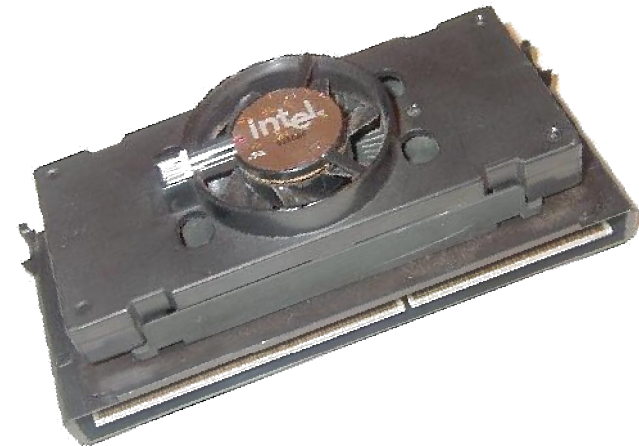




Suorittimen rakenne ja toiminta



Ch 12.1-4 [Sta06]

- n Rekisterit
- n Käsikysykli
- n Liukuhihna
- n Riippuvuusongelmat
- n Hyppyjen käsittely

CPU:n yleisrakenne

(Sta06 Fig 12.2)

n ALU

- u Laskenta, vertailut

n Rekisterit

- u Nopeaa työtilaa

n Prosessoriväylä

- u Bittien siirto paikasta toiseen

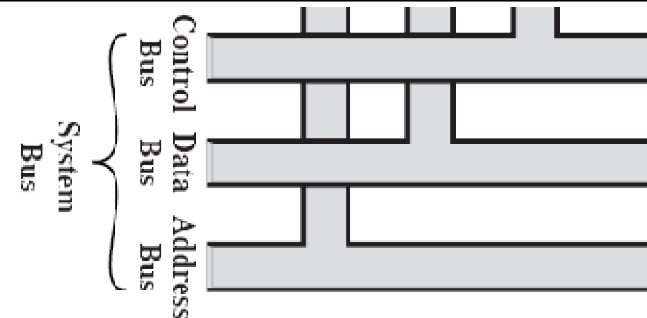
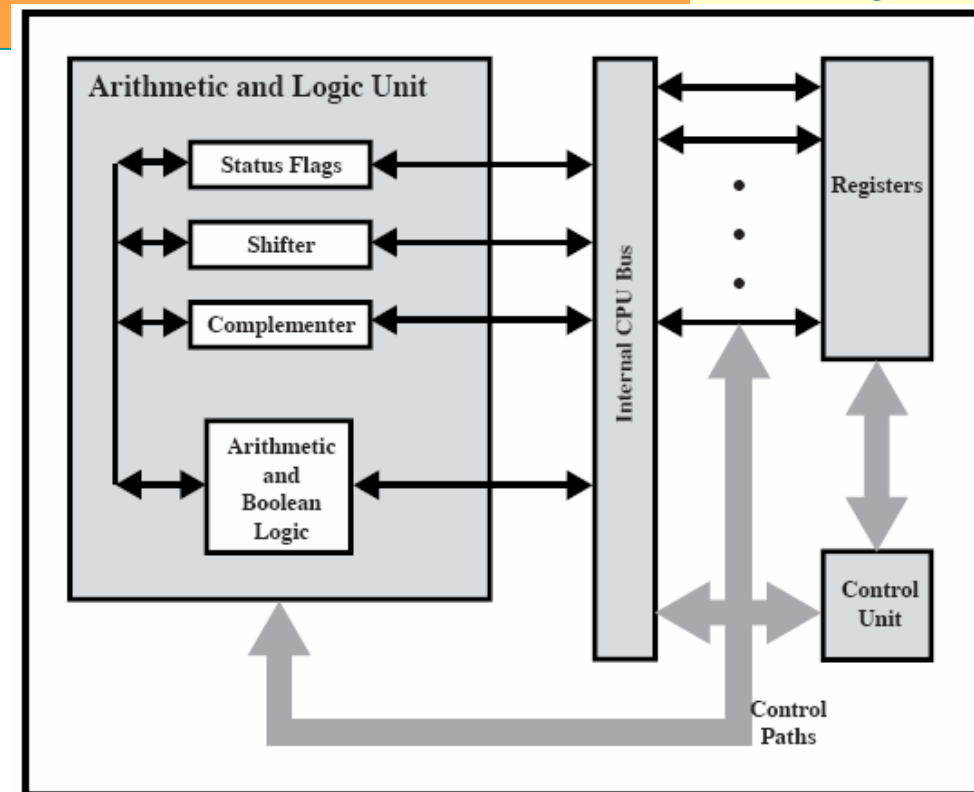
n Ohjausyksikkö (Ch 16-17)

- u Kuka? Mitä? Milloin?
- u Kellopulssi
- u Generoi ohjaussignaalit

§ mitä seuraavalla kellopulssilla tapahtuu?

n MMU?

n Välimuisti?





Rekisterit

- n Muistihierarkian huipulla
- n Käyttäjälle näkyvät rekisterit
 - u Ohjelmoija/kääntäjä päättää niiden käytöstä
 - u Paljonko? Nimeäminen? **ADD R1,R2,R3**
- n Ohjaus ja statusrekisterit
 - u Osa epäsuorasti ohjelman viitattavissa
 - § PC, PSW, flags, ... **BNEQ Loop**
 - u Osa vain CPU:n sisäiseen käyttöön
 - § MAR, MBR, ...
- n Sisäisiä apurekistereitä käsikyn välivaiheita varten
 - u Esim. - Käskyrekisteri (IR) käsikyn tulkintaa varten
 - operandi ensin käsikystä apurekisteriin, vasta sitten ALU:uun



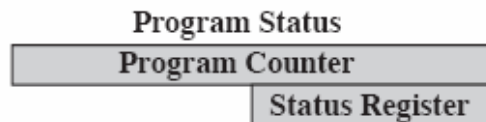
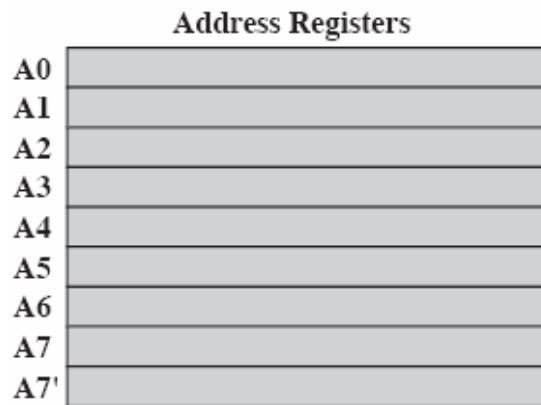
Käyttäjälle näkyvät rekisterit

- n Eri prosessoriperhe \bar{O}
 - eri määrä erilaisia rekistereitä,
 - erilaiset nimeämistavat
 - erilaiset käyttötarkoitukset
- n Yleisrekisterit
- n Datarekisterit
- n Osoiterekisterit
- n Segmenttirekisterit
- n Pino-osoitin, ympäristöosoitin (frame pointer)
- n Status- ja tilarekisterit



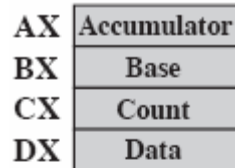
Esimerkkejä

(Sta06 Fig 12.3)

