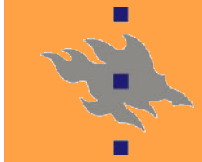
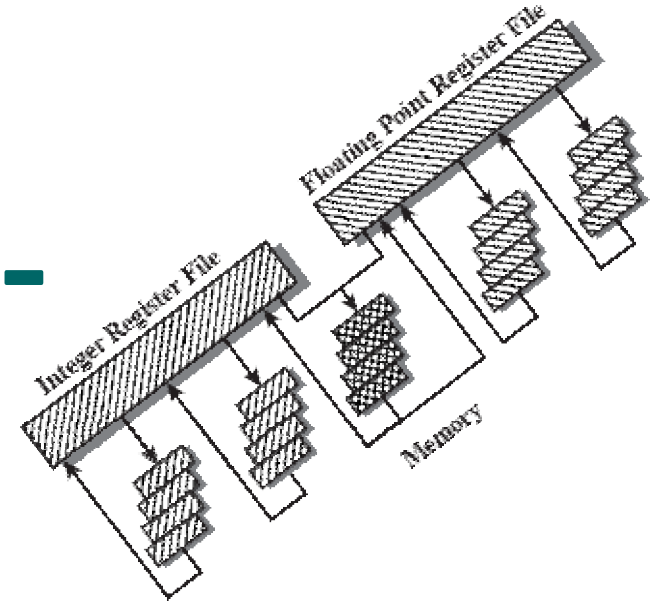


Tietokoneen rakenne

Superskalaari-prosessointi



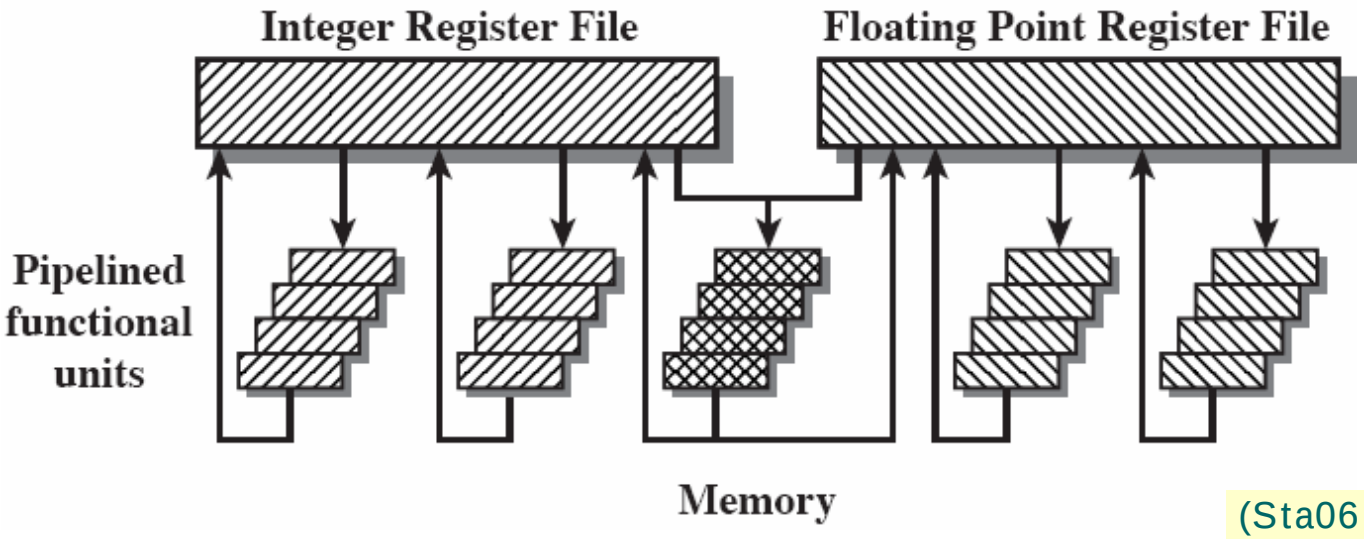
Stallings: Ch 14

- Käskyjen väliset riippuvuudet
- Rekistereiden uudelleennimeäminen
- Pentium / PowerPC

Superskalaariprosessointi

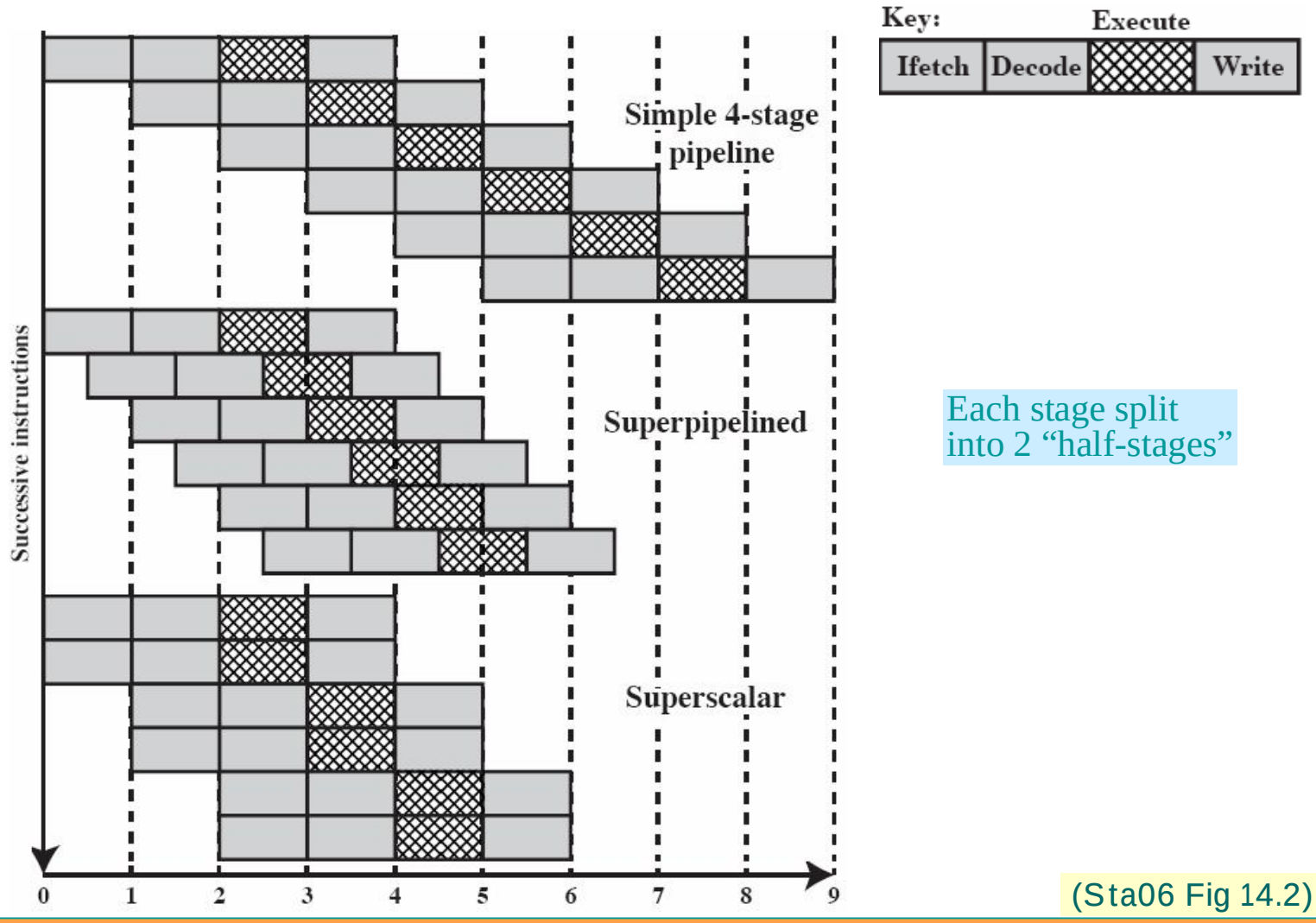
- **Tavoite**
  - Nopeuttaa skalaarikäskyjen prosessointia
- **Useita itsenäisiä liukuhihnoja**
  - Ei siis pelkästään enemmän vaiheita liukuhihnalla

Reference	Speedup
[TJAD70]	1.8
[KUCK72]	8
[WEIS84]	1.58
[ACOS86]	2.7
[SOHI90]	1.8
[SMIT89]	2.3
[JOUP89b]	2.2
[LEE91]	7



(Sta06 Fig 14.1, Tbl 14.1)

Superskalaariprosessointi



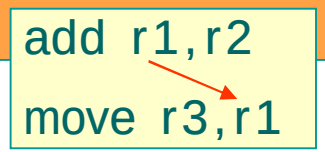
Each stage split into 2 “half-stages”

(Sta06 Fig 14.2)

