Laboratorium 1

Zad. 1.1

Napisz program size sprawdzający, ile bajtów zajmują typy: char, short, int, long, long int, long long oraz float, double, long double.

- z ilu bitów składa się jeden bajt? Jeden bajt składa się z 8 bitów

- ile wartości może przechowywać bajt? Bajt może przechowywać 2^8 = 256 wartości

- jakie wartości może przechowywać bajt? Bajt może przechowywać wartości od 0 do 255

- jaki typ w języku C odpowiada bajtowi? Jednemu bajtowi odpowiada char w języku C

https://en.wikipedia.org/wiki/C\_data\_types

Zad. 1.2 \*

Napisz program size2 sprawdzający, ile bajtów zajmują typy: char, short, int, long, long int i long long bez znaku.

Zad. 1.3

Która z poniższych odpowiedzi jest prawdziwa:

- system 32 bitowy pozwala na uruchamianie programu 32 bitowego T

- system 32 bitowy pozwala na uruchamianie programu 64 bitowego N

- system 64 bitowy pozwala na uruchamianie programu 32 bitowego T

- system 64 bitowy pozwala na uruchamianie programu 64 bitowego T

Zad. 1.4

Która z poniższych odpowiedzi jest fałszywa:

- na systemie 32 bitowym można skompilować program do kodu 32 bitowego T

- na systemie 32 bitowym można skompilować program do kodu 64 bitowego T

- na systemie 64 bitowym można skompilować program do kodu 32 bitowego T

- na systemie 64 bitowym można skompilować program do kodu 64 bitowego T

Zad. 1.5

- ile bajtów zajmują adresy w kodzie 32 bitowym? 4 bajty

- ile bajtów zajmują adresy w kodzie 64 bitowym? 8 bajtów

Zad. 1.6 \*

Jaki obszar pamięci można zaadresować przy pomocy adresów 16, 20, 24, 32, 40, 48 i 64 bitowych?

Jedna kombinacja adresu pamięci to jakby jedna „komórka”, a taka jedna komórka to 1 bajt.

Adres 16 bitowy – 2^16 = 65 536 bajtów

Adres 20 bitowy – 2^20 = 1 048 576 bajtów

Adres 24 bitowy – 2^24 = 16 777 216 bajtów

Adres 32 bitowy – 2^32 = 4 294 967 296 bajtów

Adres 40 bitowy – 2^40 = 1 099 511 627 776 bajtów

Adres 48 bitowy – 2^48 = 281 474 976 710 656 bajtów

Adres 64 bitowy – 2^64 = 18 446 744 073 709 551 616 bajtów

Zad. 1.7

Napisz program bits rozpoznający do ilu bitowego kodu został skompilowany.

Zad. 1.8

Załóżmy, że typ int zajmuje 4 bajty. Na ile sposobów można umieścić w pamięci pod adresem p wartość 1 typu int? Zadanie rozwiąż w pliku sposoby.txt.

p -> [ ][ ][ ][ ] \*p = 1

Zad. 1.9

Procesory w architekturze little-endian zapisują dane w pamięci od lewej do prawej zaczynając od najmłodszego bajtu (LSB – least significant byte). Procesory w architekturze big-endian zapisują dane w pamięci od lewej do prawej zaczynając od najstarszego bajtu (MSB – most significant byte). Załóżmy, że pod adresem p znajduje się liczba 5 typu int. W pliku endian.txt wypełnij komórki pamięci odpowiednimi wartościami dla obu architektur.

little-endian

p -> [ ][ ][ ][ ] \*p = 5

big-endian

p -> [ ][ ][ ][ ] \*p = 5

Zad. 1.10

W pliku szereg.txt rozwiń w szereg i wyznacz wartości dziesiętne dla liczb:

1011 – liczba binarna

8732 – liczba dziesiętna

[2][2][1][1] - reprezentacja little-endian

1234 - liczba ósemkowa \*

3A5B – liczba szesnastkowa \*

Zad. 1.11

Załóżmy, że pod adresem p znajduje się liczba 260 typu int. Wypełnij komórki pamięci odpowiednimi wartościami dla obu architektur. Zadanie rozwiąż w pliku 260.txt.

little-endian

p -> [ ][ ][ ][ ] \*p = 260

big-endian

p -> [ ][ ][ ][ ] \*p = 260

Zad. 1.12

Napisz program bytes wypisujący reprezentację bajtową dla podanej liczby x typu int. Przykładowa sesja:

value = 260

bytes = 004 001 000 000

Zad. 1.13

Sprawdź na terminalu Linux w jakiej architekturze pracuje twój procesor. Architektura 64-bitowa little-endian.

Zad. 1.14

Napisz program endian rozpoznający w jakiej architekturze pracuje procesor.

Zad. 1.15 \*

Napisz program endian2 wypisujący little-endian architecture lub big-endian architecture w zależności od architektury, w której pracuje procesor.

Laboratorium 2

Zad. 2.1

Napisz program stack umieszczający na stosie kolejno dwie zmienne x i y typu int z wartościami 1 i 2 oraz odczytaj adresy tych zmiennych. Skopiuj poniższy schemat do komentarza w programie i wypełnij go odpowiednimi wartościami.

&var1 [ ][ ][ ][ ] var1

&var2 [ ][ ][ ][ ] var2

Czy adresy zmiennych są zgodne z mapą pamięci dla procesu? Tak

Zad. 2.2 \*

W programie data umieść w sekcji danych kolejno zmienne a, b, c, d, e, f, g, h typu int oraz odczytaj adresy tych zmiennych. Do zmiennych zainicjowanych podstaw kolejne liczby naturalne od 1 do 4. Rozpatrz następujące przypadki:

- zmienne a i b są zainicjowane

- zmienne c i d są niezainicjowane

- zmienna e jest zainicjowana i zmienna f jest niezainicjowana

- zmienna g jest niezainicjowana i zmienna h jest zainicjowana

Skopiuj poniższy schemat do komentarza w programie oraz wypełnij go odpowiednimi wartościami.

