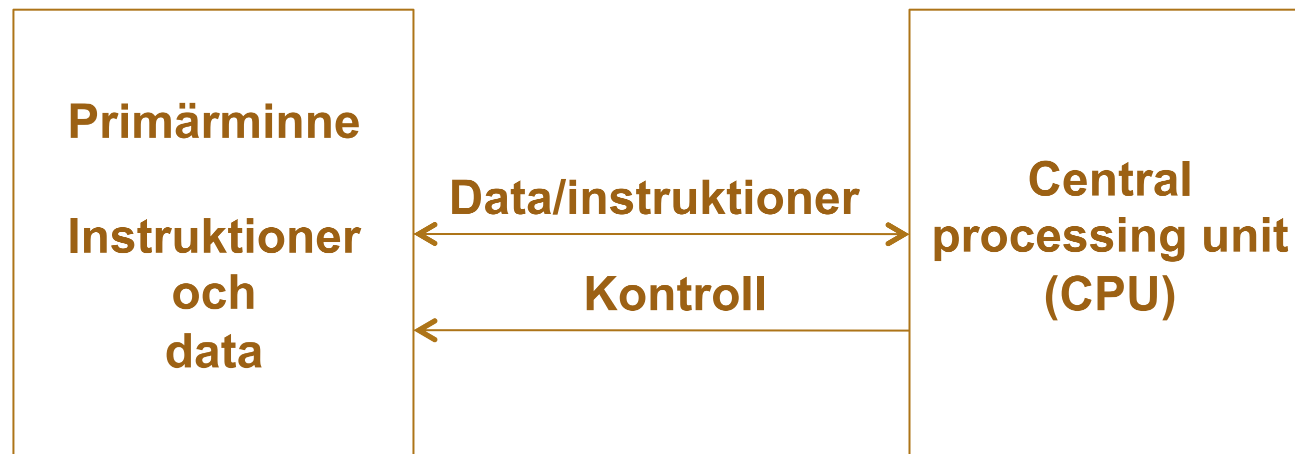
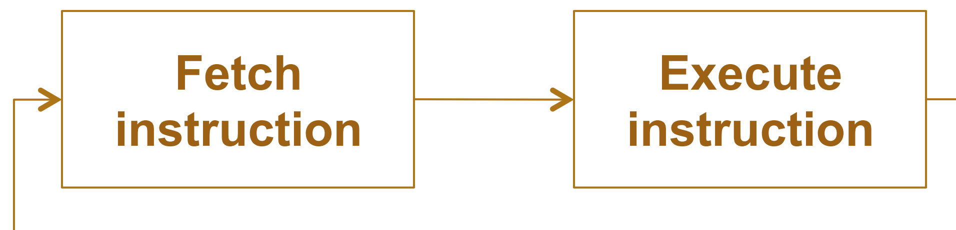
Digitala System: Datorteknik