Superskalaariprosessointi

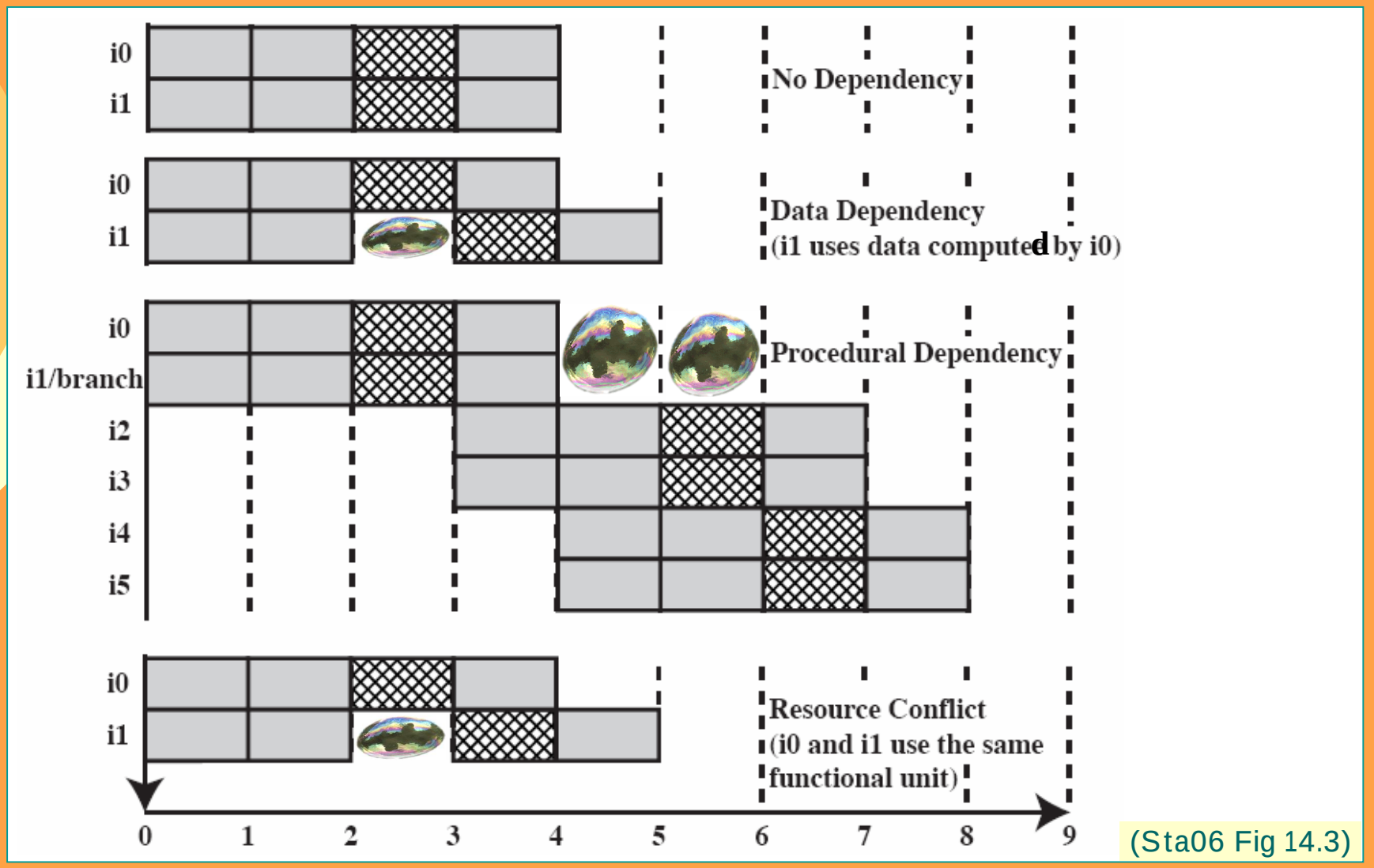
- **Muistin käytön oltava tehokas**
  - Nouda useita käskyjä yhtäaikaa, ennaltanouto
  - Datan nouto / talletus
  - Rinnakkaisuus
- **Saman prosessin useita käskyjä yhtäaikaa suorituksessa eri liukuhihnoilla**
  - Valitse noudetuista sopivassa järjestyksessä suoritukseen (in-order issue/out-of-order issue)
- **Enemmän kuin yksi käsky valmistuu per sykli**
  - Saattavat valmistua eri järjestyksessä kuin aloitettu (out-of-order completion)
- **Milloin käsky saa valmistua ennen edeltävää käskyä?**

Tutut riippuvuudet



- **Datariippuvuus** (True Data/Flow Dependency)
  - write-read (Read after Write, RaW) riippuvuus
  - Jäljempi käsky tarvitsee edeltävän tuottamaa dataa
- **Kontrolliriippuvuus** (Procedural/Control Dependency)
  - Hyppyä seuraavat käskyt suoritetaan vain, jos hyppy ei toteudu
  - Superskalaarihihnalla hukattavana enemmän käskyjä
  - Vaihtelevanpituisten käskyjen lisäosista tietoa vasta suoritettaessa
- **Resurssiriippuvuus** (Resource Conflict)
  - Yksi tai useampi liukuhihna osa tarvitsee samoja resursseja
  - Muistipuskuri, ALU, pääsy rekisterijoukkoon, ...

Tutut riippuvuudet



(Sta06 Fig 14.3)

Uudet riippuvuudet

Nimiriippuvuudet =

- **Kirjoitusriippuvuus** (Output Dependency)
  - write-after-write (WaW) riippuvuus
  - Kun kaksi käskyä muuttaa samaa rekisteriä tai muistipaikan sisältöä, niin alkuperäisessä koodissa jäljempänä oleva talletus jäätävä voimaan

```
load r1,X
add r2,r1,r3
add r1,r4,r5
```

```
move r2,r1
add  r1,r4,r5
```

- **Antiriippuvuus** (Antidependency, Read-write dependency)
  - Write-after-read (WaR) riippuvuus
  - Edeltävän käskyn ehdittävä noutaa rekisterin tai muistipaikan sisältö, ennenkuin jäljempi käsky tallettaa sinne uuden arvon
- **Aliakset?**
  - Esim. Kaksi rekisteriä osoittaa epäsuorasti samaan muistipaikkaan

40(R1) vs. 0(R2)

Kuinka käsitellä riippuvuudet?

- **Peruslähtökohta**
  - Kaikki riippuvuudet käsitellään jollain tavoin
- **Perusratkaisu (kuten ennenkin)**
  - laitteisto huomaa riippuvuuden,
    pysäyttää liukuhihnan odottamaan (bubble)
- **Ratkaisu 2**
  - Kääntäjä generoi käskyt sellaisessa järjestyksessä, että riippuvuuksia ei tule
  - Tällöin ei tarvita erityislaitteistoa
    - Yksinkertaisempi suoritin, jonka ei tarvitse havaita riippuvuuksia
  - Kääntäjän laatijan tunnettava kohdeprosessorin toiminta tarkoin

