# Architektúra moderných procesoroch

* Základné usporiadanie funkčných častí CPU

## Prúdové spracovanie

* Naraz sa v procesore spracováva viac inštrukcií
* Každý úkon sa vykonáva v samostatnom funkčnom bloku (skupina log. obvodov, ktoré vykonávajú spoločnú prácu)
* Funkčné bloky sú zoradené za sebou – tvoria kanál

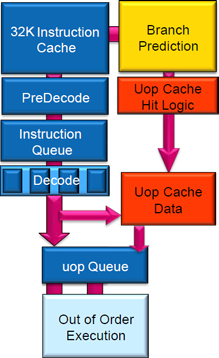
### Úkony spracovania inštrukcie

* + Prenos inštrukcie z pamäte do procesora (IF)
  + Dekódovanie (ID) – konvertovanie do jednoduchých povelov (mikro-operácií)
  + Výber operanda z pamäte (DA)
  + Vykonanie (EX)
  + Zápis výsledku do pamäte (WB)

### Narušenie plynulosti toku

1. Dátová závislosť – zdrojový operand jednej inštrukcie je cieľovým operandom predchádzajúcej inštrukcie
2. Konflikty riadenia – pri inštrukciách vetvenia, preruší sa sekvenčné vykonávanie inštrukcií a riadenie sa prenesie na vzdialené miesto
3. Štrukturálne konflikty – viac inštrukcií súčasne požaduje rovnaký prostriedok procesora

## Intel Haswell

1. Cache pamäť niekoľkých úrovní (L2 Cache, L1 Data Cache, L1 Instruction Cache, L0 Instruction Cache)
2. Jednotka na predpovedanie skokov (Branch Prediction)
3. Jednotka na výber inštrukcie (Instruction Fetch Unit) – prináša inštrukcie z L2 cache do pamäti prvej úrovne
4. Dekodéry
5. Jednotka spracovania inštrukcií mimo poradia (Out-of-Order Unit) – usporiada inštrukcie tak, aby sa odstránila dátová závislosť
6. Výkonný podsystém (Execution Units) – ALU, FPU...

## Dynamické spracovanie inštrukcií

* Súbor metód, ktorých cieľom je zvýšiť efektívnosť práce procesora
* Nepodmienené skoky – procesor vie, na ktorej inštrukcii bude vykonávanie programu pokračovať (môže načítavať ďalšie inštrukcie z pamäte)
* Podmienené skoky – procesor až v okamihu vyhodnotenia podmienky vie, ktorá inštrukcia sa má vykonať – môže však predvídať

### Dynamická predikcia

* Ak sa podmienený skok opakuje, je viac pravdepodobné, že sa jeho podmienka vyhodnotí rovnako ako v predchádzajúcom kroku
* V tabuľke skokov (Branch History Table - BHT) sú uložené stovky posledných skokových inštrukcií (či sa skok vykonal alebo nie)
* Dvojúrovňová predikcia - každá položka BHT má 2 úrovne
  + 1. úroveň – 4-bitový posuvný register, ktorý si pamätá 4 posledné udalosti (ak sa skok vykonal, zapíše sa do najnižšieho bitu 1, ak sa nevykonal, zapíše sa 0. Pred každou aktualizáciou sa register posunie doľava)
  + 2. úroveň – 16 2-bitových čítačov. Obsah registra z prvej úrovne sa vyhodnotí ako poradové číslo čítača z druhej úrovne. Číslo v čítači sa zvýši o 1, ak sa podmienka skoku vyhodnotila ako pravdivá, inak sa zníži o 1.
  + ak sa čítači na 2. úrovni nachádza 0,1 – predpovedá sa, že skok sa nevykoná. Ak tam je 2,3 skok sa vykoná.
* Ak sa podmienený skok doteraz nevyskytol (nemá položky v BHT) – použije sa statická predikcia (štatistické odhady chovania vetviacich inštrukcií)

## Cache pamäť

* Rýchla, slúži ako vyrovnávacia medzi rýchlym procesorom a pomalou hlavnou pamäťou
* Z hľadiska programátora je neprístupná – nemožno ju adresovať
* Stratégia presúvania údajov medzi cache a hlavnou pamäťou je založená na overených vlastnostiach časovej a miestnej lokality
  + **Časová lokalita**
    - Zakladá sa na pravdepodobnosti, že adresa, ktorá bola práve teraz požadovaná, bude čoskoro požadovaná znova
    - Každý údaj, ktorý sa číta z hlavnej pamäti sa automaticky ukladá do cache
  + **Miestna lokalita**
    - Program v krátkom časovom úseku pristupuje k susedným pamäťovým miestam
    - Pri čítaní s hlavnej pamäti sa do cache presúva blok susediacich údajov
* Pri zápise do pamäti musí byť zachovaná konzistencia údajov – zmeny v cache pamäti sa musia premietnuť aj do hlavnej pamäte
  + Priamy zápis (write-through) – údaje sa zapisujú súčasne do cache aj do hlavnej pamäti
  + Oneskorený zápis (write-back) – riadiaca jednotka cache pamäte len eviduje, či bol údaj v cache pamäti zmenený, ak áno, vykoná zápis do hlavnej pamäti v okamihu vylúčenia údaja z cache pamäte
* Rozlišujú sa rôzne úrovne cache pamäti
  + Cache pamäť prvej úrovne – L1 dátová cache, zapisuje v režime write-through do L2
  + Cache pamäť druhej úrovne – L2 cache – dáta sa zapisujú do hlavnej pamäte v režime write-back

## Technológia SIMD

* Vykonanie jednej operácie s viacerými dátovými položkami
* Technológie: MMX, SSE, SSE2, SSE3, SSE4, AVX

## Pracovný režim procesora

* V procesoroch sa používa segmentová organizácia pamäti
* Segment – časť pamäte, ktorá obsahuje inštrukcie alebo dáta. Polohu segmentu v pamäti určí operačný systém pri spustení programu

### Adresa v pamäti

* Logická adresa ľubovoľného miesta v pamäti pozostáva z 2 zložiek:
  + Bázová adresa segmentu – poloha segmentu v pamäti
  + Offset – relatívna adresa vzhľadom na začiatok segmentu
* Zapisuje sa v tvare <báza>:<offset>
* Kombináciou bázovej adresy segmentu a offsetu dostaneme lineárnu adresu (nemusí byť totožná s fyzickou adresou)
* Výhodou segmentovej organizácie je skrátenie adresovej časti inštrukcie
* Typy segmentov
  + Kódový – obsahuje strojový kód inštrukcií
  + Dátový – obsahuje dáta
  + Zásobníkový – návratové hodnoty procedúr, parametre, lokálne premenné, medzivýsledky

