



HÁLÓZATI RENDSZEREK
ÉS SZOLGÁLTATÁSOK
TANSZÉK



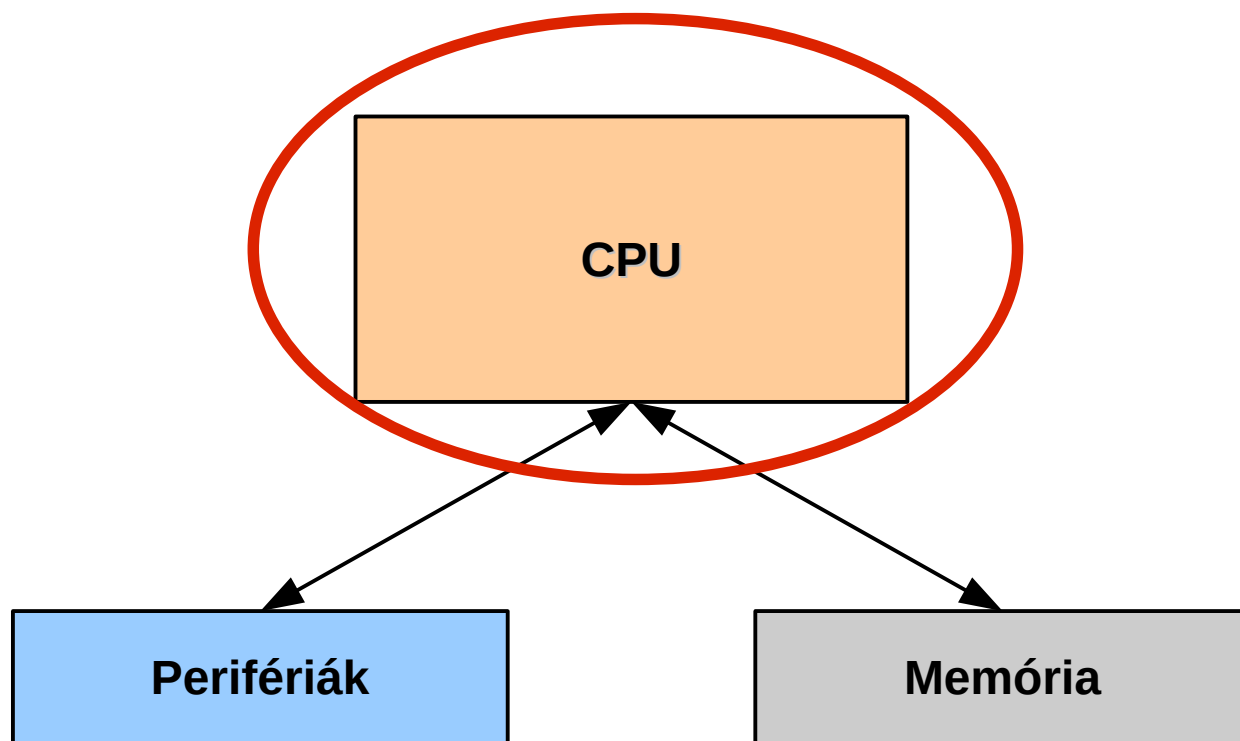
Budapest,
2025.04.01.

SZÁMÍTÓGÉP ARCHITEKTÚRÁK

Utasításkészlet architektúrák

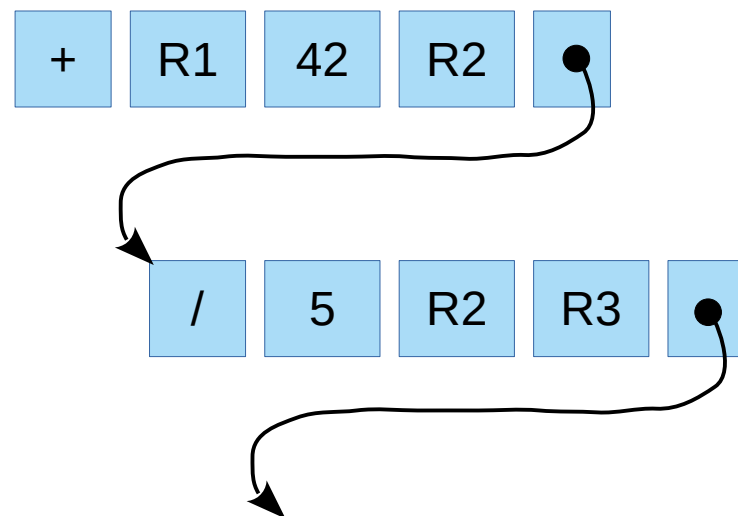
Horváth Gábor, Belső Zoltán

BME Hálózati Rendszerek és Szolgáltatások Tanszék
ghorvath@hit.bme.hu, belso@hit.bme.hu



- Processzorok jellemzője:
Programozási felület / Utasításkészlet architektúra
- Részei:
 - Utasítások
 - Támogatott adattípusok
 - Regiszterek
 - Címzési módok
 - Jelzőbitek
 - Perifériakezelési mód
 - Megszakítás- és kivételkezelés

- Utasítások felépítése:
 - Utasítás kódja / művelet típusa
 - Operandusok címei / értékei
 - Eredmény címe
 - Következő utasítás címe
- Takarékoság:
 - Köv. utasítás címe elhagyható
 - Operandusok száma:
 - 3 operandusú műveletek: **$R1 \leftarrow R2 + R3$**
 - 2 operandusú műveletek: **$R1 \leftarrow R1 + R2$**
 - 1 operandusú műveletek: **$ADD\ R1$**



- Utasítások fajtái:

- Adatmozgatás

$R1 \leftarrow R2, R1 \leftarrow \text{MEM}[100], R1 \leftarrow 42, \text{MEM}[100] \leftarrow R1, \text{MEM}[100] \leftarrow 42$

- Aritmetikai és logikai

$R1 \leftarrow R2+R3, R1 \leftarrow \text{MEM}[100]*42, \text{MEM}[100] \leftarrow R1 \ \& \ R2$

- Vezérlésátadó:

$\text{JUMP } -42, \text{JUMP } +28 \text{ IF } R1==R2, \text{CALL proc}, \text{RETURN}$

- Veremkezelő

$\text{PUSH } R1, \text{PUSH } 42, R2 \leftarrow \text{POP}$

- I/O műveletek

$\text{IO}[42] \leftarrow R1, R1 \leftarrow \text{IO}[42]$

- Transzcendens függvények

$R2 \leftarrow \text{SIN } R1, R2 \leftarrow \text{SQRT } 42$

- Egyebek

- Címzési mód megadja, hogy hol az operandus
- Ahol lehet:
 - Magában az utasításkódban
 - Regiszterben
 - Memóriában