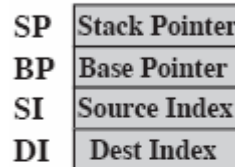


(a) MC68000

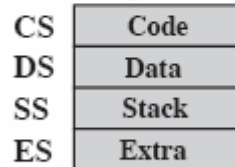
General Registers



Pointer & Index



Segment

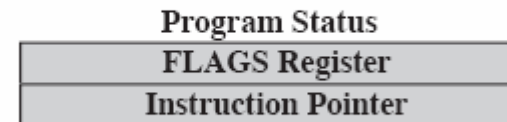
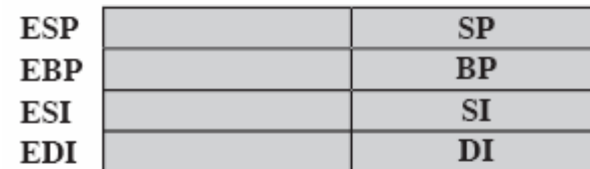
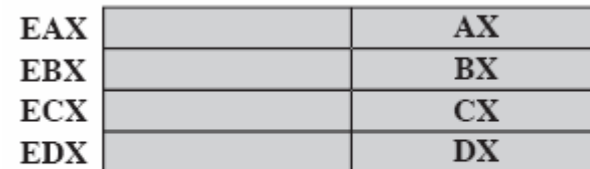


Program Status



(b) 8086

General Registers



(c) 80386 - Pentium 4



PSW - Program Status Word

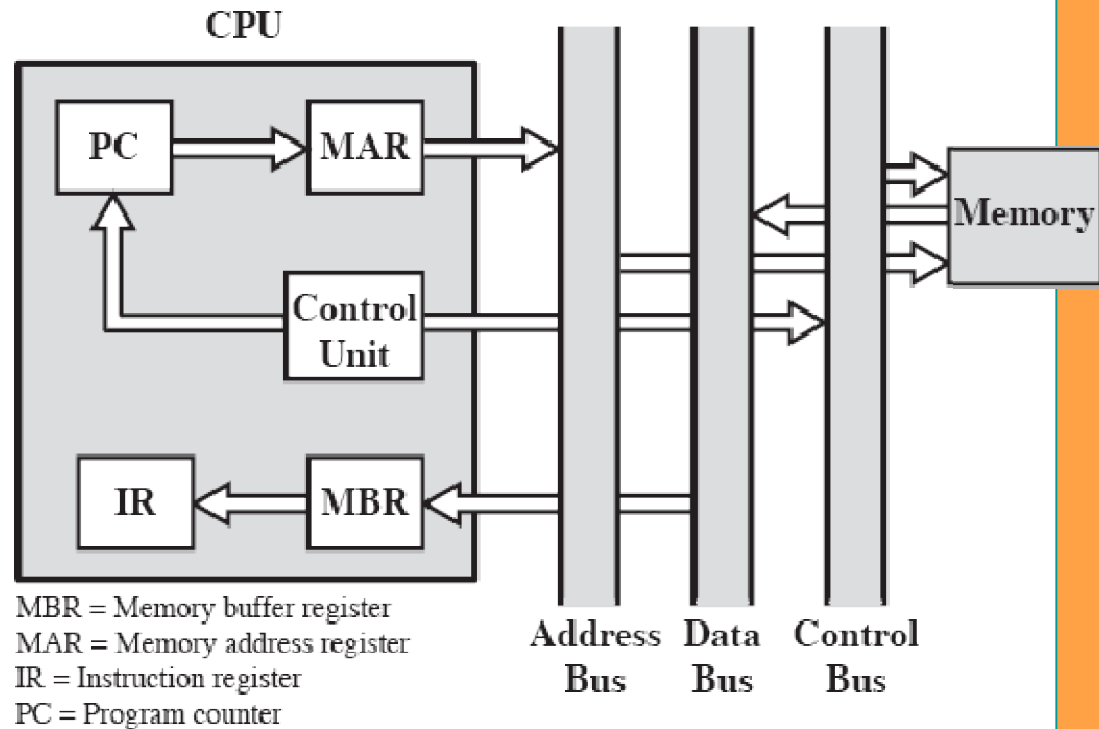
- n Nimi vaihtelee eri arkkitehtuureissa
- n CPU:n tila
 - u Etuoikeutettu vs. käyttäjätila
- n Vertailun tulos
 - u Esim. Greater, Equal, Less, Zero, ...
- n Sattuiko käskyn suorituksessa poikkeuksia?
 - u Nollallajako, ylivuoto
 - u Page fault, "memory violation"
- n Tarvitseeko ulkoinen laite ohjausta?
 - u Bitti keskeytyksen ilmaisemiseksi
- n Keskeytyksen esto / salliminen
 - u Kullekin luokalle omat bittinsä



Käskyn nouto (10)

n
n
n
n
n
n
n

- MAR • PC
- MAR • MMU(MAR)
- Control Bus • Reserve
- Control Bus • Read
- PC • ALU(PC+1)
- MBR • MEM[MAR]
- Control Bus • Release
- IR • MBR



Välimuisti! Ennaltanouto!

(Sta06 Fig 12.6)

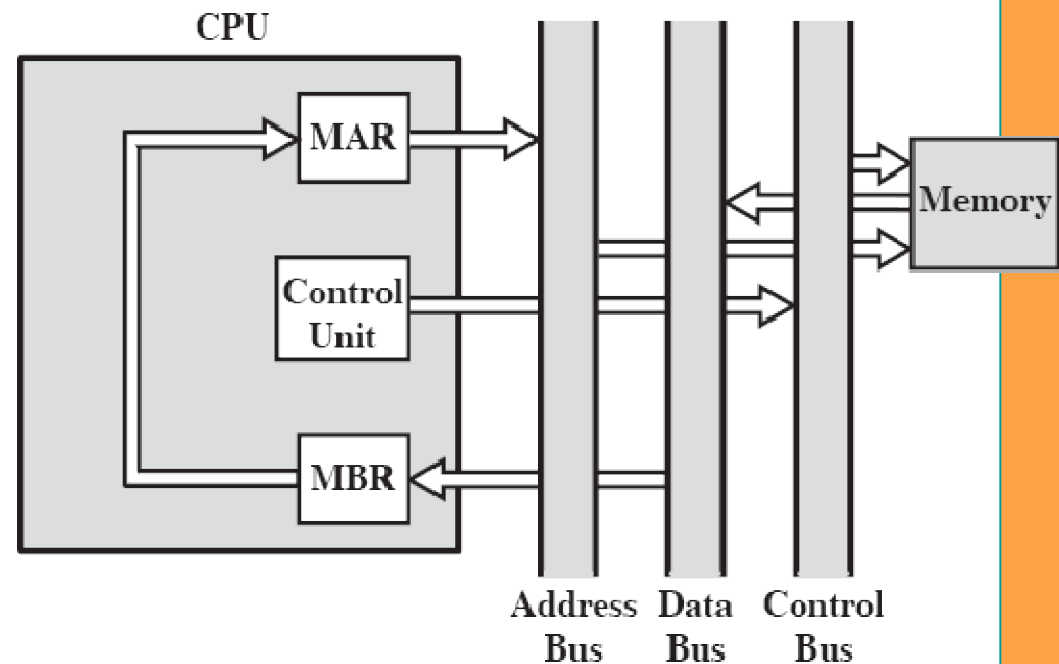


Operandin nouto, Epäsuora osoitus (13)

- MAR • Address
- MAR • MMU(MAR)
- Control Bus • Reserve
- Control Bus • Read
- MBR • MEM[MAR]

- MAR • MBR
- MAR • MMU(MAR)
- Control Bus • Read
- MBR • MEM[MAR]
- Control Bus • Release
- ALU? Regs? • MBR

Välimuisti!

