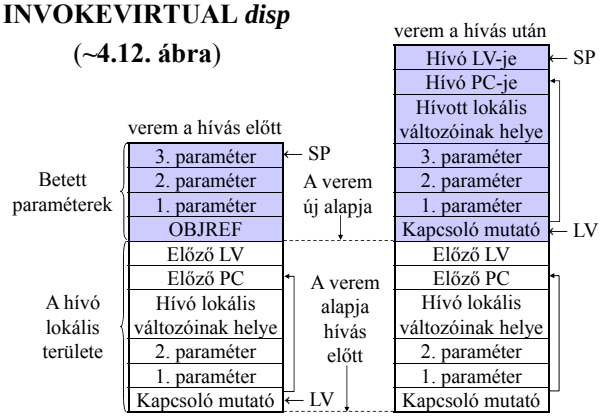
INVOKEVIRTUAL *disp* (**4.12. ábra**),

- 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.

INVOKEVIRTUAL *disp*

(~4.12. ábra)



INVOKEVIRTUAL *disp*

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

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

INVOKEVIRTUAL: a metódus elején lévő 2 bájt tartalmazza a paraméterek számát.

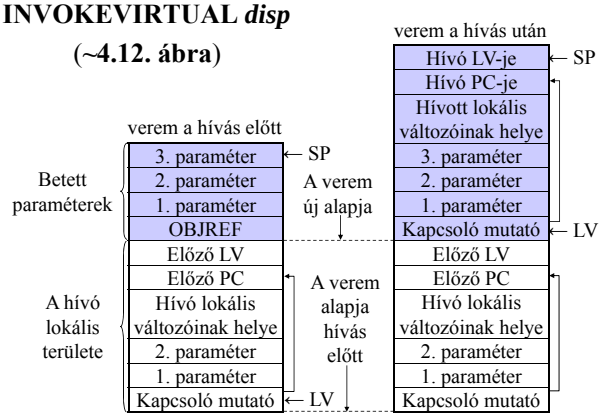
invo6	PC = MDR; fetch // PC: új metódus eleje
invo7	PC = PC + 1; fetch // paraméterek száma
invo8	H = MBRU << 8
invo9	H = H *OR* MBRU // H = paraméterek száma
invo10	TOS = SP – H // OBJREF is paraméter!
invo11	TOS = MAR = TOS + 1 // OBJREF címe

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

Az utasítások sorrendje más, mint a könyvben!

INVOKEVIRTUAL *disp*

(~4.12. ábra)



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

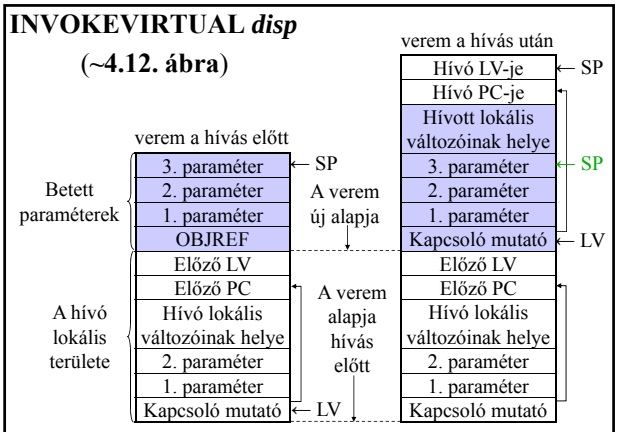
TOS = MAR = OBJREF címe

invo12	PC = PC + 1; fetch // lokálisok száma 1. bájt
invo13	PC = PC + 1; fetch // lokálisok száma 2. bájt
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

INVOKEVIRTUAL *disp*

(~4.12. ábra)

