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

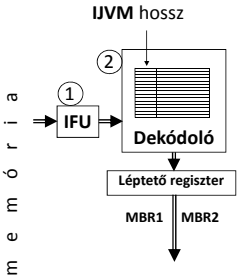
	swap1	swap2	swap3	swap4	swap5	swap6
cy	MAR= SP-1;rd	MAR= SP	H=MDR; wr	MDR= TOS	MAR= SP-1;wr	TOS=H; goto(MBR1)
1	B=SP					
2	C=B-1	B=SP				
3	MAR=C	C=B	*Várni kell!*			
4	rd	MAR=C	*Várni kell!*			
5			B=MDR			
6			C=B	B=TOS		
7			H=C	C=B	B=SP	
8			wr	MDR=C	C=B-1	B=H
9					MAR=C	C=B
10					wr	TOS=C
11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

Valódi függőség
RAW – Read After Write!
Elakadás

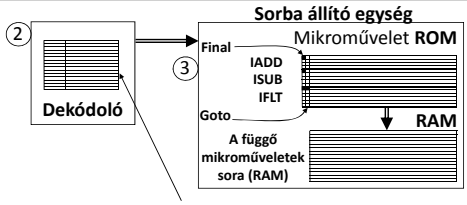
eldugaszolja a csővezetéket!

Hétszakaszú csővezeték: Mic-4 (4.35. ábra)

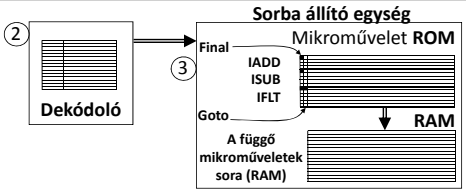
1. Az **IFU** a bejövő bájtfolyamot a dekódolóba küldi.



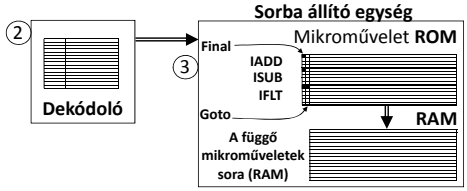
2. A dekódoló a **WIDE** prefixumot felismeri, pl. **WIDE ILOAD** -ot átalakítja **WIDE_ILOAD** -dá: pl. 9 bites utasítás kód.
A dekódolóban van egy táblázat, amely minden utasításnak tudja a hosszát. Ez alapján el tudja különíteni az utasítás kódokat és az operandusokat.
Az operandusokat a léptető regiszterbe 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 (*µ*műveletek).
Az egyes **IJVM** utasításokat megvalósító *µ*műveletek egymás után vannak a **ROM**-ban, az utolsónál a **Final** bit be van állítva.



- A legtöbb *µ*műveletben nincs **NEXT_ADDRESS**.
- A legtöbb *µ*műveletben nincs **JAM** mező.
- A vezérlés átadó *µ*műveletekben a **Goto** bit be van állítva.



3. A sorba állító egység a **ROM**-ból a **RAM**-ba másolja a *µ*műveleteket, amint van hely a **RAM**-ban. A kódhoz tartozó utolsó *µ*művelet **Final** bitje jelzi, hogy nincs több átmásolandó *µ*művelet.
Ha a *µ*műveletek 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 *µ*műveletek speciális utasítások, ezeket külön *µ*műveletként kell megadni. Ezeknél be van állítva a **Goto** bit, tartalmazzák a **JAM** biteket is, továbbá egy a *µ*műveletek **ROM**-jába mutató indexet (**NEXT_ADDRESS**). Ez az index mutat arra a *µ*művelet sorozatra, amelyet a feltétel teljesülése esetén végre kell hajtani.

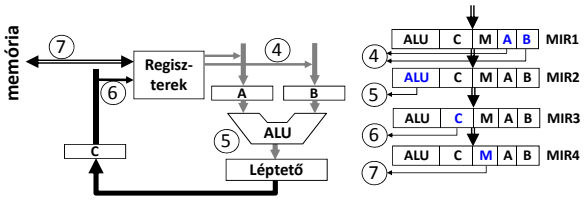
A **Goto** bit arra is szolgál, hogy a sorba állító egység le tudja állítani további utasítások dekódolását. Mindaddig nem lehet tudni, hogy melyik utasítás következik a feltételes utasítás után, amíg a feltétel ki nem értékelődött.