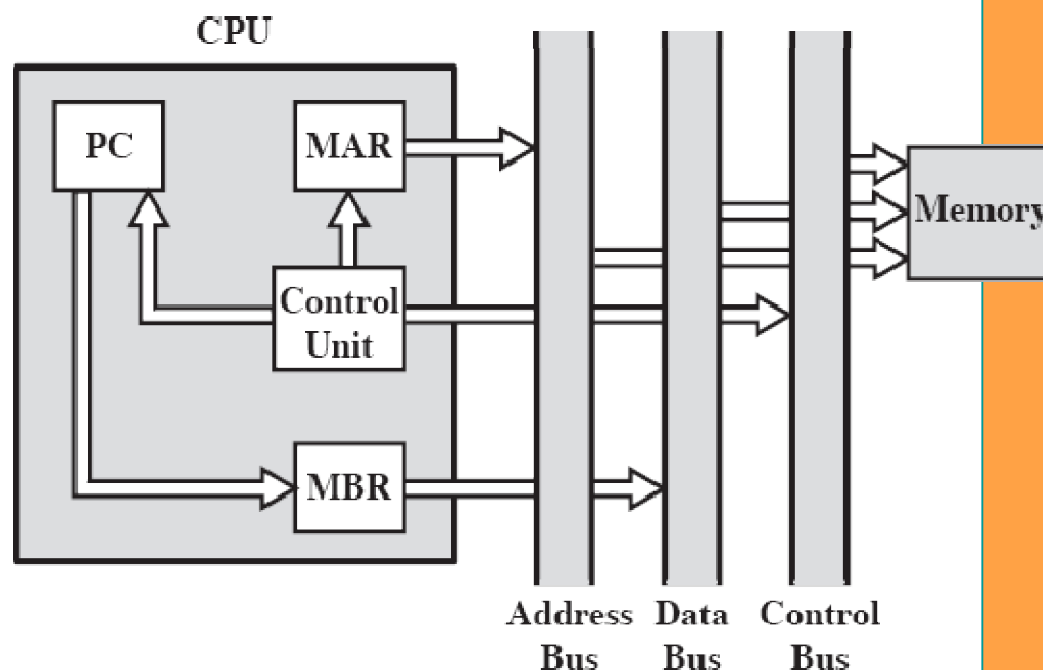


(Sta06 Fig 12.7)



Keskeytyskäsitteilyn alkutoimet (esim.) (16)

- MAR • SP
- MAR • MMU(MAR)
- Control Bus • Reserve
- MBR • PC
- Control Bus • Write
- MAR • SP • ALU(SP+1)
- MAR • MMU(MAR)
- MBR • PSW
- Control Bus • Write
- SP • ALU(SP+1)
- PSW • privileged & disable
- MAR • Interrupt nbr
- Control Bus • Read
- PC • MBR • MEM[MAR]
- Control Bus • Release



Ei osoitteenmuunnosta!

SP = Stack Pointer (Sta06 Fig 12.8)



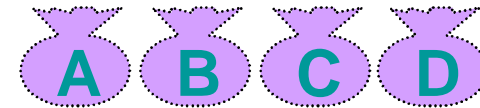
Tietokoneen rakenne

Liukuhihna



Esim. Pesula (by David A. Patterson)

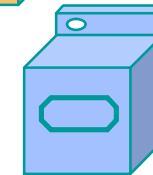
n Ann, Brian, Cathy, Dave:
kullakin satsi pyykättävää



n Pesu vie 30 min



n Kuivaus vie 40 min



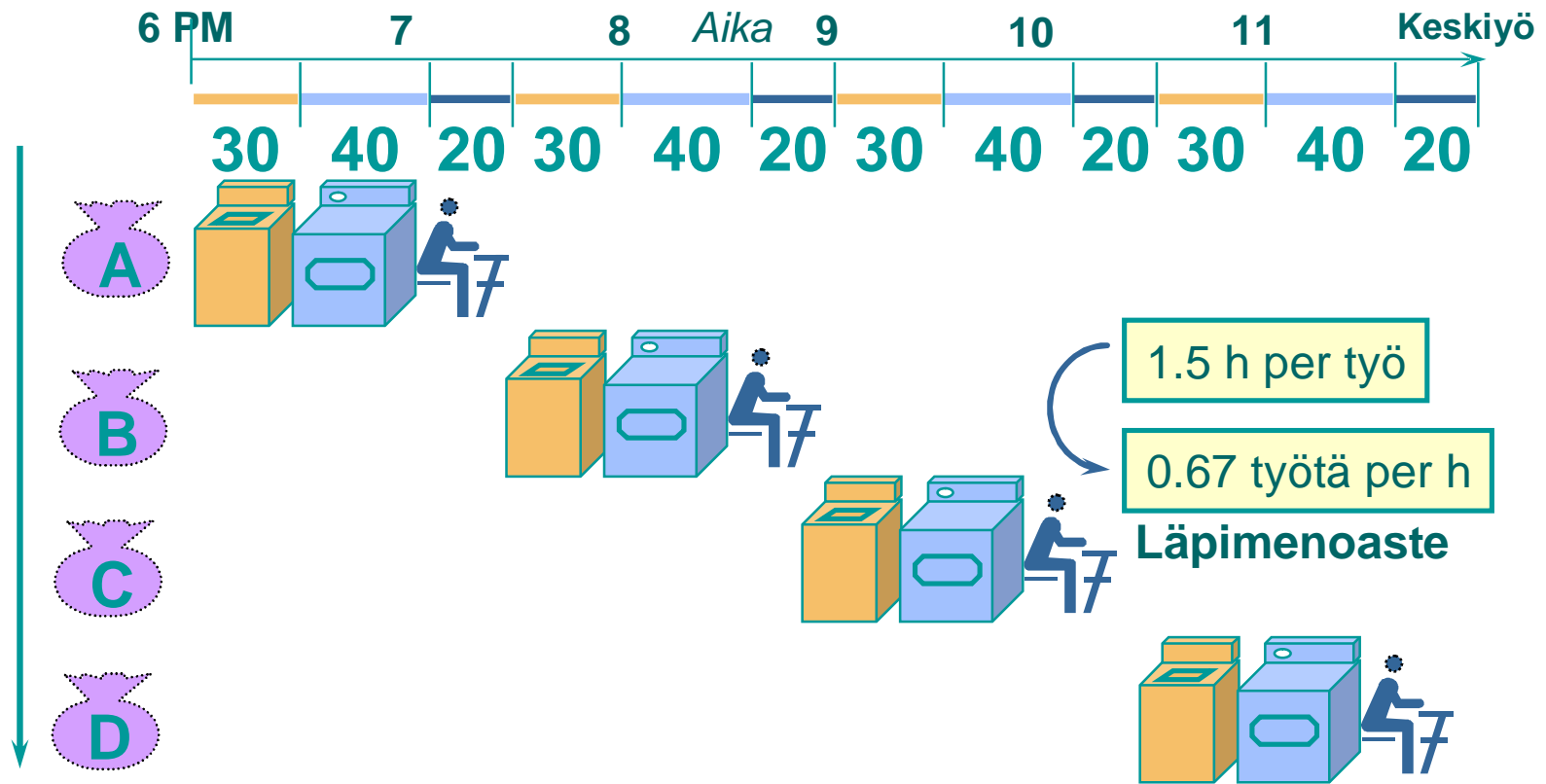
n Viimeistely vie 20 min





Peräkkäin: Pyykkitupa varattu vain yhdelle (8)

