**MAX 1/1**

#include <stdio.h> //( 1)

int run(int level, int \* max, ...); //( 2)

int main(){ //( 3)

register int a = 1; //( 4)

int max =0.0, wynik=0.0; //( 5) nic specjalnego

wynik = run(5, &max,2,6,4,-2,10,5,3,0) //( 6) obliczenie wyniku

printf("%d %d\n", wynik, max); //( 7) wypisanie obliczenia i wybranej liczby

wynik = run(3, &max,2,6,4,-2,10,0); //( 8) obliczenie wyniku

printf("%d\n", wynik); //( 9) wypisanie obliczenia

printf("%d\n", a); //(10) zakładam że sprawdzenie czy coś się nie zmieniło w pamięci

} //(11)

// wynik:

// 21 5

// 20

// 1

BITS 32 ;( 1)

section .text ;( 2)

global \_run, run ;( 3)

\_run: ;( 4)

run: ;( 5)

enter 8, 0 ;( 6) tworzenie ramki stosu

mov [esp], ebx ;( 7)

mov edi, 0 ;( 8)

sub edx, edx ;( 9)

mov ecx, [ebp+8] ;(10)

lea esi, [ebp+16] ;(11) ustawianie adresu

cld ;(12) czyszczenie flag (direction)

.loop: ;(13) początek pętli

lodsd ;(14) ładuje 4 bajty z esi do eax i zwiększa (zależy od flagi direction) esi o 4

cmp eax, 0 ;(15)

je .end ;(16) wyjście z petli

cmp ecx, eax ;(17) ustawienie flag porównania

cmovg eax, edi ;(18) przenieś jeżeli większe (flagi z porównania powyżej)

cmovle ebx, eax ;(19) przenieś jeżeli mniejsze lub równe (flagi z porównania powyżej)

add edx, eax ;(20)

jmp .loop ;(21) koniec pętli

.end: ;(22)

mov esi, [ebp+12] ;(23) kopiowanie wyników - tak naprawdę max jest tam gdzie wywoła się ostatni raz instrukcja z lini 14

mov eax, edx ;(24)

mov [esi],ebx ;(25)

pop ebx ;(26)

leave ;(27) zakonczenie funkcji

ret ;(28)

;(29)

**FUNKCJA 1/2**

#include <stdio.h> //( 1)

int fun(int \* tab, int rozmiar); //( 2)

int main(){ //( 3)

const int rozmiar = 3; //( 4)

int tab[] = {1, 2, 3, 4, 5}; //( 5)

int i = 0; //( 6) nic specjalnego

int wynik = fun(tab, rozmiar); //( 7) wykonanie funkcji

for(i=0; i<rozmiar; i++) //( 8)

printf("%d ", tab[i]); //( 9) wypisanie danych z tablicy

printf("\n%d\n", wynik); //(10) wypisanie wyniku funkcji

} //(11)

// wynik

// 4 2 16

// 2

BITS 32 ;( 1)

section .data ;( 2)

dane1: dd 0, 0, 4 ;( 3)

dane2: dd 0, 100, 200 ;( 4)

section .text ;( 5)

change1: ;( 6)

push ebx ;( 7)

mov ebx, [esp+8] ;( 8)

lea eax, [1+ebx\*4+ebx] ;( 9) obliczenie adresu

pop ebx ;(10)

ret ;(11)

change2: ;(12)

mov eax, 2 ;(13)

ret ;(14)

change3: ;(15)

enter 0,0 ;(16) alokacja ramki stosu ilość bitów na zmienne, poziom zagłębienia procedury

shl eax, 2 ;(17) przesunięcie bitów w lewo w eax o 2 (puste miejsca wypełnione zerami)

leave ;(18) skasowanie ramki stosu

ret ;(19)

global \_fun, fun ;(20)

