|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| logowydzialu | | Instytut Informatyki Politechniki Śląskiej  Zespół Mikroinformatyki i Teorii Automatów Cyfrowych  **Laboratorium JA** | | logoii | |
| **Rok akademicki** | | **Rodzaj studiów\*: SSI/NSI/NSM** | **Numer ćwiczenia:** | **Grupa** | **Sekcja** |
| **2017/2018** | | **SSI** | **1** | **6** | **1** |
| **Data i godzina planowana ćwiczenia:**  dd/mm/rrrr - gg:mm | | **06/03/2018-11:45** | **Prowadzący**:  OA/AO | **OA** | |
| **Data i godzina wykonania ćwiczenia:**  dd/mm/rrrr - gg:mm | | **06/03/2018-11:45** |
| ***Sprawozdanie*** | | | | | |
| **Temat ćwiczenia:**  Tryb adresacji procesorów x86 | | | | | |
| **Skład sekcji:** | 1.Bartłomiej Krasoń | | | | |

# Cel

Celem ćwiczenia jest poznanie trybów adresacji procesora x86. Konstrukcja projektu zakłada możliwość wywoływania funkcji bibliotecznych napisanych w asemblerze z poziomu aplikacji oraz pokazuje prawidłową konfigurację środowiska umożliwiającą debugowanie kodu do poziomu asemblera, obserwację stanu rejestrów i flag procesora czy obszarów pamięci danych.

*~Źródło – Instrukcja ćwiczenia LAB2*

Dodatkowo: Przetestować jak szybko wykonują się zadane procedury wyszukiwania danego znaku w łańcuchu znaków oraz je porównać. Napisać swoją jeszcze bardziej optymalną wersję takiej procedury.

# Rozwiązanie

Zadaniem procedury jest znalezienie znaku ‘J’ w ciągu znaków ’AGIJKSZ’. Nasza podsekcja miała do przetestowania procedurę ”FindChar\_2”, której kod jest następujący:

.486

.model flat, stdcall

.data

DataString DB 'AGIJKSZ', 0FFH ; definicja ciagu znakow

.code

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Procedura FindChar\_2 wyszukiwania znaku 'J' w ciagu 'LocalString' \*

;\* \*

;\* Bezposrednia adresacja indeksowa \*

;\* Parametry wejsciowe: \*

;\* AH - szukany znak 'J' \*

;\* Parametry wyjsciowe: \*

;\* EAX - BOOL TRUE Found, FALSE not found \*

;\* \*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FindChar\_2 PROC

LocalString DB 0C3H,'AGIJKSZ', 0FFH ; definicja ciagu znakow

MOV ESI, OFFSET LocalString ; zaladuj offset zmiennej 'LocalString' do rej. ESI

MOV AH, 'J' ; zaladuj kod litery 'J' do rej. AH

Check\_End:

CMP BYTE PTR [ESI], 0FFH ; czy koniec lancucha (znak specjalny FF)?

JE Not\_Find ; znaleziono znak konca (wartownik)

CMP AH, [ESI] ; porownaj znak z elementem lancucha 'LocalString'

JE Got\_Equal ; znaleziono znak!

ADD ESI, 1 ; inkrementuj offset

JMP Check\_End ; petla wyszukiwania

Got\_Equal:

MOV DL, [ESI] ; zaladuj znaleziony znak do DL

JMP Done

Not\_Find:

MOV EAX,0 ; nie znaleziono znaku

RET ; powrot z procedury

Done:

MOV EAX,1 ; znaleziono znak

RET ; powrot z procedury

FindChar\_2 ENDP ; koniec FindChar\_2

END

**Na podstawie powyższego pliku wygenerowaliśmy następujący listing:**

Microsoft (R) Macro Assembler Version 6.14.8444 03/10/18 18:32:11

plik.asm Page 1 - 1

.486

.model flat, stdcall

00000000 .data

00000000 41 47 49 4A 4B 53 5A DataString DB 'AGIJKSZ', 0FFH ; definicja ciagu znakow

