Skúška – Strojovo Orientované Jazyky

1. Assembler 8086

- **Casti operačného systému z pohľadu používateľa:**

- **Zivotný Cyklus programu:**
  - Specifikacia problemu, navrh programu
  - Potrebujeme editor na editaciu zdrojoveho kodu
  - Potrebujeme komplilátor / prekladač:
    - Robí analýzu zdrojového textu
      - lexikálna analýza
      - syntaktická analýza
      - sémantická analýza
    - Syntéza – Interpretácia
  - Napiseme zdroj (.asm), skompilujeme -> (.obj), zlinkujeme -> (.exe / .com)

- **Co je to assemblér, jeho funkcie a činnosť:**
  - Je to prekladač z JSI do priameho kódu procesora. Je strojovo závislý. Vytvára spustiteľné súbory zo zdrojového kódu.

- **Segment v JSI, definícia, použitie**
  - Je časť kódu obsahujúca bud kod samotný, alebo dáta, alebo zásobník
  - syntax:

```
[meno] SEGMENT [umiestnenie] [spajanie] [velkost] [trieda]
    ...
[meno] ENDS
```

    - **Umiestnenie –** určuje ako bude vyzerať segmentová adresa
      - **BYTE** – segment sa začne na lubovolnej adrese
      - **WORD** – segment sa zavedie na adrese deliteľnej 2
      - **DWORD** – Segment sa zavedie na adrese delitelnej 4
      - **PARA** – Segment sa zavedie na adrese deliteľnej 16 – **default**
      - **PAGE** – Segment sa zavedie na adresu, ktorá je deliteľná 256
    - **Spájanie** – určuje ako bude segment kombinovaný so segmentami iných modulov
      - **PUBLIC** – **default;** všetky segmenty, ktoré majú rovnakéí meno, budú spojené do jedného segmentu
      - **COMMON** – Na začiatku sa segmenty prikryjú, výsledný je tak veľký ako najväčší z nich
      - **STACK** – prekryjú sa na dne, ale ich veľkosť bude súčet všetkých segmentov
    - **Veľkosť** – veľkosť segmentu. Môžme použiť iba ak direktívou **.386** povolíme 80386 procesor
      - **USE16** – segment môže obsahovať 64KB kódu akebo dát
      - **USE32** – segment može obsahovať až 4GB kódu / dát
    - **Trieda** – Používa sa na identifikáciu segmenta, ktorý sa má umiestniť do rovnakého pamäťového miesta ako ostatné segmenty tej istej triedy.

- **Strukúra v JSI, definícia, použitie**

```
[meno] STRUC
   ...
   prvky struktury
   ...
[meno] ENDS
```

  - Struktura je zlozena z jedneho / viac prvkov. Je to zlozeny datovy typ. Kazdu polozku vieme samostatne adresovat

- **Zaznam v JSI, definicia, pouzitie**

```
[meno] RECORD prvok1, prvok2, prvok3, ...
```

priklad:

```
flag RECORD A:=1, B:1=1, C:4
```

- Prvky pola v datovom type RECORD su zlozene z bitov. Pouziva sa tam, kde chceme setrit pamatou. Priklad inicializacie:

```
flag   RECORD A:1, B:1, C:1, D:1, E:1, F:1
znak   flag <0,0,0,0,0,0>
znak1  flag {a=1,f=0}
```

- **Macro v JSI, definicia, pouzitie**

```
[meno] MACRO [parameter1] [parameter2] ...
  ; instrukcie tvoriace makro
ENDM
```

  - Macro pod svojim menom zoskupuje skupinu prikazov, ktore sa maju vykonat. Je to nieco ako substitucia

- **Operatory v JSI, definicia, pouzitie**
  - Aritmeticke: **+, - *, /, MOD, SHL, SHR**
  - Relacne: **EQ, NE, LT, LE, GT, GE**
  - Logicke: **AND, OR, XOR, NOT**

  - Podla priority:
    1. **LENGTH, SIZE, WIDTH, MASK, (), [], <>**
    2. strukturovane premenne
    3. explicitne vyjadrenie segmentu: **mov ax, ds:p1**
    4. **PTR, OFFSET, SEGMENT, TYPE**
    5. **HIGH, LOW**
    6. unarne operacie: +, -
    7. ***, /, MOD, SHL, SHR**
    8. binarne +, -
    9. **EQ, NE, LT, LE, GT, GE**
    10. **NOT**
    11. **AND**
    12. **OR, XOR**
    13. **SHORT**

- **Procedury v JSI, definicia, pouzitie, sposoby odovzdavania parametrov**
  - Je to podprogram, tj. program, ktory vykonava istu specificku cinnost vramci hlavneho programu

```
meno PROC [jazyk] [typ] USES
[USES items], [argument1] [argument2] ...
RETURNS [argument1] [argument2] ...
     ; tu sa nachadza telo procedury
RET
meno ENDP
```

