

Sprawozdanie z laboratorium Architektury Komputerów

Sprawozdanie zawiera kod programów do zadań wykonanych na zajęciach (13.1.3 oraz 14.1) oraz zadań na poprawę (13.1.4 i 14.3.2).

1. Zadanie 13.1.3

a) Kod programu

- 1) Plik: funkcja_dodaj.s – deklaracja funkcji dodaj, która pobiera dwie liczby, dodaje je i zwraca wynik w %eax.

```
.global dodaj
.type dodaj, @function
dodaj:
    pushl %ebp
    movl  %esp, %ebp
    movl  8(%ebp), %eax
    movl  12(%ebp), %ecx
    addl  %ecx, %eax
    # wynik w %eax
    movl  %ebp, %esp
    popl  %ebp
    ret

.global exit
.type exit, @function
exit:
    pushl %ebp
    movl  %esp, %ebp
    movl  8(%ebp), %ebx
    movl  $1, %eax
    int   $0x80
    ret
```

- 2) Plik: call.c – program wywołujący funkcje z pliku bibliotecznego, zadeklarowane w pliku dodaj.s

```
#include <stdio.h>
/* prototype for asm function */
int dodaj(int x,int y);
int exit(int result);

int main() {
    int arg1, arg2, wynik;
    printf("Podaj a i b\n");
    scanf("%i %i", &arg1, &arg2);
```

```

/* call the asm function */
wynik = dodaj(arg1, arg2);
printf("wynik %d\n", wynik);
exit(wynik);
return 0;
}

```

b) Kompilacja programu według następujących komend:

```

$as funkcja_dodaj.s -32 -g -o funkcja_dodaj.o
$ar rcsv libbiblioteka.a funkcja_dodaj.o
$gcc call.c -m32 -L. -lbiblioteka -g -o call

```

Uruchomienie programu:

```
$./call
```

c) Screen konsoli z kompilacji programu

```

artur@Artur: ~
Plik Edycja Widok Wyszukiwanie Terminal Pomoc
artur@Artur:~$ as funkcja_dodaj.s -32 -g -o funkcja_dodaj.o
funkcja_dodaj.s: Assembler messages:
funkcja_dodaj.s: Warning: koniec pliku nie jest na końcu wiersza; nowy wiersz
wstawiony
artur@Artur:~$ ar rcsv libbiblioteka.a funkcja_dodaj.o
a - funkcja_dodaj.o
artur@Artur:~$ nm libbiblioteka.a

funkcja_dodaj.o:
00000000 T dodaj
0000000f T exit
artur@Artur:~$ gcc call.c -m32 -L. -lbiblioteka -g -o call
call.c:4:5: warning: conflicting types for built-in function 'exit' [-Wbuiltin
-declaration-mismatch]
    int exit(int result);
    ^~~~~
artur@Artur:~$ ./call
Podaj a i b
12 25
Podano a = 12 i b = 25
wynik 37
artur@Artur:~$ echo $?
37
artur@Artur:~$

```

2. Zadanie 14.1 – wywołanie funkcji zadeklarowanej w assemblerze z poziomu języka C.

d) Kod programu

- 1) Plik: casma.s – deklaracja funkcji pobierającej wskaźnik na pierwszy element tablicy i ilość elementów, następnie zwiększa każdy element o 1.

```

.section .text
.globl asm_mod_array
.type asm_mod_array, @function
asm_mod_array:
    pushl %ebp                # zabezpieczamy stary %ebp
    movl %esp, %ebp          # nowy base pointer = %esp (szczyt stosu)

```

```

movl 8(%ebp),%eax          # pierwszy par. - wsk. na pocz. tablicy do
%eax
movl 12(%ebp),%ecx         # rozmiar tablicy do %ecx
xorl %edi, %edi           # zerujemy indeks biezacy w %edi

start_loop:               # start petli
cmpl %edi, %ecx           # czy koniec tablicy
je loop_exit             # skok do koniec
movl (%eax,%edi,4), %edx   # przesyłamy biezacy element tablicy do %edx
addl $1, %edx             # dodaj do elementu 1
movl %edx, (%eax,%edi,4)   # nadpisz nowy element nową wartoscia
incl %edi                 # zwieksz indeks, przesuwasz się po tablicy
jmp start_loop            # skocz na poczatek petli

loop_exit:                # zakonczenie funkcji
movl %ebp, %esp
popl %ebp
ret

