Realizacija igrice „Vešala” u asembleru

Stefan Tešanović 675/2016, *Univerzitet u Beogradu, Elektrotehnički fakultet*, Predrag Mitrović 678/2016, *Univerzitet u Beogradu, Elektrotehnički fakultet*

***Abstract* — U ovom radu je opisana realizacija igrice „Vešala” u *Microsoft visual studio 2015*, koristeći se asemblerskim programskim jezikom, *Kip Irvine's i MASM* bibliotekama. Demonstrirano je osnovno poznavanje rada u softverskom alatu, kao i napredno poznavanje asemblera i rada sa bibliotečkim funkcijama.**

***Keywords* — visual studio 2015, irvine32, masm, assembly, hangman.**

I. Uvod

Ovaj rad predstavlja izveštaj za projekat pod rednim brojem 13 za školsku godinu 2016/2017 na predmetu računarska elektronika na 3. godini odseka za elektroniku. Osnovna ideja ovog rada bila je izrada igrice „Vešala“ koristeći se *Microsoft visual studio 2015* kao softverskim alatatom kao i određenih biblioteka radi lakše realizacije projektnog zadatka.

II. Projektni zadatak

Potrebno je realizovati igru „Vešala“ (*Hangman*). Primer izgleda prozora prikazan je na slici 1. Potrebno je uneti niz pojmova koji se pogađaju u igrici (minimum 10) i omogućiti igraču da unošenjem slova sa standardnog ulaza pogađa pojam. Ukoliko slovo koje je uneseno sa standardnog ulaza ne postoji, dodaje se deo Čiča Gliše, dok ukoliko postoji upisuje se na odgovarajuću poziciju u reči. Ukoliko je slovo već ranije pogađano, potrebno je o tome obavestiti korisnika, i ne preduzimati nikakve druge akcije. Svaki put kada se sa standardnog ulaza unese novo slovo, potrebno ga je ispisati na ekranu. Kada korisnik pogodi trenutni pojam, omogućava mu se da pogađa novi. Ukoliko korisnik pogreši slova dovoljan broj puta, tako da se iscrta ceo Čiča Gliša, gubi igru. Igricu je moguće prekinuti i pritiskom tastera ESC.

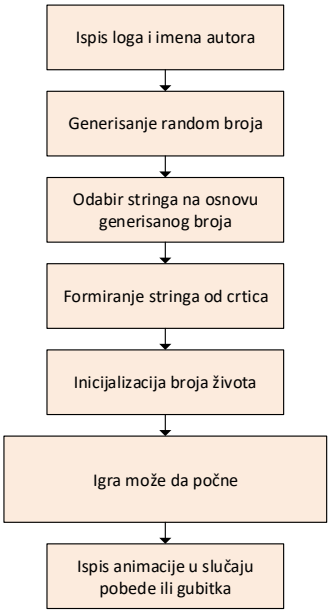
Slika. 1. Izgleda prozora igrice „Vešala“.

Stefan Tešanović sa Univerziteta u Beogradu, Elektrotehnički fakultet, odsek elektronika, Bul. kralja Aleksandra 73, 11120 Belgrad, Srbija (e-mail: tesanovic.stefan@yahoo.com).

Predrag Mitrović sa Univerzizeta u Beogradu, Elektrotehnički fakultet, odsek elektronika, Bul. kralja Aleksandra 73, 11120 Belgrad, Srbija (e-mail: mp160678d@student.etf.rs).

III. Opis projektnog koga

Opis projektnog koda i princip rada igrice je najlakše opisati preko dijagrama. Na slici 2. je dat pojednostavljen dijagram toka programa. Svaki od segmenata dijagrama će biti objašnjen u posebnom poglavlju.



Slika. 2. Pojednostavljen dijagram programa.

A. Ispis loga i imena autora

Na samom početku programa ispisujemo logo kao niz stringova, koji su defnisani kao na slici 3. Promenjiva endl, prestavlja string u koji smo upisali dva znaka koji predstavljaju prelazak u novi red. U Promenjivoj messageSize je sačuvana duzina ovog stringa.

```
endl EQU <0dh,0ah>  ; end of line sequence
message LABEL BYTE
    BYTE "                                                                  " ,endl
    BYTE "                                                                  " ,endl
    BYTE "                                                                  " ,endl
    BYTE "     |  | |  ||   _   ||  |  | ||       ||  |_|  ||   _   ||  |  | |" ,endl
    BYTE "     |  |_|  ||  |_|  ||  |_| ||  ___  ||       ||  |_|  ||  |_| | " ,endl
    BYTE "     |       ||       ||       ||  | __ |       ||       ||       | " ,endl
    BYTE "     |       ||       ||   _   ||  ||  ||       ||       ||   _   | " ,endl
    BYTE "     |   _   ||   _   ||  | |  ||  |_| ||  ||_||  ||   _   ||  | |  | " ,endl
    BYTE "     |__| |__||__| |__||__|  |__||_____||__|   |__||__| |__||__|  |__|" ,endl
    BYTE "                                                                  " ,endl
    BYTE "     Copyright (C) 2017  Stefan TESANOVIC 2016/675                " ,endl
    BYTE "                         Predrag MITROVIC 2016/678                " ,endl
    BYTE "                                                                  " ,endl
messageSize DWORD ($-message)
consoleHandle HANDLE 0     ; handle to standard output device
bytesWritten  DWORD ?      ; number of bytes written
```

Slika. 3. Definisanje stringa loga.

Za ispis stringa koristimo biblioteku *WriteConsole*. Ona ispisuje niz karaktera na ekranu počev od trenutne pozicije kursora i pomera kursor na poziciju poslednjeg ispisanog znaka. Da bi ova funkcija mogla da se koristi pre njenog korišćenja potrebno je uzeti pristup za standardni ulaz ili izlaz. Funkcijom *GetStdHandle* uzimamo *handle* za standardni ulaz i izlaz, koji posle mogu da koriste neke od

sledećih funkcija: *ReadFile*, *WriteFile*, *ReadConsoleInput*, *WriteConsole* ili *GetConsoleScreenBufferInfo*.

```
; Get the console output handle:
  INVOKE GetStdHandle, STD_OUTPUT_HANDLE
  mov consoleHandle,eax

; Write a string to the console:
  INVOKE WriteConsole,
      consoleHandle,              ;console output handle
      ADDR message,               ; string pointer
      messageSize,                ; string length
      ADDR bytesWritten,          ; returns num bytes written
      0                           ; not used
```

Slika. 4. Kod za stampanje stringa loga.