  - Klucove slovo **RETURNS** sa pouziva na vracanie premennych, resp na ich odstranenie zo stacku. Pomocou **RETURNS** procedura vracia paramerte

2. Strojova uroven

- **Procesor klasickeho pocitaca, casti, zakladna cinnost**
  - Procesor sa stara o dekodovanie a vykonavanie instrukcii.
    1. Vyber instrukcie
    2. Dekodovanie instrukcie
    3. Vykonanie instrukcie
  - Procesor = CPU. Sklada sa z Aritmeticko-Logickej Jednotky, a Ovladacej (Control) jednotky

- **Technicke prostriedky procesora 8086 z pohladu programatora**
  - Procesor 8086 ma 14 16-bitovych registrov, je schopny adresovat 220 pamatovych miest -> 1MB pamate.
  - Registre si mozno predstavit ako specialne pamatove miest ulozene priamo v procesore. Su extremne rychle v porovnami s pamatou. Rozdelujeme ich podla pouzitia na:

1. Vseobecne registre
2. Registre na specialne ucely
3. Segmentove registre
4. Riadiace registre

- **Vseobecne registre**
  - **AX –** accumulator.
  - **BX –** base. (pri nepriamom adresovani)
  - **CX** – counter. Pouziva sa ako pocitadlo pri cykloch
  - **DX** – data. Sluzi na uchovavanie dat.

- **Registre na specialne ucely**
  - **SI –** source index. Vyuziva sa pri registrovom adresovani
  - **DI** – destination index. (ako SI)
  - **BP** – base pointer. Adresacia parametrov a lokalnych premennych na zasobniku
  - **SP** – stack pointer. Ukazuje na OFFSET zasobnika.

- **Segmentove registre**
  - **CS –** code segment. Segmentova cast adresy, kde je ulozeny program, ktory sa bude vykonavat
  - **DS** – data segment. Segment urcuje blok pamati, kde sa nachadzaju data programu
  - **ES** – extra segment. Pomocny segmentovy register
  - **SS** – stack segment. Segmentova cast adresy zasobnika. SS:SP davaju celu adresu.

- **Riadiace registre**
  - **IP** – instruction pointer. Ukazuje na instrukciu, ktora sa prave vykonava (na jej offsetovu adresu)\
  - **F** – flag register. Flags: **OF, DF, IF, TF, SF, ZF, AF, PF, CF**

- **<u>Instrukcia, jej casti, principy adresovania operandov</u>**
  - Stavba strojovej instrukcie:

OK	Adresa

    OK: Operacny Kod
    Adresa: Adresova cast instrukcie obsahuje predpis, ktory procesoru hovori, kde sa nachadzaju operandy. (zaroven moze byt sucasne operandom)
  - Principy adresovania:
    Procesor 8086 ma segmentovu architekturu pamate. Je schopny adresovat 1MB pamate (sirka adresovej zbernice je 20 bitov). Logicka adresa sa sklada z dvoch 16-bitovych hodnot: z adresy segmentu a adresy offsetu. Z takejto logickej adresy sa vytvara fyzicka 20-bitova adresa tak, ze segmentova cast sa posunie o 4 bity vlavoa k takto vzniknutemu cislu sa pripocita offset.

- **<u>Adresovanie operandov v system 8086</u>**
  - Priame adresovanie:
    Operand instrukcie tvori priamo offsetova adresa, napr:
    - `mov ax, [00FFH]`
    - `farba db 7`

      ```
      ...
      start: mov al, [farba]
      ```
  - Nepriame adresovanie:
    Offsetova cast adresy je ulozena v niektorom s registrov SI, DI, BP, BX. Napr:
    `mov bx, [di]`
  - Bazove adresovanie
    Na vytvorenie offsetovej adresy sa vyuzivaju bazove registre BX, BP a konstanta. Offsetova adresa je dana suctom bazoveho registra a konstanty. Pouziva sa pri adresovanie pevnej datovej struktury. Napr:
    `mov ax, [bx+n]`
    `mov ax, [bx+4]`
  - Indexove adresovanie
    Podobne ako bazove adresovanie, ale namiesto bazovych registrov sa pouzivaju indexove registre DI, SI. Pouziva sa pri datovych strukturach ako pole, string... napr:
    `mov ax, [pole+si]`
    `mov ax, [pole+di]`
  - Bazove – indexove adresovanie
    Kombinacia predoslych 2 sposobov adresovania. Offsetova cast je tvorena suctom obsahu bazoveho registra (BX, BP), indexoveho registra (SI, DI), a konstanty. Pouziva sa na adresovanie dyn. pola.