FF

00000000 .code

00000000 FindChar\_2 PROC

00000000 C3 41 47 49 4A 4B 53 LocalString DB 0C3H,'AGIJKSZ', 0FFH

5A FF

00000009 1 BE 00000000 R MOV ESI, OFFSET LocalString

0000000E 1 B4 4A MOV AH, 'J'

00000010 Check\_End:

00000010 2 80 3E FF CMP BYTE PTR [ESI], 0FFH

00000013 3,1 74 0D JE Not\_Find

00000015 2 3A 26 CMP AH, [ESI]

00000017 3,1 74 05 JE Got\_Equal

00000019 1 83 C6 01 ADD ESI, 1

0000001C 3 EB F2 JMP Check\_End

0000001E Got\_Equal:

0000001E 1 8A 16 MOV DL, [ESI]

00000020 3 EB 06 JMP Done

00000022 Not\_Find:

00000022 1 B8 00000000 MOV EAX,0

00000027 5 C3 RET

00000028 Done:

00000028 1 B8 00000001 MOV EAX,1

0000002D 5 C3 RET

0000002E FindChar\_2 ENDP

END

Microsoft (R) Macro Assembler Version 6.14.8444 03/10/18 18:32:11

plik.asm Symbols 2 - 1

Segments and Groups:

N a m e Size Length Align Combine Class

FLAT . . . . . . . . . . . . . . . . . GROUP

\_DATA . . . . . . . . . . . . . . . . 32 Bit 00000008 Para Public 'DATA'

\_TEXT . . . . . . . . . . . . . . . . 32 Bit 0000002E Para Public 'CODE'

Procedures, parameters and locals:

N a m e Type Value Attr

FindChar\_2 . . . . . . . . . . . . . . P Near 00000000 \_TEXT Length= 0000002E Public STDCALL

Check\_End . . . . . . . . . . . . . L Near 00000010 \_TEXT

Got\_Equal . . . . . . . . . . . . . L Near 0000001E \_TEXT

Not\_Find . . . . . . . . . . . . . . L Near 00000022 \_TEXT

Done . . . . . . . . . . . . . . . . L Near 00000028 \_TEXT

Symbols:

N a m e Type Value Attr

@CodeSize . . . . . . . . . . . . . . Number 00000000h

@DataSize . . . . . . . . . . . . . . Number 00000000h

@Interface . . . . . . . . . . . . . . Number 00000003h

@Model . . . . . . . . . . . . . . . . Number 00000007h

@code . . . . . . . . . . . . . . . . Text \_TEXT

@data . . . . . . . . . . . . . . . . Text FLAT

@fardata? . . . . . . . . . . . . . . Text FLAT

@fardata . . . . . . . . . . . . . . . Text FLAT

@stack . . . . . . . . . . . . . . . . Text FLAT

DataString . . . . . . . . . . . . . . Byte 00000000 \_DATA

LocalString . . . . . . . . . . . . . Byte 00000000 \_TEXT

0 Warnings

0 Errors

Na podstawie powyższego listingu, obliczyliśmy, że wykonanie się procedury ”FindChar\_2” zabiera 60 taktów zegara (wliczając rozkaz RET). Następnie nasz wynik, porównaliśmy z wynikami procedur badanych przez inne podsekcje:

|  |  |
| --- | --- |
| **Procedura** | **Liczba taktów zegarowych** |
| FindChar\_1 | 50 |
| FindChar\_2 | 60 |
| FindChar\_3 | 50 |
| FindChar\_4 | 50 |
| FindChar\_5 | 21 |
| FindChar\_6 | 51 |
| My\_Procedure | 34 |