\_fun: ;(21) wejście do funkcji

fun: ;(22)

enter 24, 0 ;(23) alokacja ramki stosu ilość bitów na zmienne, poziom zagłębienia procedury

mov [esp], edi ;(24) operacje przygotowujące

mov [esp+4], esi ;(25)

mov [esp+8], ebx ;(26)

std ;(27) ustawianie flag (direction)

lea edi, [ebp-4] ;(28) ładuje obliczony adres do pamięci

mov esi, dane2 ;(29)

mov ecx, 3 ;(30)

rep movsd ;(31) kopiowanie tablicy z esi do edi o długości ecx

cld ;(32) czyszczenie flag (direction)

mov ecx, 3 ;(33)

rep lodsd ;(34) ładuje 4 bajty z esi do eax i zwiększa (zależy od flagi direction) esi o 4 (tyle razy ile wynosi ecx)

mov edi, [ebp+8] ;(35)

mov ecx, [ebp+12] ;(36)

mov ebx, eax ;(37)

label0: ;(38) początek pętli

lea eax, [ecx-1] ;(39) ładuje obliczony adres do pamięci

**FUNKCJA 2/2**

cdq ;(40) kopiuje najwyższy bit z eax do edx i powtarza go na każdym miejscu

idiv dword [jump\_size] ;(41) dzielenie ze znakiem edx:eax przez argument, wynik trafia do eax a reszta do edx

mov eax, [edi] ;(42)

neg ebx ;(43) negacja bitów

add [ebp + ebx - 8], eax ;(44)

;(45)

push eax ;(46) dodaj na stos

call [jump + edx\*4] ;(47) skocz do etykiety o numerze edx-1 (reszta zabiegów służy obliczeniu adresu)

add esp, 4 ;(48)

stosd ;(49) kopiuje 4 bajty z eax do edi i zwiększa (zależy od flagi direction) edi o 4

loop label0 ;(50) koniec pętli (zmniejsza ecx o 1, jeśli nie zero to skacze do podanej etykiety)

mov eax, [ebp-12] ;(51)

sub eax, [ebp-4] ;(52)

;(53)

pop edi ;(54) opróżnienie stosu

pop esi ;(55)

pop ebx ;(56)

leave ;(57) wyjście z funkcji, skasowanie ramki stosu

ret ;(58)

jump: dd change1, change2 ;(59) 'zmienne' wskazujące miejsce skoku

dd change3 ;(60)

jump\_size: dd 3 ;(61) 'zmienna' przechowująca 3

**WYMIANA 1/3**

#include <cstdio> //( 1)

using namespace std; //( 2)

extern "C" void wymiana(char\*, char, char, char); //( 3)

extern "C" void wypelnij(char, char\*); //( 4)

extern "C" int szukaj(char\*); //( 5)

//( 6)

int main() { //( 7)

char z = 'A'; //( 8)

char tab[]="12345678901234567890"; //( 9) nic specjalnego

wypelnij(z, tab); //(10) wykonanie funkcji

printf("[%s]\n", tab); //(11) wypisanie łańcucha znaków

printf("%d %d\n", szukaj(tab), szukaj(tab+9)); //(12) wypisanie wartości wyszukiwania jako liczba

char txt[]="To jest tekst probny"; //(13)

wymiana(txt, 'k','A'-'a','\*'); //(14) wykonanie funkcji

//wymiana(txt, 'f',1,'#'); //(15) wykonanie funkcji

printf("[%s]\n", txt); //(16) wypisanie łańcucha znaków

return 0; //(17)

} //(18)

// wynik

// [AAAAAAAAAAAAAAAA7890]

// 16 0

// [\*O\*\*\*ST\*T\*KST\*PRobny]

// a z odkomentowaną linią 15 ostatni napis to [################obny]

// sama linia 15 daje wynik [#p#k#tu#u#ltu#qsobny]

BITS 64 ;( 1)

;( 2)

section .text ;( 3)

global wypelnij, \_wypelnij, wymiana, \_wymiana, szukaj, \_szukaj ;( 4)

wypelnij: ;( 5) \*\*\* funkcja wypelnij start

\_wypelnij: ;( 6)

push rcx ;( 7)

xor rax, rax ;( 8)

mov al, dil ;( 9) This is enforced by changing (AH, BH, CH, DH) to (BPL, SPL, DIL, SIL) for instructions using a REX prefix. Cytat ze strony Intela

mov rdx, 3 ;(10) według wykładu DIL to 8 młodszych bitów rdi

mov cl, 8 ;(11)