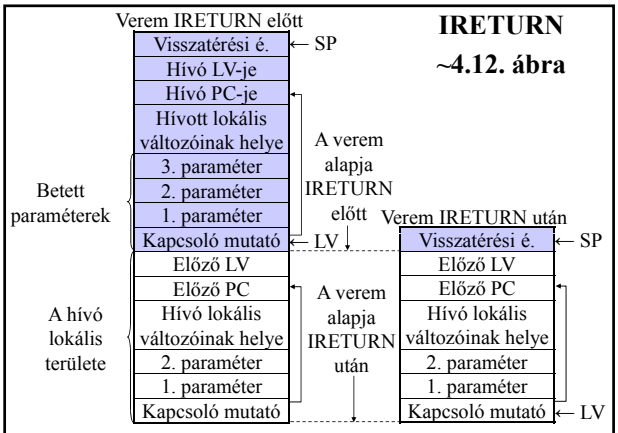


INVOKEVIRTUAL: a hívó **PC**-je és **LV**-je TOS = OBJREF címe	
invo17	MAR = SP = MDR // hívó PC-jének helye
invo18	MDR = OPC; wr // hívó PC-jének vermelése
invo19	MAR = SP = SP + 1 // hívó LV-jének a helye
invo20	MDR = LV; wr // hívó LV-jének vermelése
invo21	PC = PC + 1; fetch // utasítás olvasás
invo22	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!

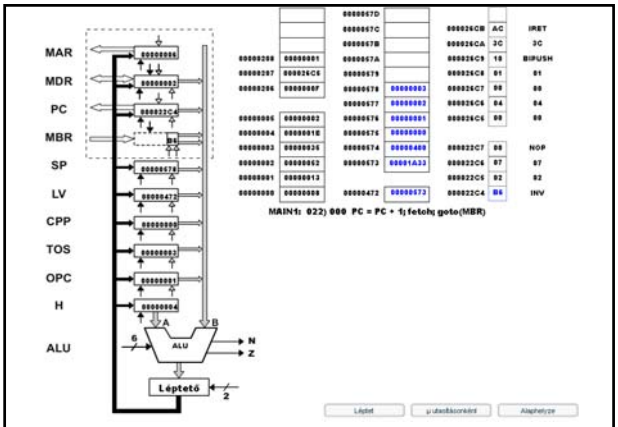
IRETURN

~4.12. ábra



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!



mic1_6.swf

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

Továbbfejlesztések: több sínes rendszerek.

A mikroarchitektúra szint tervezése

Mic-1: olcsó, de lassú. **Sebesség növelés:**

- rövidebb óraciklus,
- kevesebb mikroutasítás 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**: **goto Main1** néha megspórolható, jobb microprogram vagy pl. **PC** növelésére külön áramkör (ez legtöbbször ***fetch***-csel együtt történik).

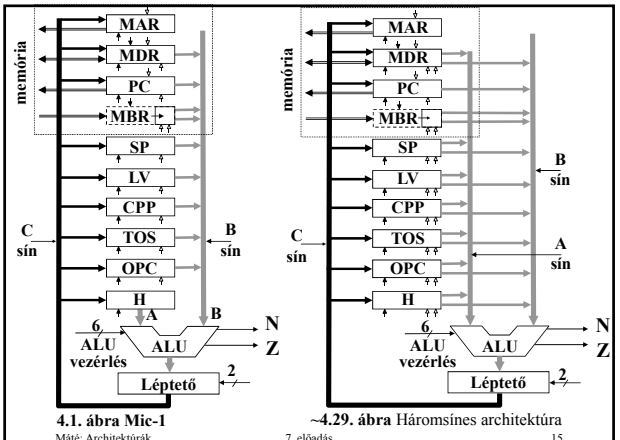
goto Main1 néha megspórolható (**4.23-24. ábra**):

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; ~~fetch~~ //következő ut. olvasása
pop3	TOS=MDR; fetch; goto(MBR)



4.1. ábra Mic-1 ~**4.29. ábra** Háromsínes architektúra

Három sínes architektúra

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

A működése: Minden **IJVM** utasítás értelmezése akkor fejeződik be, amikor a ***fetch; goto(MBR)*** végrehajtásra kerül. Ilyenkor **MBR**-nek a következő **IJVM** utasítás kódját kell tartalmaznia, és megkezdődik ennek az utasításnak az értelmezése. Ez a ***fetch*** kezdeményezi a program következő bájtjának olvasását. Korábbi mikroutasítás ezt nem kezdeményezheti, mert akkor **MBR** tartalmát fölülírná a ***goto(MBR)*** végrehajtása előtt. A következő **IJVM** értelmezésének első mikroutasítása nem használhatja **MBR**-t az adatúton, mert ilyenkor **MBR**-ben még az **IJVM** utasítás kódja található.