IFLT *offset*

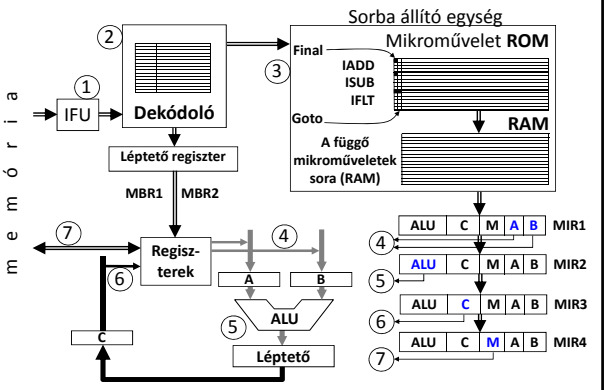
- Ha létrejön az elágazás, akkor a csővezeték nem folytatódhat. „Tiszta lapot” kell csinálni **IFU**-ban, a **dekódolóban**, majd az ***offset***-nek megfelelő címtől folytatódik a betöltés.
- Ha az ugrás feltétele nem teljesül, akkor a **dekódoló** megkapja a nyugtázó jelet, és a következő utasítással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



Az ábra ajánlott

IFLT *offset* programozása **Mic-4**-en, 1. kísérlet:

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

A 9. ciklus feladata túl bonyolult! **MBR2 - 1** előre kiszámítható.

IFLT *offset* programozása **Mic-4**-en, 2. kísérlet:

	iflt1	iflt2	iflt3	iflt4	iflt5 (Final=1, Goto=1)	
cy	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	C=B	B=MBR2			
4	rd	OPC=C	C=B-1	*Várni kell!*		
5			H=C	B=MDR		
6				C=B	B=OPC	
7				TOS=C	C=B	
8					if(N)	
9					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

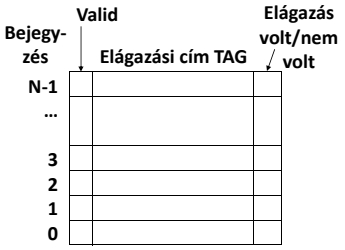
A feltételes utasításoknak néha olyan változata is van (pl. **UltraSPARC III**), mely tartalmaz bitet a jóslásra. A fordító ezt a bitet valahogy beállítja.

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

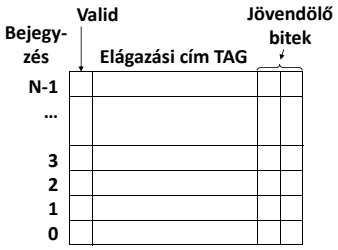
Dinamikus elágazás jövendölés

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

- Egy jövendölő bit: mi volt legutóbb,

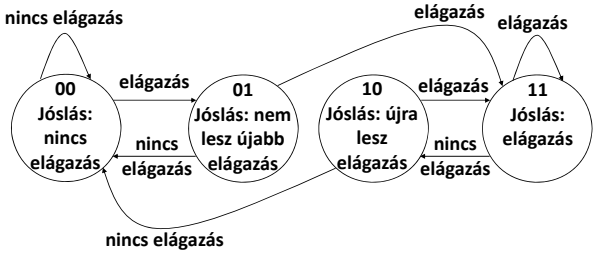


- Két jövendölő bit: mi várható és mi volt legutóbb.



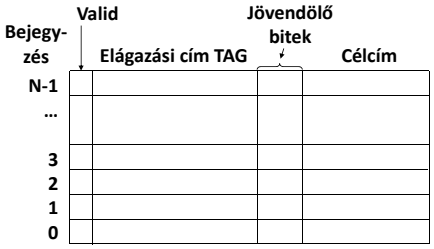
Ha egy belső ciklus újra indul, akkor az várható, hogy a ciklus végén vissza kell ugrani, pedig legutóbb nem kellett.

A „várható” bitet csak akkor írja át, ha egymás után kétszer téves volt a jóslat.



Véges állapotú gép (**FSM**) 2 bites elágazás jövendölésre (**4.42. ábra**).

- A táblázat a legutóbbi célcímet is tartalmazhatja.