.loop1: ;(12)

mov r9, rax ;(13)

shl rax, cl ;(14) przesunięcie bitów w lewo w eax o cl (puste miejsca wypełnione zerami)

add rax, r9 ;(15)

shl cl, 1 ;(16) przesunięcie bitów w lewo w eax o 1 (puste miejsca wypełnione zerami)

sub rdx, 1 ;(17)

jnz .loop1 ;(18) skok jeśli rdx nie równa się 0

mov rcx, 2 ;(19)

.loop2: ;(20)

dec rcx ;(21) --

mov [rsi + rcx \* 8], rax ;(22)

jnz .loop2 ;(23) skok jeśli rcx nie równa się 0

pop rcx ;(24)

ret ;(25) \*\*\* funkcja wypelnij koniec

;(26)

**WYMIANA 2/3**

szukaj: ;(27) \*\*\* funkcja szukaj start

\_szukaj: ;(28)

push rdi ;(29)

xor rax, rax ;(30)

.search: ;(31)

scasb ;(32) porównuje (ustawia flagi) bajt w AL z bajtem w EDI (zakładam że najstarszym inaczej nie miało by to sensu) i następnie zwiększa EDI o 1

jne .search ;(33) wyjście z funkcji jeśli równe jeśli nierówne powtórz

dec rdi ;(34) --

mov rax, rdi ;(35)

pop rdi ;(36)

sub rax, rdi ;(37)

and rax, -16 ;(38) logiczne and na bitach wynik do rax

ret ;(39)

;(40) \*\*\* funkcja szukaj koniec

**WYMIANA 3/3**

wymiana: ;(41)

\_wymiana: ;(42) \*\*\* funkcja wymiana start

push rbp ;(43)

mov rbp, rsp ;(44)

shl rcx, 8 ;(45) przesunięcie bitów w lewo w eax o 8 (puste miejsca wypełnione zerami)

mov cl, dl ;(46)

shl rcx, 8 ;(47) przesunięcie bitów w lewo w eax o 8 (puste miejsca wypełnione zerami)

mov cl, sil ;(48) This is enforced by changing (AH, BH, CH, DH) to (BPL, SPL, DIL, SIL) for instructions using a REX prefix. Cytat ze strony Intela

push rcx ;(49)

and rsp, 0xfffffffffffffff0 ;(50) logiczne and na bitach wynik do rsp

sub rsp, 64 ;(51)

push rdi ;(52)

;(53)

mov rcx, 3 ;(54)

lea rsi, [rsp+8] ;(55) oblicz adres i załaduj go do pamięci

.loop1: ;(56)

mov dil, [rbp+rcx-9] ;(57) This is enforced by changing (AH, BH, CH, DH) to (BPL, SPL, DIL, SIL) for instructions using a REX prefix. Cytat ze strony Intela

call wypelnij ;(58) wywołaj funkcję wypełnij

add rsi, 16 ;(59)

loop .loop1 ;(60)

;(61)

mov rdi, [rsp] ;(62)

call szukaj ;(63) wywołaj funkcję szukaj

pop rdi ;(64)

mov rsi, rdi ;(65)

add rdi, rax ;(66)

;(67)

.loop2: ;(68)

cmp rsi, rdi ;(69) zawiera opis zestawu instrukcji w tym sse

jge .koniec ;(70) zakończ wywołanie (jeśli większe bądź równe)

movdqu xmm1, [rsi] ;(71) przesuwa niewyrównane zmiennoprzecinkowe 32 bity do xmm1

movdqa xmm2, xmm1 ;(72) przesuwa wyrównane zmiennoprzecinkowe 32 bity do xmm2

movdqu xmm4, [rsp] ;(73) j.w.

movdqu xmm0, [rsp +32] ;(74) j.w.

pcmpgtb xmm0, xmm1 ;(75) Compare packed signed byte integers

paddb xmm1, [rsp+16] ;(76) Add packed byte integers

pand xmm4, xmm0 ;(77) operacja and na bitach wynik do xmm4

pandn xmm0, xmm1 ;(78) negacja bitowa xmm0 a potem bitowe and wynik do xmm0

por xmm4, xmm0 ;(79) bitowe or wynik do xmm4

movdqu [rsi], xmm4 ;(80) j.w.

add rsi, 16 ;(81)

jmp .loop2 ;(82)

.koniec: ;(83)

mov rsp,rbp ;(84) wyjście z funkcji

pop rbp ;(85)

ret ;(86) \*\*\* funkcja wymiana koniec