```

2) Plik: casmc.c – wywołanie funkcji zadeklarowanej w pliku: casma.s

```

#include <stdio.h>
/* prototype for asm function */
int * asm_mod_array(int *ptr,int size);

int main() {
int fren[5]={ 1, 2, 3, 4, 5 };
printf("Tablica przed wywołaniem: %d %d %d %d
%d\n",fren[0],fren[1],fren[2],fren[3],fren[4]);
/* call the asm function */
asm_mod_array(fren, 5);
printf("Po wywołaniu: %d %d %d %d
%d\n",fren[0],fren[1],fren[2],fren[3],fren[4]);
return 0;
}

```

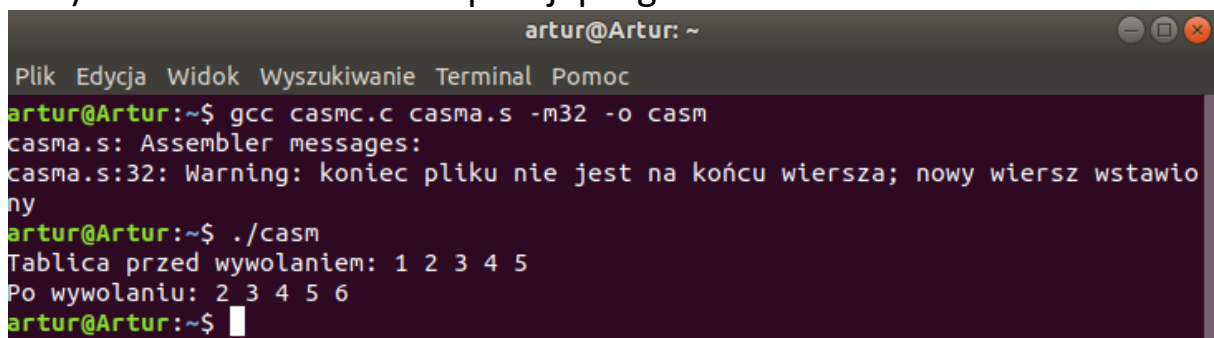
e) Kompilacja programu:

```
$gcc casmc.c casma.s -m32 -o casm
```

Uruchomienie programu:

```
$./casm
```

f) Screen konsoli z kompilacji programu



```

artur@Artur: ~
Plik Edycja Widok Wyszukiwanie Terminal Pomoc
artur@Artur:~$ gcc casmc.c casma.s -m32 -o casm
casma.s: Assembler messages:
casma.s:32: Warning: koniec pliku nie jest na końcu wiersza; nowy wiersz wstawio
ny
artur@Artur:~$ ./casm
Tablica przed wywołaniem: 1 2 3 4 5
Po wywołaniu: 2 3 4 5 6
artur@Artur:~$

```

3. Zadanie 13.1.4 –

a) Kod programu

1) Plik: mylib.s – deklaracja funkcji read, write, exit, open, close

```
#Definicje stałych użytych w programie
.data
SYSEXIT = 1
SYSREAD = 3
SYSWRITE = 4
OPEN = 5
CLOSE = 6
STDIN = 0
STDOUT = 1
EXIT_SUCCESS = 0
READ_ONLY = 0

.text
.global write
.type write, @function      #Definicja funkcji wypisującej znaki
write:
push %ebp
movl %esp, %ebp
mov $SYSWRITE, %eax         #Ustawienie wartości %eax na 4 (wywołanie
wypisywania)
movl 8(%ebp), %ebx          #Pobranie drugiego parametru (uchwyt pliku)
movl 12(%ebp), %ecx         #Pobranie trzeciego parametru (adres bufora)
movl 16(%ebp), %edx         #Pobranie czwartego parametru (długość
bufora)
int $0x80                  #Wywołanie funkcji systemowej
movl %ebp, %esp
popl %ebp
ret

.global read
.type read, @function      #Definicja funkcji wczytującej znaki z
konsoli
read:
pushl %ebp
movl %esp, %ebp
mov $SYSREAD, %eax         #Ustawienie pierwszego parametru (wywołanie
wczytywania)
mov $STDIN, %ebx           #Ustawienie drugiego parametru (standardowe
wejście-konsola)
movl 8(%ebp), %ecx         #Pobranie trzeciego parametru (uchwyt pliku)
movl 12(%ebp), %edx         #Pobranie czwartego parametru (długość
bufora)
int $0x80                  #Wywołanie funkcji systemowej
movl %eax, %edi            #Przesłanie długości wczytanego ciągu do
%edi
sbb $1, %edi               #Pomniejszenie liczby o wczytany znak \n
movl %ebp, %esp
popl %ebp
ret

.global exit
.type exit, @function      #Definicja funkcji kończącej działanie
programu
exit:
pushl %ebp
movl %esp, %ebp
```

```

movl $1, %eax
zakończenia prog.)
movl 8(%ebp), %ebx
zwracana)
int $0x80
movl %ebp, %esp
popl %ebp
ret

.global open
.type open, @function
open:
push %ebp
movl %esp, %ebp
movl $OPEN, %eax
funkc. open)
movl 8(%ebp), %ebx
movl 12(%ebp), %ecx
movl 16(%ebp), %edx
dostępu)
int $0x80
movl %eax, %esi
movl %ebp, %esp
popl %ebp
ret

.global close
.type open, @function
close:
push %ebp
movl %esp, %ebp
movl $CLOSE, %eax
funkc. close)
movl 8(%ebp), %ebx
pliku)
int $0x80
movl %ebp, %esp
popl %ebp
ret