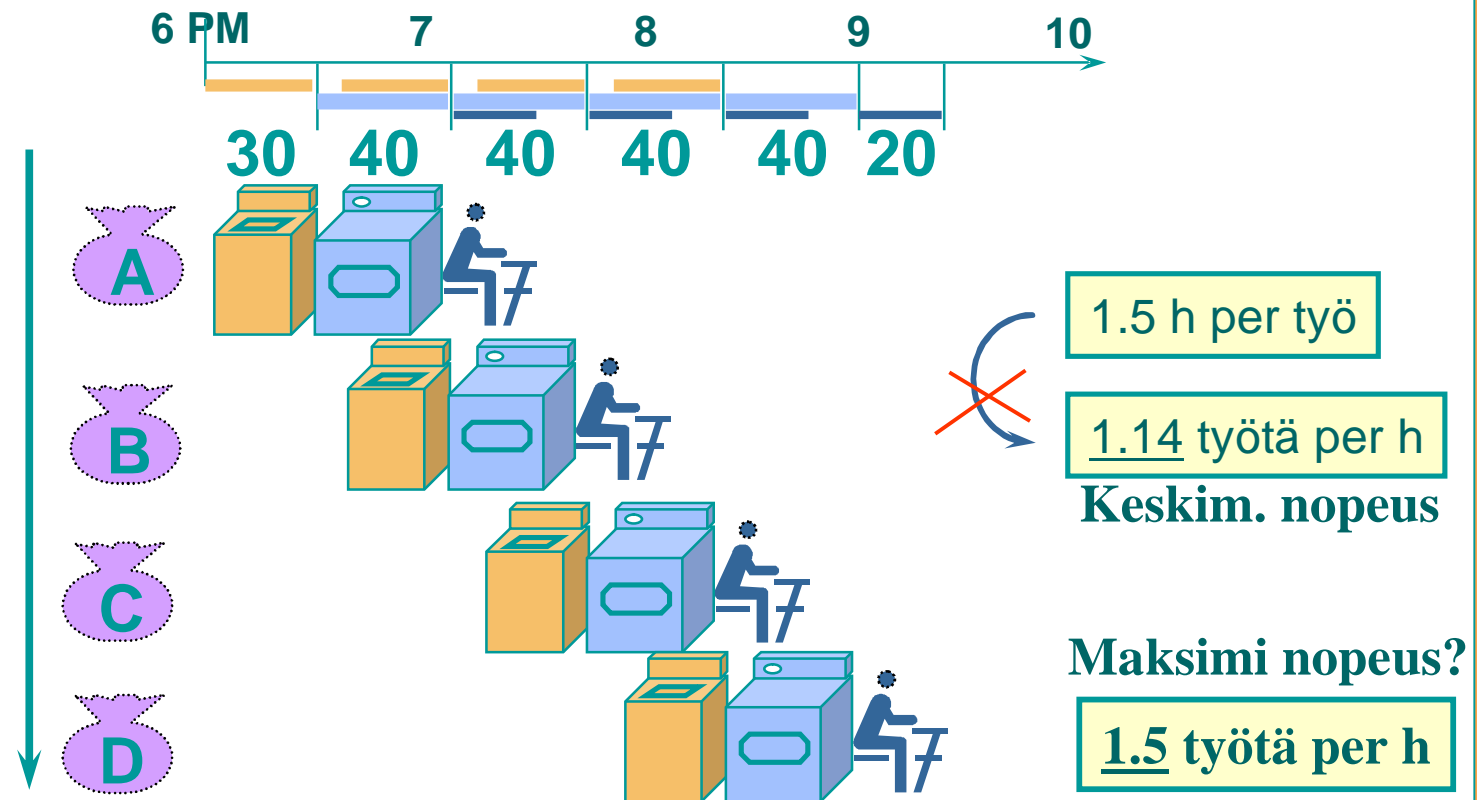
- n Vie 6 tuntia, syntyy neljä satsia puhdasta
- n Entäpä, jos toimisivat liukuhihnaperiaatteella?





Liukuhihna: Varaus työvaiheittain (13)

n Vie 3,5 tuntia, syntyy 4 satsia puhdasta



Töitä valmistuu parhaimmillaan 40 min välein! (0.67 h / valmistunut työ)



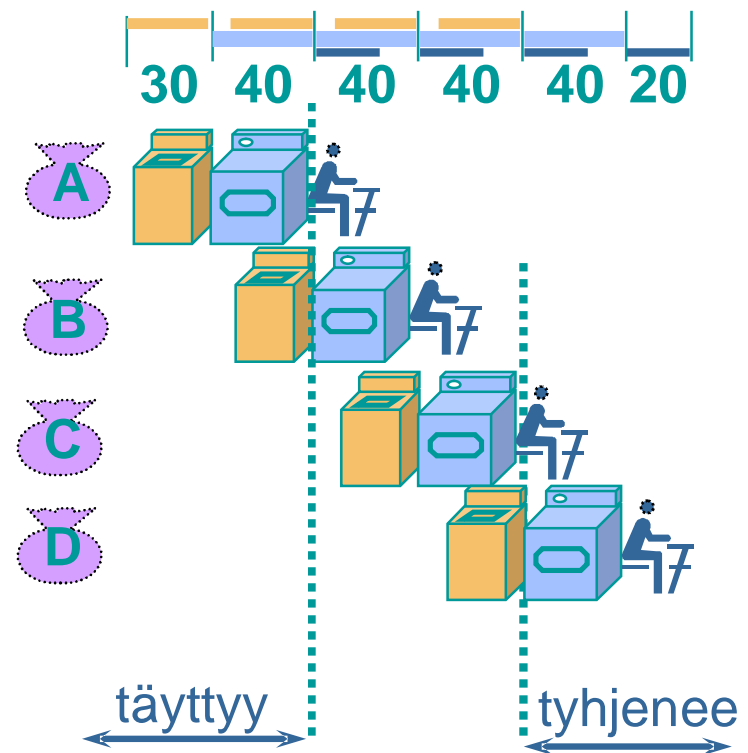
Huomioita

- n Liukuhihna ei nopeuta yhden työn aktiivisena oloaikaa järjestelmässä
 - u Mutta keskimääräinen läpimenoaika paranee
- n Liukuhihna voi aiheuttaa yhdelle työlle lisäviipeitä verrattuna siihen, jos se työ olisi yksin järjestelmässä
 - u Jos hihnan seuraava vaihe varattuna, ei voi edetä
- n Useita töitä etenee yhtä aikaa, mutta ne ovat eri vaiheissa
- n Hitain vaihe määrää koko liukuhihnan nopeuden
 - u Liukuhihna nytkähtää, kun kaikki osat valmiita
 - u Tehokkuus kärsii, jos kestot eripituisia
- n Yksi työ nopeutuu maksimissaan saman verran kuin liukuhihnassa vaiheita



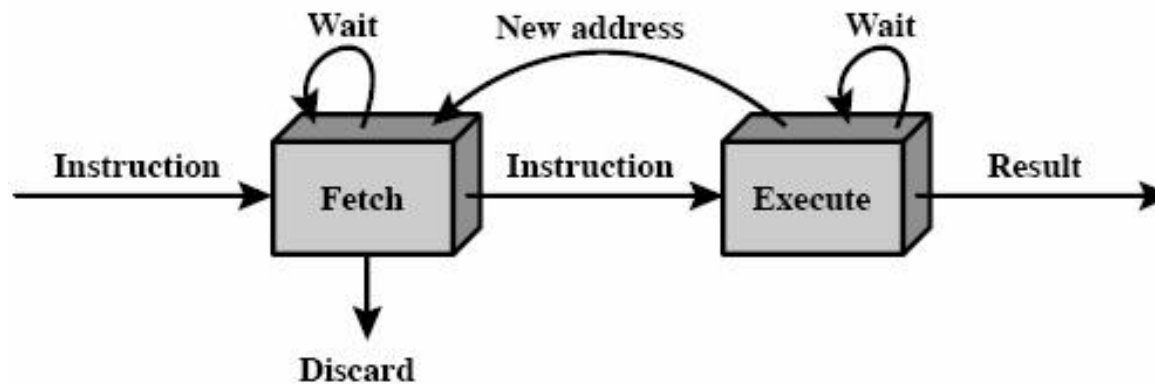
Huomioita

- n Toteutus monimutkaisempaa, lisäresurssien tarve
 - u Tuplapistorasia? ž pesukone ja kuivaaja voivat pörrätä yhtäaikaan
 - u Kaksi tai kolme ihmistä yhtäaikaan pyykkituovassa
- n Hihna ei toimi alussa eikä lopussa täydellä teholla
- n Hihna "yskii"
 - u Töitä ei tule tasaisella tahdilla





CPU ja 2-vaiheinen liukuhihna?



