LUNDS UNIVERSITET

Datorteknik

ERIK LARSSON

Fetch-Execute

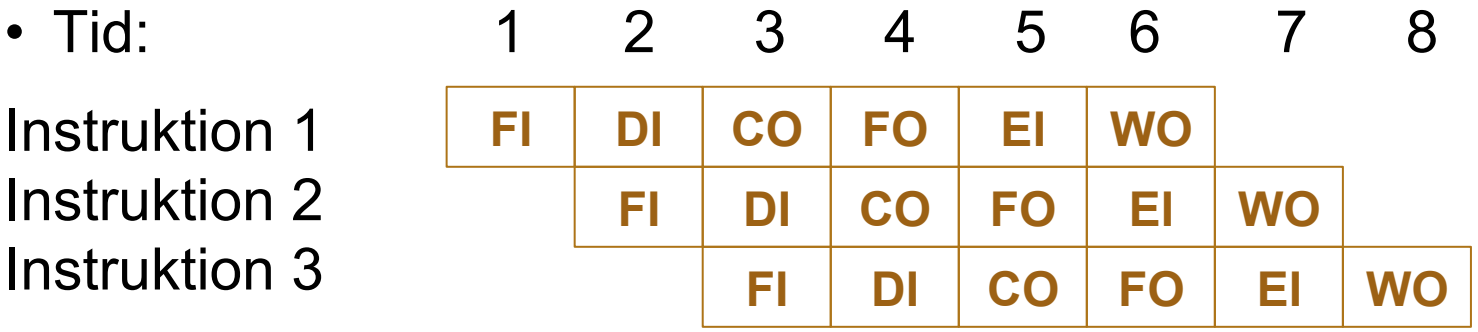
- Utan pipelining:
  - Tid: 1 2 3 4 5 6

- Med pipelining:
  - Tid: 1 2 3 4

LUNDS UNIVERSITET

Pipelining

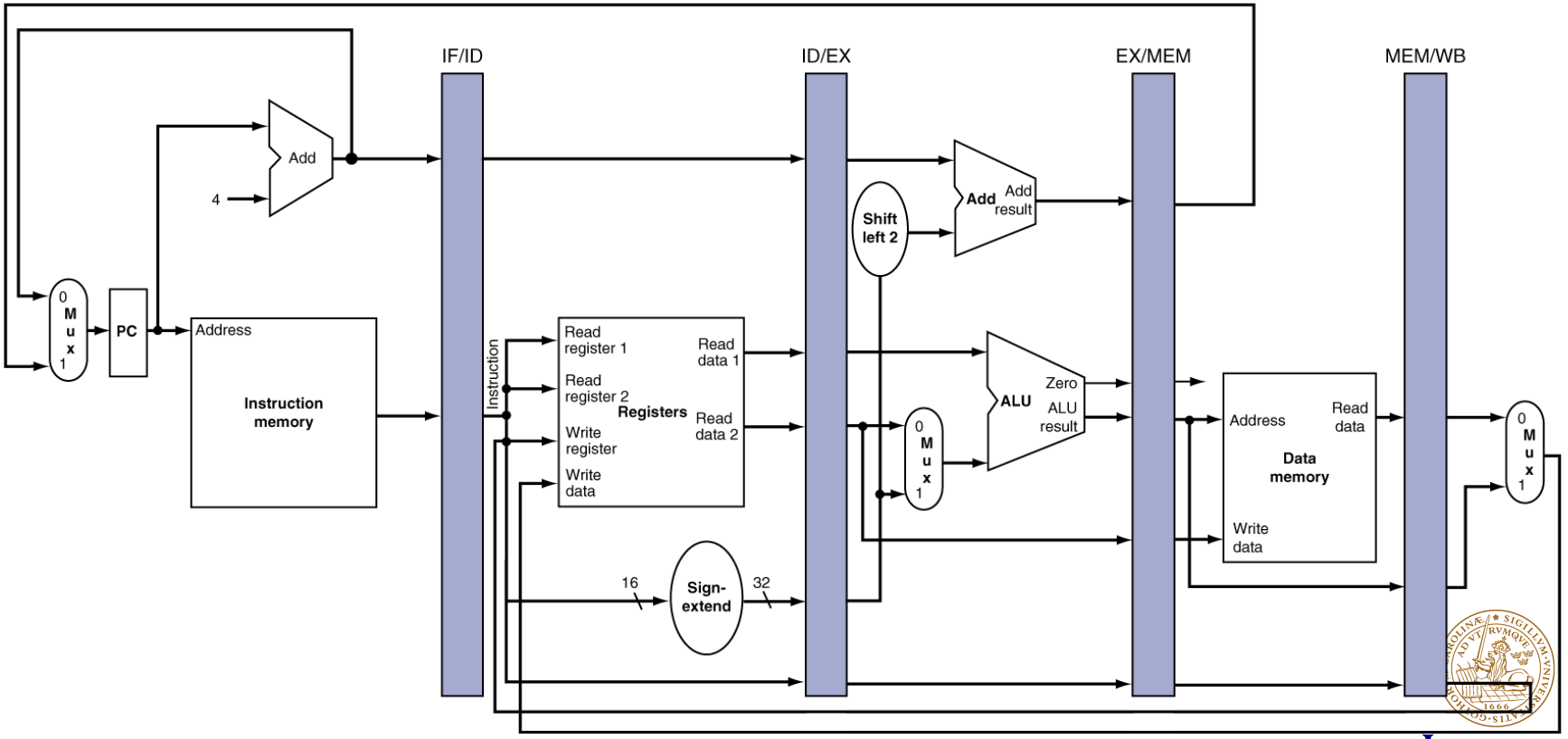
- 6-stegs pipeline:
  - Fetch instruction (FI)
  - Decode instruction (DI)
  - Calculate operand address (CO)
  - Fetch operand (FO)
  - Execute instruction (EI)
  - Write operand (WO)
- Tid:



LUNDS UNIVERSITET

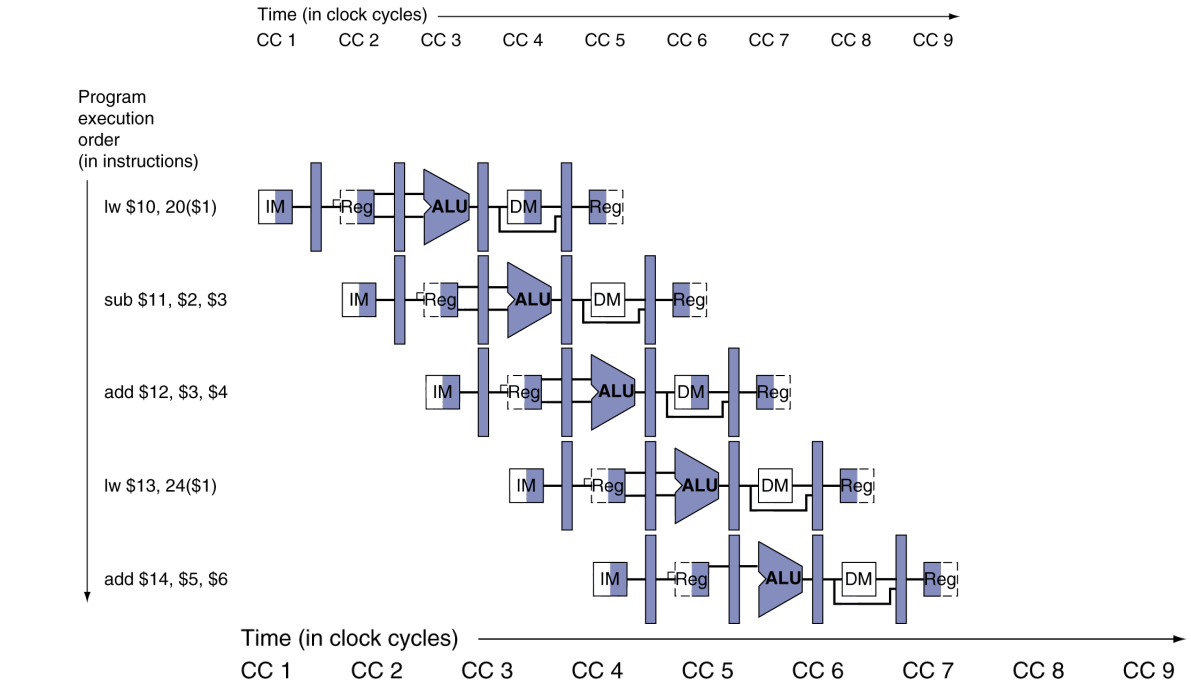
Pipeline diagram (vid en viss tidpunkt)

