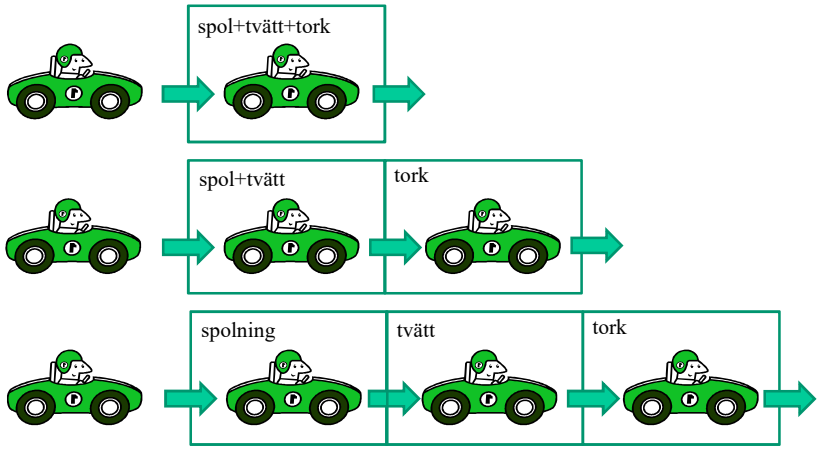
4. Pipelining

2

4. Pipelining

Det finns en pipelinad biltvätt i Linköping …



- De tre momenten tar lika lång tid
- Alla bilar går igenom samma program

→ Väntetid 1/3
Genomströmning 3

OBS, vi får inte uppsnabbningen gratis.
En del utrustning måste finnas på flera ställen!

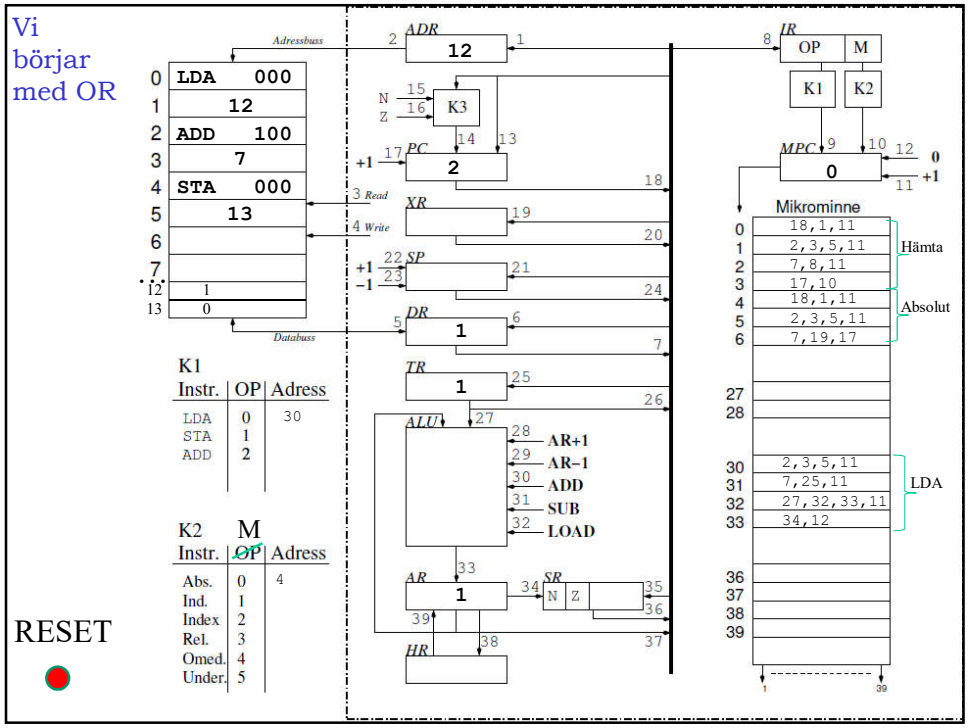
3

4. Pipelining

Alternativa arkitekturer eller riktiga datorer

- kritik av OR-datorn
  - - för många klockcykler,
    går att dock att snabba upp
  - + går att göra komplicerade instruktioner:
    sortering, matrisinvertering, …
- RISC = reduced instruction set computer
  enkla,lika instruktioner => pipelining möjligt

4

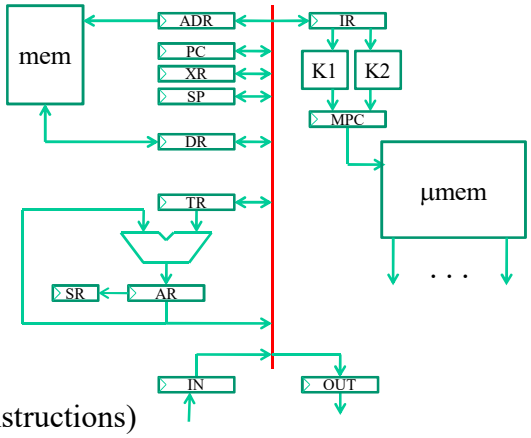


5

OR-datorn är för långsam!

- LDA 12 (exempelvis)
  - Hämta : 3 CP
  - K2 1 CP
  - Absolut: 3 CP
  - EXE: 4 CP

  Summa: 11 CP



Alltså 11 CPI (clocks per instructions)
Vi siktar på 1 CPI!

6

Vad kan göras på OR-original?

- Parallellism
  - Tryck ihop mikrokoden, dvs gör flera µop samtidigt
- Förhämtning:
  - hämta nästa instruktion under exekvering av pågående instruktion

```
adr->M->dr
dr->tr
tr->ar
status
```

Exe av LDA

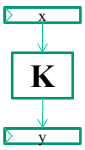
```
pc++->adr
adr->M->dr
dr->ir
```

Hämta nästa instruktion

```
mpc++
mpc++
mpc++
K2->mpc
```

7

Apropos klockfrekvensen



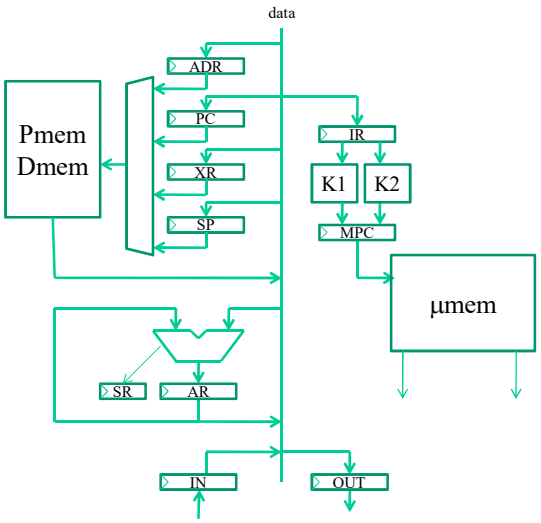
Leta rätt på längsta tidsfördröjningen mellan två register. Kalla den T. Då gäller $f < 1/T$