&var1 [ ][ ][ ][ ] var1

&var2 [ ][ ][ ][ ] var2

&var3 [ ][ ][ ][ ] var3

&var4 [ ][ ][ ][ ] var4

&var5 [ ][ ][ ][ ] var5

&var6 [ ][ ][ ][ ] var6

&var7 [ ][ ][ ][ ] var7

&var8 [ ][ ][ ][ ] var8

Czy adresy zmiennych są zgodne z mapą pamięci dla procesu? Tak

Zad. 2.3

W programie heap umieść na stercie dwie liczby 1 i 2 typu int. Adresy tych liczb zapisz w zmiennych odpowiednio x i y. Wypisz te adresy oraz wartości, które się pod nimi znajdują. Oblicz rozmiar przydzielonego obszaru pamięci. Skopiuj poniższy schemat do komentarza w programie i wypełnij go odpowiednimi wartościami.

[ ][ ][ ][ ]

[ ][ ][ ][ ]

[ ][ ][ ][ ]

var1 [ ][ ][ ][ ] \*var1

[ ][ ][ ][ ]

[ ][ ][ ][ ]

[ ][ ][ ][ ]

var2 [ ][ ][ ][ ] \*var2

Czy do wskaźnika typu void\* można stosować operator wyłuskania? Nie można bezpośrednio stosować operatora wyłuskania (\*) do wskaźnika typu void\*, najpierw należy rzutować void\* na wskaźnik na konkretny typ danych przed wyłuskaniem wartości

Ile razy należy wywołać funkcję malloc, aby obliczyć rozmiar przydzielonego obszaru pamięci? 2 razy

Czy adresy liczb są zgodne z mapą pamięci dla procesu? Tak

Jaki najmniejszy obszar pamięci przydziela na stercie system operacyjny dla kodu 32 i 64 bitowego?

Kod 32-bitowy – 16 bajtów

Kod 64-bitowy – 32 bajty

Zad. 2.4

W pliku konwersje.txt dokonaj konwersji liczb:

- dziesiętne 11 na liczbę binarną

- dziesiętne 99 na liczbę binarną \*

- szesnastkowe 10AF na liczbę binarną

- szesnastkowe 3A58 na liczbę binarną \*

Zad. 2.5

Napisz program konwersje implementujący następujące funkcje:

int dec2bin(int x);

int bin2dec(int x); // \*

void dec2byte(unsigned int x); // reprezentacja little-endian \*

- jaką maksymalną liczbę binarną można zapisać przy pomocy typu int? UBIN\_MAX = 1111111111

- jaka jest wartość dziesiętna maksymalnej liczby binarnej, jaką można zapisać przy pomocy typu int? UBIN\_MAX = 1111111111 = 2^10 - 1 = 1KB - 1 = 1023

- dla jakich wartości parametrów aktualnych powyższe funkcje będą działać poprawnie? Dla wartości od 0 do 1023

Przykładowa sesja:

dec2bin(1023) = 1111111111

bin2dec(1111111111) = 1023

dec2byte(1023) = [255][003][000][000]

Zad. 2.6

Napisz program number wyliczający wartość liczby bez znaku na podstawie jej n bajtowej reprezentacji pod adresem p przy pomocy funkcji:

- funkcja polinomial wylicza wartość wielomianu w sposób klasyczny

int polinomial(unsigned char \*p, int n);

- funkcja horner wylicza wartość wielomianu schematem Hornera \*

int horner(unsigned char \*p, int n);

Przetestuj funkcje dla reprezentacji jedno, dwu i czterobajtowych. Reprezentację liczby należy określić przy pomocy tablicy typu char. Wskazówka:

http://www.balois.pl/java/proste/wielomiany.htm

Przykładowa sesja:

number(0061FE94, 4) = 16711680

number(0061FE94, 4) = 16711680

Laboratorium 3

Zad. 3.1

1. Zainstaluj program Embarcadero\_Dev-Cpp\_6.3\_TDM-GCC 9.2\_Setup.exe

2. Uruchom jako administrator program nasm-2.16.02rc5-installer-x64.exe

3. Zainstaluj program ConTEXTv0\_986.exe

4. Do folderu C:\Program Files (x86)\ConTEXT\Highlighters skopiuj plik NASM.chl

5. Uruchom plik rejestru context.reg i potwierdź zmiany

6. W folderze do zajęć z ASK do folderu lab3 skopiuj pliki asmloader.exe i char.asm

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

F9 – kompilacja

F10 – uruchomienie

F11 – [deasemblacja](https://sjp.pl/deasemblacja)

F12 – debugger

W debuggerze wpisz kolejno komendy:

file asmloader

run char

q

Zad. 3.2

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? 0x00020100

- ile bajtów zajmuje ten addres logiczny? 4 bajty

- ile bajtów w pamięci zajmuje ten program? 12 bajtów

- jaki jest wynik działania tego programu? H

Zad. 3.3

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?

Pierwsza kolumna - adres instrukcji w pamięci.

Druga kolumna - kod maszynowy instrukcji.

Trzecia kolumna - mnemonik (kod-słowo) instrukcji wraz z argumentami (czytelna reprezentacja instrukcji dla programisty).

