Volání funkcí a předávání parametrů funkcím registry, knihovna pro vstupní a výstupní operace

Ing. Dominika Regéciová
Výzkumná skupina formálních modelů

Vysoké učení technické v Brně, Fakulta informačních technologií
Božetěchova 1/2, 612 66 Brno - Královo Pole

iregeciova@fit.vutbr.cz

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Řízení toku programu

- Zdrojový kód programu je uložen v kódovém segmentu paměti.
- Každá instrukce má svoji adresu, aktivní instrukci určuje registr **EIP**
- Hodnota EIP se změní provedením libovolné instrukce:
  - Normální instrukce - přesun na další instrukci v pořadí (další řádek kódu)
  - Skok - přesun na návěští
  - Volání funkce (**call**) - na vrchol zásobníku se uloží adresa dalšího řádku, a přesune se na návěští
  - Návrat z funkce (**ret**) - přesun na adresu z vrcholu zásobníku

Zásobník

- Adresa vrcholu zásobníku je určena registrem **ESP**
- Zásobník "roste směrem dolů", tedy od vyšší adresy směrem k nižší
- Zásobník se vždy posouvá o násobky čtyř ve 32bitovém režimu, tedy o 4 byty = 32 bitů
- Velikost zásobníku je omezena HW počítače
- Příliš mnoho rekurzivních volání způsobí přetečení

Registry

- **SS** (stack segment) - definuje začátek segmentu zásobníku (ten nemodifikujte)
- **ESP** (stack pointer) - definuje ukazatel vrcholu zásobníku
- **EBP** (base pointer) – obecný registr pro přístup k zásobníku, typicky se používá pro definici rámce zásobníku (stack frame) při volání pod-programů

Instrukce pro práci se zásobníkem

- Instrukce **PUSH *val***
  - Dekrementuje ESP (2/4B)
  - Poté uloží hodnotu *val* na vrchol zásobníku
  - Možné operandy: paměť 16/32-bitů, obecný registr 16/32-bitů, konstanta 8/16/32-bitů, segmentové registry
- Instrukce **POP *reg***
  - Inkrementuje ESP (2/4B)
  - Hodnotu na vrcholu zásobníku uloží do registru *reg*
- Instrukce **PUSHA**
  - 16b registry se v tomto pořadí uloží na zásobník: AX, CX, DX, BX, SP, BP, SI, DI
- Instrukce **PUSHAD**
  - 32b registry se v tomto pořadí uloží na zásobník: EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI
- Instrukce **POPA**, **POPAD**
- Instrukce **PUSHF (PUSHFD)/ POPF (POPFD)**
  - Uloží/obnoví příznakový registr (E)FLAGS na/ze zásobníku

Instrukce pro práci se zásobníkem II

```
; vlozeni konstatny, pocatek ESP = 1000

push dword 1234h   ; ESP = 996
; prima pamet
push dword (ebx)   ; ESP = 992
; registry
push ax            ; ESP = 990 (-2!)
push byte 1        ; ESP = 986 (-4!)


; pocatek ESP = 984
pop ebx            ; ESP = 988
pop eax            ; ESP = 992
pop dword (ebx)    ; EPS = 996
pop edx            ; ESP = 1000
```

Podprogram

- (Často využívaná) skupina instrukcí. Volá se jako samostatná část programu
- Instrukce **CALL *dest***
  - Na zásobník uloží návratovou adresu (adresu následující instrukce: registr EIP v případě blízkého volání, registry CS a EIP v případě vzdáleného volání).
  - Přejde na návěští *dest*
  - Instrukce **neukládá** parametry na zásobník
- Instrukce **RET *(int)***
  - Zabezpečuje návrat z podprogramu
  - Do EIP uloží návratovou hodnotu z vrcholu zásobníku
  - Volitelný parametr *(int)* - odstraní ze zásobníku int hodnot velikosti byte (úklid + správné nastavení ESP)

Předávání parametrů přes zásobník

- Pokud předáváme parametry přes zásobník (v jazyce C), v podprogramu se vytváří *stack frame* (rámec zásobníku)
- Na zásobníku si uchováváme:
  - Návratovou adresu z podprogramu
  - parametry podprogramu
  - lokální proměnné podprogramu
- S jednotlivými parametry a lokálními proměnnými pracujeme přes registr EBP
- Musíme uklidit zásobník a obnovit ESP

Předávání parametrů přes zásobník II

```
#include <stdio.h>

int Soucet(int a, int b) {
  return a + b;
}

int main() {
  printf("%d\n", Soucet(128,2));
  return 0;
}
```

- Chceme získat součet dvou čísel, dva parametry na zásobníku, výsledek v registru EAX