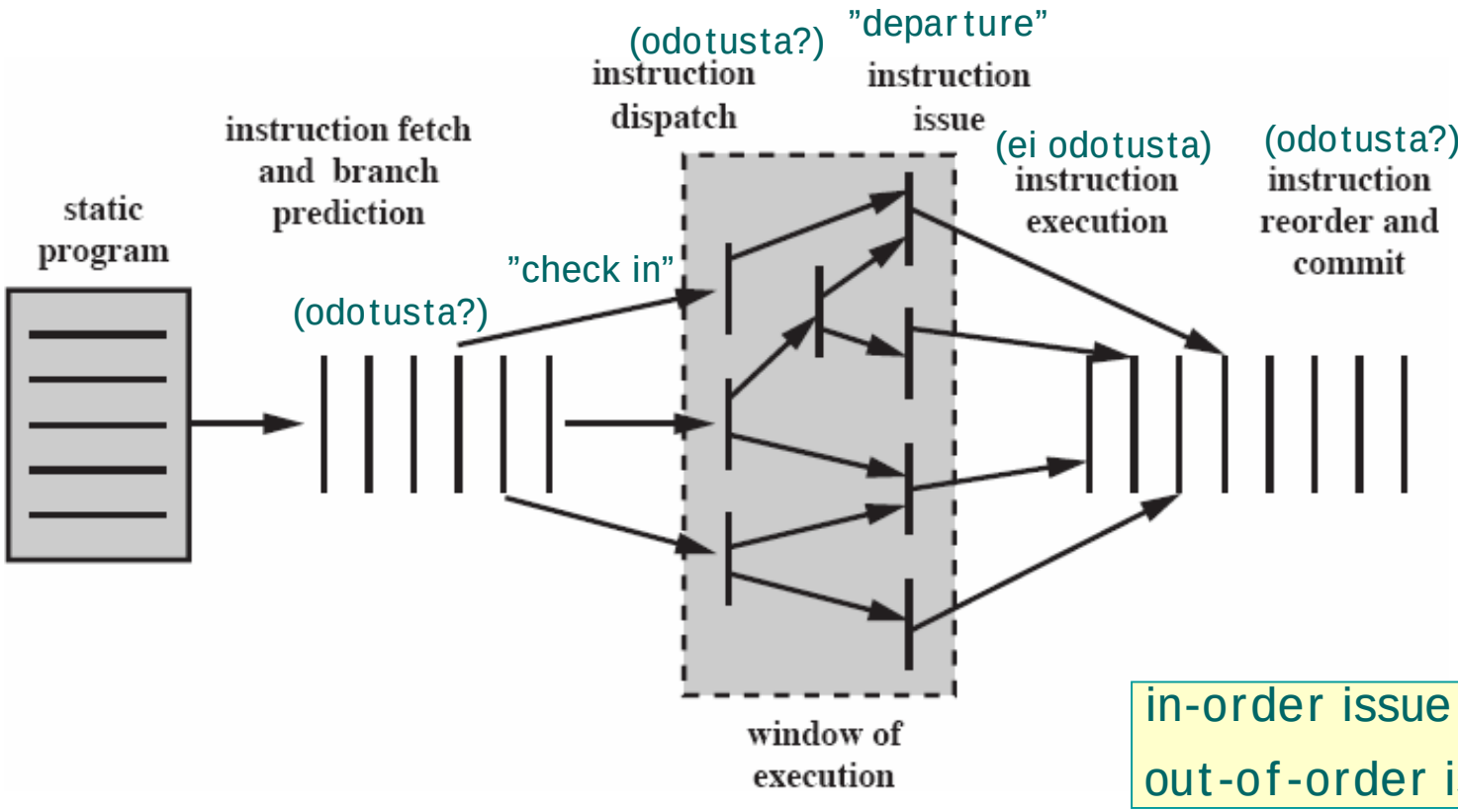
Rinnakkaisuus

- **Käskytason rinnakkaisuus** (Instruction-level parallelism)
  - Montako käskyä pystyttäisiin teoreettisesti suorittamaan rinnakkain
    - Riippuu suoritettavasta koodista
- **Konetason rinnakkaisuus** (Machine parallelism)
  - Todellinen rinnakkaisuus
  - Mitä tietty kone tai arkkitehtuuri todella voi tehdä rinnakkain
    - Montako käskyä voi hakea yhtäaikaa?
    - Montako käskyä voi suorittaa yhtäaikaa?
  - ~ Montako liukuhihnaa käytettävissä
  - Aina pienempi kuin käskytason rinnakkaisuus
    - Riippuvuudet?
    - Huonosti optimoitu toteutus?

```
load r1 • r2
add r3 • r3+1
add r4 • r4, r2
```

```
add r3 • r3+1
add r4 • r3, r2
load r0 • r4
```

Superskalaariprosessointi



in-order issue vs. out-of-order issue

in-order complete vs. out-of-order complete

issue ~ laukaisu, liikkeellelaskeminen

dispatch ~ vuorottaminen, lähettää suorittamaan

(Sta06 Fig 14.6)

Superskalaariprosessointi

- **Käskyjen nouto muistista**
  - Hyppyjen ennustus ž ennaltanouto muistista CPU:hun
  - Valintaikkuna (window-of–execution)
    ~ Muistista noudetut käskyt
- **Käskyn päästäminen liukuhihnalle** (dispatch/issue)
  - Selvitä data-, kontrolli- ja resurssiriippuvuudet
  - Uudelleenjärjestele, päästä sopivat liukuhihnoille
  - Valittujen päästävä etenemään ilman odotusjaksoja
  - Jos sopivaa ei löydy, odotuta tässä kohtaa
- **Kun suoritus valmistuu** (complete, retire)
  - Hyväksy tai hylkää (commit/abort)
  - Selvitä kirjoitus- ja antiriippuvuudet
    ž odota / järjestä uudelleen (reorder)

In-order issue, in-order complete

- Se perinteinen peräkkäisjärjestys
- Ei käyttöä valintaikkunalle
- Käskyjä hihnoille vain alkuperäisessä järjestyksessä
  - Kääntäjä huolehtinut pääosin riippuvuuksista
  - Tarkista silti riippuvuus edeltäjistä
  - Tsekkaa etenemisvauhti, jätä tarvittaessa kuplia
  - Voi päästää useita yhtäaikaakin baanalle
- Valmistuminen vain alkuperäisessä järjestyksessä
  - Viereisellä baanalla ei saa ohittaa
  - Useita voi valmistua yhtäaikaa
  - Commit/Abort

In-order issue, in-order complete

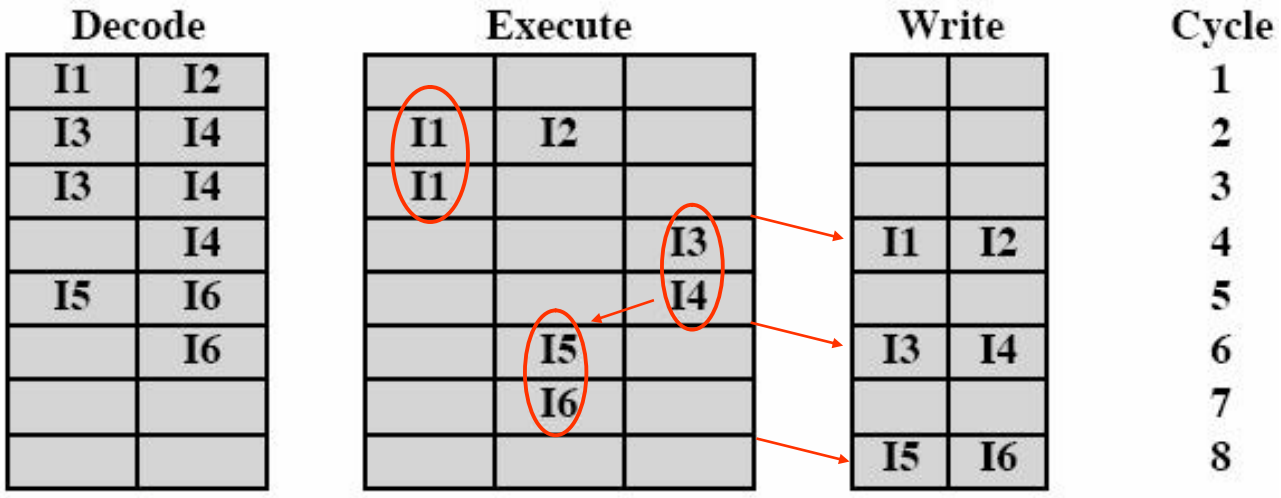
Nouto 2 käskyä kerralla

I1 tarvitsee suoritukseen 2 sykliä

I3 ja I4: resurssiriippuvuus

I5 (käyttää) ja I4 (tuottaa): datariippuvuus

I5 ja I6: resurssiriippuvuus



(Sta06 Fig 14.4a)

In-order issue, out-of-order complete

- **Kuten edellinen, mutta**
  - Salli valmistua eri järjestyksessä kuin käskyjä aloitettu
  - Huolehdi kirjoitus- ja antiriippuvuudesta ennen tulosten kirjoittamista

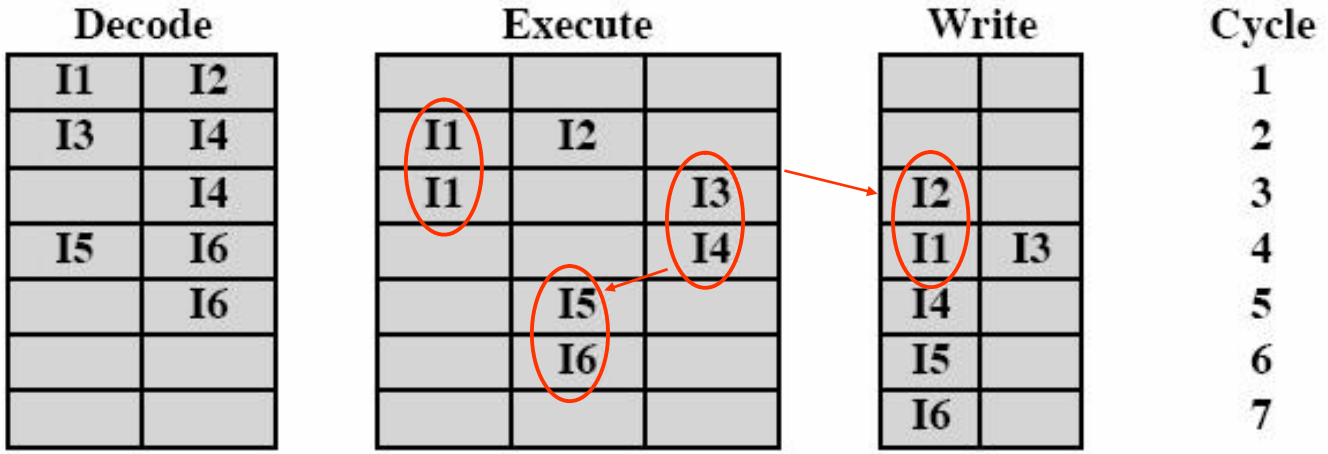
Nouto 2 käskyä kerralla

I1 tarvitsee suoritukseen 2 sykliä

I3 ja I4: resurssiriippuvuus

I5 (käyttää) ja I4 (tuottaa): datariippuvuus

I5 ja I6: resurssiriippuvuus



(Sta06 Fig 14.4b)