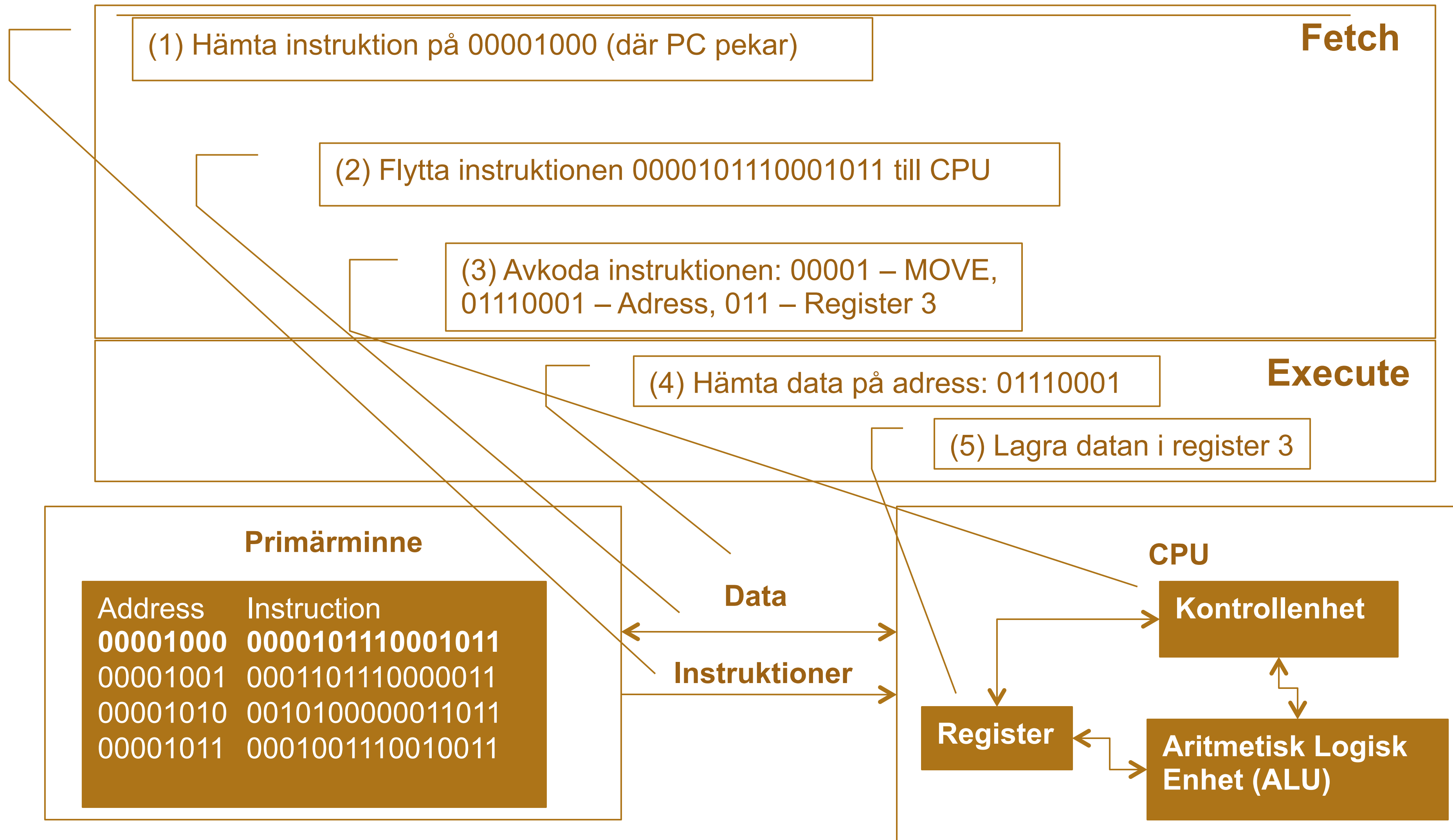
ERIK LARSSON

Dator





Programexekvering

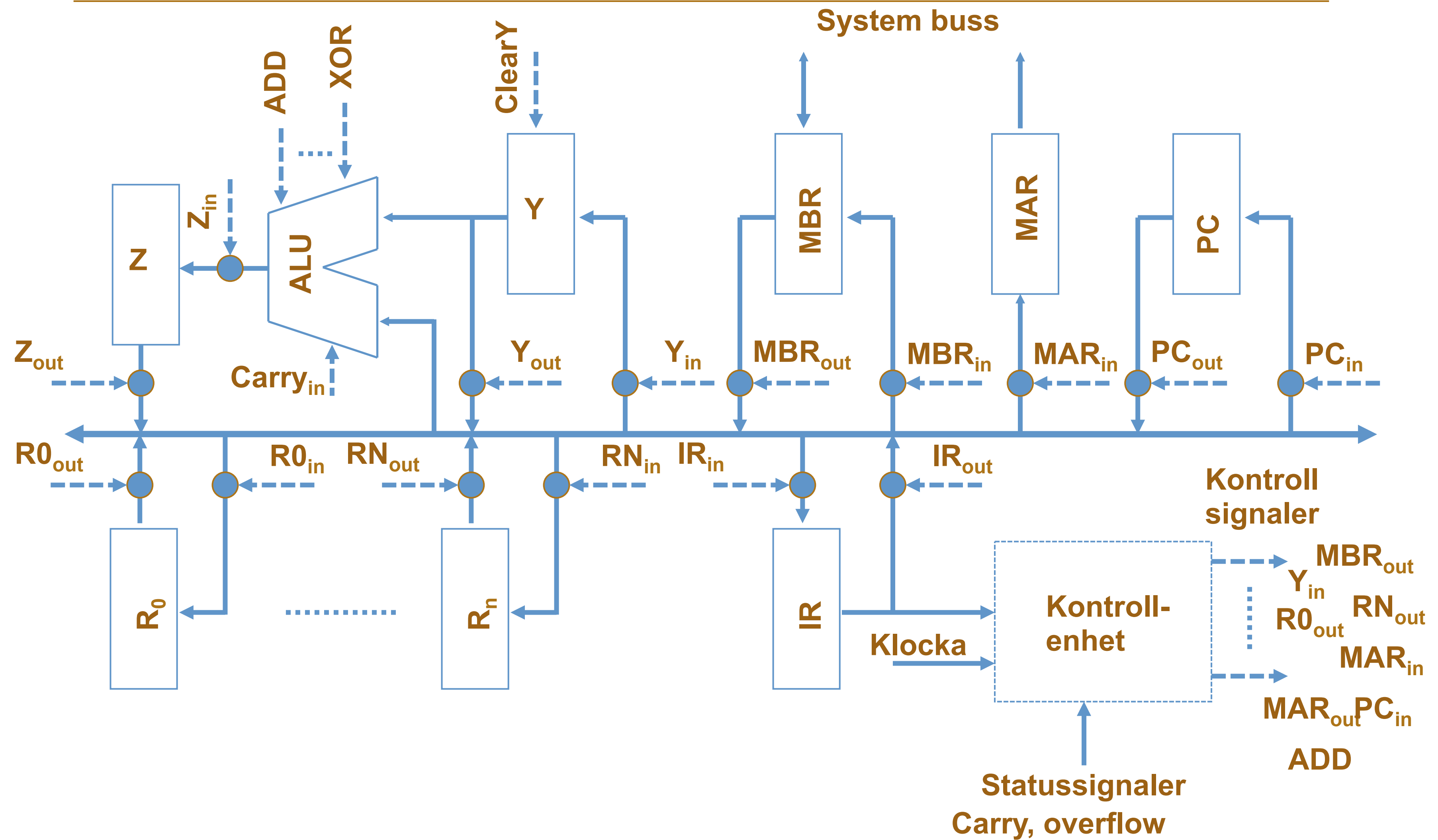


Kontrollenhet

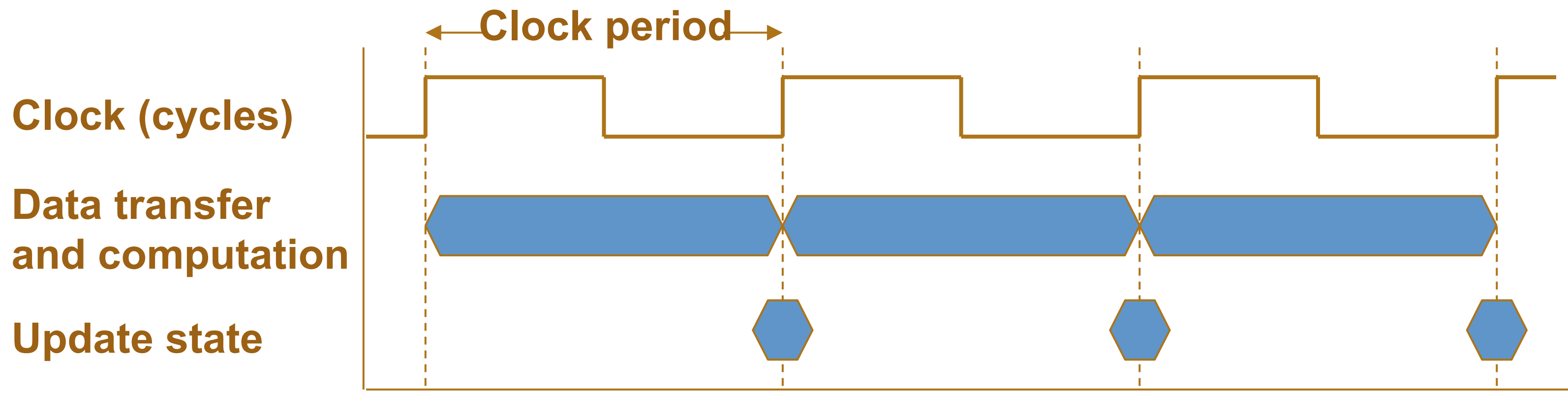
- Instruktion:
  - ADD R1, R3   // R1 <- R1 + R3
- Kontrollsteg:
  1. PC_{out}, MAR_{in}, Read, Clear Y, Carry-in, Add, Z_{in}
  2. Z_{out}, PC_{in}
  3. MBR_{out}, IR_{in}
  4. $R1_{out}$, Y_{in}
  5. $R3_{out}$, Add, Z_{in}
  6. Z_{out}, $R1_{in}$, End

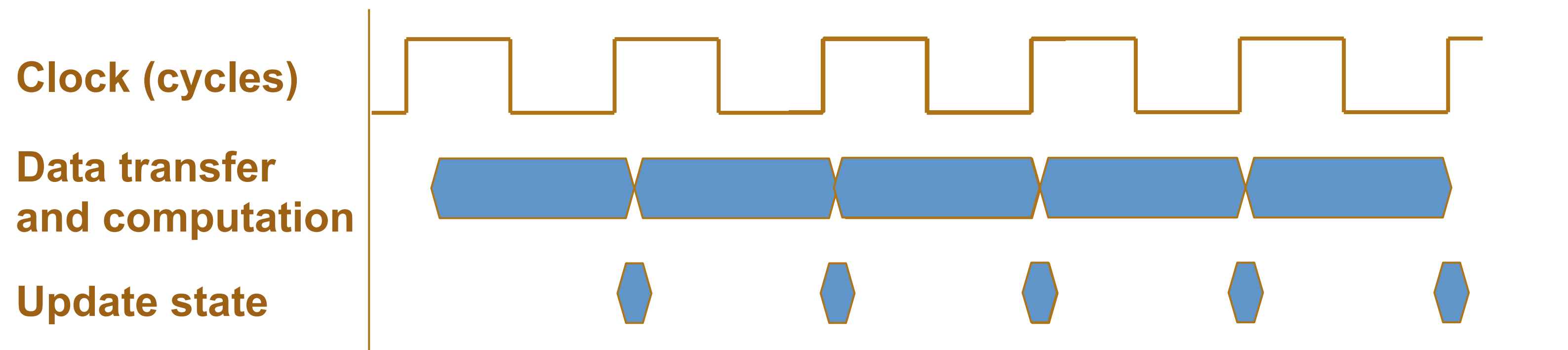
LUNDS UNIVERSITET

Kontrollenhet

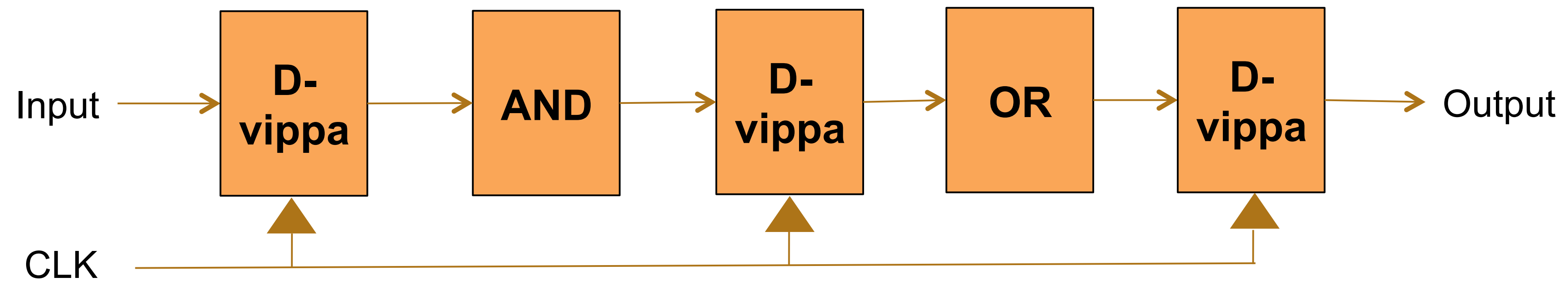


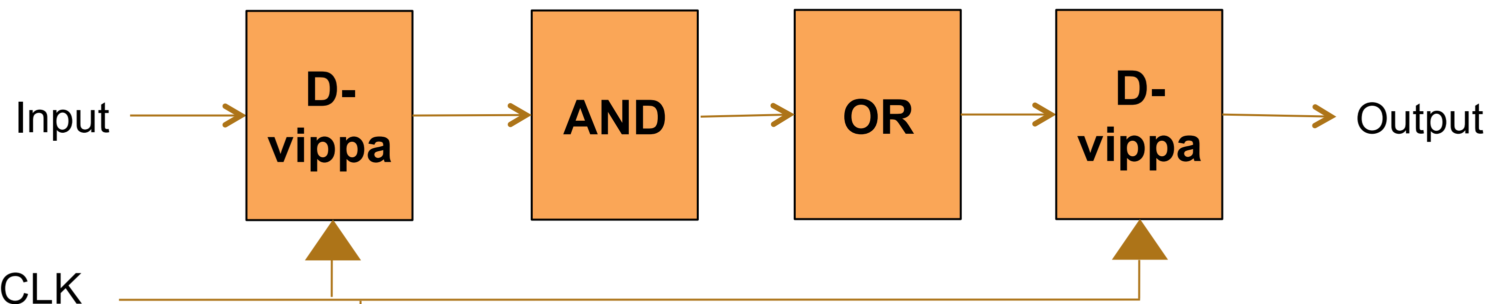
Frekvens och klockperiod





Vilket är snabbast?

