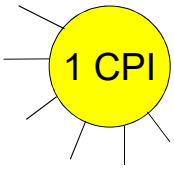
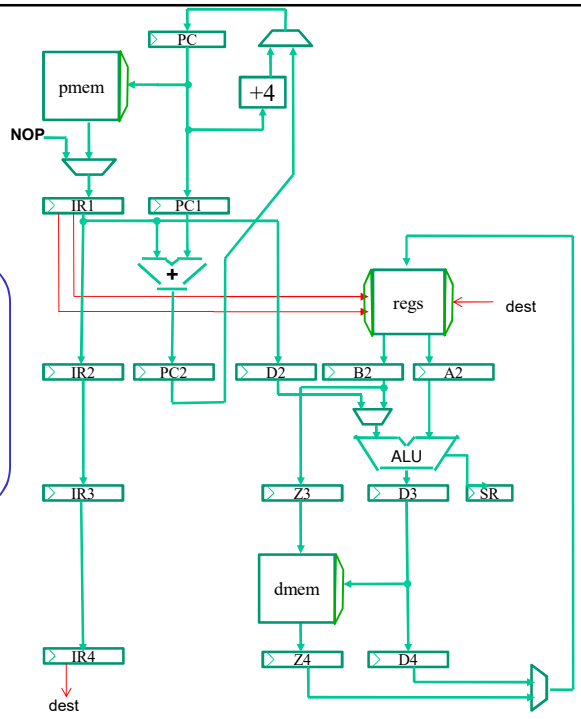
Föreläsning 5



- Sammanfattning pipelining
- Cacheminnen
  - associativt minne som cache
  - associativt minne som BPT
  - direkt-mappad cache
  - flervägscache (2,4)
  - I/D-cache

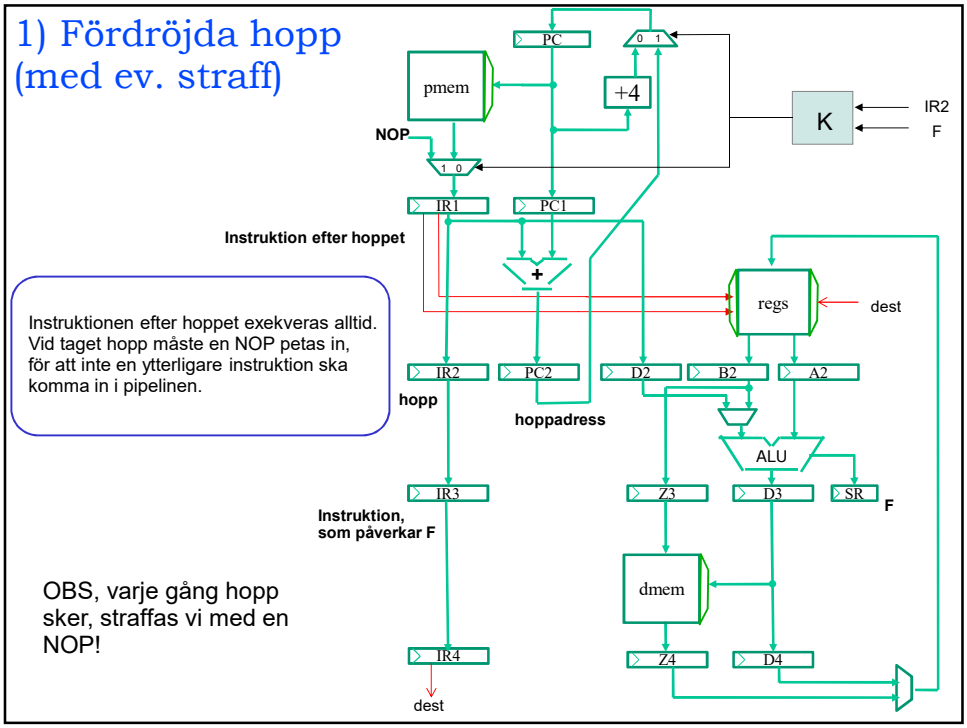
1

Så här långt är pipelining enkelt!
Vi har sett till 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



2

1) Fördröjda hopp (med ev. straff)