Out-of-order issue, out-of-order complete

- Päästä käskyjä liikkeelle parhaaksi katsotussa järjestyksessä
  - Tarvitaan valintaikkuna
- Salli valmistuminen parhaaksi katsotussa järjestyksessä
  - Huolehdi kirjoitus- ja antiriippuvuudesta

Out-of-order issue, Out-of-order complete

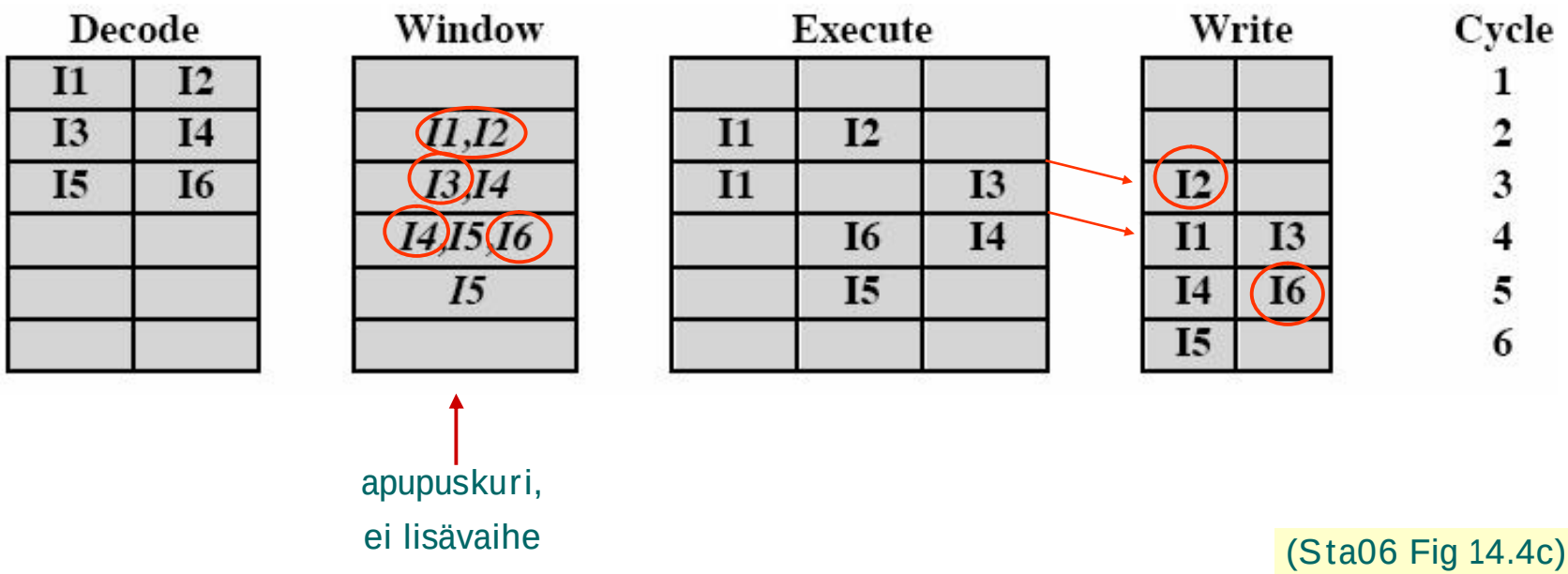
Nouto 2 käskyä kerralla

I1 tarvitsee suoritukseen 2 sykliä

I3 ja I4: resurssiriippuvuus

I5 (käyttää) ja I4 (tuottaa): datariippuvuus

I5 ja I6: resurssiriippuvuus



(Sta06 Fig 14.4c)

Rekistereiden uudelleennimeäminen

- **Ongelman syynä usein se, että toisistaan riippumattomille asioille käytetty samaa rekisteriä**
  - Käskyjen välille syntyy riippuvuus, tarpeettomasti
  - Odoteltava, että edellinen valmistuu
- **Ratkaisu: Rekistereiden uudelleennimeäminen**
  - Laitteistossa enemmän rekistereitä kuin ohjelmoijalle näkyy
  - Laitteisto allokoi todelliset rekisterit suoritusaikana
  - Allokointi s.e. vältetään nimiriippuvuus
- **Tarvitaan**
  - Enemmän sisäisiä työrekistereitä (rekisterijoukot), esim. Pentium II:ssa 40 työrekisteriä
  - Laitteistoa, joka allokoi työrekistereitä ja pitää kirjaa ohjelmoijalle näkyvistä rekistereistä

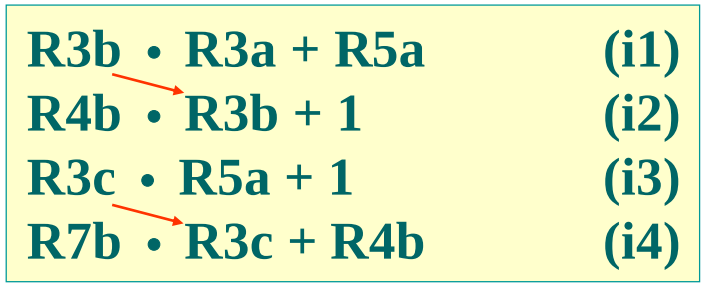
Rekistereiden uudelleennimeäminen

Kirjoitusriippuvuus (WaW):
i3 ei saa valmistua ennen i1:stä

Antiriippuvuus (RaW):
i3 ei saa valmistua ennenkuin i2 lukenut arvon R3:sta

Uudelleennimeä R3 s.e. käytössä työrekisterit R3a, R3b, R3c
Muut rekisterit vastaavasti:
R4b, R5a, R7b

Ei enää nimiriippuvuuksia!

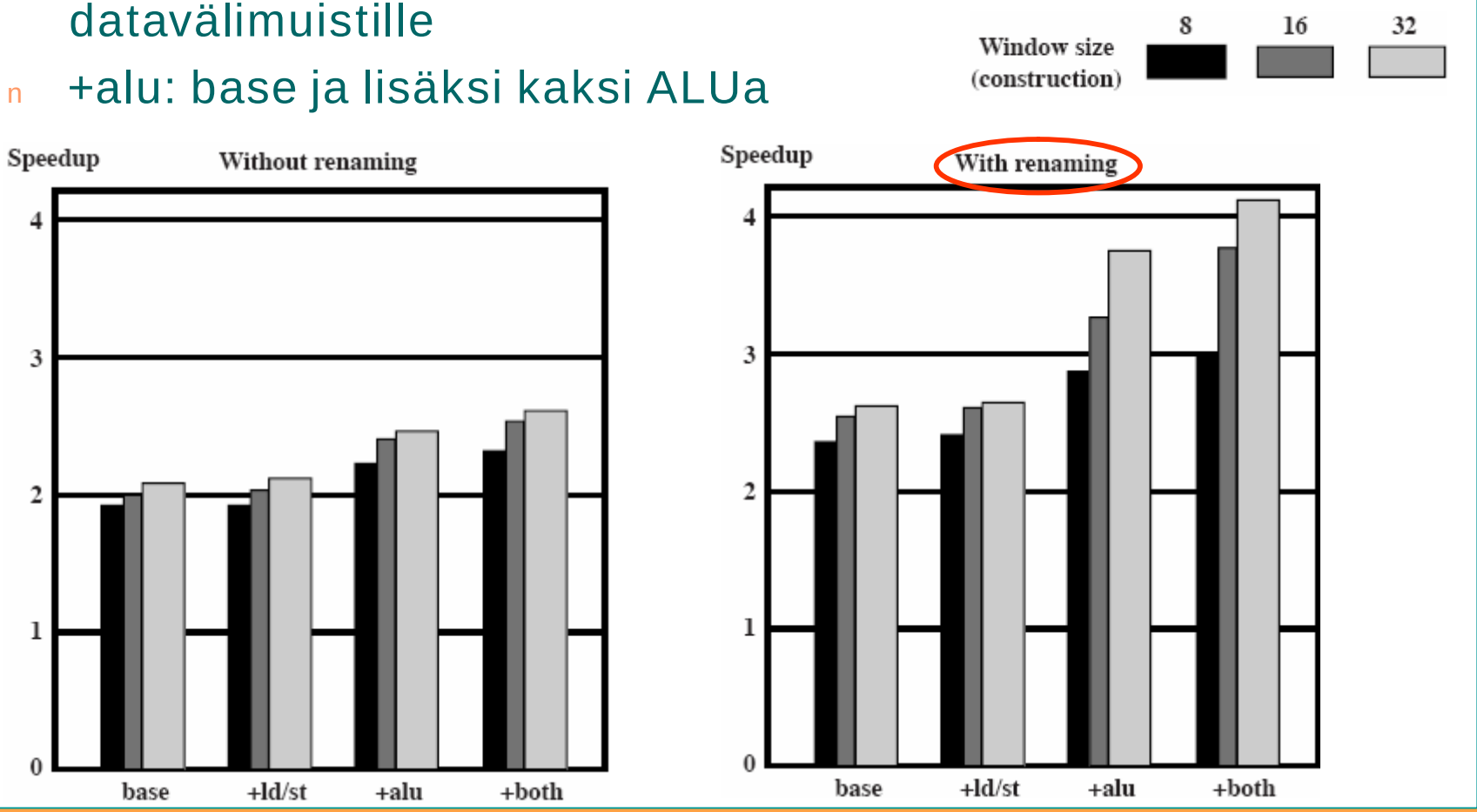


Miksi R3a ja R3b?