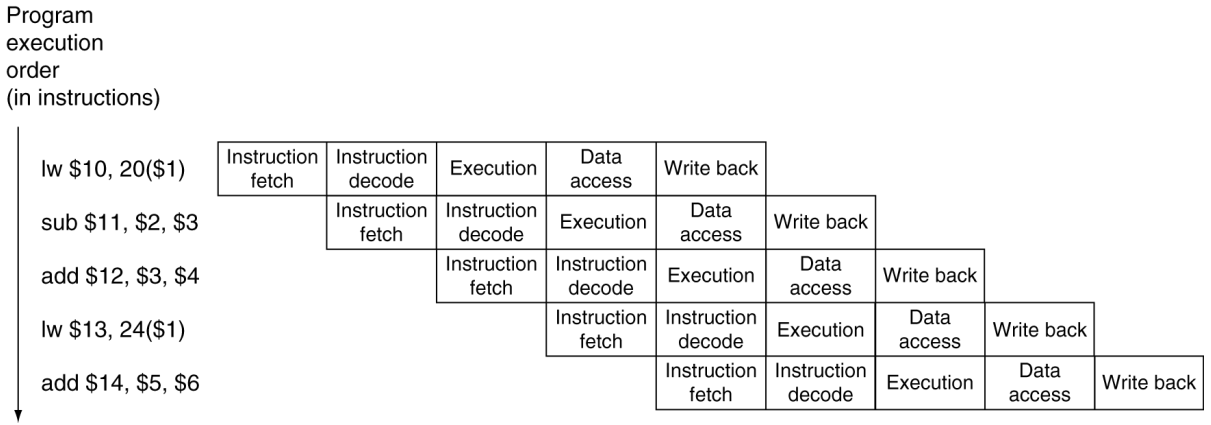
add $14, $5, $6	lw $13, 24 ($1)	add $12, $3, $4	sub $11, $2, $3	lw $10, 20($1)
Instruction fetch	Instruction decode	Execution	Memory	Write-back



LUNDS UNIVERSITET

Pipeline diagram





LUNDS UNIVERSITET

Översikt

- Pipelining
- Pipeline problem (hazards)
  - Strukturella problem (hazards)
  - Data problem (hazards)
  - Kontroll problem (hazards)
- Delayed branching
- Branch prediction
  - Statiskt prediction
  - Dynamisk prediction
  - Branch history table

LUNDS UNIVERSITET

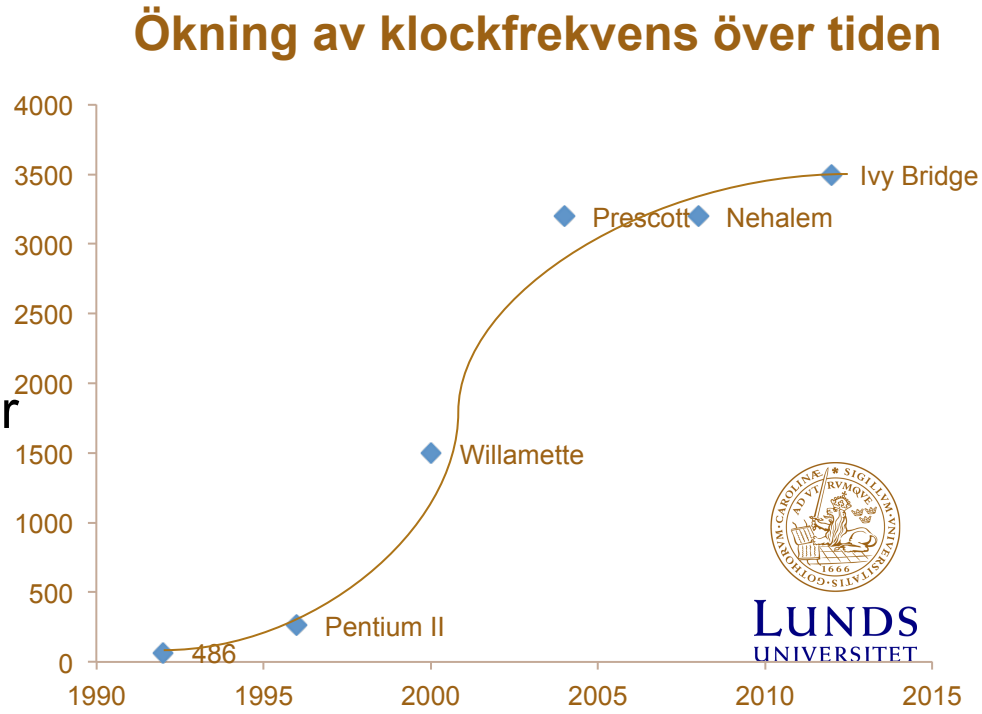
Översikt

- Parallellism
- Instruktions-nivå
  - Superscalar processors
  - Very Long Instruction Word processors
- Trådar

LUNDS UNIVERSITET

Inledning

- Konstant behov av högre prestanda
- Prestanda har uppnåtts genom:
  - Utveckling inom halvledarteknik
  - Tekniker som:
    - » Cacheminne
    - » Flera bussar
    - » Pipelining
    - » Superscalar arkitekturer



Parallellberäkning

- När single core CPUer inte räcker till, fundera på:
  - Fler CPUer (cores)
  - Gemensamma resurser (delat minne (shared memory))
  - Busstruktur
- Avvägningar:
  - Komplexitet av CPUer
  - Gemensama resurser
  - Prestanda på busstruktur

LUNDS UNIVERSITET

Klassificering av datorarkitekturer

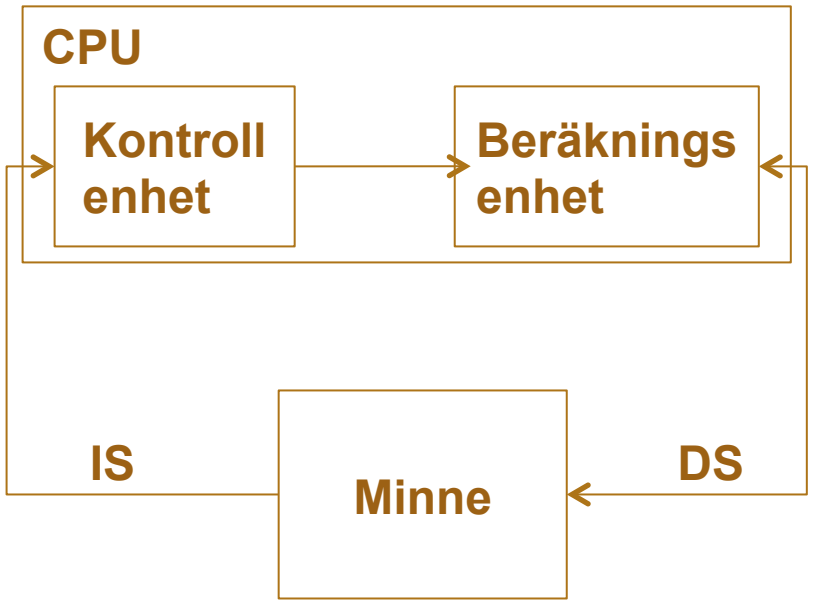
- Baserat på instruktionsflödet och dataflödet, klassficerade Flynn datorer:
  - Single-instruction single-datastream (SISD)
  - Single-instruction multiple-datastream (SIMD)
  - Multiple-instruction single-datastream (MISD)
  - Multiple-instruction multiple-datastream (MIMD)

- Flynn, M., Some Computer Organizations and Their Effectiveness, IEEE Transactions on Computers, Vol. C-21, pp. 948, 1972.