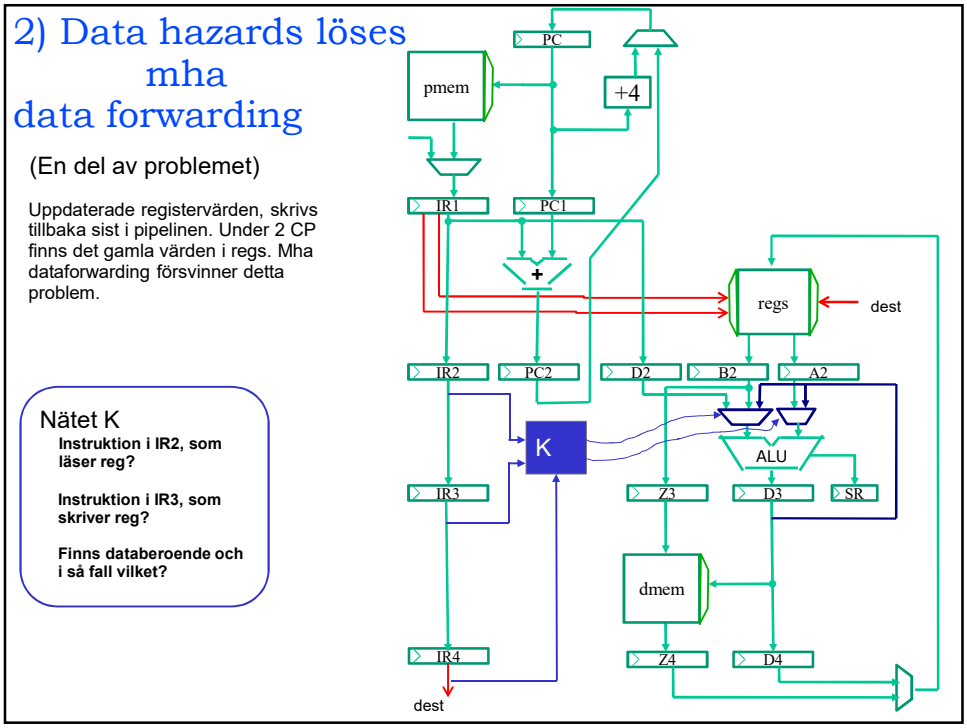


Instruktionen efter hoppet exekveras alltid. Vid taget hopp måste en NOP petas in, för att inte en ytterligare instruktion ska komma in i pipelinen.

OBS, varje gång hopp sker, straffas vi med en NOP!

3

2) Data hazards löses mha data forwarding



(En del av problemet)

Uppdaterade registervärden, skrivs tillbaka sist i pipelinen. Under 2 CP finns det gamla värden i regs. Mha dataforwarding försvinner detta problem.

Nätet K

Instruktion i IR2, som läser reg?

Instruktion i IR3, som skriver reg?

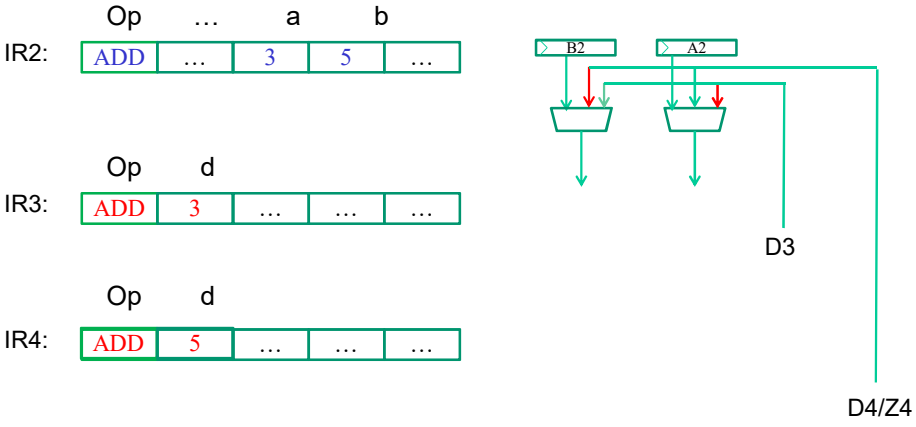
Finns databeroende och i så fall vilket?

4

Databeroende

Regs kan innehålla gamla data!
Var kan nya resultat finnas? D3,D4/Z4

Hur ska muxarna styras?