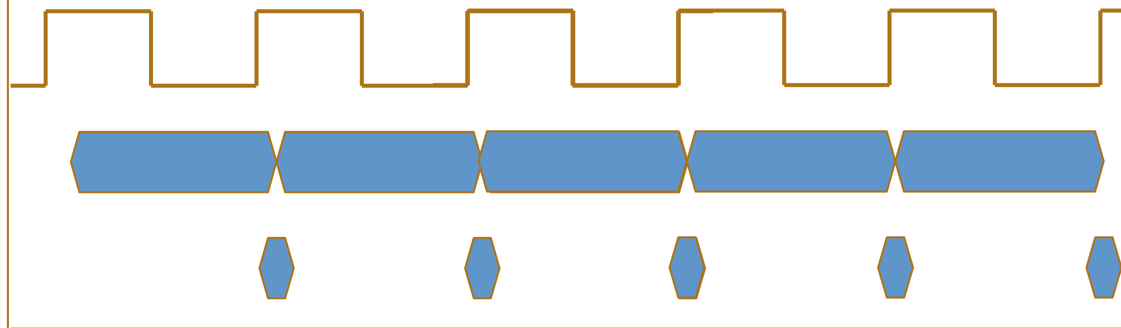




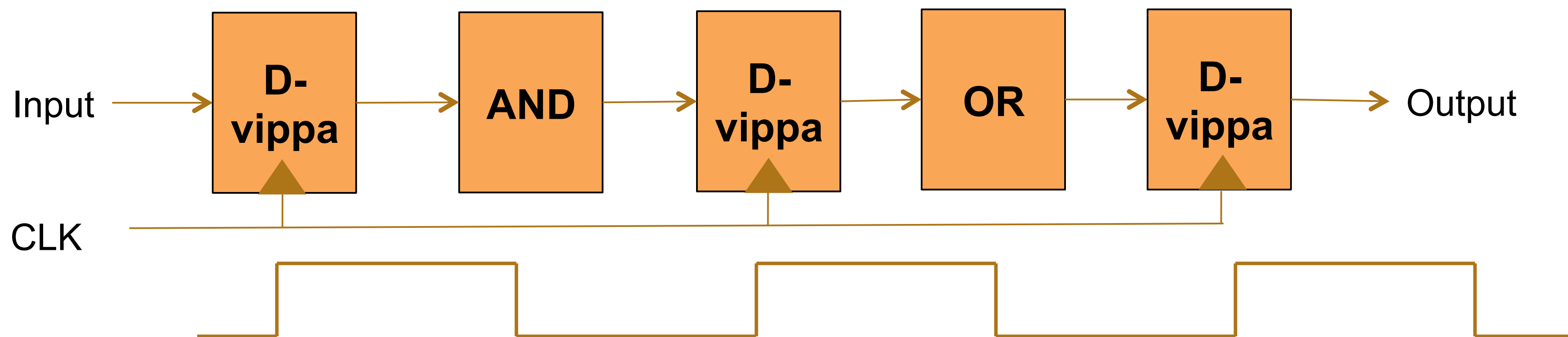
Clock (cycles)

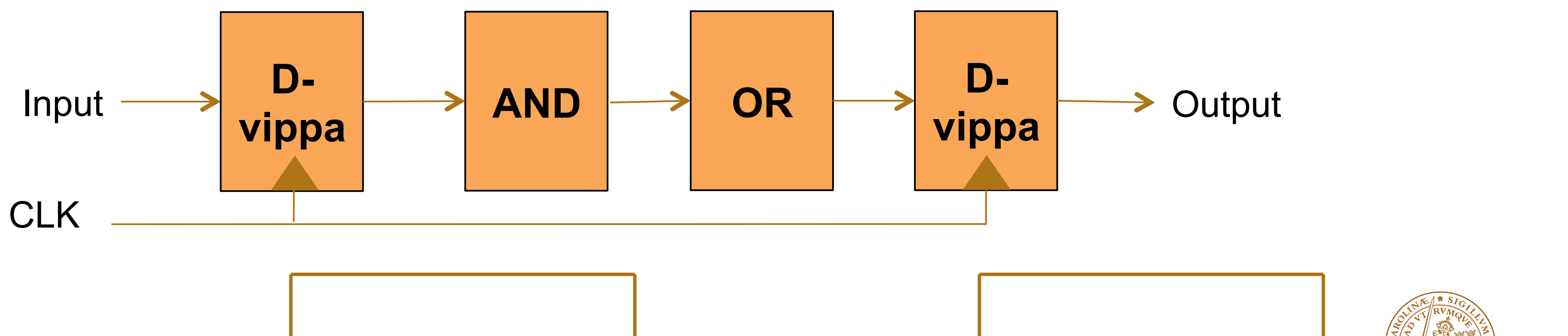
Data transfer and computation

Update state



Vilket är snabbast?





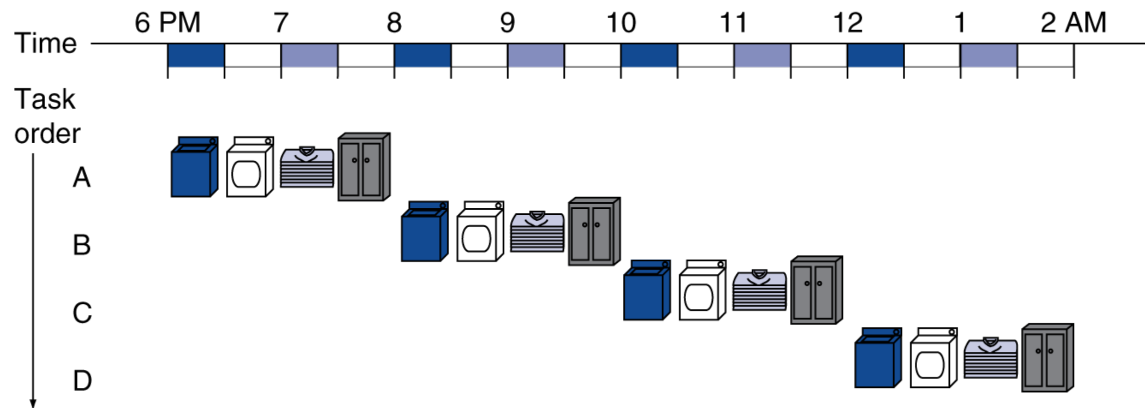
LUNDS UNIVERSITET

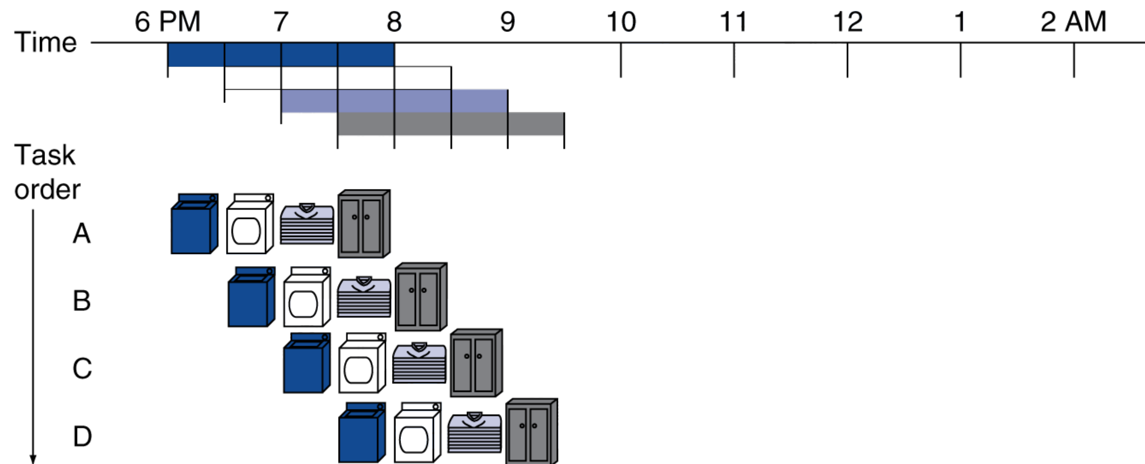
Budskap

- Hög klockfrekvens ger kortare klockperiod.
- Kort klockperiod gör att arbete kan behöva delas upp på flera klockperioder
- Hög klockfrekvens leder alltså inte nödvändigtvis till att arbetet utförs snabbare