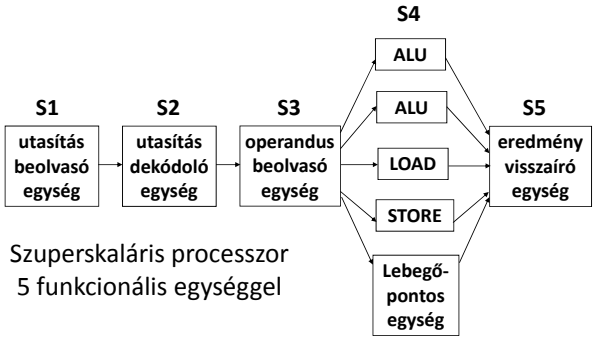
Ha az a jövendölés, hogy lesz elágazás, akkor arra számít, hogy a legutóbb tárolt célcímre kell ugrani.

Elágazás jövendölés

- Figyeljük, hogy az utolsó *k* feltételes elágazást végre kellett-e hajtani. Ez egy *k* bites számot eredményez, és az elágazási előzmények blokkos regiszterében tároljuk. Ha a *k* bites szám megegyezik a táblázat valamely bejegyzésének a kulcsával (találat), akkor az ott talált jövendölést használja.

  Meglepő, de elég jól működik.

Szuperskaláris architektúrák (2. 6. ábra)



Szuperskaláris processzor
5 funkcionális egységgel

Szuperskaláris architektúra esetén a dekódoló egység az utasításokat μutasításokra darabolhatja. Legegyszerűbb, ha a μutasítások végrehajtási sorrendje megegyezik a betöltés sorrendjével, de ez nem mindig optimális.

Függőségek

Ha egy utasítás írni/olvasni akar egy regisztert, akkor meg kell várja azon korábbi utasítások befejezését, amelyek ezt a regisztert írni/olvasni akarták!

Természetesen ugyanez μutasításokra is érvényes.

Függőségek

Egy utasítás nem hajtható végre az alábbi esetekben:

- **RAW** (valódi) függőség (Read After Write):
  Onnan akarunk olvasni operandust,
  ahova még nem fejeződött be egy korábbi írás.
- **WAR** függőség (Write After Read):
  Olyan regiszterbe szeretnénk írni az eredményt,
  ahonnan még nem fejeződött be egy korábbi olvasás.
- **WAW** függőség (Write After Write):
  Olyan regiszterbe szeretnénk írni az eredményt,
  ahova még nem fejeződött be egy korábbi írás.
  Ne boruljon föl az írások sorrendje!

Függőségek: nem olvashatjuk, aminek az írása még nem fejeződött be (**RAW**), és nem írhatjuk felül, amit korábbi utasítás olvasni (**WAR**) vagy írni akar (**WAW**).

Annak nincs akadálya, hogy onnan olvassunk, ahonnan egy korábbi olvasás még nem fejeződött be, tehát az olvasás utáni olvasás nem okoz függőséget. Nincs **RAR** függőség.

Függőségek nyilvántartása: Regiszterenként egy-egy számláló, hogy hányszor használják a végrehajtás alatt lévő utasítások a regisztert olvasásra illetve írásra.

Pl. Tegyük fel, hogy a dekódoló ciklusonként két utasítást tud kiosztani végrehajtásra (a valóságban 4-6 utasítást). Az n. ciklusban dekódolt utasítás végrehajtása legkorábban az (n+1). ciklusban kezdődhet, és az (n+2)-ben fejeződik be, de a szorzás csak az (n+3)-ban.

Az utasítások indítása és befejezése az eredeti sorrendben történjék!

C= ciklus, **K**=kiosztás, **B**=befejezés (**~4.43. ábra**).

					Olvasott regiszterek								Írt regiszterek						
C	#	Dekódolt	K	B	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
1	1	R3=R0*R1	1		1	1										1				
	2	R4=R0+R2	2		2	1	1									1	1			
2	3	R5=R0+R1	3		3	2	1									1	1	1		
	4	R6=R1+R4	-		3	2	1		RAW R4 miatt							1	1	1		
3					3	2	1		I2 csak I1 után fejeződhet be				1	1	1					
4				1	2	1	1		Csak a ciklus végére történik							1	1			
				2	1	1			meg a visszaírás R4-be								1			
				3																
5			4			1			1										1	
	5	R7=R1*R2	5			2	1		1										1	1
6	6	R1=R0-R2	-			2	1		1	WAR R1 miatt									1	1
7				4		1	1													1
8				5																
9			6		1		1						1	0						
	7	R3=R3*R1	-		1		1		RAW R1 miatt				1	0						
10					1		1						1	0						
11				6																
12			7			1		1								1				
	8	R1=R4+R4	-			1		1	WAR R1 miatt							1				
13						1		1								1				
14						1		1								1				
15				7																
16			8						2				1							
17									2				1							
18				8																