A három sínes architektúra előnye a két sínes 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.

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 sínes 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)

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

Három sínes 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)

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 sínes kód(**4.26. ábra**)

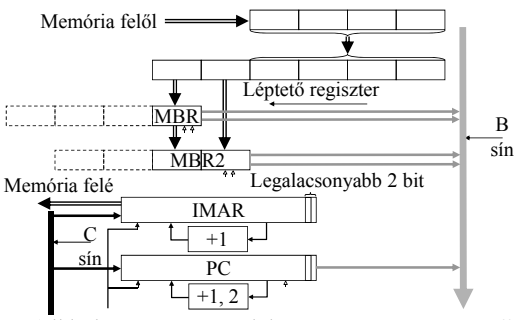
iload1	PC = PC + 1; fetch
iload2	MAR = MBRU + LV; rd
iload3	MAR = SP = SP + 1
iload4	TOS = MDR; wr
iload5	PC = PC + 1; fetch; goto(MBR)

A **PC**-vel kapcsolatos teendők:
- **PC** növelése **1**-gyel,
- **fetch**,
- 2 bájtos operandus olvasás a memóriából.

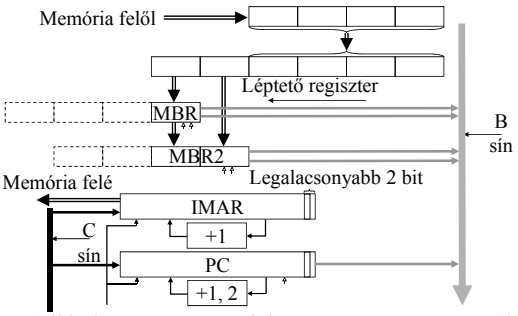
ALU-nál egyszerűbb áramkörrel megvalósíthatók.

Utasításbetöltő 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**).

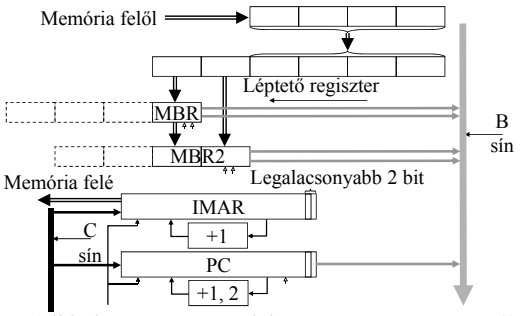
Utasításbetöltő egység
(**IFU –** Instruction Fetch Unit) ~**4.27. ábra**



IMAR módosul, amint a léptető regiszterbe írta a következő 4 bájtot, de **PC** csak akkor, ha **MBR1** vagy **MBR2** olvasása történik.



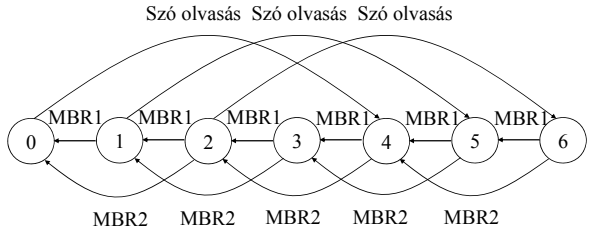
Ha **PC** értéket kap a **C** sínről, azt **IMAR** is megkapja. Ilyenkor a mikroprogramnak várnia kell a léptető regiszter, **MBR1** és **MBR2** feltöltésére.

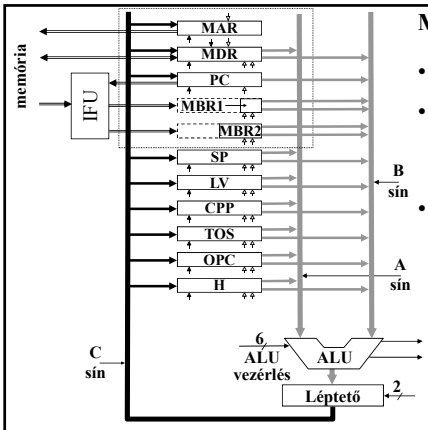


Véges állapotú gép

FSM – Finite State Machine (**4.28. ábra**)

0, …, 6: állapotok, élek: események.