LUNDS UNIVERSITET

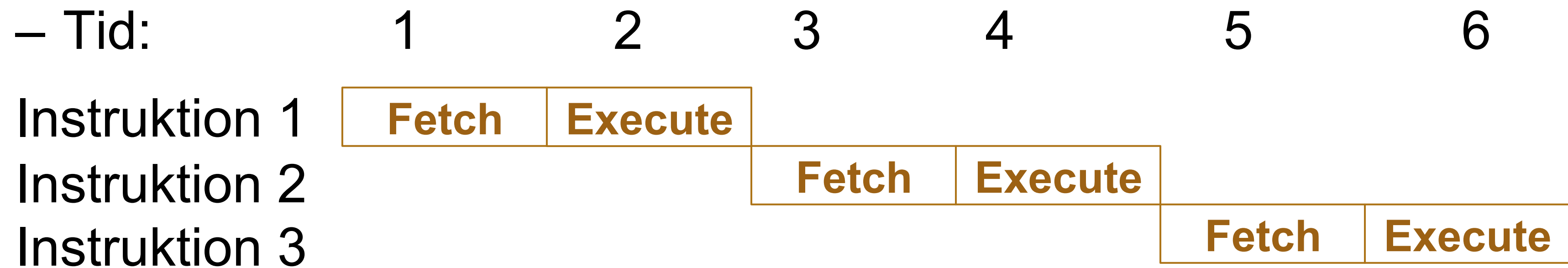
Pipeline - analogi



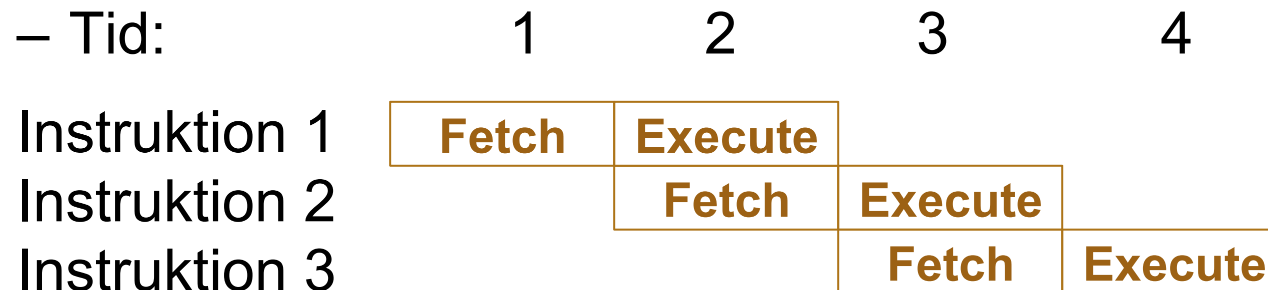


Fetch-Execute

- Utan pipelining:



- Med pipelining:



LUNDS UNIVERSITET

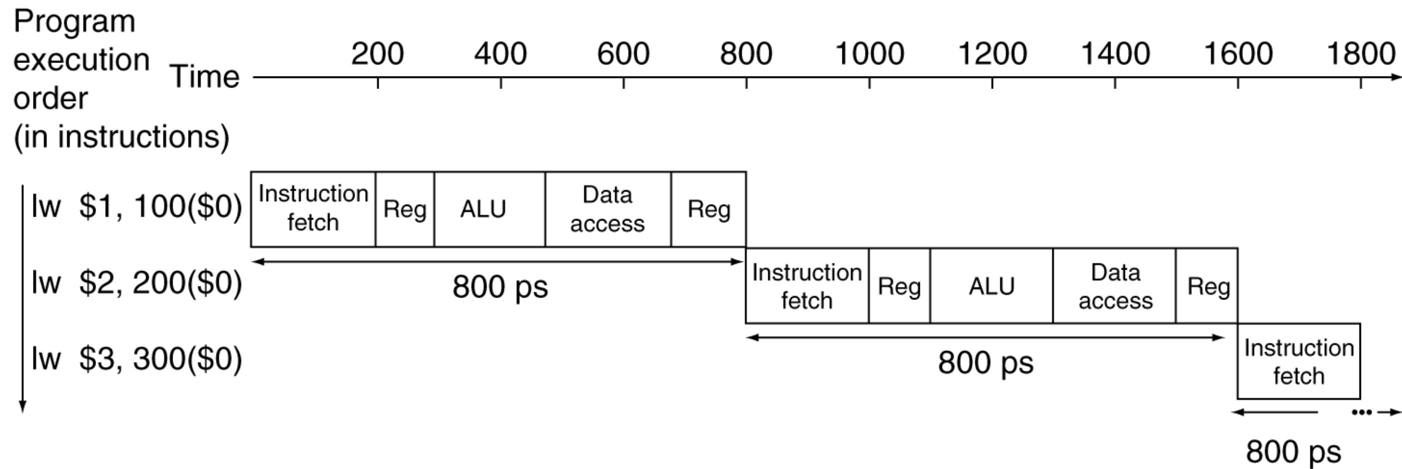
Pipelining

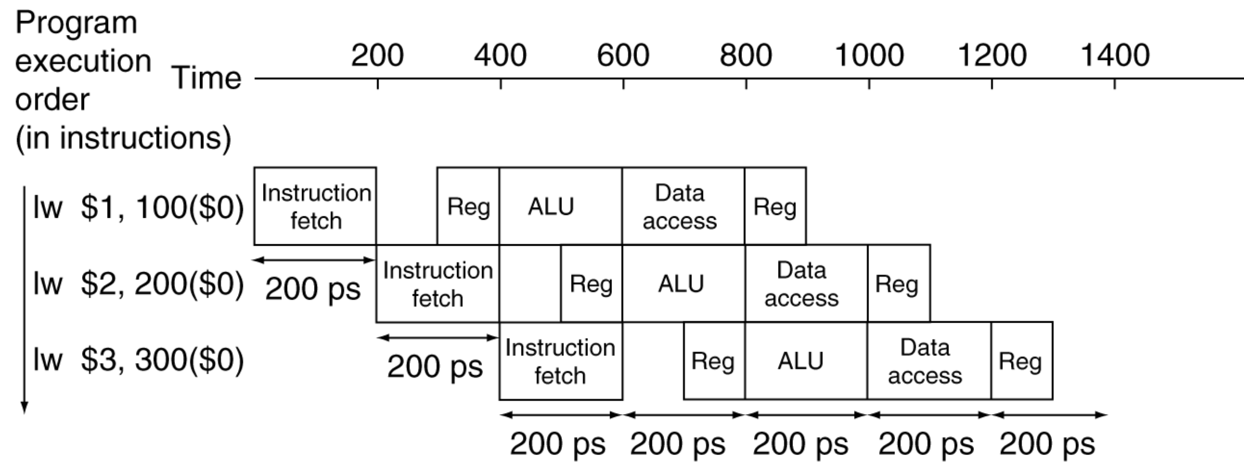
- 6-stegs pipeline:
  - Fetch instruction (FI)
  - Decode instruction (DI)
  - Calculate operand address (CO)
  - Fetch operand (FO)
  - Execute instruction (EI)
  - Write operand (WO)
- Tid:

	1	2	3	4	5	6	7	8
Instruktion 1	FI	DI	CO	FO	EI	WO		
Instruktion 2		FI	DI	CO	FO	EI	WO	
Instruktion 3			FI	DI	CO	FO	EI	WO

