

Az **GOTO offset** utasítás. PC relatív: PC értékéhez hozzá kell adni a két bájtos, előjeles **offset** értékét. **Mic-1** program:

Main1	PC = PC + 1; <b>fetch</b> ; goto(MBR)	
goto1	OPC=PC-1 // Main1-nél : PC=PC+1	MBR ← 1. bájt
goto2	PC=PC+1; <b>fetch</b> // <b>offset</b> 2. bájtja	
goto3	H=MBR<<8 // 1. (előjeles) bájt <<8	MBR ← 2. bájt
goto4	H=MBRU OR H // 16 bites <b>offset</b>	
goto5	PC=OPC+H; <b>fetch</b> ; goto Main1	// PC új értéke

Main1	PC = PC + 1; <b>fetch</b> ; goto(MBR)	MBR ← opkód
-------	---------------------------------------	-------------

**goto5** kezdeményezi a PC új értékénél lévő bájt olvasását, **Main1**-ben **goto(MBR)** már a folytatás első opkódjának megfelelő címre ugrik!

Máté: Architektúrák 6. előadás 1

Előkészület a gép indításakor: PC a végrehajtandó utasítás címét, MBR az utasítás kódját tartalmazza.

Máté: Architektúrák Gyakorlásra 6. előadás mic\_5.swf 2

A **IFLT offset** utasítás (**Mic-1**)  
Kivesz egy szót a veremből és ugrik, ha negatív.

iflt1	MAR=SP=SP-1; rd	// 2. szó a veremből
iflt2	OPC=TOS	// TOS mentése
iflt3	TOS=MDR	// TOS= a verem új teteje
iflt4	N=OPC; if(N) goto T; else goto F	//elágazás
T	OPC=PC-1; goto goto2	// igaz ág
F	PC=PC+1	// hamis ág, át kell lépni <b>offset</b> -et
F2	PC=PC+1; <b>fetch</b> ; goto (Main1)	// PC új értéke, az <b>offset</b> utáni utasításának betöltése

**Fontos:** T címe = F címe + 0x100

Máté: Architektúrák 6. előadás 3

Előkészület a gép indításakor: PC a végrehajtandó utasítás címét, MBR az utasítás kódját tartalmazza.

Máté: Architektúrák Gyakorlásra 6. előadás mic\_5.swf 4

**INVOKEVIRTUAL disp** (4.12. ábra),

- A **CPP** által mutatott területen a **disp** (2 bájt) indexű szó mutat a meghívandó metódus kezdő szavára.
- Ennek a szónak
  - az első két bájtja tartalmazza a metódus paramétereinek számát,
  - a második két bájtja a metódus lokális változóinak számát.
- A metódus végrehajtása a metódus 5. bájtján indul.

Máté: Architektúrák 6. előadás 5

**INVOKEVIRTUAL disp** (~4.12. ábra)

Az ábra ajánlott

Máté: Architektúrák 6. előadás 6

**INVOKEVIRTUAL disp** **4.17. ábra**

A **CPP** által mutatott táblázat **disp** indexű eleme a meghívandó metódusra mutat. **disp** első bájtnak **MBR**-be olvasását már **Main1** kezdeményezte.

invo1	PC = PC + 1; fetch // <b>disp</b> 2. báját olvassa
invo2	H = MBRU << 8 // <b>disp</b> 1. báját lépteti
invo3	H = MBRU OR H // H = <b>disp</b>
invo4	MAR = CPP + H; rd // kezdő cím olvasása
invo5	OPC = PC + 1 // OldPC = visszatérési cím

Máté: Architektúrák **Nem kell** 6. előadás 7

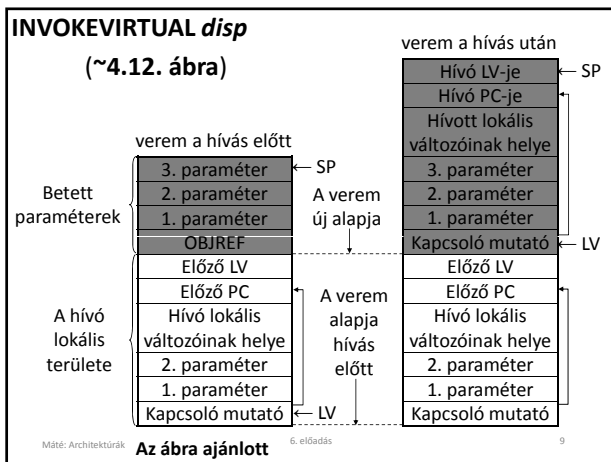
**INVOKEVIRTUAL**: a metódus elején lévő 2 bájtt tartalmazza a paraméterek számát.