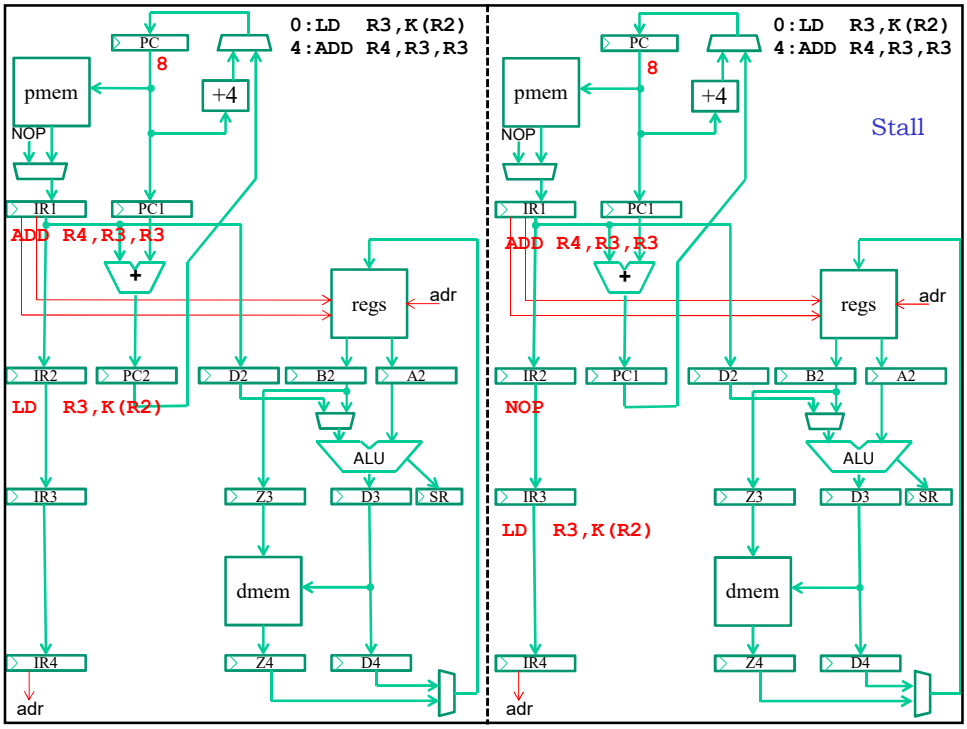
5

Problem 3: Stall

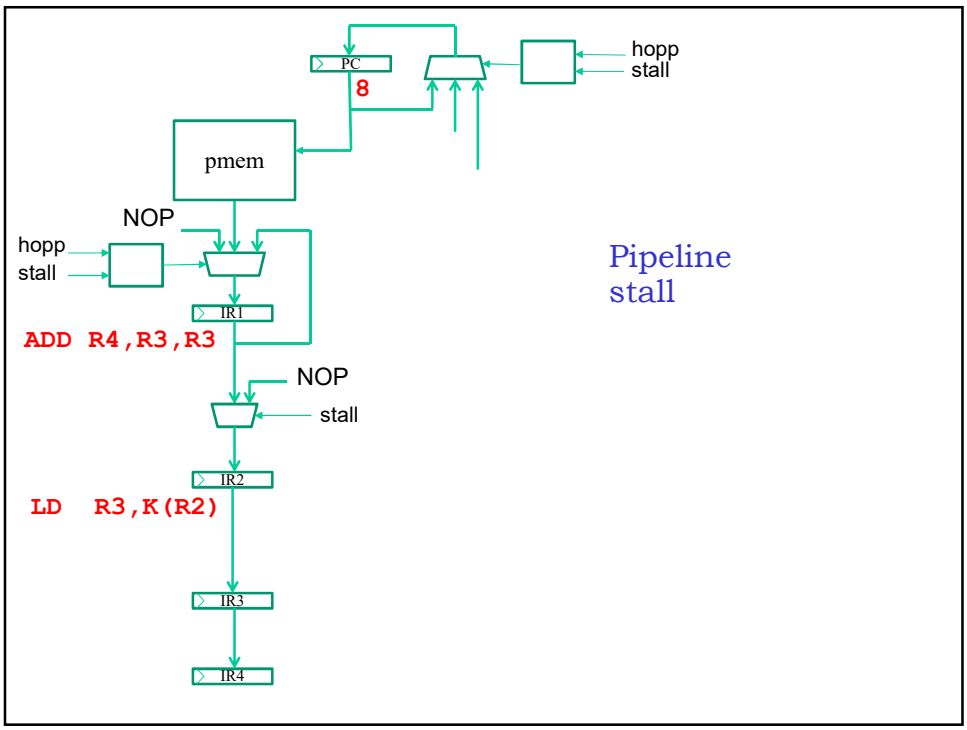
```
0:LD  R3,K(R2) ; läs från minnet
4:ADD R4,R3,R3 ;
```

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

6


0:LD R3,K(R2)
4:ADD R4,R3,R3
PC
8
pmem
+4
NOP
IR1
PC1
ADD R4,R3,R3
+
regs
adr
IR2
PC2
D2
B2
A2
LD R3,K(R2)
ALU
IR3
Z3
D3
SR
dmem
IR4
Z4
D4
adr
0:LD R3,K(R2)
4:ADD R4,R3,R3
Stall
NOP
NOP
LD R3,K(R2)

7


PC
8
hopp
stall
pmem
NOP
hopp
stall
IR1
ADD R4,R3,R3
NOP
stall
IR2
LD R3,K(R2)
IR3
IR4
Pipeline
stall

9

Observera att stall innebär prestandaförlust!

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

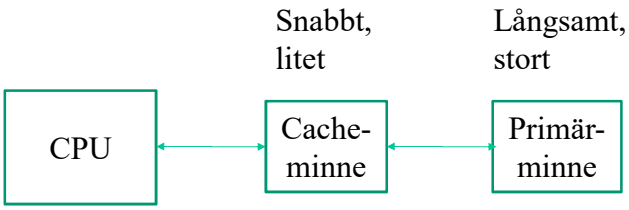
Vi kan lika gärna skriva

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

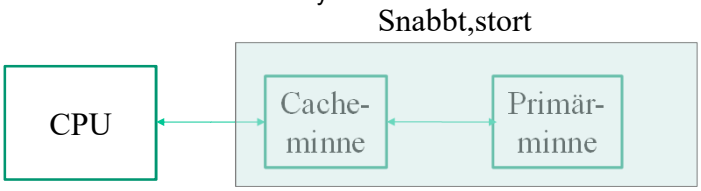
Det går kanske att flytta en annan instruktion till NOP:ens plats?
Ovanstående bor kanske i **heta loopen**, där varje klockcykel är viktig.