- jaki adres ma instrukcja push 'H' ? 00000000

- ile bajtów ma instrukcja push 'H' ? 2 bajty

- jaki kod rozkazu ma instrukcja push 'H' ? 6A

- jaki kod ASCI ma literka 'H' ? 48

- jaki adres ma instrukcja call [ebx+1\*4] ? 00000002

- ile bajtów ma instrukcja call [ebx+1\*4] ? 3 bajty

- jaki kod rozkazu ma instrukcja call [ebx+1\*4] ? FF53

- ile bajtów zajmuje kod rozkazu instrukcji call [ebx+1\*4] ? 2 bajty

- jaki kod ma argument instrukcji call [ebx+1\*4] ? 04

- jaki adres ma instrukcja add esp, 4 ? \* 00000005

- ile bajtów ma instrukcja add esp, 4 ? \* 3 bajty

- jaki kod rozkazu ma instrukcja add esp, 4 ? \* 83C4

- ile bajtów zajmuje kod rozkazu instrukcji add esp, 4 ? \* 2 bajty

- jaki kod ma argument instrukcji add esp, 4 ? \* 04

- jaki adres ma instrukcja call [ebx+0\*4] ? \* 0000000A

- ile bajtów ma instrukcja call [ebx+0\*4] ? \* 2 bajty

- jaki kod rozkazu ma instrukcja call [ebx+0\*4] ? \* FF13

- ile bajtów zajmuje kod rozkazu instrukcji call [ebx+0\*4] ? \* 2 bajty

- czy instrukcja call [ebx+0\*4] ma kod argumentu? \* Nie

Zad. 3.4

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? 1 kolumna

- w której kolumnie umieszczamy instrukcje? 10 kolumna

- ile spacji dajemy przed średnikiem w komentarzu? 2 spacje

Zad. 3.5

Zapisz program labels.asm pod nazwą labels2.asm oraz zamień w nim nazwy etykiet adresami właściwymi dla uruchomionego programu.

- jaki adres odkłada na stos instrukcja call ? \_20105

- pod jaki adres skacze instrukcja call ? \_20107

Zad. 3.6

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 ? 00000000

- ile bajtów ma instrukcja call getaddr ? 5 bajtów

- jaki kod rozkazu ma instrukcja call getaddr ? E8

- ile bajtów ma argument instrukcji call getaddr ? 4 bajty

- co przechowuje etykieta format ? Adres

- jaką wartość ma etykieta format ? 00000005

- jaką wartość na stosie ma format ? 0x00020100 + 00000005 = 0x00020105

Zad. 3.7

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 – wyświeta dwie stałe a i b w podprogramie print \*

printf6.asm – wyświeta 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. 3.8

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 zawartośc 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. 3.9

Napisz program ilustrujący działanie instrukcji dodawania z przeniesieniem adc (add with carry) kolejno ze zgaszoną i ustawioną flagą CF. Instrukcja clc (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. 3.10 \*

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

sbb.asm – odejmuje od rejestru eax zawartośc 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.

Laboratorium 4

Zad. 4.1

Napisz program ascii.asm, który dla znaku odczytanego z konsoli wypisze jego kod ASCII. Przykładowa sesja:

znak = H

ascii = 48

Zad. 4.2

Przepisz programy scanf i scanf2 oraz przeanalizuj działanie tych programów.

Zad. 4.3

Napisz program scanf.asm ilustrujący wywołanie funkcji scanf z asmloader api dla liczb całkowitych. Przykładowa sesja:

a = 5

a = 5

Zad. 4.4 \*

Napisz program scanf2.asm ilustrujący wywołanie funkcji scanf z biblioteki msvcrt.

Zad. 4.5 \*

Napisz program scanf3.asm analogicznie do scanf2.asm, ale tym razem zmienną a umieść w sekcji danych niezainicjowanych.

Zad. 4.6

Napisz program xadd.asm ilustrujący działanie instrukcji xadd. Dane wejściowe podajemy jako stałe, a efekt działania instrukcji ma być wypisany na konsoli. Przykładowa sesja:

(eax, ebx) = (3, 5)

(eax, ebx) = (8, 3)

Zad. 4.7 \*

Napisz program xchg.asm ilustrujący działanie instrukcji xchg. Dane wejściowe podajemy jako stałe, a efekt działania instrukcji ma być wypisany na konsoli. Przykładowa sesja:

(esi, edi) = (3, 5)

(esi, edi) = (5, 3)

Zad. 4.8

Napisz program modul.asm obliczający moduł liczby. Liczbę podajemy jako stałą, a moduł ma być wypisany na konsoli. Przykładowa sesja:

liczba = -5

modul = 5

Zad. 4.9 \*

Napisz program modul2.asm obliczający moduł liczby z wykorzystaniem instrukcji test. Liczbę podajemy jako stałą, a moduł ma być wypisany na konsoli.

Zad. 4.10 \*

Napisz program modul3.asm obliczający moduł liczby. W tym przypadku liczbę odczytujemy z konsoli.

Zad. 4.11 \*

Napisz program relokowalny modul4.asm obliczający moduł liczby. W tym przypadku liczbę odczytujemy z konsoli.

Zad. 4.12 \*

Napisz program okno.asm sprawdzający, czy liczba x należy do przedziału [a,b]. Dane wejściowe podajemy jako stałe. Przykładowe dwie sesje:

25 nalezy do [18, 99]

-6 nie nalezy do [18, 99]

Zad. 4.13 \*

Napisz program okna.asm sprawdzający, czy liczba x należy do przedziału (a,b) lub (c,d). Dane wejściowe podajemy jako stałe. Przykładowe dwie sesje:

15 nalezy do (5, 19)

15 nalezy do (12, 24)

3 nie nalezy do (5, 19) i (12, 24)

Laboratorium 5

Zad. 5.1

Napisz program mul.asm ilustrujący operację mnożenia bez znaku. Program wypisuje 4 młodsze bajty wyniku. Dane wejściowe podajemy jako stałe, a iloczyn ma być wypisany na konsoli.