megjegyzés *hiba*

					Olvasott regiszterek								Írt regiszterek						
C	#	Dekódolt	K	B	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
9			6		1		1						1	0						
	7	R3=R3*R1	-		1		1		RAW R1 miatt				1	0						
10					1		1						1	0						
11				6																
12			7			1		1								1				
	8	R1=R4+R4	-			1		1	WAR R1 miatt							1				
13						1		1								1				
14						1		1								1				
15				7																
16			8						2				1							
17									2				1							
18				8																

Néhány gép bizonyos (mikro)utasításokat átugorva függőben hagy, előbb későbbi utasításokat hajt végre, és később tér vissza a függőben hagyott utasítások végrehajtására (**~4.44. ábra**).

Az átugrott utasítások is okozhatnak függőséget!

Sorrendtől eltérő végrehajtás
(kezdés és befejezés) esetén (**4.44. ábra**)

					Olvasott regiszterek										Írt regiszterek								
C	#	Dekódolt	K	B	0	1	2	3	4	5	6	7	*1*	*2*	0	1	2	3	4	5	6	7	*1*	*2*
1	1	R3=R0*R1	1		1	1												1						
	2	R4=R0+R2	2		2	1	1											1	1					
2	3	R5=R0+R1	3		3	2	1											1	1	1				
	4	R6=R1+R4	-		3	2	1		0	RAW								1	1	1	0			
3	5	R7=R1*R2	5		3	3	2		0	I5 megelőzi I4-et								1	1	1	0	1		
	6	R1*(S1)*=R0-R2	6		4	3	3		0	WAR: R1 helyett S1								1	1	1	0	1	*1*	
				2	3	3	2		0									1		1	0	1	*1*	

I4 nem indulhat **RAW** függőség (**R4**) **I2** miatt, de **adminisztrációt igényel**, hogy melyik regisztereket használja (függőséget okozhat az átugrott utasítás is!).

I5 megelőzheti **I4** –et.

I6 **R1=R0-R2** helyett ***S1*=R0-R2.** Az ***S1*** segéd (firkáló) regisztert használja **R1** helyett (**regiszter átnevezés**). Az eredményt később átmásolhatja **R1** -be, ha **R1** fölszabadult.

					Olvasott regiszterek										Írt regiszterek								
C	#	Dekódolt	K	B	0	1	2	3	4	5	6	7	*1*	*2*	0	1	2	3	4	5	6	7	*1*	*2*
1	1	R3=R0*R1	1		1	1												1						
	2	R4=R0+R2	2		2	1	1											1	1					
2	3	R5=R0+R1	3		3	2	1											1	1	1				
	4	R6=R1+R4	-		3	2	1		0	RAW								1	1	1	0			
3	5	R7=R1*R2	5		3	3	2		0	I5 megelőzi I4-et								1	1	1	0	1		
	6	R1*(S1)*=R0-R2	6		4	3	3		0	WAR: R1 helyett S1								1	1	1	0	1	*1*	
				2	3	3	2		0									1		1	0	1	*1*	
4			4		3	4	2		1									1		1	1	1	*1*	
	7	R3=R3*R1*(S1)*	-		3	4	2	0	1				*0*					1		1	1	1	*1*	

I6 eredménye **R1** helyett ***S1*****-ben képződik (regiszter átnevezés)!**
A későbbi utasításokban **R1** helyett ***S1*****-et kell használni!**

I7 **RAW** és **WAW** függőség **R3** miatt (**I1**),
RAW függőség **R1** (***S1***) miatt (**I6**),
regiszter átnevezés miatt: **R3=R3*R1** helyett **R3=R3****S1***