invo6	PC = MDR; fetch // PC: új metódus eleje
invo7	PC = PC + 1; fetch // paraméterek száma
invo8	H = MBRU << 8
invo9	H = H OR MBRU // H = paraméterek száma
invo10	TOS = SP - H // <b>OBJREF is paraméter!</b>
invo11	TOS = MAR = TOS + 1 // <b>OBJREF címe</b>

**TOS**-ban tároljuk ideiglenesen **OBJREF** címét, ide mutat majd a hívott metódus **LV**-je.

**A mutasítások sorrendje más, mint a könyvben!**

Máté: Architektúrák **Nem kell** 6. előadás 8



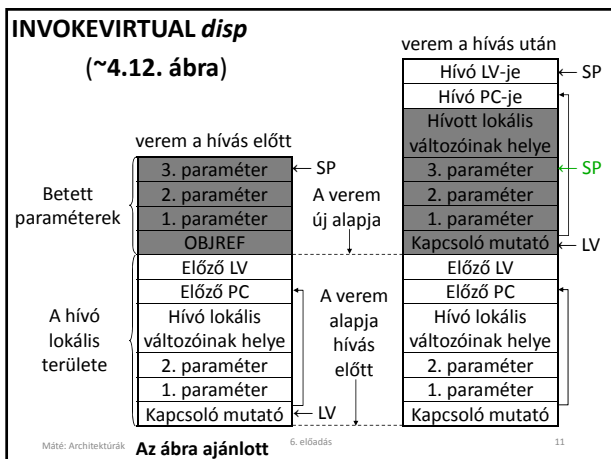
**INVOKEVIRTUAL**: lokálisok száma 2 bájtt a paraméterek száma után, **OBJREF** helyére kerül a lokális változók fölötti címre mutató **Kapcsoló mutató**. A mutatott címre kerül majd a **Hívó PC-je**.

TOS = MAR = OBJREF címe

invo12	PC = PC + 1; fetch // lokálisok száma 1. bájtt
invo13	PC = PC + 1; fetch // lokálisok száma 2. bájtt
invo14	H = MBRU << 8
invo15	H = H OR MBRU // H = lokálisok száma
invo16	MDR = H + SP + 1; wr // OBJREF cseréje

MDR = Hívó PC-jének címe

Máté: Architektúrák **Nem kell** 6. előadás 10

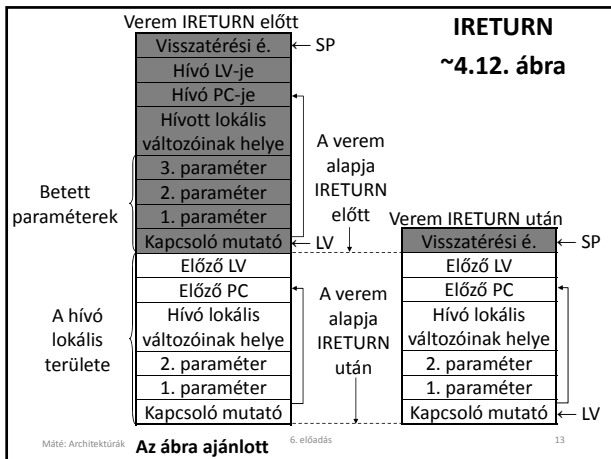


**INVOKEVIRTUAL**: a hívó PC-je és LV-je TOS = OBJREF címe

invo17	MAR = SP = MDR // hívó PC-jének helye
invo18	MDR = OPC; wr // hívó PC-jének veremlése
invo19	MAR = SP = SP + 1 // hívó LV-jének a helye
invo20	MDR = LV; wr // hívó LV-jének veremlése
invo21	PC = PC + 1; fetch // utasítás olvasás
invo22	LV = TOS // LV új értéke (a verem új alapja)
invo23	TOS = MDR; goto Main1 // TOS=hívó LV-je