### Reálny režim

* Program má neobmedzený prístup ku všetkým pamäťovým miestam a periférnym zariadeniam
* Neposkytuje nástroje na ochranu pamäti ani multitasking
* Procesor 8086 mal 20-bitovú zbernicu (dovoľovala adresovanie 220 bitov pamäti – 1MB). Bázová adresa segmentu aj offset sú 16-bitové čísla.
* Lineárnu adresu dostaneme tak, že bázovú adresu vynásobíme 16-timi (posunieme o 4 bity doľava) a pripočítame offset (v 16-tkovej sústave posunieme len o 1 bit doľava)
* Fyzická adresa je totožná s lineárnou

### Chránený režim

* Procesor 80386 má 32 bitovú adresovú zbernicu – umožňuje adresovať 4 GB pamäti
* Bázová adresa segmentu má 32 bitov. Dĺžka offsetu sa určuje pri deklarácií segmentu v programe (16/32 bitov)
* Lineárnu adresu dostaneme sčítaním bázovej adresy a offsetu
* Lineárna adresa sa prepočíta na fyzickú pomocou stránkovania
* Ak je veľkosť offsetu nastavená na 16 bitov, hovoríme o 16-bitovom režime bez ohľadu na to, či procesor pracuje v reálnom alebo v chránenom režime.
* Ak je veľkosť offsetu 32 bitov, hovoríme o 32-bitovom režime.

### Virtuálny režim

* Predstiera sa reálny režim procesora 8086
* Lineárna 20-bitová adresa sa pomocou stránkovania prepočíta na 32/36 bitovú adresu
* Do virtuálne režimu sa procesor prepne keď spracováva DOS aplikáciu na Windows

### Legacy režim

* V 64-bitovom procesore, kvôli spätnej kompatibilite
* Pre 32-bitový operačný systém – umožňuje spúšťať aplikácie vyvinuté pre reálny/chránený/virtuálny režim

### Long režim

* Umožňuje spúšťať 32-bitové programy napísané pre chránený režim

# Registre

* Pamäťové miesta na čipe procesora
* Umožňujú rýchlejší prístup k operandom
* Používajú sa na výpočet adresy operandu
* Dve skupiny:
  + Užívateľské – univerzálne, segmentové, čítač inštrukcií, register príznakov
  + Systémové – programovanie na úrovni OS

## Univerzálne registre

* Môžu byť naplnené z pamäte, ich obsah môže byť zapísaný do pamäti
* Slúžia na uloženie zdrojových a cieľových operandov
* Môžu obsahovať adresu pamäťového miesta alebo sa podieľať na jej výpočte

### EAX

* akumulátor, najčastejšie používaný register pri presunoch dát do/z pamäte
* Vždy sa používa pri násobení/delení

### BX, BP, SI, DI

* využívajú sa pri relatívnom adresovaní v 16-bit. režime
* BX, BP – bázové registre (ukladá sa do nich prvá adresa (báza) poľa dát)
* ESI, EDI – obsahujú index poľa (source/destination index)

### ECX

* Počítadlo v inštrukciách cyklu (counter)

### EDX

* Pri násobení a delení
* Jediný register, ktorý môže obsahovať adresu portu vstupného alebo výstupného zariadenia v inštrukciách *in* a *out*

### ESP

* Ukazovateľ zásobníka (stack pointer)
* Obsahuje offset vrcholu zásobníka
* Neodporúča sa používať pri aritmetických operáciách

## Segmentové registre

* Obsahujú v reálnom režime bázovú adresu segmentu
* Sú 16 bitové
* CS – bázová adresa kódového segmentu
* SS – bázová adresa zásobníkového segmentu
* DS, ES, FS, GS – bázové adresy dátových segmentov
  + ES – používa sa pri spracovaní polí dát
* V chránenom režime segmentové registre obsahujú selektor – index do tabuľky, kde nájdeme bázovú adresu

## Číta inštrukcií

* EIP (instruction pointer)
* Obsahuje 32-bitový offset inštrukcie, ktorá sa má vykonať ako nasledujúca
* Nasledujúca inštrukcia leží na adrese CS:EIP
* Pri spustení programu na nastavuje na offset prvej inštrukcie, aktualizuje sa po vykonaní inštrukcie

## Register príznakov

* 32-bitový register, obsahuje informácie o výsledku poslednej aritmetickej alebo logickej operácie, o stave procesora a o stave práve spracovávanej úlohy

### Príznak CF (carry flag)

* Nastaví sa na 1, ak výsledok aritmetickej operácie s číslom bez znamienka sa nezmestí do určeného registra, v opačnom prípade sa nastaví na 0
* Nastaví sa na 1 ak došlo k prenosu z najvýznamnejšieho rádu výsledku, aj keď odčítame väčšie číslo od menšieho
* Indikuje výsledok porovnávania dvoch čísel bez znamienka

### Príznak OF (overflow flag)

* Informuje, či výsledok aritmetickej operácie s číslami so znamienkom je v stanovenom rozsahu
* Rovnaká funkcia ako CF s bez-znamienkovými operandami
* Nastaví sa na 1
  + keď pri aritmetickej operácii došlo k pretečeniu do najvyššieho (znamienkového) bitu, ale nedošlo k pretečeniu zo znamienkového bitu
  + Alebo keď došlo k pretečeniu len zo znamienkového bitu

### Príznak SF (sign flag)

* Používa sa, keď spracovávane údaje budeme považovať za čísla so znamienkom
* Indikuje, či výsledok operácie so znamienkovými číslami je kladný (nastaví sa na 0), alebo záporný (nastaví sa na 1)
* Má rovnakú hodnotu ako najvýznamnejší (znamienkový) bit výsledku. napr. po vykonaní inštrukcií

### Príznak AF (auxiliary flag)

* Používa sa pri operáciách s číslami v desiatkovej sústave, ktoré sú uložené vo formáte BCD
* Nastaví sa na 1, ak pri sčítaní/odčítaní došlo k prenosu z 3. do 4. bitu

### Príznak ZF (zero flag)

* Nastaví sa na 1, keď je výsledok operácie 0

### Príznak PF (parity flag)

* Nastaví sa na 1, keď v najnižšom bajte výsledku predchádzajúcej operácie má párny počet bitov hodnotu 1

### Príznak DF (direction flag)

* Jeho hodnotu nastavujeme inštrukciami pred spracovaním polí dát
* AK je nastavený na 0, po spracovaní jedného prvku poľa sa obsah indexového registra zvýši o počet bajtov, ktoré zaberá jeden prvok poľa
* Ak je nastavený na 1, hodnota indexu sa zmenšuje o dĺžku prvku poľa (pole sa spracováva od konca)

### Príznak IF (interrupt enable flag)

* Tiež nastavujeme inštrukciami v programe
* Keď je nastavený na 1, môže byť spracovanie programu procesorom prerušené signálom od periférneho zariadenia
* Keď je nastavené na 0, zakážeme výskyt externého prerušenia