Zad. 5.2

Napisz program mul ilustrujący mnożenie dwóch liczb typu int bez znaku z wynikiem typu int bez znaku. Przeprowadź testy porównawcze z programem mul.asm. Przykładowa sesja:

a = 4294967295

b = 2

iloczyn = 4294967294

Zad. 5.3

Napisz program mul2.asm ilustrujący operację mnożenia bez znaku. Program wypisuje pełny 8 bajtowy wynik. Dane wejściowe podajemy jako stałe, a iloczyn ma być wypisany na konsoli.

Zad. 5.4 \*

Napisz program mul2 ilustrujący mnożenie dwóch liczb typu int bez znaku z wynikiem typu long long bez znaku. Przeprowadź testy porównawcze z programem mul2.asm. Przykładowa sesja:

a = 4294967295

b = 2

iloczyn = 8589934590

Zad. 5.5 \*

Napisz program imul.asm ilustrujący operację mnożenia ze znakiem. Program wypisuje 4 młodsze bajty wyniku. Dane wejściowe podajemy jako stałe, a iloczyn ma być wypisany na konsoli.

Zad. 5.6 \*

Napisz program imul ilustrujący mnożenie dwóch liczb typu int ze znakiem z wynikiem typu int ze znakiem. Przeprowadź testy porównawcze z programem imul.asm. Przykładowa sesja:

a = -65535

b = 2

iloczyn = -131070

Zad. 5.7 \*

Napisz program imul2.asm ilustrujący operację mnożenia ze znakiem. Program wypisuje pełny 8 bajtowy wynik. Dane wejściowe podajemy jako stałe, a iloczyn ma być wypisany na konsoli.

Zad. 5.8 \*

Napisz program imul2 ilustrujący mnożenie dwóch liczb typu int ze znakiem z wynikiem typu long long ze znakiem. Przeprowadź testy porównawcze z programem imul2.asm. Przykładowa sesja:

a = -2147483647

b = 2

iloczyn = -4294967294

Zad. 5.9

Napisz program expression.asm obliczający wartość wyrażenia a + b\*c dla stałych typu int bez znaku z wynikiem typu int bez znaku. Dane wejściowe ładujemy do rejestrów, a wynik ma być wypisany na konsoli.

Zad. 5.10

Napisz program add4.asm dodający dwie liczby a i b typu int bez znaku z wynikiem 8 bajtowym bez znaku. Dane wejściowe ładujemy do rejestrów, a wynik ma być wypisany na konsoli.

Zad. 5.11 \*

Napisz program add4 dodający dwie liczby a i b typu int bez znaku z wynikiem 8 bajtowym bez znaku. Przeprowadź testy porównawcze z programem add4.asm. Przykładowa sesja:

a = 4294967295

b = 1

suma = 4294967296

Zad. 5.12

Napisz program add5.asm dodający dwie liczby a i b typu int z wynikiem 8 bajtowym ze znakiem. Dane wejściowe ładujemy do rejestrów, a wynik ma być wypisany na konsoli.

Zad. 5.13 \*

Napisz program add5 dodający dwie liczby a i b typu int z wynikiem 8 bajtowym ze znakiem. Przeprowadź testy porównawcze z programem add5.asm. Przykładowa sesja:

a = -2147483648

b = -1

suma = -2147483649

Zad. 5.14 \*

Napisz 64-bitowy program add6.asm dodający dwie liczby a i b typu int bez znaku z wynikiem 8 bajtowym bez znaku. Dane wejściowe ładujemy do rejestrów, a wynik ma być wypisany na konsoli.

Zad. 5.15 \*

Napisz 64-bitowy program add7.asm dodający dwie liczby a i b typu int z wynikiem 8 bajtowym ze znakiem. Dane wejściowe ładujemy do rejestrów, a wynik ma być wypisany na konsoli.

Zad. 5.16 \*

Napisz program expression2.asm obliczający wartość wyrażenia a + b\*c dla stałych typu int bez znaku z wynikiem 8 bajtowym bez znaku. Dane wejściowe ładujemy do rejestrów, a wynik ma być wypisany na konsoli.

Zad. 5.17 \*

Napisz program expression3.asm obliczający wartość wyrażenia a + b\*c dla stałych typu int z wynikiem 8 bajtowym ze znakiem. Dane wejściowe ładujemy do rejestrów, a wynik ma być wypisany na konsoli.

Zad. 5.18 \*

Napisz program expression4.asm obliczający wartość wyrażenia a\*b + c\*d dla stałych typu int bez znaku z wynikiem typu int bez znaku. Dane wejściowe ładujemy do rejestrów, a wynik ma być wypisany na konsoli.

Zad. 5.19 \*

Napisz program expression5.asm obliczający wartość wyrażenia a\*b + c\*d dla stałych typu int bez znaku z wynikiem 8 bajtowym bez znaku. Dane wejściowe ładujemy do rejestrów, a wynik ma być wypisany na konsoli.

Zad. 5.20 \*

Napisz program expression6.asm obliczający wartość wyrażenia a\*b + c\*d dla stałych typu int z wynikiem 8 bajtowym ze znakiem. Dane wejściowe ładujemy do rejestrów, a wynik ma być wypisany na konsoli.

Laboratorium 6

Zad. 6.1

Napisz program div.asm ilustrujący operację dzielenia bez znaku. Program wykonuje obliczenia dla dzielnej 4 bajtowej. Dane wejściowe podajemy jako stałe, a iloraz i reszta mają być wypisane na konsoli.

- w jakim przypadki instrukcja div wyrzuca błąd #DE ?

Instrukcja div wyrzuca błąd #DE, kiedy dzielnik jest równy zero.

Instrukcja div wyrzuca błąd #DE, kiedy wynik nie mieści się w rejestrze przeznaczonym na iloraz.