Brukar kallas kritisk väg

8

Steg 1

+ Omplacering av register
+ Ta bort onödiga register
+ Förbättrat SR
- Lång kritisk väg



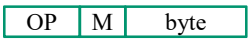
Nya LDA 12

– Hämta	1 CP
– K2	1 CP
– Absolut	1 CP
– EXE	1 CP

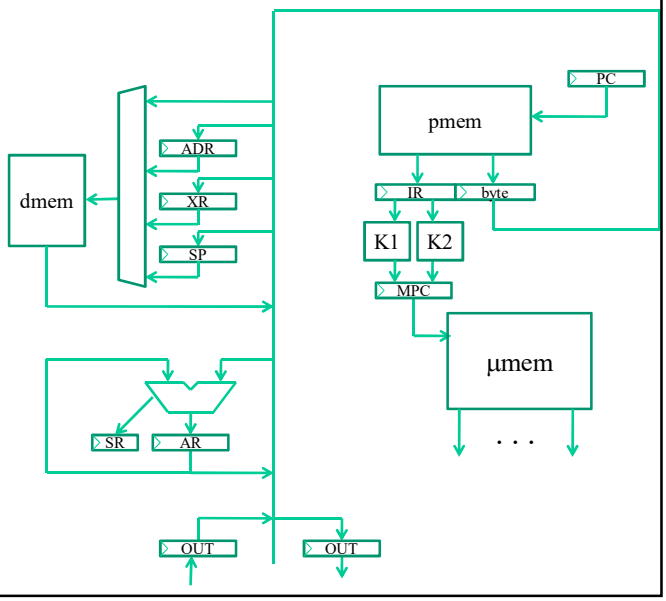
9

Steg 2

• Skilda program- och dataminnen
• Bredare programminne, så att hela instruktionen kan hämtas på 1 CP

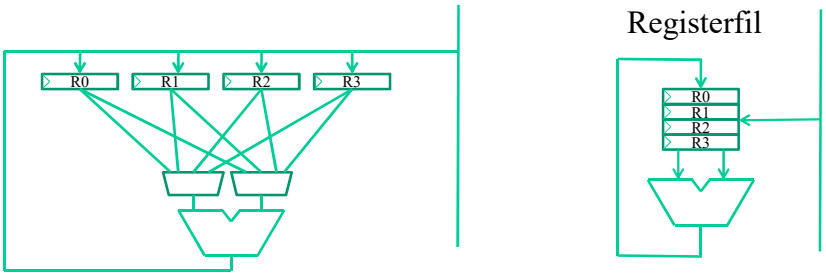


- Nya LDA 12
  - Hämta 1 CP
  - K2 1 CP
  - Absolut
  - EXE 1 CP



10

Flera register!



Exempelvis:
ADD R3,R2,R1 ; R3 = R2+R1

11

RISC = reduced instruction set computer

- Vi avskaffar XR,SP och inför generella register
- Vi avskaffar flera a-moder per instruktion
  - ADD Rx,Ry,Rz inga andra adr-moder!
  - LD Rx,(Ry) enda sättet att läsa i minnet!
  - ST (Rx),Ry enda sättet att skriva i minnet!
- Vi avskaffar MPC och mikroprogrammering
  - Mikroprogrammen försvinner inte utan finns på annan form i maskinen

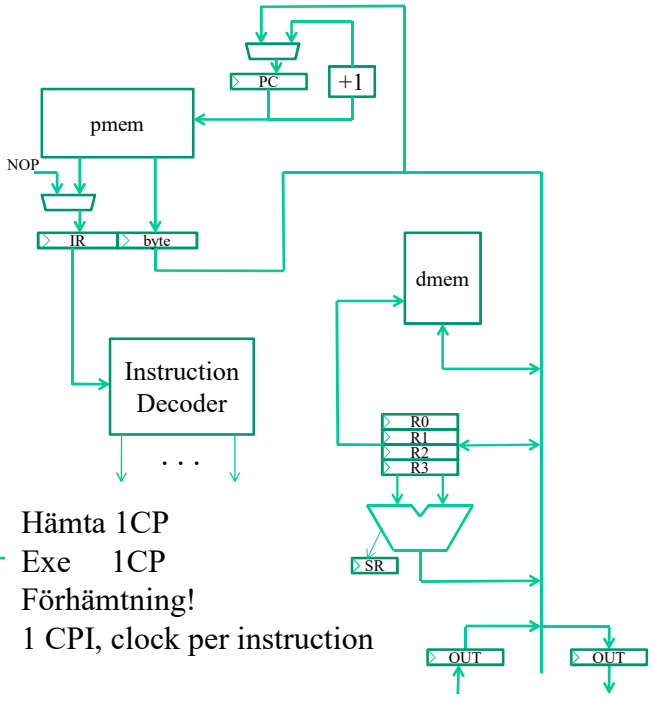
12

Steg 3

Hämtfasen har blivit HW

Instruction Decoder innehåller 1-rads µprog

- ADD Rx,Ry,Rz
- LD Rx,(Ry)
- ST (Ry),Rx
- JMP N
- ADDI Rx,Ry,K

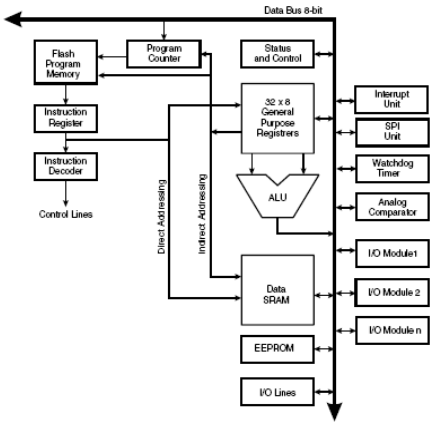


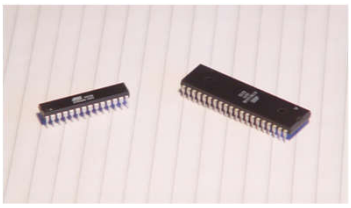
Hämta 1CP
Exe 1CP
Förhämtning!
1 CPI, clock per instruction

