

```

hexa  proc          ; ax kiírása hexadecimálisan
; legyen a példa kedvéért:  ax = 1234H
      xchg  ah,al    ; ah és al felcserélése
; most:
      ax = 3412H, al = 12H
      call   hexa_b  ; al (az eredeti ah) kiírása
; kiírtuk, hogy 12
      xchg  ah,al    ; ah és al visszacserélése
; most újra:
      ax = 1234H, al = 34H
      call   hexa_b  ; al kiírása
; most kiírtuk, hogy 34, tehát eddig kiírtuk, hogy 1234
      ret          ; visszatérés
hexa  endp         ; a hexa eljárás vége

```

Máté: Assembly programozás 6. előadás 1

```

hexa_b proc        ; al kiírása hexadecimálisan
; az első híváskor:    al = 12H
      push  cx       ; mentés a verembe
      mov   cl,4     ; 4 bit-es rotálás előkészítése
      ROR   al,CL    ; az első jegy az alsó 4 biten
; most:
      al = 21H
      call   h_jegy  ; az első jegy kiírása
; kiírtuk, hogy 1
      ROR   al,CL    ; a 2. jegy az alsó 4 biten
; most újra:
      al = 12H
      call   h_jegy  ; a második jegy kiírása
; most kiírtuk, hogy 2, tehát eddig kiírtuk, hogy 12
      pop   cx       ; visszamentés a veremből
      ret          ; visszatérés
hexa_b endp        ; a hexa_b eljárás vége

```

Máté: Assembly programozás 6. előadás 2

```

h_jegy proc        ; hexadecimális jegy kiírása
      push  ax       ; mentés a verembe
      AND   al,0FH   ; a felső 4 bit 0 lesz,
                        ; a többi változatlan
      add   al,'0'   ; + 0 kódja
      cmp   al,'9'   ; ≤ 9 ?
      jle   h_jegy1 ; ugrás h_jegy1 -hez, ha igen
      add   al,'A'-0AH-'0'; A...F hexadecimális
                        ; jegyek kialakítása
h_jegy1: mov   ah,14 ; BIOS hívás:
      int   10H    ; karakter kiírás
      pop   ax     ; visszamentés a veremből
      ret          ; visszatérés
h_jegy endp        ; a h_jegy eljárás vége

```

Máté: Assembly programozás 6. előadás 3

Processzor vezérlő utasítások I.

A processzor állapotát módosít(hat)ják.

| A STATUS bitjeinek módosítása | | | |
|-------------------------------|------------|------------|------------|
| Flag | CLear | SeT | CoMplement |
| C | CLC | STC | CMC |
| D | CLD | STD | |
| I | CLI | STI | |

Máté: Assembly programozás 6. előadás 4

Processzor vezérlő utasítások II.

NOP ; **NO** oPeration, üres utasítás,
; nem végez műveletet.

WAIT; A processzor várakozik, amíg más
; processzortól (pl. lebegőpontos
; segédprocesszortól) kész jelzést nem kap.

HLT ; **HaLT**, leállítja a processzort.
; A processzor külső megszakításig
; várakozik.

Máté: Assembly programozás 6. előadás 5

Input, output (I/O) utasítások (I8086/88)

A külvilággal történő információ csere **port**-okon (kapukon) keresztül zajlik. A kapu egy memória cím, az információ csere erre a címre történő írással, vagy erről a címről való olvasással történik. Egy-egy cím vagy cím csoport egy-egy perifériához kötődik. A központi egység oldaláról a folyamat egységesen az **IN** (input) és az **OUT** (output) utasítással történik.

Máté: Assembly programozás 6. előadás 6

Input, output (I/O) utasítások (I8086/88)

A perifériától függ, hogy a hozzá tartozó port 8 vagy 16 bites. A központi egységnek az **AL** illetve **AX** regisztere vesz részt a kommunikációban. A port címzése 8 bites közvetlen adattal vagy a **DX** regiszterrel történik.

IN AL/AX, port

; **AL/AX** \leftarrow egy byte/word a port-ról

OUT port, AL/AX

; **port** \leftarrow egy byte/word **AL/AX**-ből

Máté: Assembly programozás

6. előadás

7

A periféria oldaláról a helyzet nem ilyen egyszerű. Az input információ „**csomagokban**” érkezik, az output információt „**csomagolva**” kell küldeni. A csomagolás (vezérlő információ) mondja meg, hogy hogyan kell kezelni a csomagba rejtett információt (adatot). Éppen ezért az operációs rendszerek olyan egyszerűen használható eljárásokat tartalmaznak (**BIOS** – Basic Input/Output System – rutinok, stb.), amelyek elvégzik a ki- és becsomagolás munkáját, és ezáltal lényegesen megkönnyítik a külvilággal való kommunikációt.