LUNDS UNIVERSITET

Single-instruction single-datastream (SISD)

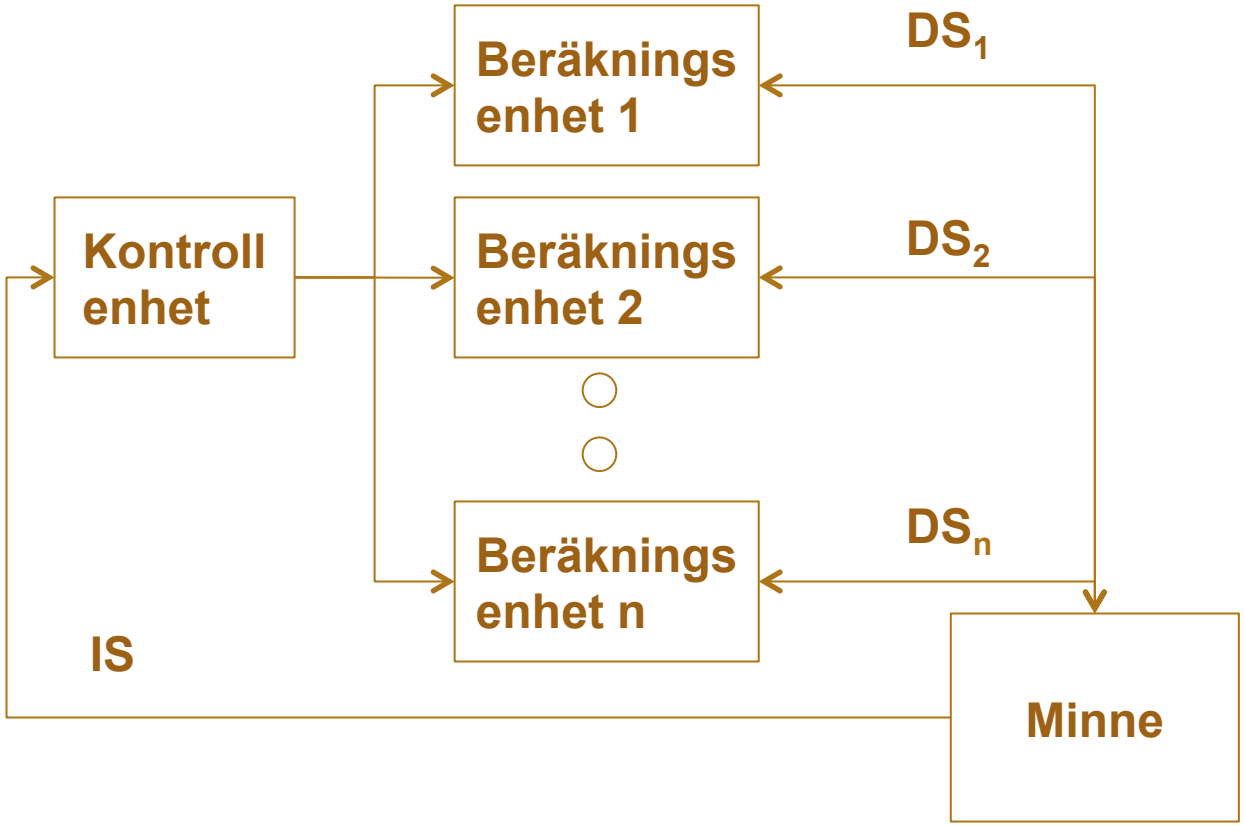
- En ström av instruktioner (IS) och en ström av data (DS)



LUNDS UNIVERSITET

Single-instruction multiple-datastream (SIMD)

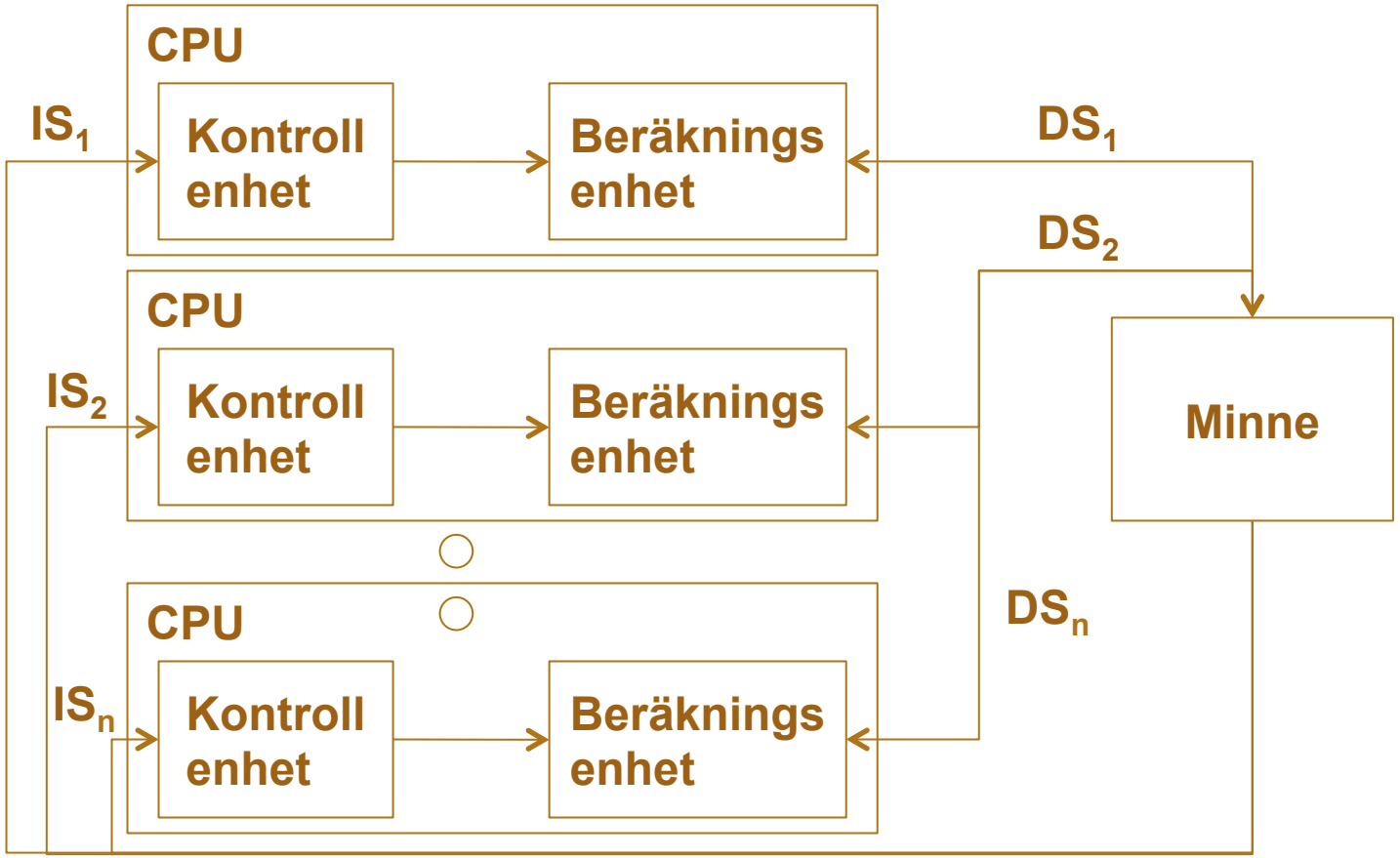
- En ström av instruktioner (IS) och flera strömmar av data (DS)



LUNDS UNIVERSITET

Multiple-instruction multiple-datastream (MIMD)

- Flera strömmar av instruktioner (IS) och flera strömmar av data (DS)



Multicore och multicomputer

- Multicore chips: Flera processorer på varje chip
  - Intel x86: Intel Core Duo, Intel Core i7
- Multicomputers: Flera datorer “sammankopplade”
- Vectorprocessorer; har instruktioner för vektorberäkninar
- Multimedia extentions
  - MMX för Intel x86, VIS UltraSparc, MDMX för MIPS, MAX-2 för HP PA-RISC

LUNDS UNIVERSITET

Prestanda på parallella arkitekturer

- Hur snabbt kan arkitekturer maximallt exekvera?
- Hur snabbt exekverar arkitekturer ett normalt program?
- Hur mäta prestanda på en parallell arkitektur?
- Hur mäta prestanda om ytterligare funktionella beräkningsenheter läggs till?