13

ATMEL AVR

- Trevlig 8-bitars controller
- C-kompilator finns: avr-gcc
- Massor med kul I/O





14

Pipelinediagram

```
0: LD  R1,(R0)   ; R1:=DM(R0)
1: ADD R3,R2,R1  ; R3:=R1+R2
2: JMP 0         ;
3: XXX
```

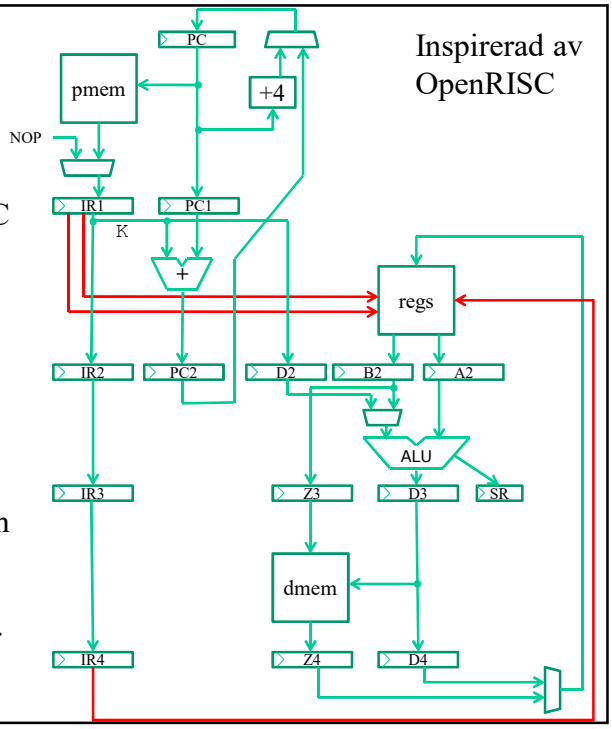
PC	0	1	2	3	0	1
IR		LD	ADD	JMP	NOP	LD
regs			LD	ADD	-	NOP

1 inst/CK
JMP ger en NOP i pipelinen

15

Klassisk 5-stegs pipeline

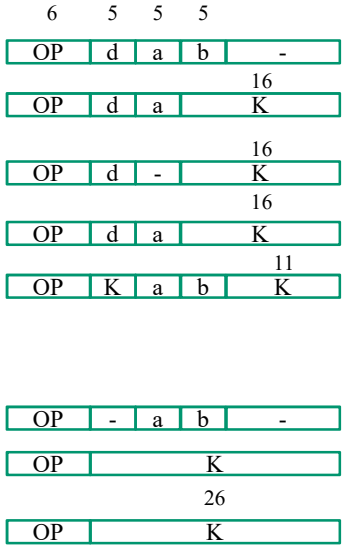
- **IF**: instruction fetch
  hämta instr och ny PC
- **RR**: register read
  läs reg/beräkna hopp
- **EXE**: execute
  kör ALU
- **MEM**: read/write dmem
  läs/skriv/ingenting
- **WB**: write back register
  skriv reg/ingenting



