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!

Dinamikus elágazás jövendölés

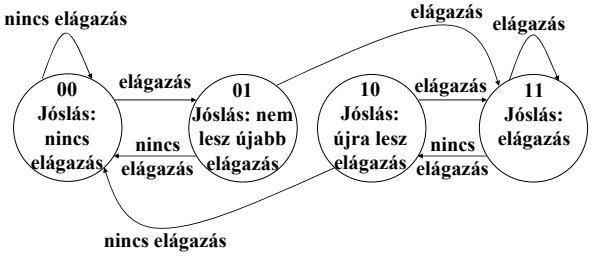
Elágazás előzmények tábla (**4.41. ábra**), hasonló jellegű, mint a gyorsító tár. Lehet több utas is!

- Egy jövendölő bit: mi volt legutóbb,

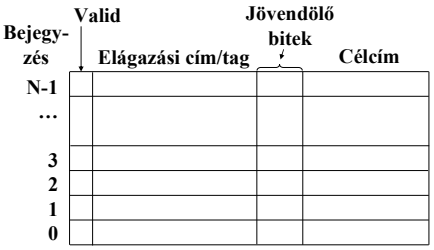
- Két jövendölő bit: mi várható és mi volt legutóbb.

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**).



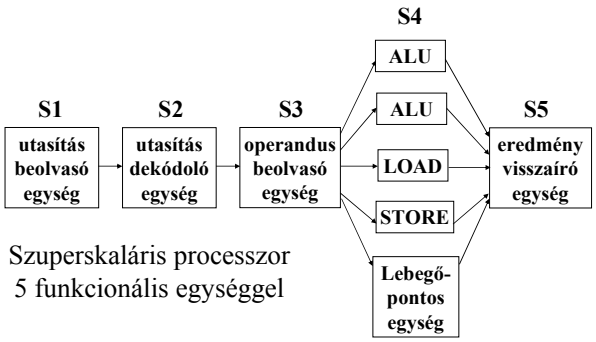
- A táblázat a legutóbbi célcímet is tartalmazhatja.



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

- Figyeljük, hogy az utolsó *k* feltételes elágazást végre kellett-e hajtani. Ez egy *k* bites számot eredményez, ezt 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.

Szuperskaláris architektúrák (2. 6. ábra)



Szuperskaláris processzor
5 funkcionális egységgel

Szuperskaláris architektúra esetén a dekódoló egység az utasításokat mikroutasításokra darabolhatja. Legegyszerűbb, ha a mikroutasítások 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!

Függőségek

Egy utasítás nem hajtható végre az alábbi esetekben:

- **RAW** (valódi) függőség (Read After Write):
  Onnan akarunk olvasni (operandus), 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**). Regiszterenként egy-egy számláló, hogy hányszor használják a végrehajtás alatt lévő mikroutasítások a regisztert olvasásra illetve írásra.

Pl. Tegyük fel, hogy az n. ciklusban dekódolt utasítás végrehajtása legkorábban az (n+1). ciklusban kezdődhet, és a következőben fejeződik be, a szorzás csak két ciklussal később. A dekódoló ciklusonként két utasítást tud kiosztani végrehajtásra (a valóságban 4-6 utasítást).

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										1	1		
				2	1	1	*Csak a ciklus végére történik*											1		
				3			*meg a visszaírás R4-be*													
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				

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

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

C	#	Dekódolt	K	B	Olvasott regiszterek 0	1	2	3	4	5	6	7	*1*	*2*	Írt 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*	

RAW — *I5 megelőzi I4-et* — *WAR: R1 helyett S1*

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 regisztert használja **R1** helyett (**regiszter átnevezés**). Az eredményt később átmásolhatja **R1** -be, ha **R1** fölszabadult.