```

2) Plik: read write callmylib.s – wywołanie funkcji z biblioteki

[illegible]

```

leal msg_hello, %eax
pushl %eax                                #Przesłanie adresu bufora na stos
pushl $1                                  #Przesłanie 1 na stos (ustawienie
wyjścia na STDOUT)
call write                                #Wywołanie funkcji write

leal msg_echo_len, %eax
pushl %eax                                #Przesłanie długości bufora na stos
leal msg_echo, %eax
pushl %eax                                #Przesłanie adresu bufora na stos
pushl %eax                                #Przesłanie 1 na stos (ustawienie
call read                                #Wywołanie funkcji read

leal msg_echo_len, %eax
pushl %eax                                #Przesłanie długości bufora na stos
leal msg_echo, %eax
pushl %eax                                #Przesłanie adresu bufora na stos
pushl $1                                  #Przesłanie 1 na stos (ustawienie
wyjścia na STDOUT)
call write                                #Wywołanie funkcji write

leal newline_len, %eax
pushl %eax                                #Przesłanie długości bufora na stos
leal newline, %eax
pushl %eax                                #Przesłanie adresu bufora na stos
pushl $1                                  #Przesłanie 1 na stos (ustawienie
wyjścia na STDOUT)
call write                                #Wywołanie funkcji write

# Przykładowe użycie funkcji otwarcia i zamknięcia pliku

pushl $0666                               #Przesłanie wartości określającej prawa
dostępu na stos
pushl $03101                              #Przesłanie wartości określającej flagi na
stos
pushl $file_name                          #Przesłanie adresu pierwszego symbolu nazwy
pliku na stos
call open                                  #Wywołanie funkcji open

#Zapisanie do wprowadzonego ciągu znaków do pliku
#o nazwie określonej na początku programu
leal msg_echo_len, %eax
pushl %eax                                #Przesłanie długości bufora na stos
leal msg_echo, %eax
pushl %eax                                #Przesłanie adresu bufora na stos
pushl %esi                                #Przesłanie uchwytu do pliku na stos
call write                                #Wywołanie funkcji write

pushl %esi                                #Przesłanie uchwytu do pliku na stos
call close                                #Wywołanie funkcji close

pushl %edi                                #Przesłanie długości wprowadzonego ciągu
znaków na stos
call exit                                  #Wywołanie funkcji exit

```

b) Kompilacja programu:

```

$as mylib.s -32 -g -o mylib.o
$ar rcsv libmylib.a mylib.o

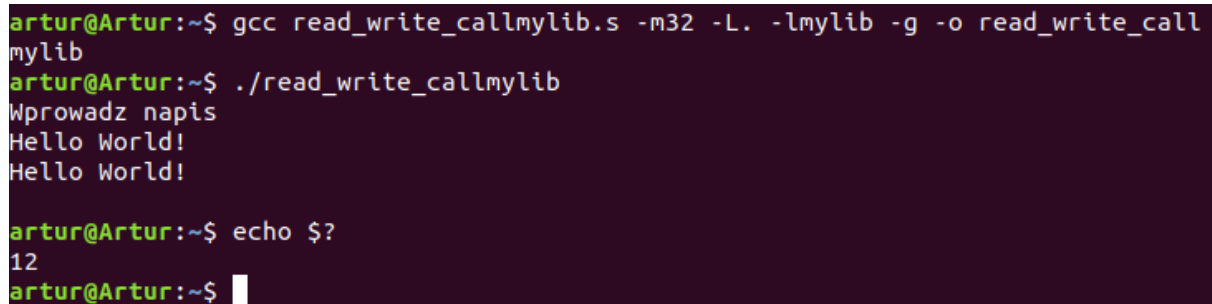
```

```
$gcc read_write_callmylib.s -m32 -L. -lmylib -g -o  
read_write_callmylib
```