### Príznak TF (trap flag)

* Keď nastavíme tento príznak na 1, prejde procesor do tzv. krokovacieho režimu, v ktorom sa vyvoláva interné prerušenie po každej inštrukcii

### Príznak RF (resume flag)

* Používa sa pri ladení programu
* Nastavenie tohto bitu na 1 spôsobí, že prípadný breakpoint pri spracovaní nasledujúcej inštrukcie bude ignorovaný

### Príznak NT (nested task)

* Spolu s IOPL podporujú multi-programovanie
* Ak je príznak NT nastavený na 1, indikuje, že spracovávaná úloha je vnorená v inej úlohe

### Príznak IOPL (I/O privilege level)

* Dvojbitový, súvisí s prioritou úloh pri viac-úlohovom programovaní

### Príznak VM (Virtual 8086 mode)

* Ak je nastavený na 1, znamená to, že procesor je vo virtuálnom režime
* Príznak sa mení prepnutím úloh

### Príznak AC (Alignment check)

* Ak je nastavený na 1 a zároveň bit AM v riadiacom registri CR0 má hodnotu 1, je povolená kontrola pamäťových odkazov na zarovnanie

### Príznak VIF (virtual interrupt flag)

* Virtuálny príznak povolenia prerušenia
* Aktivuje sa príznakom VIP
* Ak je aktivovaný, má rovnaký význam ako príznak IF

### Príznak VIP (virtual interrupt pending flag)

* Nastavením na 1 sa povoľuje činnosť príznaku VIF

### Príznak ID (identification flag)

* Určuje, či je procesor schopný vykonávať inštrukciu *cpuid*, ktorá identifikuje typ procesora

# Spôsoby adresovania

* Rôzne spôsoby, ako sa v inštrukciách odvolávame na operandy v registroch alebo v pamäti

## Implicitné adresovanie

* Operand v zápise inštrukcie nevidíme
* Napr. *mul BL* – vynásobí AL\*BL a výsledok uloží do AX

## Explicitné adresovanie

### Priamy operand (adresa 0. rádu)

* Hodnota operandu je zapísaná priamo v inštrukcií
* Napr. *mov AL, 3*

### Priama adresa (adresa 1. rádu)

* Používa sa najčastejšie
* V inštrukcií je zapísané označenie alebo číslo pamäťového registra
* Napr. *add DX, [Alfa]* – pričíta k DX obsah premennej Alfa. Obidva operandy sú priamou adresou
* Meno premennej interpretuje prekladač ako offset v dátovom segmente
* Hranaté nie sú povinné – zdôrazňujú že pracujeme s hodnotou premennej a nie s jej adresou

### Nepriama adresa (adresa 2. rádu)

* V inštrukcii je zapísaná adresa pamäťového miesta, na ktorom nájdeme adresu operandu (na tomto mieste nájdeme priamu adresu)
* Napr. *mov AL, [EBX]* – do registra AL skopíruje obsah bajtu na adrese (offset), ktorá je uložená v EBX

# Premenné a návestia

* Symbolické adresy, prekladač ich chápe ako offset dátového objektu v príslušnom segmente

## Premenné

* Definujeme pomocou direktív, ktoré vyhradzujú premenným miesto v pamäti a zároveň im prideľujú hodnotu
* Deklarácia: *[meno\_premennej] Dx výraz*
* Direktívy prideľujú pamäť v jednotkách, ktoré sú určené druhým písmenom direktívy (viď tabuľka)
* Jedna direktíva môže prideliť jednu alebo viac jednotiek (ak za Dx nasleduje viac výrazov)
* Výrazom môže byť:
  + Číselný výraz obsahujúci konštanty
  + Adresový výraz
  + Znakový výraz
  + Znak *?* – direktíva len pridelí jednotku pamäti, neinicializuje ju
  + Konštrukcia *n dup (...)* – n-násobná inicializácia hodnotou uvedenou v zátvorke

### Direktívy definujúce premenné

|  |  |  |  |
| --- | --- | --- | --- |
| **Direktíva** | **Počet bajtoch** | **Typ premennej** | **Môžeme uložiť** |
| DB | 1 | byte | <-128;127>  <0;255> |
| DW | 2 | word | <-32768;32767>  <0; 65535> |
| DD | 4 | dword | <-231; 231­-1>  <0; 232­-1> |
| DP  DF | 6 | pword  fword | <-247; 247­-1>  <0; 248­-1> |
| DQ | 8 | qword | <-263; 263­-1>  <0; 264-1> |
| DT | 10 | tbyte | <-279; 279­-1>  <0; 280-1> |

## Návestia

* Indikujú miesto v programe, na ktoré sa chceme odvolávať – symbolické adresy inštrukcií

### Návestie typu near

* Ak sa naň budeme odvolávať z rovnakého segmentu, na ktorom je návestie definované
* Definuje sa pomocou direktívy „:“
  + Napr.
    - Cyklus:
    - mov AX, BX
* Alebo pomocou direktívy LABEL
  + Napríklad
    - Cyklus LABEL near
    - mov AX, BX
* Alebo pomocou direktívy PROC
  + Napríklad
    - Faktorial PROC near

### Návestie typu far

* Ak sa odkaz na návestie nachádza v inom segmente
* Definujeme pomocou direktívy LABEL
  + Napríklad
    - Urob LABEL far
    - Mov AX, BX
* Alebo pomocou direktívy PROC
  + Napríklad
    - Faktorial PROC near

## Atribúty premenných a návestí

* Premenné a návestia sú charakterizované atribútmi:
  + Segment – bázová adresa segmentu
  + Offset – offset v segmente
  + Typ
* Môžeme používať operátory, ktoré vracajú alebo prepisujú hodnoty týchto atribútov

### Operátory premenných a návestí

|  |  |
| --- | --- |
| **Operátor** | **Význam** |
| seg | Vráti hodnotu atribútu segment |
| offset | Vráti hodnotu atribútu offset |
| type | Vráti hodnotu typu premennej  1 – byte  2 – word  4 – dword  6 – fword/pword  8 – qword  10 – tbyte  -1 – near  -2 - far |
| length | Vráti počet jednotiek pamäti pridelených premennej (vráti viac ako 1 len ak použijeme *dup*) |
| size | Vráti length\*type |
| ptr | Mení typ premennej |

# Symbolické konštanty

* Definujeme pomocou direktívy EQU alebo „=“ (konštantu definovanú pomocou „=“ možno predefinovať)
* Vyžívajú symbol $ - jeho hodnota sa prideľuje pri preklade a je ňou aktuálny offset v danom segmente
* Napr.: *PocetZnakov EQU $-Retazec*

