

ASK	Dokumentacja projektu
Autor	Bartłomiej Król, 125136
Kierunek, rok	Informatyka, II rok, st. stacjonarne (3,5-I)
Temat projektu	Program implementujący grę w kamień, papier i szczyryk

Wersja na Linuxa	2
Wstęp	2
Opis programu	2
Wymagania Systemowe	2
System operacyjny:	2
Narzędzia:	2
Instalacja narzędzi:	2
Sekcje programu	3
Sekcja .data	3
Sekcja .bss	3
Sekcja .text	3
_start	4
Wyświetlenie menu:	4
Odczyt wyboru użytkownika:	4
Pobranie aktualnego czasu:	4
Wygenerowanie losowego wyboru komputera:	4
Wyświetlenie wyboru komputera:	4
Sprawdzenie wyniku gry:	4
Zakończenie programu:	4
Etykiety i Logika Gry	5
Użycie	5
Uruchomienie Skryptu	5
Kompilacja	5
Uruchomienie	5
Przykład:	6
Wersja na asmloadera	6
Opis ogólny	6
Struktura programu	6
Sekcje programu	6
Inicjalizacja trybu 32-bitowego	6
Wypisanie menu	6
Wczytanie wyboru użytkownika	7
Walidacja wyboru użytkownika	7
Losowanie wyboru komputera	8

Wypisanie komunikatu o losowaniu	9
Komunikaty	10
Sprawdzenie wyniku gry	11
Komunikaty o wyniku	12
Zakończenie programu	13
Przykład	14

Wersja na Linuxa

Wstęp

Opis programu

Program implementuje grę "Kamień, Papier, Nożyce" w języku Assembly dla systemu Linux. Użytkownik wybiera jeden z trzech przedmiotów (kamień, papier lub nożyce), a komputer losowo wybiera jeden z nich na podstawie aktualnego czasu systemowego. Program następnie porównuje wybory użytkownika i komputera, wyświetla wynik (wygrana, przegrana, remis) oraz kończy działanie.

Wymagania Systemowe

System operacyjny:

Unix-like (np. Linux, macOS)

Narzędzia:

NASM (Netwide Assembler),

LD (Linker)

Instalacja narzędzi:

```
sudo apt-get update
sudo apt-get install nasm
sudo apt-get install binutils
```

Sekcje programu

Sekcja .data

Ta sekcja zawiera stałe dane używane w programie, takie jak komunikaty oraz opisy przedmiotów.

menu: komunikat wyświetlany użytkownikowi przy wyborze przedmiotu.

choice: zmienna przechowująca wybór użytkownika.

comp_choice: zmienna przechowująca wybór komputera.

result_msg: komunikat informujący o wyborze komputera.

rock, paper, scissors: opisy przedmiotów (kamień, papier, nożyce).

win_msg, lose_msg, draw_msg: komunikaty wyświetlane w zależności od wyniku gry.

error_msg: komunikat wyświetlany w przypadku niepoprawnego wyboru przez użytkownika.

error_len: długość komunikatu błędu.

Sekcja .bss

Ta sekcja zawiera niezainicjalizowane zmienne używane w programie.

buffer: bufor na dane wejściowe od użytkownika.

time: bufor na czas systemowy.

Sekcja .text

Ta sekcja zawiera kod programu.

_start

Główna funkcja programu, która realizuje następujące kroki:

Wyświetlenie menu:

- Kod operacji `write` (4) wyświetla menu na standardowym wyjściu (1).
- `mov ecx, menu` i `mov edx, 50` ustawia adres i długość danych do wyświetlenia.

Odczyt wyboru użytkownika:

- Kod operacji `read` (3) odczytuje dane z standardowego wejścia (0).
- `mov ecx, buffer` i `mov edx, 2` ustawia adres bufora i liczbę bajtów do odczytu.
- Następuje walidacja danych.
- Wybór jest zapisywany w zmiennej `choice` lub zostaje wyświetlony komunikat o błędzie.

Pobranie aktualnego czasu:

- Kod operacji `time` (13) pobiera aktualny czas systemowy i zapisuje go w buforze `time`.

Wygenerowanie losowego wyboru komputera:

- Losowy wybór jest generowany na podstawie wartości czasu z bufora `time`, maskowany, aby uzyskać zakres 0-3, i dopasowany do zakresu ASCII '1'-'3'. Wynik jest zapisywany w zmiennej `comp_choice`.

Wyświetlenie wyboru komputera:

- Kod operacji `write` (4) wyświetla komunikat o wyborze komputera na standardowym wyjściu (1).
- Na podstawie wartości `comp_choice` wyświetlany jest odpowiedni opis przedmiotu (kamień, papier, nożyce).

Sprawdzenie wyniku gry:

- Wybór użytkownika i komputera jest porównywany. Na tej podstawie wyświetlany jest odpowiedni komunikat (wygrana, przegrana, remis).