10

Cacheminnen



- CPUn 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.



11

Lönar sig cacheminne i vår FPGA?

I FPGA-n finns sammanlagt 64 kB sk block RAM.

Det räcker inte med dessa, så **nej**!

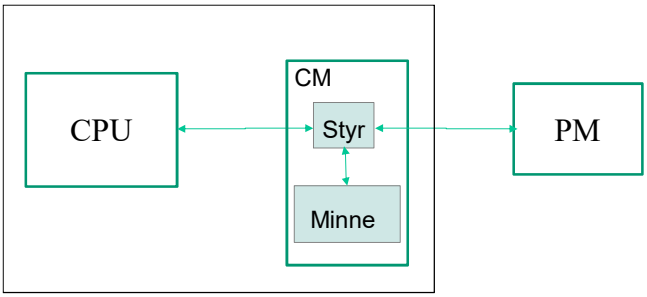
Utanför FPGA-n finns externt minne (flera MB) men med accesstiden 7 CK.

Det är stort nog, så förvisso **ja**!

Ett CM måste i FPGAn byggas med block RAM för att bli tillräckligt snabbt.

13

Cacheminnen



- När datorn startar är cacheminnet 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**.

14

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

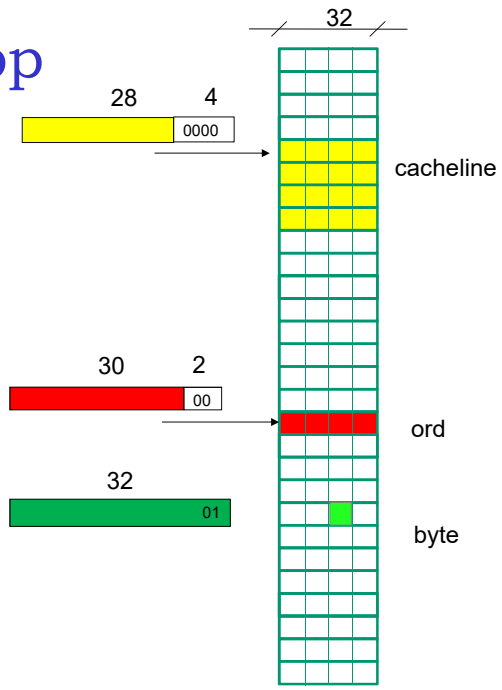
15

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



16

Enkelt cacheminne

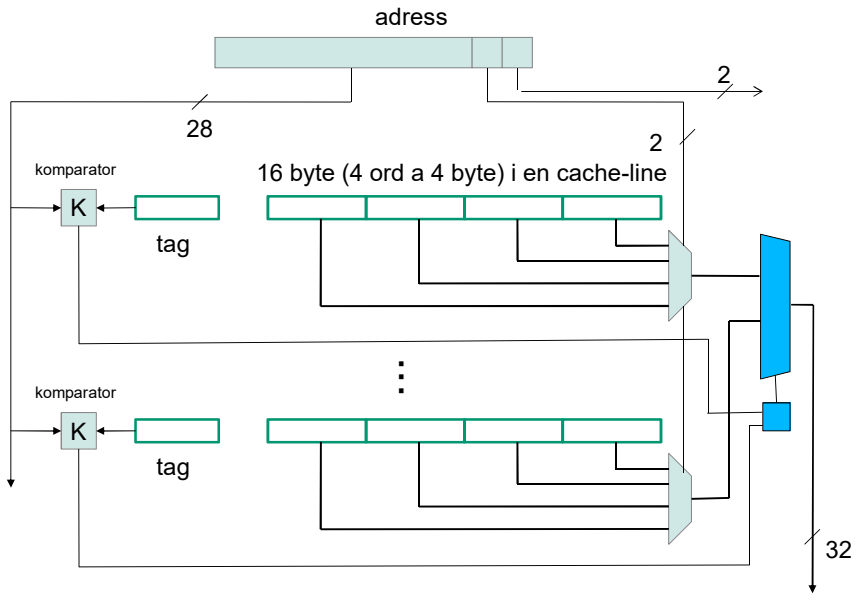
En styrenhet gör följande:

Vid **cachemiss:** Läs en hel cacheline.
Placera de 4 orden på lämplig rad i CM
Placera de 28 MSB i adressen i taggen
(annars vet man ju inte var cachelinen kommer ifrån)

Vid access: Jämför de 28 MSB i adressen samtidigt med alla taggar. Finns sökt data i cachen har vi en **cacheträff** annars cachemiss.