Zad. 6.2

Napisz program div ilustrujący dzielenie dwóch liczb typu int bez znaku z wynikiem typu int bez znaku. Przeprowadź testy porównawcze z programem div.asm. Przykładowa sesja:

a = 4294967295

b = 4

iloraz = 1073741823

reszta = 3

Zad. 6.3

Napisz program div2.asm ilustrujący operację dzielenia bez znaku. Program wykonuje obliczenia dla dzielnej 8 bajtowej. Dane wejściowe podajemy jako stałe, a iloczyn ma być wypisany na konsoli.

Zad. 6.4 \*

Napisz program div2 ilustrujący operację dzielenia bez znaku. Program wykonuje obliczenia dla dzielnej 8 bajtowej. Przeprowadź testy porównawcze z programem div2.asm. Przykładowa sesja:

a = 4294967296

b = 3

iloraz = 1431655765

reszta = 1

Zad. 6.5

Napisz program idiv.asm ilustrujący operację dzielenia ze znakiem. Program wykonuje obliczenia dla dzielnej 4 bajtowej. Dane wejściowe podajemy jako stałe, a iloraz i reszta mają być wypisane na konsoli.

Zad. 6.6

Napisz program idiv ilustrujący dzielenie dwóch liczb typu int ze znakiem z wynikiem typu int ze znakiem. Przeprowadź testy porównawcze z programem idiv.asm. Przykładowa sesja:

a = -2147483648

b = -3

iloraz = 715827882

reszta = -2

Zad. 6.7 \*

Napisz program idiv2.asm ilustrujący operację dzielenia ze znakiem. Program wykonuje obliczenia dla dzielnej 8 bajtowej. Dane wejściowe podajemy jako stałe, a iloczyn ma być wypisany na konsoli.

Zad. 6.8 \*

Napisz program idiv2 ilustrujący operację dzielenia ze znakiem. Program wykonuje obliczenia dla dzielnej 8 bajtowej. Przeprowadź testy porównawcze z programem idiv2.asm. Przykładowa sesja:

a = -2147483650

b = -3

iloraz = 715827883

reszta = -1

Zad. 6.9

Napisz program loop.asm implementujący pętlę przy pomocy instrukcji skoku. Przykładowa sesja:

i = 3

i = 2

i = 1

Zad. 6.10

Napisz program loop2.asm implementujący pętlę przy pomocy instrukcji loop.

Zad. 6.11 \*

Napisz program loop3.asm analogicznie do loop.asm dekrementujący wartość licznika bezpośrednio na stosie.

Zad. 6.12

Napisz program silnia.asm obliczający silnię dla danej liczby n z wynikiem typu int. Liczbę n podajemy jako stałą, a silnia ma być wypisana na konsoli.

0! = 1

n! = n\*(n-1)!

Zad. 6.13 \*

Napisz program silnia2.asm obliczający silnię dla danej liczby n z wynikiem 8 bajtowym. Liczbę n podajemy jako stałą, a silnia ma być wypisana na konsoli.

0! = 1

n! = n\*(n-1)!

Zad. 6.14 \*

Napisz program silnia3.asm obliczający silnię podwójną dla danej liczby n. Liczbę n podajemy jako stałą, a silnia podwójna ma być wypisana na konsoli.

0!! = 1

1!! = 1

n!! = n\*(n-2)!!

Zad. 6.15 \*

Napisz program suma.asm obliczający sumę n początkowych liczb naturalnych. Liczbę n podajemy jako stałą, a suma ma być wypisana na konsoli.

suma(1) = 1

suma(2) = 1 + 2

suma(n) = 1 + 2 + … + n

Laboratorium 7

Zad. 7.1

Napisz program length.asm wyliczający liczbę cyfr podanej nieujemnej liczby całkowitej n. Liczbę n podajemy jako stałą, a liczba cyfr ma być wypisana na konsoli.

Zad. 7.2 \*

Napisz program length2.asm wyliczający liczbę cyfr podanej liczby całkowitej n. Liczbę n podajemy jako stałą, a liczba cyfr ma być wypisana na konsoli.

Zad. 7.3

W pliku fibo.txt podaj słowną oraz rekurencyjną definicję ciągu Fibonacciego.

0 1 2 3 4 5 6 indeksy

1 1 2 3 5 8 13 wartości

Zad. 7.4

Napisz program fibo z funkcją fibo1 wyliczający wartości ciągu Fibonacciego metodą programowania dynamicznego przy pomocy ramki trójzębnej.

r0 r1 r2

|---|---|

0 1 2 3 4 5 6 indeksy

1 1 2 3 5 8 13 wartości

|---|---|

r0 r1 r2

Przesunięcie ramki w prawo:

r0 = r1

r1 = r2

r2 = r1 + r0

- Ile razy należy przesunąć ramkę w prawo, aby wyznaczyć wartość n-tego wyrazu ciągu Fibonacciego w funkcji fibo1 dla n >= 3 ? Należy przesunąć ramkę w prawo n – 2 razy.

- Dokonaj analizy wywołania fibo1(4). Analiza wywołania znajduje się w pliku „fibo.c”.

- Narysuj graf obliczeń dla fibo1(4). Graf obliczeń znajduje się w pliku „fibo.c”.

Zad. 7.5 \*

Napisz program sequence z funkcją seq1 wyliczający wartości ciągu {seqn} metodą programowania dynamicznego przy pomocy ramki trójzębnej. Narysuj ramkę i określ instrukcje przesuwające ramkę. Ciąg {seqn} zdefiniowany jest rekurencyjnie:

seq(1) = 3

seq(2) = 4

seq(n) = 0.5\*seq(n-1) + 2\*seq(n-2) dla n > 2

- Ile razy należy przesunąć ramkę w prawo, aby wyznaczyć wartość n-tego wyrazu ciągu w funkcji seq dla n >= 3 ? Należy przesunąć ramkę w prawo n – 3 razy.