LUNDS UNIVERSITET

Prestanda på parallella arkitekturer

- Peak rate: den teoretiska gränser på hur snabbt en arkitektur kan exekvera. Ofta har peak rate begränsat intresse eftersom det är svårt att uppnå peak rate
- Speedup (S): mäter vinsten med en parallell arkitektur jämfört med sekventiell exekvering:

  S = Ts / Tp

  där Ts är exekveringstid vid sekventiell exekvering och
  Tp är exekveringstid med parallell exekvering
- Efficiency (E) relaterar speedup till antal processorer, E=S/p där S är speedup och p är antal processorer.
  - Idealt (teori) ska:

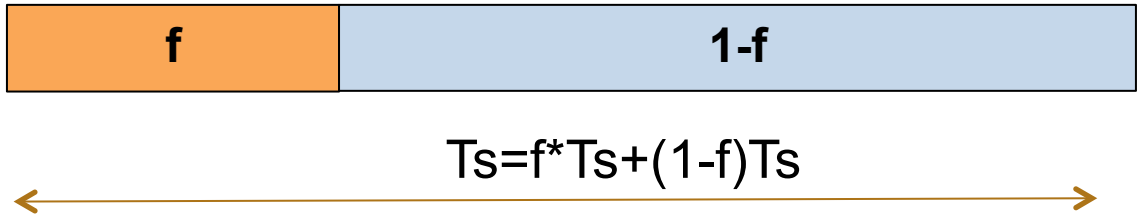
    S= Ts / (Ts/p) = p vilket leder till E=1

LUNDS UNIVERSITET

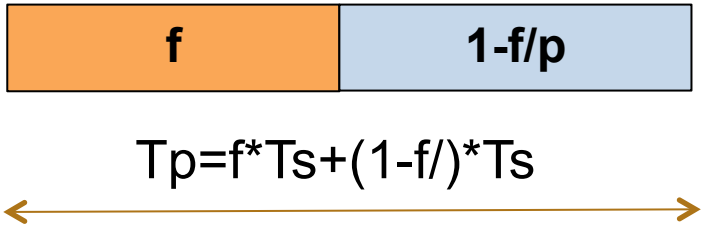
Amdahls lag

- Antag att ett program (algoritm) implementeras så att f är den del som måste exekveras sekventiellt. Då kan resten (1-f) exekveras parallellt.

- En CPU:



- P CPUs:



LUNDS UNIVERSITET

Amdahls lag

- Exekveringstid: $T_p = f*T_s + (1-f)*T_s/p$
- Speedup: $S = T_s/T_p$
- Efficiency: $E = S/p$
- Exempel: om f=20% och arkitekturen har 4 processorer, fås:
  - Speedup $S = 1/(0.2+(1-0.2)/4) = 2.5$
  - Efficiency $E = 1/(0.2*(4-1)+1) = 0.625$
- Även en liten bit sekventiell exekvering (1/f) leder till begränsningar på speedup och efficiency
  - För att öka speedup och efficiency måste mer exekveras parallellt

LUNDS UNIVERSITET

Översikt

- Parallellism
- Instruktions-nivå
  - Superscalar processors
  - Very Long Instruction Word processors
- Trådar

LUNDS UNIVERSITET

Superscalar architecture

- En superscalar architecture (SSA) tillåter att mer än en instruktion initieras samtidigt och exekveras oberoende av varandra.
  - Skalär: heltal, flyttal
  - Icke-skalär: array, matris, vektor
- I pipelining är flera instruktioner aktiva men de är alla i olika pipeline steg
- Utöver pipelining, tillåter SSA att flera instruktioner exekverar samtidigt (i samma pipeline steg). Det är möjligt att initiera flera instruktioner i samma klockcykel; instruction level parallelism (ILP)

LUNDS UNIVERSITET

Superscalar architecture (SSA)

- Ett exempel, tre parallella enheter:
  - en flyttalsinstruktion och två heltalsinstruktioner

  Kan exekveras samtidigt (som tidigare kan flera instruktioner vara aktiva i en pipeline)

Flyttal:	FI	DI	CO	FO	EI	WO
Heltal:	FI	DI	CO	FO	EI	WO
Heltal:	FI	DI	CO	FO	EI	WO

LUNDS UNIVERSITET

Instruktionsfönster

- Instruktions fönster (instruction window) är de instruktioner som processorn har att välja mellan vid en viss tidpunkt.
- Instruktions fönstret ska vara så stort som möjligt för att ge möjlighet att effektivt välja instruktioner som ska exekveras
- Problem:
  - Svårt att hämta instruktioner i hög hastighet
  - Konflikter i pipeline (branches, struktural, data)

LUNDS UNIVERSITET

Problem med parallell exekvering

- Problem som förhindrar parallell exekvering i SSA liknar de problem som skapade problem i pipelines
- Konflikter i pipelines:
  - Resource (resurs) konflikter
  - Kontrollkonflikter
  - Datakonflikter
    - » True data dependency
    - » Output dependency
    - » Anti-dependency