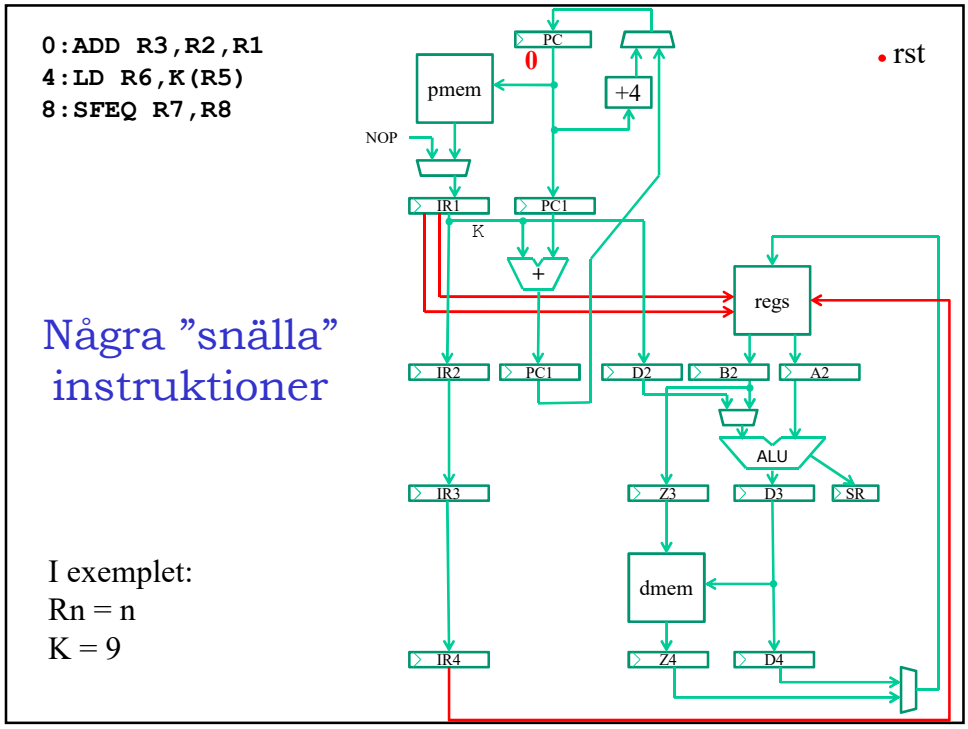
16

Några instruktioner, alla 32b

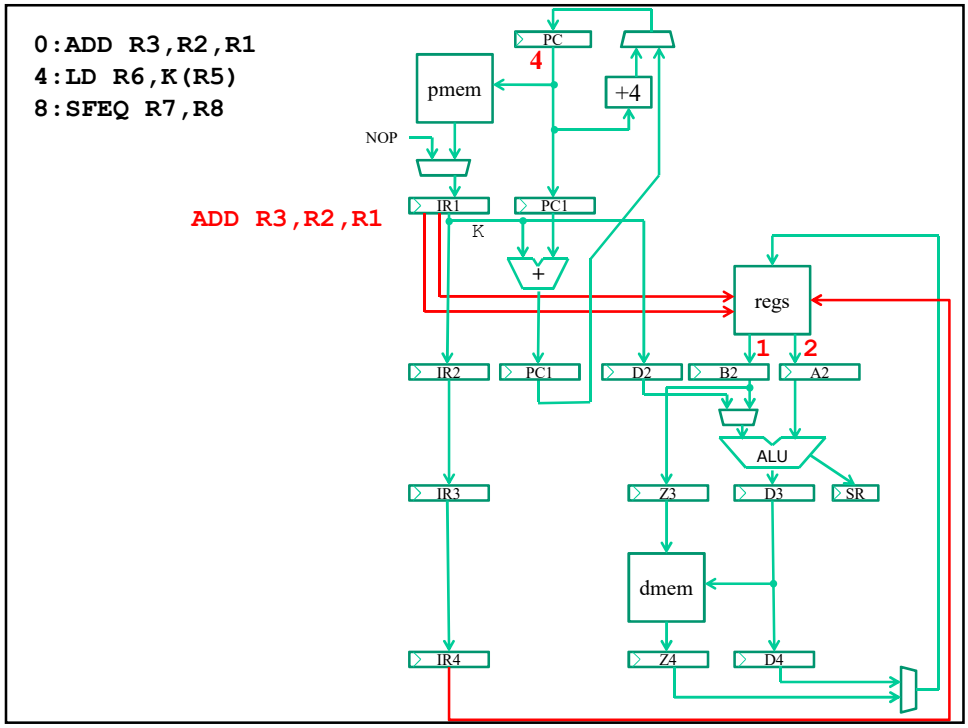
```
ADD  Rd, Ra, Rb    ; Rd=Ra+Rb
ADDI Rd, Ra, K     ; Rd=Ra+K
MOVHI Rd,K         ; RdH=K,RdL=0
LD   Rd, K(Ra)     ; Rd=dmem(Ra+K)
ST   K(Ra), Rb     ; dmem(Ra+K)=Rb

SFEQ  Ra, Rb ; F = (A==B)?1:0
JMP  K             ; PC = PC+K
BF   K     ; PC = F ? PC+K : PC+4
```



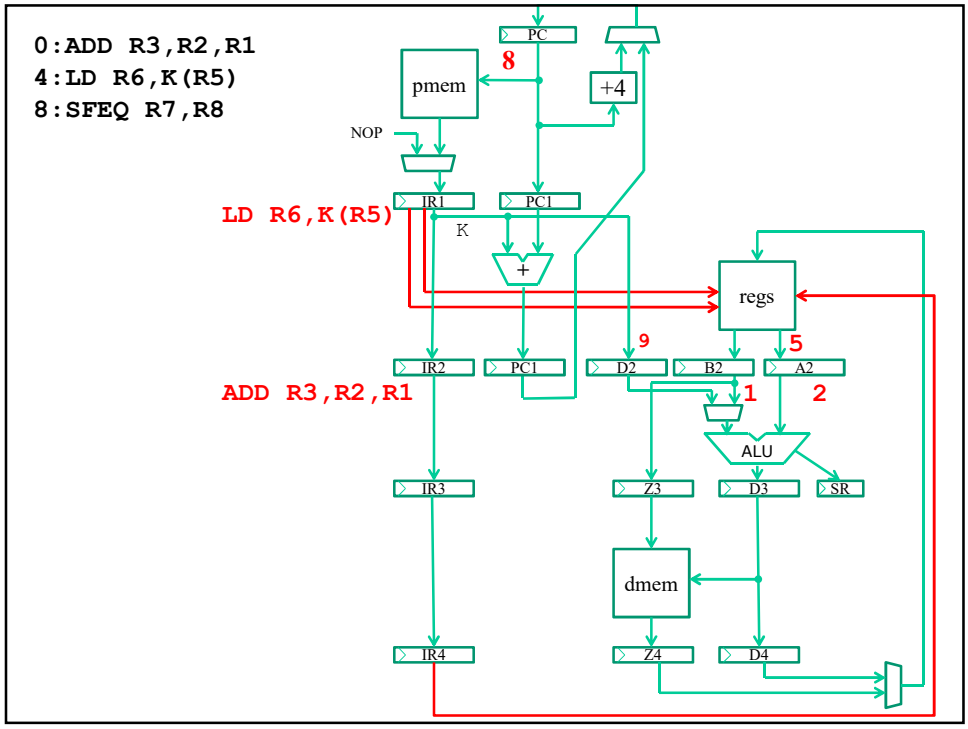
17


0:ADD R3,R2,R1
4:LD R6,K(R5)
8:SFEQ R7,R8
Några ”snälla” instruktioner
I exemplet:
Rn = n
K = 9
rst
PC
0
pmem
+4
NOP
IR1
PC1
K
+
regs
IR2
PC1
D2
B2
A2
ALU
IR3
Z3
D3
SR
dmem
IR4
Z4
D4

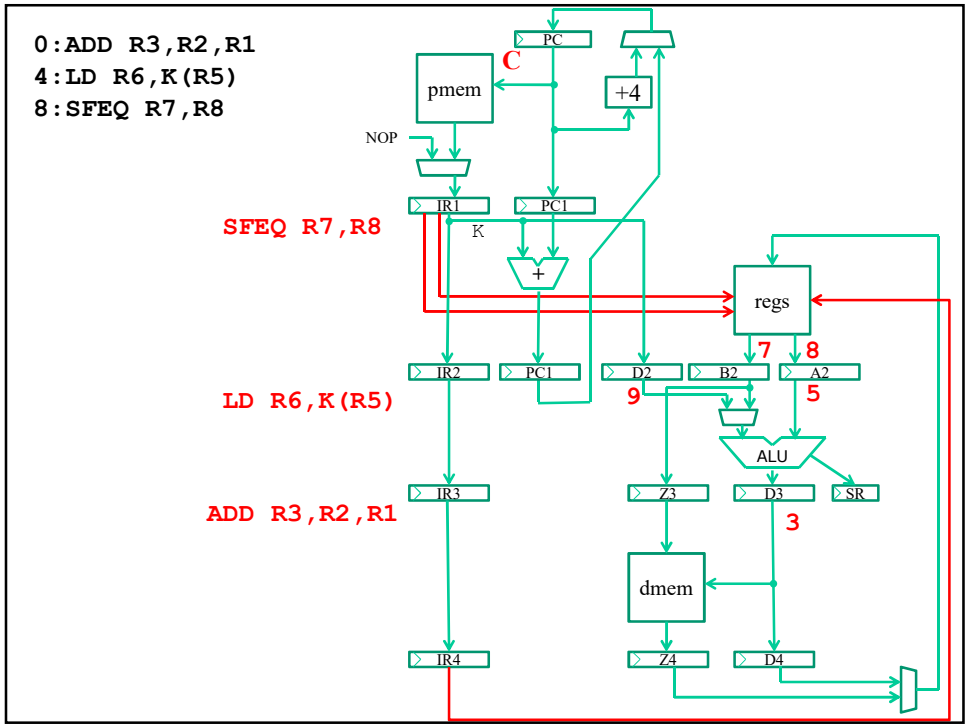
18


0:ADD R3,R2,R1
4:LD R6,K(R5)
8:SFEQ R7,R8
ADD R3,R2,R1
PC
4
pmem
+4
NOP
IR1
PC1
K
+
regs
1
2
IR2
PC1
D2
B2
A2
ALU
IR3
Z3
D3
SR
dmem
IR4
Z4
D4

