ORGANIZACJA I ARCHITEKTURA KOMPUTERÓW

LAB 2

10 MAJA 2019
ŚRODA, TN 17:05
AUTOR: WOJCIECH KUR
PROWADZĄCY: DR INŻ. PIOTR PATRONIK

Spis treści

1. Treść ćwiczenia....3
1.1. Zakres i program ćwiczenia....3
1.2. Zrealizowane zadania....3
2. Przebieg ćwiczenia....4
2.1. Konstrukcja pliku źródłowego „print_args”....4
2.2. Konstrukcja pliku Makefile z funkcjami bibliotecznymi....6
2.3. Konstrukcja pliku źródłowego „printf_args”....7
2.4 Konstrukcja pliku źródłowego „adder”....8
3. Podsumowanie....13
4. Literatura....13

1. Treść ćwiczenia

1.1. Zakres i program ćwiczenia

1.1.1. Napisanie programu drukującego na standardowe wyjście:

a. parametry wywołania (w tym ścieżkę do programu).

b. zawartość środowiska (zmienne środowiskowe).

1.1.2. Napisanie programu dodającego/odejmującego dwie liczby o dowolnej precyzji.

1.1.3. Napisanie programu mnożącego/dzielącego dwie liczby o dowolnej precyzji.

1.1.4. Wejście/Wyjście dla programów:

a. standardowe we/wy, cyfry o bazie 16.

b. standardowe we/wy, cyfry o bazie 10.

1.1.5. Wczytywanie zrealizowane z wykorzystaniem:

a. int $0x80 – z wykorzystaniem funkcji systemu operacyjnego.

b. printf/scanf – z wykorzystaniem funkcji bibliotecznych.

c. parametrów programu i/lub zmiennych środowiskowych.

1.2. Zrealizowane zadania

1.2.1. Stworzenie programu drukującego parametry wywołania oraz zawartość środowiska na standardowe wyjście z wykorzystaniem funkcji systemowej oraz funkcji bibliotecznej printf.

1.2.2. Napisanie programu dodającego (oraz odejmującego z wykorzystaniem rozkazu sub zamiast add) dwie liczby o dowolnej precyzji z wykorzystaniem cyfr o bazie 16. Wczytywanie oraz wypisywanie zrealizowane za pomocą funkcji systemowych oraz funkcji bibliotecznych.

2. Przebieg ćwiczenia

2.1. Konstrukcja pliku źródłowego „print_args”

2.1.1. Listing 1: kod źródłowy print_args.s

```
.data
EXIT = 1
READ = 3
WRITE = 4
STDIN = 0
STDOUT = 1
SYSCALL = 0x80

new_line:
    .ascii "\n"
new_line_len = . - new_line

.text
.global _start
_start:
loading:
    pop                 %esi
    pop                 %esi
    mov %esi,           %ecx

isEnd:
    cmp $0x20,          %esi
    je  end

counter:
    inc %edx
    inc %esi
    cmpb $0,            (%esi)
    jne counter

    mov $WRITE,         %eax
    mov $STDOUT,        %ebx
    int $SYSCALL

    mov $WRITE,         %eax
    mov $STDOUT,        %ebx
    mov $new_line,      %ecx
    mov $new_line_len,  %edx
    int $SYSCALL

    mov $0,             %edx
    jmp loading

end:
```

```
mov $EXIT,          %eax
mov $0,             %ebx
int $SYSCALL
```

2.1.2. Opis programu

Program pobiera ze szczytu stosu jego początkową wartość (1), a następnie pobiera ciąg znaków zawierający dane informacje o programie. Instrukcja „cmp $0x20, %esi” sprawdza czy nastąpił koniec wypisywania zmiennych środowiskowych. Następnie wyliczona zostaje długość łańcucha poprzez iteracje bajt po bajcie aż do zakończenia znakiem 0. Po wyliczeniu łańcuch zostaje wypisany oraz zakończony znakiem nowej linii. Algorytm zapętla się i trwa dopóki nie zostanie spełniony warunek wystąpienia kodu 0x20.