Lisälaitteiston vaikutus

- base: out-of-order issue
- +ld/st: base ja lisäksi kaksi load/store yksikköä datavälimuistille
- +alu: base ja lisäksi kaksi ALUa

(Sta06 Fig 14.5)



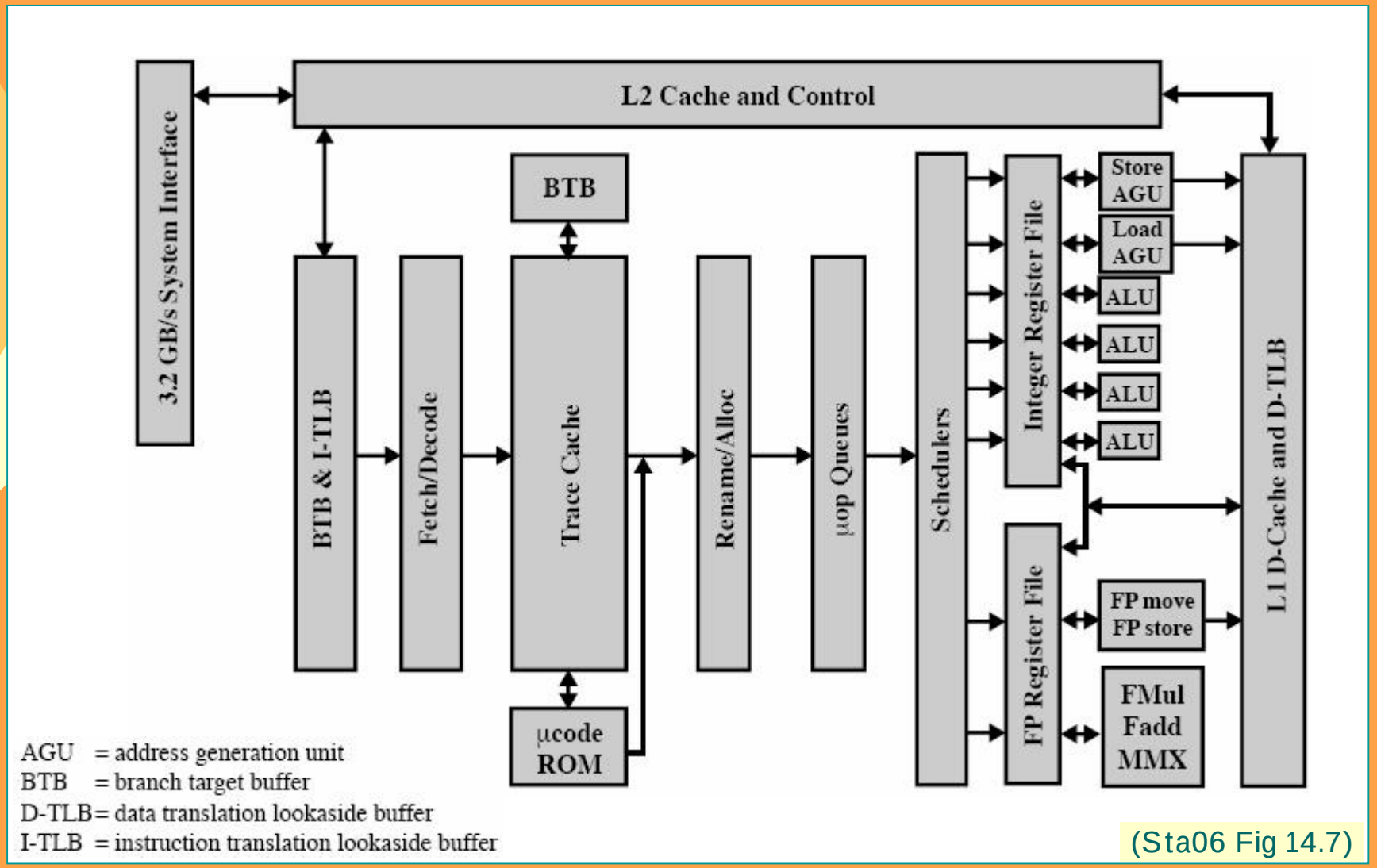
Yhteenveto

- **Useita toiminnallisesti itsenäisiä yksikköjä**
- **Muistihierarkian tehokas käyttö**
  - Sallii useita rinnakkaisia muistinoutoja/talletuksia
- **Käskyjen ennaltanouto**
  - Hyppyjen ennustuslogiikka
- **Laitetason logiikka riippuvuuksien huomaamiseksi**
  - Ohituspiirit, joilla tieto heti suoraan toiselle yksiköllä samaan aikaan kuin tulos rekisteriin tai muistiin
- **Laitetason logiikka useiden riippumattomien käskyjen liikkeellesaattamiseksi (issue)**
  - Riippuvuudet ž järjestys
- **Laitetason logiikka huolehtii käskyjen oikeasta valmistumisjärjestyksestä (completion)**
  - Riippuvuudet ž commit

Sta06 Fig 14.6

Pentium 4

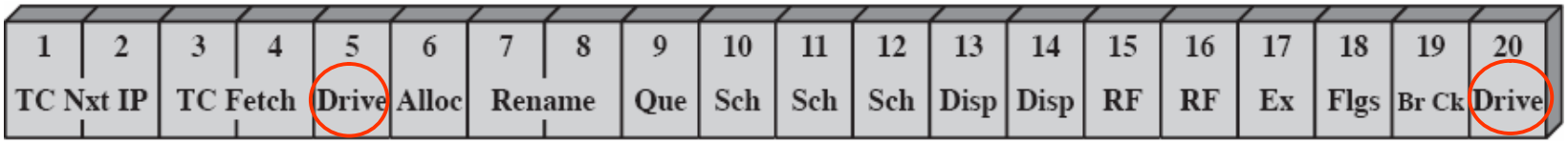
Pentium 4



(Sta06 Fig 14.7)

Liukuhihna

- Ulospäin CISC -käskykanta (IA-32)
- Suoritus kuitenkin mikro-operaatioina kuten RISC
  - Nouda CISC-käsky ja muodosta siitä mikro-operaatiot (µops) L1 tason välimuistiin (trace cache)
  - Hihnan loppuosa operoi vakiopituisilla µ-operaatioilla (118b)
- Pitkä liukuhihna
  - Lisävaiheet (5 ja 20) signaalien etenemisviipeen vuoksi



TC Next IP = trace cache next instruction pointer
TC Fetch = trace cache fetch
Alloc = allocate
Rename = register renaming
Que = micro-op queuing
Sch = micro-op scheduling
Disp = Dispatch
RF = register file
Ex = execute
Flgs = flags
Br Ck = branch check

(Sta06 Fig 14.8)

Mikro-operaatioiden generointi

Sta06 Fig 14.9a-f

a) Nouda IA-32 käsky L2 välimuistista ja generoi mikro-operaatiot L1 tason välimuistiin (trace cache)

- Käskyille oma TLB, hyppyjen kohteille oma BTB
- Staattinen hyppyjen ennustus
  - taaksepäin ”taken”, eteenpäin ”not taken”
- 1-4 µops per käsky, monimutkaisemmat ROM-muistissa

b) Määritä Trace-Cache-IP:n arvo mikro-operaatiolle

- Dynaaminen hyppyjen ennustus (4-bit)
- 512 alkion joukkoassosiatiivinen kohdepuskuri BTB (branch target buffer), joukon koko 4

c) Nouda operaatio L1 tason välimuistista

d) Drive – odotusjakso

Resurssien allokointi

Sta06 Fig 14.9a-f

e) Allokoi resurssit, rekistereiden uudelleennimeäminen

- 3 operaatiota per sykli
- Varaa alkio uudelleenjärjestelypuskurista (126:sta) (reorder buffer, ROB)
- Varaa tulokselle yksi 128 sisäisestä työrekisteristä ja mahd. yksi load ja yksi store puskuri (48:sta ja 24:stä)
- Poista nimiriippuvuuksia rekistereiden uudelleennimeämisellä
- Allokoi alkio mikro-operaatioiden valintajonosta
- Jos ei vapaita resursseja, odota (ž out-of-order)