B. *Generisanje random broja*

Generisanje slučajnog broja smo postigli uspomoć dve *Irvine* funkcije. Funkcija *Randomize* restartuje slučajno generisani broj na osnovu trenutnog vremena. Ukoliko ne bi koristili ovu funkciju imali bi stalno isti slučajno generisan broj. Funkcija *RandomRange* generiše neoznačeni pseudo-slučajan 32bitni *integer* u opsegu od 0 do N-1. U našem slučaju ona generiše broj od 0 do 9. Kod koji ovo realizuje prikazan je na slici 5.

```
;Part of code for generate random number from 0 until 9
  mov  eax,10         ;get random 0 to 9
  call Randomize      ;re-seed generator
  call RandomRange
  mov  ranNum,eax     ;save random number
```

Slika. 5. Kod generisanje slučajnog broja.

C. *Odabir reči koju pogađamo*

Na osnovu slučajno generisanog broja potrebno je da odaberemo jednu od 10 reči koje smo prethodno definisali. Na slici 6. je prikazana deklaracija stringa svih reči koje je moguće izabrati.

```
;All words what is posible to guess.
;Pick by random generartor and put in selectedWords
manyWords   BYTE "BICYCLE", 0
            BYTE "CANOE", 0
            BYTE "SCATEBOARD", 0
            BYTE "OFFSIDE", 0
            BYTE "TENNIS", 0
            BYTE "SOFTBALL", 0
            BYTE "KNOCKOUT", 0
            BYTE "CHALLENGE", 0
            BYTE "SLALOM", 0
            BYTE "MARATHON", 0
            BYTE 0                  ; End of list
len equ $ - manyWords
```

Slika. 6. Deklaracija stringa svih reči.

U glavnom programu je potrebno da slučajno generisan broj smestimo u registar EDX i pozovemo funkciju *find_str*. Funkcija vraća EDI, pokazivač na string. Nakon što smo dobili pokazivač na datu reč, potrebno je da tu reč upišemo u promenjivu *selectedWords* koja predstavlja izabranu reč koju tražimo. Za kopiranje smo iskoristili *Irvine* biblioteku *Str_copy*.

```
;Copy find world in variable selectedWords
  INVOKE Str_copy,
      ADDR [edi],
      ADDR selectedWords
```

Slika. 7. Primena funkcije *Str_copy*.

Na slici 7. je prikazan kod u glavnom programu za kopiranje reči na koju prokazuje registar EDI u promenjivu *selectedWords*. Da bi smo na ovaj način mogli da biramo izabranu reč bilo je potrebno da u deklaraciji posle svake reči ide znak 0. Funkcija iz *Irvine* biblioteke *Str_copy* kopira string od trenutne pozicije na koju polazuje registar EDI do 0 tj funkcija *Str_copy* kopira string *null-terminated* od izvorišne lokacije na novu lokaciju.

```
find_str PROC                   ; ARG: EDX = index
    lea edi, manyWords          ; Address of string list

    mov ecx, len                ; Maximal number of bytes to scan
    xor al, al                  ; Scan for 0

    @@:
    sub edx, 1
    jc done                     ; No index left to scan = string found
    repne scasb                 ; Scan for AL
    jmp @B                      ; Next string

  done:
    ret

find_str ENDP                   ; RESULT: EDI pointer to string[edx]
```

Slika. 8. Kod funkcije *find_str*.

D. *Formiranje stringa od crtica*

Nakon što smo izabrali reč koju pogađamo, potrebno je da formiramo niz crtica koji će biti iste dužine kao i izabranu reč. Na ekranu ćemo u prvoj interakciji ispisati sve crtice, a kasnije kako igrač bude unosio slova svako, zamenjivati odgovarajuće crtice slovima, na odgovarajućim mestima kao i u izabranoj reči.

U glavnom programu je potrebno pozvati funkciju *make_array_dash* koja formira niz crtica. Na slici 9. je prikazan kod funcije.

```
make_array_dash PROC
    mov  edx,OFFSET selectedWords
    call StrLength              ; Length of a null-terminated string pointed to by EDX
    mov  lengthArray,eax

    mov al, '-'                 ; Default charcter for guessWords
    mov ecx, lengthArray        ; REP counter
    mov edi, offset guessWords  ; Destination
    rep stosb                   ; Build guessWords
    mov BYTE PTR [edi], 0       ; Store the null termination

    ret
make_array_dash ENDP
```

Slika. 9. Kod funkcije *make_array_dash*.

Na početku funkcije je potrebno da odredimo dužinu reči koju smo izbrali. To smo uradili pomoću *Irvine* funkcije *StrLength* koja vraća dužinu *null-terminated* smestenog u EDX registar. Nakon njenog izrvešavanja dužina reči je smeštena u registar EAX. Nakon toga koristimo instrukciju *stosb* za smeštanje odgovarajućeg broja crtica u promenjivu *guessWords*. Zbog daljeg rada sa stringovima potrebno je da na kraj stringa *guessWords* dodamo 0, kako bi on bio *null-terminated* i sebi olakšali pretraživanje i rad sa stringovima.

Sama zamena određenih crtica slovima se radi u okviru dela igre i biće opisana u posebnom poglavlju.

E. *Inicijalizacija broja života i crtanje figura.*

Jedan od zahteva prilikom izrade ovog projekta je da na ekranu (standardnom izlazu) ispisujemo Čiča Glišu tj figure. Zbog toga smo odlučili da u promenjivoj *statusGameLive* čuvano broj života igrača i da u zavisnosti od broja zivota crtamo odgovarajuću figuru. Crtanje figura

smo realizovali pomoću funkcije *print_hangman_live*. Funkcija *print_hangman_live* je krajnje jednostavno realizovana. Ona vrši jednostavno poređenje vrednosti *statusGameLive* i u zavisnosti od njene vrednosti skače na određenu labelu. Na slici 10. je prikazana definicija jedne od figura.

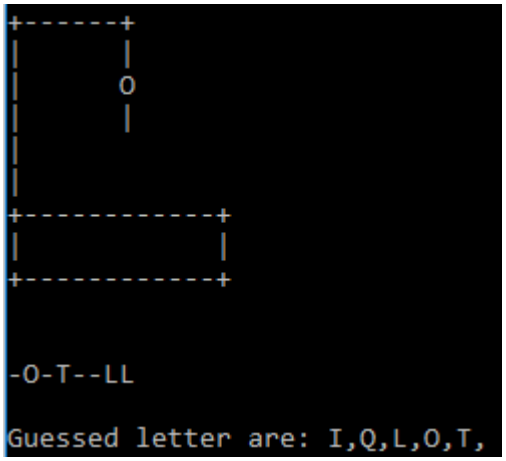
```
HANGMAN_LIVES_02 LABEL BYTE
                 BYTE "+------+       ",endl
                 BYTE "|      |       ",endl
                 BYTE "|      O       ",endl
                 BYTE "|     /|\      ",endl
                 BYTE "|              ",endl
                 BYTE "|              ",endl
                 BYTE "+------------+ ",endl
                 BYTE "|            | ",endl
                 BYTE "+------------+ ",endl
```

Slika. 10. Definicija jedne od figura.

