Gyakorlatias assembly bevezető

Fordítóprogramok előadás (A, C, T szakirány)

Mi az assembly?

- programozási nyelvek egy csoportja
- gépközeli:
  - az adott processzor utasításai használhatóak
  - általában nincsenek programkonstrukciók, típusok, osztályok stb.
- a futtatható programban pontosan azok az utasítások lesznek, amit a programba írunk
  - lehet optimalizálni
  - lehet olyan trükköket használni, amit a magasabb szintű nyelvek nem engednek meg

Sokféle assembly van...

- különféle architektúrák (processzorok) utasításkészlete különbözik
  - van olyan architektúra, ahol gépi utasítás van külön egy bináris keresőfa kiegyensúlyozására
  - Intel architektúrán ilyen nincs...
- egy architektúrán belül is lehet különböző szintaxisú assembly nyelvet definiálni
  - NASM assembly: `mov eax,0`
  - GAS assembly: `movl $0, %eax`

Mit fogunk mi használni?

- Intel x86 architektúra (386, 486, Pentium, ...)
  - mindenki számára könnyen elérhető
- NASM assembly
  - tiszta, könnyen érthető a szintaxisa

Assembly programok fordítása

- az assembly programok fordítóprogramja az *assembler*
- a magas szintű nyelvek fordítóprogramjaihoz képest:
  - az elemzés része egyszerűbb
    - általában nincsenek típusok, bonyolult szintaktikus elemek stb.
  - a kódgenerálás bonyolultabb
    - az assembler gépi kódra fordít
    - a többi fordítóprogram általában assemblyre vagy egy másik magasszintű nyelvre

A NASM fordítóprogramja

A fordítás lépései

```
$ ls
io.c programom.asm
$ nasm -f elf -o programom.o programom.asm
$ ls
io.c programom.asm programom.o
$ gcc -o programom programom.o io.c
$ ls
io.c programom programom.asm programom.o
```

- `nasm`: fordítás (asszemblálás)
- `gcc`: szerkesztés (linkelés)
  (a *C* fordítóprogramját használjuk)
- `io.o`: egy object file, amit C-ben írtam és az ebben lévő függvényeket meghívjuk a majd programunkból

A nasm kapcsolói

Az asszemblálás

```
$ nasm -f elf -o programom.o programom.asm
```

- `-f elf`: megadjuk az object fájl formátumát (elf)
- `-o programom.o`: az eredményként keletkező object fájl neve
- egyéb kapcsolók: `nasm -h`

Az assembly program „változói"

- regiszterek
  - kis méretű tárolóhelyek a processzoron
  - külön nevük van: `eax`, `ebx`, `ecx`, ...
- memória
  - itt tárolhatók a program által használt adatok
  - itt tárolódik maga a program kódja is
  - címzéssel lehet hivatkozni rá: `[tomb+4]`

Regiszterek adattároláshoz, számoláshoz

- `eax`: *„accumulator"* - elsősorban aritmetikai számításokhoz
- `ebx`: *„base"* - ebben szokás tömbök, rekordok kezdőcímét tárolni
- `ecx`: *„counter"* - számlálókat szokás tárolni benne (pl. for jellegű ciklusokhoz)
- `edx`: *„data"* - egyéb adatok tárolása; aritmetikai számításokhoz segédregiszter
- `esi`: *„source index"* - sztringmásolásnál a forrás címe
- `edi`: *„destination index"* - sztringmásolásnál a cél címe

Ezek egymással felcserélhetők, de célszerű a megadott feladatokra használni őket, ha lehet.

Regiszterek a program vermének kezeléséhez

- `esp`: *„stack pointer"* - a program vermének a tetejét tartja nyilván
- `ebp`: *„base pointer"* - az aktuális alprogramhoz tartozó verem-részt tartja nyilván

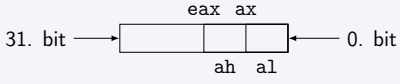
Ezeket csak a veremkezeléshez érdemes használni. Meggondolatlan elállításuk hibához vezethet!

Adminisztratív regiszterek

- `eip`: *„instruction pointer"* - a következő végrehajtandó utasítás címe
- `eflags`: *jelzőbitek* - pl. összehasonlítások eredményeinek tárolása

Ezeket közvetlenül nem tudjuk olvasni és írni.