Címzési mód	Példa
Regiszter	$R1 \leftarrow R2 + R3$
Közvetlen konstans	$R1 \leftarrow R2 + 42$
Direkt	$R1 \leftarrow R2 + \text{MEM}[42]$
Regiszter indirekt	$R1 \leftarrow R2 + \text{MEM}[R3]$
Eltolt indirekt	$R1 \leftarrow R2 + \text{MEM}[R3+42]$
Memória indirekt	$R1 \leftarrow R2 + \text{MEM}[\text{MEM}[R3]]$
Indexelt	$R1 \leftarrow R2 + \text{MEM}[R3+R4]$

- Célszerűen relatív címzés, pl. JUMP -28
- Feltételes ugrás 3 jellemző megvalósítása:

- Feltétel kódokkal:

COMPARE R1, R2

JUMP label IF GREATER

- Feltétel regiszterekkel:

R0 ← R1 > R2

JUMP label IF R0

- „Összehasonlít és ugrik”:

JUMP label IF R1 > R2

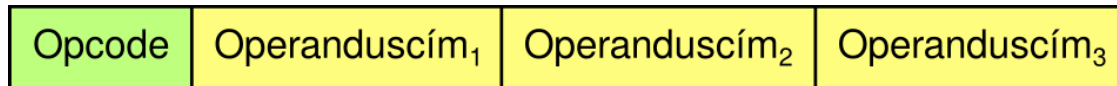
- Predikátumok: feltételhez kötött utasítások

$R1 \leftarrow R2 + 32 \text{ IF } P2$

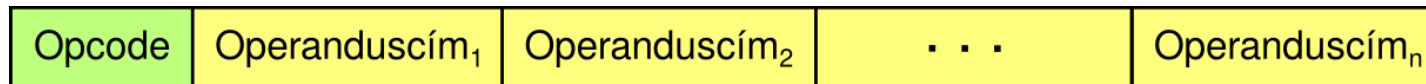
- Predikátumregiszter: végrehajtási feltételt tárol
- Állítása: összehasonlító műveletekkel

$P2 \leftarrow R3 \leq R5$

- Utasításokat binárisan kódolva tároljuk
- Binárisan kódolt utasítások hossza szerint:
 - Fix hosszú kódolás:

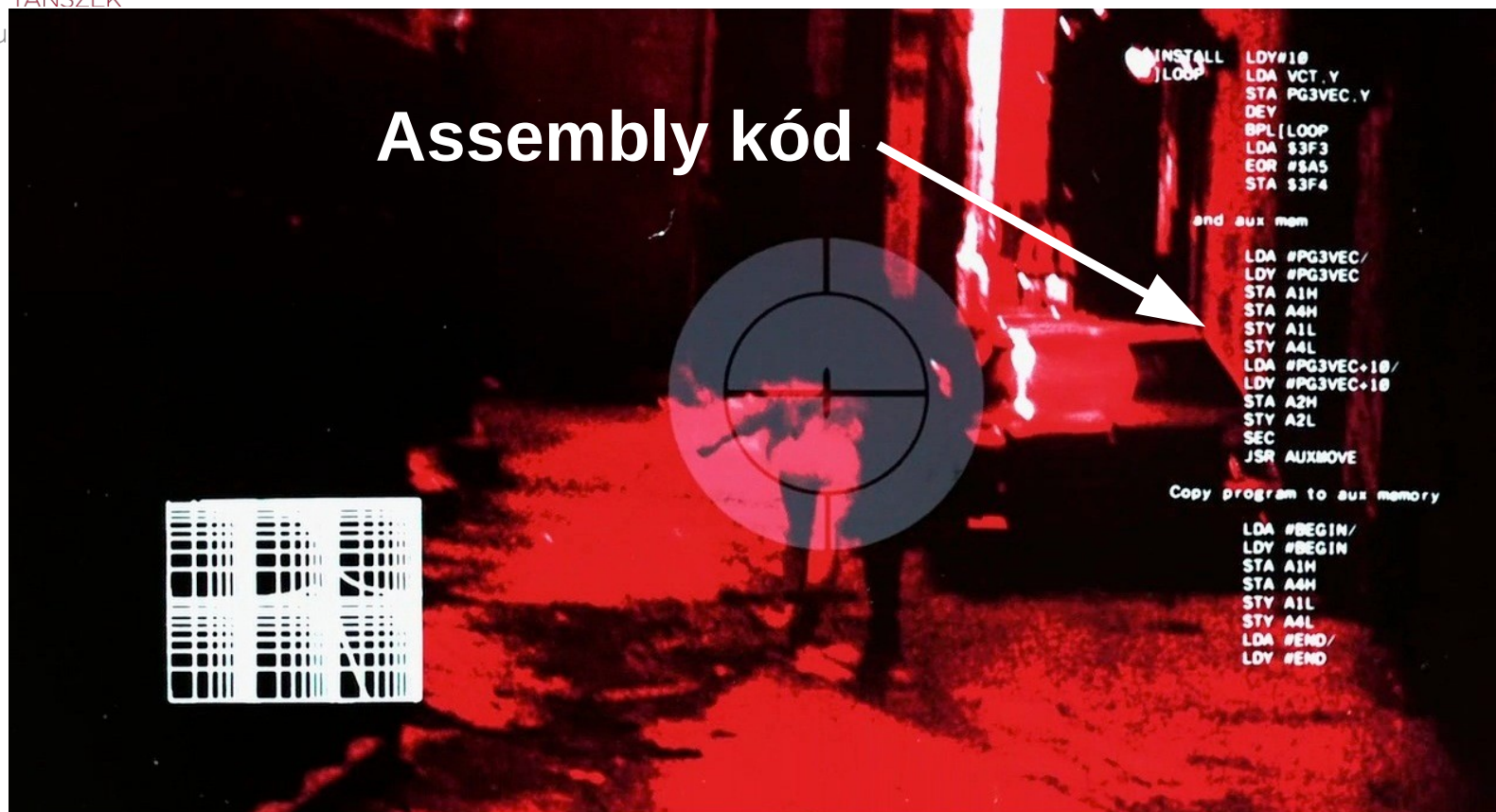


- Változó hosszú kódolás:



- Alacsony szintű programozás:
 - Utasítások kézi kódolása kényelmetlen
 - A binárisan kódolt utasítássorozat emberi fogyasztásra alkalmatlan
 - Eszköz: assembly programozás
- Assembly
 - A legalacsonyabb szintű programozási nyelv
 - A gépi utasítások szöveges megfelelője
 - 1 assembly „utasítás”-ból → 1 gépi utasítás lesz
 - Assembler: az assembly leírásból gépi kódot készít
 - Minden utasításkészlet architektúrára más és más!