Zakończenie programu:

- Kod operacji `exit` (1) kończy działanie programu.

Etykiety i Logika Gry

`.comp_rock`, `.comp_paper`, `.comp_scissors`: Etykiety odpowiedzialne za wyświetlenie wyboru komputera (kamień, papier, nożyce).

`.check_result`: Etykieta, która porównuje wybory użytkownika i komputera, określając wynik gry.

`.draw`, `.user_wins`, `.user_loses`: Etykiety odpowiedzialne za wyświetlenie odpowiednich komunikatów w zależności od wyniku gry.

`.valid_choise`: Zapis wyboru użytkownika.

Użycie

Uruchomienie Skryptu

Kompilacja

Program należy skompilować używając asemlera NASM i linkera. Otwórz terminal w katalogu z plikiem `gra.asm`. Następnie wpisz następujące komendy:

```
nasm -f elf64 gra.asm -o gra.o
```

```
ld -m elf_i386 gra.o -o gra
```

Uruchomienie

Po skompilowaniu program można uruchomić w terminalu komendą:

```
./gra
```

Przykład:

```
bartek@komputerek:/mnt/c/Users/krolb/OneDrive/Pulpit/asembler$ nasm -f elf32 gra.asm -o gra.o
bartek@komputerek:/mnt/c/Users/krolb/OneDrive/Pulpit/asembler$ ld -m elf_i386 gra.o -o gra
bartek@komputerek:/mnt/c/Users/krolb/OneDrive/Pulpit/asembler$ ./gra
1. Kamien
2. Papier
3. Scyzoryk
Wybierz przedmiot:3
Komputer wylosowal scyzoryk i zremisowal.
bartek@komputerek:/mnt/c/Users/krolb/OneDrive/Pulpit/asembler$
```

Wersja na asmloadera

Opis ogólny

Program realizuje grę "Kamień, Papier, Nożyce" w trybie 32-bitowym. Użytkownik wybiera jedną z trzech opcji, a komputer losuje swój wybór. Następnie program porównuje wybory i ogłasza wynik gry: wygraną, przegraną lub remis.

Struktura programu

Program składa się z kilku głównych części:

1. Wypisanie menu
2. Wczytanie wyboru użytkownika
3. Losowanie wyboru komputera
4. Porównanie wyborów i ogłoszenie wyniku
5. Wypisanie komunikatów i zakończenie programu

Sekcje programu

Inicjalizacja trybu 32-bitowego

```
[bits 32] ; Ustawia tryb na 32 bity.
```

Wypisanie menu

call print_menu ; Wywołuje funkcję wypisz_menu, która wypisuje menu wyboru

db "1. Kamien", 10, "2. Papier", 10, "3. Nozyce", 10, "Wybierz przedmiot:", 0 ; Definiuje string z opcjami do wyboru i kończy go zerem

print_menu:

call [ebx+3*4] ; Wywołuje funkcję printf do wypisania menu

add esp, 4 ; Czyści stos po wywołaniu funkcji (zdejmuje 4 bajty)

Wczytanie wyboru użytkownika

mov ebp, esp ; Ustawia wskaźnik bazowy (EBP) na wskaźnik stosu (ESP), co jest typowym początkiem funkcji.

sub esp, 8 ; Rezerwuje 8 bajtów na stosie dla dwóch zmiennych (po 4 bajty na zmienną).

lea eax, [ebp-4] ; Ładuje adres pierwszej zmiennej (4 bajty poniżej EBP) do rejestru EAX.

push eax ; Umieszcza adres pierwszej zmiennej na stosie.

call do_scan ; Wywołuje funkcję do_scan, która wczytuje liczbę od użytkownika.

db "%i", 0 ; Definiuje ciąg formatu dla funkcji do_scan.

do_scan: ; Etykieta dla funkcji do_scan.

call [ebx+4*4] ; Wywołuje funkcję scanf (lub odpowiednik), która wczytuje liczbę od użytkownika.

add esp, 8 ; Czyści stos po wywołaniu funkcji (zdejmuje 8 bajtów).