A regiszterek felépítése



- `eax` 32 bitből áll; két része van:
  - a „felső" 16 bitnek nincs külön neve
  - az „alsó" 16 bit: `ax`; ennek részei:
    - a „felső" bájt: `ah`
    - az „alsó" bájt: `al`
- ugyanígy épülnek fel: `ebx`, `ecx`, `edx`
- `esi`, `edi`, `esp` és `ebp` regiszterekben csak az alsó 16 bitnek van külön neve: `si`, `di`, `sp`, `bp`

Az adatterület megadása

- a programban helyet foglalunk az adatainak:
  - ha csak helyet akarunk foglalni nekik: `.bss` szakasz
  - ha kezdeti értéket is akarunk adni: `.data` szakasz

Adatterület megadása

```
section .bss
A: resb 4       ; négy bájt lefoglalása
B: resb 2       ; két bájt lefoglalása

section .data
C: db 1,2,3,4   ; négyszer 1 bájt
                ; az 1, 2, 3, 4 értékekkel
D: dd 42        ; egyszer 4 bájt a 42 értékkel
```


Az adatterület megadása

Adatterület megadása

```
section .bss
A: resb 4       ; négy bájt lefoglalása
B: resb 2       ; két bájt lefoglalása

section .data
C: db 1,2,3,4   ; négyszer 1 bájt
                ; az 1, 2, 3, 4 értékekkel
D: dd 42        ; egyszer 4 bájt a 42 értékkel
```

A	A+1	A+2	A+3	B	B+1
?	?	?	?	?	?

C	C+1	C+2	C+3	D	D+1	D+2	D+3
1	2	3	4	42	0	0	0

Adatterület megadása

- `resb`: *„reserve byte”* - egy bájt
- `resw`: *„reserve word”* - két bájt (egy gépi szó)
- `resd`: *„reserve double word”* - négy bájt (egy gépi duplaszó)
- `db`: *„define byte”* - egy bájt kezdőértékkel
- `dw`: *„define word”* - két bájt kezdőértékkel
- `dd`: *„define double word”* - négy bájt kezdőértékkel

Memóriahivatkozás

C	C+1	C+2	C+3	D	D+1	D+2	D+3
1	2	3	4	42	0	0	0

- `byte [C]`: 1 bájt a C címtől
  értéke: 1
- `byte [C+1]`: 1 bájt a C+1 címtől
  értéke: 2
- `word [C+1]`: 2 bájt a C+1 címtől
  értéke: $256^1 \cdot 3 + 256^0 \cdot 2 = 770$
- `dword [D]`: 4 bájt a D címtől
  értéke: $256^3 \cdot 0 + 256^2 \cdot 0 + 256^1 \cdot 0 + 256^0 \cdot 42 = 42$

Adatmozgatás: `mov`

- `mov eax,ebx`
  `ebx` értékét `eax`-be tölti
- `mov eax,dword [D]`
  a `D` címtől kezdődő 4 bájt értékét `eax`-be tölti
  (a `dword` itt el is hagyható, mert `eax`-ből kiderül, hogy 4 bájtot kell mozgatni)
- `mov dword [D],eax`
  `eax` értékét a `D`-től kezdődő 4 bájtba tölti
- `mov al,bh`
  `bh` értékét `al`-be tölti
- `mov byte [C],al`
  `al` értékét a `C` című bájtba tölti
- mov byte [C],byte [D]
  Hibás! Nincs memóriából memóriába mozgatás.

Aritmetikai műveletek: `inc`, `dec`, `add`, `sub`

- `inc eax`
  az `eax` értékét megnöveli eggyel
  (lehet memóriahivatkozás is az operandus)
- `dec word [C]`
  a `C` címtől kezdődő 2 bájt értékét csökkenti eggyel
  (lehet regiszter is az operandus)
- `add eax,dword [D]`
  `eax`-hez adja a `D` címtől kezdődő 4 bájt értékét
- `sub edx,ecx`
  levonja `edx`-ből `ecx`-et
- Az `add` és a `sub` utasításokra ugyanaz a szabály, mint a `mov`-ra: legfeljebb az egyik operandus lehet memóriahivatkozás.

Aritmetikai műveletek: `mul`, `div`

- `mul ebx`
  Megszorozza `eax` értékét `ebx` értékével.
  Az eredmény:
  - `edx`-be kerülnek a nagy helyiértékű bájtok
  - `eax`-be az alacsony helyiértékűek
- `div ebx`
  Elosztja a 256^4·`edx`+`eax` értéket `ebx` értékével.
  Az eredmény:
  - `eax`-be kerül a hányados
  - `edx`-be a maradék
- Ha előjeles egész számokkal dolgozunk, akkor az `imul` és `idiv` utasításokat kell használni.

Logikai műveletek: `and`, `or`, `xor`, `not`

- `and al,bl`
  Bitenkénti „és” művelet.

