Práce s registry a s pamětí

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

Příklad na začátek

- **Převeďte číslo $(1FD)_{16}$ do binární soustavy**
- **Převeďte číslo $(00111000)_2$ do osmičkové soustavy**

Příklad na začátek

- **Převeďte číslo $(1FD)_{16}$ do binární soustavy**
  - $(0001\ 1111\ 1101)_2$
- **Převeďte číslo $(0011\ 1000)_2$ do osmičkové soustavy**
  - $(070)_8$

Opakování: Registry

- **Datové**
  - *EAX* - aritmetické operace, vstup/výstup
  - *EBX* - adresování
  - *ECX* - čítač v cyklech
  - *EDX* - vstup/výstup, aritmetické operace mul, div
- **Ukazatelé**
  - *EIP* - instruction pointer, ukazatel na následující instrukci
  - *ESP* - offset v rámci programového zásobníku
  - *EBP* - pomáhá při referencování proměnných předávaných do subrutiny
- **Indexové**
  - *ESI* - zdrojový index pro operaci s řetězci
  - *EDI* - cílový index pro operace s řetězci
- **Segmentové**
  - Jejich obsah neměníme, ukládají segmentovou část adresy
  - *CS* (Code Segment), *SS* (Stack Segment), *DS* (Data Segment), *ES* (Extra Segment)

Registry

31 … 16	15 … 8	7 … 0
A … Accumulator	AH	AL
	AX	
EAX		
B … Base	BH	BL
	BX	
EBX		
C … Counter	CH	CL
	CX	
ECX		
D … Data	DH	DL
	DX	
EDX		
Code Segment	CS	
Data Segment	DS	
Extra Data Seg.	ES	
Stack Segment	SS	

31 … 16	15 … 0
SP … Stack Pointer	SP
ESP	
BP … Base Pointer	BP
EBP	
SI … Source Index	SI
ESI	
DI … Destination Index	DI
EDI	
IP … Instruction Pointer	IP
EIP	
registr příznaků	FLAGS
EFLAGS	

H … High byte, **L** … Low byte

E … Extended (= 32bitový režim)

další segmentové registry – FS, GS

(E)FLAGS

- Registr obsahuje 32 bitů (indikátorů), které procesor nastavuje podle výsledku právě provedené operace a umožňuje tak větvit program
- Každá instrukce může nastavovat různý počet příznaků, třeba i žádný
- K registru nelze přistoupit jako k celku, ale lze přistoupit k jeho jednotlivým bitům (flagům)
- Některé bity jsou rezervované a nelze je měnit

Příznaky

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	NT	IOPL		OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

- **OF** (*Overflow Flag*) ~ příznak *overflow* (O, NO)
  - aritmetika se znaménkem (indikace přetečení: OF = 1)
- **CF** (*Carry Flag*) ~ příznak *carry* (C, NC)
  - aritmetika bez znaménka (indikace přetečení: CF = 1), rotace, posuny
- **SF** (*Sign Flag*) ~ příznak *sign* (S, NS)
  - znaménko výsledku (SF = 1: záporný, SF = 0: kladný)
- **ZF** (*Zero Flag*) ~ příznak *zero* (Z, NZ)
  - výsledek = 0 (ZF = 1: výsledek = 0, ZF = 0: výsledek != 0)
- **DF** (*Direction Flag*)
  - směr řetězových instrukcí (DF = 1: zvýšení, DF = 0: snížení)
- PF (*Parity Flag*) ~ příznak *parity* (PE, PO)
  - součet (počet) jedničkových bitů nejnižších 8 bitů výsledku je sudý (P = 1, PE)/lichý (P = 0, PO)
- AF (*Auxiliary Carry Flag*) = přenos z bitu 3 do bitu 4

Poznámky

- Registry *EAX*, *ECX*, *EDX* lze používat libovolně a není potřeba jejich hodnotu na konci funkce obnovovat
- Registry *EBX*, *ESI*, *EDI* lze používat libovolně, na konci funkce je potřeba jejich hodnotu obnovit. Tyto registry mohou být využívány pro lokální proměnné.
- Registry *ESP*, *EBP* jsou používány pro práci se zásobníkem dle popsaného postupu
- V registru *FLAGS* je potřeba vždy na konci funkce zajistit, aby hodnota bytu *DF* byla nastavena vždy na hodnotu 0