# Inštrukčný súbor

## Inštrukcie pre presun údajov

* Nemenia príznakové bity

### mov – move data

* mov cieľ, zdroj
* Uloží zdrojový operand (univerzálny, segmentový register, pamäť, konštanta) do cieľového (univerzálny, segmentový register, pamäť)
* Nie je dovolená operácia typu pamäť – pamäť

### lea – load effective address

* lea registre, pamäť
* Uloží offset pamäťového operandu do univerzálneho registra

### movzx – move with zero-extend

* movzx register, register/pamäť
* Prenesie pravý 8 alebo 16-bitový operand do ľavého 16 alebo 32-bitového, pričom ho doplní binárnymi nulami do požadovanej veľkosti

### movsx – move with sign-extend

* movsx register, register/pamäť
* Vykoná znamienkové rozšírenie pravého 8 alebo 16-bitového operandu do ľavého 16 alebo 32-bitového operandu
* Nadbytočné bity sa naplnia hodnotou znamienkového bitu zdrojového operandu

### xchg – exchange

* xchg register/pamäť, register/pamäť
* Vymení obsahy pravého a ľavého operandu

## Aritmetické inštrukcie

### add – add

* add register/pamäť, register/pamäť/číslo
* Sčíta obidva operandy a výsledok uloží do ľavého

### sub – subtract

* sub register/pamäť, register/pamäť/číslo
* Odčíta pravý operand od ľavého a výsledok uloží do ľavého operandu

### adc – add with carry

* adc register/pamäť, register/pamäť/číslo
* Sčíta obidva operandy a hodnotu príznakového bitu CF a výsledok uloží do ľavého operandu
* Používa sa pri sčítaní celých čísiel dlhších než veľkosť univerzálneho registra

### sbb – subtract with borrow

* sbb register/pamäť, register/pamäť/číslo
* Odčíta pravý operand a príznak CF od ľavého operandu a výsledok uloží do ľavého operandu
* Používa sa pri odčítaní čísiel väčšieho rozsahu, ktoré sa nezmestia do univerzálnych registrov

### inc – increment

* inc register/pamäť
* Zvýši hodnotu operandu p 1
* Nemení príznak CF

### dec – decrement

* dec register/pamäť
* Zníži hodnotu operandu p 1
* Nemení príznak CF

### cmp – compare

* cmp register/pamäť, register/pamäť/číslo
* Porovná operandy tak, že pravý odčíta od ľavého a nastaví príznakové bity podľa výsledku operácie

### mul – unsigned multiply

* mul register/pamäť
* Vynásobí operand obsahom akumulátora (AL, AX, EAX)

### imul – signed multiply

* imul register/pamäť
* imul register, register/pamäť/číslo
* imul register, register/pamäť, číslo
* Vykonáva násobenie hodnôt so znamienkom

### div – unsigned divide

* div register/pamäť
* Vykonáva celočíselné delenie bez znamienka

### idiv - signed divide

* idiv register/pamäť
* Vykonáva celočíselné delenie čísiel so znamienkom

### Implicitné operandy celočíselného delenia

* Delenec/deliteľ
* Operand - deliteľ

|  |  |  |  |
| --- | --- | --- | --- |
| **Delenec** | **Deliteľ** | **Podiel** | **Zvyšok** |
| AX | 8-bit | AL | AH |
| DX:AX | 16-bit | AX | DX |
| EDX:EAX | 32-bit | EAX | EDX |

### cbw – convert byte to word

* Konvertuje 8-bitovú hodnotu so znamienkom v registri AL na 16-bitovú hodnotu so znamienkom
* Výsledok sa uloží do AX

### cwd – convert word to doubleword

* Konvertuje 16-bitovú hodnotu so znamienkom v registri AX na 32-bitovú hodnotu so znamienkom
* Výsledok sa uloží do DX:AX

### cwde – convert word to doubleword extended

* Konvertuje 16-bitovú hodnotu so znamienkom v registri AX na 32-bitovú hodnotu so znamienkom v registri EAX

### cdq – convert doubleword to quadword

* Konvertuje 32-bitovú hodnotu so znamienkom v registri EAX na 64-bitovú hodnotu so znamienkom v dvojici registrov EDX:EAX

### neg

* neg register/pamäť
* Zmení znamienko operandu (urobí jeho dvojkový komplement)
* Príznak CF nastaví na 0, ak má operand nulovú hodnotu, inak na 1

## Logické inštrukcie

### and – logical and

* and register/pamäť, register/pamäť/číslo
* Vykonáva logický súčin

### or – logical inclusive or

* or register/pamäť, register/pamäť/číslo
* Vykonáva logický súčet

### xor – logical exclusive or

* xor register/pamäť, register/pamäť/číslo
* Vykonáva antivalenciu (nezhoda)

### test – logical compare

* test register/pamäť, register/pamäť/číslo
* Vykoná logický súčin operandov, podľa výsledku operácie nastaví príznakové bity, ale výsledok neuloží
* Používa sa vtedy, keď chceme zistiť, akú hodnotu má zvolený bit ľavého operandu

### not – one’s complement negation

* not register/pamäť
* Invertuje hodnoty všetkých bitov operandu

## Inštrukcie pre posuv a rotáciu

* Majú dva operandy
  + Ľavý – register alebo pamäťové miesto (bajt, slovo, dvoj-slovo), ktorého bity sa posúvajú
  + Pravý – konštanta alebo register CL, o koľko miesta sa posunú bity

### shl – shift logical left

* Posúva všetky bity ľavého operandu doľava
* Do najnižšieho bitu sa zapisuje 0
* Najvyšší byt sa posúva do CF
* Ak je počet posuvov 1 a pred posunom boli hodnoty dvoch najvyšších bitov operandu rovnaké, nastaví sa OF na 0, inak na 1
* Používa sa na rýchle násobenie bez znamienka 2/4/8/16

### sal – shift arithmetic left

* Posúva všetky bity ľavého operandu doľava
* Do najnižšieho bitu sa zapisuje 0
* Najvyšší byt sa posúva do CF
* Ak je počet posuvov 1 a pred posunom boli hodnoty dvoch najvyšších bitov operandu rovnaké, nastaví sa OF na 0, inak na 1
* Používa sa na rýchle násobenie bez znamienka 2/4/8/16

### shr – shift logical right

* Posúva bity ľavého operandu doprava
* Do najvyššieho bitu sa zapisuje 0
* Najnižší bit sa posúva do CF
* Ak je počet posuvov 1 a po posuve sú hodnoty dvoch najvyšších bitov operandu rovnaké, nastaví sa OF na 0, inak na 1
* Používa sa na rýchle delenie bez znamienka 2/4/8/16