2.2. Konstrukcja pliku Makefile z funkcjami bibliotecznymi

2.2.1 Listing 2: kod źródłowy Makefile

```
NAME=nazwa_zadania
all:
    as -g --32 ${NAME}.s -o ${NAME}.o
    ld -m elf_i386 -dynamic-linker /lib/ld-linux.so.2 -o ${NAME} ${NAME}.o -lc
    rm ${NAME}.o
```

2.2.2 Opis programu

Makefile oprócz flag zmieniających sposób kompilacji zgodny z 32bitowym systemem dodaje także dynamiczne linkowanie bibliotek. Umożliwia to wywoływanie zewnętrznych bibliotek w kodzie assemblerowym. Argumenty funkcji pobierane zostają wtedy ze stosu.

2.3. Konstrukcja pliku źródłowego „printf_args”

2.3.1 Listing 3: kod źródłowy printf_args.s

```
.data
EXIT = 1
READ = 3
WRITE = 4
STDIN = 0
STDOUT = 1
SYSCALL = 0x80

new_line:
   .ascii "\n"
new_line_len = . - new_line

.text
.global _start
_start:
loading:
   pop %esi
   pop %esi
   mov %esi,          %ecx

isEnd:
   cmp $0x20,         %esi
   je end

   push %ecx
   call printf
   pop %ecx

   push $new_line
   call printf
   pop %ecx
   jmp loading

end:
   mov $EXIT,         %eax
   mov $0,            %ebx
   int $SYSCALL
```

2.3.2. Opis programu

Program wykonuje te same polecenia co w listingu 1, jednakże funkcje systemowe zostały zastąpione funkcją biblioteczną **printf**. W celu wypisania elementu na standardowe wyjście wymagane jest przekazanie argumentu do funkcji (**printf**) poprzez dodanie elementu na stos. Po dodaniu zostaje ściągnięty, aby wskaźnik stosu powrócił na swoją poprzednią pozycję.

2.4 Konstrukcja pliku źródłowego „adder”

2.4.1 Listing 4: kod źródłowy adder.s

```
.data
EXIT =                  1
READ =                  3
WRITE =                 4
STDIN =                 0
STDOUT =                1
SYSCALL =               0x80

.bss
.comm input1,           512
.comm input2,           512
.comm value1,           256
.comm value2,           256
.comm sum,              256
.comm output,           515

.text
.global _start
_start:
   mov $256,            %esi
   mov $0,              %al

zerowanie:
   dec %esi
   mov %al,             value1(, %esi, 1)
   mov %al,             value2(, %esi, 1)
   cmp $0,              %esi
   jg zerowanie

#   PIERWSZY CIAG

   mov $READ,           %eax
   mov $STDIN,          %ebx
   mov $input1,         %ecx
   mov $512,            %edx
   int $SYSCALL

   mov %eax,            %esi
   # USUWANIE ZNAKU NOWEJ LINII
   dec %esi
   mov $256,            %edi

dekodowanie:
   dec %esi
   dec %edi
```

```
    mov input1(, %esi, 1),  %al

    cmp $'A',               %al
    jge litera

    sub $'0',               %al
    jmp dekodowanie_1

litera:
    sub $55,                %al

dekodowanie_1:
    cmp $0,                 %esi
    jle dekodowanie_3

    mov %al,                %bl
    dec %esi
    mov input1(, %esi, 1),  %al

    cmp $'A',               %al
    jge litera_2

    sub $'0',               %al
    jmp dekodowanie_2

litera_2:
    sub $55,                %al

dekodowanie_2:
    mov $16,                %cl
    mul %cl
    add %bl,                %al

dekodowanie_3:
    mov %al,                value1(, %edi, 1)

    cmp $0,                 %esi
    jg dekodowanie

#   DRUGI CIAG

    mov $READ,              %eax
    mov $STDIN,             %ebx
    mov $input2,            %ecx
    mov $512,               %edx
    int $SYSCALL

    mov %eax,               %esi
    dec %esi
    mov $256,               %edi
```

```
dekodowanie1:
    dec %esi
    dec %edi

    mov input2(, %esi, 1),  %al

    cmp $'A',               %al
    jge litera1

    sub $'0',               %al
    jmp dekodowanie1_1

litera1:
    sub $55,                %al

dekodowanie1_1:
    cmp $0,                 %esi
    jle dekodowanie1_3

    mov %al,                %bl
    dec %esi
    mov input2(, %esi, 1),  %al

    cmp $'A',               %al
    jge litera1_2

    sub $'0',               %al
    jmp dekodowanie1_2

litera1_2:
    sub $55,                %al

dekodowanie1_2:
    mov $16,                %cl
    mul %cl
    add %bl,                %al

dekodowanie1_3:
    mov %al,                value2(, %edi, 1)

    cmp $0,                 %esi
    jg dekodowanie1

# DODAWANIE
    clc
    pushf
    mov $255,                   %esi

dodawanie:
    mov value1(, %esi, 1),      %al
    mov value2(, %esi, 1),      %bl
```

```
    popf
    adc %bl,                    %al
    pushf
    mov %al,                    sum(, %esi, 1)

    dec %esi
    cmp $0,                     %esi
    jg dodawanie

# KONWERSJA HEX
    mov $255,                   %esi
    mov $513,                   %edi

konwersja:
    mov sum(, %esi, 1),         %al
    mov %al,                    %bl
    mov %al,                    %cl

    shr $4,                     %cl
    and $0b1111,                %bl
    and $0b1111,                %cl
    add $'0',                   %bl
    add $'0',                   %cl

    cmp $'9',                   %bl
    jle dalej
    add $7,                     %bl

dalej:
    cmp $'9',                   %cl
    jle dalej1
    add $7,                     %cl

dalej1:
    mov %bl,                    output(, %edi, 1)
    dec %edi
    mov %cl,                    output(, %edi, 1)
    dec %edi

    dec %esi
    cmp $0,                     %esi
    jge konwersja

# WYSWIETLANIE
    mov $514,                   %esi
    movb $0x0A,                 output(, %esi, 1)
    mov $WRITE,                 %eax
    mov $STDOUT,                %ebx
    mov $output,                %ecx
    mov $515,                   %edx
    int $SYSCALL
```

```
# ZAKONCZENIE
    mov $EXIT,              %eax
    mov $0,                 %ebx
    int $SYSCALL
```