LUNDS UNIVERSITET

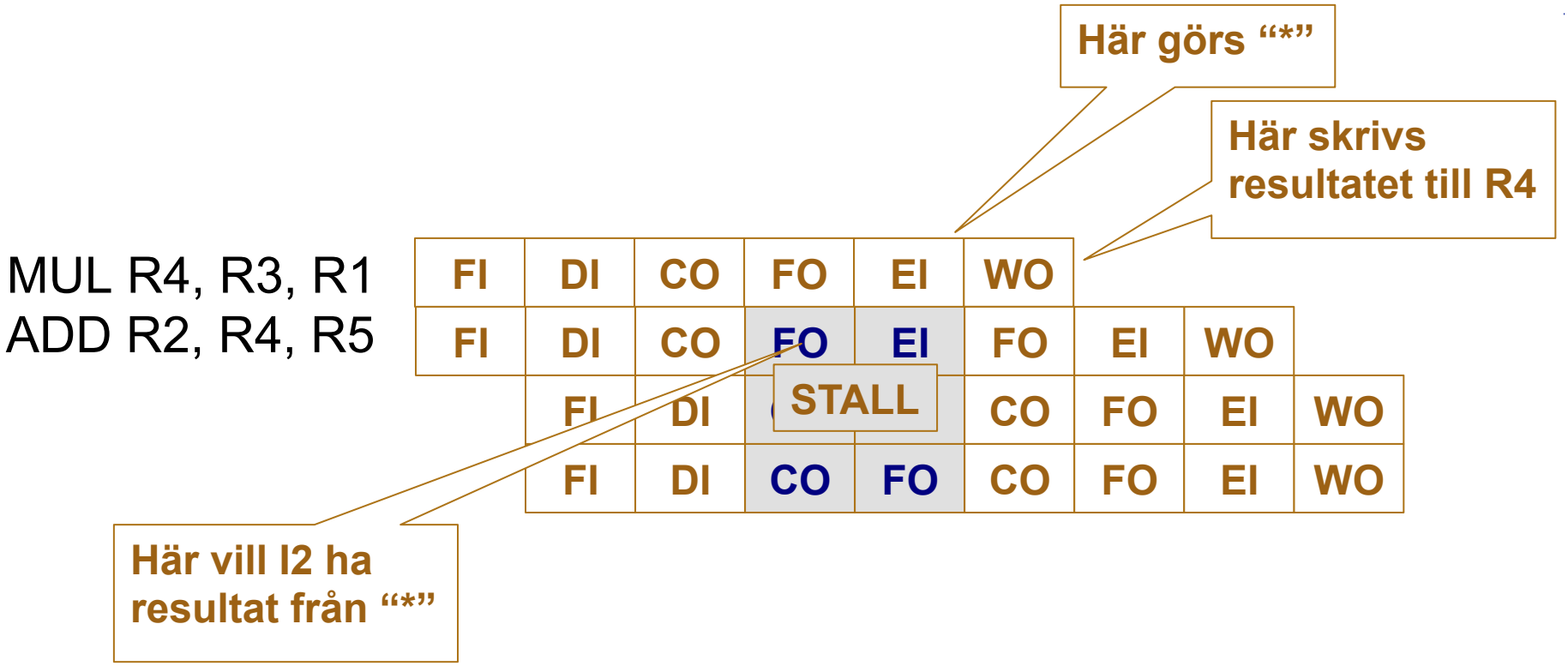
True data dependency

- True data dependency (data konflikt/hazard) uppkommer när resultat (output) från en instruktion behövs som indata i följande instruktion:

```
MUL R4, R3, R1        R4 ← R3 * R1
--------
ADD R2, R4, R5        R2 ← R4 + R5
```

LUNDS UNIVERSITET

True data dependency



LUNDS UNIVERSITET

Output dependency

- Output dependency existerar om två instruktioner skriver till samma plats. Om den senare instruktionern exekveras före den första blir det fel:

```
MUL R4, R3, R1      R4 ← R3 * R1
--------
ADD R4, R2, R5      R4 ← R2 + R5
```

LUNDS UNIVERSITET

Anti-dependency

- En anti-dependency existerar om en instruktion använder en plats (minne/register) medan någon nästföljande instruktion skriver till den platsen.
- Om den första instruktionen inte är klar medan en efterföljande instruktion skriver på samma plats, blir det fel:

```
MUL R4, R3, R1      R4 ← R3 * R1
--------
ADD R3, R2, R5      R3 ← R2 + R5
```

LUNDS UNIVERSITET

Register renaming: Output dependency och anti-dependency

- Output dependency
- Givet var:

```
MUL R4, R3, R1     R4 ← R3*R1
--------
ADD R4, R2, R5     R4 ← R2+R5
```

- Med nya register:

```
MUL R4, R3, R1     R4 ← R3*R1
--------
ADD R7, R2, R5     R7 ← R2+R5
```

- Anti dependency
- Givet var:

```
MUL R4, R3, R1     R4← R3*R1
--------
ADD R3, R2, R5     R3← R2+R5
```

- Med nya register:

```
MUL R4, R3, R1     R4←R3*R1
--------
ADD R6, R2, R5     R6←R2+R5
```

LUNDS UNIVERSITET

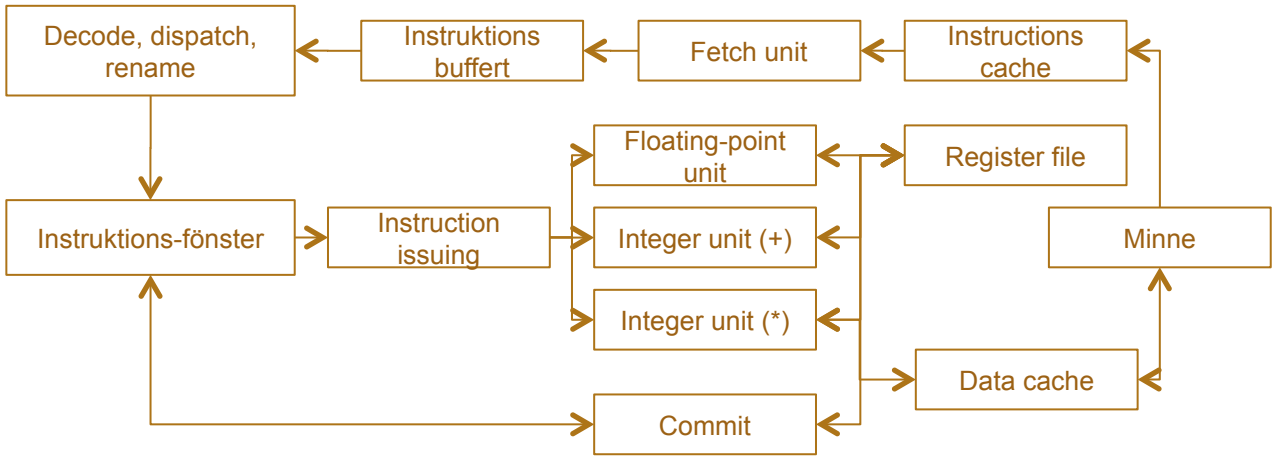
Parallell exekvering

- Instruktioner kan alltså exekveras i en annan ordning än den som är given i programmet. Detta är inget problem så länge resultatet blir rätt.
- Exekveringspolicies:
  - In-order issue with in-order completion
  - In-order issue with out-of-order completion
  - Out-of-order issue with out-of-order completion

LUNDS UNIVERSITET

Parallell exekvering

- Antag en superscalar processor:
  - Två instruktioner kan hämtade (fetch) och avkodade (decoded)
  - Tre funktionella enheter kan arbeta parallellt; en flyttals enhet, en heltals adderare, och en heltals multiplicerare
  - Resultat från två instruktioner kan skrivas tillbaka (written back) samtidigt



LUNDS UNIVERSITET

Parallell exekvering

- Antag ett program:

```
I1: ADDF R12, R13, R14    R12 ← R13 + R14 (float. pnt.)
I2: ADD  R1, R8, R9       R1 ← R8 + R9
I3: MUL  R4, R2, R3       R4 ← R2 * R3
I4: MUL  R5, R6, R7       R5 ← R6 * R7
I5: ADD  R10, R5, R7      R10 ← R5 + R7
I6: ADD  R11, R2, R3      R11 ← R2 + R3
```

I3 och I4 är i konflikt om samma funktionella enhet (*)

I5 behöver resultatet från beräkningen i I4

I2, I5 och I6 är i konflikt om samma funktionella enhet (+)

- Alla instruktioner kräver 1 klockcykel för exekvering förutom I1 som kräver två klockcykler

LUNDS UNIVERSITET

In-order issue with in-order completion

I1: ADDF R12, R13, R14	R12 ← R13 + R14 (flyttalsberäkning)
I2: ADD R1, R8, R9	R1 ← R8 + R9
I3: MUL R4, R2, R3	R4 ← R2 * R3
I4: MUL **R5**, R6, R7	R5 ← R6 * R7
I5: ADD R10, **R5**, R7	R10 ← R5 + R7
I6: ADD R11, R2, R3	R11 ← R2 + R3

Decode/ Issue		Execute			Writeback/ complete		Cykel
		FP	ADD	MUL			
I1	I2						1
I3	I4	I1	I2				2
I5	I6	I1				STALL	3
				I3	I1	I2	4
				I4	I3		5
			I5		I4		6
			I6		I5		7
					I6		8

LUNDS UNIVERSITET

In-order issue med out-of-order completion

I1: ADDF R12, R13, R14	R12 ← R13 + R14 (flyttalsberäkning)
I2: ADD R1, R8, R9	R1 ← R8 + R9
I3: MUL R4, R2, R3	R4 ← R2 * R3
I4: MUL **R5**, R6, R7	R5 ← R6 * R7
I5: ADD R10, **R5**, R7	R10 ← R5 + R7
I6: ADD R11, R2, R3	R11 ← R2 + R3