C	#	Dekódolt	K	B	Olvasott regiszterek 0	1	2	3	4	5	6	7	*1*	*2*	Írt 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*

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***
I8 **WAR** függőség: **R1**-et **I1**, **I3** olvassa, ***S1***-be **I7** ír (**WAW**) 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*	Írt 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				*RAW*	*1*				1		1	1	1	*1*	
	8	R1*(S2)*=R4+R4	8		3	4	2	0	3				*WAR*	*1*				1		1	1	1	*1*	*1*
				1	2	3	2	0	3					*1*				0		1	1	1	*1*	*1*
				3	1	2	2	0	3					*1*				0			1	1	*1*	*1*
5			6			2	1	0	3					*1*		*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*	Írt regiszterek 0	1	2	3	4	5	6	7	*1*	*2*
6			7			2	1	3					*1*		*0*		1				1	1		*1*
				4		1	1	2					*1*		*0*		1					1		*1*
				5				2					*1*		*0*		1							*1*
				8				1					*1*				1							
7		(R1=*S2*)						1					*1*				1							
8								1					*1*				1							
9				7																				

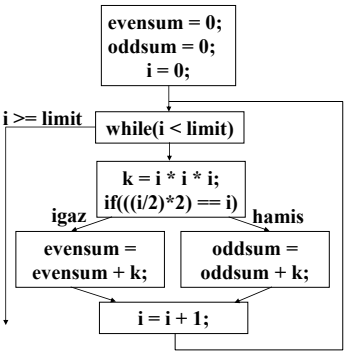
I8 eredménye a 7. ciklusban átkerülhet ***S2*** -ből **R1**-be, de jobb, ha a hardver nyilvántartja, hogy hol van.

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`**
- csapda (pl. ***`x=0`*** esetén ***`if(x>0) z=y/x;`***), **mérgezés bit**.

Pentium 4 (2000. november)

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

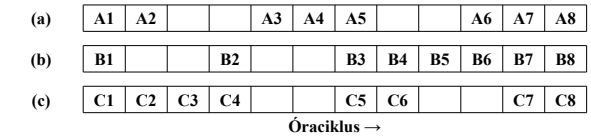
29.000, …, 42 → 55 M tranzisztor, 1,5 → 3,2 GHz, 63-82W, 478 láb (**3. 44. ábra**), 32 bites gép, 64 bites adat sín.

NetBurst architektúra.

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

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

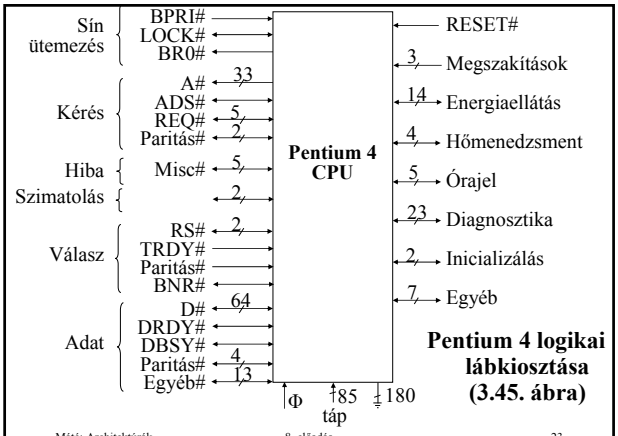
Többszálúság (hyperthreding, 8.7. ábra)



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:

EGYÜTT	A1	B1	C1	A2	B2	C2	A3	B3	C3	A4	B4	C4



Pentium 4 logikai lábkiosztása (3.45. ábra)

Pentium 4 logikai lábkiosztása (3.45. ábra)

RESET#: a CPU alapállapotba hozatala,

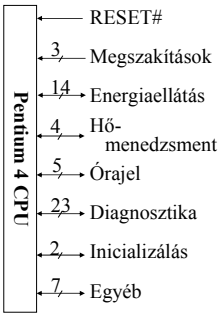
Megszakítások: régi vezérlő, és Advanced Programmable Interrupt Controller (**APIC**)

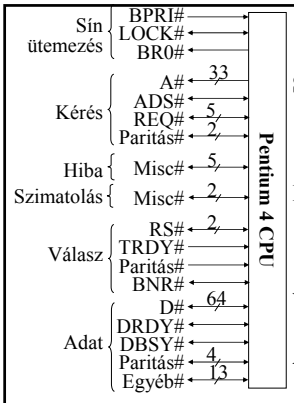
Különböző tápfeszültségek, alvási állapotok,

Jelzés 130^0 fölött, …

Rendszersín frekvenciája,

…





Pentium 4 logikai lábkiosztása (3.45. ábra)

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,

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

Pentium 4

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

2-3 szintű belső gyorsító tár.