Nakon što smo odredili na koju labelu skačemo, u njoj vršimo crtanje Čiča Gliče (figure), ispisujemo reč koju ponađamo u obliku crtica (slova koja su pogođena su prikazana) i niz slova koje smo uneli do sad odvojeni zarezom. Kod kojim je to realizovano je prikazano na slici 11., a ispis koji ona realizuje je prikazan na slici 12.

```
live_2:   ; Write a string to the console:
    INVOKE WriteConsole,
        consoleHandle,              ;console output handle
        ADDR HANGMAN_LIVES_02,      ; string pointer
        messageSizeGoodGame,        ; string length
        ADDR bytesWritten,          ; returns num bytes written
        0                           ; not used
    call Crlf                       ; new line
    call Crlf                       ; new line
    mov edx, offset guessWords
    call WriteString                ; write a string pointed to by EDX
    call Crlf                       ; new line
    call Crlf                       ; new line
    mWrite <"Guessed letter are: ">
    mov edx, offset guessLetterArray
    call WriteString                ; write a string pointed to by EDX
    call Crlf                       ; new line
    call Crlf                       ; new line
    ret
```

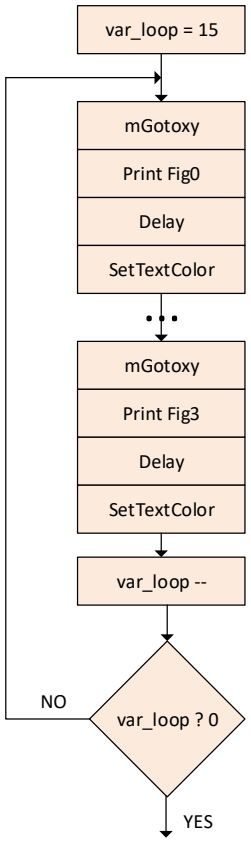
Slika. 11. Kod u slučaju da je igraču ostala 2 života.



Slika. 12. Prikaz ispisa na standardnom izlazu.

F. Ispis animacije u slučaju pobede ili gubitka

U slučaju da je igrač pobedio ili izgubio potrebno je da iscrtamo određenu figuru na ekranu. Ono za šta smo se mi odlučili je da animiramo taj prikaz uspomoć *Irvine* funkcije *Delay*, makroa *mGotoxy* koji poziva *Gotoxy* funkciju iz *Irvine* biblioteke. Na slici 13. je prikazana dijagram ove realizacija, a na kraju izveštaja se nalazi kod za ovu realizaciju. Sam kod nećemo posebno analizirati zbog jednostavnosti,a i svi ti delovi koda su analizirani posebno u prethodnim poglavljima.



Slika. 13. Dijagram realizacije crtanja animacije

IV. REALIZACIJA IGRE

Dijagram koji predstavlja srz ovog projekta i u sebi objedinjuje više funkcija je predstavljen na slici 15. Početak igre kreće ispisom Čiča Gliše na stangardni izlaz. Nakon toga je potrebno proveriti koliko života je preostalo igraču. U slučaju da mu nije preostao ni jedan život potrebno je skočiti na perlju *loop_game_over*, vršimo pokretanje animacije za gubitak igre u trajanju od jednog minuta i nakon toga izlazimo iz igre. U suprotnom nastavljamo sa izvešavanjem programa. U promenjivoj *statusGameLive* se čuva broj života.

Na slici 14. je prikazan kod koji služi za učitavanje slova sa tastature. Prvo što treba da proverimo nakon učitavanja slova sa standardnog ulaza je da li je to ESC. U slučaju da je pritisnut taster ESC tj da uneti znak ima vrednost 27, potredno je da obustavimo izvršavanje programa. U suprotnom nastavljamo izvršavanje programa.

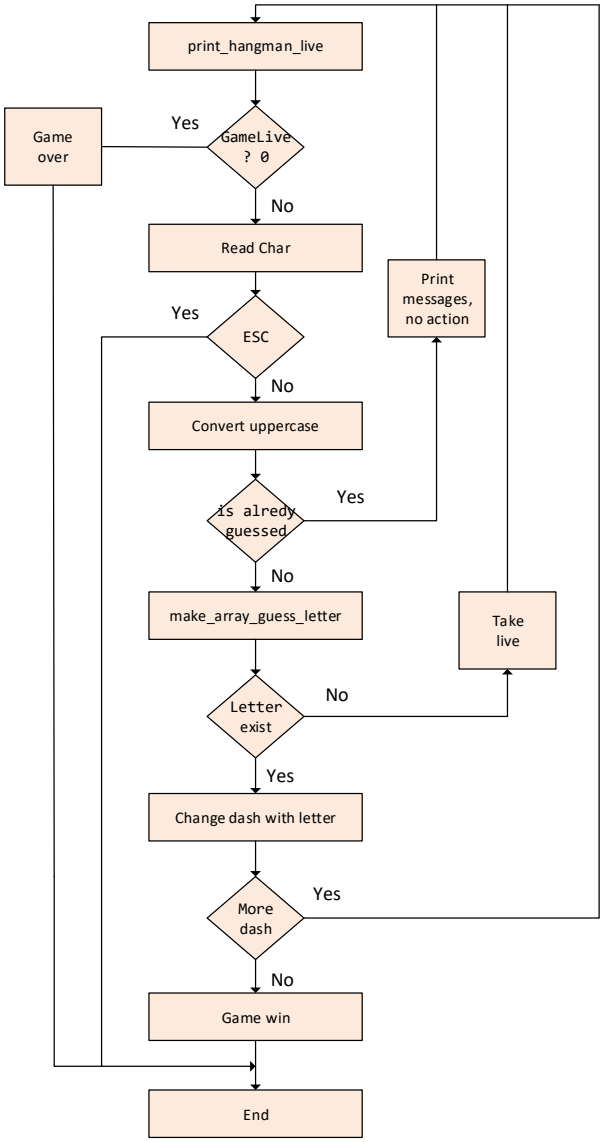
```
    mov  eax,green+(black*16)
    call SetTextColor

    mWrite <"Guess a letter: ">

    call readChar   ;User inputs char
    cmp al, 27      ;Check if is press ESC
    je exit_main    ;YES, end game
    and al, 0DFH    ;Convert lowercase input to uppercase.
                    ;If uppercase, it remains uppercase
    push eax
    sub al, 'A'     ;checks if it is a letter
    cmp al, 'Z'-'A'
    jbe uppercase
    jmp again_input_world
uppercase:
    pop eax
    mov guessLetter, al
    call WriteChar
    call Crlf       ;new line
    call Crlf       ;new line

    mov  eax,white+(black*16)
    call SetTextColor
```