### sar – shift arithmetic right

* Posúva bity ľavého operandu doprava
* Hodnota najvyššieho bitu sa nemení
* Najnižší bit sa posúva do CF
* Ak je počet posuvov 1, OF sa vynuluje
* Používa sa na rýchle delenie so znamienkom 2/4/8/16

### rol – rotate left

* Rotuje všetky bity doľava o určitý počet miest
* Hodnota najvyššieho bitu sa presúva do najnižšieho a zároveň do CF
* Ak sa vykonala rotácia o 1 bit a pred rotáciou boli hodnoty dvoch najvyšších bitov operandu rovnaké, nastaví sa OF na 0, inak na 1

### ror – rotate right

* Rotuje všetky bity doprava o určitý počet miest
* Hodnota najnižšieho bitu sa presúva do najvyššieho a zároveň do CF
* Ak sa vykonala rotácia o 1 bit a pred rotáciou boli hodnoty dvoch najvyšších bitov operandu rovnaké, nastaví sa OF na 0, inak na 1

### rcl – rotate left through carry

* Rotuje všetky bity doľava o určitý počet miest
* Hodnota z CF sa presúva do najnižšieho bitu
* Najvyšší bit sa presúva do CF
* Ak sa vykonala rotácia o 1 bit a pred rotáciou boli hodnoty dvoch najvyšších bitov operandu rovnaké, nastaví sa OF na 0, inak na 1

### rcr – rotate right through carry

* Rotuje všetky bity doprava o určitý počet miest
* Hodnota z CF sa presúva do najvyššieho bitu
* Najnižší bit sa presúva do CF
* Ak sa vykonala rotácia o 1 bit a po rotácii sú hodnoty dvoch najvyšších bitov operandu rovnaké, nastaví sa OF na 0, inak na 1

## Skokové inštrukcie

* Menia postupné poradie vykonávania inštrukcií
* 2 typy skokov
  + **Priamy skok** – operandom inštrukcie skoku je návestie
  + **Nepriamy skok** – operandom inštrukcie skoku je premenná/register ktorý obsahuje adresu novej inštrukcie

### Podmienené skoky

* Inštrukcia *jmp návestie*

### Podmienené skoky

* Umožňujú vetvenie programu v závislosti od príznakov ZF, CF, SF, PF a OF
* Môžu byť len priame
* Inštrukcia *jcxz návestie* (jump if CX is 0)
* Inštrukcia *jecxz návestie* (jump if ECX is 0)
* Inštrukcia *jcc návestie* (cc - kód)
* Podmienky ABOVE/BELOW – čísla bez znamienka
* Podmienky LESS/GREATER – čísla so znamienkom

|  |  |  |
| --- | --- | --- |
| **Inštrukcia** | **Význam** | **Podmienka** |
| jb  jnae  jc | jump if below  jump if not above or equal  jump if carry | CF=1 |
| jae  jnb  jnc | jump if above or equal  jump if not below  jump if not carry | CF=0 |
| jbe  jna | jump if below or equal  jump if not above | CF=1 or ZF=1 |
| ja  jnbe | jump if above  jump if not below or equal | CF=0 and ZF=0 |
| je  jz | jump if equal  jump if zero | ZF=1 |
| jne  jnz | jump if not equal  jump if not zero | ZF=0 |
| jl  jnge | jump if less  jump if not greater or equal | SF!=OF |
| jge  jnl | jump if greater or equal  jump if not less | SF=OF |
| jle  jng | jump if less or equal  jump if not greater | ZF=1 or SF!=OF |
| jg  jnle | jump if greater  jump if not less or equal | ZF=0 and SF=OF |
| jp  jpe | jump if parity  jump if parity even | PF=1 |
| jnp  jpo | jump if not parity  jump if parity odd | PF=0 |
| js | jump if sign | SF=1 |
| jns | jump if not sign | SF=0 |
| jo | jump if overflow | OF=1 |
| jno | jump if not overflow | OF=0 |

## Inštrukcie cyklu

### loop

* loop návestie
* Dekrementuje ECX a porovná ho s nulou (nezmení príznakové bity). AK je ECX rôzny od nuly, skočí na návestie

### loope, loopz

* loope návestie, loopz návestie
* Dekrementuje ECX a porovná ho s nulou, ak je ECX rôzny od nuly a ZF=1, skočí na návestie

### loopne, loopnz

* loopne návestie, loopnz návestie
* Dekrementuje ECX a porovná ho s nulou, ak je ECX rôzny od nuly a ZF=0, skočí na návestie

## Inštrukcie pre prácu so zásobníkom

### push (push onto stack)

* push operand
* Uloží operand na vrchol zásobníka
* Operandom môže byť 16/32 bitový univerzálny register, segmentový register, premenná typu word/dword, a číslo (zaberie dve slová)
* Procesor pri ukladaní do zásobníka zmenší ukazovateľ zásobníka (ESP) o 2/ 4 (podľa typu operanda) a potom uloží operand do zásobníkového segmentu na offset daný registrom ESP

### pop (pop from stack)

* pop operand
* Vyberá zo zásobníka jedno slovo alebo dvoj-slovo
* Operandom môže byť 16/32 bitový univerzálny register alebo pamäťové miesto typu word/dword
* Slovo/dvoj-slovo sa skopíruje z adresy ESP do operandu a potom sa ESP zvýši o 2/4

### pushad (push all general registers)

* Uloží do zásobníka obsahy všetkých 32-bitových univerzálnych registrov v poradí EAX, ECX, EDX, EBX, ESP (pôvodný obsah), EBP, ESI, EDI
* Upraví ukazovateľ zásobníka ESP -= 32

### popad (pop all general registers)

* Skopíruje 16 slov z vrcholu zásobníka do 32-bitových univerzálnych registrov v poradí EDI, ESI, EBP, ďalšie dve slová sa ignorujú, EBX, EDX, ECX, EAX
* Upraví ukazovateľ zásobníka: ESP += 32

### pushf (push flags onto stack)

* Uloží do zásobníka dolnú polovicu registra príznakov EFLAGS

### pushfd (push flags onto stack)

* Uloží do zásobníka 32-bitový register príznakov EFLAGS

### popf (pop from stack into flags)

* Skopíruje slovo z vrcholu zásobníka do dolnej polovice registra príznakov EFLAGS a zvýši ukazovateľ zásobníka o 2 (vyberie EFLAGS zo zásobníka)

### popfd (pop from stack into flags)

* Vyberie zo zásobníka register EFLAGS

## Procedúry

* Deklarujeme: *meno\_procedúry PROC [jazyk] [uses registre] [parameter1[,parameter2]…]*. Potom nasleduje telo procedúry a koniec *meno ENDP*
* Meno procedúry je symbolická adresa prvej inštrukcie v procedúre