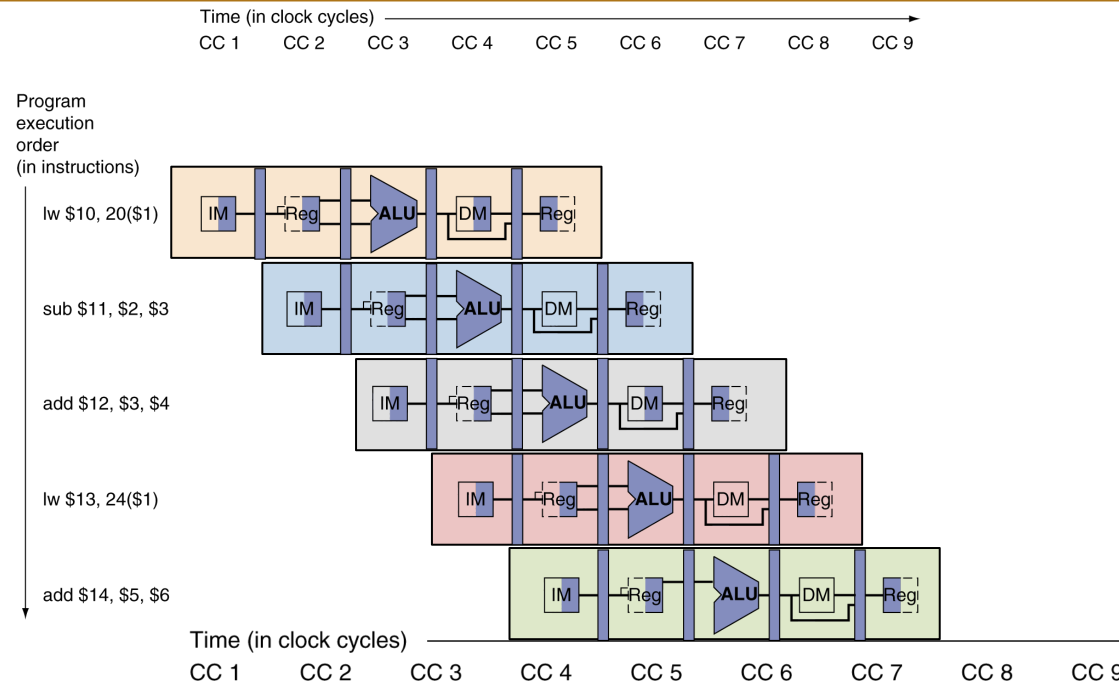
LUNDS UNIVERSITET

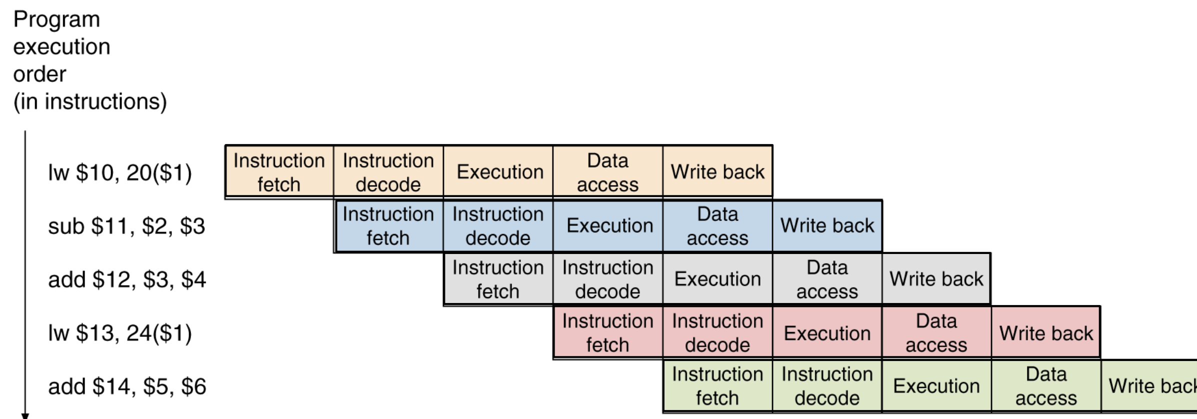
Pipelining



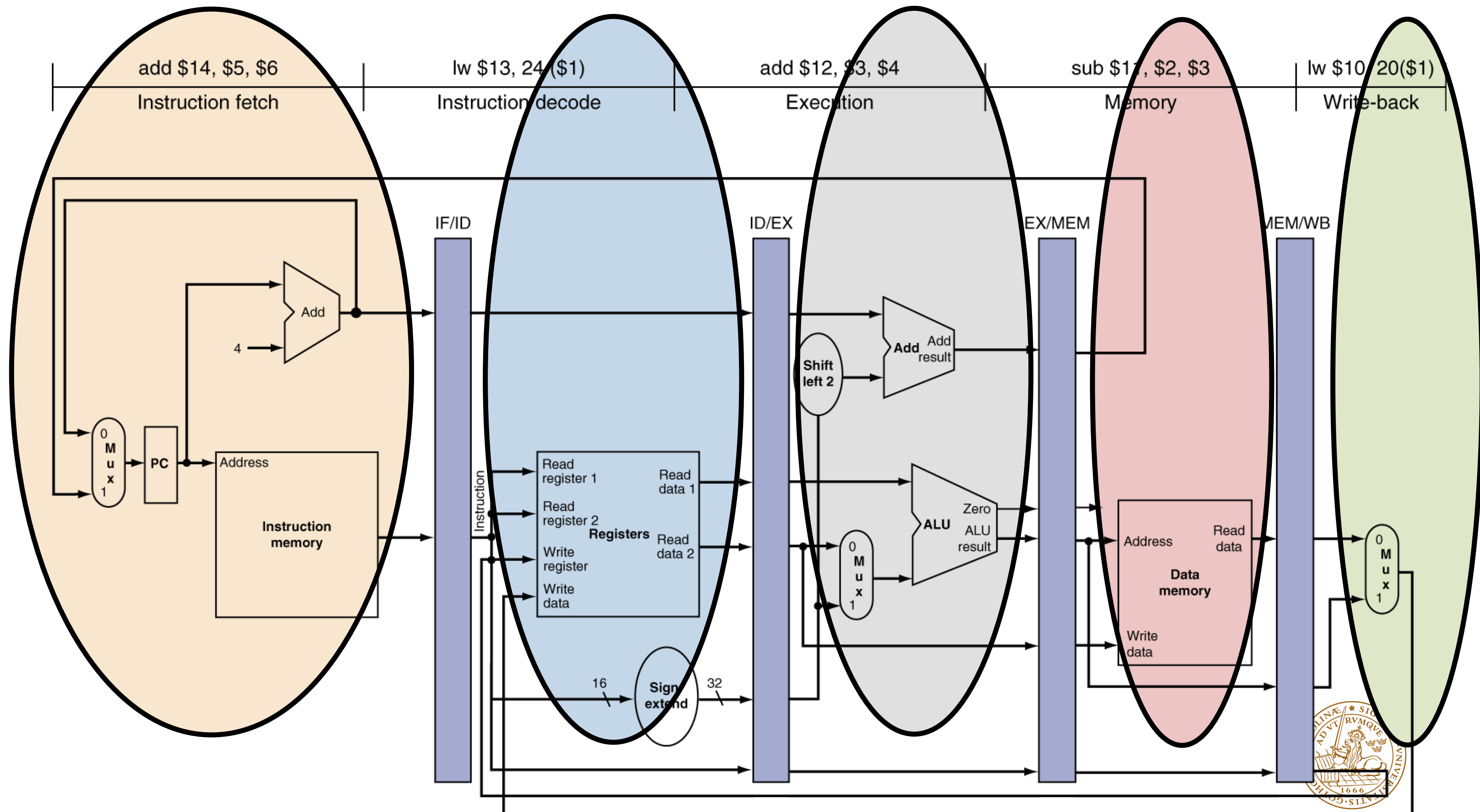


Pipeline diagram





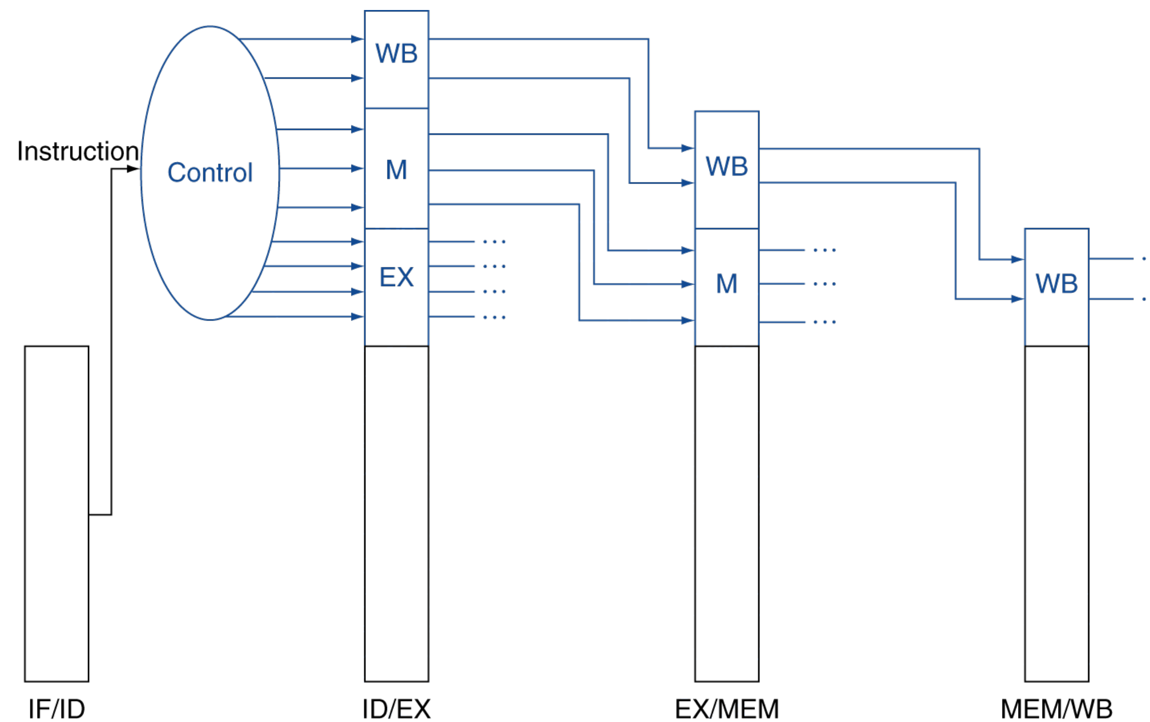
Pipeline diagram (vid en viss tidpunkt)