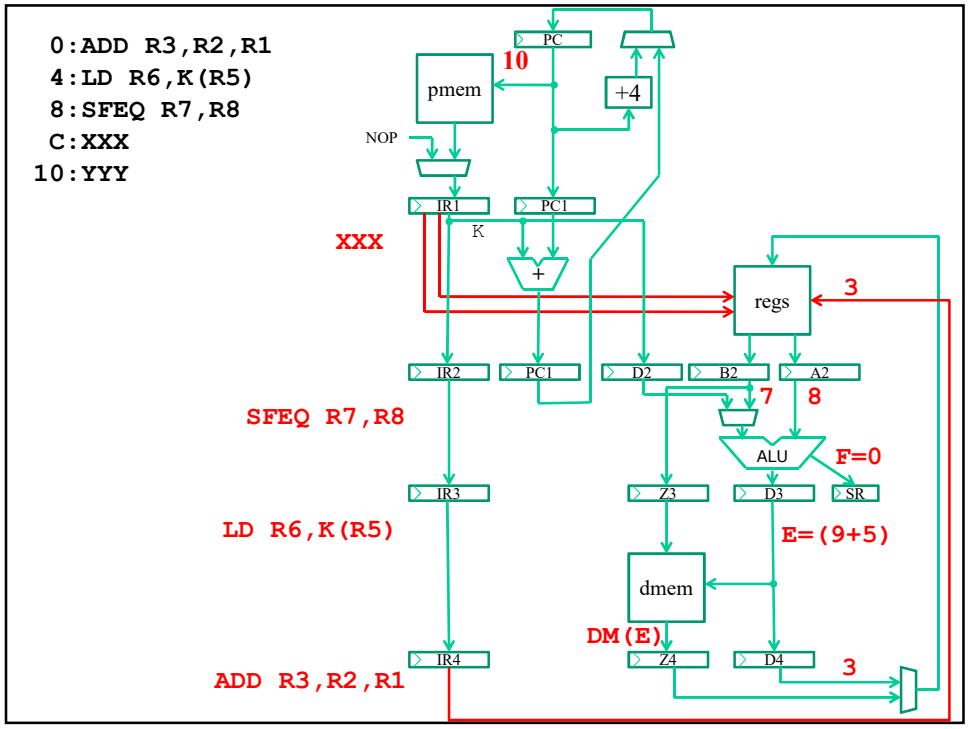
19


0:ADD R3,R2,R1
4:LD R6,K(R5)
8:SFEQ R7,R8
LD R6,K(R5)
ADD R3,R2,R1
PC
8
pmem
+4
NOP
IR1
PC1
K
+
regs
9
5
IR2
PC1
D2
B2
A2
1
2
ALU
IR3
Z3
D3
SR
dmem
IR4
Z4
D4

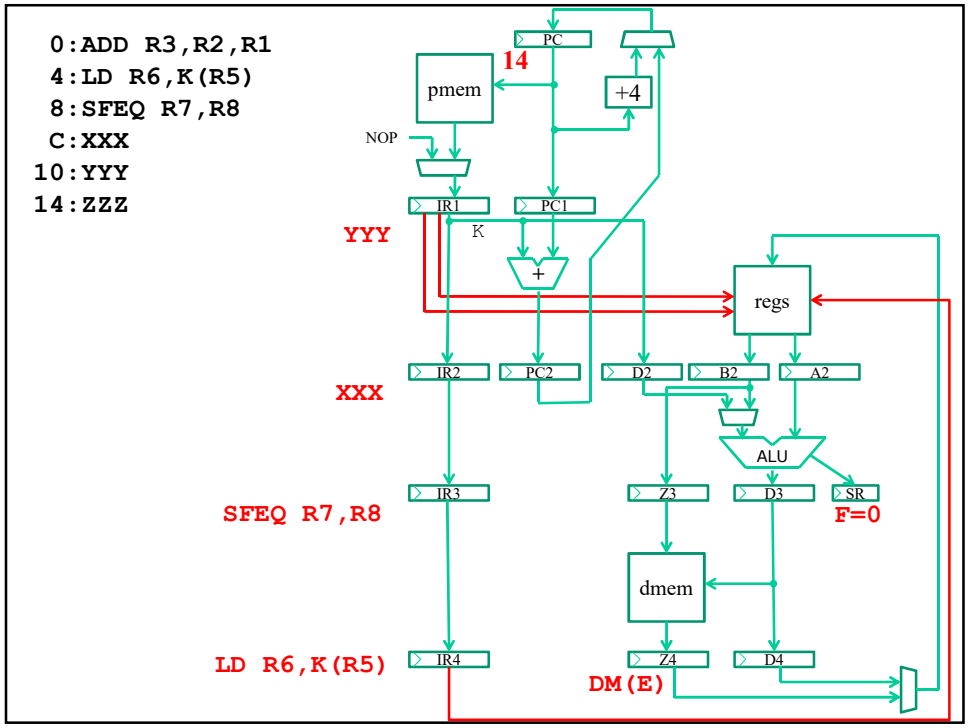
20


0:ADD R3,R2,R1
4:LD R6,K(R5)
8:SFEQ R7,R8
SFEQ R7,R8
LD R6,K(R5)
ADD R3,R2,R1
PC
C
pmem
+4
NOP
IR1
PC1
K
+
regs
7
8
IR2
PC1
D2
B2
A2
9
5
ALU
IR3
Z3
D3
SR
3
dmem
IR4
Z4
D4