Decode/ Issue		Execute			Writeback/ complete		Cykel
		FP	ADD	MUL			
I1	I2						1
I3	I4	I1	I2				2
I5	I6	I1			I2		3
				I3	I1		4
				I4	I3		5
			I5		I4		6
			I6		I5		7
					I6		8

Out-of-order completion

Out-of-Order Issue (OOI) med Out-of-Order Completion (OOC)

I1: ADDF R12, R13, R14	R12 ← R13 + R14 (flyttalsberäkning)
I2: ADD R1, R8, R9	R1 ← R8 + R9
I3: MUL R4, R2, R3	R4 ← R2 * R3
I4: MUL R5, R6, R7	R5 ← R6 * R7
I5: ADD R10, R5, R7	R10 ← R5 + R7
I6: ADD R11, R2, R3	R11 ← R2 + R3

I6 kan starta före I5 och parallellt med I4. Programmet tar bara 6 cykler (jämfört med 8 i fallet med in-order issue and in-order completion)

Decode/Issue		Execute			Writeback/ complete		Cykel
		FP	ADD	MUL			
I1	I2						1
I3	I4	I1	I2				2
I5	I6	I1		I3	I2		3
			I6	I4	I1	I3	4
			I5		I4	I6	5
					I5		6
							7
							8

LUNDS UNIVERSITET

Konflikthantering

- Med in-order issue med in-order-completion behöver processorn bara hantera true data dependency, men inte output dependency och anti-dependency.
- Med out-of-order issue med out-of-order completion behöver processorn hantera true data dependency, output dependency och anti-dependency
  - Output dependency

```
MUL R4, R3, R1      R4 ← R3 * R1
- - - -
ADD R4, R2, R5      R4 ← R2 + R5
```

  - Antidependency

```
MUL R4, R3, R1      R4 ← R3 * R1
- - - -
ADD R3, R2, R5      R3 ← R2 + R5
```

Sammanfattning

- Experiment har visat:
  - Att enbart lägga till extra enheter (typ pipelines) är inte effektivt;
  - Out-of-order är väldigt viktigt; det tillåter look ahead för “independent” instruktioner;
  - Register renaming kan förbättra prestanda med mer än 30%; in this case performance is limited only by true dependencies.
  - Viktigt med fetch/decode där instruktions fönstret är tillräckligt stort

LUNDS UNIVERSITET

Översikt

- Parallellism
- Instruktions-nivå
  - Superscalar processors
  - Very Long Instruction Word processors
- Trådar

LUNDS UNIVERSITET

Superscalar processorer

- Bra:
  - Hårdvaran löser allt:
    - » Hårdvara detekterar potentiell parallellism av instruktioner
    - » Hårdvara försöker starta exekvering (issue) av så många instruktioner som möjligt parallellt
    - » Hårdvara löser register renaming
  - Kompatibla binärer:
    - » Om en ny funktionell enhet läggs till i en ny version av arkitekturen eller några andra förbättringar göra (utan att ändra instruktionsuppsättningen) så kommer äldre program (kompilerade för äldre arkitektur) att kunna dra nytta av förbättringar eftersom den nya hårdvaran kan exekverar det gamla programmet (dess ordning av instruktioner) mer effektivt.

LUNDS UNIVERSITET

Superscalar processorer

- Problematiskt:
  - Väldigt komplext:
    - » Behövs mycket hårdvara för ”run-time” detektering av möjlig parallellism
    - » Effektförbrukningen kan bli hög, pga mycket hårdvara
  - Begränsat instruktions fönster:
    - » Det gör att det blir svårare att hitta instruktioner som har möjlighet att exekveras parallellt.

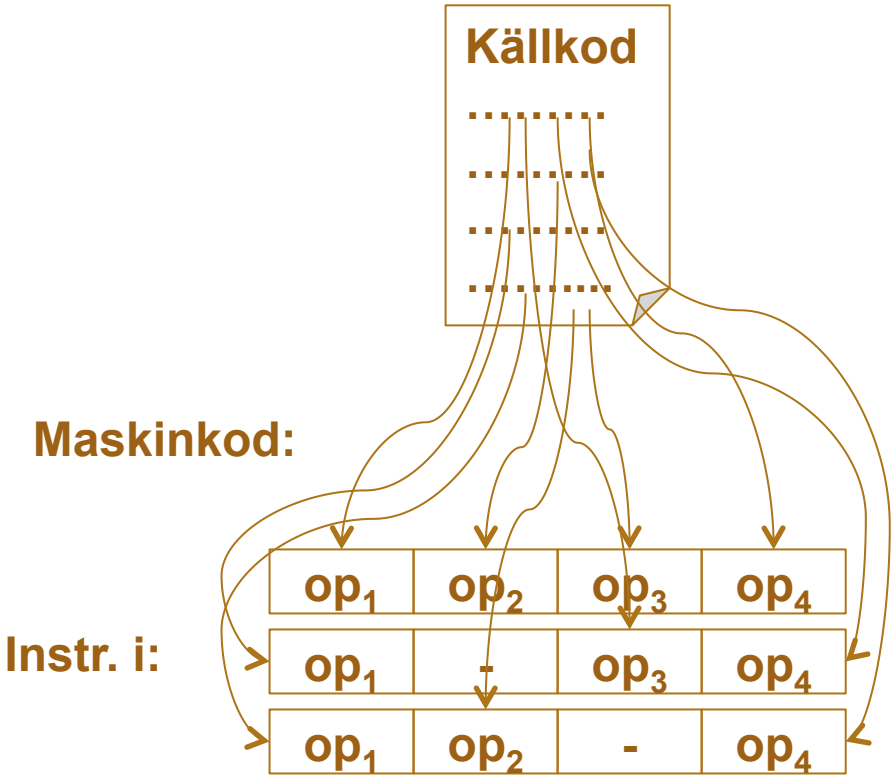
LUNDS UNIVERSITET

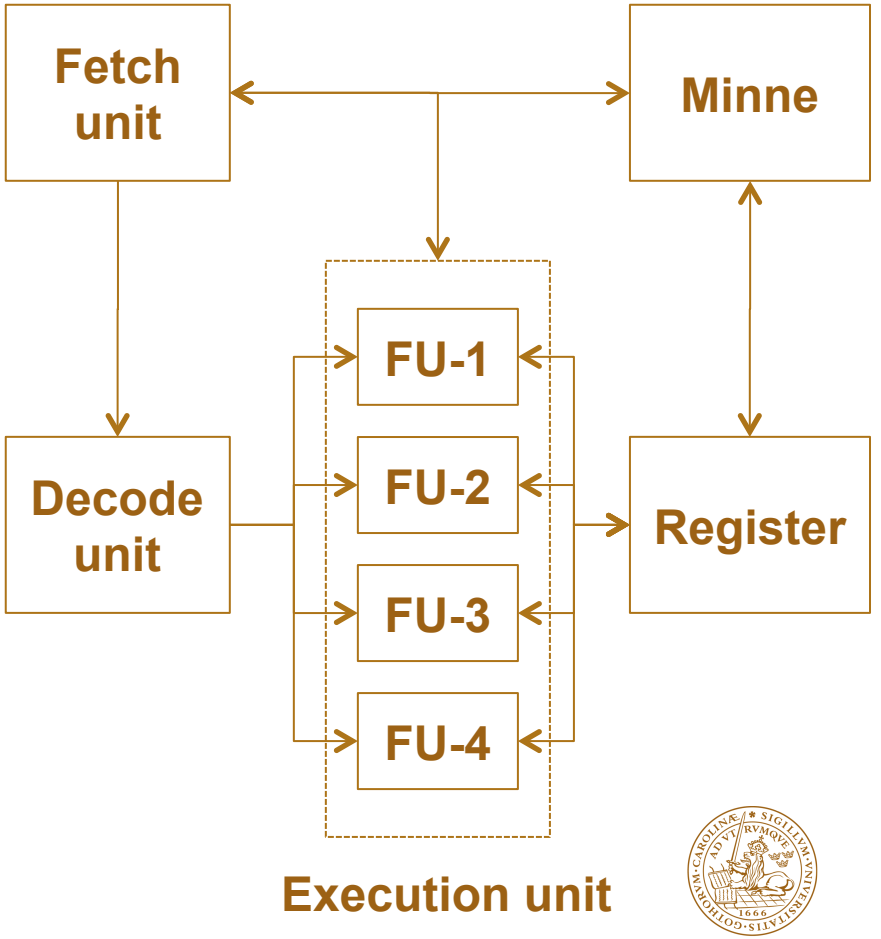
VLIW processorer

- VLIW processorer använder kompilatorn för detektering av parallellism
  - Kompilatorn analyserar och detekterar operationer som kan exekveras parallellt. Dessa operationer packas in i en ”large” instruktion
- När en ”long” instruktion hämtats, exekveras alla dess operationer parallellt
- Ingen hårdvara behövs för ”run time” detektering av parallellism
- Problemet med litet instruktions fönster är löst; kompilatorn kan vid kompilering använda hela programmet för detektera parallellism.