- A Terminátor processzora: MOS-6502 (akár az Apple II-é) (1975 és 1980 között messze a legolcsóbb CPU: a vele azonos képességű Intel és Motorola processzorok árának hatodába került)
- A Terminátor a Nibble magazin egyik példaprogramját futtatja

Assembly kód

Binárisan kódolt
(gépi) utasítások

```
136 6085 A0 0A
137 6087 B9 E1 60
138 608A 99 F0 03
139 608D 8C
140 608E 10 F7
141 6090 AD F3 03
142 6093 49 A5
143 6095 8D F4 03
144
145
146
```

```
147 6098 A9 03
148 609A A0 F0
149 609C 85 3D
150 609E 85 43
151 60A0 84 3C
152 60A2 84 42
153 60A4 A9 03
154 60A6 A0 FA
155 60A8 85 3F
156 60AA 84 3E
157 60AC 38
158 60AD 20 11 C3
159
160
161
```

```
162 60B0 A9 60
163 60B2 A0 EF
164 60B4 85 3D
165 60B6 85 43
166 60B8 84 3C
167 60BA 84 42
168 60BC A9 64
169 60BE A0 5D
170 60C0 85 3F
171 60C2 84 3E
172 60C4 38
```

```
2833 9310 - 935F
2848 9360 - 93AF
2A5C 93B0 - 93FF
295C 9400 - 944F
2A09 9450 - 949F
21C0 94A0 - 94EF
243D 94F0 - 953F
28FD 9540 - 958F
279F 9590 - 95DF
BB 95E0 - 95E1
PROGRAM CHECK IS : 0502
```

```
INSTALL LDY#10
[LOOP] LDA VCT.Y
STA PG3VEC.Y
DEY
BPL[LOOP]
LDA $3F3
EOR $8A5
STA $3F4
```

and aux mem

```
LDA #PG3VEC/
LDY #PG3VEC/
STA A1H
STA A4H
STY A1L
STY A4L
LDA #PG3VEC+10/
LDY #PG3VEC+10/
STA A2H
STY A2L
SEC
JSR AUXMOVE
```

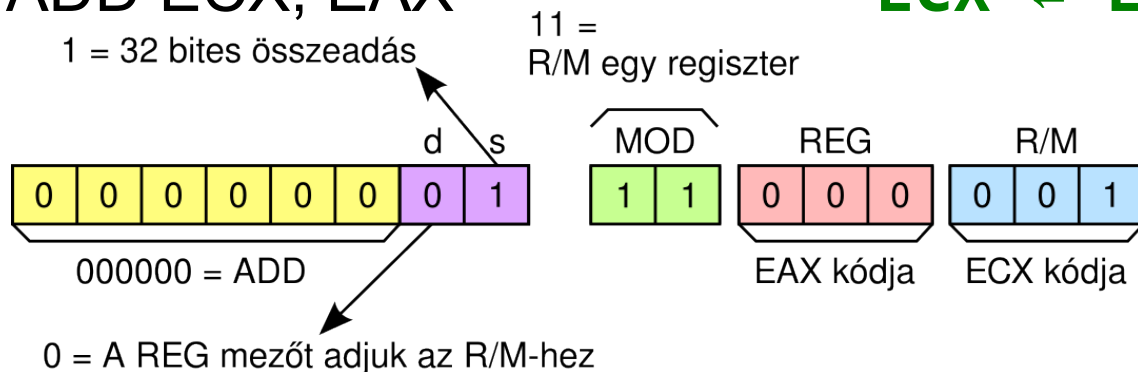
Copy program to aux memory

```
LDA #BEGIN/
LDY #BEGIN/
STA A1H
STA A4H
STY A1L
STY A4L
LDA #END/
LDY #END/
```

PÉLDA UTASÍTÁSKÓDOLÁSRA (X86)

• ADD ECX, EAX

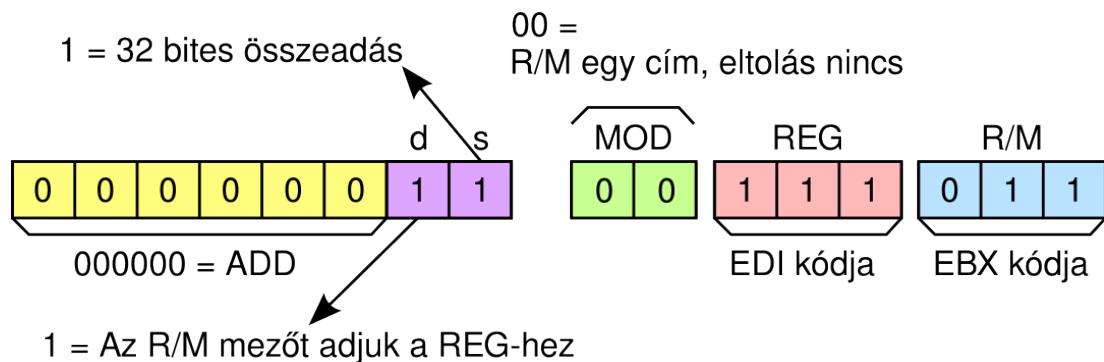
ECX ← ECX + EAX



= 01 C1
(ASCII: ☺ ⊥)

• ADD EDI, [EBX]

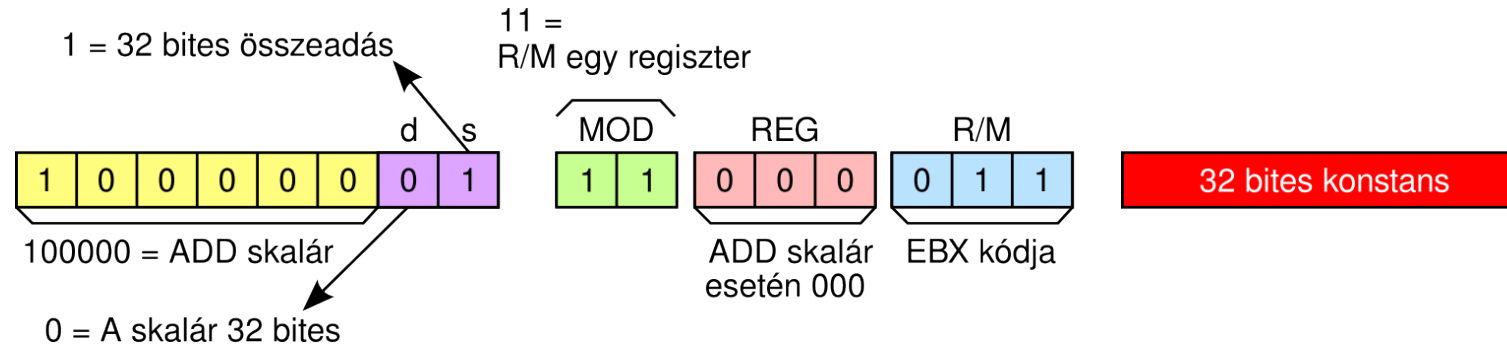
EDI ← EDI + MEM[EBX]