Slika. 14. Kod za realizaciju učitavanja slova



Slika. 15. Dijagram toka igre

Kako se u promenjivima sa kojima radimo nalaze reči tj stringovi koji se sastoje isključivo od velikih slova, potrebno je da izvršimo ispitivanje da li je uneto veliko ili malo slovo. Ukoliko je uneto malo slovo, potrebno je izvršiti konverziju u veliko slovo. Takođe u ovom delu programa se vrši ispis poruke da se unese slovo i ispis samog unetog slova. Ispis se vrši zelenom bojom radi lakše uočljivosti igrača.

```
;Check if letter is alredy guessed
mov ecx, LENGTHOF guessLetterArray
mov edi, offset guessLetterArray
mov al, guessLetter                 ; Load character to find
repne scasb                         ; Search
je loop_guess_letter_exists         ; Letter already exist


call make_array_guess_letter
```

Slika. 16. Kod za proveru da li je slovo već pogađano

Dalje je potrebno proveriti da li je uneto slovo već bilo pogađano ili ne. To je realizovano preko koda koji je prikazan na slici 16. U promenjivoj *guessLetterArray* se čuvaju sva slova koja su uneta sa standardnog ulaza. U slučaju da je slovo već bilo pogađano potrebno je skočiti preko petlje *loop_guess_letter_exists*, ispisati odgovarajuću poruku i vratiti se na početak za ponovni upis slova. Kod koji vrši ispis poruke na standardni izlaz je prikazan na slici 17.

```
loop_guess_letter_exists:

        mov  eax,red+(black*16)
        call SetTextColor

        mWrite <"Sorry, you alredy guessed letter, ">
        mov al, guessLetter
        call WriteChar
        call Crlf                       ; new line
        mWrite <"I repeat you one more time the letter what you guessed. ">
        call Crlf                       ; new line
        mWrite <"Guessed letter are: ">
        mov edx, offset guessLetterArray
        call WriteString                ; write a string pointed to by EDX
        call Crlf                       ; new line
        call Crlf                       ; new line

        mov  eax,white+(black*16)
        call SetTextColor

        jmp again_input_world           ; Guess next letter
```

Slika. 17. Kod za ispis poruke da je slovo već pogađano

U slučaju da slovo do sad nije bilo pogađano potrebno ga je dodati u niz *guessLetterArray*. To je realizovano pozivanjem funkcije *make_array_guess_letter*. Kod funkcije je prikazan na slici 18.

```
make_array_guess_letter PROC
    mov  edx, OFFSET guessLetterArray
    call StrLength              ; Length of a null-terminated string pointed to by EDX
    mov  lengthArray, eax

    mov edi, offset guessLetterArray ; Destination
    add edi, lengthArray
    mov al, guessLetter
    mov BYTE PTR [edi], al      ; Store guessLetter
    inc edi
    mov BYTE PTR [edi], ','     ; Store the null termination

    ret
make_array_guess_letter ENDP
```

Slika. 18. Kod za dodavanje unetog slova u niz unetih slova

Dalje je potrebno da proverimo da li se uneto slovo nalazi u izabranoj reči, čiji je izbor opisan u poglavlju iznad. Kod za pretraživanje stringa pogađane reči je sličan kao malo pre i prikazan je na slici 19.

```
;Check if letter is in selectedWords. If not take life
mov ecx, LENGTHOF selectedWords
mov edi, offset selectedWords
mov al, guessLetter                    ; Load character to find
repne scasb                            ; Search
jne loop_take_live                     ; Letter exist take life
```

Slika. 19. Provera da li izabrana reč sadrži uneto slovo

U slučaju da izabrana reč ne sadrži uneto slovo, skaćemo preko petlje loop_take_live, smanjujemo vrednost promenjivoj *statusGameLive* za jedan, tj oduzimamo jedan život igraču i vraćamo se na početak toga igre.

Ukoliko pogađana reč sadrži uneto slovo onda je potrebno da to slovo postavimo na odgovarajuće mesto u stringu *guessWords*, tj da odgovarajuće crtice zamenimo slovima na istim mestima kao što se nalaze i u pogađanoj reči *selectedWords*. Kod koji vrši ovu realizaciju je prikazan na slici 20.

```
; We are making new array, guess letter whange dash on right pleace
  mov esi, offset selectedWords       ; Source
  mov edi, offset guessWords          ; Destination
  mov ecx, LENGTHOF selectedWords     ; Number of bytes to check
  mov al, guessLetter                 ; Search for that character
  xor ebx, ebx                        ; Index EBX = 0

ride_hard_loop:
  cmp [esi+ebx], al                   ; Compare memory/register
  jne @F                              ; Skip next line if no match
  mov [edi+ebx], al                   ; Hang 'em lower
  @@:
  inc ebx                             ; Increment pointer
  dec ecx                             ; Decrement counter
  jne ride_hard_loop                  ; Jump if ECX != 0
```

Slika. 20. Zamena crtice sa unetim slovom

U sledećem koraku je potrebno da proverimo da li su ostale još neke crtice u stringu *guessWords* ili smo pogodili sva slova. Ako *guessWords* sadrži crtice vraćamo se na početak toka igre, u suprotnom ispisujemo animaciju i završavamo igricu.

```
;Is there more letter to guess of we finish
  mov ecx, LENGTHOF guessWords
  mov edi, offset guessWords
  mov al, letterDash                  ; Load character to find
  repne scasb                         ; Search
  jne loop_game_win                   ; No more letter
  jmp again_input_world               ; Guess next world
```

Slika. 21. Provera da li je ostalo još nepogođenih slova

V. Zaključak

. Ovaj projekat nas je podsetio i upoznao sa:

- Osnovnom sintaksom i funkcijama nižeg programskog jezika, asemblerom.
- Sa principom rada .386 procesora i sa korišćenjem direktno njegovih registara pri izvršenju raznih funkcija.
- Načinom komunikacije sa stekom i memorijom.
- Debagovanjem, testiranjem i nalaženjem rešenja za realne probleme.
- Bibliotekama od kojih je najvažnija *Irvine*.