Walidacja wyboru użytkownika

```

mov eax, [ebp-4]          ; Ładuje wczytaną wartość do
rejestru EAX

cmp eax, 1                ; Porównuje wartość z '1'

je valid_choice           ; Skok do etykiety valid_choice
jeśli równa się '1'

cmp eax, 2                ; Porównuje wartość z '2'

je valid_choice           ; Skok do etykiety valid_choice
jeśli równa się '2'

cmp eax, 3                ; Porównuje wartość z '3'

je valid_choice           ; Skok do etykiety valid_choice
jeśli równa się '3'


call print_error          ; Wywołuje funkcję wypisz_error
jeśli wartość jest nieprawidłowa

db "Podano złe dane. Musisz wybrać 1, 2 lub 3.", 0xA, 0 ;
Definiuje string z komunikatem o błędzie i kończy go
zerem

print_error:

call [ebx+3*4]            ; Wywołuje funkcję printf do
wypisania błędu

add esp, 4                ; Czyści stos po wywołaniu
funkcji (zdejmuje 4 bajty)

jmp final                 ; Skok do końca programu

```

Losowanie wyboru komputera

```

valid_choice:             ; Etykieta dla poprawnego
wyboru

RDRAND eax                ; Generuje losową liczbę i
zapisuje ją w EAX

xor edx, edx              ; Czyści rejestr EDX

mov ecx, 3                ; Ustawia ECX na 3

```



```
div ecx                ; Dzieli EAX przez ECX (3),
wynik w EAX, reszta w EDX
add edx, 1             ; Dodaje 1 do reszty (EDX), by
mieć zakres 1-3
mov [ebp-8], edx       ; Zapisuje wynik losowania
(1-3) w zmiennej na stosie
```

Wypisanie komunikatu o losowaniu

```
cmp edx, 1             ; Porównuje wylosowaną wartość
z 1
je rock                ; Skok do etykiety kamien jeśli
równa się 1
cmp edx, 2             ; Porównuje wylosowaną wartość
z 2
je paper               ; Skok do etykiety papier jeśli
równa się 2
cmp edx, 3             ; Porównuje wylosowaną wartość
z 3
je scissors            ; Skok do etykiety nozyce
jeśli równa się 3
```

rock:

```
    call msg_rock      ; Wywołuje funkcję msg_kamien
jeśli wylosowano kamień
    jmp check_result   ; Skok do sprawdzenia wyniku
```

paper:

```
    call msg_paper     ; Wywołuje funkcję msg_papier
jeśli wylosowano papier
    jmp check_result   ; Skok do sprawdzenia wyniku
```

scissors:

```
    call msg_scissors          ; Wywołuje funkcję msg_nozyce  
jeśli wylosowano nożyce  
    jmp check_result          ; Skok do sprawdzenia wyniku
```

Komunikaty

msg_rock:

```
    call  print_msg_rock      ; Wywołuje funkcję  
wypisz_msg_kamien
```

```
    db "Komputer wylosował kamien ", 0 ; Definiuje string z  
komunikatem o kamieniu
```

print_msg_rock:

```
    call [ebx+3*4]            ; Wywołuje funkcję printf do  
wypisania komunikatu
```

```
    add esp, 4                ; Czyści stos po wywołaniu  
funkcji (zdejmuje 4 bajty)
```

```
    ret                      ; Powrót z funkcji
```

msg_paper:

```
    call  print_msg_paper    ; Wywołuje funkcję  
wypisz_msg_papier
```

```
    db "Komputer wylosował papier ", 0 ; Definiuje string z  
komunikatem o papierze
```

print_msg_paper:

```
    call [ebx+3*4]            ; Wywołuje funkcję printf do  
wypisania komunikatu
```

```
    add esp, 4                ; Czyści stos po wywołaniu  
funkcji (zdejmuje 4 bajty)
```

```
    ret                      ; Powrót z funkcji
```

msg_scissors:

```
    call  print_msg_scissors  ; Wywołuje funkcję  
wypisz_msg_nozyce
```

```

    db "Komputer wylosował nozyce ", 0 ; Definiuje string z
komunikatem o nożycach
print_msg_scissors:
    call [ebx+3*4]          ; Wywołuje funkcję printf do
wypisania komunikatu
    add esp, 4              ; Czyści stos po wywołaniu
funkcji (zdejmuje 4 bajty)
    ret                    ; Powrót z funkcji

```

Sprawdzenie wyniku gry

```

check_result:
    mov eax, [ebp-4]        ; Ładuje wartość wyboru
użytkownika do rejestru EAX
    mov edx, [ebp-8]        ; Ładuje wartość wyboru
komputera do rejestru EDX

    CMP eax, edx            ; Porównuje wybór użytkownika z
wybozem komputera
    JE draw                ; Skok do etykiety remis, jeśli
wartości są równe

    CMP eax, 1             ; Porównuje wybór użytkownika z
1
    JE EQ1                 ; Skok do EQ1, jeśli wybór to 1
    CMP eax, 2             ; Porównuje wybór użytkownika z
2
    JE EQ2                 ; Skok do EQ2, jeśli wybór to 2
    CMP eax, 3             ; Porównuje wybór użytkownika z
3
    JE EQ3                 ; Skok do EQ3, jeśli wybór to 3

```

EQ1:

```
CMP edx, 2                ; Porównuje wybór komputera z 2
JE comp_won               ; Skok do gracz2_wygrał, jeśli
komputer wybrał 2
CMP edx, 3                ; Porównuje wybór komputera z 3
JE player_won             ; Skok do gracz1_wygrał, jeśli
komputer wybrał 3
```

