

Zad. 10.1

Zainstaluj program środowisko Context wraz z kompilatorem nasm zgodnie z instrukcją umieszczoną w katalogu „konfiguracja środowiska”.

W edytorze ConTEXT otwórz plik `char.asm` i przetestuj kolejno:

F9 – kompilacja
F10 – uruchomienie
F11 – deasemblacja
F12 – debugger

W debuggerze wpisz kolejno komendy:

```
file asmloader
run char
q
```

Zad. 10.2

Prześledź wynik uruchomienia programu `char.asm`. Przykładowa sesja:

```
Simplified Assembly Loader v.0.0.1 by gynvael.coldwind//vx
Code loaded at 0x002c0100 (12 bytes)
H
```

- pod jaki adres logiczny został załadowany ten program?
- ile bajtów zajmuje ten adres logiczny?
- ile bajtów w pamięci zajmuje ten program?
- jaki jest wynik działania tego programu?

Zad. 10.3

Prześledź wynik działania deasemblera dla programu `char.asm`.

```
00000000  6A48                push byte +0x48
00000002  FF5304             call [ebx+0x4]
00000005  83C404             add esp,byte +0x4
00000008  6A00                push byte +0x0
0000000A  FF13                call [ebx]
```

- co przechowuje pierwsza, druga i trzecia kolumna w powyższym listingu?

- jaki adres ma instrukcja `push 'H'` ?
- ile bajtów ma instrukcja `push 'H'` ?
- jaki kod rozkazu ma instrukcja `push 'H'` ?
- jaki kod ASCII ma literka `'H'` ?

- jaki adres ma instrukcja `call [ebx+1*4]` ?
- ile bajtów ma instrukcja `call [ebx+1*4]` ?
- jaki kod rozkazu ma instrukcja `call [ebx+1*4]` ?
- ile bajtów zajmuje kod rozkazu instrukcji `call [ebx+1*4]` ?
- jaki kod ma argument instrukcji `call [ebx+1*4]` ?

- jaki adres ma instrukcja `add esp, 4` ? *
- ile bajtów ma instrukcja `add esp, 4` ? *
- jaki kod rozkazu ma instrukcja `add esp, 4` ? *
- ile bajtów zajmuje kod rozkazu instrukcji `add esp, 4` ? *
- jaki kod ma argument instrukcji `add esp, 4` ? *

- jaki adres ma instrukcja `call [ebx+0*4]` ? *
- ile bajtów ma instrukcja `call [ebx+0*4]` ? *
- jaki kod rozkazu ma instrukcja `call [ebx+0*4]` ? *
- ile bajtów zajmuje kod rozkazu instrukcji `call [ebx+0*4]` ? *
- czy instrukcja `call [ebx+0*4]` ma kod argumentu? *

Zad. 10.4

Na poniższym przykładzie omów ogólne zasady formatowania kodu w języku asemblera.

```
label      instruction  ; comment
;          column 10    ; two spaces before a semicolon
```

- w której kolumnie umieszczamy etykiety?
- w której kolumnie umieszczamy instrukcje?
- ile spacji dajemy przed średnikiem w komentarzu?

Sprawdź czy program `char.asm` został poprawnie sformatowany.

Zad. 10.5

Przepisz i przeanalizuj program `labels.asm`. Zapisz go pod nazwą `labels2.asm` oraz zamień w nim nazwy etykiet adresami właściwymi dla uruchomionego programu.

- jak działa instrukcja `call _addr4` ?
- ile bajtów zajmuje instrukcja `call _addr4` ?
- jaki adres odkłada na stos instrukcja `call` ?
- pod jaki adres skacze instrukcja `call` ?
- jaki kod rozkazu ma instrukcja `nop` ?
- jaki kod rozkazu ma instrukcja `ret` ?
- skąd instrukcja `ret` pobiera adres powrotu?

Zad. 10.6

Napisz program `printf.asm` wypisujący napis `Hello world!` przy pomocy API `asmloadera`. Wykorzystaj komentarz:

```
; push on the stack the run-time address of format and jump to getaddr
```

- jaki adres ma instrukcja `call getaddr`?
- ile bajtów ma instrukcja `call getaddr`?
- jaki kod rozkazu ma instrukcja `call getaddr`?
- ile bajtów ma argument instrukcji `call getaddr`?

- co przechowuje etykieta `format`?
- jaką wartość ma etykieta `format`?
- jaką wartość na stosie ma `format`?

Zad. 10.7

Napisz program, który przy pomocy `asmloader api`:

```
printf2.asm – wyświetla stałą a
printf3.asm – wyświetla dwie stałe a i b *
printf4.asm – wyświetla stałą a w podprogramie print
printf5.asm – wyświetla dwie stałe a i b w podprogramie print *
printf6.asm – wyświetla stałą a zapisaną w pamięci programu
printf7.asm – tak jak powyżej, ale z wykorzystaniem instrukcji pop *
printf8.asm – skrócona wersja programu print6 *
printf9.asm – skrócona wersja programu print7 *
```

Wskazówka do dwóch ostatnich zadań:

Umieść liczbę `a` i napis `a =` w jednolitym/spójnym obszarze pamięci.

Zad. 10.8

Napisz program, który przy pomocy `asmloader api`:

```
add.asm – dodaje do rejestru eax zawartość rejestru ecx i wypisuje wynik
add2.asm – dodaje do wartości a w rejestrze eax stałą b i wypisuje wynik
add3.asm – dodaje do wartości a w rejestrze eax liczbę b z pamięci i wypisuje wynik *
```

```
sub.asm – odejmuje od rejestru eax zawartość rejestru ecx i wypisuje wynik *
sub2.asm – odejmuje od wartości a w rejestrze eax stałą b i wypisuje wynik *
sub3.asm – odejmuje od wartości a w rejestrze eax liczbę b z pamięci i wypisuje wynik *
```

Zad. 10.9

Napisz program ilustrujący działanie instrukcji dodawania z przeniesieniem `adc` (add with carry) kolejno ze zgaszoną i ustawioną flagą `CF`. Instrukcja `cld` (clear carry flag) gasi flagę `CF`. Instrukcja `stc` (set carry flag) ustawia flagę `CF`.

`adc.asm` – dodaje do rejestru `eax` zawartość rejestru `ecx` i wypisuje wynik

`adc2.asm` – dodaje do wartości `a` w rejestrze `eax` stałą `b` i wypisuje wynik

Uwaga: oba programy mają wyświetlać po dwa wyniki.

Zad. 10.10 *

Napisz program ilustrujący działanie instrukcji odejmowania z pożyczką `sbb` (subtract with borrow) kolejno ze zgaszoną i ustawioną flagą `CF`. Instrukcja `cld` (clear carry flag) gasi flagę `CF`. Instrukcja `stc` (set carry flag) ustawia flagę `CF`.

`sbb.asm` – odejmuje od rejestru `eax` zawartość rejestru `ecx` i wypisuje wynik

`sbb2.asm` – odejmuje od wartości `a` w rejestrze `eax` stałą `b` i wypisuje wynik

Uwaga: oba programy mają wyświetlać po dwa wyniki.