Kao što vidimo kod se može napisati na više načina zbog raznovrsnosti funkcija i biblioteka koji su nam na raspolaganju. Mi smo se trudili da kod bude u duhu asemblera , tj. „što bliže procesoru“ što dovodi do lakšeg kompajliranja i bržeg izvršavanja od strane procesora. To nije mala stvar jer ponekad nije bitno samo da li radi , već i kako radi.

Napomena

Radi lakšeg testiranja projekta i usmene odbrane projekta na početku programa se ispisuje slučajan broj koji je generisan kao i izabrana reč koja se pogađa. Pri konačnoj verziji igre tih nekoliko linija koda je potrebno zakomentarisali ili obrisati, kako igrač ne bi varao.

Literatura

[1] Kip R. Irvine, Assembly Language for Intel-Based Computers, Fifth Edition, Prentice Hall Publishing, Inc. 2007

[2] Programmer's Guide - Microsoft® MASM Assembly-Language Development System Version 6.1, 1992

[3] Randall Hyde, The Art of Assembly Language, 2001

[4] Materijali sa vežbi iz predmeta Računarska elektronika, A. Lekić, Elektrotehnički fakultet, Univerziteta u Beogradu, 2016/2017

[5] http://programming.msjc.edu/asm/help/index.html?page=source%2Fabout.htm

[6] http://www.asmirvine.com/

[7] http://kipirvine.com/asm/#

[8] http://kipirvine.com/asm/index6th.htm

[9] https://stackoverflow.com

Realization game „Hangman“ in assembler

Stefan Tešanović, Predrag Mitrović

Kod igrice

```
INCLUDE Irvine32.inc
INCLUDE Macros.inc
INCLUDE VirtualKeys.inc

.data

endl EQU <0dh,0ah>  ; end of line sequence
message LABEL BYTE
    BYTE "                                                              " ,endl
    BYTE "                                                              " ,endl
    BYTE "     __  __ _______ __   _ _______ __  __ _______ __   _      " ,endl
    BYTE "    |  | | ||   _   ||  |  | ||       ||  |_|  ||   _   ||  |  | |      " ,endl
    BYTE "    |  |_| ||  |_|  ||   |_| ||    ___||       ||  |_|  ||   |_| |      " ,endl
    BYTE "    |       ||       ||       ||   | __ |       ||       ||       |      " ,endl
    BYTE "    |       ||       ||  _    ||   ||  ||       ||       ||  _    |      " ,endl
    BYTE "    |   _   ||   _   || | |   ||   |_| || ||_|| ||   _   || | |   |      " ,endl
    BYTE "    |__| |__||__| |__||_|  |__||_______||_|   |_||__| |__||_|  |__|      " ,endl
    BYTE "                                                              " ,endl
    BYTE "    Copyright (C) 2017  Stefan TESANOVIC 2016/675             " ,endl
    BYTE "                        Predrag MITROVIC 2016/678             " ,endl
    BYTE "                                                              " ,endl
messageSize DWORD ($-message)
consoleHandle HANDLE 0     ; handle to standard output device
bytesWritten  DWORD ?      ; number of bytes written

HANGMAN_GOODGAME_00 LABEL BYTE
                    BYTE "+-------+          ",endl
                    BYTE "|       |          ",endl
                    BYTE "|                  ",endl
                    BYTE "|       O          ",endl
                    BYTE "|      /|\         ",endl
                    BYTE "|      / \         ",endl
                    BYTE "+-------------+ ",endl
                    BYTE "| YOU   WIN   | ",endl
                    BYTE "+-------------+ ",endl
messageSizeGoodGame DWORD ($-HANGMAN_GOODGAME_00)

HANGMAN_GOODGAME_01 LABEL BYTE
                    BYTE "+-------+          ",endl
                    BYTE "|       |          ",endl
                    BYTE "|                  ",endl
                    BYTE "|       O_         ",endl
                    BYTE "|      /|          ",endl
                    BYTE "|      / \         ",endl
                    BYTE "+-------------+ ",endl
                    BYTE "| YOU   WIN   | ",endl
                    BYTE "+-------------+ ",endl

HANGMAN_GOODGAME_02 LABEL BYTE
                    BYTE "+-------+          ",endl
                    BYTE "|       |          ",endl
                    BYTE "|                  ",endl
                    BYTE "|       O/         ",endl
                    BYTE "|      /|          ",endl
                    BYTE "|      / \         ",endl
                    BYTE "+-------------+ ",endl
                    BYTE "| YOU   WIN   | ",endl
                    BYTE "+-------------+ ",endl
```

```
HANGMAN_GOODGAME_03 LABEL BYTE
                    BYTE "+------+        ",endl
                    BYTE "|       |       ",endl
                    BYTE "|               ",endl
                    BYTE "|       O_      ",endl
                    BYTE "|      /|       ",endl
                    BYTE "|      / \      ",endl
                    BYTE "+------------+ ",endl
                    BYTE "| YOU   WIN  | ",endl
                    BYTE "+------------+ ",endl

HANGMAN_GAMEOVER_00 LABEL BYTE
                    BYTE "+------+        ",endl
                    BYTE "|       |       ",endl
                    BYTE "|       O       ",endl
                    BYTE "|      /|\      ",endl
                    BYTE "|      / \      ",endl
                    BYTE "|               ",endl
                    BYTE "+------------+ ",endl
                    BYTE "|  YOU  DIE  | ",endl
                    BYTE "+------------+ ",endl

HANGMAN_GAMEOVER_01 LABEL BYTE
                    BYTE "+------+        ",endl
                    BYTE "|      /        ",endl
                    BYTE "|    _O         ",endl
                    BYTE "|   _/\         ",endl
                    BYTE "|    \          ",endl
                    BYTE "|               ",endl
                    BYTE "+------------+ ",endl
                    BYTE "|  YOU  DIE  | ",endl
                    BYTE "+------------+ ",endl

HANGMAN_GAMEOVER_02 LABEL BYTE
                    BYTE "+------+        ",endl
                    BYTE "|       |       ",endl
                    BYTE "|       O       ",endl
                    BYTE "|      /|\      ",endl
                    BYTE "|      / \      ",endl
                    BYTE "|               ",endl
                    BYTE "+------------+ ",endl
                    BYTE "|  YOU  DIE  | ",endl
                    BYTE "+------------+ ",endl

HANGMAN_GAMEOVER_03 LABEL BYTE
                    BYTE "+------+        ",endl
                    BYTE "|        \      ",endl
                    BYTE "|         O_    ",endl
                    BYTE "|        /\_    ",endl
                    BYTE "|         /     ",endl
                    BYTE "|               ",endl
                    BYTE "+------------+ ",endl
                    BYTE "|  YOU  DIE  | ",endl
                    BYTE "+------------+ ",endl

HANGMAN_LIVES_06 LABEL BYTE
                    BYTE "+------+        ",endl
                    BYTE "|       |       ",endl
                    BYTE "|               ",endl
                    BYTE "|               ",endl