(Sta06 Fig 12.9)

- n Seuravan käskyn ennaltanouto samaan aikaan kuin edellistä suoritetaan
 - u Paikallisuus: oletta, että suoritus "esiintymisjärjestyksessä"
- n Ongelmat
 - u Suoritusvaihe voi kestää kauemmin \checkmark noutovaihe jouten
 - u Suoritusvaihe muuttaa PC:n arvoa \checkmark haettu väärä käsky
 - § Seuraavan käskyn osoite "ennustettiin" väärin!
- n Rinnakkaisuus vielä vähäistä \checkmark Lisää vaiheita?



6-vaiheinen liukuhihna

	Time →													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

FE - Fetch instruction

DI - Decode instruction

CO - Calculate operand addresses

FO - Fetch operands

EI - Execute instruction

WO - Write operand

(Sta06 Fig 12.10)



Paljonko liukuhihna nopeuttaa?

- n 6 vaiheinen liukuhihna ž 14 aikayksikköä
- n Ei liukuhihnaa ž $9 \cdot 6 = 54$ aikayksikköä
- n Nopeutus = $54/14 = 3.86 < 6$!
- n Todellisuudessa nopeutus vieläkin pienempi

- n Jokainen käsky ei tarvitse kaikkia vaiheita
 - u Ei-liukuhihnoitettu voi edetä heti seuraavaan vaiheeseen
 - u Liukuhihnoitettu joutuu odottamaan seuraavan vaiheen vapautumista, ko. CPU:n yksikkö jouten
 - § Suorituksessa on "kupla" (bubble)



Yhden vaiheen kesto liukuhinalla?

$$\tau = \max_{i=1..k} [\tau_i] + d = \tau_m + d \gg d$$

Yhden vaiheen kesto

Vaiheen i kesto

Viive vaiheesta toiseen siirtymiselle
~ yksi kellopulssi

Hitaimman vaiheen (max) kesto

- n Hitain vaihe määrää etenemisvauhdin
 - u Kauimmin kestävä vaihe on pullonkaula
- n Kukin vaihe asetetaan kestämään yhtä kauan
 - u Montako kellopulssia?



Paljonko nopeuttaa?

n käskyä, k vaihetta, τ =vaiheen kesto

Ei liukuhihnaa: $T_1 = nk\tau$

Pessimistinen: olettaa, että kaikki vaiheet kestävät yhtä kauan

Liukuhihna: $T_k = [k + (n - 1)]\tau$

Ks. Sta06 Fig 12.10 ja tarkista itse!

k vaihetta ennenkuin ensimmäinen työ valmistuu

seuraavat (n-1) valmistuvat yhden vaiheen välein

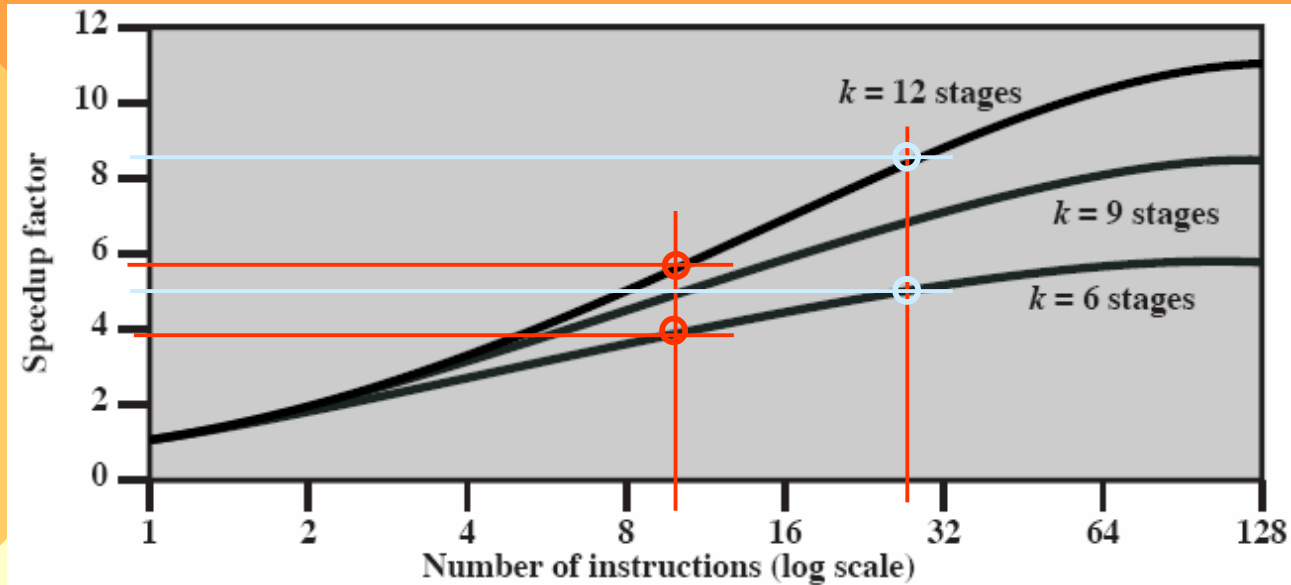
Nopeutus:

$$S_k = \frac{T_1}{T_k} = \frac{nk\tau}{[k + (n - 1)]\tau} = \frac{nk}{[k + (n - 1)]}$$



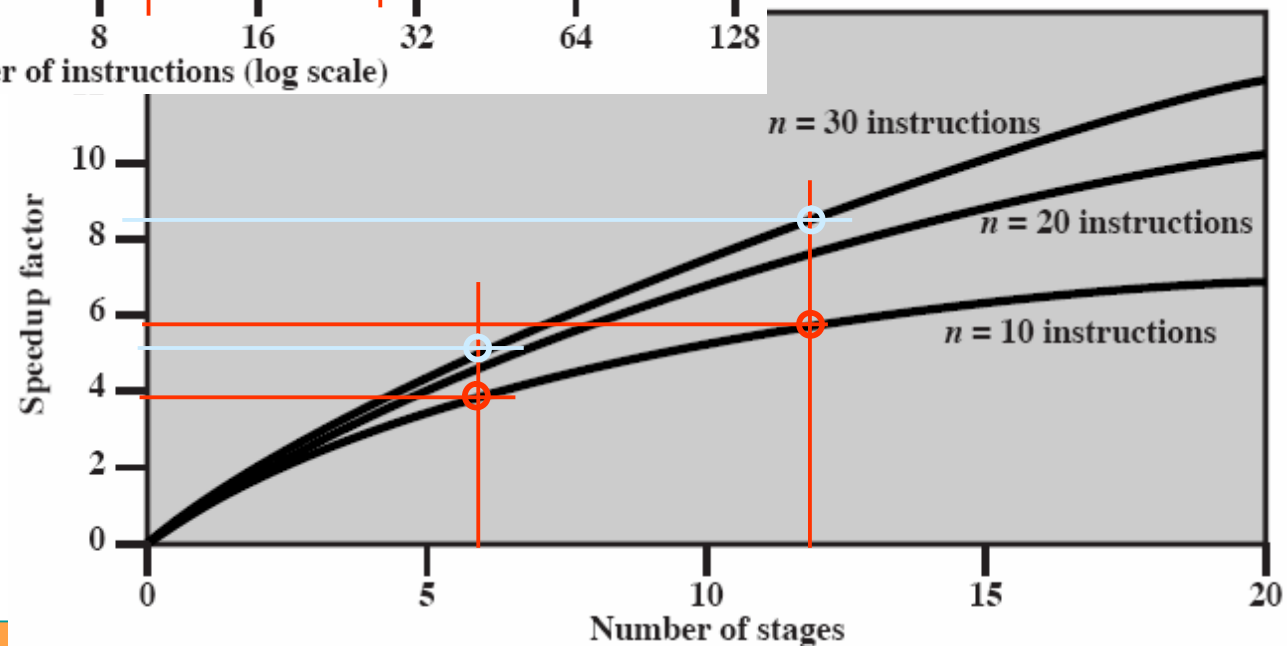
Paljonko nopeuttaa?