C	#	Dekódolt	K	B	Olvasott regiszterek 0	1	2	3	4	5	6	7	1	2	Irt regiszterek 0	1	2	3	4	5	6	7	1	2
1	1	R3=R0*R1	1		1	1												1						
	2	R4=R0+R2	2		2	1	1											1	1					
2	3	R5=R0+R1	3		3	2	1											1	1	1				
	4	R6=R1+R4	-		3	2	1		0									1	1	1	0			
3	5	R7=R1*R2	5		3	3	2		0									1	1	1	0	1		
	6	R1(S1)=R0-R2	6		4	3	3		0									1	1	1	0	1	1	
				2	3	3	2		0									1		1	0	1	1	
4			4		3	4	2		1									1		1	1	1	1	
	7	R3=R3*R1(S1)	-		3	4	2	0	1				0					1		1	1	1	1	
	8	R1(S2)=R4+R4	8		3	4	2	0	3				0					1		1	1	1	1	1
				1	2	3	2	0	3				0					0		1	1	1	1	1
				3	1	2	2	0	3				0					0			1	1	1	1

I8 függőségek: **R1**-et **I1**, **I3** olvassa (**WAR**), ***S1***-be **I6** ír (**WAW**), ***S1***-et **I7** olvassa (**WAR**), ezért **R1=R4+R4** helyett ***S2*=R4+R4** (mostantól **R1** helyett ***S2*** kell).

C	#	Dekódolt	K	B	Olvasott regiszterek 0	1	2	3	4	5	6	7	1	2	Irt regiszterek 0	1	2	3	4	5	6	7	1	2
1	1	R3=R0*R1	1		1	1												1						
	2	R4=R0+R2	2		2	1	1											1	1					
2	3	R5=R0+R1	3		3	2	1											1	1	1				
	4	R6=R1+R4	-		3	2	1		0									1	1	1	0			
3	5	R7=R1*R2	5		3	3	2		0									1	1	1	0	1		
	6	R1(S1)=R0-R2	6		4	3	3		0									1	1	1	0	1	1	
				2	3	3	2		0									1		1	0	1	1	
4			4		3	4	2		1									1		1	1	1	1	
	7	R3=R3*R1(S1)	-		3	4	2	0	1				0					1		1	1	1	1	
	8	R1(S2)=R4+R4	8		3	4	2	0	3				0					1		1	1	1	1	1
				1	2	3	2	0	3				0					0		1	1	1	1	1
				3	1	2	2	0	3				0					0			1	1	1	1
5				6		2	1	0	3				0		0			0		1	1	0	1	
6			7			2	1	1	3				1		0			1		1	1		1	
				4		1	1	1	2				1		0			1			1		1	
				5				1	2				1		0			1					1	
				8				1					1					1						
7		(R1=S2)						1					1					1						
8								1					1					1						
9				7																				

C	#	Dekódolt	K	B	Olvasott regiszterek 0	1	2	3	4	5	6	7	1	2	Irt regiszterek 0	1	2	3	4	5	6	7	1	2
6			7			2	1	1	3				1		0		1				1	1		1
				4		1	1	1	2				1		0		1					1		1
				5				1	2				1		0		1							1
				8				1					1											
7		(R1=S2)						1					1					1						
8								1					1											
9				7																				

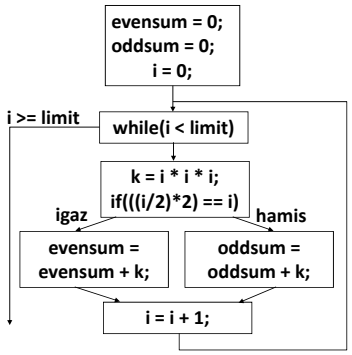
I8 **R1*(S2)*=R4+R4** eredménye a 7. ciklusban átkerülhet ***S2*** –ből **R1**-be, de jobb, ha a hardver nyilvántartja, hogy hol van.

A modern **CPU**-k gyakran titkos regiszterek tucatjait használják regiszter átnevezésre, hogy ezáltal kiküszöböljék a **WAR** és **WAW** függőségeket.

Feltételezett végrehajtás (4.45. ábra)

Páros és páratlan számok köbének összege:

```
evensum = 0;
oddsum = 0;
i = 0;
while(i < limit) {
  k = i * i * i;
  if(((i/2)*2) == i)
    evensum = evensum + k;
  else
    oddsum = oddsum + k;
  i = i + 1;
}
```



Feltételezett végrehajtás (4.45. ábra)
Speculative Execution

Alap blokk (**basic block**): lineáris kód sorozat. Sokszor rövid, nincs elegendő párhuzamosság, hogy hatékonyan kihasználjuk.