- **ROB-alkiossa kirjanpito operaation etenemisestä hihnalla**
  - Mikro-operaatio, ja alkuperäisen IA-32 käskyn osoite
  - State: scheduled, dispatched, completed, ready
  - Register Alias Table (RAT):
    mikä IA-32 rekisteri ž mikä työrekisteri

Window of Execution

Sta06 Fig 14.9a-f

f) 2 FIFO jonoa mikro-operaatioiden valitsemiseksi

- Tarvittavat resurssit saatavilla, ei riippuvuutta
- Muistiin viittaaville oma, muille oma

g) Mikro-operaatioiden nouto jonoista (scheduling)

Sta06 Fig 14.9g-l

h) Päästäminen liukuhihnalle (dispatching)

- Tutki FIFO-jonojen keulimmaisten ROB-alkioita
- Jos tarvittava suoritusyksikkö vapaa, operaation voi päästää suoritusliukuhihnalle
- Kaksi jonoa ž out-of-order issue
- max 6 mikro-op liikkeelle yhden syklin aikana
  - ALU:ille tai FPU:ille kummallekin max 2 per sykli
  - Load ja store -yksiköille kummallekin max 1 per sykli

Integer ja FP yksiköt

Sta06 Fig 14.9g-l

i) Nouda data rekistereistä tai välimuistista

j) Suorita käsky, aseta lipukkeet

- Hae operandit rekistereistä / L1 välimuistista
- Useita liukuhihnoitettuja suoritusyksiköitä
  - 2 * Alu, 2 * FPU, 2 * load/store
  - Esim. nopea ALU helpoille, kertolaskuille oma ALU
- Tulosten talletus: in-order complete
- Päivitä ROB, salli uusien operaatioiden tulo hihnalle

k) Tarkista miten hyppykäskyssä kävi

- Menikö niinkuin ennustettiin?
- Abortoi väärät käskyt hihnalta (estä tulosten talletus)

l) Kirjaa hypyn tulos ennustuslogiikkaa varten

- Anna aikaa signaalien etenemiseksi

Pentium 4 Hyperthreading

- Yksi fyysinen IA-32 CPU, mutta 2 loogista CPU:ta
- Näkyy KJ:lle kahden CPU:n SMP-järjestelmänä
  - Molemmat suorittavat eri prosesseja, tai saman prosessin eri säikeitä (kuten SMP)
  - Ei tarvitse huomioida kooditasolla
  - KJ:n osattava SMP-temput (mm. vuorottaminen)
- Perustuu CPU:n odotussyklien hyötykäyttöön
  - Muistiinviittaus (cache miss)
  - Riippuvuudet, väärä hyppyennustus
- Jos toinen käyttää FP-yksikköä, toinen voi käyttää INT-yksikköä
  - Hyödyt pitkälti sovellusriippuvia

Pentium 4 Hyperthreading

- **Kahdennettu**
  - IP, EFLAGS ja muut kontrollirekisterit
  - KäskyTLB
  - Rekistereiden uudelleennimeämislogiikka

Sta06 Fig 14.7

Sta06 Fig 14.8

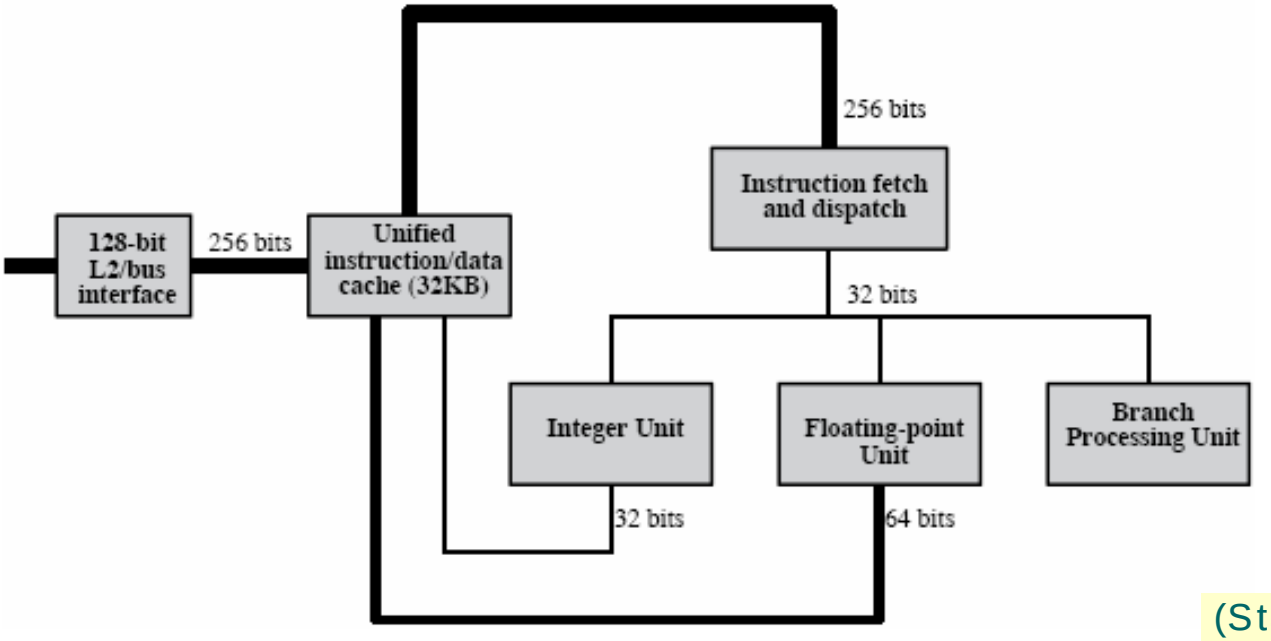
- **Puolitettu**
  - Kristillinen tasajako, ei monopolia
  - Uudelleenjärjestelypuskurit (ROB)
  - Mikro-operaatioiden valintajonot (2 keulaa?)
  - Load/store puskurit
- **Yhteiskäytössä**
  - Rekisterijoukot (128 GPRs, 128 FPRs)
  - Välimuistit: trace cache, L1, L2, L3
  - Mikro-operaatioiden suorituksessa tarvittavat rekisterit
  - Suoritusyksiköt: 2 ALUa, 2 FPUta, 2 ld/st-units