= 03 3B
(ASCII: ♥ ;)

- ADD EBX, 23423765

EBX ← EBX + 23423765



= 81 C3 15 6B 65 01 (ASCII: Qüşke 😊)

- Bájtsorrend:
 - Little endian: legkisebb helyiértékkel kezdi
 - Big endian: legnagyobb helyiértékkel kezdi
 - Példa: 23423765 (=1656B15)
 - Little endian: 15 6B 65 01
 - Big endian: 01 65 6B 15
- Perifériakezelő utasítások:
 - Külön I/O utasítások (IN/OUT)
 - Memóriára leképzett
- Ortogonalitás ill. közel ortogonalitás
 - Valamennyi utasítása valamennyi címzési módot használhatja

- „Trend” a 70-es években: sok, összetett utasítás
- Motiváció:
 - Lassú memória
 - Drága memória
 - Egyszerűbb fordítóprogram
- Ez a **CISC** (Complex Instruction Set Computer)
- Jellemzői:
 - Kényelmes, összetett műveletek
 - Regiszter-memória utasítások (pl. **$R1 \leftarrow R2 + \text{MEM}[42]$**)
 - Redundancia
 - Sokféle címezési mód
 - Változatos utasításhossz
 - Változatos utasítás-végrehajtási idő

- „Trend” a 80-as/90-es években: egyszerű utasítások
- Motiváció:
 - Egyszerűbb processzor
 - Átláthatóbb mikroarchitektúra
→ hatékonyabb implementáció
- Ez a **RISC** (Reduced Instruction Set Computing)
- Jellemzői:
 - Elemi utasítások, redundancia kerülése
 - Load-Store és regiszter-regiszter műveletek
R1 ← R2+MEM[42] helyett
R3 ← MEM[42]; R1 ← R2+R3.
 - Kevés címezési mód
 - Fix utasításhossz
 - Egyforma utasítás-végrehajtási idők

- Összehasonlítás:
 - CISC: tömör
 - RISC: egyszerű
 - Kevesebb tervezési hiba
 - Huzalozott vezérlés (vs. Mikroprogramozás)
 - Kisebb IC
 - Alacsonyabb fogyasztás
 - Jobb gyártási kihozatal
 - Járulékos eszközökkel integrálható
 - CISC: kevés regiszter vs. RISC: több regiszter

- **MISC: Minimal Instruction Set Computer**
 - pl. verem alapú
- **OISC: One Instruction Set Computer**
 - pl. kivonás és ugrás, ha az eredmény negatív
- **NISC: No Instruction Set Computer**
 - Nanokódot használ: közvetlenül a programozó vezérli a CPU funkcionális egységeit



Elterjedt utasításkészletek

x86

- Első tag: 1978, Intel 8086
 - Elődök: 1971, Intel 4004, 1972: 8008, 1974: 8080
- 1981: Az Intel 8088-as lesz az IBM PC processzora
- Eredetileg 16 bites volt, később 32 és 64 bites kiterjesztések
- Nagy teljesítményű szerverektől a mobil eszközök piacáig
- 43 éves kompatibilitási kényszer
- x86 licenccel rendelkeznek: Intel, AMD, (VIA)
- Intel stratégia: félvezetőipari beruházások
 - hatalmas gyártástechnológiai előny, mára elolvadt



ARM

- Első megvalósítás: 1987
- A legelterjedtebb utasításkészlet architektúra
- Kezdetektől 32 bites, 2011 óta 64 bites
- Elsődleges szempont: egyszerűség és a fogyasztás
- Az ARM nem gyárt processzort, de licenszel
 - Utasításkészletet
 - Saját tervezésű processzort (pl. ARM Cortex processzorcsalád)
- Licenszelők:
 - Qualcomm: csak utasításkészletet vett, a processzor saját tervezésű
 - Apple: vett processzor licenszet, de továbbfejlesztette
 - Nvidia, Samsung, Huawei, Rockchip, Broadcom, stb.: ARM Cortex magokat licenszelnek, gyártanak
- Fejlődés:
 - 2008: A legerősebb ARM processzor: 680MHz (850 DMIPS, Apple iPhone 1: 412 MHz)
 - 2016: Samsung Exynos 8890: 2.3 GHz, 4 mag, 46920 DMIPS
 - 2024: 4 GHz (Snapdragon X Elite, 12 mag) / Apple M3 Max (16 mag, 225 000 DMIPS)

Power

- PowerPC: 1991-ben, az IBM, Apple és Motorola összefogásában
 - A PC-kben nem terjedt el, munkaállomásokban és szerverekben igen
- IBM Power processzorok:
 - Felülmúlják az x86 processzorok teljesítményét
 - Óriási memória és I/O sávszélesség
 - 2007: 5 GHz (POWER6)
 - 2017: 12 magos, magonként 8 szál (POWER8)
 - 2019 november: a világ leggyorsabb szuperszámítógépe (Summit)
 - 9216 POWER9 core + 27648 Nvidia V100 GPU
 - 2021: POWER10 (15 db SMT8 mag, vagy 30 db SMT4 mag)



Power Systems

SPARC



- 1987, SUN, most már Oracle
- Kezdetektől 64 bites
- Nyílt platform
A UltraSPARC T1 és T2 VeriLog szinten elérhető!!!
- 2011: a világ legerősebb számítógépe SPARC alapú
(2013: a negyedik legerősebb)
- 2013: SPARC T5: 3.6 GHz, 8 mag, 16 szál/mag
- 2016: SPARC M7: 4.13 GHz, 32 mag, 8 szál/mag (256 szál!)
... és 16 foglalat/szerver!
- 2017: SPARC M8: 5 GHz, 32 mag, 8 szál/mag (256 szál!)
- 2017: Oracle leállítja a fejlesztéseket
- ... de a Fujitsu folytatja!