### call (call procedure)

* call meno-procedúry
* Uloží do zásobníka návratovú adresu a vykoná skok na prvú inštrukciu procedúry.
* Návratová adresa je aktuálna hodnota EIP (offset inštrukcie, ktorá nasleduje na inštrukciou call)
* Volanie procedúry môže byť priame alebo nepriame (call register/pamäťový operand)

### ret (return from procedure)

* ret, ret číslo
* Vyberie zo zásobníka návratovú adresu a uloží ju do registra EIP
* Môže myť priamy operand, ktorý sa po výbere návratovej adresy pripočíta k ESP – po výbere návratovej hodnoty sa v zásobníku preskočí o toľko bajtov, koľko udáva operand inštrukcie
* Ret s parametrom sa používa, keď sa do procedúry odovzdávajú parameter cez zásobník

### Odovzdávanie parametrov procedúre

* Dva druhy:

1. V registroch – čisté assemblerovské programy
2. V zásobníku
   * Parametre musíme do zásobníka uložiť predtým ako vyvoláme procedúru. V procedúre parameter sprístupnime pomocou nepriameho adresovania s EBP

#### Symbolické mená parametrov

* Z parametrov sa stavajú formálne parametre (môžeme predpísať typ – parameter:byte, implicitne je dword)
* Voláme: *INVOKE meno parametre*

#### Špecifikácia jazyka deklarácií procedúry

* Informuje prekladač o poradí ukladania parametrov do zásobníka pri volaní procedúry a o tom, kto ich zo zásobníka odstráni
  + Pascal, Basic, Fortan – zľava doprava
  + C, Prolog – sprava doľava

#### Špecifikácia USES v direktíve PROC

* Definuje registre (max 8 oddelené medzerami), ktorých obsahy budú automaticky na začiatku procedúry uložené do zásobníka a pred návratom z procedúry opäť automaticky obnovené, aby sa zachovali ich pôvodné hodnoty

### Lokálne premenné v procedúre

* Pamäťové miesta v zásobníku nad návratovou adresou a odloženým registrom EBP
* Pristupujeme k nim podobne ako k parametrom (nepriame adresovanie s EBP)
* Deklarácia: LOCAL premenná1 (implicitne word)

## Inštrukcie pre nastavenie príznakových bitov

* Nemajú operand
* Nastavujú vždy 1 príznakový bit na 0 alebo 1. Ostatné príznaky nemenia

### Inštrukcie

* clc - clear carry flag
* stc – set carry flag
* cmc – complement carry flag (neguje CF)
* cld – clear direction flag
* std – set direction flag
* cli – clear interrupt flag
* sti – set interrupt flag
* lahf – load flags into AH (skopíruje dolných 8 bitov FLAGS do AH). AH - SF,ZF,0,AF,0,PF,1,CF
* sahf – store AH to flags (skopíruje AH do dolného bajtu FLAGS)

## Reťazcové inštrukcie

* Slúžia na spracovanie polí typu BYTE, WORD alebo DWORD
* Polia môžu obsahovať akékoľvek čísla, nie len znaky
* Do ESI musíme uložiť OFFSET zdrojového reťazca
* Do EDI musíme uložiť OFFSET cieľového reťazca
* Po vykonaní operácie s jedným prvkom reťazca sa ESI/EDI aktualizujú
  + Ak je DF 0 - ESI/EDI sa zvyšujú (reťazec sa spracováva dopredu)
  + A k je DF 1 - ESI/EDI sa znižujú (reťazec sa spracováva dozadu)
* Typ reťazca oznámime inštrukcii
  + Meno reťazca uvedieme ako operand
  + Alebo inštrukciu zapíšeme s príponou
    - B – BYTE
    - W – WORD
    - D – DWORD
* Prefix REPE (REPZ) opakuje reťazcovú inštrukciu, kým ECX!=0 a zároveň ZF=1
* Prefix REPNE (REPNZ) opakuje reťazcovú inštrukciu, kým ECX!=0 a zároveň ZF=0

### movs (move data from string to string)

* movs cieľový reťazec, zdrojový reťazec
* movsb, movsw, movsd
* Kopíruje dáta z pamäti, na ktorú ukazuje ESI, do pamäti, na ktorú ukazuje EDI

### lods (load data from string)

* lods zdrojový reťazec
* lodsb, lodsw, lodsd
* Skopíruje bajt reťazca do AL (resp. WORD do AX alebo DWORD do EAX)
* Nemení príznaky

### stos (store data to string)

* stos cieľový reťazec
* stosb, stosw, stosd
* Skopíruje obsah AL (resp. AX alebo EAX) do reťazca
* Nemení príznaky

### scas (scan string)

* scas cieľový reťazec
* scasb, scasw, scasd
* Porovná bajt (slovo alebo dvojslovo) reťazca s obsahom AL/AX/EAX tak, že odčíta prvok reťazca od AL/AX/EAX a nastaví príznaky

### cmps (compare string operands)

* cmps cieľový reťazec, zdrojový reťazec
* cmpsb, cmpsw, cmpsd
* Porovná reťazce tak, že bajt (slovo alebo dvojslovo) cieľového reťazca odčíta od bajtu (slova alebo dvojslova) zdrojového reťazca a nastaví príznaky

## Makro inštrukcie

* Makro je blok textu, ktorému priradíme meno
* Vždy, keď prekladač nájde v zdrojovom kóde meno makra, nahradí ho textom, ktorý makro reprezentuje (makro sa rozvinie)
* Makro sa vykoná rýchlejšie ako procedúra ale nešetrí pamäť
* Syntax:
  + NázovMakra MACRO parametre
  + Kód makra
  + ENDM

## BCD aritmetika

* V BCD kóde je každá číslica desiatkového čísla zobrazená v 4 bitoch ako dvojkové číslo
* Nezhustený formát:
  + Každá číslica je uložená v jednom bajte (horné 4 bity sú nulové)
  + 01h pred prvou číslicou – číslo je kladné
  + 80h pred prvou číslicou – číslo je záporné
* Zhustený formát:
  + V jednom bajte sú uložené 2 číslice
  + 01h pred prvou číslicou – číslo je kladné
  + 8h pred prvou číslicou – číslo je záporné
* Pred číslom sa vyhradí 1 bajt – počet číslic
* Najmenej významná číslica (dvojica číslic) by mala byť na najnižšej adrese
* Operácie násobenia a delenia môžeme vykonávať len s BCD číslami v nezhustenom formáte

### daa (decimal adjust for addition)

* Vykonáva korekciu výsledku sčítania dvoch čísel v zhustenom formáte BCD v registri AL
* Ak v dolných 4 bitoch registra AL je číslo > 9 alebo AF==1, pripočíta sa 6 k obsahu registra AL a AF=1, inak AF=0
* Ak v horných 4 bitoch registra AL je číslo > 9 alebo CF==1, pripočíta sa 60h k obsahu registra AL a CF=1, inak CF=0

### aaa (ASCII adjust for addition)

* Vykonáva korekciu výsledku sčítania dvoch čísel v nezhustenom formáte BCD v registri AL
* Ak obsah registra AL > 9 alebo AF==1, pripočíta sa 0F6h k registru AX a príznaky AF a CF sa nastavia na 1. V opačnom prípade sa príznaky AF a CF vynulujú.