Tietokoneen rakenne

PowerPC

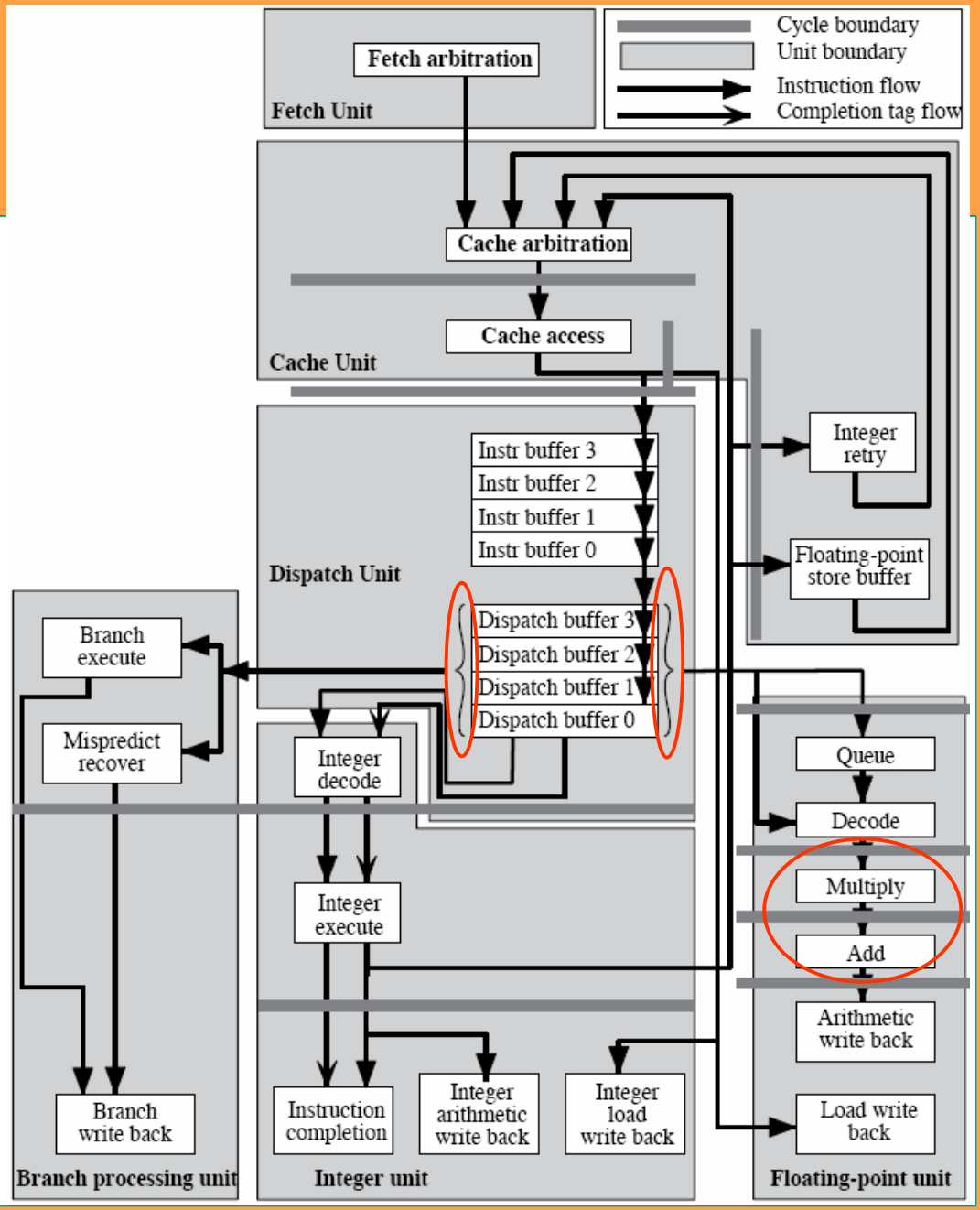
PowerPC 601

- Käskyjen noutoyksikkö
  - Voi noutaa 8 käskyä (a’ 32b) kerralla välimuistista
- Käskyjen valinta suoritukseen (dispatch)
- 3 käskyjä suorittavaa yksikköä
  - Kokonaisluvut, liukuluvut, hypyt



(Sta06 Fig 14.10)

PowerPC 601 Pipeline



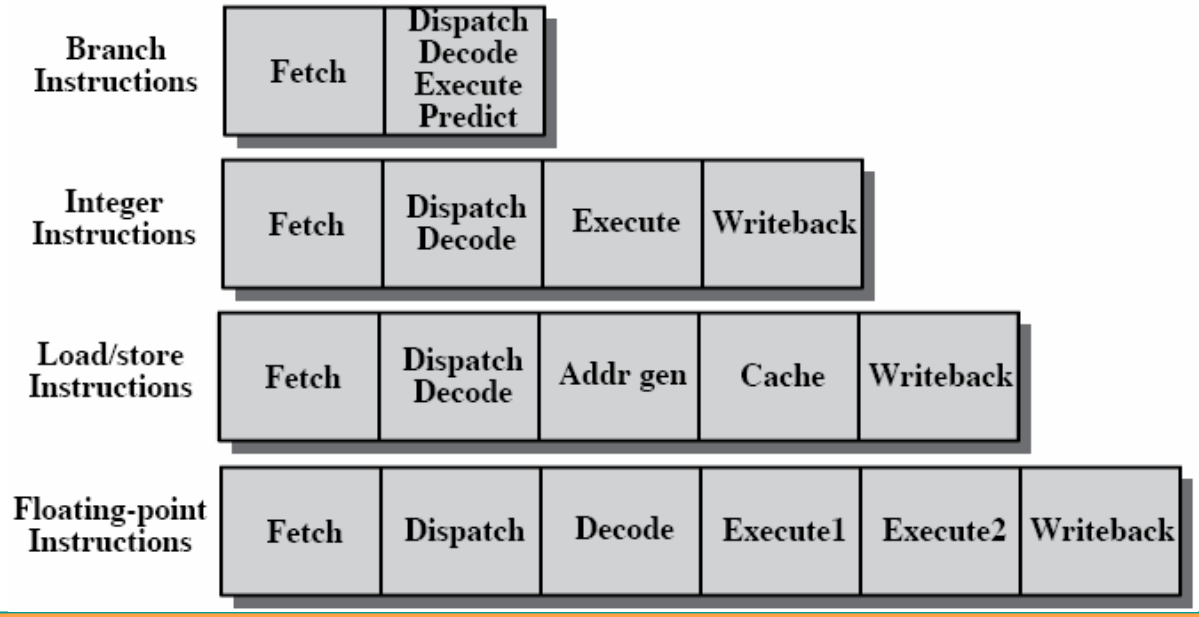
(Sta06 Fig 14.11)

Dispatch unit

- = Käskyjen suoritukseen valinta
- 4:n alkion apupuskuri + 4:n alkion valintaikkuna
  - dispatch buffer = window of execution
- Käsky ikkunasta suoritettavaksi, kun hihnalla tilaa
  - Integer-yksikölle vain jonon alusta
  - Muille se, joka lähinnä jonon keulaa
  - Max 3 käskyä yhtäaikaa (out-of-order issue)
- Kun käsky suoritukseen, muita jonoissa eteenpäin
- Jos riippuvuus, ei päästä hihnalle (stall, bubble)
- Laitetason logiikka hyppyosoitteiden laskemiseksi
  - Nopeasti jo ennen varsinaista käskyn suoritusvaihetta

Suoritus

- Muutokset rekistereihin / muistiin vasta suorituksen lopuksi “Write Back” vaiheessa
- ALU-operaatiot tallettavat tietoa CR-rekisteriin
  - 8 kenttää a’ 4 b, tallessa useita edellisiä vertailutuloksia
- Liukuluvut tarvitsevat useampia syklejä



(Sta06 Fig 14.12)

Hyppyjen käsittely

- **Zero cycle branches**
  - Hyppy ei vaikuta muiden yksikköjen toimintaan
    - Yleensä ei tarvetta tyhjentää, tai hylätä tuloksia
  - Hypyn kohdeosoite selvillä heti, kun hyppykäsky tulee käskyikkunaan (ennen suoritusta!)
- **Yhden hypyn spekulointi: Hyppääkö vai ei?**
  - Jos ehdoton ž taken
  - Jos ehdollinenja CR-rekisteri asetettu aiemmin

    tutki ž taken / not taken

    muuten jos taaksepäin ž taken

    jos eteenpäin ž not taken
- **Jos spekulointi meni pieleen**
  - abortoi spekuloidut käskyt ennen write-back -vaihetta

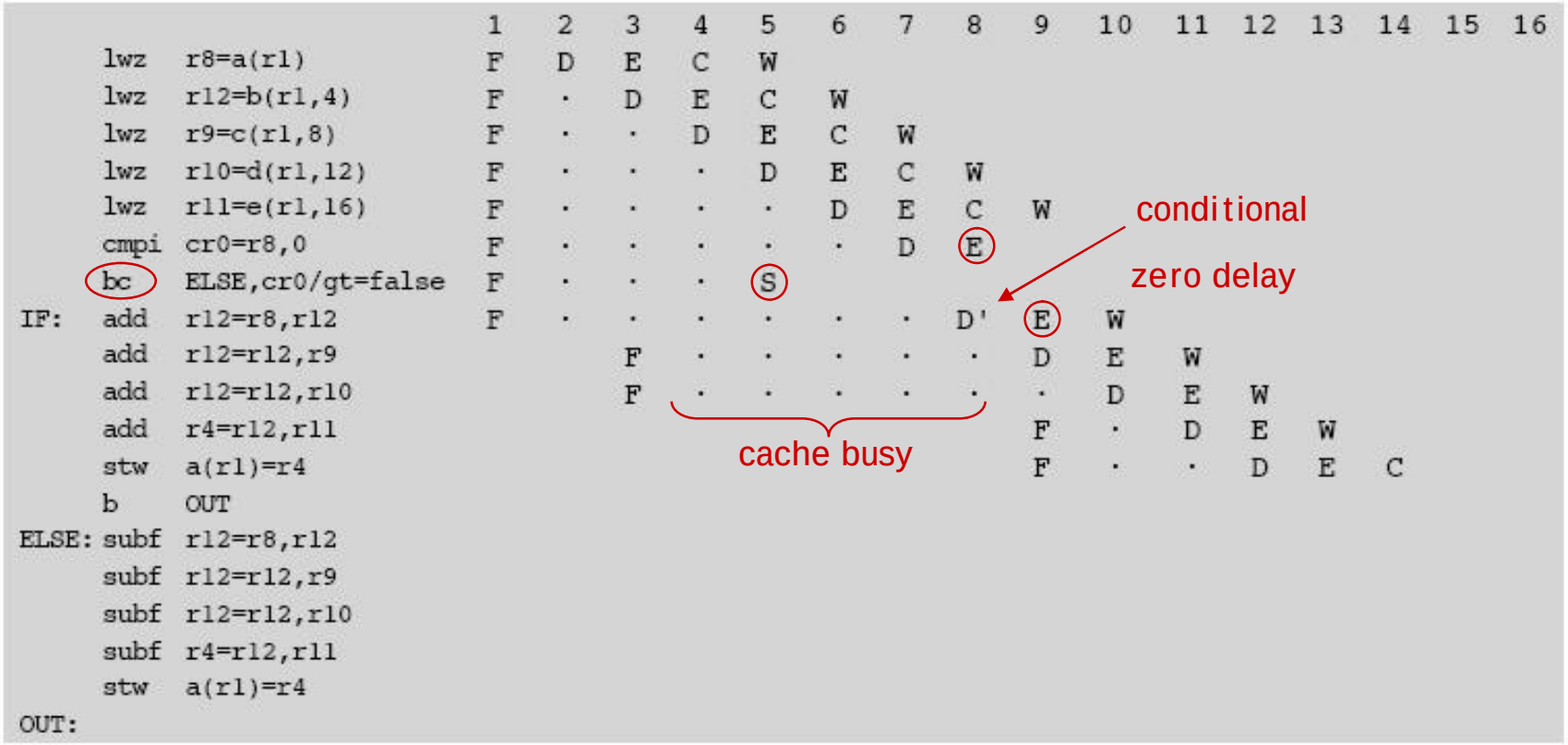
Hyppyjen käsittely

```
        if (a > 0)
                a = a + b + c + d + e;
        else
                a = a - b - c - d - e;
```

```
                                        #r1 points to a,
                                        #r1+4 points to b,
                                        #r1+8 points to c,
                                        #r1+12 points to d,
                                        #r1+16 points to e.
        lwz     r8=a(r1)                #load a
        lwz     r12=b(r1,4)             #load b
        lwz     r9=c(r1,8)              #load c
        lwz     r10=d(r1,12)            #load d
        lwz     r11=e(r1,16)            #load e
        cmpi    cr0=r8,0                #compare immediate
        bc      ELSE,cr0/gt=false       #branch if bit false
IF:
        add     r12=r8,r12              #add
        add     r12=r12,r9              #add
        add     r12=r12,r10             #add
        add     r4=r12,r11              #add
        stw     a(r1)=r4                #store
        b       OUT                     #unconditional branch
ELSE:
        subf    r12=r12,r8              #subtract
        subf    r12=r9,r12              #subtract
        subf    r12=r10,r12             #subtract
        subf    r4=r12,r11              #subtract
        stw     a(r1)=r4                #store
OUT:
```