                    BYTE "|               ",endl
                    BYTE "|               ",endl
                    BYTE "+------------+ ",endl
                    BYTE "|             | ",endl
                    BYTE "+------------+ ",endl

HANGMAN_LIVES_05 LABEL BYTE
                    BYTE "+------+        ",endl
                    BYTE "|       |       ",endl
                    BYTE "|       O       ",endl
                    BYTE "|               ",endl
                    BYTE "|               ",endl
                    BYTE "|               ",endl
                    BYTE "+------------+ ",endl
                    BYTE "|             | ",endl
                    BYTE "+------------+ ",endl

HANGMAN_LIVES_04 LABEL BYTE
                    BYTE "+------+        ",endl
                    BYTE "|       |       ",endl
                    BYTE "|       O       ",endl
                    BYTE "|       |       ",endl
                    BYTE "|               ",endl
                    BYTE "|               ",endl
                    BYTE "+------------+ ",endl
                    BYTE "|             | ",endl
                    BYTE "+------------+ ",endl

HANGMAN_LIVES_03 LABEL BYTE
                    BYTE "+------+        ",endl
                    BYTE "|       |       ",endl
                    BYTE "|       O       ",endl
                    BYTE "|      /|       ",endl
                    BYTE "|               ",endl
                    BYTE "|               ",endl
                    BYTE "+------------+ ",endl
                    BYTE "|             | ",endl
                    BYTE "+------------+ ",endl

HANGMAN_LIVES_02 LABEL BYTE
                    BYTE "+------+        ",endl
                    BYTE "|       |       ",endl
                    BYTE "|       O       ",endl
                    BYTE "|      /|\      ",endl
                    BYTE "|               ",endl
                    BYTE "|               ",endl
                    BYTE "+------------+ ",endl
                    BYTE "|             | ",endl
                    BYTE "+------------+ ",endl

HANGMAN_LIVES_01 LABEL BYTE
                    BYTE "+------+        ",endl
                    BYTE "|       |       ",endl
                    BYTE "|       O       ",endl
                    BYTE "|      /|\      ",endl
                    BYTE "|      /        ",endl
                    BYTE "|               ",endl
                    BYTE "+------------+ ",endl
                    BYTE "|             | ",endl
                    BYTE "+------------+ ",endl

HANGMAN_LIVES_00 LABEL BYTE
                    BYTE "+------+        ",endl
                    BYTE "|       |       ",endl
                    BYTE "|       O       ",endl
                    BYTE "|      /|\      ",endl
                    BYTE "|      / \      ",endl
                    BYTE "|               ",endl
                    BYTE "+------------+ ",endl
                    BYTE "|             | ",endl
                    BYTE "+------------+ ",endl
```

```
; random number what we generete
ranNum DWORD ?

;All words what is posible to guess.
;Pick by random generartor and put in selectedWords
manyWords   BYTE "BICYCLE", 0
            BYTE "CANOE", 0
            BYTE "SCATEBOARD", 0
            BYTE "OFFSIDE", 0
            BYTE "TENNIS", 0
            BYTE "SOFTBALL", 0
            BYTE "KNOCKOUT", 0
            BYTE "CHALLENGE", 0
            BYTE "SLALOM", 0
            BYTE "MARATHON", 0
            BYTE 0                  ; End of list
len equ $ - manyWords

; number what we make to know where are you in game
statusGameLive DWORD ?

;Wordls what we select by rundom code
selectedWords BYTE "                    ", 0
;Use as variable in funcstion for lenght of Array
lengthArray DWORD ?

;Letter what we guess, input from keyboard
guessLetter BYTE ?
;World what we print with -------,0
guessWords BYTE 50 DUP (?)
;Array of guess Letter
guessLetterArray BYTE 50 DUP (?)
;Letter what are unknows, change with -
letterDash BYTE '-'

drowDelay = 1000    ; delay 1 sec
var_loop BYTE 15    ; repeat 15 times

.code

main PROC

  ; Get the console output handle:
    INVOKE GetStdHandle, STD_OUTPUT_HANDLE
    mov consoleHandle,eax

  ; Write a string to the console:
    INVOKE WriteConsole,
        consoleHandle,              ;console output handle
        ADDR message,               ; string pointer
        messageSize,                ; string length
        ADDR bytesWritten,          ; returns num bytes written
        0                           ; not used

  ;Part of code for generate random number from 0 until 9
    mov  eax,10         ;get random 0 to 9
    call Randomize      ;re-seed generator
    call RandomRange
    mov  ranNum,eax     ;save random number

    call WriteDec
    call Crlf           ;new line

  ;Find a selectedWords base on generate ranNum from manyWords
    mov edx, ranNum     ;Index
    call find_str       ;Returns EDI = pointer to string, we pick world

  ;Copy find world in variable selectedWords
    INVOKE Str_copy,
        ADDR [edi],
        ADDR selectedWords

  ;Print selectedWords on screen
    mov edx, offset selectedWords
    call WriteString
    call Crlf           ;new line

  ;Make array of dash. It would be world what we guess
    call make_array_dash

  ;Inicialization number of life what you have
    mov statusGameLive, 6

again_input_world:

  ;Print figure depending on the number of lives
    call print_hangman_live

  ;Check if you have more live. If player lost all lives, game is over
    cmp statusGameLive, 0
    je loop_game_over


    mov  eax,green+(black*16)
    call SetTextColor

    mWrite <"Guess a letter: ">

    call readChar   ;User inputs char
    cmp al, 27      ;Check if is press ESC
    je exit_main    ;YES, end game
    and al, 0DFH    ;Convert lowercase input to uppercase.
                    ;If uppercase, it remains uppercase
    push eax
    sub al, 'A'     ;checks if it is a letter
    cmp al, 'Z'-'A'
    jbe uppercase
    jmp again_input_world
uppercase:
    pop eax
    mov guessLetter, al
    call WriteChar
    call Crlf       ;new line
    call Crlf       ;new line

    mov  eax,white+(black*16)
    call SetTextColor

   ;Check if letter is alredy guessed
   mov ecx, LENGTHOF guessLetterArray
   mov edi, offset guessLetterArray
   mov al, guessLetter                  ; Load character to find
   repne scasb                          ; Search
   je loop_guess_letter_exists          ; Letter already exist


   call make_array_guess_letter

   ;Check if letter is in selectedWords. If not take life
   mov ecx, LENGTHOF selectedWords
   mov edi, offset selectedWords
   mov al, guessLetter                  ; Load character to find
   repne scasb                          ; Search
   jne loop_take_live                   ; Letter exist take life