RISC-V



- 2010, University of California, Berkeley
- Cél: utasításkészlet architektúra létrehozása
 - Az akadémiai szféra számára is hozzáférhető
 - Nyílt licence, royalty mentes
 - Gyakorlatban is alkalmazható
- RISC-V Foundation: 2015, 2020 márciustól Svájcba költözött
- 32 és 64 bites változat, 128 bites tervezet
- RISC architektúra, 32 általános célú regiszter
- Több részből áll, kötelező és opcionális kiegészítésekkel
- Érdekesség: szubrutin hívás nem a stack-re menti a címet, hanem egy regiszterbe: jal (jump and link) és jalr (indirekt)
 - Az ortogonalitás jegyében ugyanezt használja a sima ugráshoz is

RISC-V



- Jelentősebb implementációk:
 - SiFive: ez első
+ nagy teljesítményű core-ok: szuperskalár, soron kívüli végrehajtás, stb.
 - Google Titan M2 security module for the Pixel 6
 - Seagate a diszk vezérlőiben
 - Espressif ESP32-S2 mikrokontorller
- Fejlesztés alatt:
 - NVIDIA tervezi a Falcon processzort leváltani a GeForce grafikus kártyákon
- Nyílt implementációk:
 - Alibaba group XuanTie 910
 - Western Digital 2 utatas szuper-skalár implementáció SSD vezérlő céljára
- Software support:
 - GCC, LLVM, QEMU, OpenJDK
 - Linux (csak 64 bites), FreeBSD, NetBSD, OpenBSD, FreeRTOS

- **Alpha (DEC, 1992)**
 - Kezdetektől 64 bites
 - Rendkívül innovatív:
 - 21164: első CPU nagy, a CPU-val egy szilíciumon elhelyezett cache-el
 - 21264: első CPU magas órajellel ÉS sorrenden kívüli végrehajtással
 - 21364: első CPU integrált memóriavezérlővel
 - 21464: első többszálúságot támogató CPU (lett volna, ha forgalomba kerül)
 - Nagyon erős lebegőpontos egység
21264 @ 833 MHz > 3x Pentium III @ 1 GHz!
 - Kézi tervezés
 - Halála: Compaq felvásárolja és leállítja a fejlesztést
- **PA-RISC (1986, HP)**
 - Eleinte 32, majd 64 bites
 - Nagyon erős lebegőpontos egység
PA-8600 @ 552 MHz > 2x Pentium III @ 1 GHz!
 - Halála: HP az Intel Itanium mellett teszi le a voksát

IA-64 (Itanium)

- 1994-ben indul a HP & Intel közös fejlesztés
- Hatalmas sajtófigyelem, költséges fejlesztés
- Első tag: 2001, kiábrándító teljesítménnyel, pár ezret adtak el
- Hardveres kompatibilitás az x86-tal: a 100MHz Pentium sebességén...
- Gond: spéci compiler kell, nem gondolták, hogy ez ilyen nehéz
- Azóta is fejlesztik, 2001-2007 között 55.000 db-ot adtak el
- Egyre több cég hagy fel a platform támogatásával
 - 2008: Microsoft
 - 2011 március: Oracle
- 2018: bejelentik a kivezetését (2021-ig)



- **Jim Keller**

- Az Alpha 21164 és 21264 processzorok tervezője
- 1998: Az AMD Athlon K7 és K8 processzorok tervezője (gyorsabbak voltak, mint az Intel P4!)
- 2004: P.A. Semi vezető tervezője → Apple felvásárolja (2008), Apple A4 és A5 processzorok tervezője (meghatározza az Apple CPU-k erejét)
- 2012: Vissza az AMD-hez, az új AMD Zen architektúra tervezését irányítja (megjelenés: 2017)
- 2016: a Tesla alkalmazottja (Autopilot Hardware Engineering)
- 2018 április: csatlakozik az Intelhez!
- 2020 június: elhagyja az Intelt
- Azóta: Tenstorrent



	x86	ARM	Power	SPARC
Hány bites	64	32/64	64	64
Megjelenés éve	1978	1983	1991	1985
Operandusok	2	3	3	3
Műveletek	Reg-mem	Reg-reg	Reg-reg	Reg-reg
CISC vs. RISC	CISC	RISC	RISC	RISC
Regiszterek sz.	8/16	16/32	32	32
Utasításkódolás	Vált. (1-17)	Fix (4)	Fix (4 – töm.)	Fix (4)
Felt. utasítások	Feltétel kód	Feltétel kód	Feltétel kód	Feltétel kód
Bájtrend	Little	Big	Big/Bi	Bi
Címzési módok	5	6	4	2
Perifériakezelés	I/O utasítások	Mem. lekép.	Mem. lekép.	Mem. lekép.
Predikátumok	Nincs	Van	Nincs	Nincs

	m68k	Alpha	PA-RISC	Itanium	RISC-V
Hány bites	32	64	64	64	32, 64, (128)
Megjelenés éve	1979	1992	1986	2001	2010
Operandusok	2	3	3	3	3
Műveletek	Reg-mem	Reg-reg	Reg-reg	Reg-reg	Reg-reg
CISC vs. RISC	CISC	RISC	RISC	EPIC	RISC
Regiszterek sz.	16	32	32	128	32
Utasításkódolás	Vált. (2-22)	Fix (4)	Fix (4)	Fix (16)	Fix (4)
Felt. utasítások	Feltétel kód	Feltétel reg.	Összeh. & ugr.	?	Összeh. & ugr.
Bájtsorrend	Big	Bi	Big	Bi	Little
Címzési módok	9	1	5	?	3
Perifériakezelés	Mem. lekép.	Mem. lekép.	Mem. lekép.	Mem. lekép.	Mem. lekép.
Predikátumok	Nincs	Nincs	Nincs	Van	Nincs



HÁLÓZATI RENDSZEREK
ÉS SZOLGÁLTATÁSOK
TANSZÉK



```
#include <stdio.h>

int main () {
    int i, n, a, b, f;
    scanf("%d", &n);
    a = 0;
    b = 1;
    for (i=0; i<n; i++) {
        f = a + b;
        a = b;
        b = f;
    }

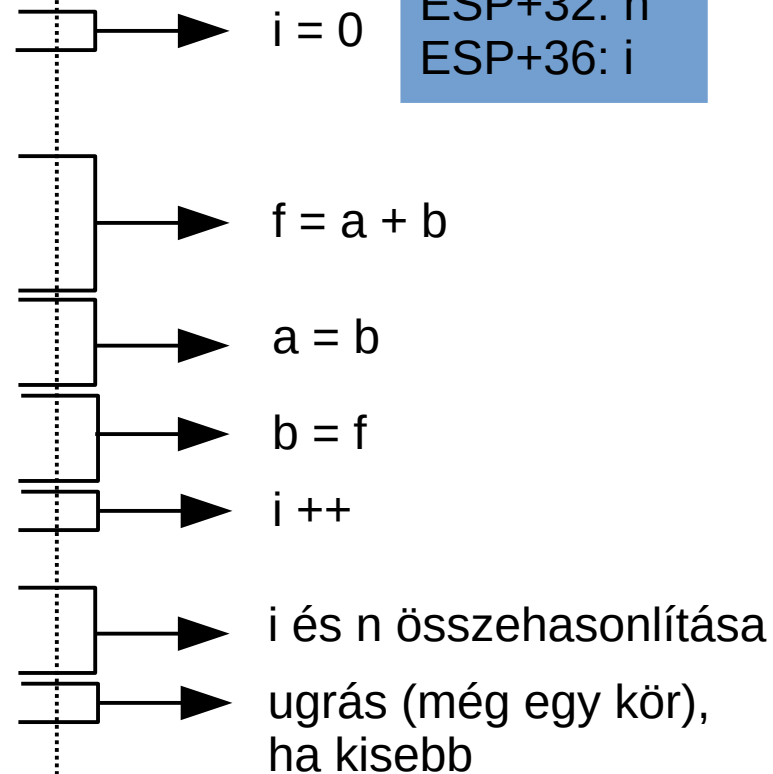
    printf ("%d\n", f);
}
```

