

Forum:[Hack Forums](#)

Thread: [asm source code of the program written in C?](#)

SecuresSpy - July 14, 2018, 6:29 am

What is the assembly source code of the following C code?

```
char string[1001];

while(string[index] != '\0') {
    diff = string[index] - string[index - 1];
    if(diff < 0) {
        count += (26 + diff);
    } else {
        count += diff;
    }
    index += 1;
}
```

hints:

PHP Code

<?php

loc_4013B4:

```
1. lea    edx, [esp+13h]
2. mov    eax, [esp+408h]
3. add    eax, edx
4. movzx  eax, byte ptr [eax]
5. movsx  edx, al
6. mov    eax, [esp+408h]
7. sub    eax, 1
8. movzx  eax, byte ptr [esp+eax+13h]
9. movsx  eax, al
sub    edx, eax
mov    eax, edx
mov    [esp+400h], eax
cmp     dword ptr [esp+400h], 0
jns     short loc_401401
```

note:

line 1 loading address of array from stack.

what the 2 and 3 and others line are doing?

Thanks

Vanilla - July 27, 2018, 6:13 pm

(Jul 14, 2018 06:29 AM)SecuresSpy Wrote: [\[CITE\]](#) What is the assembly source code of the following C code?

```
char string[1001];
```

```

while(string[index] != '\0') {
    diff = string[index] - string[index - 1];
    if(diff < 0) {
        count += (26 + diff);
    } else {
        count += diff;
    }
    index += 1;
}

```

hints:

PHP Code

```
<?php
```

loc_4013B4:

```

1. lea    edx, [esp+13h]
2. mov    eax, [esp+408h]
3. add    eax, edx
4. movzx  eax, byte ptr [eax]
5. movsx  edx, al
6. mov    eax, [esp+408h]
7. sub    eax, 1
8. movzx  eax, byte ptr [esp+eax+13h]
9. movsx  eax, al
sub    edx, eax
mov    eax, edx
mov    [esp+400h], eax
cmp    dword ptr [esp+400h], 0
jns     short loc_401401

```

note:

line 1 loading address of array from stack.

what the 2 and 3 and others line are doing?

Thanks

Ok, your code was completely screwed up but added some stuff to make it compile.

C code: Code char string[1001];

```

int main() {
    int index;
    int diff;
    int count;
    while(string[index] != '\0') {
        diff = string[index] - string[index - 1];
        if (diff < 0) {
            count += (26 + diff);
        } else {
            count += diff;
        }
    }
    index += 1;
}

```

ASM code: Code main:

```

    push    rbp
    mov     rbp, rsp
    jmp     .L2
.L4:
    mov     eax, DWORD PTR [rbp-8]
    cdqe
    movzx   eax, BYTE PTR string[rax]
    movsx   edx, al
    mov     eax, DWORD PTR [rbp-8]
    sub     eax, 1
    cdqe
    movzx   eax, BYTE PTR string[rax]
    movsx   eax, al
    sub     edx, eax
    mov     eax, edx
    mov     DWORD PTR [rbp-12], eax
    cmp     DWORD PTR [rbp-12], 0
    jns     .L3
    mov     eax, DWORD PTR [rbp-12]
    add     eax, 26
    add     DWORD PTR [rbp-4], eax
    jmp     .L2
.L3:
    mov     eax, DWORD PTR [rbp-12]
    add     DWORD PTR [rbp-4], eax
.L2:
    mov     eax, DWORD PTR [rbp-8]
    cdqe
    movzx   eax, BYTE PTR string[rax]
    test    al, al
    jne     .L4
    add     DWORD PTR [rbp-8], 1
    mov     eax, 0
    pop     rbp
    ret

```

Enjoy!

DebugBSD - September 8, 2018, 10:00 am

Hi, good afternoon!

This is another version of the same algorithm. I've tested with random data and it seems to work properly.

Code .data

```

string BYTE 1001 DUP(0)
len EQU($ - string)

```

```

.code
main PROC

```

```

    mov RDI, OFFSET string
    mov RCX, 1001

```

```

; Load random data
L2:
    RDRAND RAX
    stosb
    loop L2

    mov RDI, OFFSET string
    mov RDX, 0
L1:
    scasb
    je EXIT
    mov RBX, [RDI+1]
    sub BL, BYTE PTR [RDI]
    AND RBX, 0FFh
    cmp BL, 0
    jae I1
    add BL, 26
    add RDX, RBX
I1:
    add RDX, RBX
    jmp L1

EXIT:
    call ExitProcess
main ENDP
Enjoy Assembly Language Programming!

