנית. המחרוזת מועתקת למחסנית. מחרוזת הקלט נמצאת גם היא במחסנית.

לאחר סיום ההשוואה מתבצעת החלטה בהתאם לאיזה חלק בקוד לקפוץ, יעד הקפיצה הוא למקום בו מודפסת ההודעה המתאימה.

חלק ד':

.file "crackme.c"

.intel\_syntax noprefix

.def \_\_\_main; .scl 2; .type 32; .endef // **defines \_\_\_main as external**

.section .rdata,"dr"

LC0:

.ascii "Enter Serial Key\0"

LC1:

.ascii "%s\0"

.align 4

LC2:

.ascii "[-] Serial Key must be 9 chars long\0"

LC3:

.ascii "[-] Wrong\0"

LC4:

.ascii "[+] Good job ! \0"

.text

.globl \_main

.def \_main; .scl 2; .type 32; .endef

\_main:

LFB9:

push ebp

mov ebp, esp

and esp, -16

sub esp, 288 //**allocates stack memory**

call \_\_\_main

mov DWORD PTR [esp+284], 0

mov DWORD PTR [esp], OFFSET FLAT:LC0 //**move address of string LC0 for puts**

call \_puts //**prints “Enter Serial Key\0"**

lea eax, [esp+24]

mov DWORD PTR [esp+4], eax // **passes address of input string to scanf**

mov DWORD PTR [esp], OFFSET FLAT:LC1 // **”format string of scanf”**

call \_scanf

lea eax, [esp+24] //**loads the address of the input string from the stack**

mov DWORD PTR [esp+280], eax // **stores the address of the input string in esp+280**

mov eax, DWORD PTR [esp+280]

mov DWORD PTR [esp], eax

call \_strlen // **calls strlen on input string we got from scanf**

cmp eax, 9 // **checks if len is 9**

je L2

mov DWORD PTR [esp], OFFSET FLAT:LC2 // **pushes “serial key… 9 chars long” address to stack**

call \_puts // **prints that string**

mov eax, 0 // **return value of main**

jmp L7 // **goes to exit**

L2:

mov DWORD PTR [esp+284], 0

jmp L4

L6:

mov eax, DWORD PTR [esp+284]

lea edx, [eax+1]

mov eax, DWORD PTR [esp+280]

add eax, edx

mov al, BYTE PTR [eax]

movsx eax, al

mov ecx, DWORD PTR [esp+284]

mov edx, DWORD PTR [esp+280]

add edx, ecx

mov dl, BYTE PTR [edx]

movsx edx, dl

add edx, 2

cmp eax, edx

je L5

mov DWORD PTR [esp], OFFSET FLAT:LC3 // **pushes address of string “wrong”**

call \_puts // **prints “wrong”**

mov eax, 0 // **return value of main**

jmp L7 // **goes to exit**

L5:

inc DWORD PTR [esp+284]

L4:

cmp DWORD PTR [esp+284], 7

jle L6

mov DWORD PTR [esp], OFFSET FLAT:LC4 // **pushes** **“good job” string address to stack**

call \_puts // **prints good job**

mov eax, 0 // **return value of main**

L7:

leave

ret