L1: **8 KB** utasítás + nyomkövető tár akár 12000 dekódolt mikroutasítás tárolására + **16 KB** adat.

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

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

Esemény	Saját gyorsító tár	Többi gyorsító tár
Olvasás hiány	Olvasás a memóriából	
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 bonyolultabb a helyzet a késleltetve visszaíró **L2** miatt (Házi feladat).

Pentium 4 memória sín

A memóriaigények, tranzakciók 6 állapota: 6 fázisú csővezeték (**3.45. ábra** bal oldal) fázisonként külön vezérlő vonalakkal (amint a mester megkap valamit, elengedi a vonalakat):

0. **Sín ütemezés (kiosztás, bus arbitration)**: eldől, hogy melyik sínmester következik,
1. **Kérés**: cím a sínre, kérés indítása,
2. **Hibajelzés**: a szolga hibát jelez(het),
3. **Szimatolás**: a másik **CPU** gyorsító tárában,
4. **Válasz**: kész lesz-e az adat a következő ciklusban,
5. **Adat**: megvan az adat.

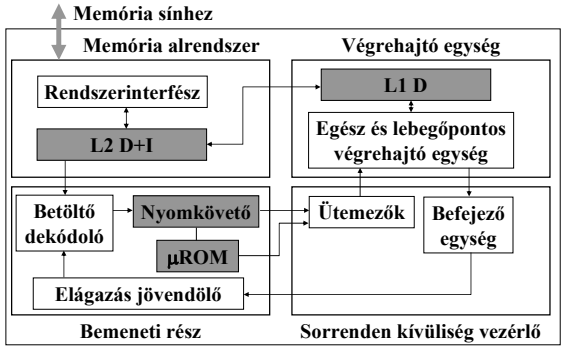
Pentium 4 memória sín csővezetéke (3.46. ábra)

Φ: tranzakció	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}
1	K	H	S	V	A							
2		K	H	S	V	A						
3			K	H	S	V	A					
4				K	H	S	V		A			
5					K	H	S	V		A		
6						K	H	S		V	A	
7							K	H	S		V	A

Ütemezés (nem ábrázoltuk), csak akkor kell, ha másé a sín.

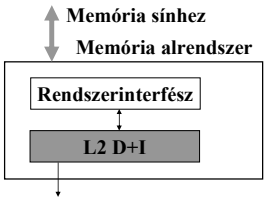
K: kérés, **H**: hiba, **S**: szimatolás, **V**: válasz, **A**: adat

A Pentium 4 mikroarchitektúrája



4.46. ábra. A Pentium 4 blokkdiagramja

4.46. ábra. A Pentium 4 memória alrendszere



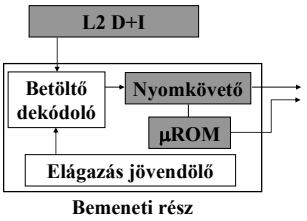
L2 256 KB az első, 512 KB a második, 1 MB a harmadik generációs Pentium 4-ben.

L2 8 utas halmaz kezelésű, késleltetve visszaíró 128 bájtos gyorsító sor, minden második ciklusban kezdődhet egy 64 bájtos feltöltés a memóriából.

Előre betöltő: megpróbálja **L2**-be tölteni azt a gyorsító sort, amelyre majd szükség lesz (nincs az ábrán).