Bitenkénti "és"

Ha előtte `al`$= 12 = 0000\,1100_2$ és `bl`$= 6 = 0000\,0110_2$,
akkor utána `al`$= 0000\,0100_2 = 4$.

- `or eax,dword [C]`
  Bitenkénti „vagy” művelet.
- `xor word [D],bx`
  Bitenkénti „kizáró vagy” művelet.
- `not bl`
  Bitenkénti „nem” művelet.

Bitenkénti "nem"

Ha előtte `bl`$= 9 = 0000\,1001_2$,
akkor utána `bl`$= 1111\,0110_2 = 246$.

Igaz, hamis értékek

A magasszintű programozási nyelvek *logikai* (bool) típusának egy lehetséges megvalósítása:

- 1 bájton tároljuk
- *hamis* érték: 0
- *igaz* érték: 1
- logikai *és*: `and` utasítás
- logikai *vagy*: `or` utasítás
- *tagadás*: `not` és `and 1`

Ugró utasítás: `jmp`

Végtelen ciklus ugró utasítással

```
eleje: inc ecx
       jmp eleje
```

Utasítások átugrása

```
       mov ecx,0
       jmp vege
       add ecx,ebx
       inc ecx
vege:  mov edx,ecx
```


Feltételes ugró utasítások

- cmp: „compare” - két érték összehasonlítása
- feltételes ugró utasítások: „ugorj a megadott címkére, ha . . . ”

Ha eax=0, ugorj az A címkére!

```
cmp eax,0
je A
```

Ha eax≠0, ugorj az A címkére!

```
cmp eax,0
jne A
```

Ha eax<ebx, ugorj az A címkére!

```
cmp eax,ebx
jb A
```

Ha eax>ebx, ugorj az A címkére!

```
cmp eax,ebx
ja A
```

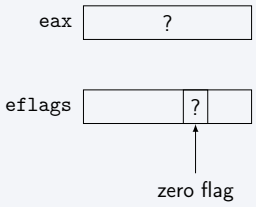

Feltételes ugró utasítások

- `je`: *„equal”* - ugorj, ha egyenlő
- `jne`: *„not equal”* - ugorj, ha nem egyenlő
- `jb`: *„below”* - ugorj, ha kisebb
  ≡ `jnae`: *„not above or equal”* - nem nagyobb egyenlő
- `ja`: *„above”* - ugorj, ha nagyobb
  ≡ `jnbe`: *„not below or equal”* - nem kisebb egyenlő
- `jnb`: *„not below”* - nem kisebb
  ≡ `jae`: *„above or equal”* - nagyobb egyenlő
- `jna`: *„not above”* - nem nagyobb
  ≡ `jbe`: *„below or equal”* - kisebb egyenlő
- Ha előjeles egészekkel számolunk:
  `jl` (*„less”*), `jg` (*„greater”*), `jnl`, `jng`, `jle`, `jge`, . . .

Feltételes ugró utasítások működése

```
mov eax,10
cmp eax,10
je A
```

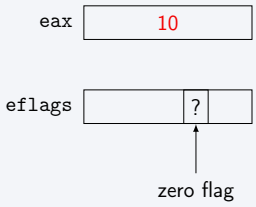
- Az eax regiszter értéke 10 lesz.
- Mivel eax=10, az összehasonlítás beállítja a *zero flag*-et 1-re.
- A je akkor ugrik, ha a *zero flag*=1.



Feltételes ugró utasítások működése

```
mov eax,10
cmp eax,10
je A
```

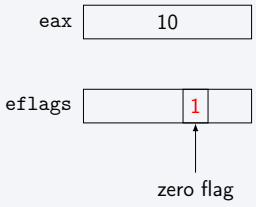
- Az eax regiszter értéke 10 lesz.
- Mivel eax=10, az összehasonlítás beállítja a *zero flag*-et 1-re.
- A je akkor ugrik, ha a *zero flag*=1.



Feltételes ugró utasítások működése

```
mov eax,10
cmp eax,10
je A
```

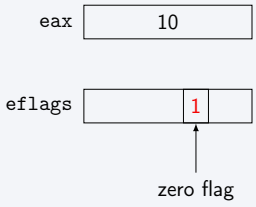
- Az eax regiszter értéke 10 lesz.
- Mivel eax=10, az összehasonlítás beállítja a *zero flag*-et 1-re.
- A je akkor ugrik, ha a *zero flag*=1.