Máté: Assembly programozás

6. előadás

8

Megszakítás rendszer, interrupt utasítások

- Az **I/O** utasítás lassú \leftrightarrow a **CPU** gyors, a **CPU** várakozni kényszerül
- **I/O** regiszter (**port**): a **port** és a központi egység közötti információ átadás gyors, a periféria autonóm módon elvégzi a feladatát. Újabb perifériához fordulás esetén a **CPU** várakozni kényszerülhet.
- **Pollozások technika** (~tevékeny várakozás): a futó program időről időre megkérdezi a periféria állapotát, és csak akkor ad ki újabb **I/O** utasítást, amikor a periféria már fogadni tudja. A hatékonyság az éppen futó programtól függ.

Máté: Assembly programozás

6. előadás

9

Megszakítás

A (program) megszakítás azt jelenti, hogy az éppen futó program végrehajtása átmenetileg megszakad – a processzor állapota megőrződik, hogy a program egy későbbi időpontban folytatódhassék – és a processzor egy másik program, az úgynevezett **megszakítás kezelő** végrehajtását kezdi meg.

Miután a megszakítás kezelő elvégezte munkáját, gondoskodik a processzor megszakításkori állapotának visszaállításáról, és visszaadja a vezérlést a megszakított programnak: **Átlátszóság**.

Máté: Assembly programozás

6. előadás

10

Pl.: nyomtatás

- Nyomtatás pufferbe, később a tényleges nyomtatást vezérlő program indítása.
- Nyomtatás előkészítése (a nyomtató megnyitása), **HLT**.
- A továbbiak során a nyomtató megszakítást okoz, ha kész újabb adat nyomtatására. Ilyenkor a **HLT** utasítást követő címre adódik a vezérlés. A következő karakter előkészítése nyomtatásra, **HLT**. A bekövetkező megszakítás hatására a megszakító rutin mindig a következő adatot nyomtatja. Ha nincs további nyomtatandó anyag, akkor a nyomtatást vezérlő program lezárja a nyomtatót (nem következik be újabb megszakítás a nyomtató miatt), és befejezi a működését.

Máté: Assembly programozás

6. előadás

11

A **HLT** utasítás csak akkor szükséges, ha a nyomtatást kérő program befejezte a munkáját. Ellenkező esetben visszakaphatja a vezérlést. Ilyenkor az ő feladata az esetleg szükséges várakozásról gondoskodni a program végén.

Bevitel esetén olyankor is várakozni kell, ha még a beolvasás nem történt meg, és a további futáshoz szükség van ezekre az adatokra.

Jobb megoldás, ha a **HLT** utasítás helyett az operációs rendszer fölfüggeszti a program működését, és elindítja egy másik program futását.

Ez vezetett a **multiprogramozás** kialakulásához.

Máté: Assembly programozás

6. előadás

12

A megszakítás kezelő (megszakító rutin) megszakítható-e? Gyors periféria kiszolgálása közben megszakítás kérés, ...
 „Alap” állapot – „megszakítási” állapot, megszakítási állapotban nem lehet újabb megszakítás.
Hierarchia: megszakítási állapotban csak magasabb szintű ok eredményezhet megszakítást.
 Bizonyos utasítások csak a központi egység bizonyos kitüntetett állapotában hajthatók végre, alap állapotban nem → csapda, szoftver megszakítás.
 Megoldható az operációs rendszer védelme, a tár védelem stb.
 A megoldás nem tökéletes: **vírus**.

Máté: Assembly programozás

6. előadás

13

I8086/88

Megszakítás kiszolgálásakor a **STATUS**, **CS** és **IP** a verembe kerül, az **I** és a **T** flag **0** értéket kap (az úgynevezett maszkolható megszakítások tiltása és folyamatos üzemmód beállítása), majd **(CS:IP)** felveszi a megszakítás kezelő kezdőcímét.

Átlátszóság: Amikor bekövetkezik egy megszakítás, akkor bizonyos utasítások végrehajtottak, de amikor ennek vége, a **CPU** ugyanolyan állapotba kerül, mint amilyenben a megszakítás bekövetkezése előtt volt.