21


0:ADD R3,R2,R1
4:LD R6,K(R5)
8:SFEQ R7,R8
C:XXX
10:YYY
PC
10
pmem
+4
NOP
IR1
PC1
K
XXX
+
regs
3
IR2
PC1
D2
B2
A2
7
8
SFEQ R7,R8
ALU
F=0
IR3
Z3
D3
SR
LD R6,K(R5)
E=(9+5)
dmem
DM(E)
IR4
Z4
D4
3
ADD R3,R2,R1

22


0:ADD R3,R2,R1
4:LD R6,K(R5)
8:SFEQ R7,R8
C:XXX
10:YYY
14:ZZZ
PC
14
pmem
+4
NOP
IR1
PC1
K
YYY
+
regs
IR2
PC2
D2
B2
A2
XXX
ALU
IR3
Z3
D3
SR
F=0
SFEQ R7,R8
dmem
IR4
Z4
D4
LD R6,K(R5)
DM(E)

23

Problem

1. **Hopp,** Oönskade instr kommer ibland in i pipelinen
   - >Instruktionen efter ett hopp exekveras alltid
   - >Ytterligare en instr. därefter ersätts med NOP
2. **Databeroenden,** Reg läses i steg 2 o skrivs i steg 5
   - **>data forwarding**
3. Pipelinen måste i vissa lägen stängas av, **stall**

24

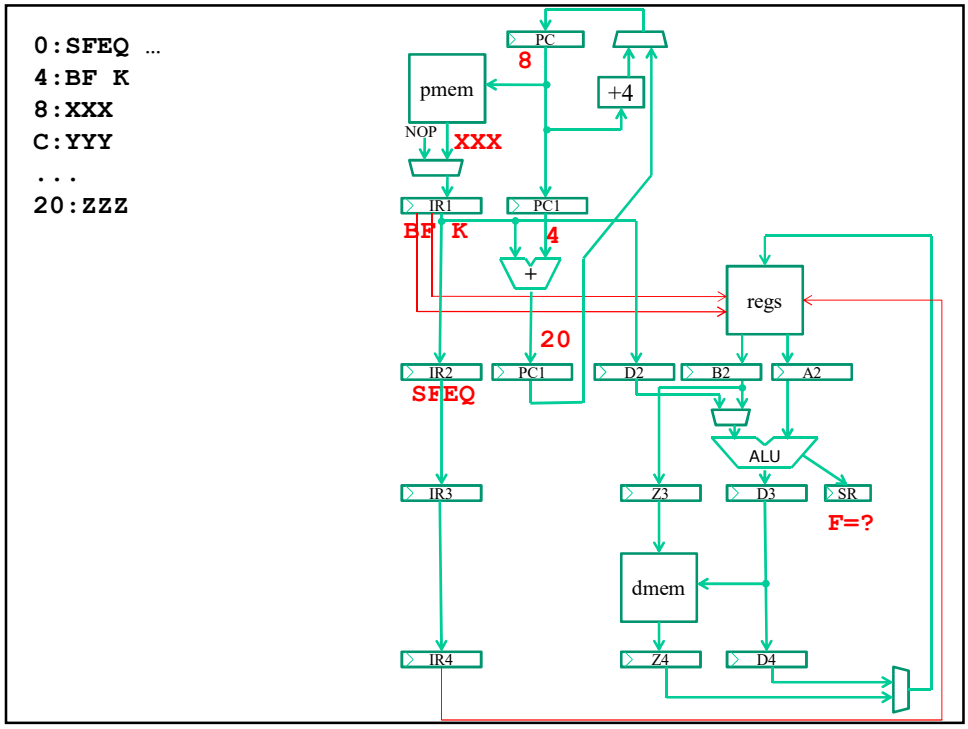
Problem 1: Hopp

```
0:SFEQ …  ; sätter flaggan F
4:BF K    ; hoppar eventuellt till 20
8:XXX     ; instr. exekveras alltid
C:YYY
...
20:ZZZ
```

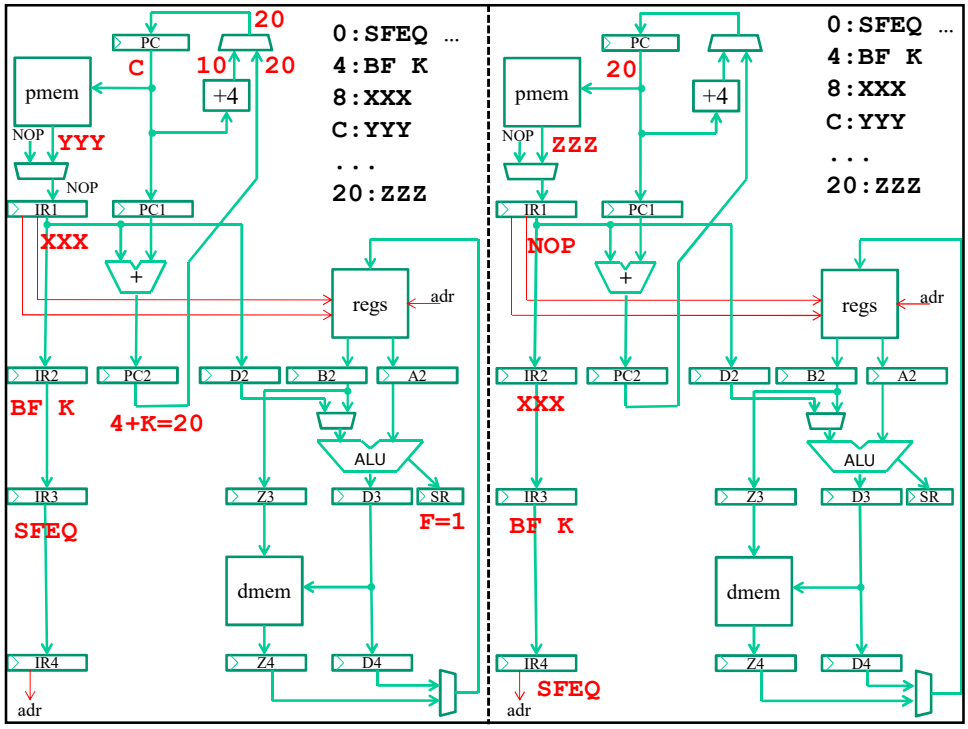
Vi bestämmer:
Instruktionen
XXX
exekveras alltid

Fördröjt hopp: XXX kan vara
- en nyttig instr flyttad hit av kompilatorn
- (software) NOP

25


0:SFEQ ...
4:BF K
8:XXX
C:YYY
...
20:ZZZ
PC
8
pmem
+4
NOP
XXX
IR1
PC1
BF K
4
+
regs
20
IR2
PC1
D2
B2
A2
SFEQ
ALU
IR3
Z3
D3
SR
F=?
dmem
IR4
Z4
D4