Předávání parametrů přes zásobník III

```
Soucet:
  push ebp            ; stack-frame enter (ulozeni EBP)
  mov  ebp,esp        ; stack-frame enter (EBP ukazuje na vrchol zasobniku)

  mov  edx,[ebp+8]    ; EDX = a = 128 -- prvni argument
  mov  ecx,[ebp+12]   ; ECX = b = 2   -- druhy argument
  add  edx,ecx        ; EDX = a + b
  mov  eax,edx        ; return a + b (EAX <- EDX)

  mov  esp,ebp        ; stack-frame leave (obnoveni ESP)
  pop  ebp            ; stack-frame leave (obnoveni EBP)
  ret  8              ; stdcall odstraneni parametru ze zasobniku (2 x 32-bitu == 8 byte)

main:

  push dword 2        ; b = 2
  push dword 128      ; a = 128
  call Soucet         ; Soucet(128,2)
  call WriteInt32
  call WriteNewLine
ret
```

Předávání parametrů přes zásobník IV

```
            |      ...       | offset relativne k EBP (stack frame pointer)
            |________________|
            |                |
            |    b = 2       |
            |                |
            |________________| +12
            |                |
            |    a = 128     |
            |                |
            |________________| +8
            |                |
            |     EIP        |
            |                |
            |________________| +4
            |                |
            |     EBP        |
            |                |
ESP/EBP ->  |________________|  0
            |                |
            |   zasobnik     |
            | pod-programu   |
            |      ...       |
            |________________|
            |                |
```

Lokální proměnné podprogramu

```
int Soucet(int a, int b) {
  int c;
  c = a + b;
  return c;
}
```

```
Soucet:
  push ebp            ; stack-frame enter (ulozeni EBP)
  mov  ebp,esp        ; stack-frame enter (EBP ukazuje na vrchol zasobniku)
  sub  esp,4          ; stack-frame enter (ESP preskoci az za lokalni promenne)

  mov  edx,[ebp+8]    ; EDX = a = 128 -- prvni argument
  mov  ecx,[ebp+12]   ; ECX = b = 2   -- druhy argument
  add  edx,ecx        ; EDX = a + b
  mov  [ebp-4],edx    ; c = EDX
  mov  eax,[ebp-4]    ; return c

  mov  esp,ebp       ; stack-frame leave (obnoveni ESP)
  pop  ebp           ; stack-frame leave (obnoveni EBP)
  ret  8             ; stdcall odstraneni parametru ze zasobniku (2 x 32-bitu == 8 byte)
          |     ...      | offset relativne k EBP (stack frame pointer)
```

Lokální proměnné podprogramu II

```
        |_______________|
        |               |
        |    b = 2      |
        |               |
        |_______________| +12
        |               |
        |    a = 128    |
        |               |
        |_______________| +8
        |               |
        |     EIP       |
        |               |
        |_______________| +4
        |               |
        |     EBP       |
        |               |
EBP ->  |_______________|  0
        |               |
        |      c        |
        |   (a + b)     |
ESP ->  |_______________| -4
        |               |
        |   zasobnik    |
        | pod-programu  |
        |     ...       |
        |_______________|
        |               |
```

STACK
FRAME

Doplnění

- **ENTER** a **LEAVE**

```
; enter
push ebp
mov ebp, esp
; sub esp, N ; enter N, 0

; leave
mov esp, ebp
pop ebp
```

- **REP** ***instrukce***
  - Opakuje instrukci tolikrát, kolik je uvedeno v registru ECX
  - `ecx-=1, if exc != 0 then opakuj`
  - RPENE, REPNZ, REPZ, REPE

Řetězové instrukce

- MOVS, CMPS, SCAS, LODS, STOS, INS, OUTS
- Ukazatele na data v registrech DS:ESI a ES:EDI
- Zdrojová data jsou uložena na adrese DS:ESI ("SI-Source Index)
- Cílová data jsou uložena na adrese ES:EDI ("DI-Destination Index)
- Instrukce pracují s:
  - Slabikami (LODSB, SCASB, . . .)
  - Slovy (LODSW, MOVSW, . . .)
  - Dvojslovy (LODSD, MOVSD, . . .)
- Instrukce automaticky zvyšuje/snižuje (podle DF) indexové registry ESI a EDI
  - Nastavení DF: instrukce CLD, STD
- CMPS, SCAS - ovlivňují příznaky v EFLAGS

Čtení z klávesnice

- ReadChar || ReadString
- ReadInt8 || ReadUInt8
- ReadInt16 || ReadUInt16
- ReadInt32 || ReadUInt32
- ReadFloat || ReadDouble

Výpis na obrazovku

- WriteChar || WriteNewLine || WriteString
- WriteBin8 || WriteHex8 || WriteInt8 || WriteUInt8
- WriteBin16 || WriteHex16 || WriteInt16 || WriteUInt16
- WriteBin32 || WriteHex32 || WriteInt32 || WriteUInt32
- WriteFlags
- WriteFloat || WriteDouble