Emelés: egy utasítás előre hozatala egy elágazáson keresztül (lassú műveletek esetén nyerhetünk vele). Pl. evensum és oddsum regiszterbe tölthető az elágazás előtt. Az egyik **LOAD** – természetesen – fölösleges.

Ha valamit nem biztos, hogy meg kell csinálni, de nincs más dolga a gépnek, akkor megteheti, de csak „firkáló” regiszterekbe írhat. Ha később kiderül, hogy kell, akkor átírja az eredményeket a valódi regiszterekbe, ha nem kell, elfelejti.

Feltételezett végrehajtás (Speculative Execution)

Mellékhatások:

- fölösleges gyorsító sor csere,
  SPECULATIVE_LOAD: megpróbálja a betöltést a gyorsító tárból. Ha nincs ott, akkor föladja.
- ***if(x>0) z=y/x;*** Ha az osztást a feltétel ellenőrzése előtt végzi el a processzor (emelés), akkor ***x=0*** esetén csapda következik be 0-val való osztás miatt.
  Megoldás: **mérgezés bit**. Emelés esetén a csapda csak akkor következik be, ha az emelt műveletet tényleg végre kell hajtani.

Pentium 4 (2000. november)

Felülről kompatibilis az **I8088**, **…**, **Pentium III**-mal.

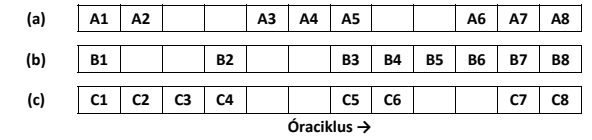
29.000, …, 42 → 55 M tranzisztor, 1,5 → 3,2 GHz, 63-82W, 478 láb (**3. 44. ábra**),
32 bites gép, 64 bites adat sín.

NetBurst architektúra.

2 fixpontos **ALU**. Mindkét **ALU** kétszeres órajel sebességgel fut.

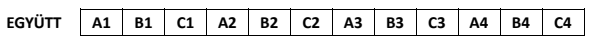
Többszálúság (hyperthreding): 5% többlet a lapkán ~ két **CPU**.

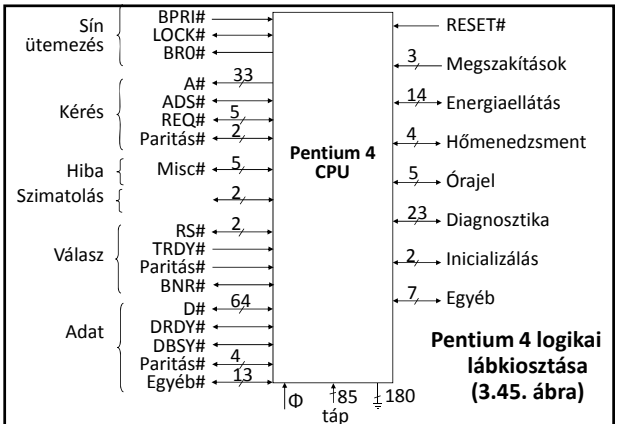
Többszálúság (hyperthreding, 8.7. ábra)



Az (a), (b) és (c) processzus külön futtatva az üres téglalapoknál várakozni kényszerül a memóriához fordulások miatt.

A többszálúság többszörözött regiszter készlet és némi szervező hardver hozzáadásával valósítható meg:





Pentium 4 logikai lábkiosztása (3.45. ábra)

Pentium 4 logikai lábkiosztása (3.45. ábra)

RESET#: a CPU alapállapotba hozatala,

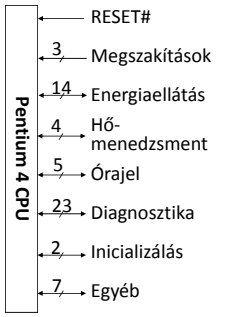
Megszakítások: régi vezérlő, és Advanced Programmable Interrupt Controller (**APIC**)