Feltételes ugró utasítások működése

```
mov eax,10
cmp eax,10
je A
```

- Az eax regiszter értéke 10 lesz.
- Mivel eax=10, az összehasonlítás beállítja a *zero flag*-et 1-re.
- A je akkor ugrik, ha a *zero flag*=1.



Feltételes elágazás

Ha eax<10, akkor eax:=eax+1.

```
            cmp eax,10
            jnb vege
            inc eax
    vege:
```

Ha eax<10, akkor eax:=eax+1, különben eax:=eax-1.

```
             cmp eax,10
             jnb hamis_ag
             inc eax
             jmp vege
    hamis_ag: dec eax
    vege:
```


Ciklus

Amíg eax<10, eax:=eax+1.

```
    eleje:    cmp eax,10
              jnb vege
              inc eax
              jmp eleje
    vege:
```

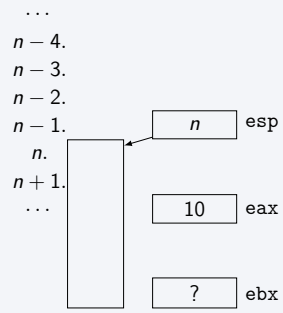
Verem

- Minden futó programhoz hozzá van rendelve egy saját memóriaterület, a program *verme*.
- Ebben lehet tárolni a lokális változókat.
- Ebben adjuk át a paramétereket alprogramhívás esetén.
- Ebbe kerül bele a *visszatérési cím*, azaz, hogy hova kell visszatérni az alprogram végén.

Veremműveletek: push, pop

```
push eax
pop ebx
```

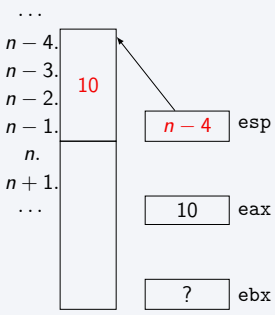
- eax értéke bekerül a verembe (4 bájt)
- a verem tetején lévő (4 bájtos) érték bemásolódik ebx-be, a veremmutató visszaáll



Veremműveletek: push, pop

```
push eax
pop ebx
```

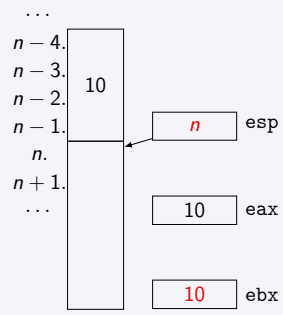
- eax értéke bekerül a verembe (4 bájt)
- a verem tetején lévő (4 bájtos) érték bemásolódik ebx-be, a veremmutató visszaáll



Veremműveletek: push, pop

```
push eax
pop ebx
```

- eax értéke bekerül a verembe (4 bájt)
- a verem tetején lévő (4 bájtos) érték bemásolódik ebx-be, a veremmutató visszaáll



Veremműveletek

- Csak 2 vagy 4 bájtot lehet a verembe tenni (vagy kivenni).
  - Tehát pl. `push ah` hibás!
- A verem tetején lévő adat pop művelet esetén nem törlődik a memóriából, csak lemásolódik, és a veremmutató megváltozik.
- Ha csak a veremmutató visszaállítása a cél, akkor lehet
  `pop ebx`
  helyett
  `add esp,4`
  utasítást használni.
  - Ez visszaállítja a veremmutatót, de nem másolja sehova a verem tetején lévő értéket.

Alprogramhívás: call

A `kiir` eljárás hívása

```
push eax   ; Betesszük a verembe a paramétert.
call kiir  ; Meghívjuk az eljárást.
           ; Az eljárás a veremből használhatja
           ; az átadott paramétert.
add esp,4  ; Visszaállítjuk a veremmutatót
           ; a push előtti állapotba.
```

Alprogramhívási konvenciók

- Szabályozni kell, hogy a *hívó* és a *hívott* kódrészlet pontosan hogyan kommunikál egymással:
  - Milyen sorrendben kerülnek a verembe a paraméterek?
  - Kiszedi-e az alprogram a paramétereket a veremből, vagy ez a hívó kódrészlet dolga?
  - Hol kapja meg a hívó a függvények visszatérési értékét?
  - Melyik regisztereket változtathatja meg az alprogram?
- A C nyelv hívási konvenciói:
  - A paramétereket fordított sorrendben kell a verembe tenni, azaz az első paraméter lesz legfelül.
  - Az alprogram bennt hagyja a paramétereket a veremben.
  - A visszatérési érték az `eax` regiszterbe kerül.
  - Az alprogram csak az `eax`, `edx`, `ecx` regisztereket módosíthatja. (*Intel ABI* konvenció.)