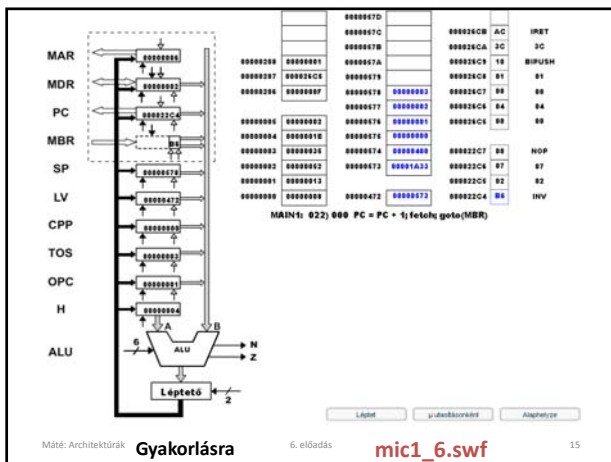
**TOS = MDR** nélkül **TOS** a **Kapcsoló mutató** címét (a **hívott LV-jét**) tartalmazná!

Máté: Architektúrák **Nem kell** 6. előadás 12



IRETURN // ~4.13. ábra	
iret1	MAR = SP = LV; rd // kapcs. mut. olvasása
iret2	H = H // vár, hogy az olvasás befejeződjön
iret3	LV = MAR = MDR; rd // hívó PC olvasása
iret4	MAR = LV + 1; rd // hívó LV címe
iret5	PC = MDR; fetch // hívó PC, opkód olv.
iret6	MAR = SP // visszatérési érték címe
iret7	LV = MDR // hívó LV
iret8	MDR = TOS; wr; goto Main1

**iret3-4: MAR nem lehet SOURCE operandus!**



**Házi feladat: A 4.17. ábra többi része.**

**Továbbfejlesztések: több sines rendszerek.**

**A mikroarchitektúra szint tervezése**

**Mic-1:** olcsó, de lassú.

**Sebesség növelés:**

- rövidebb óraciklus,
- kevesebb  $\mu$ utasítás az utasítások végrehajtásához,
- az utasítások végrehajtásának átlapolása.

**B** sín 9 regiszterét 4 bittel címeztük: dekódolóra van szükség, növeli az adatút ciklus idejét! (4.6. ábra)

**Úthossz** (path length, a szükséges ciklusok száma)  
**rövidítése:** Main1 néha megspórolható, jobb mikroprogram vagy pl. PC növelésére (ez legtöbbször **fetch**-csele együtt történik) külön áramkör.

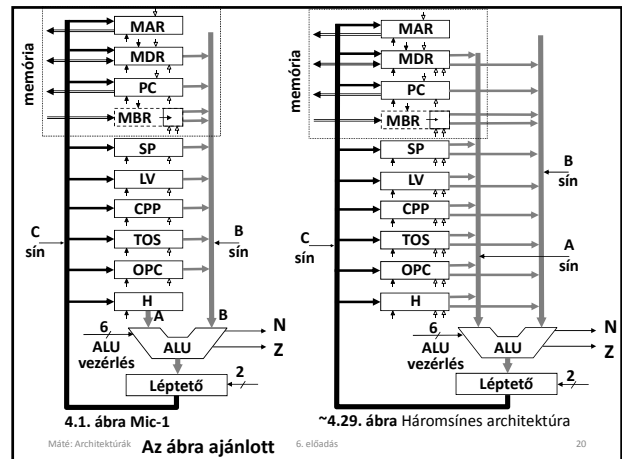
**A Main1 néha megspórolható:**

- Minden **IJVM** utasítás értelmezése akkor fejeződik be (kezdődik), amikor a **fetch; goto(MBR)** végrehajtásra kerül (ez a **Main1**-ben volt). Ilyenkor **MBR**-nek a következő **IJVM** utasítás kódját kell tartalmaznia, és megkezdődik ennek az utasításnak az értelmezése. Ez a **fetch** az utasítás kódja **utáni** bájtot olvasását kezdeményezi. Korábbi  $\mu$ utasítás ezt nem kezdeményezheti, mert akkor **MBR** tartalmát fölülírná a **goto(MBR)** végrehajtása előtt.
- A következő **IJVM** értelmezésének első  $\mu$ utasítása nem használhatja **MBR**-t az adatúton, mert ilyenkor **MBR**-ben még az **IJVM** utasítás kódja található.