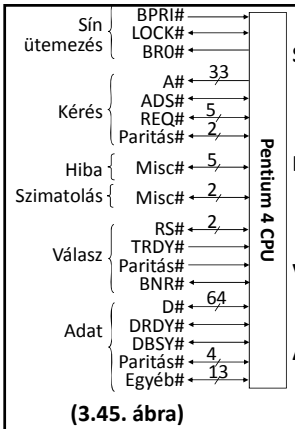
Különböző tápfeszültségek, alvási állapotok,

Jelzés 130^0 fölött, …

Rendszersín frekvenciája,

…





(3.45. ábra)

Pentium 4 logikai lábkiosztása

Sín ütemezés: BPRI#: magas,
BR0#: normál prioritású igény
LOCK#: sín foglalás több ciklusra,

Kérés: A#: 8 bájtos adat címe (64 GB címezhető),
ADS#: a cím érvényes,
REQ#: kívánság,

Válasz: RS#: státus, **TRDY#**: a szolga tud adatot fogadni,
BNR#: WAIT

Adat: D#: 8 bájtos adat,
DRDY#: az adat a sínen van,
DBSY#: a sín foglalt.

Pentium 4

Gépi utasítások → **RISC** szerű μműveletek sorozata, több μművelet futhat egyszerre: szuperskaláris gép, megengedi a sorrenden kívüli végrehajtást is.

Pentium 4

2-3 szintű belső gyorsító tár.

L1: **8 KB** adat, 4 utas halmaz kezelésű, írás átteresztő, 64 bájtos gyorsító sor.
(utasítás) nyomkövető tár akár 12000 dekódolt μművelet tárolására.

L2: **256 KB – 1 MB**, egyesített, 8 utas halmaz kezelésű, késleltetve visszaíró, 128 bájtos gyorsító sor. Előre betöltő egység.

Az Extrem Edition-ban **2 MB** (közös) **L3** is van.

Multiprocesszoros rendszerekhez szimatolás - snoop.

Szimatolás – snoop

Minden processzornak van saját gyorsító tára.

Minden processzor figyeli a sínen a többi processzor memóriához fordulásait (szimatol). Ha valamelyik processzor olyan adatot kér, amely bent van valamely másik processzor gyorsító tárában, akkor ez a másik processzor a saját gyorsító tárából megadja a kért adatot, és letiltja a memóriához fordulást.

Szimatolás – snoop

Ha minden processzornak **saját** írás átteresztő gyorsító tára van (**8.25. ábra**)

Esemény	Saját processzor	Többi processzor
Olvasás hiány	Olvasás a memóriából	Szimatolás
Olvasás találat	Olvasás a gyorsító tárból	
Írás hiány	Írás a memóriába	Ha az írandó szó a gyorsító tárban van, akkor érvényteleníti a gyorsító tár bejegyzést
Írás találat	Írás a gyorsító tárba és a memóriába	

A **Pentium 4** esetén **L1** az írás átteresztő gyorsító tár, a késleltetve visszaíró **L2** a „memória”.

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 függőségnek?

Milyen függőségeket ismer?

Mely függőségek oldhatók fel, és hogyan?

Hogy oldható meg sorrendtől eltérő végrehajtás esetén a függőségek nyilvántartása?

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**?

Mit jelent a sorrenden kívüli végrehajtás?

Mi az előnye a sorrendtől eltérő végrehajtásnak?

Mire szolgál a regiszter átnevezés?

Feladatok

Mi a feltételezett végrehajtás?

Mit nevezünk emelésnek?

Mikor előnyös az emelés?

Milyen mellékhatásai lehetnek a feltételezett végrehajtásnak?

Mi a **SPECULATIVE_LOAD** lényege?

Mi a mérgezés bit?

Mi a többszálúság lényege, haszna?

Mik a többszálúság megvalósításának feltételei?

Mi a szuperskaláris gép lényege?

Feladatok

Hogy érvényesül a **RISC** elv a **Pentium 4** esetén?

Milyen gyorsító tárakat használ a **Pentium 4**?

Jellemezze a **Pentium 4 L2** gyorsító tárát!

Mire szolgál az előre betöltő?

Mit jelent a szimatolás?

Az előadáshoz kapcsolódó

Fontosabb témák

Egy hét szakaszú szállítószalag: a Mic-4 csővezetéke

Elágazás, eltolási rés. Statikus és dinamikus elágazás jövendölés

Sorrendtől eltérő végrehajtás, szuperskaláris architektúra, függőségek, regiszter átnevezés

Feltételezett végrehajtás