- Dokonaj analizy wywołania seq1(4). Analiza wywołania znajduje się w pliku „sequence.c”.

- Narysuj graf obliczeń dla seq1(4). Graf obliczeń znajduje się w pliku „sequence.c”.

Zad. 7.6

Rejestry b i d przyjmują wartości początkowe odpowiednio 1 i 2. Napisz program shift.asm realizujący na tych rejestrach poniższe operacje. Dokonaj analizy fragmentu kodu, który wykonuje te operacje.

a = b

b = d

d = a + b

Zad. 7.7

Napisz program fibo.asm obliczający n-ty wyraz ciągu Fibonacciego metodą programowania dynamicznego przy pomocy ramki trójzębnej.

0 1 2 3 4 5 6 indeksy

a b d

|---|---|

1 1 2 3 5 8 13 wartosci

|---|---|

a b d

Przesuniecie ramki:

a = b ; a = 1

b = d ; b = 2

d = a + b ; d = 1 + 2 = 3

- Ile razy należy przesunąć ramkę w prawo, aby wyznaczyć wartość n-tego wyrazu ciągu Fibonacciego dla n >= 3 ? Należy przesunąć ramkę w prawo n – 2 razy.

Zad. 7.8 \*

Rejestry b i d przyjmują wartości początkowe odpowiednio 1 i 2. Napisz program shift2.asm realizujący na tych rejestrach poniższe operacje. Dokonaj optymalizacji i analizy fragmentu kodu, który wykonuje te operacje.

a = b

b = d

d = a + b = d + a

Zad. 7.9 \*

Napisz program fibo2.asm obliczający n-ty wyraz ciągu Fibonacciego metodą programowania dynamicznego przy pomocy ramki trójzębnej z optymalizacją przesunięcia ramki.

0 1 2 3 4 5 6 indeksy

a b d

|---|---|

1 1 2 3 5 8 13 wartosci

|---|---|

a b d

Przesuniecie ramki:

a = b ; a = 1

b = d ; b = 2

d = a + b = d + a ; d = 2 + 1 = 3

Laboratorium 8

Zad. 8.1

Do programu fibo dodaj funkcję fibo2 wyliczającą wartości ciągu Fibonacciego metodą programowania dynamicznego przy pomocy ramki dwuzębnej.

r0 r1

|---|

0 1 2 3 4 5 6 indeksy

1 1 2 3 5 8 13 wartości

| |---|

pom r0 r1

Przesunięcie ramki w prawo:

pom = r0

r0 = r1

r1 = pom + r0

- Ile razy należy przesunąć ramkę w prawo, aby wyznaczyć wartość n-tego wyrazu ciągu Fibonacciego w funkcji fibo2 dla n >= 2 ? Należy przesunąć ramkę w prawo n – 1 razy.

- Dokonaj analizy wywołania fibo2(4). Analiza wywołania znajduje się w pliku „fibo.c”.

- Narysuj graf obliczeń dla fibo2(4). Graf obliczeń znajduje się w pliku „fibo.c”.

- Która funkcja ma mniejszą złożoność obliczeniową fibo1 czy fibo2 ? Obie funkcje mają taką samą złożoność obliczeniową, jednak funkcja fibo2 ma mniejszą złożoność pamięciową, ponieważ używa tylko dwóch zmiennych do przechowywania wartości ciągu, podczas gdy fibo1 używa trzech zmiennych.

Zad. 8.2 \*

Do programu sequence dodaj funkcję seq2 wyliczającą wartości ciągu {seqn} metodą programowania dynamicznego przy pomocy ramki dwuzębnej. Narysuj ramkę i określ instrukcje przesuwające ramkę.

- Ile razy należy przesunąć ramkę w prawo, aby wyznaczyć wartość n-tego wyrazu ciągu w funkcji seq2 dla n >= 2 ? Należy przesunąć ramkę w prawo n – 2 razy.

- Dokonaj analizy wywołania seq2(4). Analiza wywołania znajduje się w pliku „sequence.c”.

- Narysuj graf obliczeń dla seq2(4). Graf obliczeń znajduje się w pliku „sequence.c”.

- Która funkcja ma mniejszą złożoność obliczeniową seq1 czy seq2 ? Obie funkcje mają taką samą złożoność obliczeniową, jednak funkcja seq2 ma mniejszą złożoność pamięciową, ponieważ używa tylko dwóch zmiennych do przechowywania wartości ciągu, podczas gdy seq1 używa trzech zmiennych.

Zad. 8.3

Napisz program fibo3.asm obliczający n-ty wyraz ciągu Fibonacciego metodą programowania dynamicznego przy pomocy ramki dwuzębnej.

0 1 2 3 4 5 6 indeksy

a b

|---|

1 1 2 3 5 8 13 wartości

| |---|

d a b

Przesuniecie ramki:

d = a ; d = 1

a = b ; a = 1

b = a + d = b + d ; b = 1 + 1 = 2

- ile razy należy przesunąć ramkę w prawo, aby wyznaczyć wartość n-tego wyrazu ciągu Fibonacciego dla n >= 2 ? Należy przesunąć ramkę w prawo n – 1 razy.

Zad. 8.4 \*

Napisz program fibo4.asm analogicznie do fibo3.asm z optymalizacją za pomocą jednej instrukcji test dla dwóch pierwszych wyrazów ciągu.

Zad. 8.5

Napisz program fibo5.asm analogicznie do fibo3.asm z optymalizacją za pomocą jednej instrukcji cmp dla dwóch pierwszych wyrazów ciągu.