Oletuksella: ei hyppykäskeyjä



(Sta06 Fig 12.14)

more gains from multiple stages when more instructions without jumps





Muita huomioita

n Lisävaiheita

- u CPU:n laitettava välitulos jonnekin
- u Vaikka yksittäinen käsky etenisi yksin hinnalla, sillä kuluu enemmän aikaa kuin ei-liukuhihnoitetussa

n Silti

- u Ison käskyjoukon suorittaminen sujuu nopeammin
- u Parempi läpimenoaste (käskyä/sec)

n Liukuhihnoituksen rinnakkaisuus nopeuttaa kokonaisuutta, mutta hidastaa yhden käskyn suoritusta



Ongelmakohtia

n Rakenteellinen riippuvuus (structural dependency)

- u Usea vaihe voi tarvita samaa HW-yksikköä
- u Muisti: FI, FO, WO
- u ALU: CO, EI

```
STORE R1,VarX  
ADD R2,R3,VarY  
MUL R3,R4,R5
```

n Kontrolliriippuvuus (control dependency)

- u Ehdollisen hyppykäselyn tulos selvillä vasta EI -vaiheen jälkeen
- ž Ennaltanoudettu väärää käskyjä

```
ADD R1,R7, R9  
Jump There  
ADD R2,R3,R4  
MUL R1,R4,R5
```

n Datariippuvuus (data dependency)

- u Tarvitaan edeltävän käselyn tulosta, eikä se ole vielä valmistunut

```
MUL R1,R2,R3  
LOAD R6, Arr(R1)
```






Ratkaisuja?

- n Huomattava laitteistolla, odotettava, että riippuvuus poistettu
 - u Hihnalle ylimääräisiä odotusjaksoja eli kuplia, "bubbles"
 - u Kupla viivästää kaikkia sen takana tulevia vaiheita
 - u Yleisesti käytetty tapa
- n Rakenteellinen riippuvuus
 - u Lisää laitteistoa, esim. CO- ja EI -vaiheelle omat ALUt
 - u Paljon rekistereitä, vähemmän muistioperandeja
- n Kontrolliriippuvuus
 - u Tyhjennä hihna, ja hae uudet käskyt
 - u Hyppyjen ennustuslogiikka: haenko ennalta tästä vai tuolta?
- n Datariippuvuus
 - u Vaihda käskyjen suoritusjärjestystä
 - u Oikopolut laitteistossa (by-pass): tulokseen pääsee käsiksi ennen WO-vaihetta



Datariippuvuudet ja niiden poisto

MUL R1, R2, R3
ADD R4, R5, R6
SUB R7, R1, R8
ADD R1, R1, R3

1	2	3	4	5	6	7	8	9	10	11
FI	DI	CO	FO	EI	WO					
	FI	DI	CO	FO	EI	WO				
		FI	DI	CO		FO	EI	WO		
			FI	DI	CO		FO	EI	WO	

MUL R1, R2, R3
ADD R4, R5, R6
SUB R7, R7, R8
ADD R1, R1, R3

1	2	3	4	5	6	7	8	9	10	11
FI	DI	CO	FO	EI	WO					
	FI	DI	CO	FO	EI	WO				
		FI	DI	CO	FO	EI	WO			
			FI	DI	CO	FO	EI	WO		

too far,
no effect





Käskeyjen suoritusjärjestyksen muutos

```
MUL R1, R2, R3
ADD R4, R5, R6
SUB R7, R1, R8
ADD R9, R0, R8
```



```
MUL R1, R2, R3
ADD R4, R5, R6
ADD R9, R0, R8
SUB R7, R1, R8
```