### das (decimal adjust for subtraction)

* Vykonáva korekciu výsledku odčítania dvoch čísel v zhustenom formáte BCD v registri AL
* Ak v dolných 4 bitoch registra AL je číslo > 9 alebo AF==1, odčíta sa 6 od obsahu registra AL a AF=1, inak AF=0
* Ak v horných 4 bitoch registra AL je číslo > 9 alebo CF==1, odčíta sa 60h od obsahu registra AL a CF=1, inak CF=0

### aas (ASCII adjust for subtraction)

* Vykonáva korekciu výsledku odčítania dvoch čísel v nezhustenom formáte BCD v registri AL
* Ak obsah registra AL > 9 alebo AF==1, odčíta sa 0F6h od registra AX a príznaky AF a CF sa nastavia na 1. V opačnom prípade sa príznaky AF a CF vynulujú.

### aam (ASCII adjust for multiplication)

* Vykonáva korekciu výsledku násobenia dvoch BCD čísiel v nezhustenom formáte v registri AX
* Číslo v registri AX upraví tak, aby významnejšia číslica výsledku bola v registri AH a menej významná číslica v AL.

### aad (ASCII adjust for division)

* Prevedie dve BCD číslice v nezhustenom formáte v registri AX na dvojkové číslo (vynásobí obsah AH desiatimi a pripočíta AL)
* Nasledujúca inštrukcia div potom nechá výsledok delenia v nezhustenom BCD formáte v registri AL a zvyšok po delení v nezhustenom BCD formáte v registri AH

## Reálna aritmetika

* Zabezpečuje FPU, časti:
  + Registre (80 bitov) – sú v nich operandy inštrukcií, st(0) - st(7)
  + Register známok
  + Stavový register
  + Riadiaci register

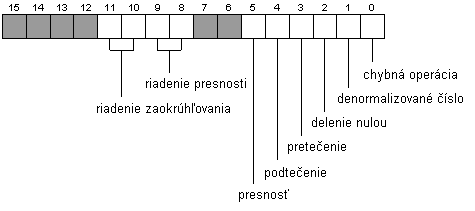
### Register známok

* 16 bitový
* Obsahuje 8 známok (pre každý register st(i))
* Známka je dvojbitová hodnota, ktorá informuje o obsahu registra
  + 00 – register obsahuje platné číslo
  + 01 – register obsahuje nulu
  + 10 – obsah registra je špeciálny (denormalizované číslo, nenormalizované číslo, nekonečno alebo neplatné číslo)
  + 11 – register je prázdny

### Stavový register

* Bity 0-5 – výnimky (informujú o výsledku predchádzajúcej operácie)
* Ukazovateľ zásobníka – číslo registra, ktorý je na vrchole zásobníka
* C0, C2, C3 – podmienkové bity, nastavujú sa pri porovnaní čísel
* Chybná operácia – napr. odmocnina zo záporného čísla
* Presnosť – ak výsledok nie je presne zobraziteľný (napr. 1/3)
* Chyba zásobníka – pretečenie alebo podtečenie zásobníka (podľa C1 zistíme, čo nastalo)

### Riadiaci register



* Bity 0 až 5 maskujú výnimky 0 až 5 zaznamenané v stavovom registri
  + Ak maska obsahuje 0 - pri vzniku príslušnej výnimky FPU generuje prerušenie
  + Ak maska obsahuje 1 - FPU výnimku len zaznamená v stavovom registri
* Bity 8-9 – určujú presnosť priebehu výpočtu
  + 00 – 24 bitov
  + 01 – nepoužitá
  + 10 – 53 bitov
  + 11 – 61 bitov
* Bity 10-11 – spôsob zaokrúhľovania
  + 00 – k najbližšiemu alebo párnemu číslu
  + 01 – dole
  + 10 – hore
  + 11 – odrezanie (zaokrúhľovanie k 0)

### Zaokrúhľovanie

* Na 3 desatinné miesta

|  |  |  |
| --- | --- | --- |
| **Metóda** | **1.0111** | **-1.0111** |
| K najbližšiemu | 1.100 | -1.100 |
| Dole | 1.011 | -1.100 |
| Hore | 1.100 | -1.011 |
| Odrezanie | 1.011 | -1.011 |

### fld

* fld premenná/st(i)
* Prenesie operand do st(0)
* Operand – reálna premenná, st(i)

### fild

* fild premenná
* Prenesie operand do st(0)
* Operand – celočíselná premenná

### fbld

* fbld premenná
* Prenesie operand do st(0)
* Operand – BCD premenná

### fst

* Premenná/st(i)
* Prenesie st(0) do operandu

### fist

* Premenná
* Prenesie st(0) do operandu

### fstp, fistp, fbstp

* Prenesie st(0) do operandu a odstráni st(0) zo zásobníka

### fxch

* fsch st(i)
* Vymení st(0) a st(i)
* Tvar bez operandu – st(i) je st(1)

### Vloženie konštanty do st(0)

* fldz – 0
* fld1 – 1
* fldpi – PI
* fldl2t – log 2(10)
* fldl2e – log 2(e)
* fldlg2 – log 10(2)
* fldln2 – log e(2)

### fadd, fiadd

* fadd premenná
* Pripočíta hodnotu operandu k st(0)

### fadd

* fadd st(0), st(i)
* fadd st(i), st(0)
* Sčíta hodnoty dvoch operandov a výsledok uloží do ľavého operandu

### faddp

* faddp st(i), s(0)
* Spočíta hodnoty registrov st(i) a st(0), výsledok uloží do st(i) a odstráni položku z vrcholu zásobníka

### Funkčné operácie s st(0)

* fchs st(0) = -(st(0))
* fsqrt st(0) = sqrt(st(0))
* fabs st(0) = |st(0)|
* fsin1 st(0) = sin(st(0))
* fcos1 st(0) = cos(st(0))

### fcom, ficom

* fcom operand
* Porovná operand s st(0) a nastaví príznaky C0, C2 a C3
* fcom bez operandu – porovná st(1) a st(0)

### fcomp, ficomp

* Porovná operand s st(0) a nastaví príznaky C0, C2 a C3 a odstráni položku z vrcholu zásobníka