17

Associativt minne



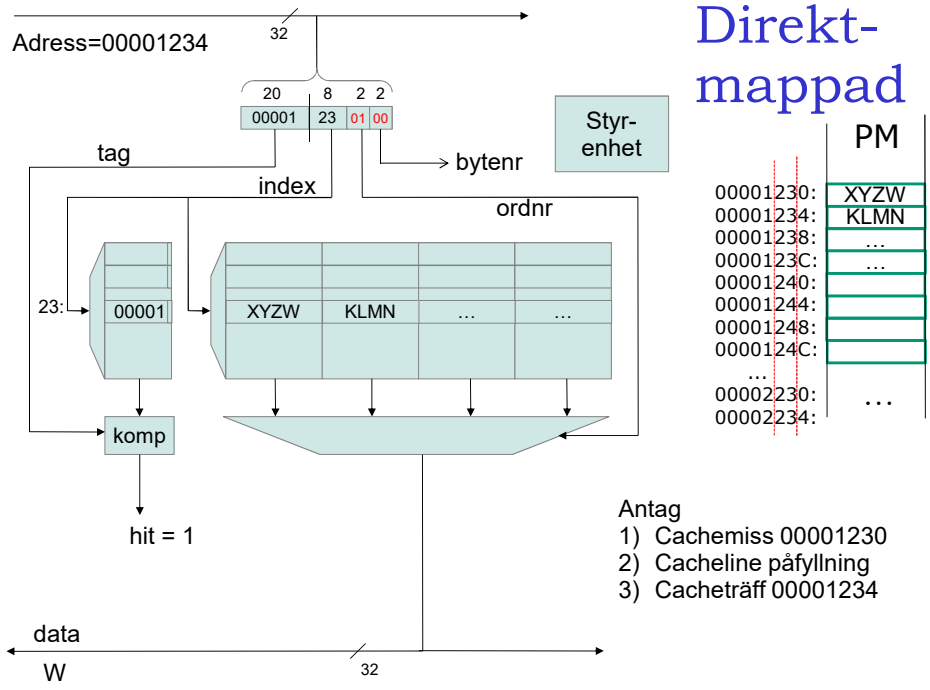
18

Cacheminnen

- 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

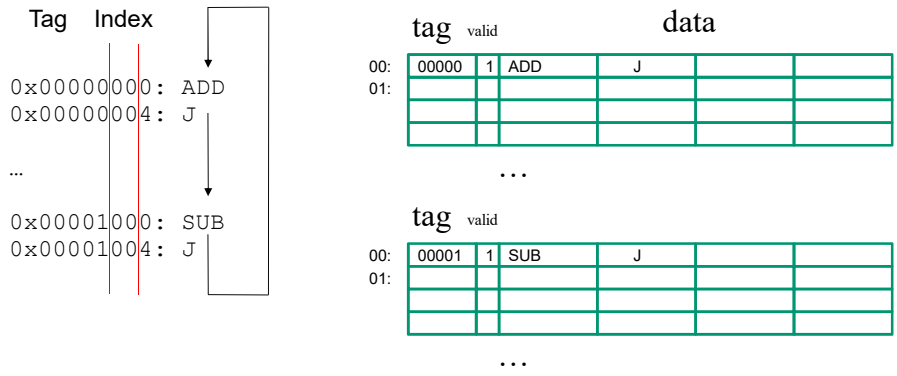
- En sk **direct mapped cache** har endast 1 komparator
- Data kan **inte** placeras var som helst!

19



24

Ett enkelt exempel?



Fastän cachen för övrigt är tom, får vi cachemiss hela tiden! (för ADD och SUB)
Detta är nackdelen med en 1-vägs direktmappad cache.

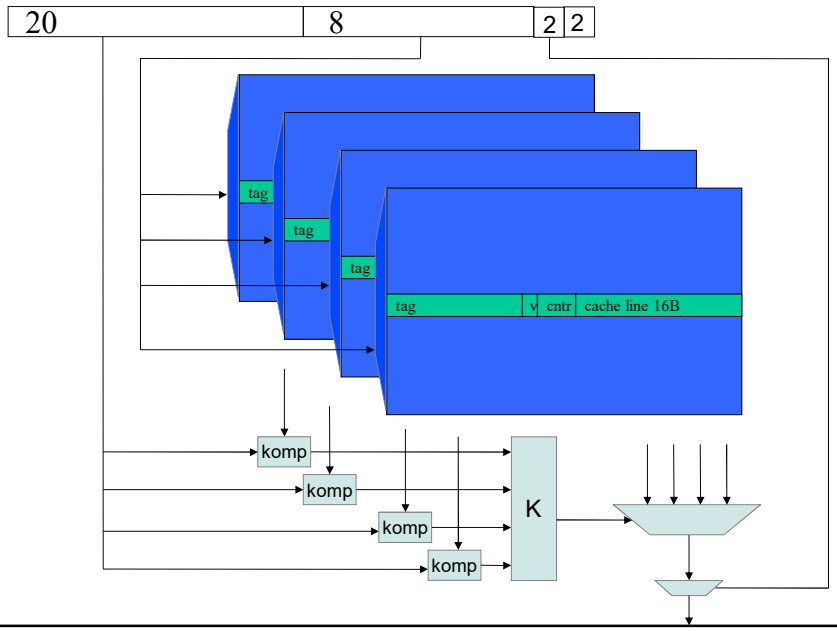
25

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.