  ; We are making new array, guess letter whange dash on right pleace
   mov esi, offset selectedWords        ; Source
   mov edi, offset guessWords           ; Destination
   mov ecx, LENGTHOF selectedWords      ; Number of bytes to check
   mov al, guessLetter                  ; Search for that character
   xor ebx, ebx                         ; Index EBX = 0

ride_hard_loop:
   cmp [esi+ebx], al                    ; Compare memory/register
   jne @F                               ; Skip next line if no match
   mov [edi+ebx], al                    ; Hang 'em lower
   @@:
   inc ebx                              ; Increment pointer
   dec ecx                              ; Decrement counter
   jne ride_hard_loop                   ; Jump if ECX != 0


  ;Is there more letter to guess of we finish
   mov ecx, LENGTHOF guessWords
   mov edi, offset guessWords
   mov al, letterDash                   ; Load character to find
   repne scasb                          ; Search
   jne loop_game_win                    ; No more letter
   jmp again_input_world                ; Guess next world


exit_main:

   INVOKE ExitProcess,0

loop_guess_letter_exists:

       mov  eax,red+(black*16)
       call SetTextColor

       mWrite <"Sorry, you alredy guessed letter, ">
       mov al, guessLetter
       call WriteChar
       call Crlf                        ; new line
       mWrite <"I repeat you one more time the letter what you guessed. ">
       call Crlf                        ; new line
       mWrite <"Guessed letter are: ">
       mov edx, offset guessLetterArray
       call WriteString                 ; write a string pointed to by EDX
       call Crlf                        ; new line
       call Crlf                        ; new line

       mov  eax,white+(black*16)
       call SetTextColor

       jmp again_input_world            ; Guess next letter

loop_take_live:

       dec statusGameLive
       jmp again_input_world            ; Guess next letter
```

```
loop_game_win:

    mGotoxy 0, 15

  ; Write a string to the console:
    INVOKE WriteConsole,
        consoleHandle,              ;console output handle
        ADDR HANGMAN_GOODGAME_00,   ; string pointer
        messageSizeGoodGame,        ; string length
        ADDR bytesWritten,          ; returns num bytes written
        0                           ; not used

    mov eax, drowDelay
    call Delay
    mGotoxy 0, 15
    mov  eax,green+(black*16)
    call SetTextColor

  ; Write a string to the console:
    INVOKE WriteConsole,
        consoleHandle,              ;console output handle
        ADDR HANGMAN_GOODGAME_01,   ; string pointer
        messageSizeGoodGame,        ; string length
        ADDR bytesWritten,          ; returns num bytes written
        0                           ; not used

    mov eax, drowDelay
    call Delay
    mGotoxy 0, 15
    mov  eax,yellow+(black*16)
    call SetTextColor

  ; Write a string to the console:
    INVOKE WriteConsole,
        consoleHandle,              ;console output handle
        ADDR HANGMAN_GOODGAME_02,   ; string pointer
        messageSizeGoodGame,        ; string length
        ADDR bytesWritten,          ; returns num bytes written
        0                           ; not used

    mov eax, drowDelay
    call Delay
    mGotoxy 0, 15
    mov  eax,cyan+(black*16)
    call SetTextColor

  ; Write a string to the console:
    INVOKE WriteConsole,
        consoleHandle,              ;console output handle
        ADDR HANGMAN_GOODGAME_03,   ; string pointer
        messageSizeGoodGame,        ; string length
        ADDR bytesWritten,          ; returns num bytes written
        0                           ; not used

    mov eax, drowDelay
    call Delay
    mGotoxy 0, 15
    mov  eax,red+(black*16)
    call SetTextColor

    dec var_loop
    cmp var_loop, 0
    jne loop_game_win

    jmp exit_main


loop_game_over:

    mGotoxy 0, 15

  ; Write a string to the console:
    INVOKE WriteConsole,
        consoleHandle,              ;console output handle
        ADDR HANGMAN_GAMEOVER_00,   ; string pointer
        messageSizeGoodGame,        ; string length
        ADDR bytesWritten,          ; returns num bytes written
        0                           ; not used

    mov eax, drowDelay
    call Delay
    mGotoxy 0, 15
    mov  eax,green+(black*16)
    call SetTextColor

  ; Write a string to the console:
    INVOKE WriteConsole,
        consoleHandle,              ;console output handle
        ADDR HANGMAN_GAMEOVER_01,   ; string pointer
        messageSizeGoodGame,        ; string length
        ADDR bytesWritten,          ; returns num bytes written
        0                           ; not used

    mov eax, drowDelay
    call Delay
    mGotoxy 0, 15
    mov  eax,yellow+(black*16)
    call SetTextColor

  ; Write a string to the console:
    INVOKE WriteConsole,
        consoleHandle,              ;console output handle
        ADDR HANGMAN_GAMEOVER_02,   ; string pointer
        messageSizeGoodGame,        ; string length
        ADDR bytesWritten,          ; returns num bytes written
        0                           ; not used

    mov eax, drowDelay
    call Delay
    mGotoxy 0, 15
    mov  eax,cyan+(black*16)
    call SetTextColor

  ; Write a string to the console:
    INVOKE WriteConsole,
        consoleHandle,              ;console output handle
        ADDR HANGMAN_GAMEOVER_03,   ; string pointer
        messageSizeGoodGame,        ; string length
        ADDR bytesWritten,          ; returns num bytes written
        0                           ; not used

    mov eax, drowDelay
    call Delay
    mGotoxy 0, 15
    mov  eax,red+(black*16)
    call SetTextColor

    dec var_loop
    cmp var_loop, 0
    jne loop_game_over

    jmp exit_main

main ENDP

find_str PROC                   ; ARG: EDX = index
    lea edi, manyWords          ; Address of string list

    mov ecx, len                ; Maximal number of bytes to scan
    xor al, al                  ; Scan for 0

    @@:
    sub edx, 1
    jc done                     ; No index left to scan = string found
    repne scasb                 ; Scan for AL
    jmp @B                      ; Next string

  done:
    ret

find_str ENDP                   ; RESULT: EDI pointer to string[edx]

make_array_dash PROC
    mov  edx,OFFSET selectedWords
    call StrLength              ; Length of a null-terminated string pointed to by EDX
    mov  lengthArray,eax

    mov al, '-'                 ; Default charcter for guessWords
    mov ecx, lengthArray        ; REP counter
    mov edi, offset guessWords  ; Destination
    rep stosb                   ; Build guessWords
    mov BYTE PTR [edi], 0       ; Store the null termination

    ret
make_array_dash ENDP

make_array_guess_letter PROC
    mov  edx, OFFSET guessLetterArray