**Main1** néha megspórolható (4.23-24. ábra):  
0x57 POP A verem legfelső szavát eldobja.

pop1	MAR=SP=SP-1; rd	//2. szó címe, olvas
pop2		// vár
pop3	TOS=MDR; goto main1	//TOS=a verem teteje
Main1	PC=PC+1; fetch; goto(MBR)	//következő ut.
Új változat		
pop1	MAR=SP=SP-1; rd	
pop2	PC=PC+1; <del>fetch</del>	
pop3	TOS=MDR; <del>fetch</del> ; goto(MBR)	//következő ut.

Máté: Architektúrák 6. előadás 19



**Három sínés architektúra**

Sok regiszter csatlakozhat az A sínhez, nemcsak H (4.1., 4.29. ábra).

A három sínés architektúra előnye a két sínés architektúrával szemben:  
pl. **iload** -ban nem kell  $H = LV$  (4.25-26. ábra).

**ILOAD varnum** // lokális változó a verembe  
**varnum** a lokális változó 8 bites indexe.

Máté: Architektúrák 6. előadás 21

**Mic-1 kód (4.25. ábra)**

iload1	H = LV
iload2	MAR = MBRU + H; rd
iload3	MAR = SP = SP + 1
iload4	PC = PC + 1; fetch; wr
iload5	TOS = MDR; goto main1
main1	PC = PC + 1; fetch; goto(MBR)

~~Három sínés kód(4.26. ábra)~~

iload1	MAR = MBRU + LV; rd
iload2	MAR = SP = SP + 1
iload3	PC = PC + 1; fetch; wr
iload4	TOS = MDR
iload5	PC = PC + 1; fetch; goto(MBR)

Máté: Architektúrák 6. előadás 22

Hibás a könyvben lévő kód (4.26. ábra), mert még nem áll rendelkezésre **MBRU** értéke, mert az előző utasítás utolsó  $\mu$ utasításában volt az a **fetch**, amely az **ILOAD** operandusát olvassa.

~~Három sínés kód(4.26. ábra)~~

iload1	MAR = MBRU + LV; rd
iload2	MAR = SP = SP + 1
iload3	PC = PC + 1; fetch; wr
iload4	TOS = MDR
iload5	PC = PC + 1; fetch; goto(MBR)

Máté: Architektúrák 6. előadás 23

**Mic-1 kód (4.25. ábra)**

iload1	H = LV
iload2	MAR = MBRU + H; rd
iload3	MAR = SP = SP + 1
iload4	PC = PC + 1; fetch; wr
iload5	TOS = MDR; goto main1
main1	PC = PC + 1; fetch; goto(MBR)

~~Három sínés kód(4.26. ábra)~~

iload1	PC = PC + 1; fetch
iload2	MAR = MBRU + LV; rd
iload3	MAR = SP = SP + 1
iload4	TOS = MDR; wr
iload5	PC = PC + 1; fetch; goto(MBR)

Máté: Architektúrák 6. előadás 24

A PC-vel kapcsolatos teendők:

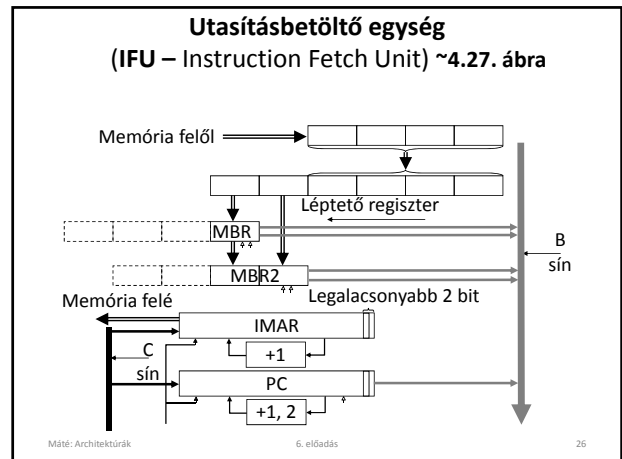
- PC növelése 1-gyel,
- fetch,
- 2 bájtos operandus olvasás a memóriából.

ALU-nál egyszerűbb áramkörrel megvalósíthatók.

**Utásításbetöltő egység (IFU – Instruction Fetch Unit)**

- értelmezhet minden kódot, hogy kell-e operandus,
- de egyszerűbb, ha a kódtól függetlenül előkészíti a következő 8 és 16 bites részt (4.27. ábra).