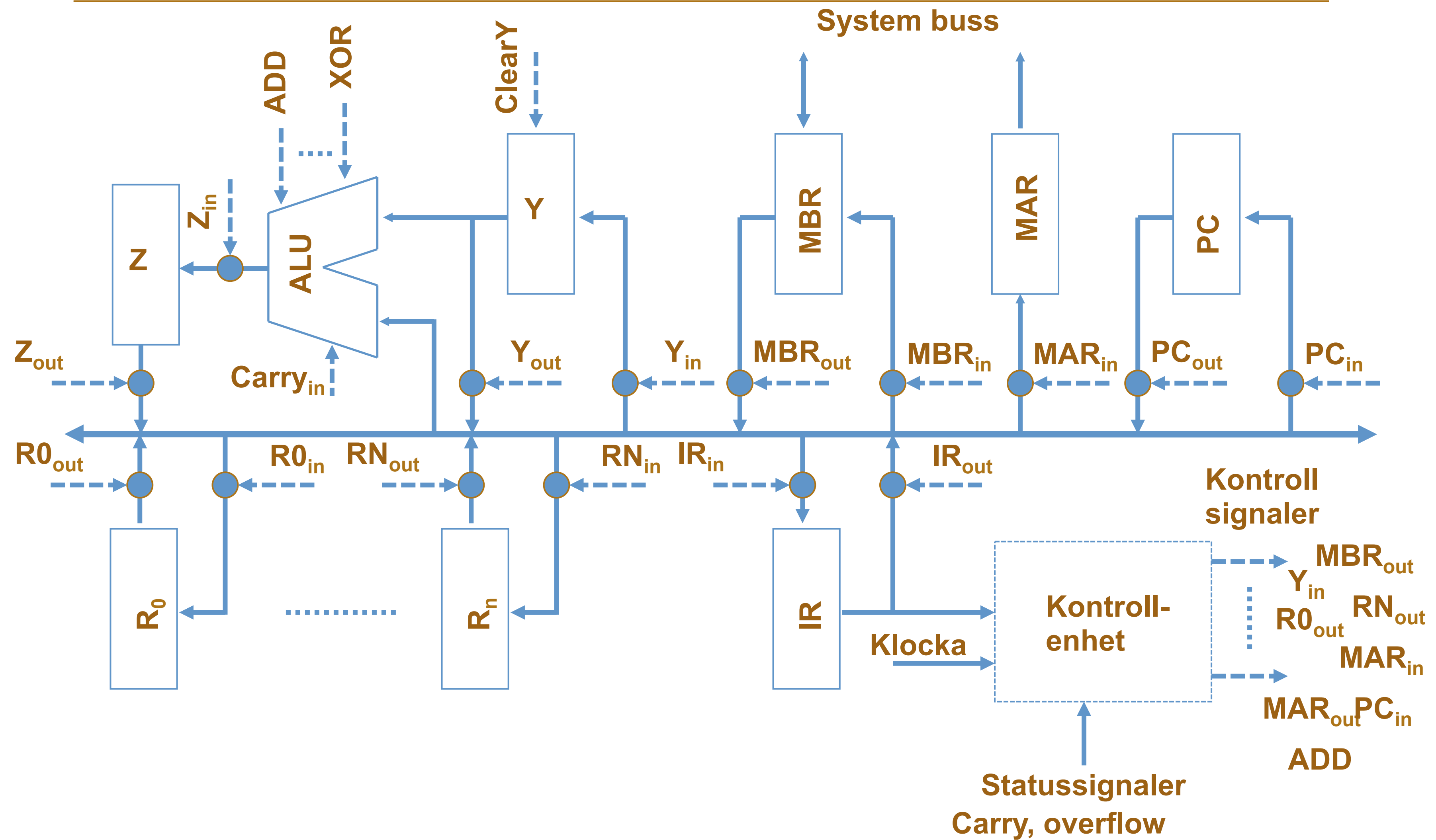
LUNDS UNIVERSITET

Pipeline kontroll

- Kontrollsignaler fås från instruktion



Kontrollenhet utan pipeline



Kontrollenhet utan pipeline

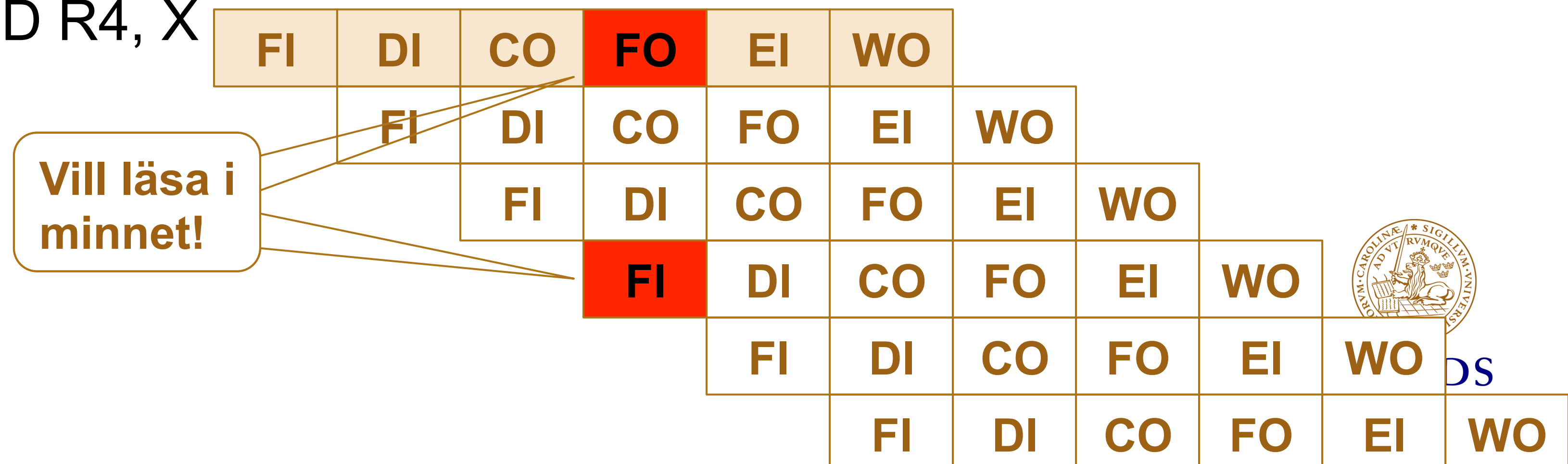
- Instruktion:
  - ADD R1, R3 // R1 <- R1 + R3
- Kontrollsteg:
  1. PC_{out}, MAR_{in}, Read, Clear Y, Carry-in, Add, Z_{in}
  2. Z_{out}, PC_{in}
  3. MBR_{out}, IR_{in}
  4. $R1_{out}$, Y_{in}
  5. $R3_{out}$, Add, Z_{in}
  6. Z_{out}, $R1_{in}$, End

LUNDS UNIVERSITET

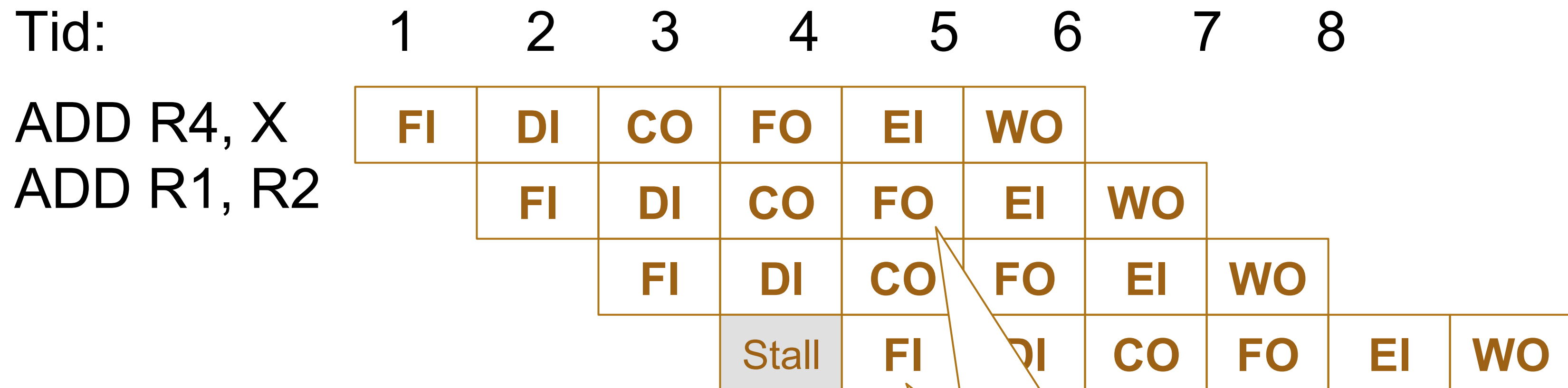
Strukturella hazards

- Resurs (minne, funktionell enhet) behövs av fler än en instruktion samtidigt
- Instruktionen ADD R4, X   R4 <- R4 + X
  - I steget FO hämtas operanden X från minnet. Minnen accepterar en läsning i taget
- Tid: 1 2 3 4 5 6 7 8