Uruchomienie programu:

```
$/read_write_callmylib
```

c) Screen konsoli z działania programu



```
artur@Artur:~$ gcc read_write_callmylib.s -m32 -L. -lmylib -g -o read_write_call  
mylib  
artur@Artur:~$ ./read_write_callmylib  
Wprowadz napis  
Hello World!  
Hello World!  
  
artur@Artur:~$ echo $?  
12  
artur@Artur:~$
```

4. Zadanie 14.3.2

a) Kod programu

- 1) Plik: mylib.s – deklaracje funkcji w assemblerze, wykonane na wcześniejszych zajęciach

```
.data  
SYSEXIT = 1  
SYSREAD = 3  
SYSWRITE = 4  
OPEN = 5  
CLOSE = 6  
STDIN = 0  
STDOUT = 1  
EXIT_SUCCESS = 0  
READ_ONLY = 0  
  
.text  
.global write  
.type write, @function  
write:  
push %ebp  
movl %esp, %ebp  
mov $SYSWRITE, %eax  
movl 8(%ebp), %ebx #uchwył pliku  
movl 12(%ebp), %ecx #adres bufora  
movl 16(%ebp), %edx #długość ciągu  
int $0x80  
movl %ebp, %esp  
popl %ebp  
movl %edx, %eax  
ret  
  
.global read  
.type read, @function  
read:  
pushl %ebp  
movl %esp, %ebp  
mov $SYSREAD, %eax
```



```

int main()
{
    //Deklaracja otwierającego program napisu
    char open_message[] = "Podaj napis:\n";
    //Deklaracja bufora wczytywanego napisu o długości 80 znaków
    char read_message[80];
    int open_size, read_size, bufor_size;
    //Wyliczenie długości otwierającego napisu
    open_size = strlen(open_message);
    //Wywołanie funkcji wypisującej napis
    write(STDOUT, open_message, open_size);
    //Wywołanie funkcji wczytującej napis i zwracającej jego długość
    read_size = read(read_message, 80);
    //Wywołanie funkcji wypisującej napis
    write(STDOUT, read_message, read_size);
    //Pomniejszenie długości wczytanego napisu o wczytany znak nowej linii
    read_size = read_size - 1;
    //Wywołanie funkcji zamykającej program i zwracającej długość
    wczytanego napisu
    exit(1);
    return 0;
}

```

b) Kompilacja programu

```

$as mylib.s -32 -g -o mylib.o
$ar rcsv libmylib.a mylib.o
$gcc read_write_callmylib.s -m32 -L. -lmylib -g -o
read_write_callmylib

```

Uruchomienie programu:

```

$./read_write_callmylib

```

c) Screen konsoli z działania programu

```

artur@Artur: ~
Plik Edycja Widok Wyszukiwanie Terminal Pomoc
artur@Artur:~$ as mylib.s -32 -g -o mylib.o
mylib.s: Assembler messages:
mylib.s: Warning: koniec pliku nie jest na końcu wiersza; nowy wiersz wstawiony
artur@Artur:~$ ar rcsv libmylib.a mylib.o
r - mylib.o
artur@Artur:~$ nm libmylib.a

mylib.o:
0000005c T close
00000006 a CLOSE
00000032 T exit1
00000000 a EXIT_SUCCESS
00000043 T open
00000005 a OPEN
00000019 T read
00000000 a READ_ONLY
00000000 a STDIN
00000001 a STDOUT
00000001 a SYSEXIT
00000003 a SYSREAD
00000004 a SYSWRITE
00000000 T write
artur@Artur:~$ gcc read_write_callmylib.c -m32 -L. -lmylib -g -o read_write_cal
lmylib
artur@Artur:~$ ./read_write_callmylib
Podaj napis:
Architektura komputerow Sr. 11:15 TN
Architektura komputerow Sr. 11:15 TN
artur@Artur:~$ echo $?
36
artur@Artur:~$

```