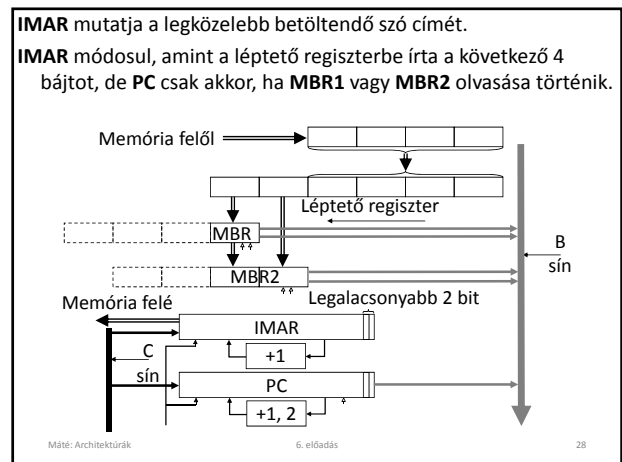
Máté: Architektúrák 6. előadás 25



**Utásításbetöltő egység (IFU – Instruction Fetch Unit) ~4.27. ábra**

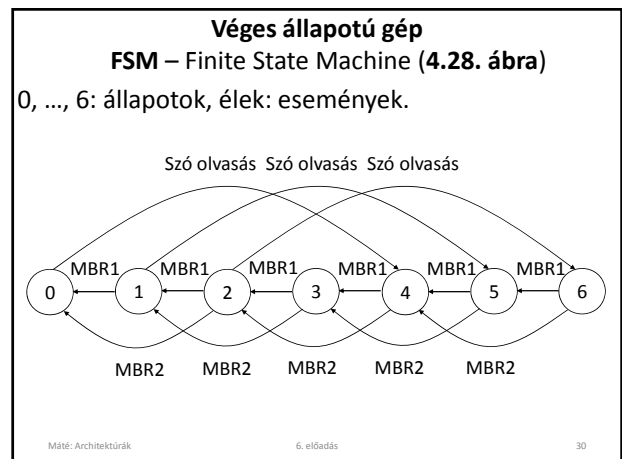
A memóriából szavanként történik az olvasás a léptető regiszter utolsó 4 bájtyára, amint lehetséges. A léptető regiszterből újra feltöltésre kerül MBR és MBR2, amint olvasás történik MBR-ből, vagy MBR2-ből, ilyenkor a léptető regiszter tartalma 1 vagy 2 bájttal balra lép.

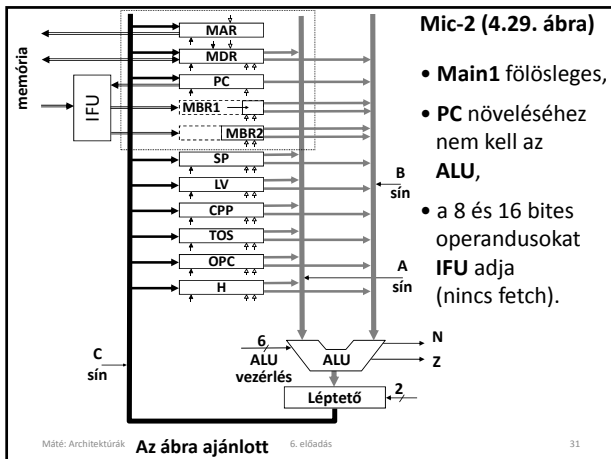
This diagram is similar to the previous one but highlights the step-by-step reading of instructions from memory into the instruction register. It shows the 'Memória felől' input, the 'Léptető regiszter', and the 'MBR' and 'MBR2' buffers. The 'B sín' is also shown. The diagram is labeled 'Máté: Architektúrák 6. előadás 27'.



Ha PC értéket kap a C sínról, azt IMAR is megkapja. Ilyenkor a mikroprogramnak várnia kell a léptető regiszter, MBR1 és MBR2 feltöltésére.

This diagram is similar to the previous one but highlights the update of the IMAR and PC. It shows the 'Memória felől' input, the 'Léptető regiszter', and the 'MBR' and 'MBR2' buffers. The 'B sín' is also shown. The diagram is labeled 'Máté: Architektúrák 6. előadás 29'.