Megvalósítás x86-ban (=IA-32)

- Csak a ciklus:

```

mov DWORD PTR [ESP + 36], 0
jmp .LBB0_1
.LBB0_2:
mov EAX, DWORD PTR [ESP + 28]
add EAX, DWORD PTR [ESP + 24]
mov DWORD PTR [ESP + 20], EAX
mov EAX, DWORD PTR [ESP + 24]
mov DWORD PTR [ESP + 28], EAX
mov EAX, DWORD PTR [ESP + 20]
mov DWORD PTR [ESP + 24], EAX
inc DWORD PTR [ESP + 36]
.LBB0_1:
mov EAX, DWORD PTR [ESP + 36]
cmp EAX, DWORD PTR [ESP + 32]
jl .LBB0_2
    
```



Megvalósítás x86-ban (=IA-32)

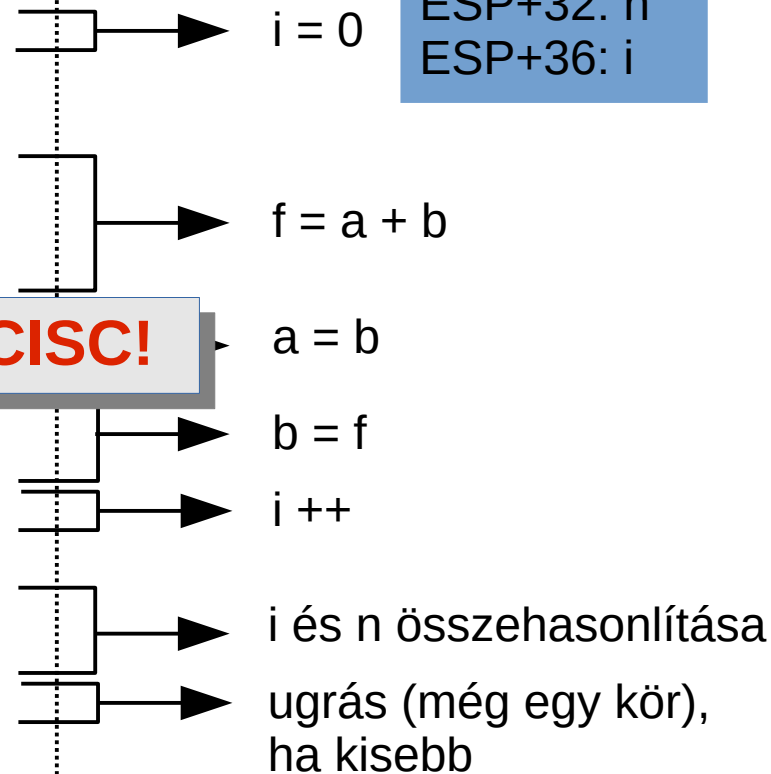
- Csak a ciklus:

```

mov DWORD PTR [ESP + 36], 0
jmp .LBB0_1
.LBB0_2:
mov EAX, DWORD PTR [ESP + 28]
add EAX, DWORD PTR [ESP + 24]
mov DWORD PTR [ESP + 20], EAX
mov EAX, DWORD PTR [ESP + 20]
mov DWORD PTR [ESP + 24], EAX
inc DWORD PTR [ESP + 36]
.LBB0_1:
mov EAX, DWORD PTR [ESP + 36]
cmp EAX, DWORD PTR [ESP + 32]
jle .LBB0_2

```

Memóriareferens műveletek: CISC!



ESP+20: f
ESP+24: b
ESP+28: a
ESP+32: n
ESP+36: i

Megvalósítás x86-ban (=IA-32)

- Csak a ciklus:

```

mov DWORD PTR [ESP + 36], 0
jmp .LBB0_1
.LBB0_2:
  1 mov EAX, DWORD PTR [ESP + 28]
  2 add EAX, DWORD PTR [ESP + 24]
  mov DWORD PTR [ESP + 20], EAX
  mov EAX, DWORD PTR [ESP + 20]
  mov DWORD PTR [ESP + 24], EAX
  inc DWORD PTR [ESP + 36]
.LBB0_1:
  mov EAX, DWORD PTR [ESP + 36]
  cmp EAX, DWORD PTR [ESP + 32]
  jl .LBB0_2
  
```

Kétooperandusú műveletek: CISC!

i = 0

f = a + b

a = b

b = f

i ++

i és n összehasonlítása

ugrás (még egy kör),
ha kisebb

ESP+20: f
ESP+24: b
ESP+28: a
ESP+32: n
ESP+36: i

Megvalósítás x86-ban (=IA-32)