```

Infamia - September 24, 2018, 8:59 am

I'll go off Hysteric's diassembly:

Code main:

```

    push    rbp
    mov     rbp, rsp
    jmp     .L2

```

Set up an x64 stack-frame, jump to the label .L2:

Code .L2:

```

    mov     eax, DWORD PTR [rbp-8]
    cdqe
    movzx   eax, BYTE PTR string[rax]
    test    al, al
    jne     .L4
    add     DWORD PTR [rbp-8], 1
    mov     eax, 0
    pop     rbp
    ret

```

Code mov eax, DWORD PTR [rbp-8]
cdqe

is kinda odd because it'd be better in the form

Code movq -8(%rbp), %rax

; mov rax, QWORD PTR[rbp-8]

but it's probably some intrinsic detail; regardless it just moves a locally allocated pointer to the

eax/rax register

Code .L2:

```
movzx  eax, BYTE PTR string[rax]
test   al, al
jne    .L4
```

```
add     DWORD PTR [rbp-8], 1
mov     eax, 0
pop     rbp
ret
```

moves (with zero-extension, i.e the high 4 bytes of %rax are zeroed) the rax'th index of the `string` into `eax` as a character pointer (BYTE has the same range as a character, from 0 to 255), if the lower byte of eax (that is, the character that has just been stored) is not zero (not the end of the string, since C strings are null-terminated by a character whose ordinal is 0x00) then jump to the actual loop, else, add 1 to the index and return 0 (eax is the return variable for the stack-frame in this convention).

note: the `add DWORD PTR [rbp-8], 1` is a flaw of the code, there shouldn't be an `index += 1;`, it doesn't make sense and has no purpose.

Code .L4:

```
mov     eax, DWORD PTR [rbp-8]
cdqe
```

```
movzx   eax, BYTE PTR string[rax]
movsx   edx, al
```

```
mov     eax, DWORD PTR [rbp-8]
sub     eax, 1
cdqe
```

```
movzx   eax, BYTE PTR string[rax]
movsx   eax, al
```

```
sub     edx, eax
```

```
mov     eax, edx
mov     DWORD PTR [rbp-12], eax
cmp     DWORD PTR [rbp-12], 0
```

```
jns     .L3
```

```
mov     eax, DWORD PTR [rbp-12]
add     eax, 26
add     DWORD PTR [rbp-4], eax
jmp     .L2
```

firstly the `index` is reloaded into `eax` for some reason, even though it was just loaded prior to that; probably due to a lack of optimization, then the character is loaded back into `eax` as last time, `movsx edx, al` moves the character with sign-extension (i.e., if al is negative then the high four bytes of %rax are 0xFFFF else 0x0000), afterwards the index is loaded again into `eax`, subtracted by 1, converted to a quadword/long long, then the character before is loaded into eax, the difference between the two characters is found by `sub edx, eax`, the result of the subtraction is in `eax`, the

old character is moved into the new character, then the character is stored in another locally allocated variable at ``rbp-12``, thus the character is compared to 0, if the comparison result of ``[rbp-12] - 0`` is not signed (i.e. the result is non-negative) then jump to `.L3`.

else if it's negative then move the old character into `eax`, add 26 to the character, and then add the previous addition to a new variable which i didn't know existed up until now, hereby reference this as the 1st locally stored variable at `rbp-4`, didn't take into account that they were `DWORD` sizes opposed to `QWORD` so I took a multiple of 8.

afterwards unconditionally jump back to the `.L2` label which checks for NULLs, if so, quit, else, quits.

EDIT:

On GCC 8.2 x86-64 with parameters ``-O3 -mmodel=large`` (we don't want rip relative addressing fuck that shit)

```
Code #include <cstdint>
#include <stdio>
```

```
int main(int argc, char **argv) {
    size_t index;
    size_t diff;
    size_t count;

    char* string = argv[1];

    while(string[index]) {
        diff = string[index] - string[index - 1];
        if (diff < 0) {
            count += (26 + diff);
        } else {
            count += diff;
        }
    }
}
```

disassembles to

Code main:

```
    movq    8(%rsi), %rax
    cmpb    $0, (%rax)
    je      .L2
.L3:
    jmp     .L3
.L2:
    xorl    %eax, %eax
    ret
```

which is just a bit shorter:

first we move the 1st argument from `argv[1]` into `rax`, compare the character at `rax` by `NULL`, if equal then jump into an infinite loop, else return 0.

points out the flaw in logic in the code Hysteric wrote.

macheti228 - November 20, 2018, 8:01 am
good information for ?? men