    call StrLength              ; Length of a null-terminated string pointed to by EDX
    mov  lengthArray, eax

    mov edi, offset guessLetterArray ; Destination
    add edi, lengthArray
    mov al, guessLetter
    mov BYTE PTR [edi], al      ; Store guessLetter
    inc edi
    mov BYTE PTR [edi], ','     ; Store the null termination

    ret
make_array_guess_letter ENDP
```

```
print_hangman_live PROC

    mov eax, statusGameLive

    cmp eax, 6
    je live_6
    cmp eax, 5
    je live_5
    cmp eax, 4
    je live_4
    cmp eax, 3
    je live_3
    cmp eax, 2
    je live_2
    cmp eax, 1
    je live_1
    cmp eax, 0
    je live_0

live_6:   ; Write a string to the console:
    INVOKE WriteConsole,
        consoleHandle,              ; console output handle
        ADDR HANGMAN_LIVES_06,      ; string pointer
        messageSizeGoodGame,        ; string length
        ADDR bytesWritten,          ; returns num bytes written
        0                           ; not used
    call Crlf                       ; new line
    call Crlf                       ; new line
    mov edx, offset guessWords
    call WriteString                ; write a string pointed to by EDX
    call Crlf                       ; new line
    call Crlf                       ; new line
    mWrite <"Guessed letter are: ">
    mov edx, offset guessLetterArray
    call WriteString                ; write a string pointed to by EDX
    call Crlf                       ; new line
    call Crlf                       ; new line
    ret

live_5:   ; Write a string to the console:
    INVOKE WriteConsole,
        consoleHandle,              ;console output handle
        ADDR HANGMAN_LIVES_05,      ; string pointer
        messageSizeGoodGame,        ; string length
        ADDR bytesWritten,          ; returns num bytes written
        0                           ; not used
    call Crlf                       ; new line
    call Crlf                       ; new line
    mov edx, offset guessWords
    call WriteString                ; write a string pointed to by EDX
    call Crlf                       ; new line
    call Crlf                       ; new line
    mWrite <"Guessed letter are: ">
    mov edx, offset guessLetterArray
    call WriteString                ; write a string pointed to by EDX
    call Crlf                       ; new line
    call Crlf                       ; new line
    ret

live_4:   ; Write a string to the console:
    INVOKE WriteConsole,
        consoleHandle,              ;console output handle
        ADDR HANGMAN_LIVES_04,      ; string pointer
        messageSizeGoodGame,        ; string length
        ADDR bytesWritten,          ; returns num bytes written
        0                           ; not used
    call Crlf                       ; new line
    call Crlf                       ; new line
    mov edx, offset guessWords
    call WriteString                ; write a string pointed to by EDX
    call Crlf                       ; new line
    call Crlf                       ; new line
    mWrite <"Guessed letter are: ">
    mov edx, offset guessLetterArray
    call WriteString                ; write a string pointed to by EDX
    call Crlf                       ; new line
    call Crlf                       ; new line
    ret

live_3:   ; Write a string to the console:
    INVOKE WriteConsole,
        consoleHandle,              ;console output handle
        ADDR HANGMAN_LIVES_03,      ; string pointer
        messageSizeGoodGame,        ; string length
        ADDR bytesWritten,          ; returns num bytes written
        0                           ; not used
    call Crlf                       ; new line
    call Crlf                       ; new line
    mov edx, offset guessWords
    call WriteString                ; write a string pointed to by EDX
    call Crlf                       ; new line
    call Crlf                       ; new line
    mWrite <"Guessed letter are: ">
    mov edx, offset guessLetterArray
    call WriteString                ; write a string pointed to by EDX
    call Crlf                       ; new line
    call Crlf                       ; new line
    ret

live_2:   ; Write a string to the console:
    INVOKE WriteConsole,
        consoleHandle,              ;console output handle
        ADDR HANGMAN_LIVES_02,      ; string pointer
        messageSizeGoodGame,        ; string length
        ADDR bytesWritten,          ; returns num bytes written
        0                           ; not used
    call Crlf                       ; new line
    call Crlf                       ; new line
    mov edx, offset guessWords
    call WriteString                ; write a string pointed to by EDX
    call Crlf                       ; new line
    call Crlf                       ; new line
    mWrite <"Guessed letter are: ">
    mov edx, offset guessLetterArray
    call WriteString                ; write a string pointed to by EDX
    call Crlf                       ; new line
    call Crlf                       ; new line
    ret
```

```
live_1:   ; Write a string to the console:
    INVOKE WriteConsole,
        consoleHandle,              ;console output handle
        ADDR HANGMAN_LIVES_01,      ; string pointer
        messageSizeGoodGame,        ; string length
        ADDR bytesWritten,          ; returns num bytes written
        0                           ; not used
    call Crlf                       ; new line
    call Crlf                       ; new line
    mov edx, offset guessWords
    call WriteString                ; write a string pointed to by EDX
    call Crlf                       ; new line
    call Crlf                       ; new line
    mWrite <"Guessed letter are: ">
    mov edx, offset guessLetterArray
    call WriteString                ; write a string pointed to by EDX
    call Crlf                       ; new line
    call Crlf                       ; new line
    ret

live_0:   ; Write a string to the console:
    INVOKE WriteConsole,
        consoleHandle,              ;console output handle
        ADDR HANGMAN_LIVES_00,      ; string pointer
        messageSizeGoodGame,        ; string length
        ADDR bytesWritten,          ; returns num bytes written
        0                           ; not used
    call Crlf                       ; new line
    call Crlf                       ; new line
    mov edx, offset guessWords
    call WriteString                ; write a string pointed to by EDX
    call Crlf                       ; new line
    call Crlf                       ; new line
    mWrite <"Guessed letter are: ">
    mov edx, offset guessLetterArray
    call WriteString                ; write a string pointed to by EDX
    call Crlf                       ; new line
    call Crlf                       ; new line
    ret

print_hangman_live ENDP

END main
```