1	2	3	4	5	6	7	8	9	10	11
FI	DI	CO	FO	EI	WO					
	FI	DI	CO	FO	EI	WO				
		FI	DI	CO		FO	EI	WO		
			FI	DI	CO		FO	EI	WO	

1	2	3	4	5	6	7	8	9	10	11
FI	DI	CO	FO	EI	WO					
	FI	DI	CO	FO	EI	WO				
		FI	DI	CO	FO	EI	WO			
			FI	DI	CO	FO	EI	WO		

switched instructions



Oikopolut

MUL R1, R2, R3
ADD R4, R5, R1
SUB R7, R4, R1

1	2	3	4	5	6	7	8	9	10	11	
FI	DI	CO	FO	EI	WO						
	FI	DI	CO			FO	EI	WO			
		FI	DI	CO					FO	EI	WO

MUL R1, R2, R3
ADD R4, R5, R1
SUB R7, R4, R1

1	2	3	4	5	6	7	8	9	10	11
FI	DI	CO	FO	EI	WO					
	FI	DI	CO		FO	EI	WO			
		FI	DI	CO			FO	EI	WO	

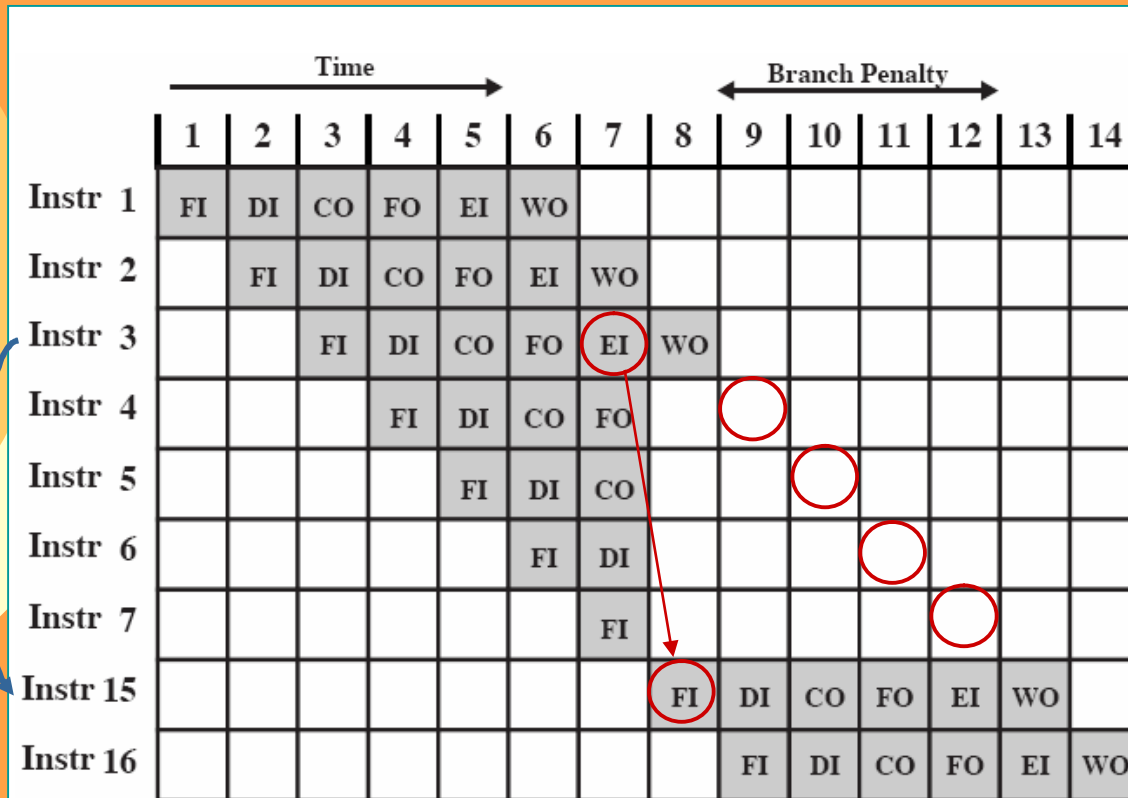
With by-pass



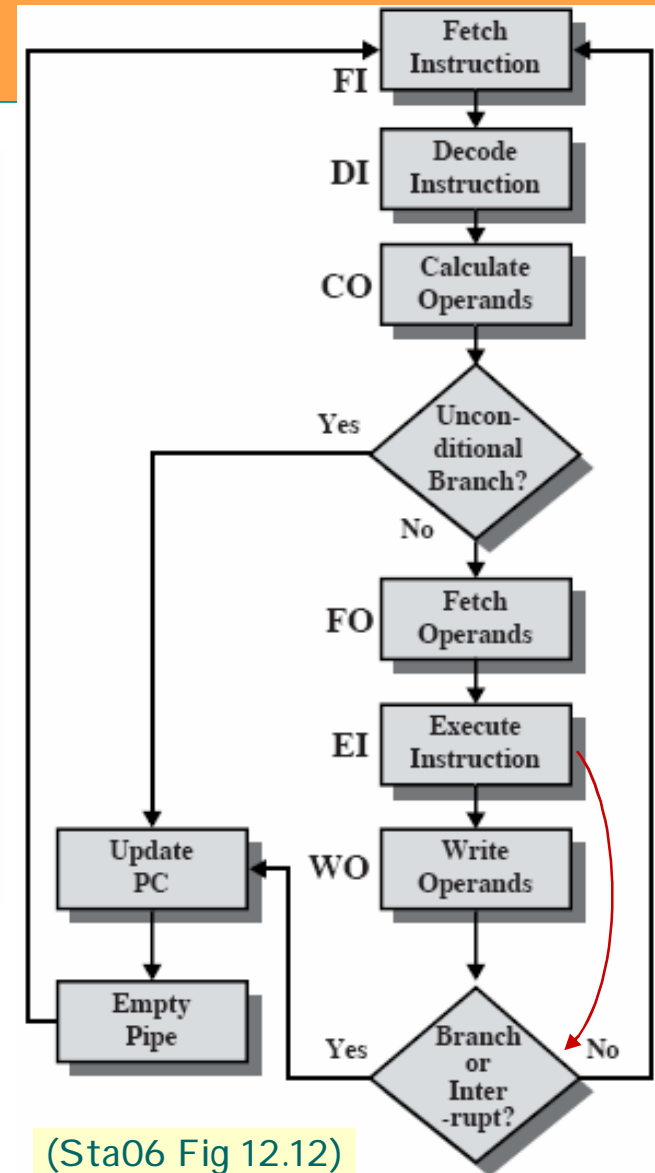
Hypyt ja liukuhihna

- n Viivästetty hyppy
- n Monta suorituspolkua
- n Kohteen ennaltanouto
- n Silmukkapuskuri
- n Ennustuslogiikka

Hypyn vaikutus liukuhihnaan



(Sta06 Fig 12.11)



(Sta06 Fig 12.12)



Viivästetty haarautuminen (delayed branch)

- n Kääntäjä laittaa hyppykäskyn perään muita käskyjä, jotka suoritetaan aina
 - u Hihnalle ei päästetä mitään, mikä pitäisi ehkä myöhemmin perua
 - § tehdyn työ peruuttaminen on tosi vaikeata!
 - u Jos ei hyödyllisiä käskyjä, sitten NOP-käskyjä
- n Jos hyppy toteutuu, nuo käskyt jo hyvässä vauhdissa ja annetaan niiden valmistua
 - u Hihnaa ei tarvitse erikseen tyhjentää
- n Jos hyppy ei toteudu, NOP-käskyt vain hukkaavat syklejä
- n Helpompi toteuttaa näin kuin tyhjentää hihna

```
sub r5, r3, r7
add r1, r2, r3
jump There
...
```



```
sub r5, r3, r7
jump There
add r1, r2, r3
...
```

delay slot

add r1, r2, r3



Monta suorituspolkua (multiple instr. streams)

n Spekuloi

- u Hae hinnalle hyppykäskyä seuraavia käskyjä
- u Hae toiselle hinnalle käskyjä hypyn kohdeosoitteesta

n Anna polkujen edetä kunnes selviää mikä oikea

- u Hylkää väärä polku (tai ainakin sen tulokset)

n Ongelmia

- u Hypyn kohde saattaa selvitä vasta osoitelaskennan jälkeen
- u Polku voi haarautua edelleen
 - § Jatketaanko samoin? Yksi spekulointi kerrallaan?
- u Tarvitaan lisälaitteistoa
 - § Useampia liukuhihnoja
 - § Spekulatiivisia tuloksia ei saa tallettaa oikeisiin rekistereihin
- u Spekulointi käyttää resursseja ja voi hidastaa oikeaa työtä
 - § CPU:n sisäiset työrekisterit, väylä, ALU

n Liukuhihna osattava tyhjentää



Kohteen ennaltanouto (prefetch branch target)

- n Nouda etukäteen käsky hypyn kohteesta, mutta älä päästä sitä muuten liukuhihnalle
 - u Tee siis vain FI -vaihe
 - u Jos hyppy toteutuu, säästyy muistinoutoon muuten tarvittava aika
- n Liukuhihnaa osattava tyhjentää **IBM 360/91 (1967)**



Silmukkapuskuri (loop buffer)

- n Pidä n viimeisintä käskyä CPU:n sisällä nopeassa puskurissa
 - u Hyödynnä myös ennaltanoutoa
 - § Hyvällä tuurilla myös hypyn kohdekäsky tuli samalla
 - § Esim. IF-THEN ja IF-THEN-ELSE rakenne
- n Toimii hyvin pienille silmukoille
 - u Nouto muistista vain kerran
- n Parempi alueellinen paikallisuus kuin pelkkää välimuistia käytettäessä

CRAY-1
Motorola 68010



Hyppyjen ennustus (branch prediction)

n Arvaa älykkäästi kumpi todennäköisempää:
hyppy vai ei hyppyä ?

n Staattinen ennustus

u Kiinteästi: aina hypätään

u Kiinteästi: ei koskaan hypätä

§ ~ 50% oikein

u Operaatiokoodin perusteella

§ Selvitetty etukäteen (ennen CPU:n valmistusta)
operaatiokoodit, joilla hyppy yleisempää kuin ei hyppyä

§ Esim. BLE käskyä käytetään tyypillisesti "steppailevan"
for-silmukan lopussa, joten arvaa että hyppää