Zad. 8.6

Napisz program fibo6.asm analogicznie do fibo3.asm obliczający n-ty wyraz ciągu Fibonacciego metodą programowania dynamicznego przy pomocy instrukcji xadd.

a b a+2b

|---| |---|

1 1 2 3 5 8 13 wartosci

|---| |---|

b a+b 2a+3b

Przesuniecie ramki:

xadd (b, a) = (a+b, b) // wynik w rejestrze b

Schemat obliczeń:

xadd xadd xadd

(a, b) -> (b, a) => (a+b, b) => (a+2b, a+b) => (2a+3b, a+2b) => ...

- ile razy należy przesunąć ramkę w prawo, aby wyznaczyć wartość n-tego wyrazu ciągu Fibonacciego dla n >= 2 ? Należy przesunąć ramkę w prawo n – 1 razy.

Zad. 8.7 \*

Napisz program fibo7.asm analogicznie do fibo6.asm z optymalizacją za pomocą jednej instrukcji test dla dwóch pierwszych wyrazów ciągu.

Zad. 8.8 \*

Napisz program fibo8.asm analogicznie do fibo6.asm z optymalizacją za pomocą jednej instrukcji cmp dla dwóch pierwszych wyrazów ciągu.

Laboratorium 9

Zad. 9.1 \*

Przeanalizuj prezentację silnia.pps ilustrującą analizę funkcji rekurencyjnej silnia.

Zad. 9.2

Napisz program silnia wyliczający wartość n! metodą dziel i zwyciężaj.

0! = 1

n! = n\*(n-1)!

- dokonaj analizy wywołania silnia(3). Analiza wywołania znajduje się w pliku „silnia.c”.

- narysuj graf wywołań dla silnia(3). Graf wywołań znajduje się w pliku „silnia.c”.

Zad. 9.3 \*

Dodaj do programu silnia funkcję silniap wyliczającą wartość n!! metodą dziel i zwyciężaj.

0!! = 1

1!! = 1

n!! = n\*(n-2)!!

- dokonaj analizy wywołania silniap(3). Analiza wywołania znajduje się w pliku „silniap.c”.

- narysuj graf wywołań dla silniap(3). Graf wywołań znajduje się w pliku „silniap.c”.

Zad. 9.4

Napisz program r\_silnia.asm obliczający rekurencyjnie silnię dla danej liczby n, gdzie parametry aktualne wywołania zapamiętujemy na stosie.

- dokonaj analizy wywołania r\_silnia(2). \* Analiza wywołania znajduje się w pliku „r\_silnia.asm”.

Zad. 9.5

Napisz program r\_silnia2.asm obliczający rekurencyjnie silnię dla danej liczby n bez zapamiętywania na stosie parametrów aktualnych wywołania.

- dokonaj analizy wywołania r\_silnia2(2). \* Analiza wywołania znajduje się w pliku „r\_silnia2.asm”.

Zad. 9.6 \*

Napisz program r\_silnia3.asm obliczający rekurencyjnie silnię dla danej liczby n z zastosowaniem metody akumulacji statycznej.

- dokonaj analizy wywołania r\_silnia3(2). Analiza wywołania znajduje się w pliku „r\_silnia3.asm”.

Zad. 9.7 \*

Napisz program r\_silniap.asm obliczający rekurencyjnie silnię podwójną dla danej liczby n, gdzie parametry aktualne wywołania zapamiętujemy na stosie.

0!! = 1

1!! = 1

n!! = n\*(n-2)!!

Zad. 9.8 \*

Napisz program r\_silniap2.asm obliczający rekurencyjnie silnię podwójną dla danej liczby n bez zapamiętywania na stosie parametrów aktualnych wywołania.

Zad. 9.9

Do programu fibo dodaj funkcję fibo3 wyliczającą wartości ciągu Fibonacciego metodą dziel i zwyciężaj.

- dokonaj analizy wywołania fibo3(4). Analiza wywołania znajduje się w pliku „fibo.c”.

- narysuj graf wywołań dla fibo3(4). Graf wywołań znajduje się w pliku „fibo.c”.

Zad. 9.10

Napisz program FiboTree wypisujący, jak wyglądają kolejne wywołania funkcji fibo3 razem z wartościami przez nie zwracanymi. Przykładowa sesja:

fibo1(4) = 5

fibo2(3) = 3

fibo3(2) = 2

fibo4(1) = 1

fibo5(0) = 1

fibo6(1) = 1

fibo7(2) = 2

fibo8(1) = 1

fibo9(0) = 1

- sprawdź czy drzewo wywołań z wcześniejszego zadania zostało poprawnie narysowane. Drzewo wywołań z wcześniejszego zadania zostało poprawnie narysowane.

Zad. 9.11 \*

Do programu sequence dodaj funkcję seq3 wyliczającą wartości ciągu {seqn} metodą dziel i zwyciężaj.

- dokonaj analizy wywołania seq3(4). Analiza wywołania znajduje się w pliku „sequence.c”.

- narysuj graf wywołań dla seq3(4). Graf wywołań znajduje się w pliku „sequence.c”.

Zad. 9.12 \*

Napisz program SequenceTree wypisujący, jak wyglądają kolejne wywołania funkcji seq3 razem z wartościami przez nie zwracanymi.

Zad. 9.13

Napisz program r\_fibo.asm obliczający n-ty wyraz ciągu Fibonacciego metodą dziel i zwyciężaj, gdzie parametry aktualne wywołania zapamiętujemy na stosie.

- dokonaj analizy wywołania r\_fibo(2). Analiza wywołania znajduje się w pliku „r\_fibo.asm”.

Zad. 9.14 \*

Napisz program r\_fibo2.asm obliczający n-ty wyraz ciągu Fibonacciego metodą dziel i zwyciężaj bez zapamiętywania na stosie parametrów aktualnych wywołania.

- dokonaj analizy wywołania r\_fibo2(2). Analiza wywołania znajduje się w pliku „r\_fibo2.asm”.

Laboratorium 10

Zad. 10.1

Napisz program ylog2x.asm obliczający logarytm przy podstawie 2 z liczby x.