ADD R4, X



Strukturella hazards

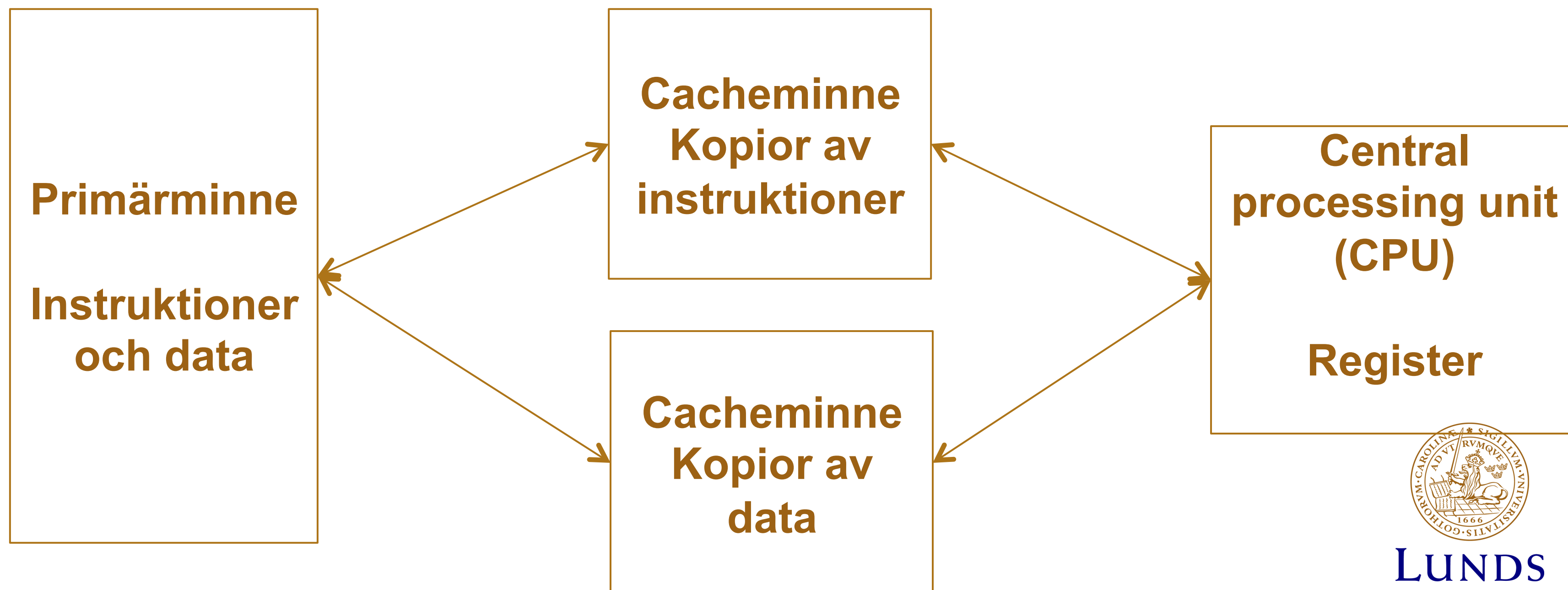


- Penalty: 1 cykel

Men här då? Inget problem då instruktionern (ADD R1, R2) använder register

Strukturella hazards

- För att minska strukturella hazards – öka antalet resurser.
- Ett klassiskt sätt att undvika hazards vid minnesaccess är att ha separat data och instruktions cache



Data hazards

- Antag två instruktioner, I1 och I2. Utan pipeline, så är I1 helt färdig innan I2 startar. Men, så är det inte med pipelines. Det kan bli så att I2 behöver resultat från I1 men I1 är inte klar

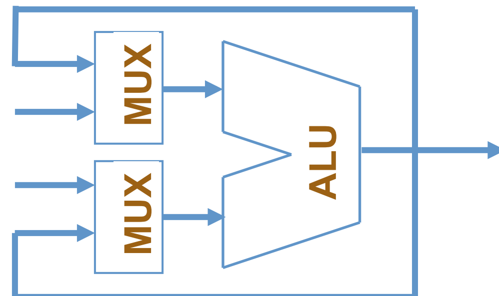
I1: MUL R2, R3 R2 <- R2 * R3
I2: ADD R1, R2 R1 <- R1 + R2

MUL R2, R3
ADD R1, R2

- Penalty: 2 cykler

Data hazards

- För att minska "penalty" kan forwarding/bypassing användas
- Resultat från ALUn kopplas direkt (forward/bypass) till ingångar. Man behöver alltså inte vänta på WO-steget

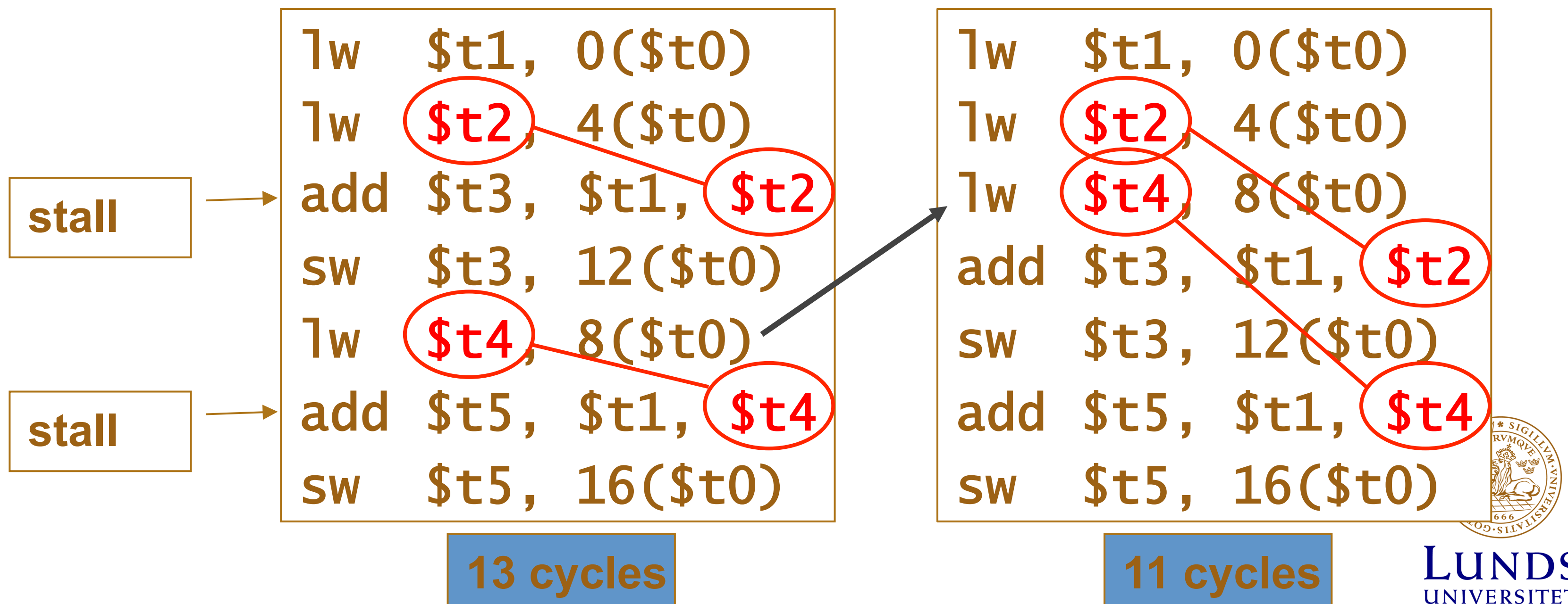


MUL R2, R3	FI	DI	CO	FO	EI	WO		
ADD R1, R2		FI	DI	CO	Stall	FO	EI	WO

LUNDS UNIVERSITET

Data hazards

- Ändra ordning på instruktioner (re-schedule) för att undvika att använda resultat från load i direkt följande instruktion
- C kod: A=B+E;
  C=B+F;



LUNDS UNIVERSITET

Kontroll hazards

- Kontroll hazards uppkommer på grund av hopp (branch) instruktioner
- Ovillkorliga hopp (unconditional branch)

Adress	Instruktion	Kommentar
.....	....	
01011	Instruktion 29	//Instruktion 29, PC=PC+1
01100	BR 10101	// PC = 10101, PC=PC+1
01101	Instruktion 31	//Instruktion 31, PC=PC+1
……	………	
10101:	Instruktion 89	//Instruktion 89, PC=PC+1
……	………	

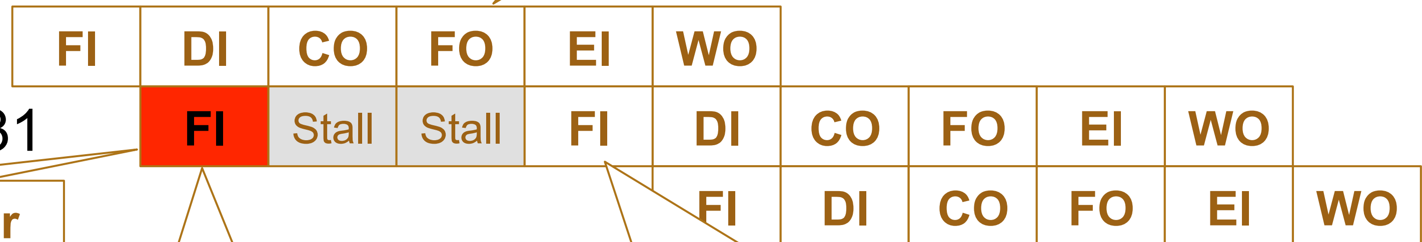
LUNDS UNIVERSITET

Kontroll hazards

01011	Instruktion 29
01100	BR 10101
01101	Instruktion 31
……	………
10101:	Instruktion 89

Efter FO är hoppadressen känd

BR 10101
Instruktion 31



Här vet vi att det är en hoppinstruktion

Instruktion 31 är alltså fel

Här börjar vi med rätt instruktion (instruktion 89)

