

# TSEA83 : Datorkonstruktion

## Fö5

Cacheminnen

# Fö5 : Agenda

- Pipelining
    - Repetition
  - Cacheminnen
    - Associativt minne som cache
    - Associativt minne som BPT
    - Direktmappad cache
    - Flervägs-cache (2,4)
    - I/D-cache
  - Benchmark
  - Gruppbildning
-

# Pipelining

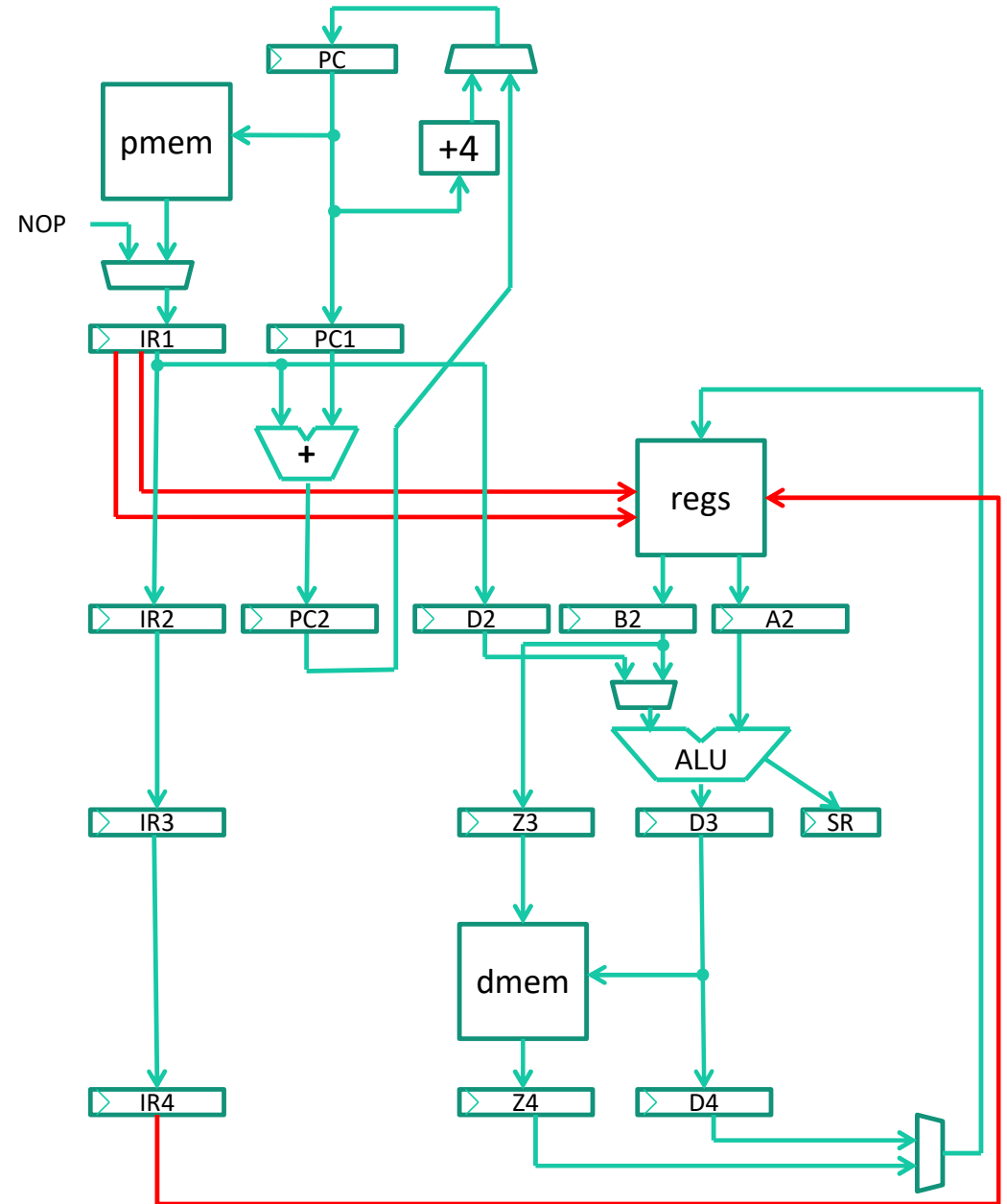
*Repetition*

# Pipelining, Repetition

Så här långt är pipelining enkelt!

Vi har sett att:

- Alla instruktioner tar 5 CP
- Och kan utföras med överlapp
- En instruktion går in i PL varje CP
- Och en lämnar PL varje CP



# Pipelining, Repetition

## Problem 1: Hopp (med ev. Straff)

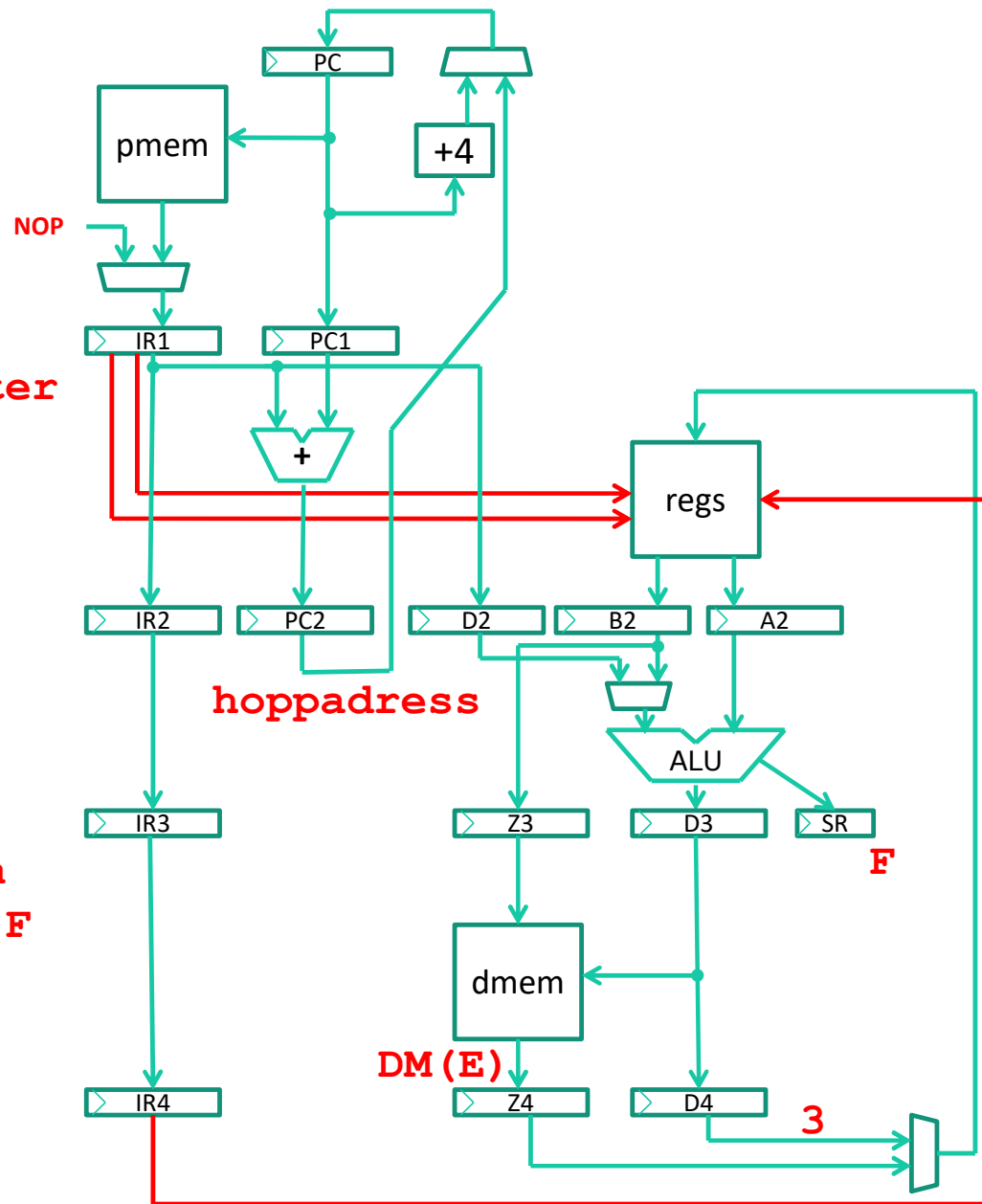
Instruktionen efter hoppet exekveras alltid.

Vid taget hopp måste en NOP petas in,  
för att inte ytterligare en instruktion ska  
komma in i pipen.