(Sta06 Fig 14.13)

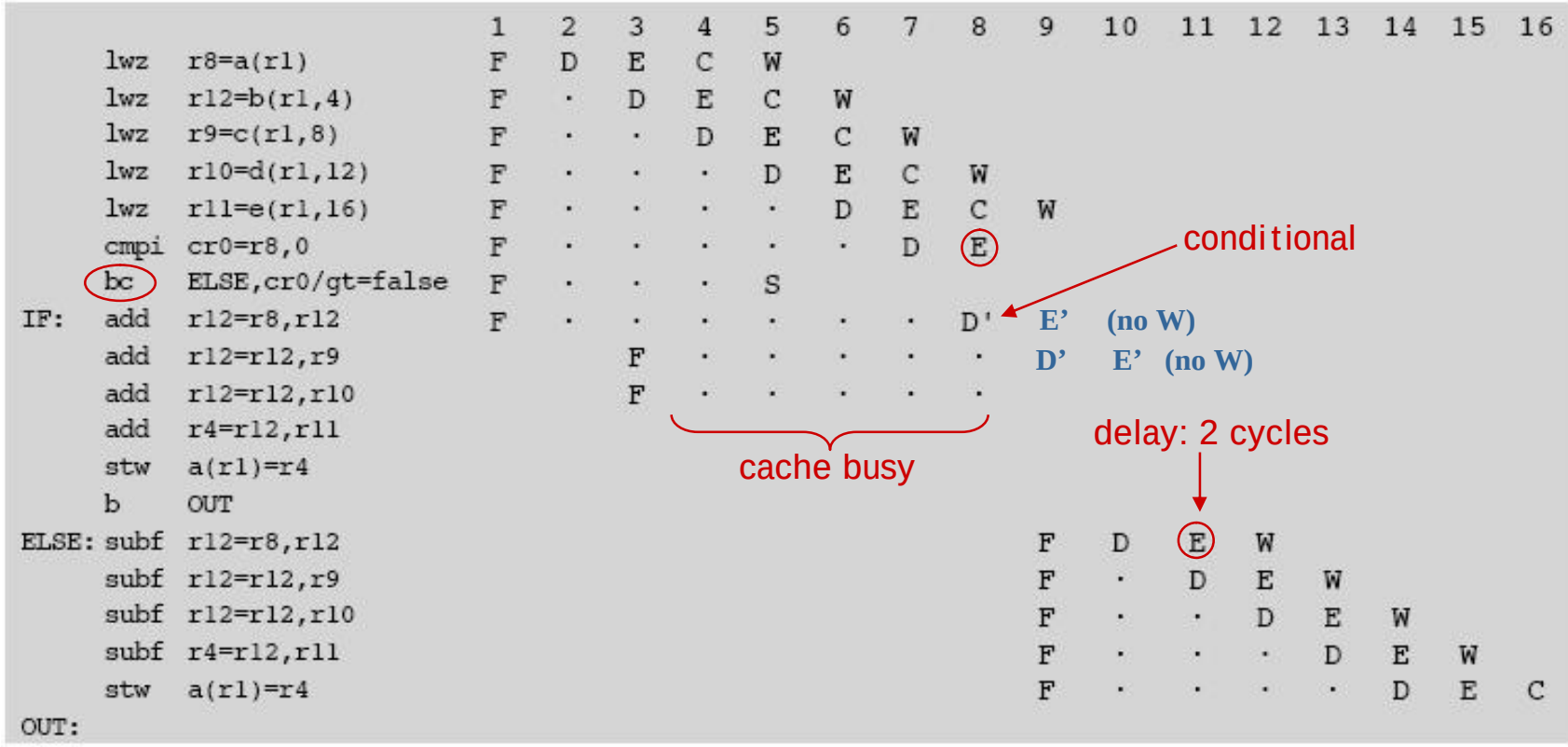
Hyppyjen käsittely



(a) Correct prediction: Branch was not taken

(Sta06 Fig 14.14a)

Hyppyjen käsittely



			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	lwz	r8=a(r1)	F	D	E	C	W											
	lwz	r12=b(r1,4)	F	·	D	E	C	W										
	lwz	r9=c(r1,8)	F	·	·	D	E	C	W									
	lwz	r10=d(r1,12)	F	·	·	·	D	E	C	W								
	lwz	r11=e(r1,16)	F	·	·	·	·	D	E	C	W							
	cmpi	cr0=r8,0	F	·	·	·	·	·	D	E								
	bc	ELSE,cr0/gt=false	F	·	·	·	S											
IF:	add	r12=r8,r12	F	·	·	·	·	·	·	D'	E'	(no W)						
	add	r12=r12,r9			F	·	·	·	·	·	D'	E'	(no W)					
	add	r12=r12,r10			F	·	·	·	·	·								
	add	r4=r12,r11																
	stw	a(r1)=r4																
	b	OUT																
ELSE:	subf	r12=r8,r12									F	D	E	W				
	subf	r12=r12,r9									F	·	D	E	W			
	subf	r12=r12,r10									F	·	·	D	E	W		
	subf	r4=r12,r11									F	·	·	·	D	E	W	
	stw	a(r1)=r4									F	·	·	·	·	D	E	C
OUT:																		

(b) Incorrect prediction: Branch was taken

(Sta06 Fig 14.14b)

PowerPC 620

HePa96 Fig. 4.49

64b:n arkkitehtuuri

- 6 suoritusyksikköä
  - Instruction-yksikkö (dispatcher)
  - 3 integer-yksikköä
  - Load/Store yksikkö
  - FP-yksikkö
- Max 4 käskyä suoritukseen yhtäaikaa
- Reservation stations
  - Kullakin yksiköllä kaksi tai useampia
  - Jos käsky ei voi edetä (riippuvuudet) se odottaa tässä, eikä estä jäljempänä olevien etenemistä
- Uudelleennimeäminen: 8 integer ja 12 FP lisärekisteriä
  - Vähentää riippuvuuksia
  - Väliaikaistulosten tallettamiseksi
- In-order-complete
  - max 4 käskyä yhtäaikaa

PowerPC 620

- **Hyppyjen spekulointi**
  - 256:n alkion branch target buffer (BTB)
    - Joukkoassosiatiivinen, joukon koko 2
  - 2048:n alkion branch history table
    - Käyttää, jos kohde ei löydy BTB:stä
- **Spekuloitavana max 4 ratkaisematonta hyppyä**
- **Tulokset uudelleennimeämisrekistereissä**
  - Commit: kopioi spekuloidut tulokset todellisiin rekistereihin
  - Abort: vapauta rekisterit muuhun käyttöön

Kertauskysymyksiä

- Miten superskalaaritoteutus eroaa tavallisesta liukuhihnoitetusta toteutuksesta?
- Mitä uusia rakenteesta johtuvia ongelmia tulee ratkottavaksi?
- Miten niitä ongelmia ratkotaan?
- Mitä tarkoittaa rekistereiden uudelleennimeäminen ja mitä hyötyä siitä on?