26


20
PC
C
10
20
pmem
+4
NOP
YYY
NOP
IR1
PC1
XXX
+
regs
adr
IR2
PC2
D2
B2
A2
BF K
4+K=20
ALU
IR3
Z3
D3
SR
F=1
SFEQ
dmem
IR4
Z4
D4
adr
0:SFEQ ...
4:BF K
8:XXX
C:YYY
...
20:ZZZ
PC
20
pmem
+4
NOP
ZZZ
IR1
PC1
NOP
+
regs
adr
IR2
PC2
D2
B2
A2
XXX
ALU
IR3
Z3
D3
SR
BF K
dmem
IR4
Z4
D4
SFEQ
adr

27

Pipelinediagram

PC	IR1	IR2	IR3	IR4
0				
4	SFEQ			
8	BF	SFEQ		
C	XXX	BF	SFEQ	
20	NOP	XXX	BF	SFEQ

Instr efter hoppet exekveras alltid

Vid taget hopp måste en NOP muxas in

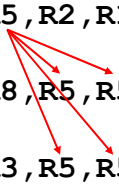
Diskussion: Hur ska de två muxarna i första steget styras?

```
if  ((IR2.op==BF and F==1) or (IR2.op==J))       /* taget hopp  */
   IR1 = NOP;
   PC = PC2;
else // annan instr, F=0
   IR1 = pmem;
   PC = PC+4;
```

29

Problem 2: Databeroende

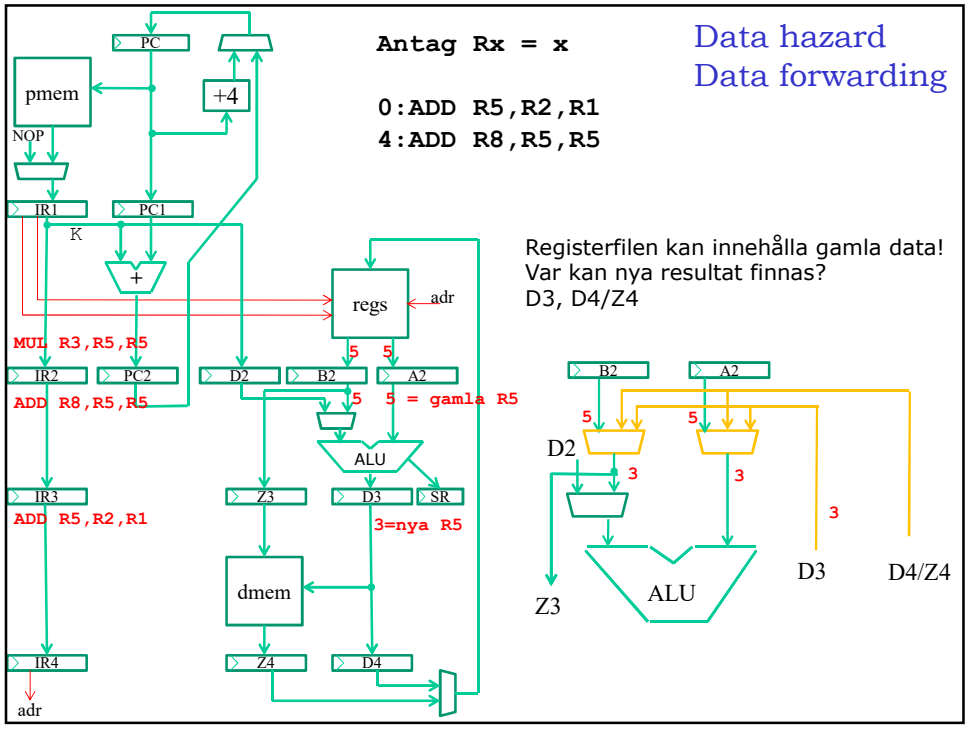
```
0:ADD  R5,R2,R1
4:ADD  R8,R5,R5
8:MUL  R3,R5,R5
```



Farlig situation, som måste åtgärdas direkt! Data Hazard

Problemet beror på att det tar flera klockcykler innan registret R5 uppdateras!

30



31

Hur ska de gula muxarna styras?
Det finns två situationer:

```
if IR2.op == ”instruktion som läser reg.”
   if  IR3.op == ”instruktion som skriver reg.”
     if IR2.a == IR3.d or IR2.b == IR3.d
        muxarna = …;

if IR2.op == ”instruktion som läser reg.”
   if  IR4.op == ”instruktion som skriver reg.”
     if IR2.a == IR4.d or IR2.b == IR4.d
        muxarna = …;
```