instr efter  
hopp

hopp

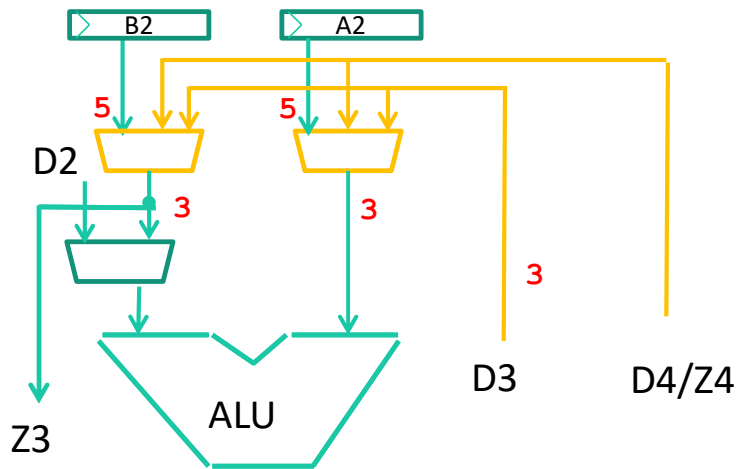
instr som  
påverkar F



# Pipelining, Repetition

Problem 2: Databeroende (data hazard)  
 Löses med data forwarding

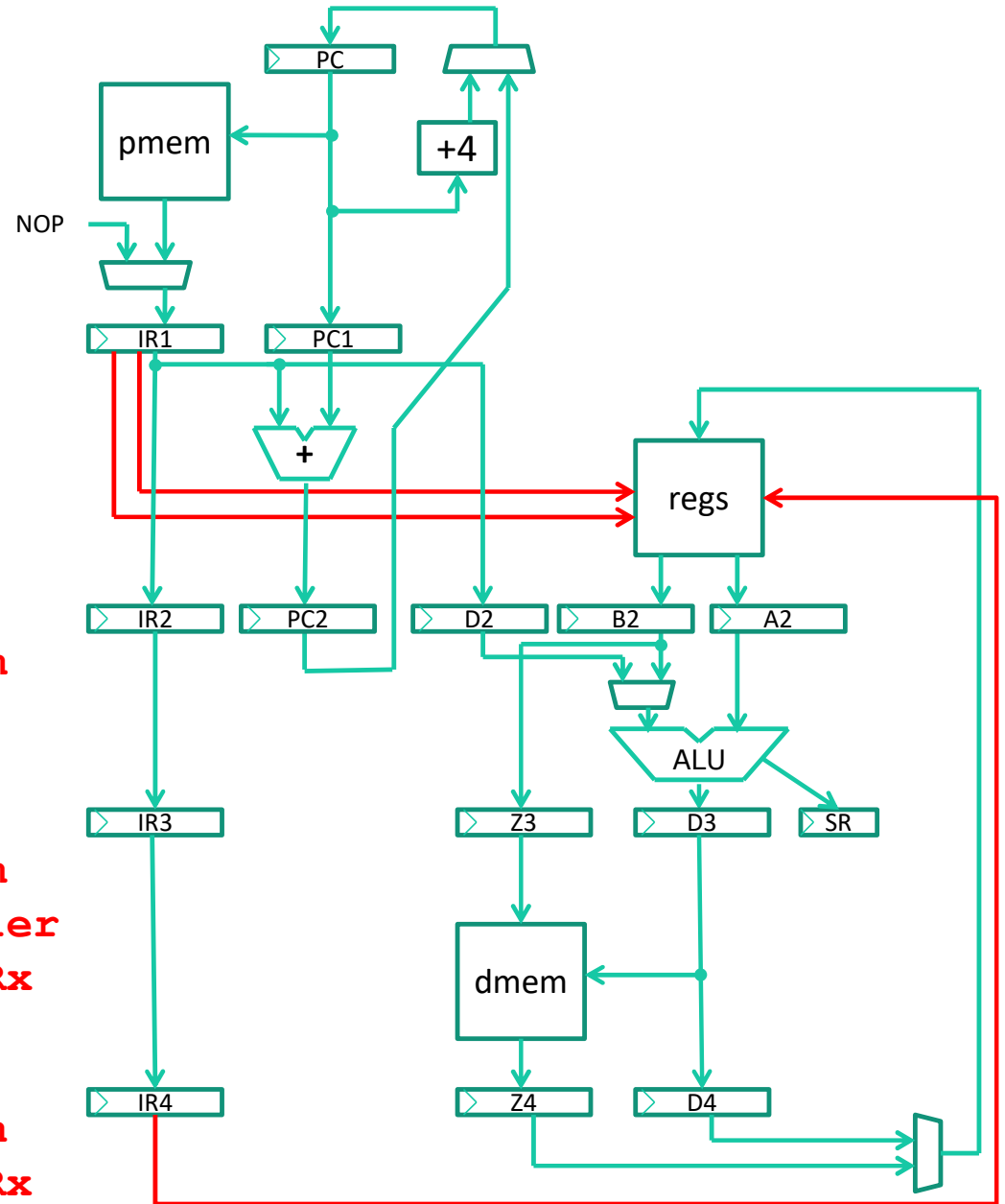
Uppdaterade registervärden skrivs tillbaka  
 sist i pipelinen. Under 2 CP finns det gamla  
 värden i regs.  
 Mha data forwarding försvinner problemet.



**instr som  
läser Rx**

**instr som  
läser eller  
skriver Rx**

**instr som  
skriver Rx**



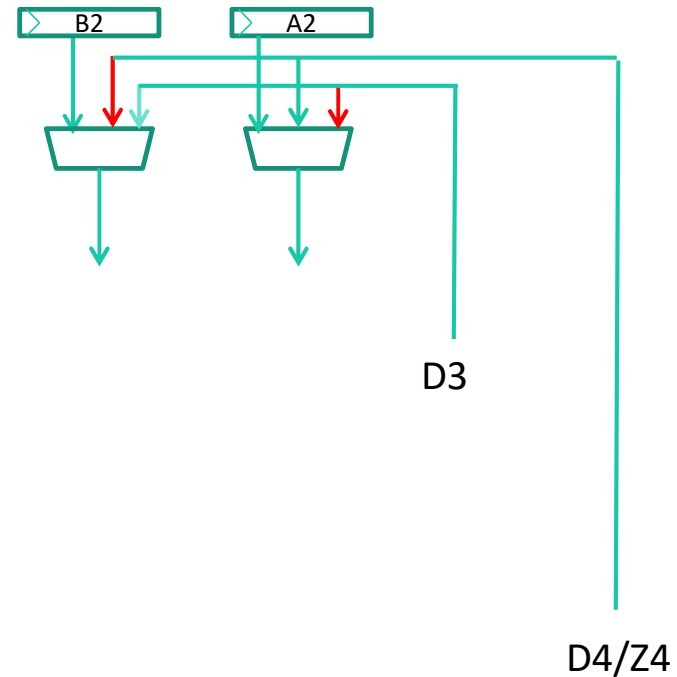
# Datorkonstruktion **Pipelining, Repetition**

## Problem 2: Databeroende (data hazard) Lösas med data forwarding

**Regs kan innehålla gamla data!**

Var kan nya resultat finnas? D3,D4/Z4

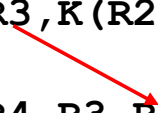
Hur ska muxarna styras?



# Datorkonstruktion **Pipelining, Repetition**

Problem 3: Minnesberoende  
Lösas med pipeline stalling + data forwarding

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



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



# Pipelining, Repetition

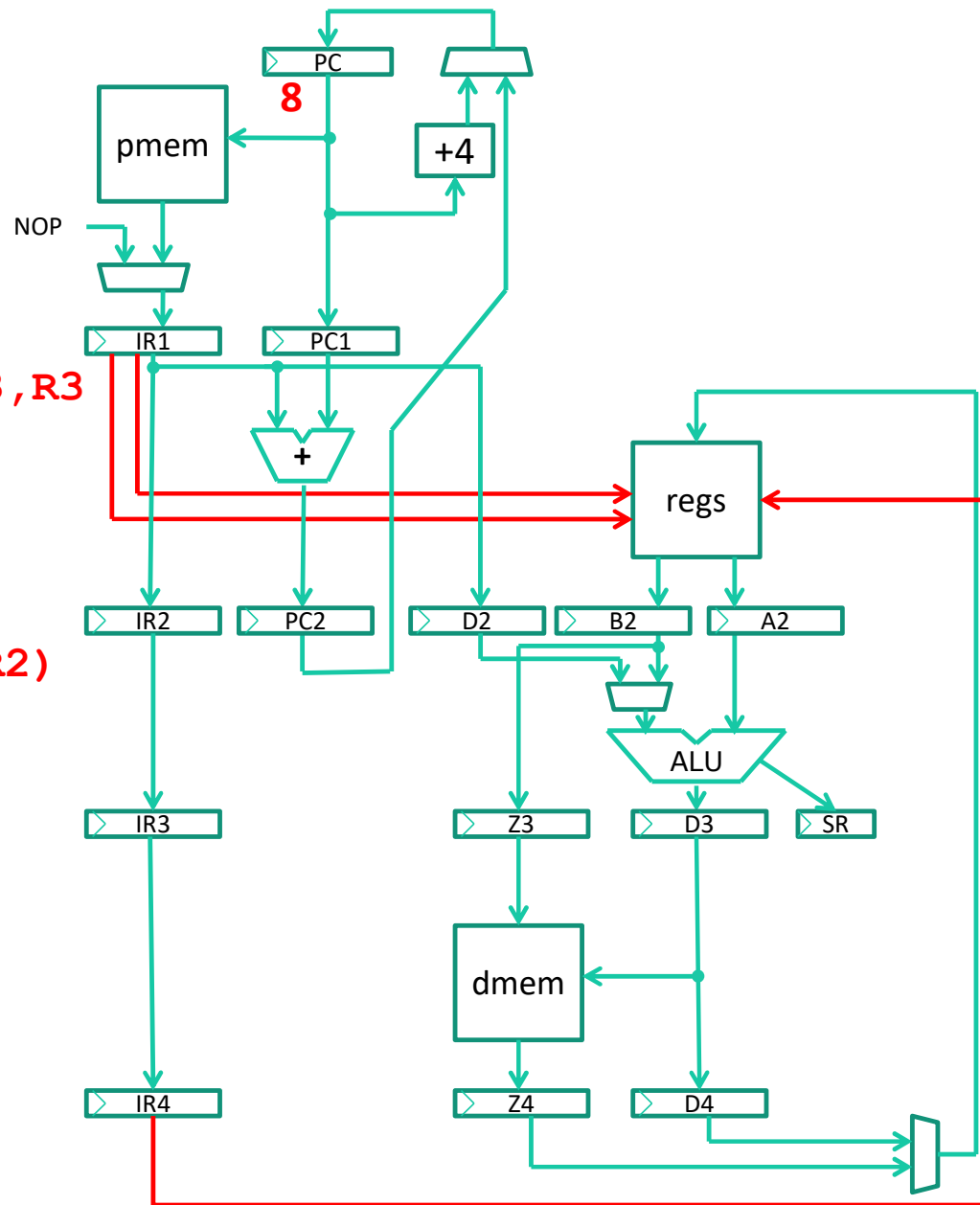
Problem 3: Minnesberoende  
 Löses med pipeline stalling + data forwarding

```
0: LD R3, K(R2)
4: ADD R4, R3, R3
```

**ADD**-instruktionen läser gamla R3  
**LD** har ännu inte läst minnet, och nytt värde på R3 kan fås (via Z4) först två steg senare (i writeback).