Mic-2 (4.29. ábra)

- **Main1** fölösleges,
- **PC** növeléséhez nem kell az **ALU**,
- a 8 és 16 bites operandusokat **IFU** adja.

Mic-2 (4.29. ábra)

Több hardver kell az **A** sín címzése és **IFU** miatt, de kevesebb mikroutasítás 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 // vermelés előkészítése
iload3	TOS=MDR; wr; goto (MBR1)

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

GOTO *offset*

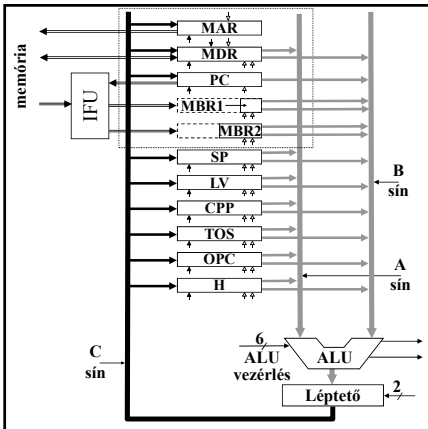
az utasítás **PC** relatív: **PC** értékéhez hozzá kell adni a két bájtos, előjeles ***offset*** értékét. **Mic-2** program:

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

Az **IFLT *offset*** utasítás (**Mic-2**)

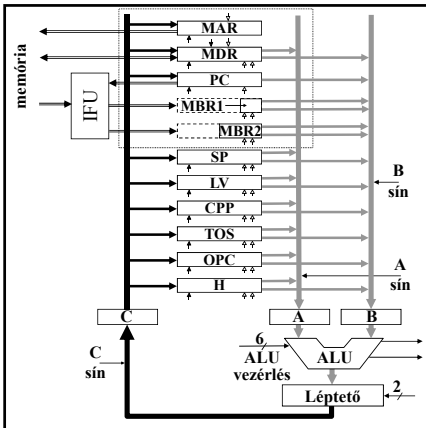
Kivesz egy szót a veremből és ugrik, ha negatív.

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



A **Mic-2** adatút idejének összetevői (**4.29. ábra**):

- az **A** és **B** sínek feltöltése a regiszterekből,
- az **ALU** és a léptető munkája,
- az eredmények tárolása a **C** sínről.



Csővonalas terv: Mic-3 (4.31. ábra)

A, **B** és **C** tároló.

3 mikrolépés:
- **A**, **B** feltöltése,
- **C** feltöltése az **ALU** és a léptető eredménye alapján,
- **C** tárolása regiszterbe.

A 3 mikrolépés egyidejűleg (párhuzamosan) végrehajtható!

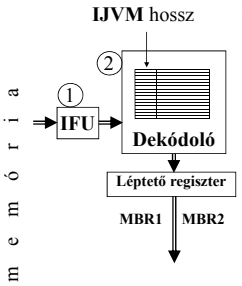
Pl.: a verem két felső szavának cseréje **Mic-3**-on (**4.33. ábra**):

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

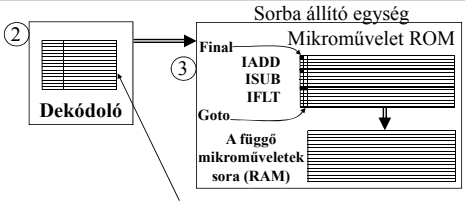
Valódi függőség
RAW – Read After Write!
Elakadás

Hétszakaszú csővezeték: Mic-4 (4.35. ábra)

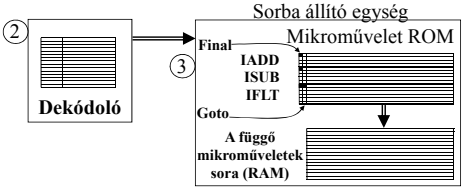
1. Az **IFU** a bejövő bájtfolyamot a dekódolóba küldi.



