Instytut Informatyki Politechniki Śląskiej
Zespół Mikroinformatyki i Teorii Automatów Cyfrowych

Laboratorium JA

Rok akademicki	Rodzaj studiów*: SSI/NSI/NSM	Numer ćwiczenia:	Grupa	Sekcja
2017/2018	**SSI**	**4**	**6**	**1**

Data i godzina planowana ćwiczenia: dd/mm/rrrr - gg:mm	**20/03/2018-11:45**	**Prowadzący**: OA/AO	**OA**
Data i godzina wykonania ćwiczenia: dd/mm/rrrr - gg:mm	**20/03/2018-11:45**		

Sprawozdanie

Temat ćwiczenia:

Zapoznanie się z programowaniem w j. asemblera MASM 32 w środowisku Windows.

Skład sekcji:	1.Bartłomiej Krasoń

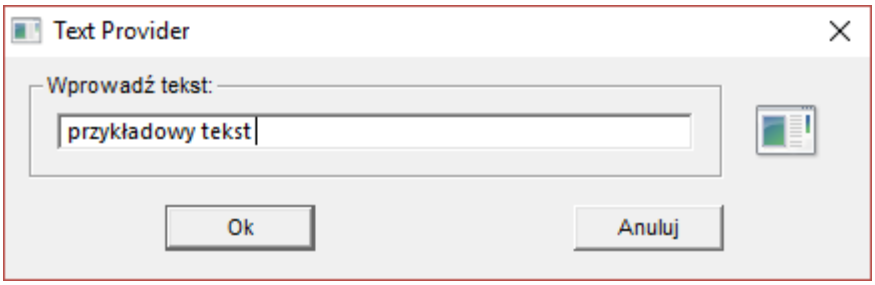
Cel

Celem ćwiczenia jest poznanie innych niż Microsoft VC asemblerów i programów debugerów procesorów x86 dla środowiska Windows. Za pomocą narzędzia QEDITOR napisać własny program asemblerowy wykorzystujący podstawowe elementy GUI.
~Źródło – Instrukcja ćwiczenia LAB4

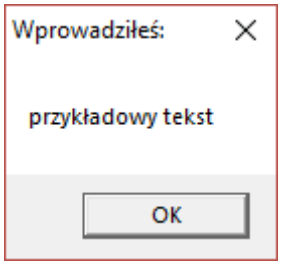
Rozwiązanie

Wykorzystując narzędzie jakim jest QEDITOR dostarczony wraz z pakietem masm32 na mój komputer oraz z pomocą przykładowych programów realizujących podstawowy interfejs GUI w aplikacji, przy użyciu języka asemblera utworzyłem własny, dość podstawowy program „TextProvider", który pobiera od użytkownika tekst za pomocą kontrolki **Edit** następnie po wciśnięciu ENTER lub kontrolki **Button** „Ok" wyświetla wprowadzony tekst w nowym oknie dialogowym **MessageBox** i kończy pracę programu. Kontrolka **Button** „Anuluj" powoduje zamknięcie aplikacji, po uprzednim powiadomieniu użytkownika. Poniżej prezentuję wynik mojej pracy w postaci:

a.) Głównego okna programu:



b.) Okna po wprowadzeniu i zatwierdzeniu tekstu:



c.) Ciekawszych fragmentów kodu programu:

- funkcja inicjalizującą główne okno programu:

```
GetTextDialog proc dgltxt:DWORD,grptxt:DWORD,iconID:DWORD

    LOCAL arg1[4]:DWORD
    LOCAL parg  :DWORD

    lea eax, arg1
    mov parg, eax

    mov ecx, dgltxt
    mov [eax], ecx
    mov ecx, grptxt
    mov [eax+4], ecx
    mov ecx, iconID
    mov [eax+8], ecx

    Dialog "Get User Text", \               ; caption
           "Arial",8, \                     ; font,pointsize
            WS_OVERLAPPED or \              ; styles for
            WS_SYSMENU or DS_CENTER, \      ; dialog window
            5, \                            ; number of controls
            50,50,292,80, \                 ; x y co-ordinates
            4096                            ; memory buffer size

    DlgIcon   0,250,12,299
    DlgGroup  0,8,4,231,31,300
    DlgEdit   ES_LEFT or WS_BORDER or WS_TABSTOP,17,16,212,11,301
    DlgButton "Ok",WS_TABSTOP,53,42,50,13,IDOK
    DlgButton "Anuluj",WS_TABSTOP,189,42,50,13,IDCANCEL

    CallModalDialog hInstance,0,dlgproc,parg

    ret

GetTextDialog endp
```

- główna instrukcja switch/case obsługująca „eventy” programu:

```
switch uMsg
  case WM_INITDIALOG
    push esi
    mov esi, lParam
    fn SetWindowText,hWin,[esi]                          ; title text address
    fn SetWindowText,rv(GetDlgItem,hWin,300),[esi+4]     ; groupbox text addres
    mov eax, [esi+8]                                     ; icon handle
    .if eax == 0
      mov hIcon, rv(LoadIcon,NULL,IDI_WINLOGO)          ; use default system ic
    .else
      mov hIcon, eax                                     ; load user icon
    .endif
    pop esi

    fn SendMessage,hWin,WM_SETICON,1,hIcon
    invoke SendMessage,rv(GetDlgItem,hWin,299),STM_SETIMAGE,IMAGE_ICON,hIcon
    xor eax, eax
    ret

  case WM_COMMAND
    switch wParam
      case IDOK
        mov tlen, rv(GetWindowTextLength,rv(GetDlgItem,hWin,301))
        .if tlen == 0
          invoke SetFocus,rv(GetDlgItem,hWin,301)
          ret
        .endif
        add tlen, 1
        mov hMem, alloc(tlen)
        fn GetWindowText,rv(GetDlgItem,hWin,301),hMem,tlen
        invoke EndDialog,hWin,hMem
      case IDCANCEL
        invoke EndDialog,hWin,0
    endsw
  case WM_CLOSE
    invoke EndDialog,hWin,0
endsw
```

Wnioski

Wykonanie tego ćwiczenia uświadomiło mi, że przy wykorzystaniu odpowiedniej biblioteki, jak i użyciu odpowiednich makr, w języku asemblera można utworzyć aplikację z interfejsem graficznym. Jednakże jest to nieco czasochłonne przedsięwzięcie ze względu na niskopoziomowy charakter asemblera. Do tworzenia małych, nieskomplikowanych pod względem graficznego interfejsu użytkownika aplikacji, rozwiązanie to może się okazać bardzo trafne, gdyż zyskujemy tutaj na optymalizacji czasowej jak i pamięciowej w porównaniu do wyżej poziomowych języków programowania. Natomiast więcej czasu, jak i wysiłku musimy poświęcić na utworzenie takiej aplikacji.