ADD R4, R3, R3

LD R3, K(R2)



# Pipelining, Repetition

Problem 3: Minnesberoende

Löses med pipeline stalling + data forwarding

0 : LD R3 ,K (R2)

4 : ADD R4 ,R3 ,R3

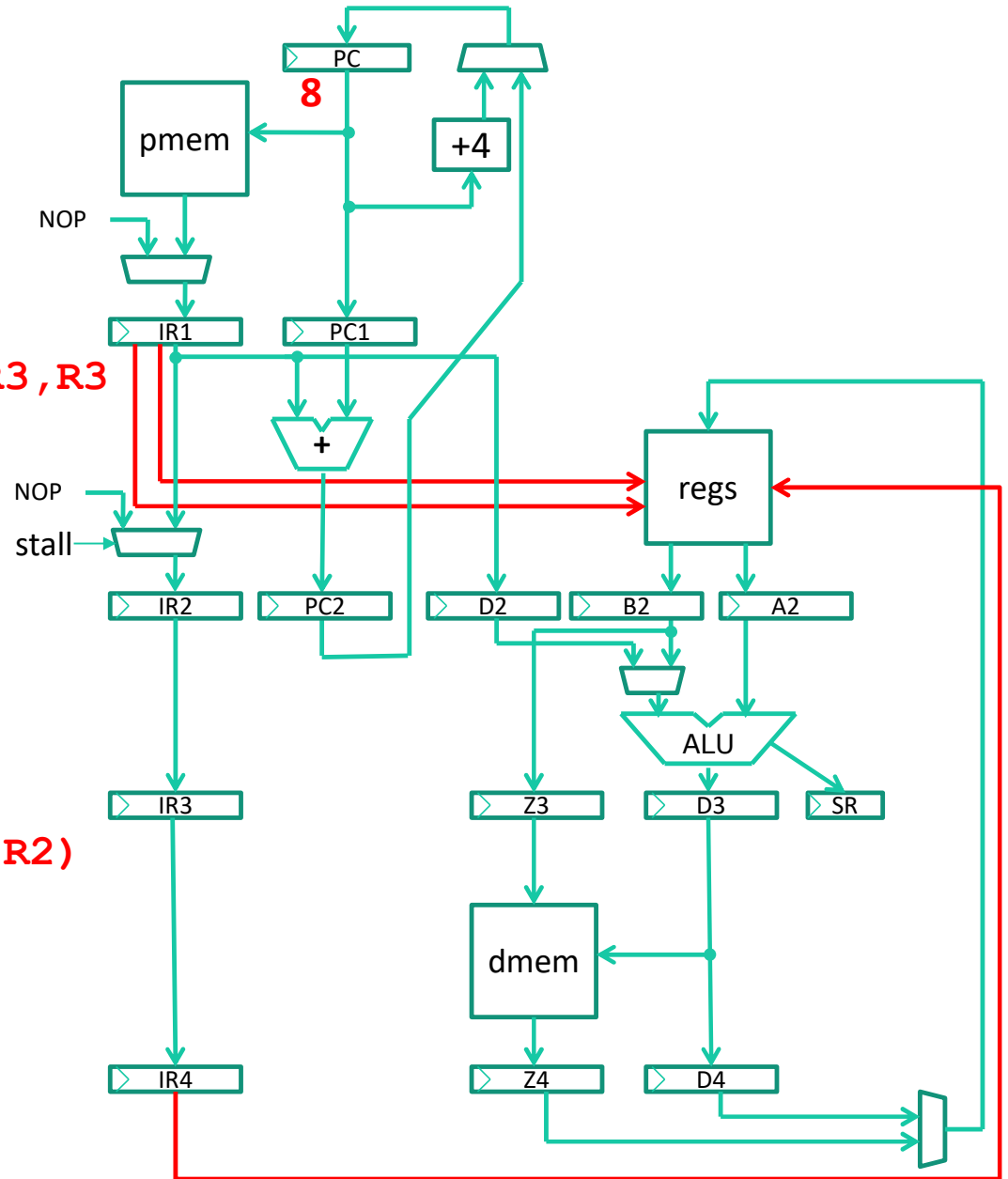
Löses med Pipeline stall

kräver en till mux, som petar in en **NOP** (mellan **ADD** och **LD**), samt att **ADD** står stilla (**stall**).

ADD R4 ,R3 ,R3

NOP

LD R3 ,K (R2)

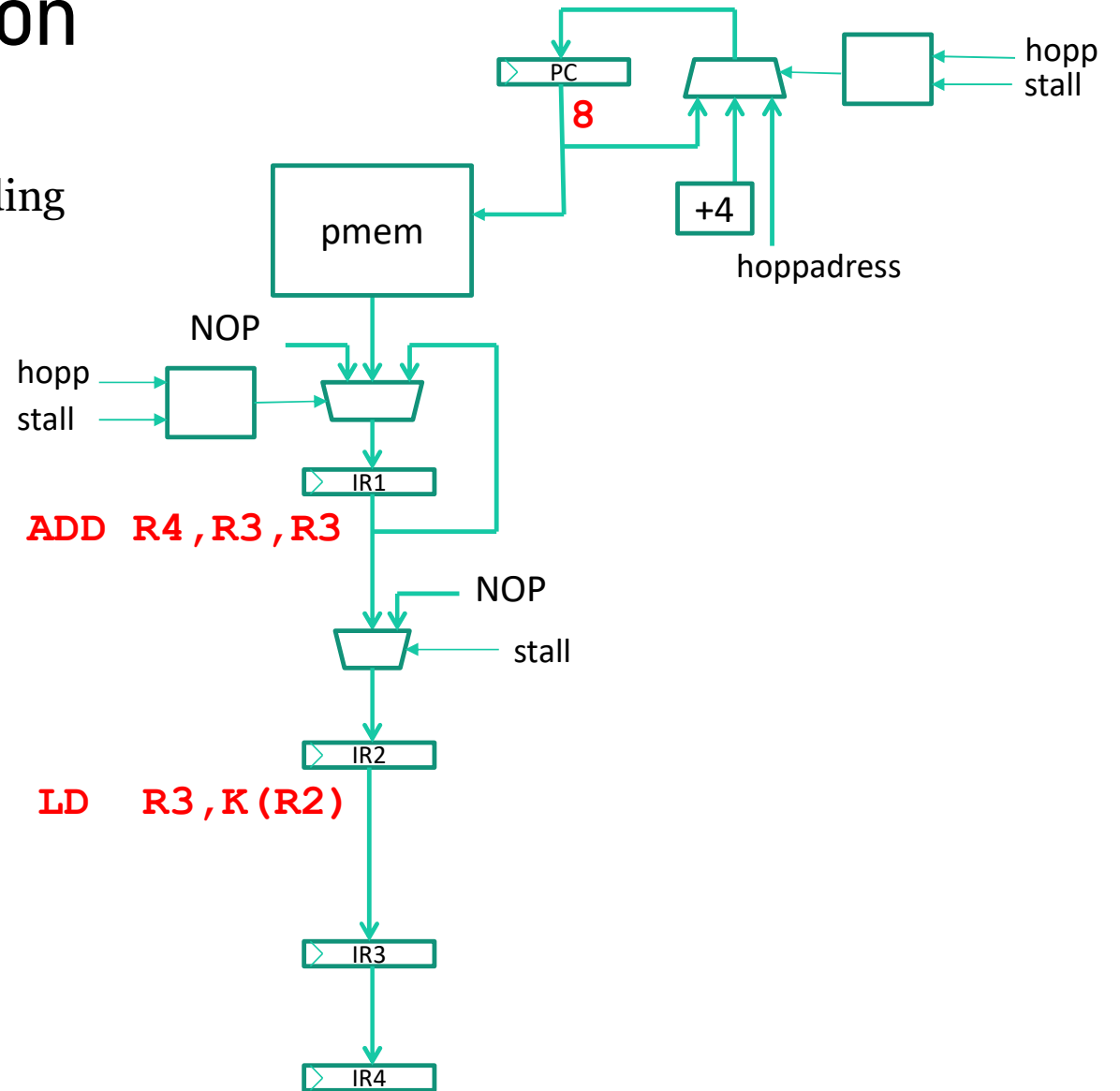


Datorkonstruktion **Pipelining, Repetition**

Problem 3: Minnesberoende  
Lösas med pipeline stalling + data forwarding

**Vid stall:**

- NOP -> IR2
- Behåll IR1
- Behåll PC



# Datorkonstruktion **Pipelining, Repetition**

Problem 3: Minnesberoende

Löses med pipeline stalling + data forwarding

**Observera att stall innebär prestandaförlust!**

```
LD R2, 0(R0)
ADD R3, R2, R1
```

Vi kan lika gärna skriva

```
LD R2, 0(R0)
NOP
ADD R3, R2, R1
```

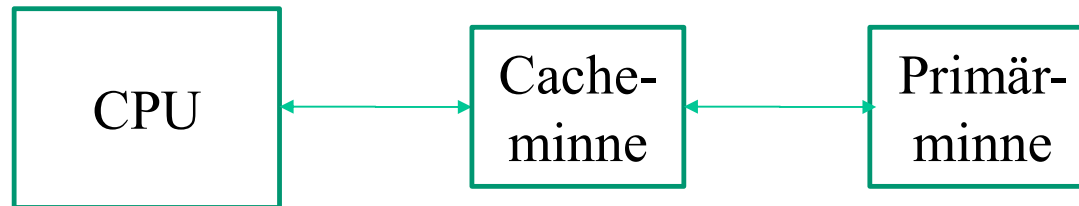
Det går kanske att flytta en annan instruktion till NOP:ens plats?  
Ovanstående bor kanske i **heta loop**, där varje klockcykel är viktig.