4.46. ábra. A Pentium 4 bemeneti rész

L2-ből betölti és dekódolja a programnak megfelelő sorrendben az utasításokat. Az utasításokat **RISC** szerű mikroműveletek sorozatára bontja. Ha több, mint 4 mikroművelet szükséges, akkor **µROM**-ra történik utalás. A dekódolt mikroműveletek a **Nyomkövető**be kerülnek (nem kell újra dekódolni).



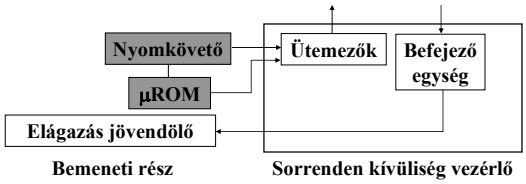
Elágazás jövendölés.

A bemeneti rész az utasításokat **L2**-ből kapja. Ezeket dekódolja, **RISC** szerű mikroműveletekre bontja, a nyomkövető gyorsító tárban tárolja (akár 12 K mikroműveletet) a programnak megfelelő sorrendben.
6 mikroműveletet csoportosít minden nyomkövető sorban.

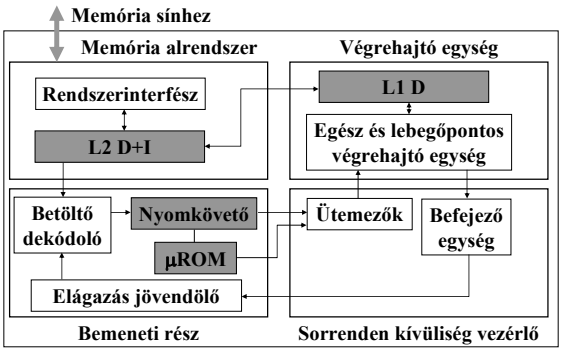
Feltételes elágazásnál az utolsó **4 K** elágazást tartalmazó **L1 BTB**-ből (Branch Target Buffer – elágazási cél puffer) kikeresi a jövendölt címet, és onnan folytatja a dekódolást. Ha az elágazás nem szerepel **L1 BTB**-ben, akkor statikus jövendölés történik: visszafelé ugrást végre kell hajtani, előre ugrást nem.

4.46. ábra. Sorrenden kívüliség vezérlő

Az utasítások a programnak megfelelő sorrendben kerülnek az ütemezőbe, eltérő sorrendben kezdődhet a végrehajtásuk (esetleg regiszter átnevezéssel), de a pontos megszakítás követelménye miatt az előírt sorrendben fejeződnek be.



A Pentium 4 mikroarchitektúrája



4.46. ábra. A Pentium 4 blokkdiagramja

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 nevezünk függőségnek?

Milyen függőségeket ismer?

Mely függőségek oldhatók fel, és hogyan?

Feladatok

Mi az előnye a sorrendtől eltérő végrehajtásnak?

Mire szolgál a regiszter átnevezés?

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?

Feladatok

Mi a többszálúság lényege, haszna?

Mik a többszálúság megvalósításának feltételei?

Hogy érvényesül a RISC elv a Pentium 4 esetén?

Mi a szuperskaláris gép lényege?

Mit jelent a sorrenden kívüli végrehajtás?

Milyen gyorsítótárakat használ a Pentium 4?

Jellemezze a Pentium 4 L2 gyorsítótárát!

Mire szolgál az előre betöltő?

Mit jelent a szimatolás?

Milyen sorrendben dekódolja a Pentium 4 az utasításokat?

Mire szolgál a **µROM**?

Feladatok

Mire szolgál a nyomkövető gyorsítótár?

Milyen elágazás jövendölést használ a Pentium 4?

Mire szolgál az **L1 BTB**?

Mire szolgál a nyomkövető **BTB**?

Milyen sorrendben kezdődik az utasítások végrehajtása a Pentium 4-en?

Az előadáshoz kapcsolódó

Fontosabb tételek

Sorrendtől eltérő végrehajtás, szuperskaláris architektúra, függőségek, regiszter átnevezés

Feltételezett végrehajtás

A Pentium 4 processzor, a Pentium 4 mikroarchitektúrája