Zad. 10.2 \*

Napisz program log2.asm obliczający logarytm przy podstawie 2 z liczby x bez odczytywania wartości y z pamięci.

Zad. 10.3 \*

Napisz program log10.asm obliczający logarytm przy podstawie 10 z liczby x. Wskazówka:

log10(x) = log2(x) / log2(10)

log10(x) = log10(2) \* log2(x)

Zad. 10.4 \*

W pliku liczba.txt wyznacz funkcję f(n) wyliczającą liczbę cyfr dla dowolnej liczby całkowitej n. Wykorzystaj funkcję log10(x).

Zad. 10.5 \*

Napisz program length3.asm wyliczający liczbę cyfr podanej liczb całkowitej n przy pomocy instrukcji koprocesora arytmetycznego FPU. Liczba całkowita n jest pobierana z konsoli. Przykładowa sesja:

n = -357

length = 3

Zad. 10.6 \*

Napisz program liniowe2.asm obliczający rozwiązania równania a\*x + b = 0. Współczynniki funkcji liniowej są pobierane z konsoli.

Zad. 10.7

Napisz program fpu\_exp\_i.asm obliczający wartość wyrażenia a + b\*c dla zmiennych całkowitych przy pomocy koprocesora arytmetycznego FPU.

Zad. 10.8 \*

Napisz program fpu\_exp\_d.asm obliczający wartość wyrażenia a + b\*c dla zmiennych rzeczywistych przy pomocy koprocesora arytmetycznego FPU.

Zad. 10.9 \*

Napisz program fpu\_exp2\_i.asm obliczający wartość wyrażenia a\*b + c\*d dla zmiennych całkowitych przy pomocy koprocesora arytmetycznego FPU.

Zad. 10.10 \*

Napisz program fpu\_exp2\_d.asm obliczający wartość wyrażenia a\*b + c\*d dla zmiennych rzeczywistych przy pomocy koprocesora arytmetycznego FPU.

Zad. 10.11

Napisz program reverse.asm odwracający napis z wykorzystaniem bloku pamięci.

Laboratorium 11

Zad. 11.1

Napisz program reverse2.asm odwracający napis bez wykorzystania bloku pamięci.

Zad. 11.2

Napisz program dodawanie realizujący dodawanie pisemne. Interfejs programu musi wyglądać dokładnie tak samo, jak w programie dodawanie.exe na stronie autora. Zakładamy, że dane wejściowe mają postać:

Z: a = 0, 1, 2, ...

b = 0, 1, 2, ...

- w jakich przypadkach program może dawać niepoprawne wyniki? Program może dawać niepoprawne wyniki na przykład w przypadku przepełnienia wartości INT.

Przykładowa sesja:

a = 9237

b = 1267

1 11

9237

+ 1267

------

10504

- jaka liczba jest wyświetlana jako pierwsza? Jako pierwsza wyświetlana jest większa liczba.

- ile może być maksymalnie przeniesień przy dodawaniu? Maksymalna liczba przeniesień to długość większej z dwóch liczb.

- ile wynosi i od czego zależy szerokość słupka dodawania? Szerokość słupka dodawania wynosi długość większej z dwóch liczb plus jedno miejsce na przeniesienie i plus jedno miejsce na znak „+”. Szerokość słupka dodawania zawsze zależy od długości większej liczby.

Zad. 11.3 \*

Napisz analogiczny program dodawanie2 przechowujący przeniesienia w tablicy znaków oraz dokonaj optymalizacji kodu, którą umożliwia to rozwiązanie.

Laboratorium 12

Zad. 12.1

Napisz program u2neg.asm ilustrujący negację podanej liczby z wykorzystaniem algorytmu dla kodu U2 oraz instrukcji negacji procesora.

Zad. 12.2

Napisz program konwersje2 z funkcjami hexDigit i hexDigit2 konwertującymi liczby z zakresu 0..15 na znak reprezentujący cyfrę szesnastkową. Druga funkcja wykorzystuje tablicę konwersji.

char hexDigit(char x);

Zad. 12.3

Napisz program hex-digit.asm konwertujący liczby z zakresu 0..15 na znak reprezentujący cyfrę szesnastkową.

Zad. 12.4

Napisz program hex-digit2.asm konwertujący liczby z zakresu 0..15 na znak reprezentujący cyfrę szesnastkową z wykorzystaniem tablicy konwersji.

Zad. 12.5

Do programu konwersje2 dodaj funkcję byte2hex wypisującą bajt w postaci liczby szesnastkowej z wykorzystaniem operatorów iloczynu logicznego i rotacji bitów.

void byte2hex(unsigned char byte);

Zad. 12.6 \*

Do programu konwersje2 dodaj funkcję byte2hex2 wypisującą bajt w postaci liczby szesnastkowej z wykorzystaniem operatorów dzielenia bez reszty oraz reszty z dzielenia.

void byte2hex2(unsigned char byte);

Zad. 12.7

Napisz program byte2hex.asm konwertujący bajt do napisu przechowującego liczbę szesnastkową.

Zad. 12.8

Napisz program byte2hex2.asm konwertujący bajt do napisu przechowującego liczbę szesnastkową z wykorzystaniem tablicy konwersji.

Zad. 12.9

Do programu konwersje2 dodaj funkcję dec2hex wypisującą liczbę dziesiętną w postaci liczby szesnastkowej z wykorzystaniem operatorów dzielenia bez reszty oraz reszty z dzielenia.

void dec2hex(unsigned int dec);

Zad. 12.10

Do programu konwersje2 dodaj funkcję dec2hex2 wypisującą liczbę dziesiętną w postaci liczby szesnastkowej z wykorzystaniem operatorów iloczynu logicznego i rotacji bitów.

void dec2hex2(unsigned int dec)