# Cacheminnen

# Cacheminnen

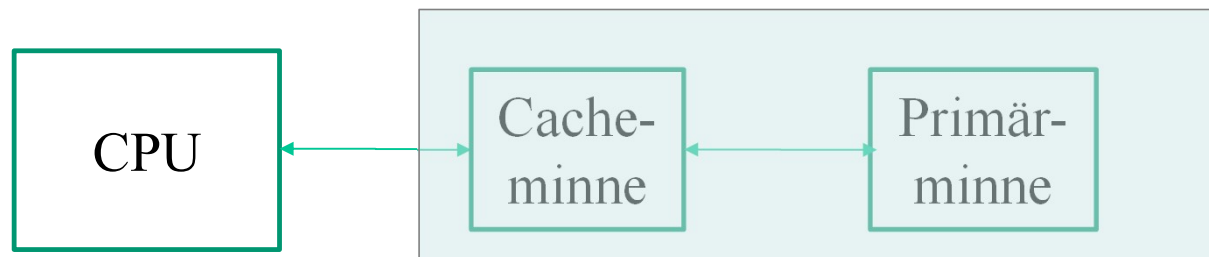
Snabbt,  
litet

Långsamt,  
stort



- CPU:n är normalt mycket snabbare än primärminnet. Det tar kanske 10-50 CP för att läsa ett data från minnet.
- Cacheminnet innehåller kopior av de instruktioner/data som används mest
- Kopieringen sker helt automatiskt, utan att programmeraren behöver tänka på det
- CM finns i CPU-chipet
- CM innehåller ett minne och en styrenhet.

Snabbt, stort

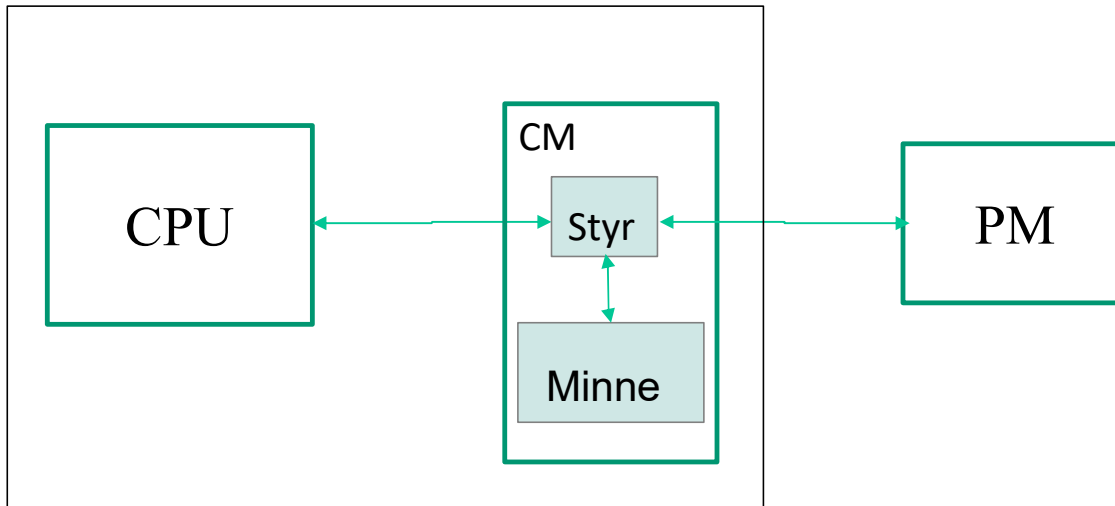


# Cacheminnen

Lönar sig cacheminne i vår FPGA?

- I FPGA-n finns sammanlagt 64 kB sk block RAM.
- Det räcker inte med dessa, så **nej!**
- Utanför FPGA-n finns externt minne (flera MB) men med accesstiden 7 CK.
- Det är stort nog, så förvisso **ja!**
- Ett CM måste i FPGA:n byggas med block RAM för att bli tillräckligt snabbt.

# Datorkonstruktion Cachelinnen



- När datorn startar är cachelinnet tomt
- Varje minnesaccess ger en **cachemiss**, kopiering till PM->CM
- Vanligt är att vid en cachemiss kopiera flera (t ex 4) instruktioner/data till CM. Kallas en **cacheline**.
- Sannolikheten är nu stor (t ex 95%) för **cacheträff**.



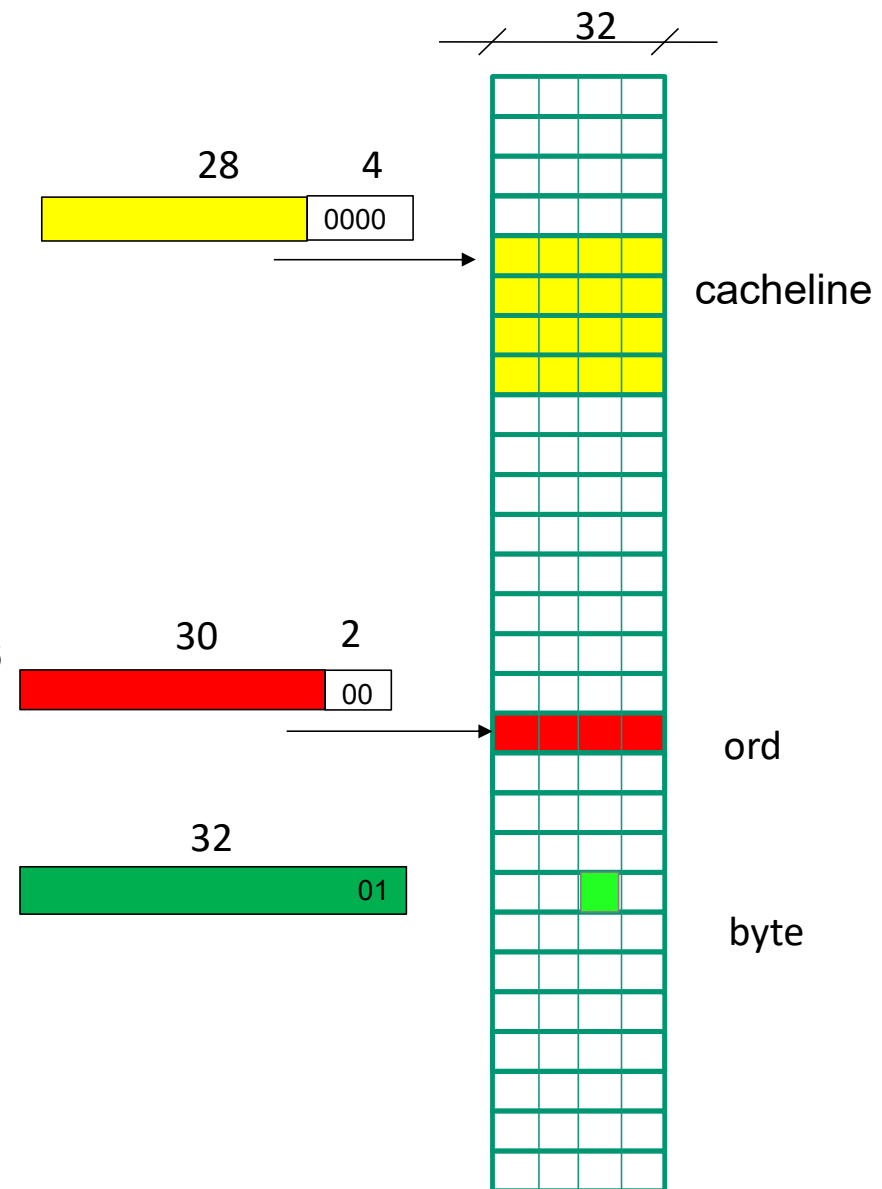
# Cacheminnen

- ✦ Program innehåller loopar => sannolikheten är stor att samma instruktion accessas igen  
**(temporal lokalitet)**
- ✦ Samma sak gäller ofta för data
- ✦ Program är sekvensiella => sannolikheten är stor att också nästa instruktion accessas  
**(spatial lokalitet)**
- ✦ Dataaccesser brukar också vara lokala

# Cacheminnen

## Minnesbegrepp

- Minnesadress: 32 bitar
- Minnets bredd: 32 bitar, dvs 4 bytes
- Cacheline: 4 ord, dvs 16 bytes
- Möjligt att adressera bytes
- Ord finns på en multipel adress av 4
- Cachelines på en multipel adress av 16