LUNDS UNIVERSITET

VLIW processorer





LUNDS UNIVERSITET

VLIW processorer

- Fördelar:
  - Enklare hårdvara.
    - » Antalet FU kan ökas utan behov av mer och sofistikerad hårdvara för att detektera parallellism
    - » Effektförbrukning kan reduceras
  - Kompilatorn kan detektera parallellism genom en global analys av hela programmet (inget problem med begränsat instruktions fönster)

LUNDS UNIVERSITET

VLIW processorer

- Nackdelar:
  - Många register behövs för att hålla alla FU aktiva (lagring av operander och resultat)
  - Stor bandbredd behövs mellan:
    - » FU och register
    - » Register och minne
    - » Instruktions cache och fetch unit (exempel: en instruktion med 7 operationer där varje operation behöver 24, krävs 168 bitar per instruktion)

LUNDS UNIVERSITET

VLIW processorer

- Nackdelar:
  - Stora program (många instruktioner) pga icke använda operationer i instruktioner
  - Binär (exekverbar kod) är inte kompatibel. Om en ny processor har fler FU så kan parallellism öka (fler operationer). Problemet är att instruktionerna ändras. Gammal kod kan inte exekvera på den nya processorn. Och om den kan det så utnyttjas inte alla FUs.

LUNDS UNIVERSITET

Sammanfattning

- För att möta det konstanta behov av prestanda så har super scalar processorer blivit väldigt komplexa
- VLIW processorer undviker komplexitet genom att låta kompilatorn hantera detektering av parallellism
- Operationsnivå parallellism ges direkt av instruktionerna i VLIW processorer
- I och med att en VLIW processor inte behöver hårdvara för att hantera parallellism kan VLIW processorer ha fler funktionella enheter för att öka möjligheter till parallellism. Dock, leder det till att behoven ökar för fler register och mer bandbredd för kommunikation

LUNDS UNIVERSITET

Sammanfattning

- För att hålla alla funktionella enheter aktiva, måste kompilatorer för VLIW vara offensiva i sökandet efter parallellism. Tekniker som används är:
  - Loop unrolling; flera iterationer unrollas för att hanteras parallellt
  - Trace scheduling (villkorliga hopp); kompilatorn försöker förutse (predikt) vilken branch som tas. Instruktioner i den branchen schemaläggs så att de exekveras så snabbt som möjligt. Kompensationskod kan behöva läggas till för att få programmet korrekt. Detta leder till extra tid när dessa instruktioner exekveras

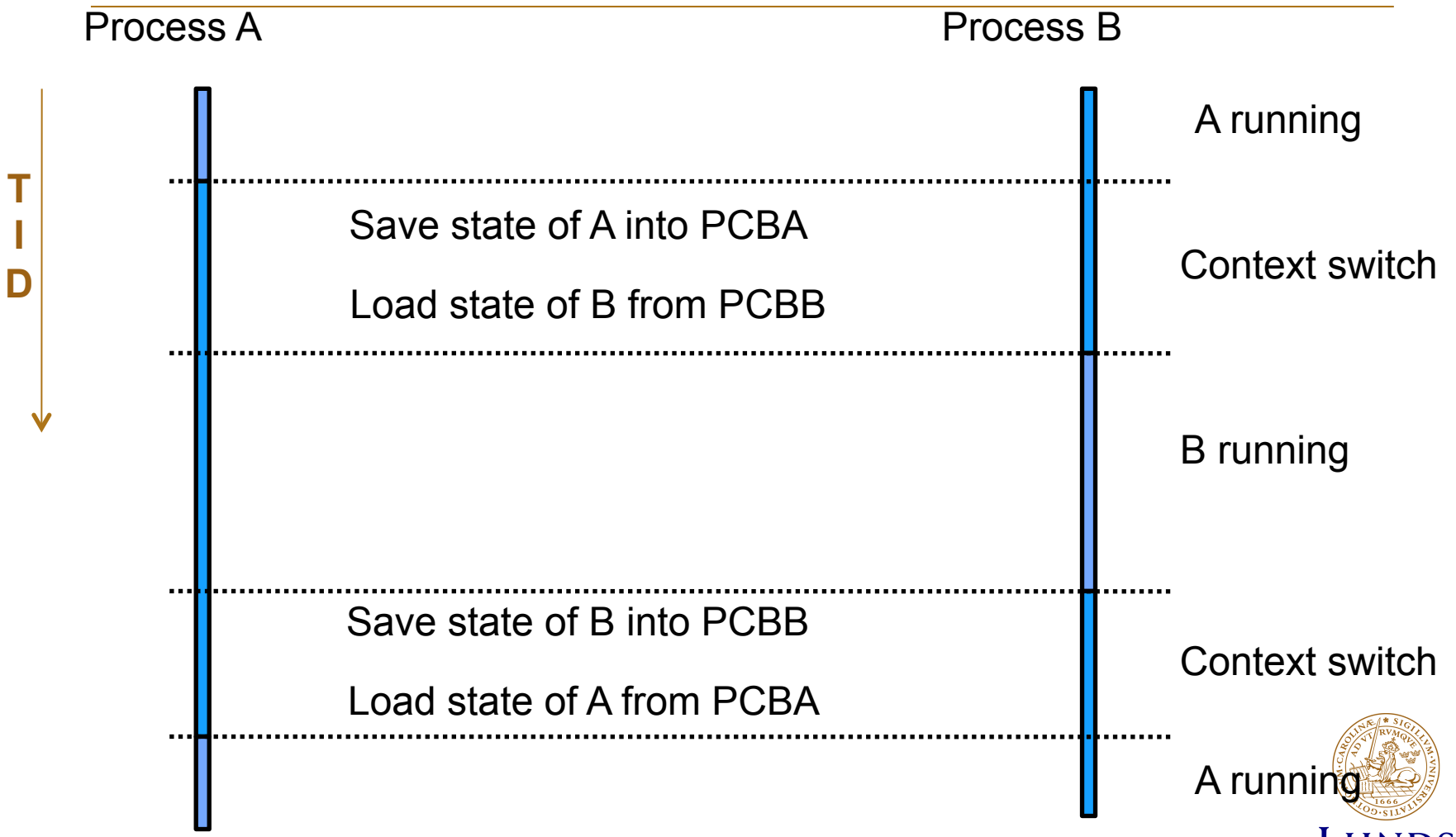
LUNDS UNIVERSITET

Översikt

- Parallellism
- Instruktions-nivå
  - Superscalar processors
  - Very Long Instruction Word processors
- Trådar

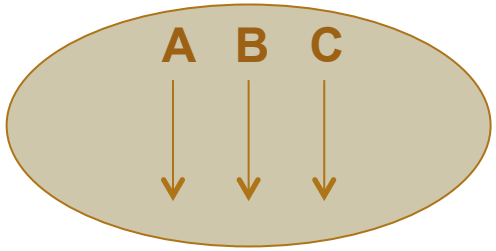
LUNDS UNIVERSITET

Kontextbyte (context switch)