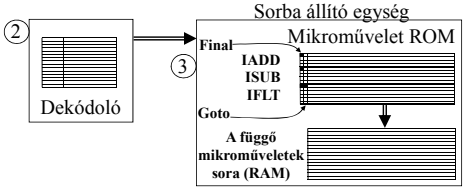
2. A dekódolóban van egy táblázat, amely minden utasításnak tudja a hosszát. A **WIDE** prefixumot felismeri, pl. **WIDE ILOAD** -ot átalakítja **WIDE_ILOAD** -dá: pl. 9 bites utasítás kód.
El tudja különíteni az utasítás kódokat és az operandusokat. Az operandusokat a léptető regiszterbe teszi, onnan tölti fel **MBR1**-et és **MBR2**-t.



A dekódoló egy másik táblázata megmutatja, hogy a sorba állító egységben lévő **ROM** melyik címén kezdődnek a kódhoz tartozó mikroműveletek.
Az egyes **IJVM** utasításokat megvalósító mikroműveletek egymás után vannak a **ROM**-ban, az utolsónál a **Final** be van állítva.



A legtöbb mikroműveletben nincs **NEXT_ADDRESS** és **JAM** mező. Az egyes **IJVM** utasításokat megvalósító mikroműveletek egymás után vannak a **ROM**-ban, az utolsónál a **Final** be van állítva.



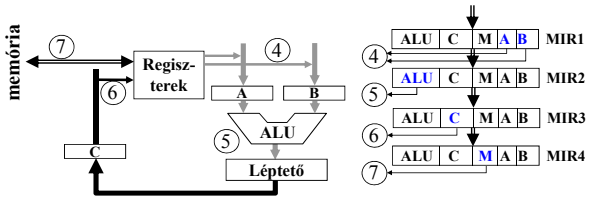
3. A sorba állító egység a **ROM**-ból a **RAM**-ba másolja a mikroműveleteket, amint van hely a **RAM**-ban. A kódhoz tartozó utolsó mikroművelet **Final** bitje jelzi, hogy nincs több átmásolandó mikroművelet.
Ha a mikroműveletek között nem volt olyan, amelyik **Goto** bitje be volt állítva, akkor nyugtázó jelet küld a dekódolónak, hogy folytathatja a munkáját.

Néhány **IJVM** utasítás (pl. **IFLT**) elágazást kíván. A feltételes mikroutasítások speciális utasítások, ezeket külön mikroműveletként kell megadni. Ezeknél be van állítva a **Goto** bit és tartalmazzák a **JAM** biteket is. A **Goto** bit arra szolgál, hogy a sorba állító egység le tudja állítani további utasítások dekódolását. Mindaddig nem lehet tudni, hogy melyik utasítás következik a feltételes utasítás után, amíg a feltétel ki nem értékelődött.

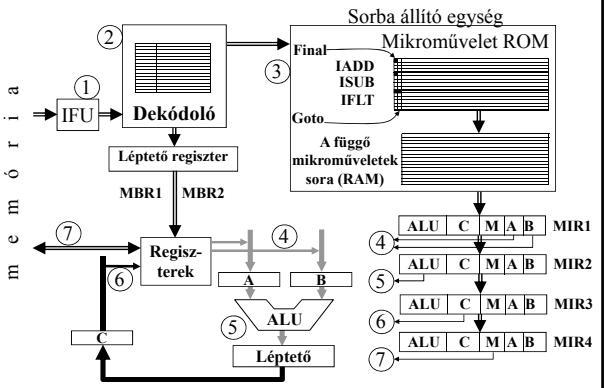
- Ha létrejön az elágazás, akkor a csővezeték nem folytatódhat. „Tiszta lapot” kell csinálni **IFU**-ban, a **dekódolóban** és a sorba állító egységben, majd az ***offset***-nek megfelelő címtől folytatódik a betöltés.
- Ha az ugrás feltétele nem teljesül, akkor a **dekódoló** megkapja a nyugtázó jelet, és a következő utasítással folytatódhat a dekódolás.

Az adatutat 4 független **MIR** vezérli. Minden óraciklus kezdetekor **MIRi** föltöltődik a fölötte lévőből, **MIR1** pedig a **RAM**-ból.

4. **MIR1** az **A**, **B** regiszterek feltöltését,
5. **MIR2** az **ALU** és a léptető működését,
6. **MIR3** az eredmény tárolását,
7. **MIR4** pedig a memória műveleteket vezérli.