2.4.2. Opis programu

Program rozpoczyna się od wyzerowania wartości w zarezerwowanych blokach pamięci. Następnie za pomocą funkcji systemowej/bibliotecznej zostaje wczytany pierwszy ciąg znaków odpowiadający pierwszej podanej liczbie. Następuje dekodowanie na liczbę w systemie o bazie 16 w formacie little endian. Druga liczba zostaje wczytana i dekodowana w ten sam sposób. Obie liczby zostają dodane do siebie bajt po bajcie, a wynik przekonwertowany do postaci łańcucha znaków i wyświetlony na standardowym wyjściu.

3. Podsumowanie

Zadanie pierwsze zostało zrealizowane w całości. Istotnym elementem było poznanie zawartości stosu po uruchomieniu programu i poruszania się po nim w odpowiedni sposób. Dodatkowo w celu skorzystania z zewnętrznych bibliotek wymagane było zapoznanie się z sposobem ich dodawania oraz wywoływania. Zadanie drugie skupiało się na poznaniu dwóch form przedstawiania danych little endian i big endian oraz sposobu konwersji łańcucha znaków na liczbę całkowitą i odwrotnie.

4. Literatura

4.1. http://zak.ict.pwr.wroc.pl/materials/architektura/laboratorium%20AK2/wzorzec%20sprawozdania.pdf, wzorzec sprawozdania

4.2. http://jedrzej.ulasiewicz.staff.iiar.pwr.wroc.pl/Architektura-Komputerow/lab/Architektura-63.pdf, laboratorium architektury komputerów – materiały dr Jędrzeja Ułasiewicza

4.3. https://pl.wikibooks.org/wiki/Asembler_x86, teoria oraz prosty program krok po kroku

4.4. http://quantum-mirror.hu/mirrors/pub/gnusavannah/pgubook/ProgrammingGroundUp-1-0-booksize.pdf?fbclid=IwAR0b_yzxqe1Ib9ANvA1BX5r4fFWKh6TarAiws4QtwKpikw2nGNYaYwcYwfI, Programming from the Ground Up

4.5. http://www.cs.umd.edu/~meesh/cmsc311/links/handouts/ia32.pdf

4.6. http://zak.ict.pwr.wroc.pl/materials/architektura/laboratorium%20AK2/Dokumentacja/Intel%20Penium%20IV/IA-32%20Intel%20Architecture%20Software%20Developers%20Manual%20vol.%202%20-%20Instruction%20Set%20Reference.pdf, dokumentacja intela (Instruction Reference vol.2)