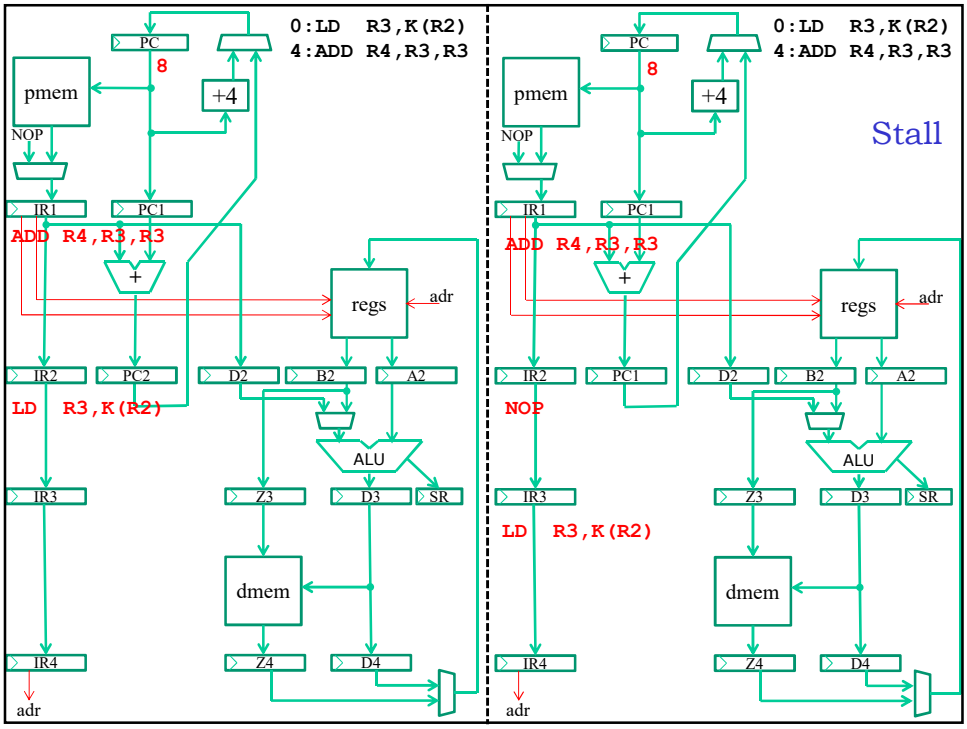
33

Problem 3: Stall

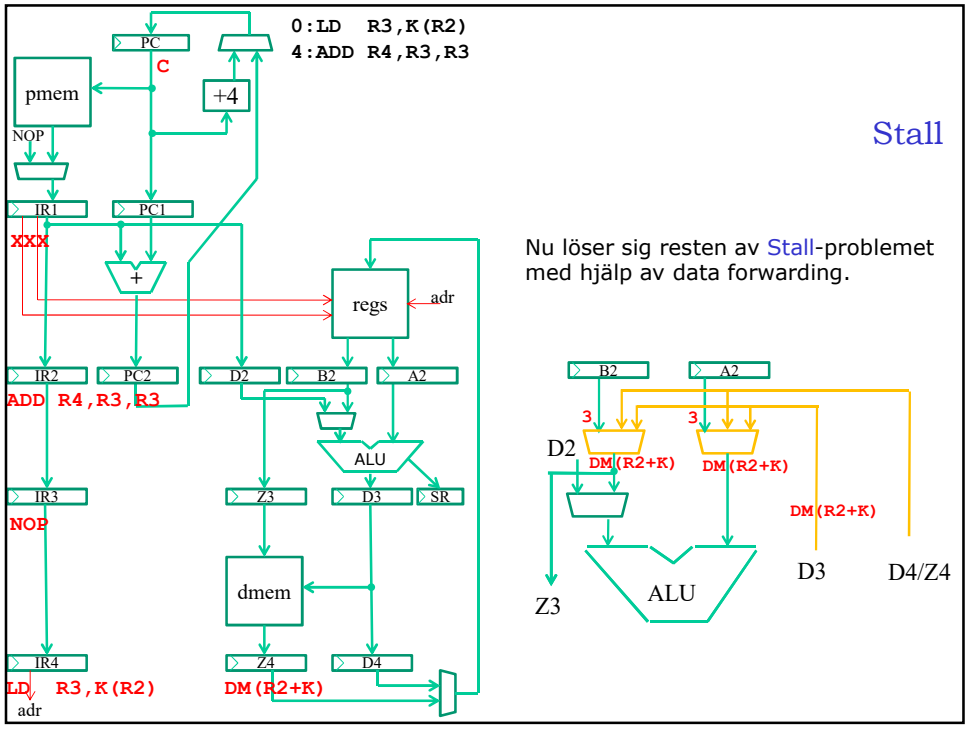
```
0:LD   R3,K(R2) ; läs från minnet
4:ADD  R4,R3,R3 ;
```

Problemet beror på att minnet sitter 1 klockcykel efter ALU-n.

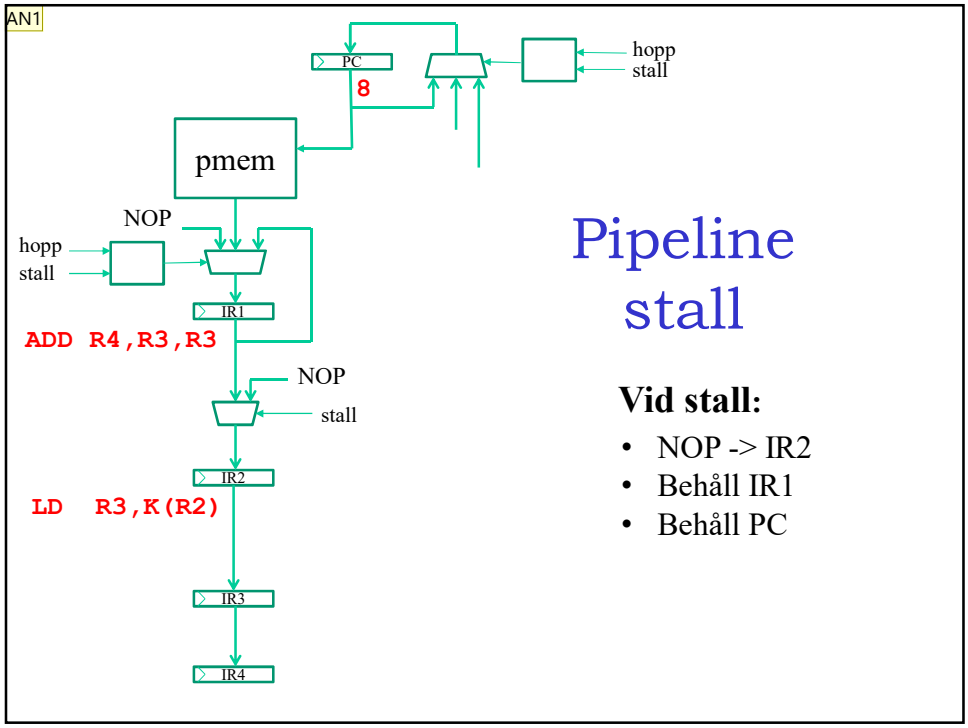
34



35


0:LD R3,K(R2)
4:ADD R4,R3,R3
PC
C
pmem
+4
NOP
IR1
PC1
XXX
+
regs
adr
IR2
PC2
D2
B2
A2
ADD R4,R3,R3
ALU
IR3
Z3
D3
SR
NOP
dmem
IR4
Z4
D4
LD R3,K(R2)
adr
DM(R2+K)
Stall
Nu löser sig resten av Stall-problemet med hjälp av data forwarding.
B2
A2
3
3
D2
DM(R2+K)
DM(R2+K)
DM(R2+K)
Z3
ALU
D3
D4/Z4

36


AN1
PC
8
hopp
stall
pmem
NOP
hopp
stall
IR1
ADD R4,R3,R3
NOP
stall
IR2
LD R3,K(R2)
IR3
IR4
Pipeline stall
Vid stall:
• NOP -> IR2
• Behåll IR1
• Behåll PC

38

Slide 38

AN1 Anders Nilsson; 2020-01-28