Hétszakaszú csővezeték: Mic-4 (4.35. ábra)



IFLT *offset* programozása **Mic-4**-en:

cy	iflt1	iflt2	iflt3	iflt4 (Final=1, Goto=1)	
	MAR=SP=SP-1; rd	OPC=TOS	TOS=MDR	N=OPC; if(N) *GOTO offset*	
1	B=SP				
2	C=B-1	B=TOS			
3	MAR=SP=C; rd	C=B	*Várni kell!*		
4	MDR=mem	OPC=C	*Várni kell!*		
5			B=MDR		
6			C=B	B=OPC	
7			TOS=C	C=B	
8				N PC=PC-1+MBR2; „tiszta lap”, majd a PC által mutatott címtől utasítás betöltés, …	#N MBR2 –t eldobni, folytatódhat a dekódolás

A 8. ciklus feladata túl bonyolult! **MBR2 - 1** előre kiszámítható.

IFLT *offset* programozása **Mic-4**-en:

cy	iflt1	iflt2	iflt3	iflt4	iflt5 (Final=1, Goto=1)	
	MAR=SP=SP-1; rd	OPC=TOS	H=MBR2 -1	TOS=MDR	N=OPC; if(N) *GOTO offset*	
1	B=SP					
2	C=B-1	B=TOS				
3	MAR=SP=C; rd	C=B	B=MBR2			
4	MDR=mem	OPC=C	C=B-1	*Várni kell!*		
5			H=C	B=MDR		
6				C=B	B=OPC	
7				TOS=C	C=B	
8					N PC=PC+H; „tiszta lap”, majd a PC által mutatott címtől utasítás betöltés, …	#N folytatódhat a dekódolás

Az **IJVM** feltétlen ugrását a dekódoló is feldolgozhatja.

Elágazás jövendölés (4.40. ábra)

Legkorábban a dekódoló veheti észre, hogy ugró utasítást kell végrehajtani, de addigra a következő utasítás már a csővezetékben van! Pl.:

Program	Címke	Gépi utasítás	Megjegyzés
`if(i==0)`		`CMP i,0`	összehasonlítás
		`BNE else`	feltételes ugrás
`k=1;`	`then:`	`MOV k,1`	`k=1`
`else`		`BR next`	feltétlen ugrás
`k=2;`	`else:`	`MOV k,2`	`k=2`
	`next:`		

A `BR next` utasítással is probléma van!

Elágazás jövendölés

Eltolás rés (delay slot): Az ugró utasítás utáni pozíció. Az ugró utasítás végrehajtásakor ez az utasítás már a csővezetékben van!

Megoldási lehetőségek:

- **Pentium 4**: bonyolult hardver gondoskodik a csővezeték helyreállításáról
- **UltraSPARC III**: az eltolás résben lévő utasítás végrehajtásra kerül(!). A felhasználóra (fordítóra) bízza a probléma megoldását, a legrosszabb esetben **NOP** utasítást kell tenni az ugró utasítás után.

Elágazás jövendölés

Sok gép megjövendöli, hogy egy ugrást végre kell hajtani vagy sem.

Egy triviális jóslás:

- a visszafelé irányulót végre kell hajtani (ilyen van a ciklusok végén),
- az előre irányulót nem (jobb, mint a semmi).

Feltételes elágazás esetén a gép tovább futhat a jövendölt ágon,

- amíg nem ír regiszterbe vagy
- csak „firkáló” regiszterekbe ír.

Ha a jóslat bejött, akkor minden rendben, ha nem, akkor sincs baj.

Több feltételes elágazás egymás után!

Statikus elágazás jövendölés

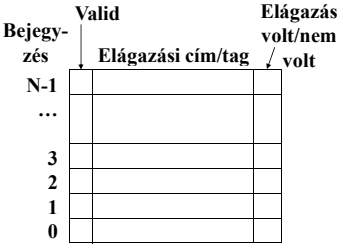
A feltételes utasításoknak néha olyan változata is van (pl. **UltraSPARC III**), mely tartalmaz bitet a jóslásra. A fordító ezt a bitet valahogy beállítja.

Olyankor is statikus elágazás jövendölés történik, ha a processzor arra számít, hogy a visszafelé ugrások bekövetkeznek, az előre ugrások nem.