# Cacheminnen

## Associativt minne

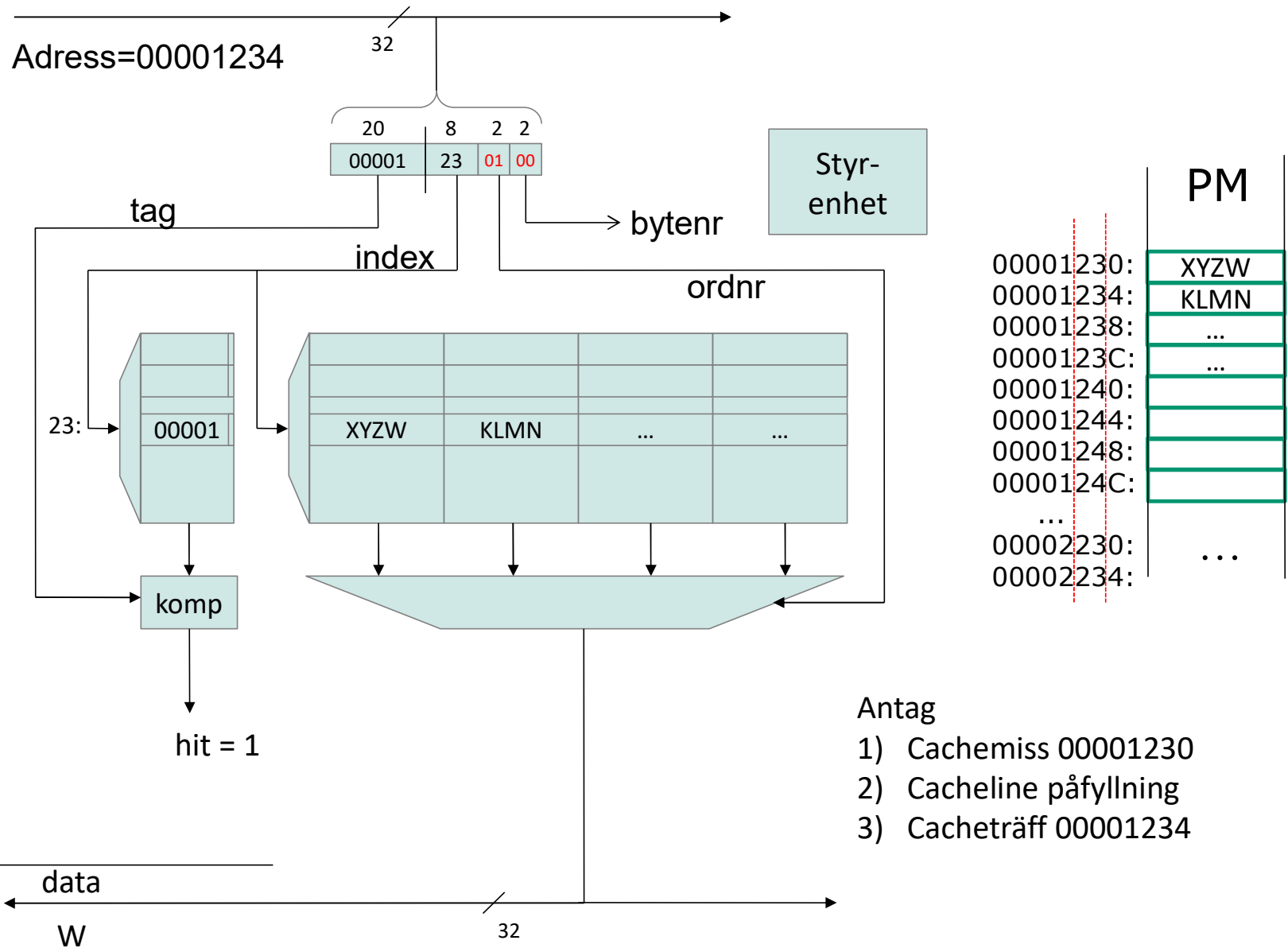


# Cacheminnen

## Associativt minne

- Associativt minne (AM) är det mest generella sättet att bygga ett CM.
- Data kan placeras var som helst i AM.
- Vi hoppar över diverse problem:
  - Vilka rader är lediga?
  - När AM är fullt, vilken rad ska skrivas över?
- AM är dyrt => en komparator per rad!
- AM används inte till cache för instruktioner/data
  
- En sk **direct mapped cache** har endast 1 komparator
- Data kan **inte** placeras var som helst!

# Cacheminnen Direktmappad



# Cacheminnen

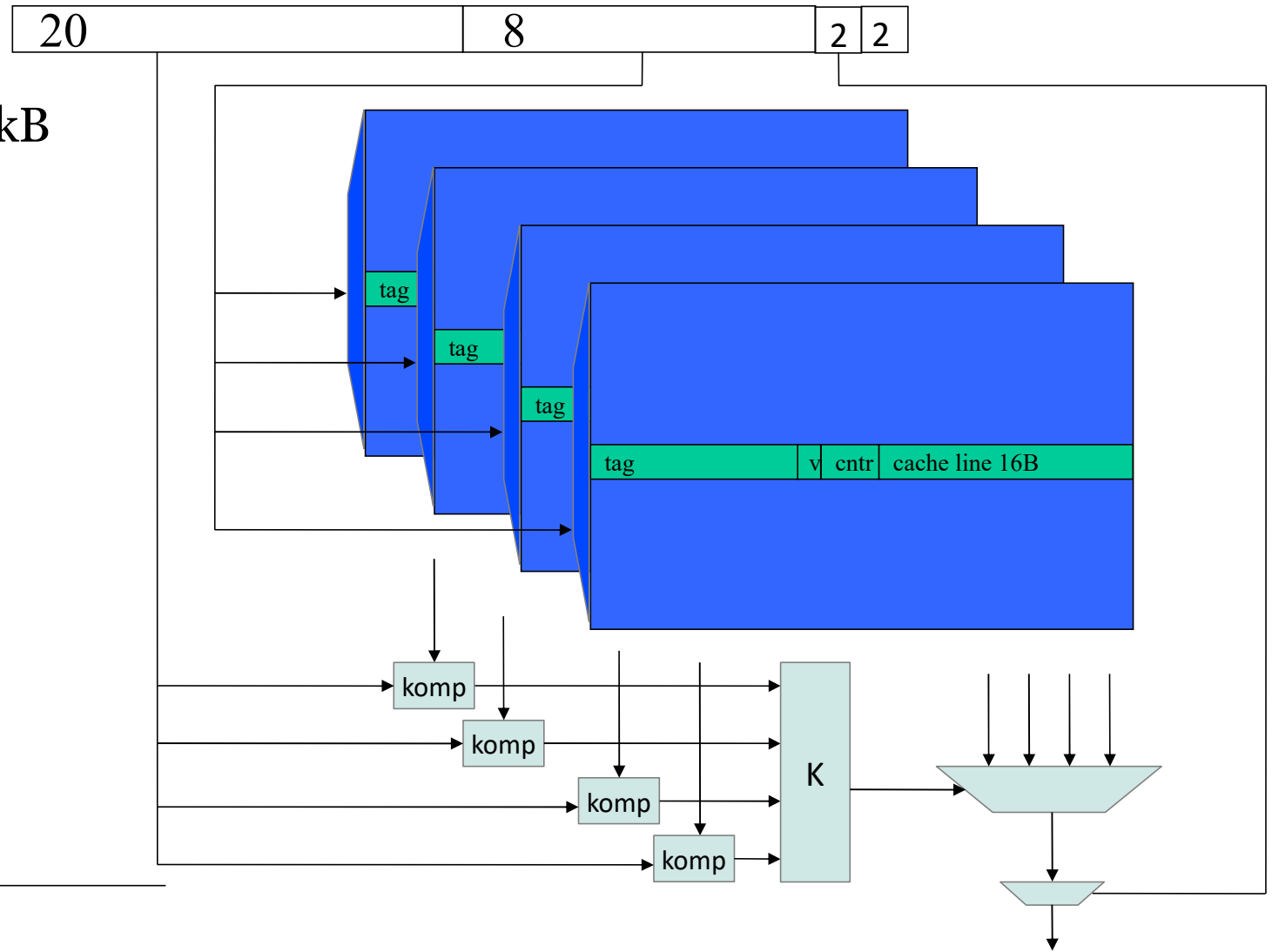
## Förbättring av direktmappad

Vi gör en riktig cache.

Vi gör cachen 4 gånger större, associativitet=4,  
genom att parallellkoppla 4 st 4kB cachar,  
alltså 16kB totalt.

# Cacheminnen

4-vägs cache, 16 kB



# Cacheminnen

- När man kört en stund, så är cachen full
- Vid en cachemiss, så ska ju cachen uppdateras, och ett data skrivs över
- En flervägscache behöver en utbytesalgoritm, vem blir offret? LRU är en vanlig metod

**LRU = least recently used.**

Till varje cacheline hör en räknare:

Hit: max->cntr

Miss: välj cacheline med minst cntr. Fyll på. max->cntr

Inget: räkna ner alla cntr (aging, inte på varje cp)



# Cacheminnen

Mycket vanligt är att ha separata cacheminnen för instruktioner/data

För datacachen tillkommer då vad som ska ske vid skrivning:

**Write-thru:** skriv till CM och PM (PrimärMinne)  
alltså CM,PM uppdateras samtidigt

**Write-back:** PM uppdateras endast vid cachemiss

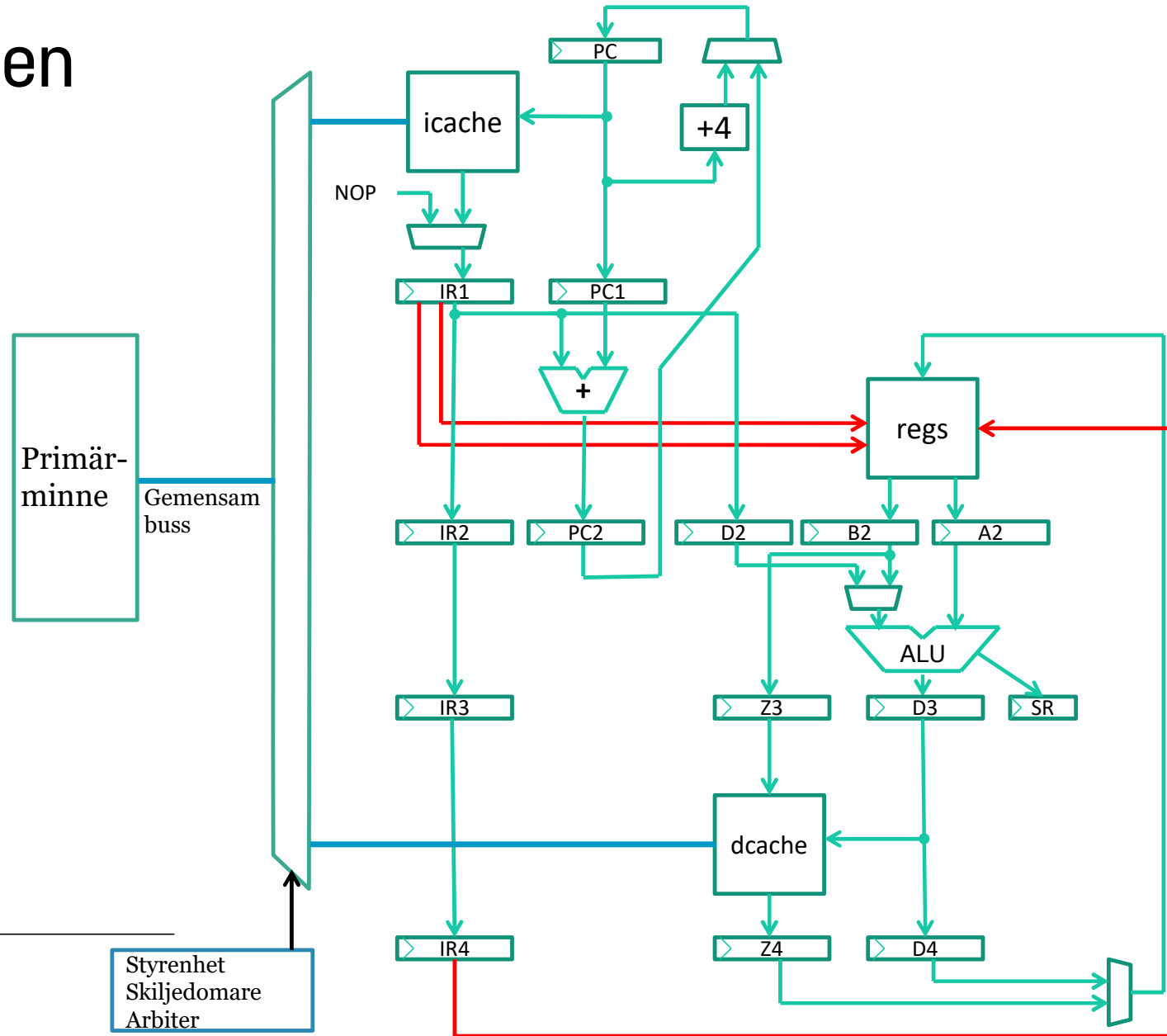
ytterligare en bit per cacheline: **dirty bit.** (CM förändrat)

Hit: läs/skriv CM

Miss: uppdatera PM om dirty=1. Skriv över.