§ ~ 75% ennusteista oikein

Motorola 68020
VAX 11/780



Hyppyjen ennustus

n Dynaaminen ennustus

- u Muistele miten tämän suorituskerran aikana kävi kyseisen hyppykäskyn kanssa aiemmin
 - § Ennustus osuu nyt paremmin kohdalleen
- u Tarvitsee CPU:n sisälle apumuistia = branch history table
 - § Käskyn osoite
 - § Hypyn kohdeosoite (tai kohdekäsky)
 - § Hypätäänkö vai ei: **taken / not taken**

n "Karvalakkimalli"

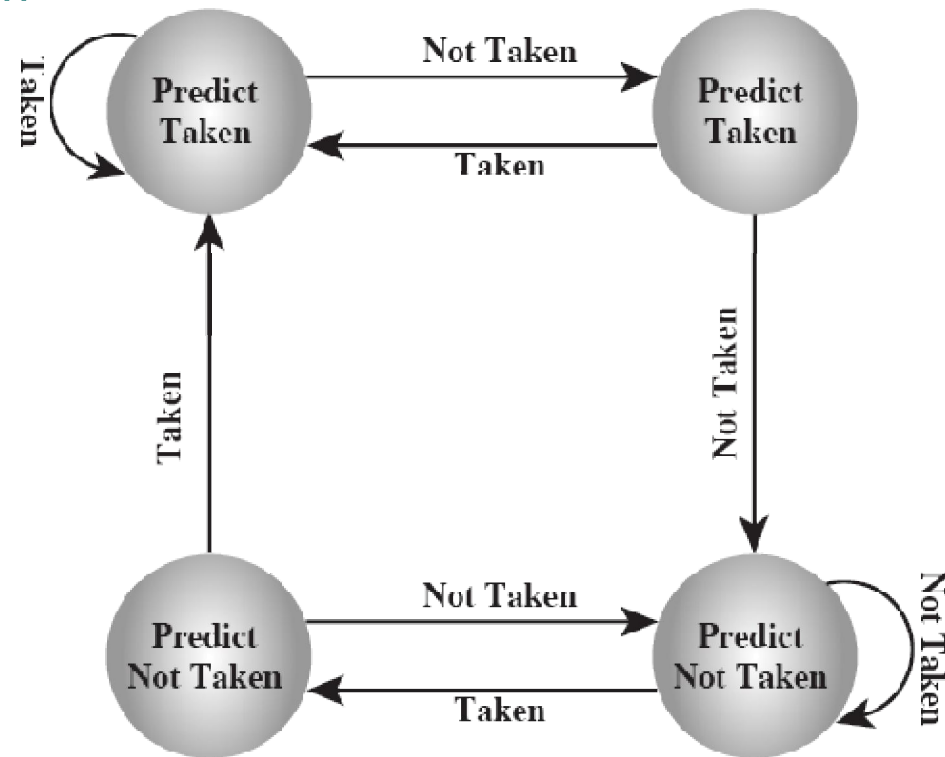
- u Ennusta sen mukaan kuinka kävi edellisellä kerralla
 - § 1 bitti riittää
- u Silmukoissa tulee 1 tai 2 väärää ennustusta



Hyppyjen ennustus

n Parannettu malli

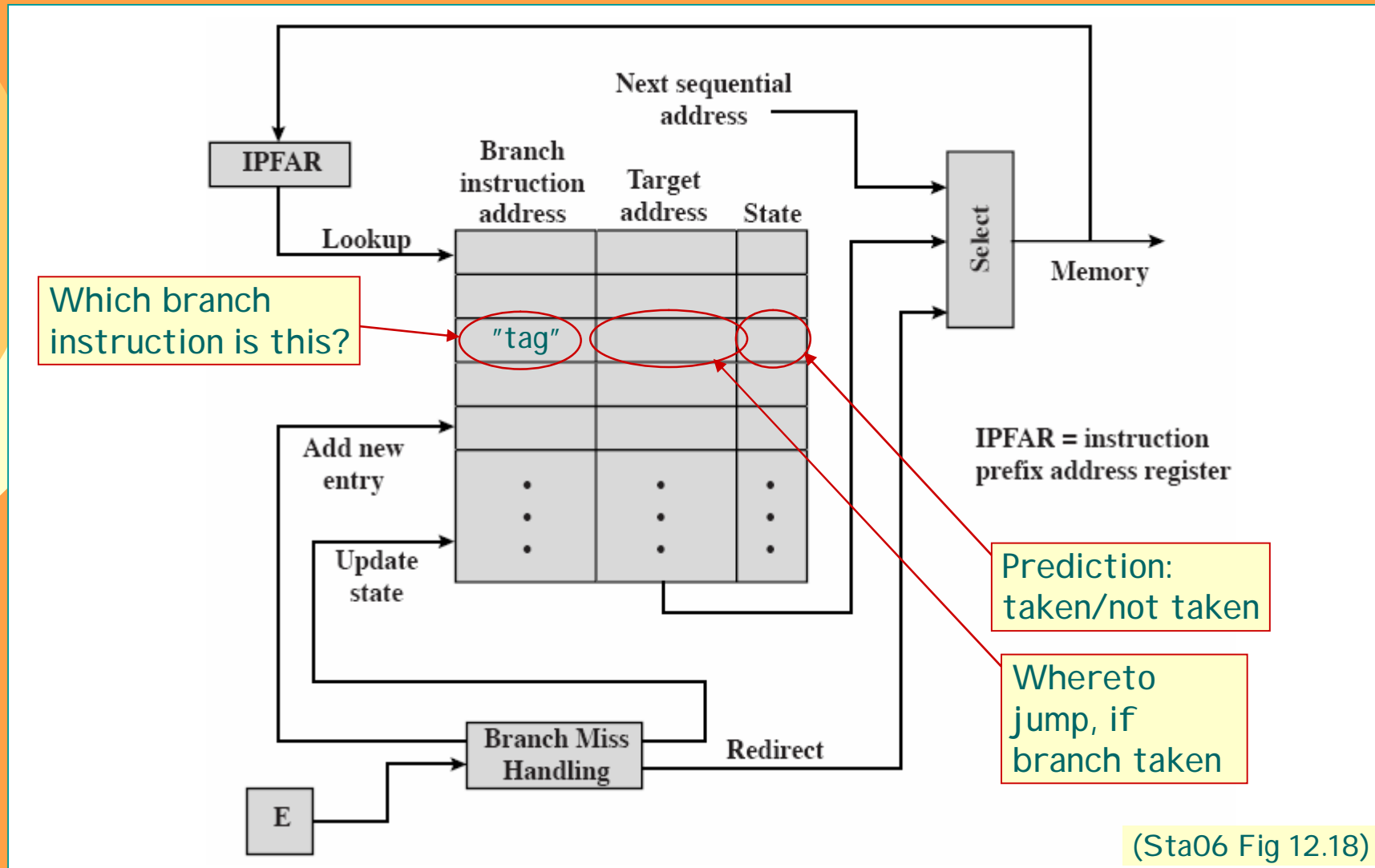
- u Älä muuta ennustetta liian helposti
- u Ennusta kahden edellisen kerran mukaan
- u 2 bittiä riittää



(Sta06 Fig 12.17)



Hyppyjen ennustus





Kertauskysymyksiä

- n Mitä tietoja on sisällytettävä PSW:hen?
- n Miksi 2-vaiheisesta liukuhihnasta ei ole paljon hyötyä?
- n Mitkä tekijät vaikeuttavat liukuhihnan toimintaa?
- n Millaisia ratkaisuja on käytetty hyppykäskyjen vaikutuksen eliminoimiseen?
- n Kuinka CPU siirtyy keskeytyskäsitteeseen?