Máté: Assembly programozás

6. előadás

14

Megszakítás és csapda

Megszakítás (interrupt): Olyan automatikus eljárás hívás, amit általában nem a futó program, hanem valamilyen **B/K** eszköz idéz elő, pl. a program utasítja a lemezegységet, hogy kezdje el az adatátvitelt, és annak végeztével megszakítást küldjön. **Megszakítás kezelő**.

Csapda (trap): A program által előidézett feltétel (pl. túlsordulás) hatására automatikus eljárás hívás.
Csapda kezelő.

A csapda a **programmal szinkronizált**, a megszakítás nem.

Máté: Assembly programozás

6. előadás

15

Interrupt (csapda) utasítások

Szoftver megszakítást (csapdát) eredményeznek.

INT i ; $0 \leq i \leq 255$,
 ; megszakítás az i . ok szerint.

Az **INT 3** utasítás kódja csak egy byte (a többi 2 byte), így különösen alkalmas nyomkövető (**DEBUG**) programokban történő alkalmazásra.

Máté: Assembly programozás

6. előadás

16

A **DEBUG** program saját magához irányítja a **3**-as megszakítást. Az ellenőrzendő program megadott pontján (törés pont, **break point**) lévő utasítást (annak 1. bájtyát) átmenetileg az **INT 3** utasításra cseréli, és átadhatja a vezérlést az ellenőrzendő programnak. Amikor a program az **INT 3** utasításhoz ér, a megszakítás hatására a **DEBUG** kapja meg a vezérlést. Kiírja a regiszterek tartalmát, és további információt kérhetünk a program állapotáról.
 Később visszaírja azt a tartalmat, amit **INT 3**-ra cserélt, elhelyezi az újabb törés pontra az **INT 3** utasítást és visszaadja a vezérlést az ellenőrzendő programnak.

Máté: Assembly programozás

6. előadás

17

Ezek alapján érthetővé válik a **DEBUG** program néhány „furcsasága”:

- Miért „felejt el” a töréspontot? Ha ugyanis nem felejtene el – azaz nem cserélné vissza a töréspontra elhelyezett utasítást az eredeti utasításra – akkor a program nyomkövetésében nem tudna továbblépni.
- Miért nem lehet egy ciklus futásait egyetlen töréspont elhelyezésével figyelgetni, stb?

Máté: Assembly programozás

6. előadás

18

INTO ; megszakítás csak **O=1 (Overflow)**
; esetén a **4.** ok szerint

Visszatérés a megszakító rutinból

IRET ; **IP, CS, STATUS** feltöltése a
; veremből

Máté: Assembly programozás

6. előadás

19

Szemafor

Legyen az **S** szemafor egy olyan word típusú változó, amely mindegyik program számára elérhető. Jelentse **S=0** azt, hogy az erőforrás szabad, és **S≠0** azt, hogy az erőforrás foglalt. Próbáljuk meg a szemafor kezelését!

; 1. kísérlet

```
ujra:  mov    cx,S
       jcxz  szabad
       ...   ; foglalt az erőforrás, várakozás
       jmp   újra
szabad: mov   cx,0FFFFh
       mov   S,cx ; a program lefoglalta az erőforrást
```

Máté: Assembly programozás

6. előadás

20

; 2. kísérlet

```
ujra:  MOV    CX,0FFFFH
       XCHG  CX,S    ; már foglaltat jelez a
                   ; szemafor!
       JCXZ  szabad ; ellenőrzés S korábbi tartalma
                   ; szerint.
       ...   ; foglalt az erőforrás,
                   ; várakozás
       jmp   újra
szabad: ...       ; szabad volt az erőforrás,
                   ; de a szemafor már foglalt
```

Máté: Assembly programozás

6. előadás

21

Az **XCHG** utasítás mikroprogram szinten:

Segéd regiszter \Leftarrow S; S \Leftarrow CX; CX \Leftarrow Segéd regiszter
olvasás – módosítás – visszairás

```
ujra:  mov    cx,0FFFFh
       LOCK  xchg  cx,S    ; S már foglaltat jelez
       jcxz  szabad ; ellenőrzés S korábbi
                   ; tartalma szerint
       ...   ; foglalt, várakozás
       jmp   újra
szabad: ...       ; használható az erőforrás,
                   ; de a szemafor már foglalt
       ...
       MOV   S,0    ; a szemafor szabadra állítása
```

Máté: Assembly programozás

6. előadás

22