Następnie, kierując się uzyskanymi wynikami napisałem własną procedurę wyszukiwania danego znaku w łańcuchu znaków. Po analizie danych z tabeli i kodu poszczególnych procedur doszedłem do wniosku, że warto jest ograniczyć wykonywanie obrotów pętli (kosztowne dalekie skoki). W swojej procedurze dokonałem pewnego kompromisu, gdyż wzorując się na procedurze ”FindChar\_5” dokonuję większego kroku w jednym obrocie pętli (sprawdzając kilka elementów na raz) jednocześnie, pozostawiając moją procedurę uniwersalną dla różnej długości łańcuchów znaków. Praktyka ta jak wynika z nabytej przez mnie ostatnio wiedzy, jest często stosowana w świecie programistycznym w celu optymalizacji kodu i nazywana jest rozwijaniem pętli. Myślę że jest to odpowiednie rozwiązanie, gdyż zachowuję użyteczność tej procedury na większą skalę oraz gwarantuje jej również lepszą niż przeciętna szybkość procedur pochodzących z instrukcji. Oto kod mojej procedury:

.486

.model flat, stdcall

.data

DataString DB 'AGIJKSZ', 0FFH ; definicja ciagu znakow

.code

MOV EBX, OFFSET DataString ; zaladuj offset zmiennej 'DataString' do rej. EBX

My\_Procedure PROC

Check\_End:

CMP BYTE PTR [EBX], 0FFH ; czy koniec łańcucha?

JE Not\_find ; znaleziono znak konca

CMP BYTE PTR [EBX], 'J' ; porownaj szukany znak z elementem łańcucha

JE Got\_Equal ; znaleziono znak!

;powtórz to samo dla 3 kolejnych elementów w celu optymalizacji

CMP BYTE PTR [EBX+1], 0FFH

JE Not\_find

CMP BYTE PTR [EBX+1], 'J'

JE Got\_Equal

CMP BYTE PTR [EBX+2], 0FFH

JE Not\_find

CMP BYTE PTR [EBX+2], 'J'

JE Got\_Equal

CMP BYTE PTR [EBX+3], 0FFH

JE Not\_find

CMP BYTE PTR [EBX+3], 'J'

JE Got\_Equal

ADD EBX, 4 ; zwiększ offset o odpowiedni krok

JMP Check\_End ; petla wyszukiwania

Got\_Equal:

MOV DL, 'J' ; załaduj znaleziony znak do DL

MOV EAX,1 ; znaleziono znak

RET ; powrot z procedury

Not\_Find:

MOV EAX,0 ; nie znaleziono znaku

RET ; powrot z procedury

My\_Procedure ENDP ;

END

**Listing mojej procedury:**

Microsoft (R) Macro Assembler Version 6.14.8444 03/13/18 00:03:32

My\_Procedure.asm Page 1 - 1

.486

.model flat, stdcall

00000000 .data

00000000 41 47 49 4A 4B 53 5A DataString DB 'AGIJKSZ', 0FFH ; definicja ciagu znakow

FF

00000000 .code

00000000 FindChar\_2 PROC

00000000 1 BB 00000000 R MOV EBX, OFFSET DataString ; zaladuj offset zmiennej 'DataString' do rej. EBX

00000005 Check\_End:

00000005 2 80 3B FF CMP BYTE PTR [EBX], 0FFH ; czy koniec łańcucha?

00000008 3,1 74 36 JE Not\_find ; znaleziono znak konca

0000000A 2 80 3B 4A CMP BYTE PTR [EBX], 'J' ; porownaj szukany znak z elementem łańcucha

0000000D 3,1 74 29 JE Got\_Equal ; znaleziono znak!