26

4-vägs cache, 16 kB



27

Cacheminnen

- När man kört en stund, så är cachen full
- Vid en cachemiss, så ska ju cachen uppdateras, och ett data skrivas 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)

28

Cacheminnen

Mycket vanligt är att ha separata cacheminnen för instruktioner/data

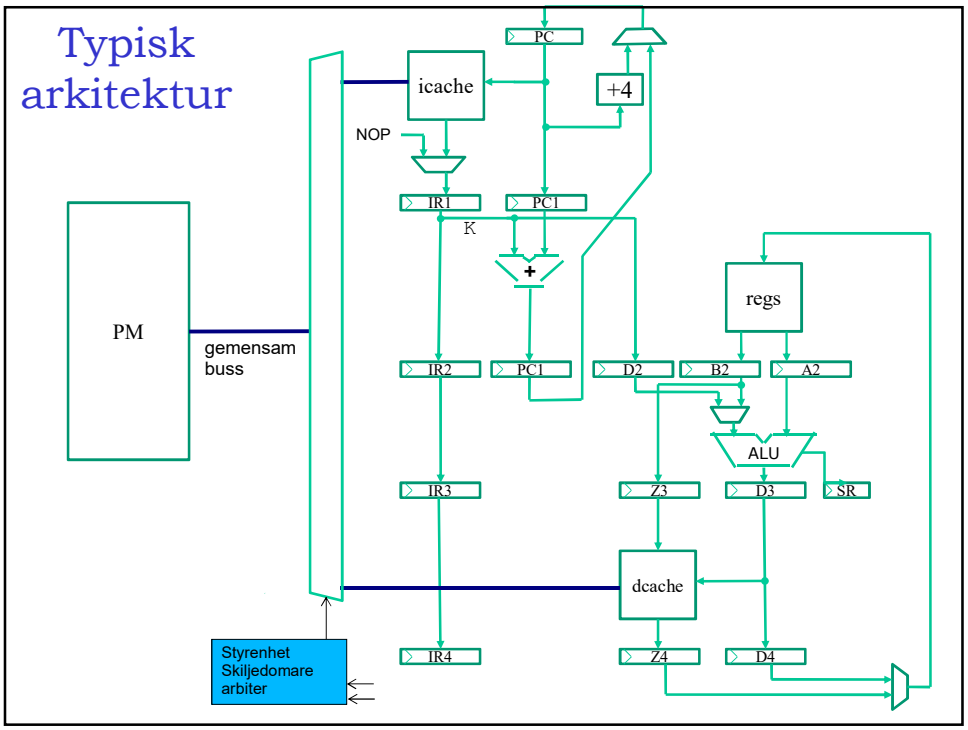
För datacachen tillkommer då vad som ska ske vid skrivning:
Write-thru: skriv till CM och PM (PrimärMinne)
alltså CM,PM uppdateras samtidigt

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.

29

Typisk arkitektur



30

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.