# Cacheminnen

Typisk  
arkitektur



# Cacheminnen

## Typisk arkitektur

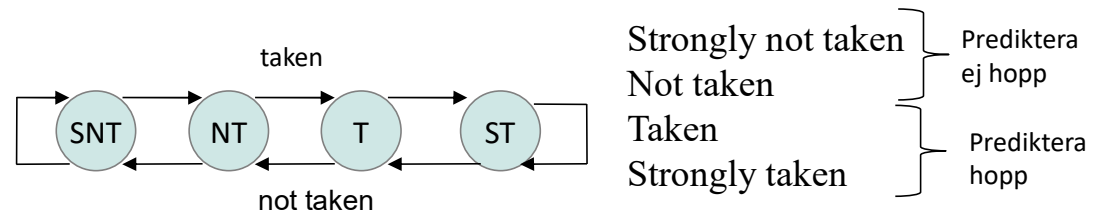
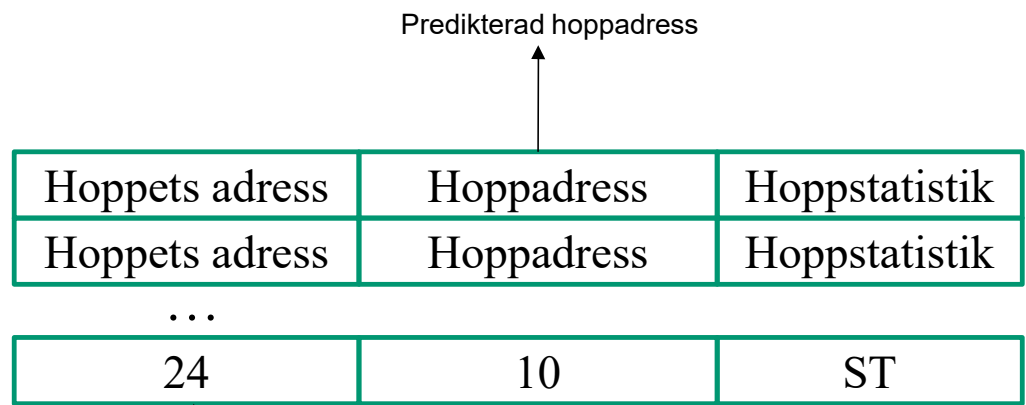
- Vid cachemiss i I-cache eller D-cache måste pipelinen stoppas.
- Båda cacharna använder en gemensam buss mot minnet. Samtidig miss i I-cache och D-cache, innebär påfyllning av en cache i taget. Pipelinen fryst hela tiden.
- Observera att write-thru belastar den gemensamma bussen.

# Cacheminnen, BPT = Branch Prediction Table

Ett försök att få bort den där NOP-en vid hopp

Skiss: Varje gång ett hopp påträffas

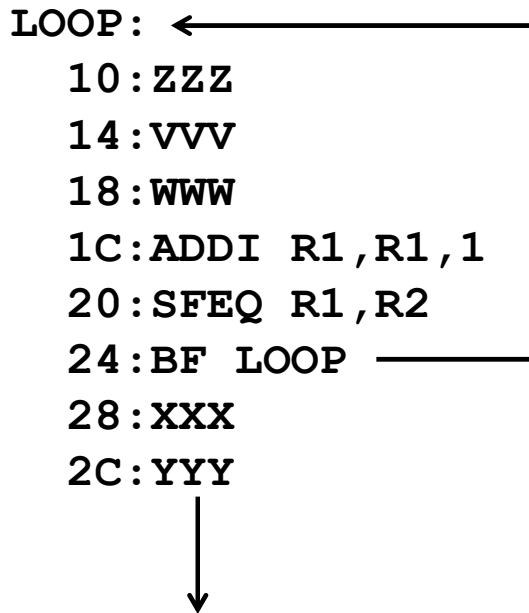
- 1) Finns inte i BPT: stoppa in det och uppdatera hoppstatistiken
- 2) Finns i BPT: läs ut hoppadressen



# Datorkonstruktion Cacheminnen

## BPT (skiss)

Branch prediction table.  
Vi väntar inte på F,  
vi chansar mha BPT!

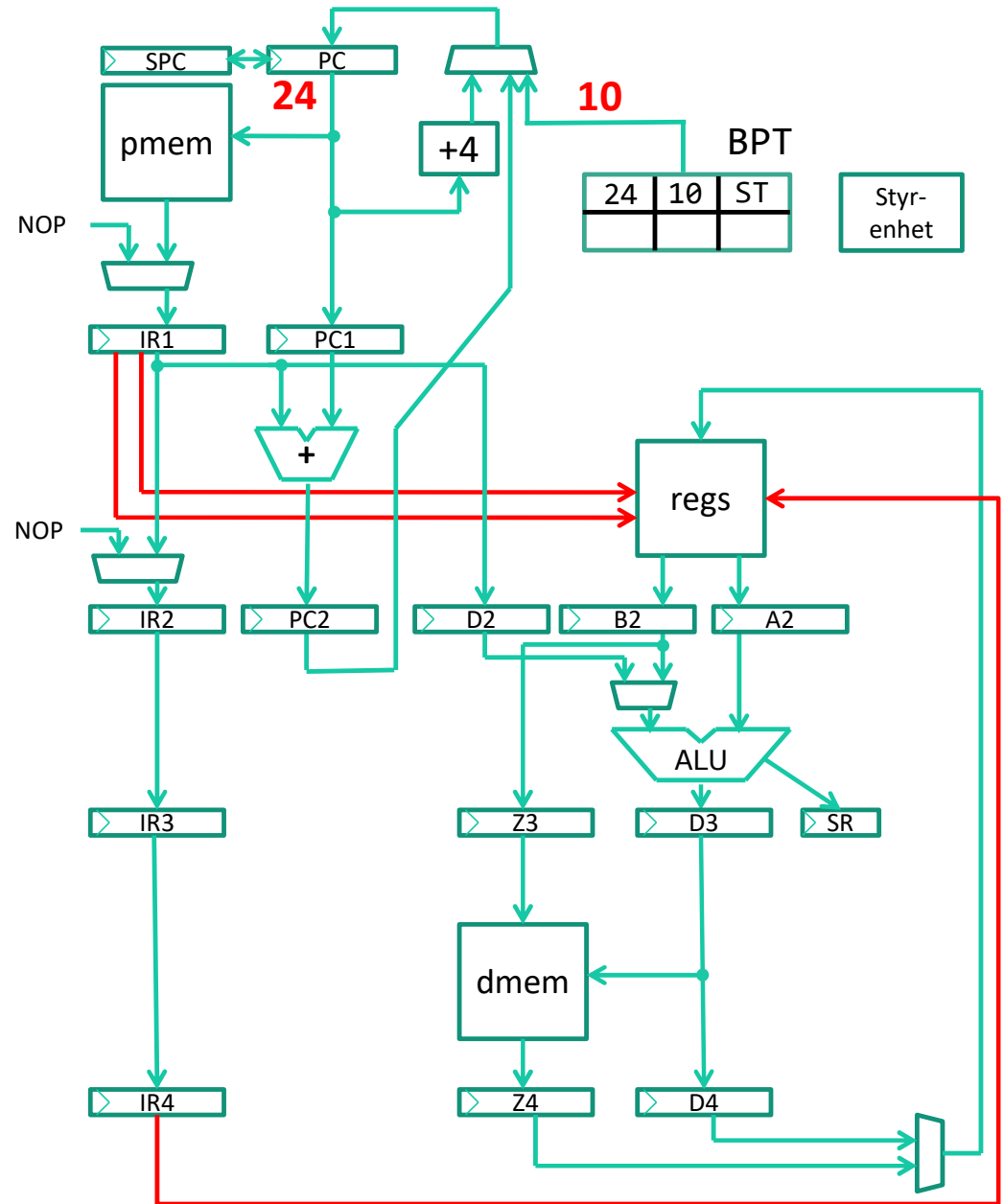


**BF**

**SFEQ**

**ADDI**

**WWW**



Datorkonstruktion **Cacheminnen**

**BPT (skiss)**

Spara alternativadressen (28) i SPC (försäkring).