;powtórz to samo dla 3 kolejnych elementów w celu optymalizacji

0000000F 2 80 7B 01 FF CMP BYTE PTR [EBX+1], 0FFH

00000013 3,1 74 2B JE Not\_find

00000015 2 80 7B 01 4A CMP BYTE PTR [EBX+1], 'J'

00000019 3,1 74 1D JE Got\_Equal

0000001B 2 80 7B 02 FF CMP BYTE PTR [EBX+2], 0FFH

0000001F 3,1 74 1F JE Not\_find

00000021 2 80 7B 02 4A CMP BYTE PTR [EBX+2], 'J'

00000025 3,1 74 11 JE Got\_Equal

00000027 2 80 7B 03 FF CMP BYTE PTR [EBX+3], 0FFH

0000002B 3,1 74 13 JE Not\_find

0000002D 2 80 7B 03 4A CMP BYTE PTR [EBX+3], 'J'

00000031 3,1 74 05 JE Got\_Equal

00000033 1 83 C3 04 ADD EBX, 4 ; zwiększ offset o odpowiedni krok

00000036 3 EB CD JMP Check\_End ; petla wyszukiwania

00000038 Got\_Equal:

00000038 1 B2 4A MOV DL, 'J' ; załaduj znaleziony znak do DL

0000003A 1 B8 00000001 MOV EAX,1 ; znaleziono znak

0000003F 5 C3 RET ; powrot z procedury

00000040 Not\_Find:

00000040 1 B8 00000000 MOV EAX,0 ; nie znaleziono znaku

00000045 5 C3 RET ; powrot z procedury

00000046 FindChar\_2 ENDP ;

END

Microsoft (R) Macro Assembler Version 6.14.8444 03/13/18 00:03:32

My\_Procedure.asm Symbols 2 - 1

Segments and Groups:

N a m e Size Length Align Combine Class

FLAT . . . . . . . . . . . . . . . . . GROUP

\_DATA . . . . . . . . . . . . . . . . 32 Bit 00000008 Para Public 'DATA'

\_TEXT . . . . . . . . . . . . . . . . 32 Bit 00000046 Para Public 'CODE'

Procedures, parameters and locals:

N a m e Type Value Attr

FindChar\_2 . . . . . . . . . . . . . . P Near 00000000 \_TEXT Length= 00000046 Public STDCALL

Check\_End . . . . . . . . . . . . . L Near 00000005 \_TEXT

Got\_Equal . . . . . . . . . . . . . L Near 00000038 \_TEXT

Not\_Find . . . . . . . . . . . . . . L Near 00000040 \_TEXT

Symbols:

N a m e Type Value Attr

@CodeSize . . . . . . . . . . . . . . Number 00000000h

@DataSize . . . . . . . . . . . . . . Number 00000000h

@Interface . . . . . . . . . . . . . . Number 00000003h

@Model . . . . . . . . . . . . . . . . Number 00000007h

@code . . . . . . . . . . . . . . . . Text \_TEXT

@data . . . . . . . . . . . . . . . . Text FLAT

@fardata? . . . . . . . . . . . . . . Text FLAT

@fardata . . . . . . . . . . . . . . . Text FLAT

@stack . . . . . . . . . . . . . . . . Text FLAT

DataString . . . . . . . . . . . . . . Byte 00000000 \_DATA

0 Warnings

0 Errors

# Wnioski

Wykonując to zadanie po pierwsze nauczyłem się, że jeden problem w programowaniu można rozwiązać na wiele różnych sposób. Co więcej te sposoby mogą się między sobą znacznie różnić, między innymi pod względem optymalizacyjnym. Warto jednak zauważyć, że nie zawsze warto brnąc po jak najszybciej działający program, gdyż traci on wtedy swoje inne walory, np. uniwersalność czy szerszą użyteczność. Jako programiści musimy podejmować odpowiednie kompromisy i wybierać takie rozwiązania, które będą najodpowiedniejsze dla konkretnej sytuacji. Ja postawiłem na rozwiązanie bardziej uniwersalne choć mam świadomość tego, że z łatwością można stworzyć znacznie szybszą procedurę lecz ograniczając się jedynie dla tej konkretnej sytuacji. Praca na tym zadaniem nauczyła mnie również jak stworzyć listing, dla programu który tworzymy. Uważam że jest to przydatna wiedza, która umożliwia tworzącemu oprogramowanie zauważenie elementów, które może zoptymalizować w swoim programie.