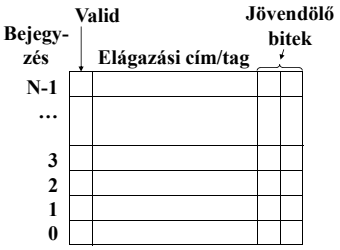
Dinamikus elágazás jövendölés

Elágazás előzmények tábla (**4.41. ábra**), hasonló jellegű, mint a gyorsító tár. Lehet több utas is!

- Egy jövendölő bit: mi volt legutóbb,

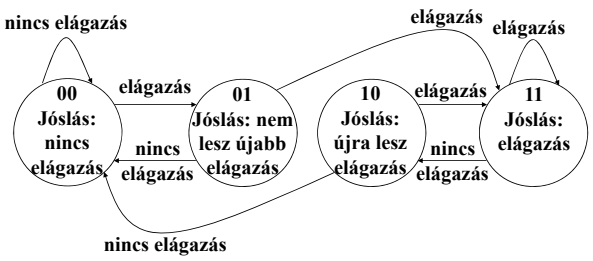


- Két jövendölő bit: mi várható és mi volt legutóbb.

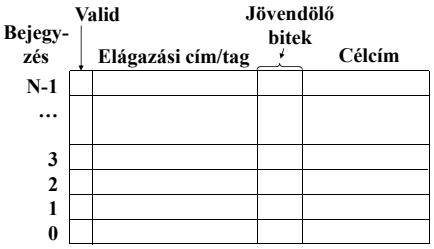


Ha egy belső ciklus újra indul, akkor az várható, hogy a ciklus végén vissza kell ugrani, pedig legutóbb nem kellett.

A „várható” bitet csak akkor írja át, ha egymás után kétszer téves volt a jóslat (**4.42. ábra**).



- A táblázat a legutóbbi célcímet is tartalmazhatja.



Ha az a jövendölés, hogy lesz elágazás, akkor arra számít, hogy a legutóbb tárolt célcímre kell ugrani (ezt persze ellenőrizni kell).

Feladatok

Melyek az **IJVM** feltételes ugró utasításai?

Mire szolgál az **IFLT** utasítás?

Hogy valósítható meg az **IFLT** utasítás?

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?

Feladatok

Mely regiszterek tartalmát kell menteni metódus hívás esetén?

IJVM melyik utasítása szolgál a metódusból való visszatérésre?

Hogy találhatók 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?

Feladatok

Az **IRETURN** utasítás mikroprogramozását úgy is megvalósíthattuk 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?

Elemezze a **4.17. ábra** programjait!

Feladatok

Mi az úthossz?

Milyen lehetőségek vannak a **Mic-1** gyorsítására?

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?

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**?

Feladatok

Hogy ábrázolható véges állapotú géppel (**FSM**) az **IFU** működése?

Mi a különbség a **Mic-2** és **Mic-3** között? Miért eredményez ez gyorsítást?

A **SWAP** utasítás (a verem két felső szavának cseréje) **Mic-3**-on négy mikroutasítással megoldható nyolc mikrolépésben. 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.

Feladatok

Milyen szakaszai vannak a **Mic-4** csővezetékének?

Mi a feladata a dekódoló egységnek?

Mi a feladata a sorba állító egységnek?

Mire szolgál a **Final** bit?

Mire szolgál a **Goto** bit?

Hogy történik **Mic-4**-en az adatút vezérlése?

Miért gyorsabb a **Mic-4**, mint a **Mic-3**?

Milyen speciális feladatokat kell megoldani **Mic-4** esetén a feltételes elágazásnál?

Feladatok

Mit nevezünk elágazás jövendölésnek?

Milyen dinamikus elágazás jövendöléseket ismer?

Milyen statikus elágazás jövendöléseket ismer?

Mi az eltolási rés (**delay slot**)?

Hogy működik az eltolási rés szempontjából a **Pentium** és az **UltraSPARC**?

Az előadáshoz kapcsolódó

Fontosabb tételek

Utasítás betöltő egység. Mic-2.

Csővonalas terv: Mic-3.

Egy hét szakaszú szállítószalag: a Mic-4 csővezetéke

Elágazás, eltolási rés. Statikus és dinamikus elágazás jövendölés