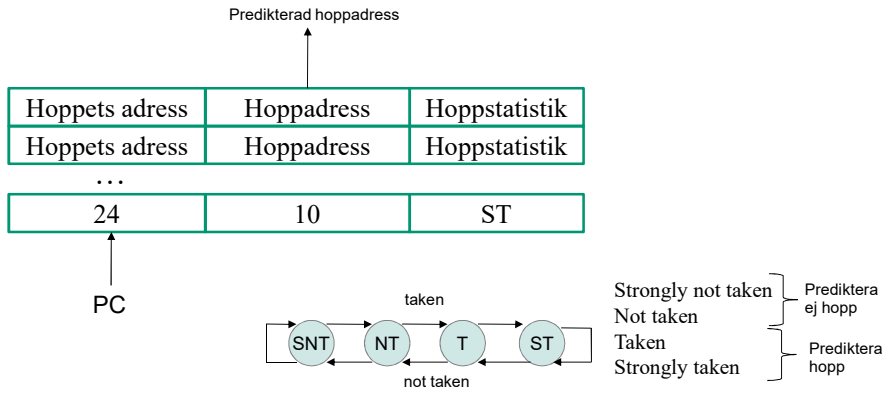
31

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

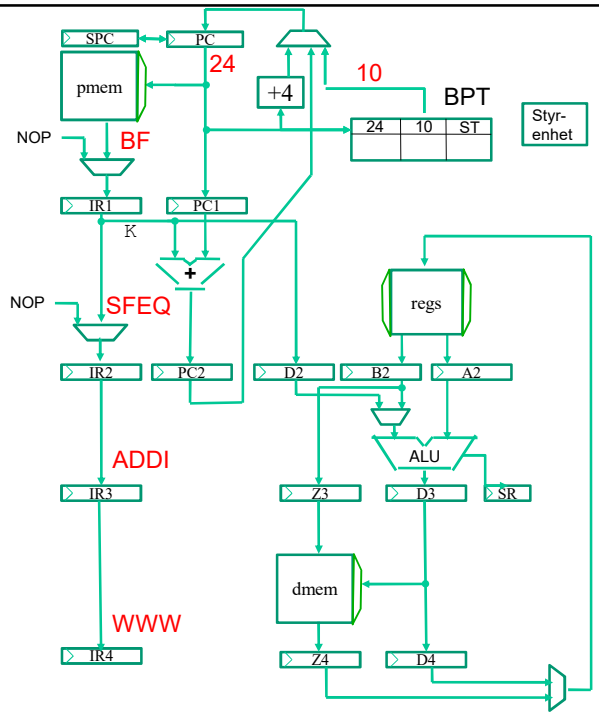


37

BPT (skiss)

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

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

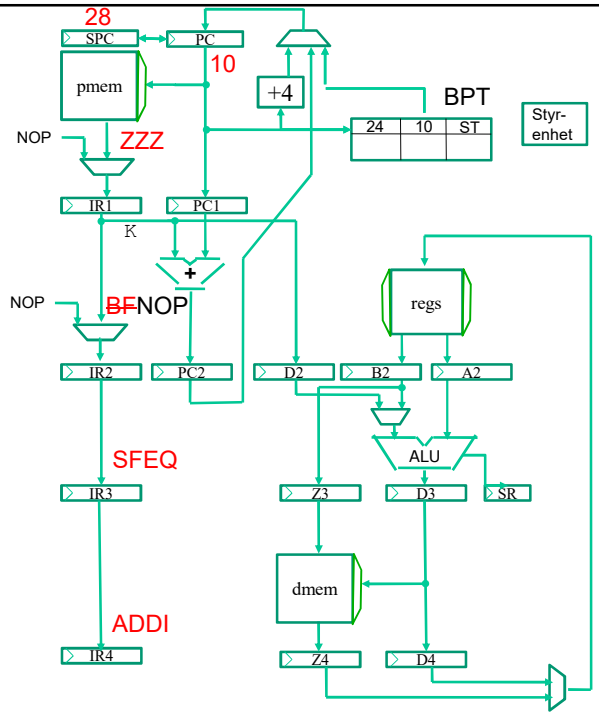


38

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

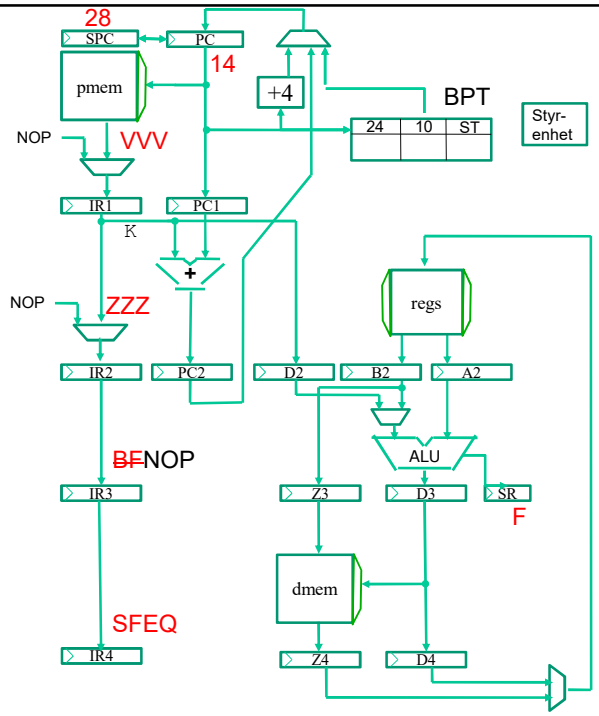


39

BPT (skiss)

Nu vet vi om hoppet faktiskt ska tas. Om det ska det så är allt OK!

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

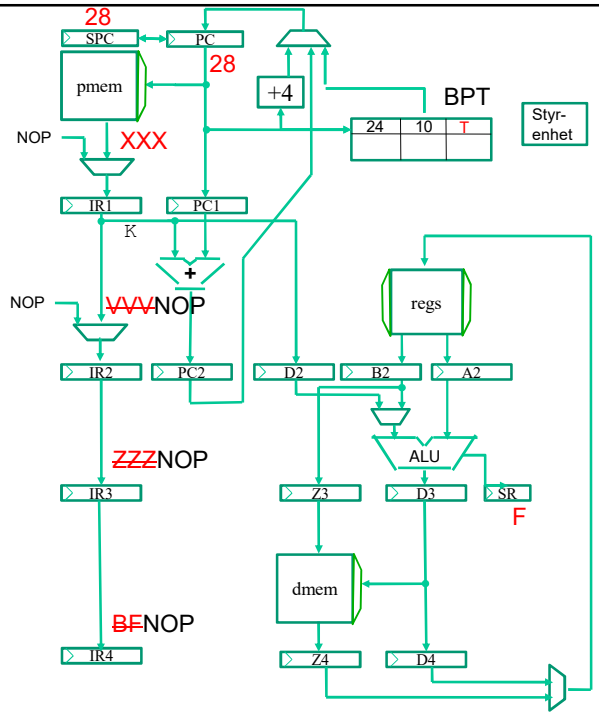


40

BPT (skiss)