- **Format instrukcie 8086**

- **Typy strojovych operacii:**
  - Vseobecne typy operacii:
    - Prenosove
    - vypoctove
    - Skokove
    - Riadiace
  - Operacie procesoru 8086:
    - Presunove (mov)
    - Aritmeticke (add, sub, div, mul)
    - Bitove – Logicke (not, or, xor, and)
    - Retazcove (movsb, lodsb, stosb)
    - Instrukcie na riadenie vnutorneho stavu CPU (clc, stc, cli)
    - Instrukcie na riadenie behu programu (skokove – jmp, call, ret, ...)

- **Mechanizmus prerusenia a jeho obsluhy v systeme 8086**
  - Procesor dostane signal INTR
    - Procesor vzdy dokonci prave vykonavanu instrukciu.
  - Procesor posle INTA (interrupt acknowledge). Tymto povie periferii, ze akceptuje prerusenie, a periferia potom po udajovej zbernici posle *N*, kde mu urcuje typ prerusenia.
  - Procesor do stacku ulozi F, IP, CS registre
  - IF (interrupt flag) a TF (trap flag) nastavi na 0 -> tym si zakaze akekolvek dalsie prerusenie
  - IP := (4 * N); CS := (4 * N + 2) -> do IP a CS registru ulozi adresy pomocou *N* (typ prerusenia) a tabulky
  - Podla typu prerusenia najde obsluzny program, ktory ho ma vykonvat, a spusti ho
  - IRET -> tymto sa ukoncuje procedura, ktora bola spustena prerusenim. IRET vracia aj F zo stacku.

- **Procesor 80286 a sposob adresovania operandov**
  - 80286 ma 24-bitovu adresu, a vo virtualnom adresovacom mode s ochranou sa da dosiahnut az 16MB adresovatelnych miest.
  - Ma nove priznaky: NT -> nested task, I/O PL -> I/O privilegium level,
  - Obrazok:

    AU zaistuje prevaznu cast funkcii suvisiacich s virtualnou pamatou. Obsahuje sadu registrov predstavujucich cache.
    BU sluzi na prenos dat medzi procesorom a okolim. Obsahuje frontu na 6 Byteov instrukcii.
    EU je jednotka, ktora prevadza samotne vyupocty
    IU dekoduje instrukcie a obsahuje frontu 3 dekodovanych instrukcii.
  - Ma novy register: TR -> task register. Pri vykonavani viacerych uloh je stavulohy opisany v TDR (task descriptor register). LDTR -> local descriptor table register. (z neho sa dozvieme bazu pre aktualnu ulohu). GDTR -> global descriptor table register, IDTR -> ?
  - MSW -> Machine status word. Obsahuje TS (task switch) a PE (protection enable).
  - Mod realnej pamate:
    - Urceny pre priame vykonavanie programov opisanych a prelozenych pre 8086. V tomto mode sa pracuje bez virtualnej pamate a bez funkcii ochrany pamate, ale podstatne rychlejsie ako starsie mikroprocesory.
  - Mod chranenej virtualnej pamate:
    - Tu je zaistena zlucitelnost zdola s 8086/88, avsak na urovni zdrojoveho kodu. Plne sa tu vyuzivaju moznosti novej architektury. V chranemom mode sa da linearnymi adresami dosiahnut 16MB. Je umoznene vyuzivanie virtualnej pamate a spustanie programov zostrojenych pre lubovolny nizsi procesor (8086). Pri praci v chranemom rezime moze napr. prevadzat specialne privilegovane instrukcie podporujuce multitaskovy rezim a ochranu dat pred neopravnenymi zasahmi. Zakladna architektura zostava rovnaka ako v realnom rezime. Hlavne rozdiely medzi chranenym a realnym modom spocivaju

vo velmi podstatom zvacseni adresovatelnej pamate a v rozdielnych mechanizmoch prevodu logickych adries na adresy linearne a fyzicke. Vyuziva selektor + tabulku deskriptorov.

- **Procesor 80386 a sposob adresovania operandov**
  - 32b adresova zbernica
  - 32b udajova zbernica
  - ako prvy riesi strankovanie pamate
  - 32 bitove registre: EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP, EF, EIP
  - novy register DB -> debug register
  - Princip cinnosti v rezime strankovania:
    - Strankovacia jednotka (PU), ktora je obsiahnuta na chipe mikroprocesora je urcena pre preklad virtualnych adries na realne. Napr. pouziva strankovany segmentovy system, v ktorom je pri preklade virtualnej adresy najskor vypocitana adresa segmentu a az pomocou hornych 10 bitov adresy je adresovatelna tabulka zvana zoznam stranok. V tejto tabulke je najdena zodpovedajuca adresa?. Adresa najdena v tabulke stanok sa potom preradi posunutim a tym vznikne realna adresa.
  - Obrazok:

    - BU: base, EUL: execution, IU: instruction, SU+PU: memory management, BU: adr+udaj+k-ty.bit