- Csak a ciklus:

```
mov DWORD PTR [ESP + 36], 0
jmp .LBB0_1
```

.LBB0_2:

```
mov EAX, DWORD PTR [ESP + 28]
add EAX, DWORD PTR [ESP + 24]
```

**Feltétel kódokra alapozott
vezérlésátadás**

```
mov DWORD PTR [ESP + 24], EAX
inc DWORD PTR [ESP + 36]
```

.LBB0_1:

```
mov EAX, DWORD PTR [ESP + 36]
cmp EAX, DWORD PTR [ESP + 32]
j1 .LBB0_2
```

i = 0

f = a + b

a = b

b = f

i ++

i és n összehasonlítása

ugrás (még egy kör),
ha kisebb

ESP+20: f
ESP+24: b
ESP+28: a
ESP+32: n
ESP+36: i

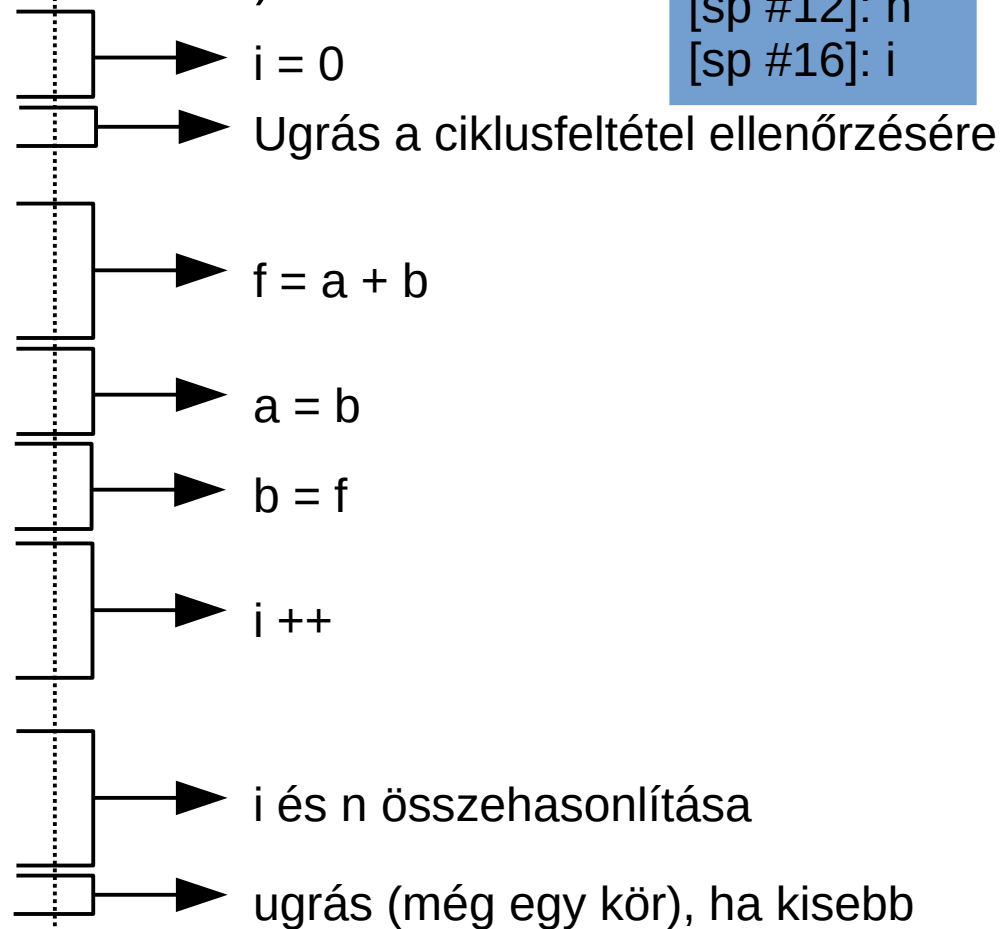
Megvalósítás ARM-ban

- Optimalizálás nélkül (csak a ciklus):

```

mov r4, #0
str r4, [sp, #16]
b .LBB0_2
.LBB0_1:
ldmib sp, {r0, r1}
add r0, r1, r0
str r0, [sp]
ldr r1, [sp, #4]
str r1, [sp, #8]
ldr r2, [sp]
str r2, [sp, #4]
ldr r1, [sp, #16]
add r0, r1, #1
str r0, [sp, #16]
.LBB0_2:
ldr r0, [sp, #12]
ldr r1, [sp, #16]
cmp r1, r0
blt .LBB0_1

```



[sp]: f
 [sp #4]: b
 [sp #8]: a
 [sp #12]: n
 [sp #16]: i

Megvalósítás ARM-ban

- Optimalizálás nélkül (csak a ciklus):

```

mov r4, #0
str r4, [sp, #16]
b .LBB0_2
.LBB0_1:
ldmi 1 2, 3, r1}
add r0, r1, r0
str r0, [sp]
ldr r1, [sp, #4]

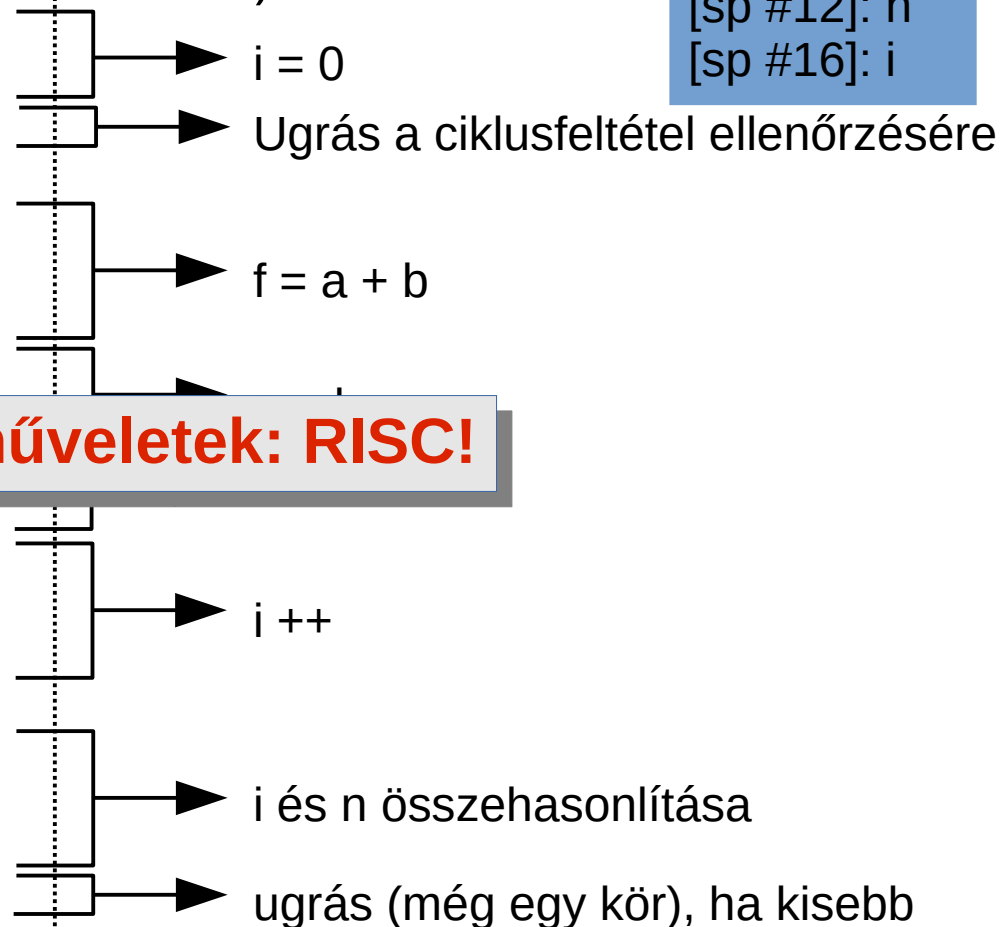
```

Háromoperandusú műveletek: RISC!

```

str r2, [sp, #4]
ldr r1, [sp, #16]
add r0, r1, #1
str r0, [sp, #16]
.LBB0_2:
ldr r0, [sp, #12]
ldr r1, [sp, #16]
cmp r1, r0
blt .LBB0_1