**Mic-2 (4.29. ábra)**

Több hardver kell az A sín címzése és IFU miatt, de kevesebb  $\mu$ utasítás kell, pl. **WIDE ILOAD**-hoz az eddigi 9 helyett csak 4 (v.ö. 4.17. ábra). **WIDE ILOAD** *varnum* //beteszi a 16 bites *varnum* indexű lokális változót a verembe:

wide1	goto (MBR1 OR 0x100)
w_iload1	MAR=LV+MBR2U; rd; goto iload2
iload1	MAR=LV+MBR1U; rd // változó olvasása
iload2	MAR=SP=SP+1 // veremelés előkészítése
iload3	TOS=MDR; wr; goto (MBR1)

Máté: Architektúrák 6. előadás 32

Vezérlés átadaskor várni kell, míg az IFU elkészül (a léptető megkapja az új értékét, MBR1 és MBR2 feltöltése megtörténik).

**GOTO offset**  
az utasítás PC relatív: PC értékéhez hozzá kell adni a két bájtos, előjeles **offset** értékét. **Mic-2** program:

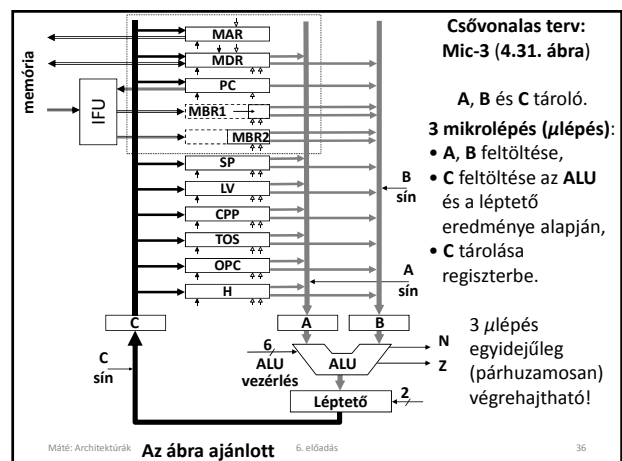
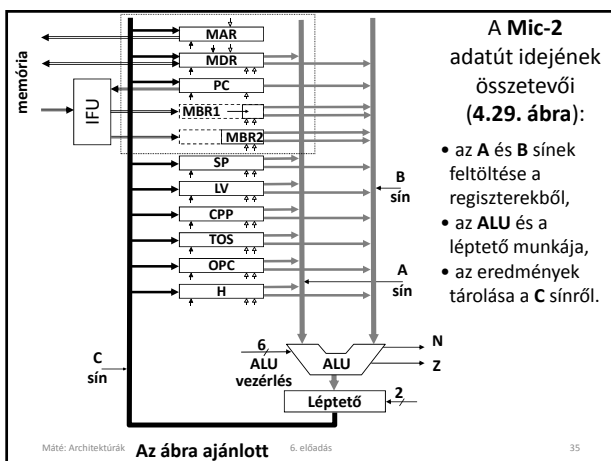
goto1	H=PC-1 // IFU már csinált PC=PC+1-et
goto2	PC=H+MBR2 // itt folytatódik a program
goto3	// IFU még nincs kész, várni kell!
goto4	goto (MBR1) // a folytatás 1. utasítása

Máté: Architektúrák 6. előadás 33

**Az IFLT offset utasítás (Mic-2)**  
Kivesz egy szót a veremből és ugrik, ha negatív.

iflt1	MAR=SP=SP-1; rd // 2. szó a veremből
iflt2	OPC=TOS // TOS mentése
iflt3	TOS=MDR // TOS= a verem új teteje
iflt4	N=OPC; if(N) goto T; else goto F //elágazás
T	H=PC-1; goto goto2 // igaz ág
F	H=MBR2 // hamis ág, eldobja offset-et
F2	goto (MBR1) // a folytatás 1. utasítása

Máté: Architektúrák 6. előadás 34



Pl.: a verem két felső szavának cseréje **Mic-3-on (4.33. ábra)**:

	swap1	swap2	swap3	swap4	swap5	swap6
cy	MAR=SP-1;rd	MAR=SP	H=MDR;wr	MDR=TOS	MAR=SP-1;wr	TOS=H;goto(MBR1)
1	B=SP					
2	C=B-1	B=SP				
3	MAR=C	C=B	Várni kell!			
4	rd	MAR=C	Várni kell!			
5			B=MDR			
6			C=B	B=TOS		
7			H=C	C=B	B=SP	
8			wr	MDR=C	C=B-1	B=H
9					MAR=C	C=B
10					wr	TOS=C
11						goto(MBR1)

Valódi függőség RAW – Read After Write!  
Elakadás

eldugaszolja a csővezetékét!