- Penalty: 3

LUNDS UNIVERSITET

Kontroll hazards

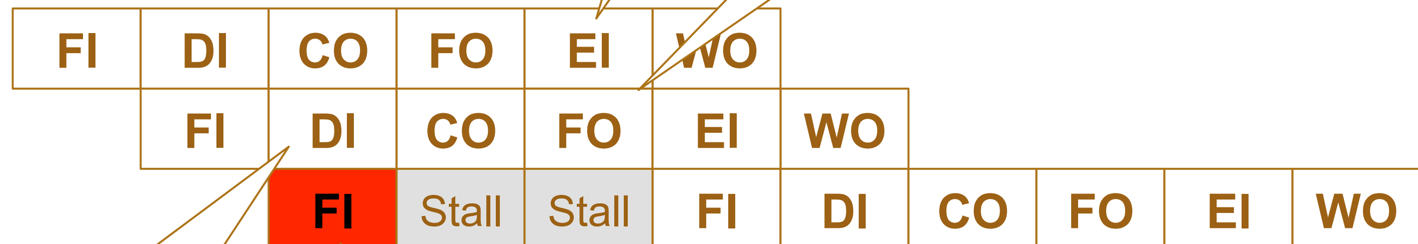
- Villkorliga hopp (conditional branch)

  ADD R1, R2 //R1 <- R1 + R2
  BEZ Target //Branch if zero
  instruktion 31
  ……

Target: instruktion 89

- Alternativ: Hoppet görs (Branch taken)

  ADD R1, R2
  BEZ Target
  instruktion 31



- Penalty: 3 cykler

Här görs additionen som påverkar Z-flaggan (om hoppet ska göras eller inte.

Här är hopp-adressen känd

Här vet vi att det är en hoppinstruktion

Instruktion 31 kan vara fel

Kan hämta instruktion 89

LUNDS UNIVERSITET

Kontroll hazards

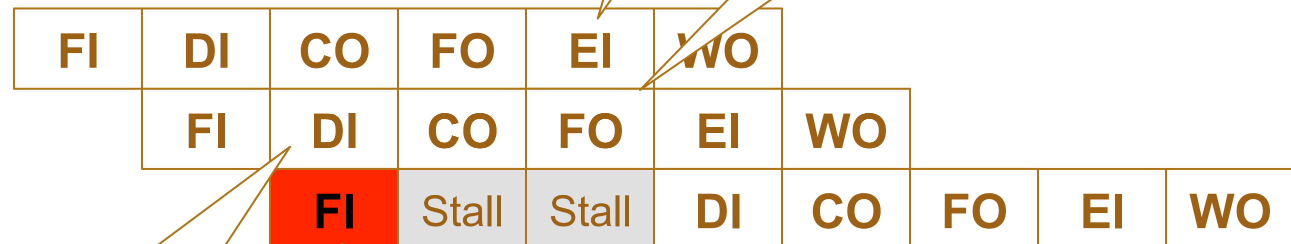
- Villkorliga hopp (conditional branch)

  ADD R1, R2 //R1 <- R1 + R2
  BEZ Target //Branch if zero
  instruktion 31
  ……

Target: instruktion 89

- Alternativ: Hoppet görs inte (Branch not taken)

  ADD R1, R2
  BEZ Target
  instruktion 31



- Penalty: 2 cykler

Här görs additionen som påverkar Z-flaggan (om hoppet ska göras eller inte.

Här är hopp-adressen känd

Här vet vi att det är en hoppinstruktion

Instruktion 31 kan vara rätt

Instruktion 31 är rätt…..

Delayed branching

- Vid villkorliga hopp undersökte vi fallen: hopp görs och görs inte (se tidigare exempel)
- Alternativ: Hoppet görs (Branch taken)

ADD R1, R2	FI	DI	CO	FO	EI	WO				
BEZ Target		FI	DI	CO	FO	EI	WO			
instruktion 31			FI	Stall	Stall	FI	DI	CO	FO	EI	WO

Penalty: 3 cykler

- Alternativ: Hoppet görs inte (Branch not taken)

ADD R1, R2	FI	DI	CO	FO	EI	WO				
BEZ Target		FI	DI	CO	FO	EI	WO			
instruktion 31			FI	Stall	Stall	DI	CO	FO	EI	WO	

Penalty: 2 cykler

LUNDS UNIVERSITET

Delayed branching

- Detta har programmeraren skrivit:

```
MUL R3, R4        R3<-R3*R4
SUB #1, R2        R2<-R2-1
ADD R1, R2        R1<-R1+R2
BEZ TAR           Branch om zero
MOVE #10, R1      R1<-10
```

Den här instruktionen påverkar inte det som händer fram till hoppet

Den här instruktionen exekveras bara om hoppet inte tas

- Detta är det som kompilatorn (assemblatorn) genererar:

```
SUB #1, R2        R2<-R2-1
ADD R1, R2        R1<-R1+R2
BEZ TAR           Branch om zero
MUL R3, R4        R3<-R3*R4
MOVE #10, R1      R1<-10
```

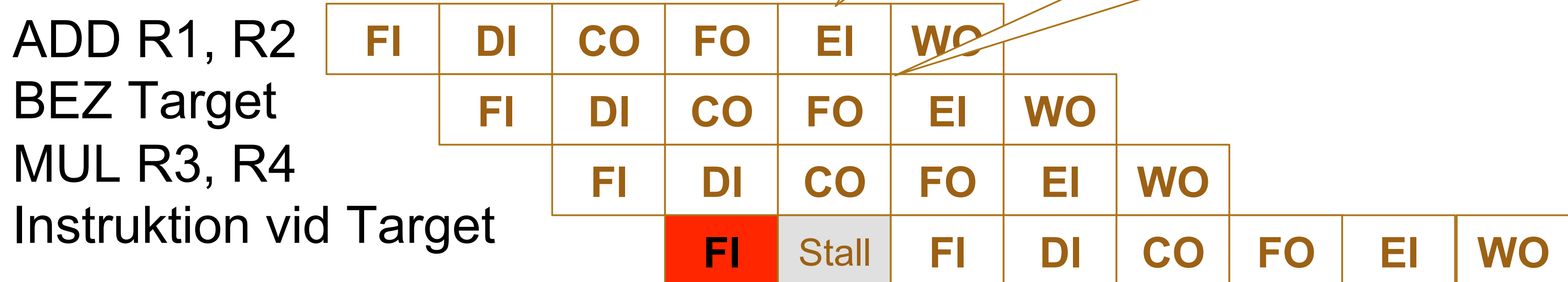
Den här instruktionen kommer exekveras oavsett om hoppet tas eller ej

Den här instruktionen kommer exekveras bara om hoppet inte tas

Delayed branching

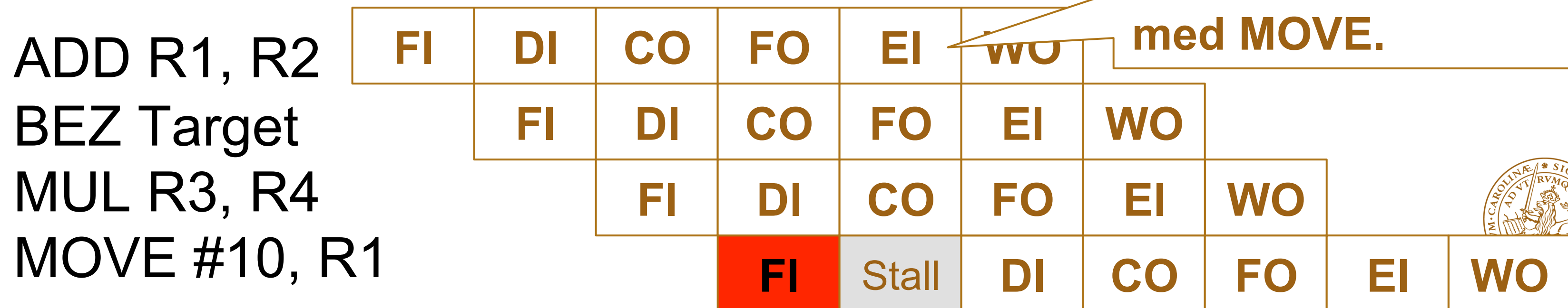
- Alternativ: Hoppet görs (Branch taken)

ADD R1, R2
BEZ Target
MUL R3, R4
Instruktion vid Target



- Penalty 2 cykler (innan 3)
- Alternativ: Hoppet görs inte (Branch not taken)

ADD R1, R2
BEZ Target
MUL R3, R4
MOVE #10, R1



- Penalty: 1 cykler (innan 2)

LUNDS UNIVERSITET

Sammanfattning

- Instruktioner exekveras som en sekvens av steg; t ex Fetch instruction (FI), Decode instruction (DI), Calculate operand address (CO), Fetch operand (FO), Execute instruction (EI) och Write operand (WO)
- En pipeline består av N steg. Vid ett visst ögonblick kan N instruktioner vara aktiva. Om 6-steg används (FI, DI, CO, FO, EI, WO) kan 6 instruktioner vara aktiva
- Att hålla piplinen full med instruktioner är svårt pga pipeline hazards.
  - Strukturella hazards beror på resurskonflikter.
  - Data hazards beror på detaberoende mellan instruktioner
  - Control hazards beror på hoppinstruktioner

LUNDS UNIVERSITET

LUNDS
UNIVERSITET