Příznaky: příklad

```
%include "rw32-2018.inc"
section .data
section .text
_main:
    push ebp
    mov ebp, esp

    mov AL, 125
    inc AL
    inc AL
    inc AL
    inc AL
    inc AL
    inc AL
    inc AL

    pop ebp
    ret
```

Sekce programu

- 3 sekce
  - `section .data`
    - Deklarace a definice **inicializovaných** dat a konstant – nemění se za běhu.
    - ~ konstantní proměnné v C
  - `section .bss`
    - Deklarace proměnných (definice v programu), nemění se za běhu.
    - Část paměti vyplněná nulami na začátku.
  - `section .text`
    - Kód, fixní velikost v paměti
- **Návěští**
  - Pojmenování **míst v paměti** počítače
  - Definice bodu v programu, na který lze **skočit**
  - **Lokální** návěští je uvozeno znakem **„tečka“** a je platné jen **mezi dvěma globálními návěštími**

```
section .data

    msg db 'Hello, world!', 0

section .bss
    var resb 10

section .text
main:
    mov esi, msg
    call WriteString

    ret
```

Deklarace proměnných

- V *.bss* sektoru, proměnné nejsou inicializované
- `byte_buffer res`**`b`** `64`
  - *rezervuje 64 slabik (byte = 8 bitů)*
- `word_buffer res`**`w`** `1`
  - *rezervuje 1 slovo (word = 2 bajty = 16 bitů)*
- `double_buffer res`**`d`** `2`
  - *rezervuje 2 dvojslova (double word = 4 bajty = 32 bitů)*
- `quad_buffer res`**`q`** `3`
  - *rezervuje 3 quad word (quad word = 8 bajtů = 64 bitů)*

Inicializace proměnných

- V *.data* sektoru, všechny konstanty jsou lokální
- Existuje klíčové slovo **global**, které je zveřejní
- Zkratky **d**efine **b**yte | **w**ord | **d**ouble word | **q**uad word
- `var_byte db 10`
  - *definuje proměnnou o velikosti 8 bitů a hodnotě* 10
- `var_char_array db 'Ahoj', 0`
  - *řetězec*
- `var_dw_array dw 100, 150, 200`
  - *definice polí, hodnoty oddělujeme čárkami*

Uložení čísla v paměti

- Dvě konvence
  - *Big Endian*: MSB (*Most Significant Byte*) je na nejnižší adrese
  - *Little Endian*: LSB (*Least Significant Byte*) je na nejnižší adrese
- Příklad uložení hodnoty 0x12345678 na adrese adresa v 16bitovém režimu znázorňuje tabulka →

	OBSAH PAMĚTI	
FYZICKÁ ADRESA	LITTLE ENDIAN	BIG ENDIAN
0xFFFFF	?	?
adresa + 3	0x12	0x78
adresa + 2	0x34	0x56
adresa + 1	0x56	0x34
adresa	0x78	0x12
0x00001	?	?
0x00000	?	?

Procesory Intel Pentium používají uložení typu **LITTLE ENDIAN**

Adresování paměti

- 32 bitovém režim → všechny adresy budou mít vždy 32 bitů
- Paměť lze indexovat pomocí:
  - Přímé adresy (konkrétní pevná adresa)
  - Nepřímé adresy (registry)
  - Ukazatele přes bázový registr
  - Ukazatel v indexu registru
- Budeme používat hranaté závorky pro čtení/zápis s dané adresy

Adresování paměti: příklad

```
%include "rw32-2018.inc"

section .data
    var1 dw 200
    var2 dw 200,100

    var3 db 7, 15, 155

section .text
_main:

    mov ax,[var1]     ;použití konstantního ukazatele (symbolické adresy)
    mov bx,[var2+2]   ;použití konstantního ukazatele (symbolické adresy) s offsetem
    mov cl,[var3]
    mov ch,[var3+1]
    mov dl,[var3+2]
    ret
```

Závorky => chci hodnotu!

pole „wordů“

1	2	3	4	5	H	e	l	l	o	!	0

C: pole[0] -> 1
pole[10] -> !

Asm: [pole + 0] -> 1
[pole + 10] -> H
[pole + 20] -> !

C vytváří abstrakci, v Asm musíte „velikost lego kostky“ znát sami!