Máté: Architektúrák 6. előadás 37

### Feladatok

Mire szolgál a **GOTO** utasítás?  
 Hogy valósítható meg a **GOTO** utasítás **Mic-1-en**?  
 Melyek az **IJVM** feltételes ugró utasításai?  
 Mire szolgál az **IFLT** utasítás?  
 Hogy valósítható meg az **IFLT** utasítás **Mic-1-en**?

Máté: Architektúrák 6. előadás 38

### Feladatok

Mit nevezünk metódusnak?  
**IJVM** melyik utasítása szolgál a metódus hívására?  
 Mire szolgál az **INVOKEVIRTUAL** utasítás?  
 Hol található az **INVOKEVIRTUAL disp** utasítással hívott metódus?  
 Milyen információ van a metódus elején?  
 Hogy néz ki a veremnek egy metódus számára látható része?

Máté: Architektúrák 6. előadás 39

### Feladatok

Mely regiszterek tartalmát kell menteni metódus hívás esetén?  
**IJVM** melyik utasítása szolgál a metódusból való visszatérésre?  
 Hogy található meg a mentett regiszter tartalmak visszatéréskor?  
 Miért nem lenne jó **IRETURN** megvalósításában:  
**iret3: MAR = MDR; rd**  
**iret4: MAR = MAR + 1 ?**  
 Hol található a metódus visszatérési értéke az **IRETURN** utasítás végrehajtása után?  
 Elemezze a **4.17. ábra** programjait!

Máté: Architektúrák 6. előadás 40

### Feladatok

Az **IRETURN** utasítás mikroprogramozását úgy is megvalósíthatjuk volna, hogy a **Kapcsoló mutatót** nem használjuk, hanem **SP** értékéből indulunk ki. Hogyan? Így 7 utasítás is elegendő lett volna. Miért jobb mégis az előadáson bemutatott megoldás?

**Nem kell**

Máté: Architektúrák 6. előadás 41

### Feladatok

Mi az úthossz?  
 Milyen lehetőségek vannak a **Mic-1** gyorsítására?  
 Jellemezze a 3 sínes architektúrát!  
 Mi az előnye a három sínes architektúrának a **Mic-1**-gyel szemben?  
 Sorolja fel a **Mic-1** és **Mic-2** közötti különbségeket!  
 Miért eredményeznek ezek gyorsítást?

Máté: Architektúrák 6. előadás 42

**Feladatok**

Mi az utasítás betöltő egység (**IFU**) feladata?  
 Milyen részei vannak az **IFU**-nak?  
 Mi az **IMAR** szerepe az **IFU**-ban?  
 Írja le az **IMAR** és a **PC** regiszter kapcsolatát?  
 Hogy működik az **IFU**?  
 Hogy ábrázolható véges állapotú géppel (**FSM**) az **IFU** működése?  
 Mi a különbség a **Mic-2** és **Mic-3** között? Miért eredményez ez gyorsítást?

Máté: Architektúrák

6. előadás

43

**Feladatok**

A **SWAP** utasítás (a verem két felső szavának cseréje) **Mic-3**-on négy  $\mu$ utasítással megoldható kilenc  $\mu$ lépésben. Hogyan?  
 A megoldás nem vihető át **Mic-2**-re. Miért?  
 A feladat nehéz! Élesen ki kell használni az adatút szakaszainak időzítését. Ezt ugyan nem tárgyaltuk, de kikövetkeztethető abból, hogy az egyes szakaszok egyidejűleg működhetnek.

Máté: Architektúrák

**Nem kell**

6. előadás

44

**Az előadáshoz kapcsolódó****Fontosabb témák**

Feltétlen és feltételes elágazó utasítás megvalósítása **Mic-1**-en.  
 Utasítás betöltő egység. **Mic-2**.  
 Csővonalas terv: **Mic-3**.

Máté: Architektúrák

6. előadás

45