### fcompp

* Porovná st(1) a st(0) a obidve položky odstráni zo zásobníka

### ftst

* Porovná st(0) s nulou

### Nastavovanie príznakov

|  |  |  |  |
| --- | --- | --- | --- |
| **C3** | **C2** | **C0** | **Výsledok operácie** |
| 0 | 0 | 0 | st(0) > operand |
| 0 | 0 | 1 | st(0) < operand |
| 1 | 0 | 0 | st(0) == operand |
| 1 | 1 | 1 | Nemožno porovnať |

### Riadiace inštrukcie

* finit – inicializuje FPU, stavový register nastaví na 0, riadiaci na 37Fh, register známok na 0FFFFh
* fldcw pamäť – prenesie riadiace slovo z pamäti do riadiaceho registra
* fstcw pamäť – uloží obsah riadiaceho registra do slova v pamäti
* fstsw pamäť/AX – uloží obsah stavového registra do slova v pamäti alebo do AX

# Služby OS Windows

## MessageBoxA

* Parametre
  + hWnd– handle okna, ktorému okno patrí. NULL reprezentuje pracovnú plochu – okno sa zobrazí v strede obrazovky
  + lpText – smerník na text ukončený nulou, ktorý sa zobrazí v dialógovom okne
  + lpCation - smerník na text ukončený nulou, ktorý sa zobrazí v hornej lište okna
  + uType – konštanta, ktorá špecifikuje tvar a správanie dialógového okna
* Návratová hodnota
  + Ak funkcia prebehne úspešne, vráti nulu
  + Inak vráti celočíselnú hodnotu – čo užívateľ stlačil, keď zatvoril okno (IDYES, IDNO, IDOK...)
* Príklad
  + INVOKE MessageBoxA, NULL, OFFSET text, OFFSET text-hore, MB\_ICONWARRING

## CreateFile

* Vytvorí súbor a zároveň ho otvorí pre požadovaný spôsob prístupu, alebo otvorí už existujúci súbor
* Popisovač súboru vráti v EAX
* V prípade chyby vráti INVALID\_HANDLE\_VALUE(-1)
* Parametre
  + lpFileName – adresa premennej s menom súboru
  + dwDesiredAccess – požadovaný prístup
    - 0- aplikácia chce zistiť, či súbor existuje alebo aké ma atribúty
    - GENERIC\_READ – na čítanie
    - GENERIC WRITE – na zapisovanie
  + dwSHareMode – režim zdieľania
    - 0 – aplikácia požaduje výhradný prístup
    - FILE\_SHARE\_READ – iné procesy môžu otvoriť súbor na čítanie
    - FILE\_SHARE\_WRITE - iné procesy môžu otvoriť súbor na zápis
  + lpSecurityAttributes
  + dwCreationDisposition – čo sa má robiť, ak súbor už existuje/neexistuje
    - CREATE\_NEW – vytvorí nový súbor, ak už existuje – vyhlási chybu
    - CREATE\_ALWAYS – vytvorí nový súbor, ak už existuje – prepíše ho
    - OPEN\_EXISTING – otvorí existujúci súbor, ak neexistuje – vyhlási chybu
    - OPEN\_ALWAYS – otvorí existujúci súbor, ak neexistuje – vytvorí ho
    - TRUNCATE\_EXISTING – otvorí existujúci súbor a jeho dĺžku nastaví na 0 bajtov Ak neexistuje – vyhlási chybu (musí byť otvorený s GENERIC\_WRITE)
  + dwFlagsAndAttributes
  + hTemplateFile
* Príklad
  + INVOKE CreateFileA, OFFSET fileName, GENERIC\_WRITE, 0, NULL, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL
  + mov fileHandle, EAX

## ReadFile

* Číta zo súboru od pozície danej hodnotu ukazovateľa v súbore. AK čítanie prebehlo bez chyby, vráti TRUE, inak FALSE
* Parametre
  + hFile – popisovač súboru
  + lpBuffer – adresa premennej, do ktorej sa uložia prečítané dáta
  + nNumberOfBytesToRead – počet bajtov, ktoré sa majú prečítať
  + lpNumberOfBytesRead – adresa premennej, do ktorej sa uloží počet načítaných bajtov
  + lpOverlapped – adresa štruktúry, ktorá sa požíva asynchrónnom prístupe. Pri synchrónnom prístupe – NULL
* Príklad
  + INVOKE ReadFile, fileHandle, OFFSET where, bytesToRead, OFFSET bytesDone, NULL

## WriteFile

* Zapisuje do súboru od pozície danej hodnotou ukazovateľa
* Parametre ako ReadFile
* Príklad:
  + INVOKE WriteFile, fileHandle, OFFSET data, bytesToWrite, OFFSET bytesDone, NULL

## CloseHandle

* Zatvorí súbor
* Parametre:
  + hObject – popisovač súboru
* Príklad:
  + INVOKE CloseHandle, fileHandle

# Knižnica IRVINE

* INCLUDE Irvine32.inc

## Procedúry na vstup a výstup

### Clrscr

* Vymaže obrazovku

### Crlf

* Vypíše nový riadok

### ReadChar

* Vstup znaku z klávesnice bez echa
* Výstupné parametre:
  + AL – ASCII kód zadaného znaku

### WriteChar

* Výpis znaku na obrazovku
* Vstupné parametre:
  + AL – ASCII kód znaku

### ReadString

* Číta znaky zadávané z klávesnice, kým sa nestlačí enter
* Za posledný znak vloží nulu
* Výstupné parametre:
  + EDX – adresa poľa bajtov, kam sa budú ukladať zadané znaky
  + ECX – max. dĺžka reťazca + 1
* Výstupné parametre
  + EAX – počet zadaných znakov

### WriteString

* Výpis reťazca na obrazovku
* Reťazec musí byť ukončený nulou
* Vstupné parametre:
  + EDX – adresa reťazca

### ReadInt

* Načíta celé číslo so znamienkom
* Výstupné parametre:
  + EAX – načítané číslo

### WriteInt

* Vypíše celé číslo s znamienkom
* Vstupné parametre
  + EAX – číslo

### ReadFloat

* Načíta celé číslo so znamienkom z konzoly
* Výstupné parametre:
  + Načítané číslo pridá do zásobníka FPU

### WriteFloat

* Vypíše celé číslo s znamienkom
* Vstupné parametre
  + st(0) – číslo

## Procedúry na náhodné čísla

### Randomize

* Nastaví generovanie náhodných čísel

### RandomRange

* Generuje náhodné číslo <0; n-1>
* Vstupné parametre
  + EAX – n
* Výstupné parametre
  + EAX - vygenerované číslo

## Makrá

* INCLUDE Macros.inc

### mWrite „TEXT“

* Vypíše TEXT do konzoly

### mWriteln „TEXT“

* Vypíše TEXT do konzoly + nový riadok

### mWriteSpace COUNT

* Vypíše COUNT medzier do konzoly