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

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



41

```
// 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);
}
```

Benchmark

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

44

Design of a baseband processor

Eric Tell & Anders Nilsson
Coresonic (uppköpt av Mediatek)

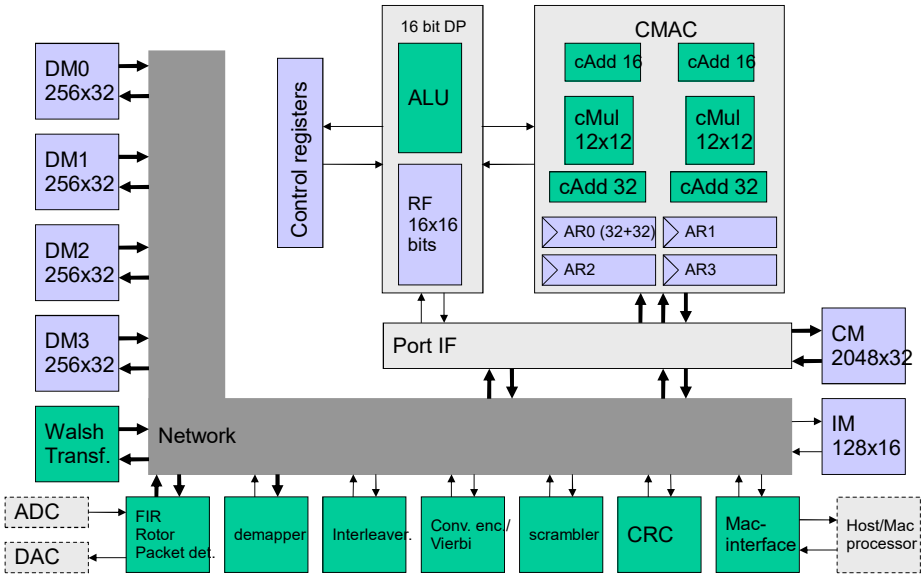
Ska klara så många standarder som möjligt

WLAN, 3G, DVB-T, …

Använder en kombination av
mjukvara och hårdvara (acceleratorer)

45

Overview of BBP1

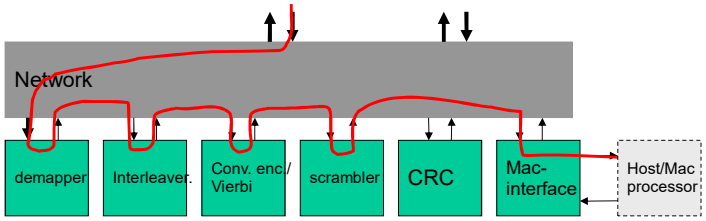


46

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!



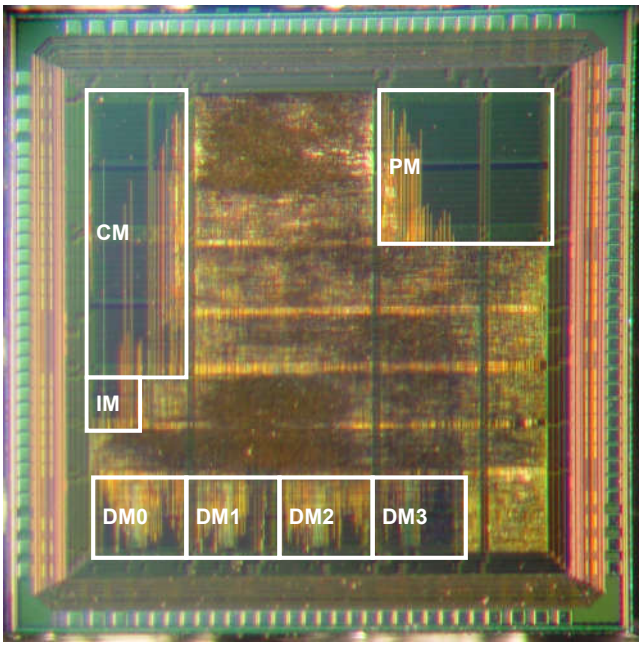
47

Chip Features:

Process	0.18 µm CMOS
Core voltage	1.8 V
I/O voltage	3.3 V
Chip area	5.0 mm^2
Core area	2.9 mm^2
Logic area	1.9 mm^2
Mem. area	1.0 mm^2
Package	144 pin fpBGA

Measured Max Frequency:
220 MHz

Measured Core Power
@160 MHz:
Idle 44 mW
11a Rx burst 126 mW



48