Ersätt BF med NOP (ska inte utföras i EXE en gång till)

```

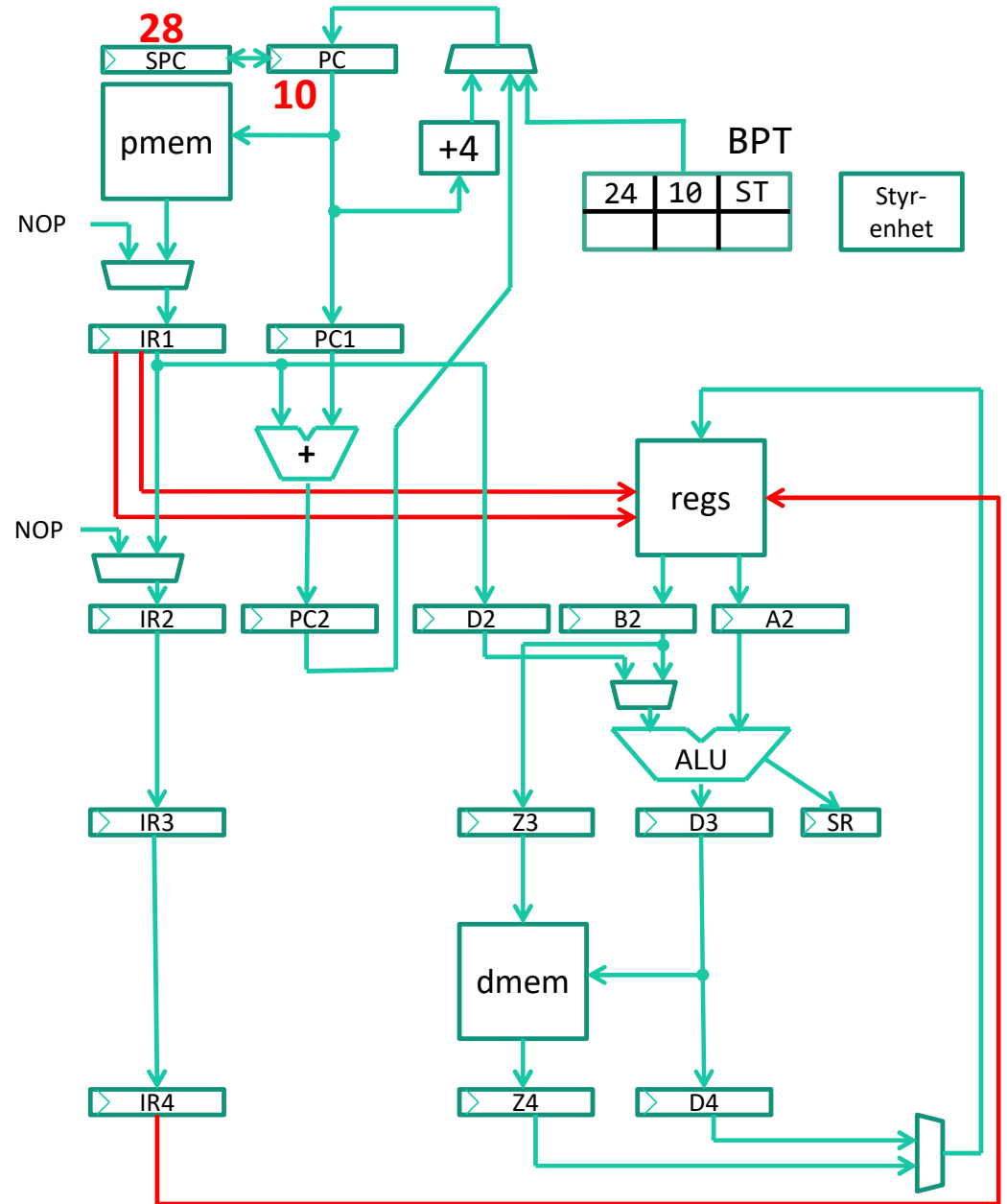
LOOP: ←
10: ZZZ
14: VVV
18: WWW
1C: ADDI R1, R1, 1
20: SFEQ R1, R2
24: BF LOOP
28: XXX
2C: YYY
    
```

**ZZZ**

**BF** NOP

**SFEQ**

**ADDI**

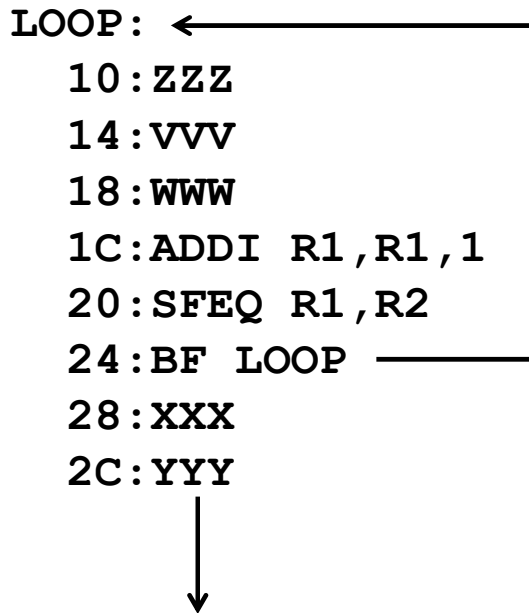




# Datorkonstruktion Cacheminnen

## BPT (skiss)

Om hoppet INTE ska tas,  
återställs PC och VVV samt  
ZZZ blir NOP.  
BPT antar T.

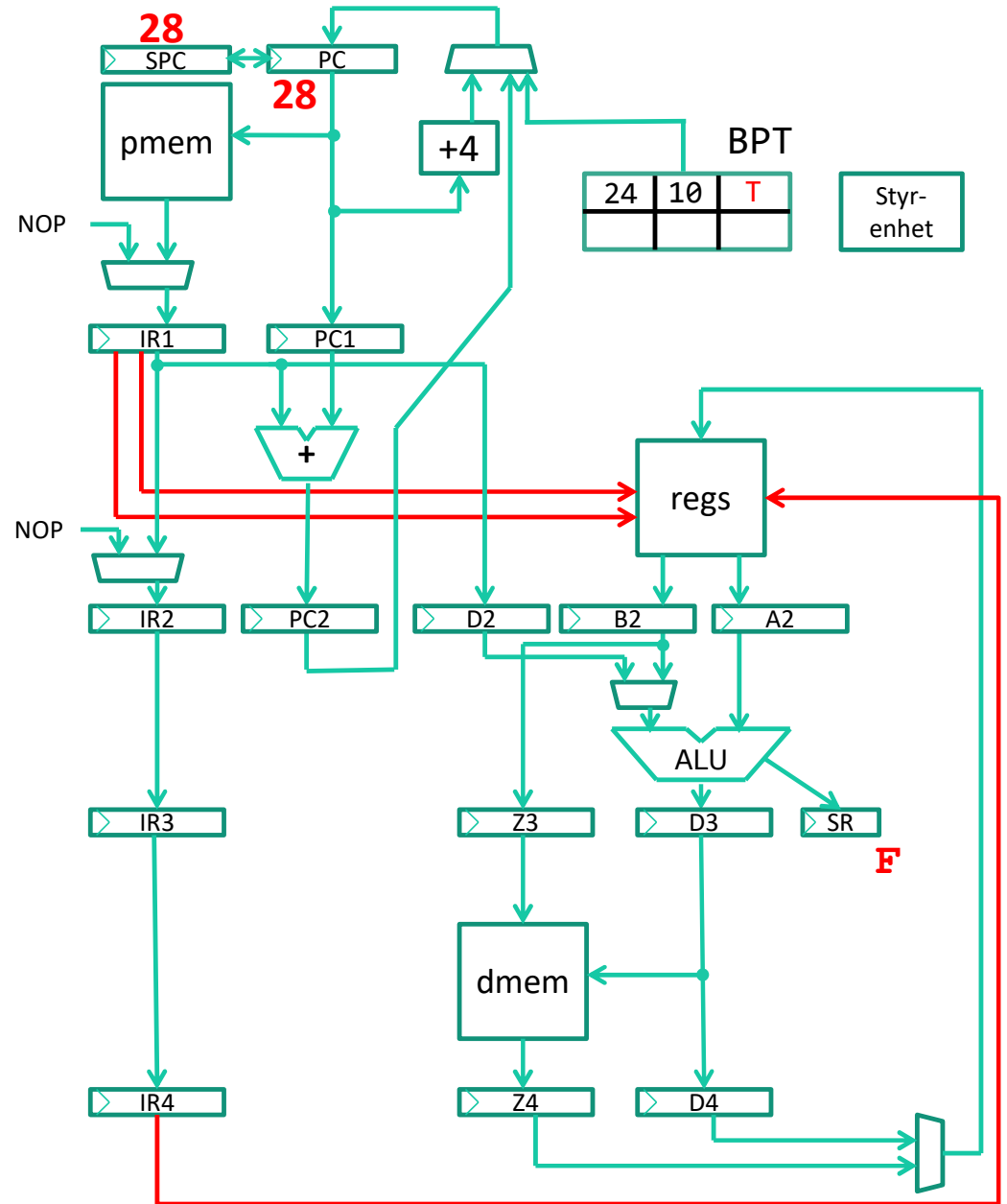


**XXX**

**VVV** NOP

**ZZZ** NOP

**BF** NOP





# Benchmark

## Benchmark

```

// Andreas Ehliar
...
int buf[4096]
...
unsigned int countones(void)
{
    unsigned int i;
    unsigned int numones = 0;
    unsigned int numzeroes = 0;
    volatile unsigned int dummy;

    for(i=0; i < BUFSIZE; i++){
        if(buf[i] == 1){
            numones++;
        }else{
            numzeroes++;
        }
    }

    dummy = numzeroes;
    return numones;
}

int main(int argc, char **argv)
{
    unsigned int i, numones;
    fillmem();

    for(i=0; i < 1000000; i++){
        numones = countones();
    }
    printf("Number of ones: %d\n", numones);
}

```

Mönster	Tid	
Bara 1:or	17 s	3.5 s
Varannan 1:a, 0:a	58 s	3.5 s
Slumpföljd av 1:or och 0:or	58 s	21 s

Kört på pandaboard  
(ARM Cortex A-9  
Smartphone-CPU)

Laptop core i5

# Design of a baseband processor

Eric Tell & Anders Nilsson

Coresonic (uppköpt av Mediatek)