Processer och trådar

- En process består av en eller flera trådar (threads) där en tråd är sekventiell kod som kan exekvera parallellt med andra trådar



- Alla trådar i en process delar data och stack, vilket gör att byte av tråd är mindre kostsamt än byte av process
- Hårdvarustöd:

  Programräknare och register per tråd
  Instruktionshämtning (fetch) på trådbasis
  Kontextbyte (byte av tråd)

- Effektiv exekvering av program med flera trådar (multithreading)
- Effektivt utnyttjande av processorns resurser

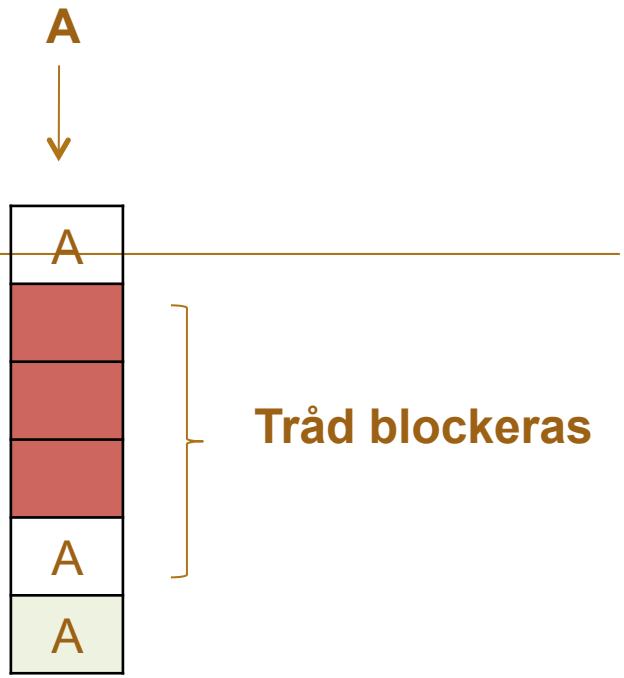
Processer och trådar

- Flera trådar (multithreading)
  - Större chans att kunna exekvera instruktioner parallellt
  - När en tråd blockeras (wait) på grund av till exempel minnesaccess så kan instruktioner från en annan tråd exekvera. Fördelen är att fördröjning på grund av minnesaccess (t ex cache miss), databeroende kan gömmas
- Multithreading kan användas på arkitekturer som är såväl skalära (scalar) och superskalära (superscalar)
- Multithreading:
  - Interleaved multithreading
  - Blocked multithreading
  - Simultaneous multithreading (SMT)

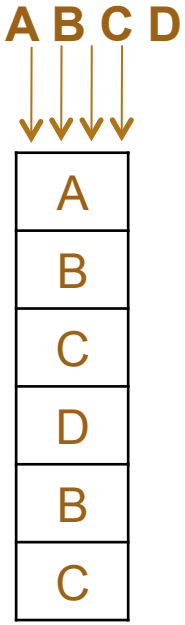
LUNDS UNIVERSITET

Multithreading

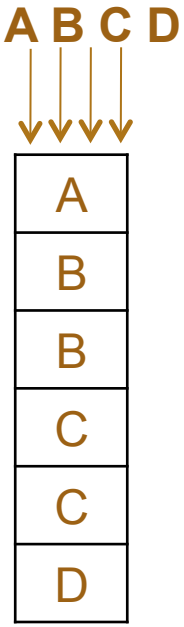
- Scalar (icke-superscalar) processor
  - En tråd



- Interleaved multithreading

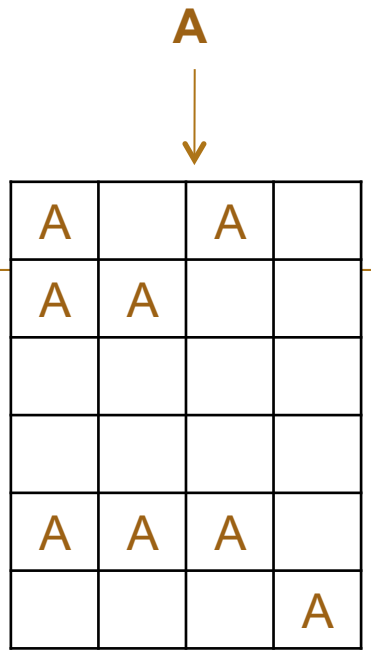


Blocked multithreading

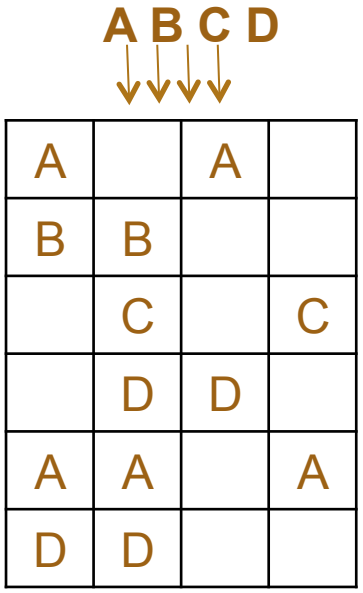


Multithreading

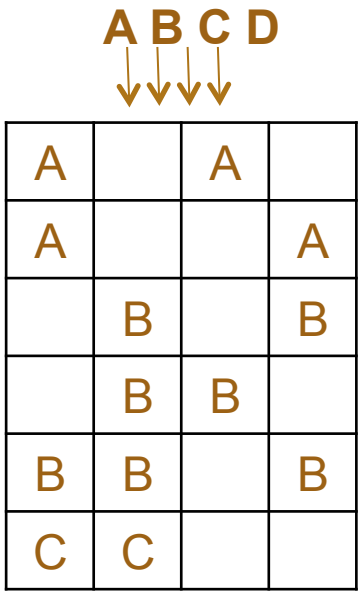
- Superscalar processor
  - En tråd



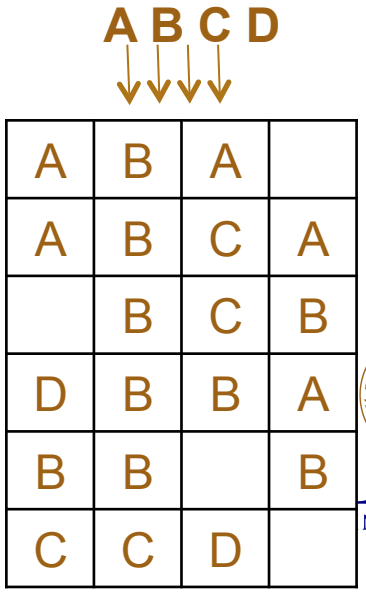
- Interleaved multithreading



Blocked multithreading



Simultaneous multithreading (SMT)



LUNDS UNIVERSITET

Sammanfattning

- Högre prestanda kan inte längre fås med högre klockfrekvens
- Parallella datorer samarbetar flera processorer för att lösa en uppgift
- Parallella program underlättar exekvering i en parallell arkitektur
- Datorer kan klassas som SISD, SIMD och MIMD
- Prestandan beror inte bara på antal beräkningsenheter, utan också hur “parallellt” ett program är
- Multithreading processorer ger via hårdvarustöd möjlighet att exekvera flera trådar

LUNDS UNIVERSITET

Hur många cores i framtiden?

- Moores lag (antal transistorer fördubblas var 18/24 månad) gäller….
- Transistorer blir mindre
- Men, Dennard skalning (konstant effektförbrukning per chip) gäller inte….
- Få komplexa cores, eller många enkla?
- Vem ska “sköta” utnyttjandet?
  - HW, kompilator, programmerare
- 3D-Stacked ICs

LUNDS UNIVERSITET

LUNDS
UNIVERSITET