EQ2:

```
CMP edx, 1                ; Porównuje wybór komputera z 1
JE player_won             ; Skok do gracz1_wygrał, jeśli
komputer wybrał 1
CMP edx, 3                ; Porównuje wybór komputera z 3
JE comp_won               ; Skok do gracz2_wygrał, jeśli
komputer wybrał 3
```

EQ3:

```
CMP edx, 1                ; Porównuje wybór komputera z 1
JE comp_won               ; Skok do gracz2_wygrał, jeśli
komputer wybrał 1
CMP edx, 2                ; Porównuje wybór komputera z 2
JE player_won             ; Skok do gracz1_wygrał, jeśli
komputer wybrał 2
```

Komunikaty o wyniku

player_won:

```
call print_player_won
                                ; Wywołuje funkcję
wypisz_gracz1_wygrał
db "i przegrał", 0xa, 0 ; Definiuje string z
komunikatem, że gracz 1 przegrał
```

```

    print_player_won:
        call [ebx+3*4]          ; Wywołuje funkcję printf do
wypisania komunikatu
        add esp, 4              ; Czyści stos po wywołaniu
funkcji (zdejmuje 4 bajty)
        jmp final              ; Skok do końca programu

```

```

draw:
    call print_draw            ; Wywołuje funkcję wypisz_remis
        db "i mamy remis", 0xa, 0 ; Definiuje string z
komunikatem o remisie
    print_draw:
        call [ebx+3*4]          ; Wywołuje funkcję printf do
wypisania komunikatu
        add esp, 4              ; Czyści stos po wywołaniu
funkcji (zdejmuje 4 bajty)
        jmp final              ; Skok do końca programu

```

```

comp_won:
    call print_comp_won
                                ; Wywołuje funkcję
wypisz_gracz2_wygral
        db "i wygral", 0xa, 0    ; Definiuje string z
komunikatem, że gracz 2 wygrał
    print_comp_won:
        call [ebx+3*4]          ; Wywołuje funkcję printf do
wypisania komunikatu
        add esp, 4              ; Czyści stos po wywołaniu
funkcji (zdejmuje 4 bajty)
        jmp final              ; Skok do końca programu

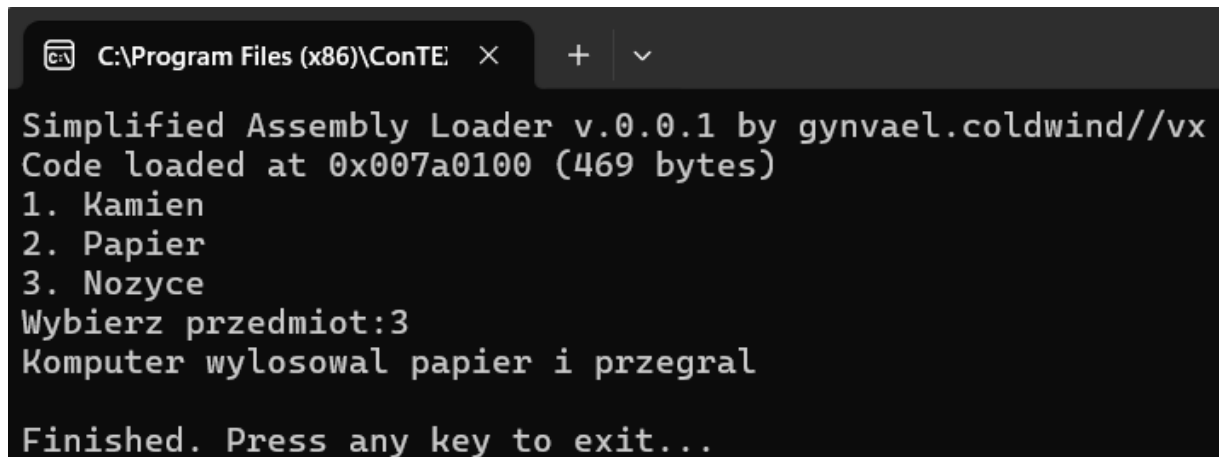
```

Zakończenie programu

final:

```
    push 0                                ; Umieszcza 0 na stosie, co  
    jest wartością zwrotu exit(0)  
    call [ebx+0*4]                        ; Wywołuje funkcję exit(0)  
    kończąca program
```

Przykład



```
C:\Program Files (x86)\ConTE  ×  +  v  
Simplified Assembly Loader v.0.0.1 by gynvael.coldwind//vx  
Code loaded at 0x007a0100 (469 bytes)  
1. Kamien  
2. Papier  
3. Nozyce  
Wybierz przedmiot:3  
Komputer wylosowal papier i przegral  
  
Finished. Press any key to exit...
```