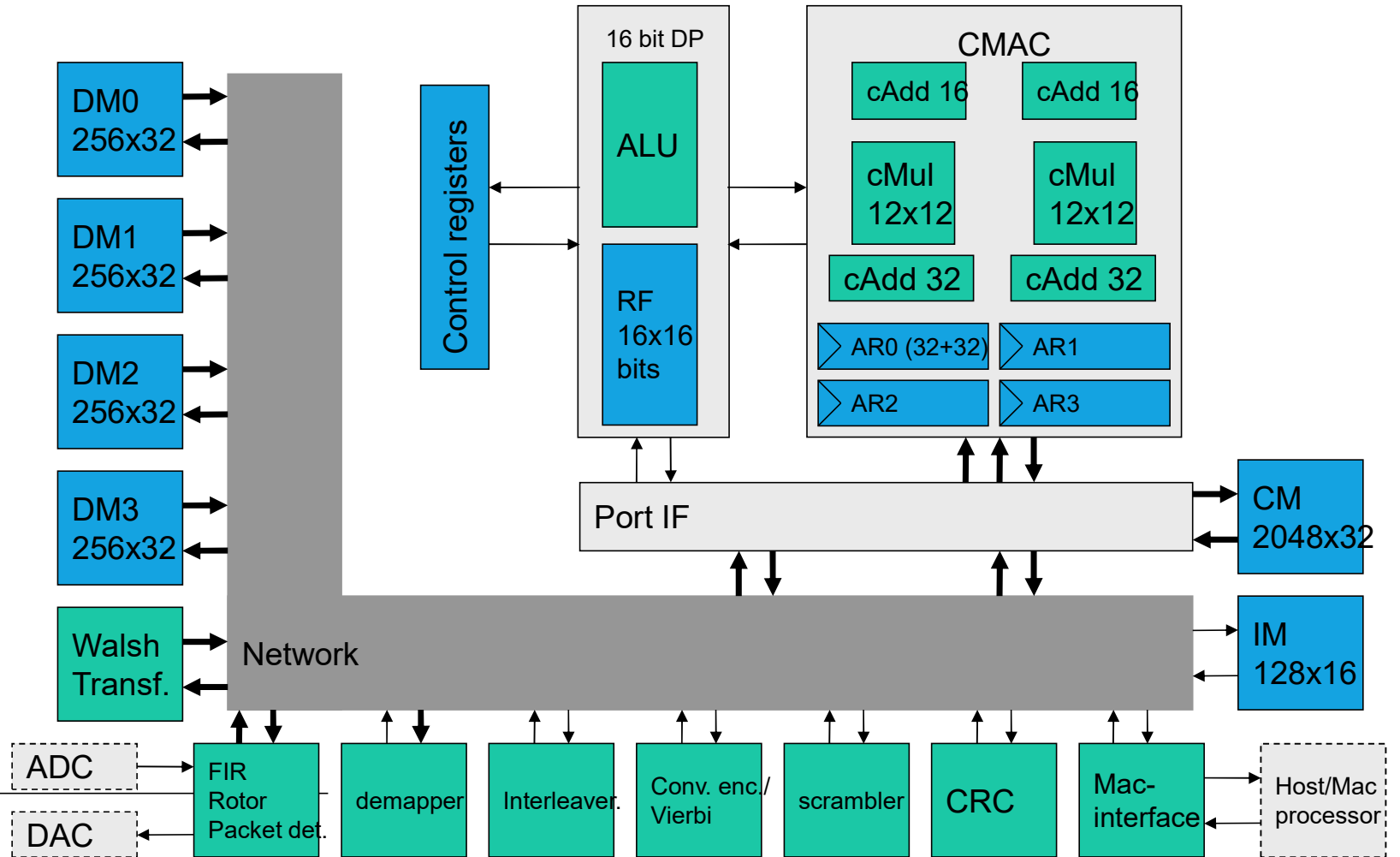
[Coresonic \(nyteknik.se\)](http://nyteknik.se)

[MediaTek Announces Acquisition of Leading DSP Technology Provider Coresonic AB | MediaTek](#)

- Ska klara så många standarder som möjligt
- WLAN, 3G, DVB-T, ...
- Använder en kombination av mjukvara och hårdvara (acceleratorer)

# Design of a baseband processor

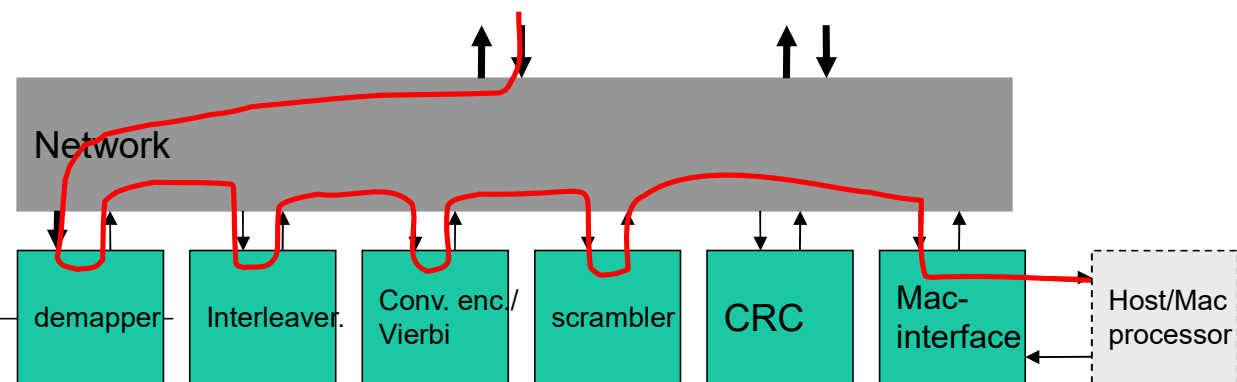
## Overview of BBP1



# Design of a baseband processor

## Network features

- A number of accelerators can be connected in a chain
  - No synchronization overhead in the core!
  - No intermediate memory storage!
  - High degree of parallelism!



# Design of a baseband processor

## Chip Features:

Process	0.18 $\mu\text{m}$ CMOS
Core voltage	1.8 V
I/O voltage	3.3 V
Chip area	5.0 mm <sup>2</sup>
Core area	2.9 mm <sup>2</sup>
Logic area	1.9 mm <sup>2</sup>
Mem. area	1.0 mm <sup>2</sup>
Package	144 pin fpBGA

## Measured Max Frequency:

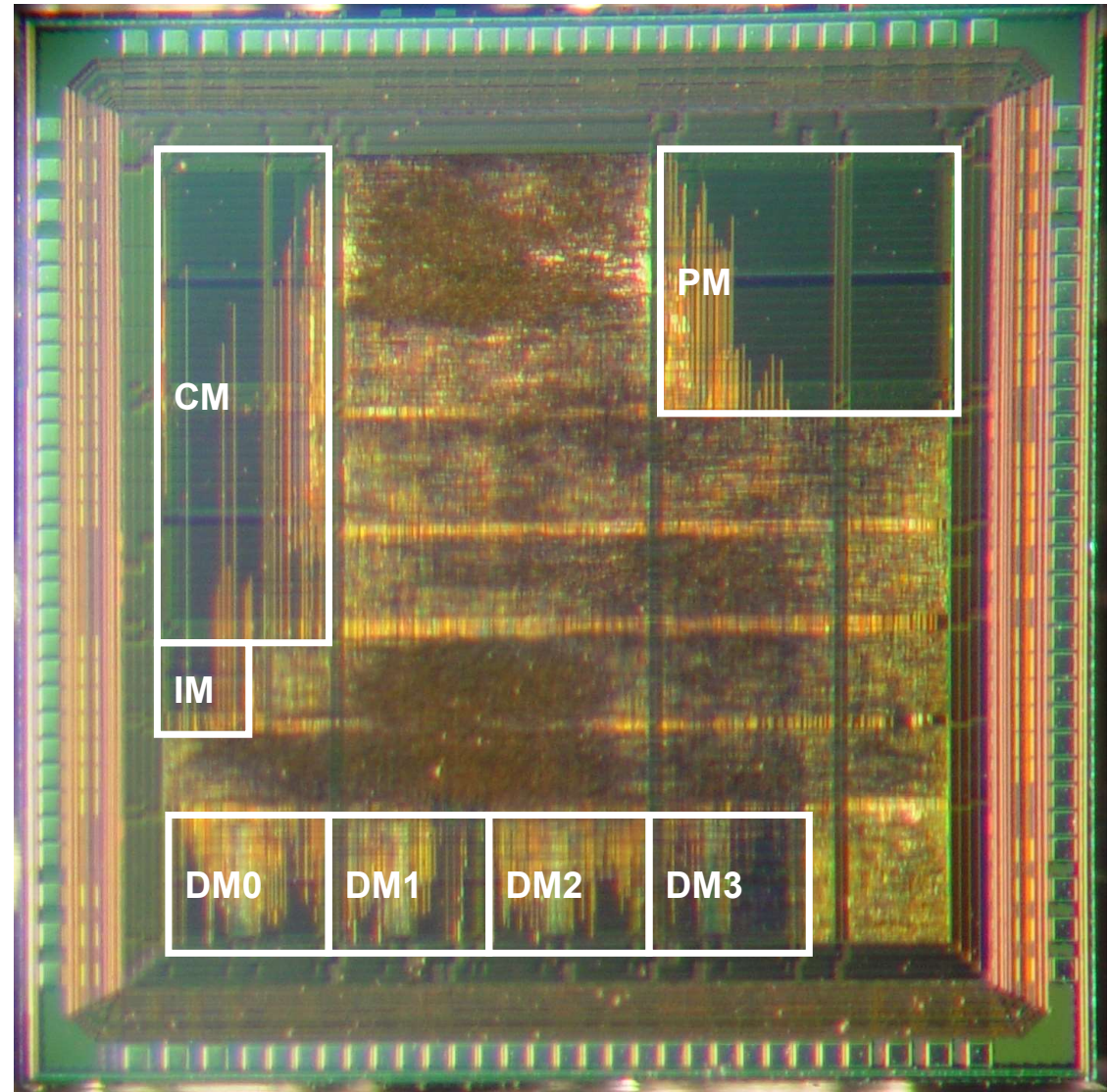
220 MHz

## Measured Core Power

@160 MHz:

Idle 44 mW

11a Rx burst 126 mW



# Gruppbildning

Använd kanalen Gruppbildning i Teams  
och fyll i dokumentet TSEA83\_Gruppbildning\_VT2022.xlsx  
under Files.



Anders Nilsson

[www.liu.se](http://www.liu.se)