```

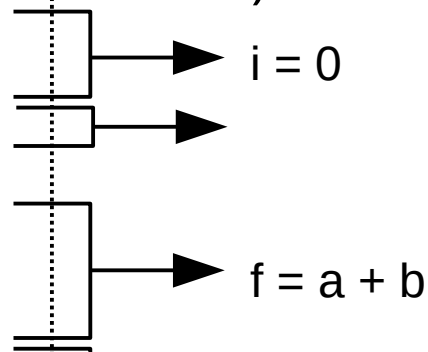


[sp]: f
[sp #4]: b
[sp #8]: a
[sp #12]: n
[sp #16]: i

Megvalósítás ARM-ban

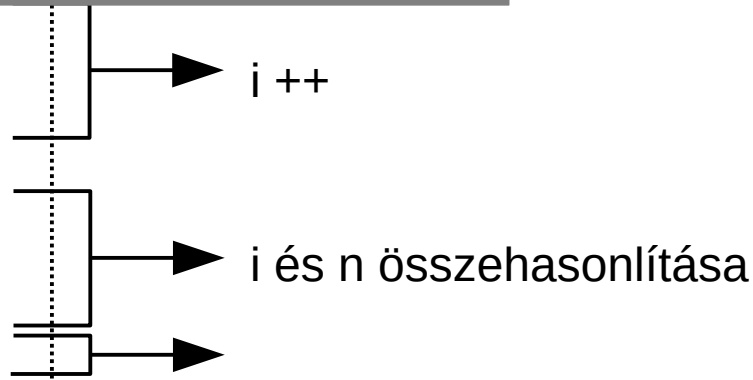
- Optimalizálás nélkül (csak a ciklus):

```
mov r4, #0
str r4, [sp, #16]
b .LBB0_2
.LBB0_1:
ldmib sp, {r0, r1}
add r0, r1, r0
str r0, [sp]
```



**Nincsenek regiszter-memória
műveletek: RISC!**

```
ldr r1, [sp, #16]
add r0, r1, #1
str r0, [sp, #16]
.LBB0_2:
ldr r0, [sp, #12]
ldr r1, [sp, #16]
cmp r1, r0
blt .LBB0_1
```



[sp]: f
[sp #4]: b
[sp #8]: a
[sp #12]: n
[sp #16]: i

Megvalósítás ARM-ban

- Optimalizálás nélkül (csak a ciklus):

```

mov r4, #0
str r4, [sp, #16]
b .LBB0_2
.LBB0_1:
ldmib sp, {r0, r1}
add r0, r1, r0
ldr r1, [sp, #16]
add r0, r1, #1
str r0, [sp, #16]
.LBB0_2:
ldr r0, [sp, #12]
ldr r1, [sp, #16]
cmp r1, r0
blt .LBB0_1
    
```

i = 0

f = a + b

i ++

i és n összehasonlítása

**Nincs redundancia, az
inkrementálásra nincs külön
utasítás: RISC!**

[sp]: f
[sp #4]: b
[sp #8]: a
[sp #12]: n
[sp #16]: i

Megvalósítás ARM-ban

- Optimalizálás nélkül (csak a ciklus):

```
mov r4, #0
str r4, [sp, #16]
b .LBB0_2
```

.LBB0_1:

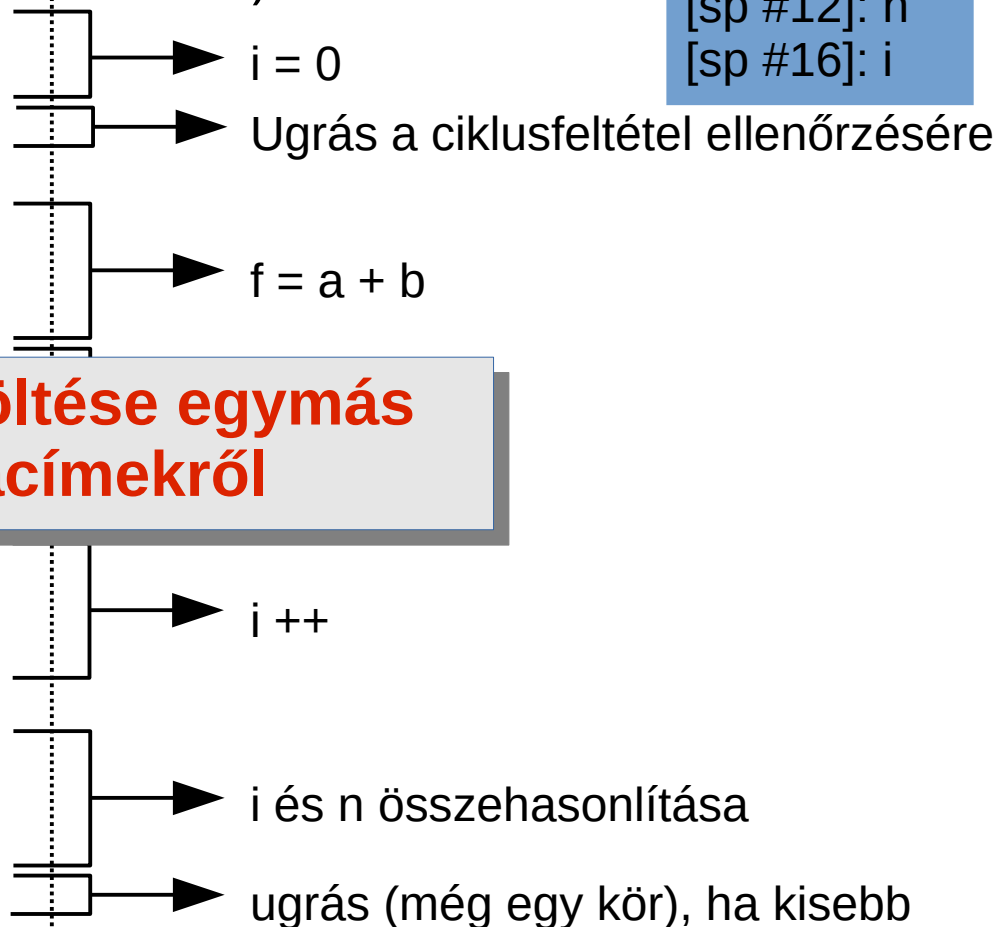
```
ldmib sp, {r0, r1}
add r0, r1, r0
str r0, [sp]
```

**Több regiszter feltöltése egymás
utáni memóriacímekről**

```
ldr r1, [sp, #16]
add r0, r1, #1
str r0, [sp, #16]
```

.LBB0_2:

```
ldr r0, [sp, #12]
ldr r1, [sp, #16]
cmp r1, r0
blt .LBB0_1
```



[sp]: f
[sp #4]: b
[sp #8]: a
[sp #12]: n
[sp #16]: i