

TSEA28 Datorteknik Y (och U)

Föreläsning 14

Kent Palmkvist, ISY



Dagens föreläsning

- Cache
 - Princip
 - Exempel
 - Olika typer

Problem med högre klockfrekvenser

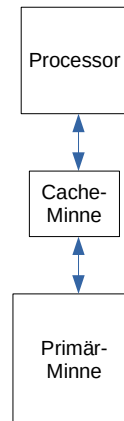
- Alla delar i datorn är inte lika snabba
 - Register skickar fort ut värden på bussen
 - ALU går olika fort beroende på operation
 - Shift, AND, OR etc. går fort
 - Add/sub går långsammare
 - Multiplikation ännu långsammare
- Minnen är långsamma
 - Större minnen långsammare än mindre
- Fysisk gräns: ljusets hastighet
 - 10 GHz motsvarar 100 ps/klockcykel = 3 cm (vakum!)
 - Långsammare på chip pga material

Minnen (program och datamminne)

- Utvecklingen av läshastighet för minne går långsammare än för processorn
 - Ca 20-50% bättre processorprestanda per år (avstannande)
 - Ca 10% bättre minnesprestanda per år
 - Större och större skillnad, processorn måste oftast vänta på minnet
- Olika hastighet och pris för olika minnen
 - Finns många typer
 - DRAM billiga per bit, långsamma och komplicerade att styra

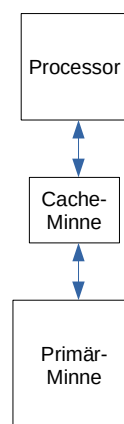
Hastighet minne vs cpu

- CPU blir allt snabbare
 - Ex: Raspberry pi: CPU går i 1 Ghz
- Stora minnen inte snabba
 - Register, små minnen (< 100KB) i CPU: läses/skrives på 1 klockcykel (< 1 ns)
 - Små minnen nära processorn (<10MB): ett fåtal klockcykler (5-10ns)
 - Stora minnen (> 1GB) har stor latency (> 100 ns)
- Använd ett litet snabbt minne (cacheminne)
 - Ungefär lika snabbt som resten av processorn
 - Slipper vänta på läsning/skrivning till lilla minnet
- Raspberry pi exempel (ARM)
 - 3 cykler för ett litet minne (SRAM)
 - 56 cykler för ett stort minne (DRAM)



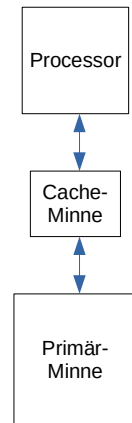
Snabbare minnesaccess

- Använd ett litet snabbt minne (cacheminne)
 - Ungefär lika snabbt som resten av processorn
 - Slipper vänta på läsning/skrivning till lilla minnet
- När cacheminnet fullt/saknar data skickas data till/från långsammare billigare minne (primärminnet).
 - Behöver inte spara mellanresultat i primärminnet
 - Skriver endast slutresultatet till primärminnet
 - Gör inte så mycket om primärminnets latens är stor
- Jfr program/data på en hårddisk
 - Arbeta med programmet mot kopia av fil i minnet, spara sedan på disk



Uppdatering av cacheminnet

- Kan hanteras manuellt/programmeringsmässigt
 - Görs mha operativsystem när det gäller hårddisk - primärminne
 - primärminnet fungerar som cacheminne för hårddisk
- Vill automatisera kopiering
 - Processor anger adress att läsa eller skriva
 - Cacheminne (liten snabbt minne) svarar med data om kopia från långsamma minnet finns i cacheminnet
 - Långsamma minnet läses om data inte finns i cacheminnet, kopia sparas i cacheminnet
 - Cacheminnet behöver komma ihåg vilken adress i primärminnet varje minnescell är en kopia av



Cacheminnets funktion

- Vanliga program har lokalitet
 - Programmet läser och skriver data i vissa delar av minnet
 - De delar av minnet som inte används behöver inte kopieras till cacheminnet
 - Samma adresser läses ofta flera gånger
- Dela upp primärminnet i små block (cachelines)
 - Hantera blocket som en enhet
 - Läs alla eller skriv alla värden i ett block
 - Block överlappar aldrig varandra
 - Block startar på adress i primärminnet med alla bytepositionsbitar i adressen = 0

Primärminne	
Adress	Data
0x5000	0x1234 0x5678 0xabcd ...
0x5010	0xbeef 0xbeef 0xbeef ...
0x5020	0x0000 0x1111 0x2222 ...

Adress från processor	
tag	Byte-position i cacheline (om 16 byte lång)
31 ... 43 .. 0	

Cacheminnet			
index	st	tag	Dataarea
0	st	tag	Dataarea
1	st	tag	Dataarea
2	st	tag	Dataarea

Cacheminnets struktur

- Multipla cachelinjer (cachelines)
 - Innehåller kopia av block från primärminnet
- Varje cachelinje i cacheminnet består av
 - Dataarea (kopia av primärminnet)
 - Vanlig storlek 16 eller 32 bytes (128 eller 256 bit per cacheline)
 - Märkarea (tag) anger adressen till kopians original i primärminnet
 - Statusbitar (st). Giltigt innehåll i dataarea, ändrad data etc.

Primärminne	
Adress	Data
0x5000	0x1234 0x5678 0xabcd ...
0x5010	0xbeef 0xbeef 0xbeef ...
0x5020	0x0000 0x1111 0x2222 ...

Adress från processor		
31	...	4 3 .. 0
tag		Byte-position i cacheline (om 16 byte lång)

Cacheminnet			
index	st	tag	Dataarea
0	st	tag	Dataarea
1	st	tag	Dataarea
2	st	tag	Dataarea

Exempel på fördel med datacache

- Enkelt exempel
 - Program: Summera 16-bitars 2-komplementtal


```

mov r0,#50
mov r1,#0x5000
mov r2,#0
loop:
ldrsh r3,[r1],#2
addr2,r2,r3
subs r0,#1
bneloop
          
```

Primärminne	
Adress	Data
0x5000	0x1234 0x5678 0xabcd ...
0x5010	0xbeef 0xbeef 0xbeef ...
0x5020	0x0000 0x1111 0x2222 ...

- Antag minnesaccess till primärminne tar 50 klockcykler
- Antag cacheminnet läses på 1 klockcykel
- Instruktionerna ligger redan i Cacheminnet (fokus på dataaccess)
 - Hela programmet läses in till cacheminnet när 1:a instruktionen ska startas

Exempel utan datacache

- Räkna samman antal klockcykler som krävs för alla instruktioner

```
-> mov r0,#50
    mov r1,#0x5000
    mov r2,#0
loop: ldrsh r3,[r1],#2
      add r2,r2,r3
      subs r0,#1
      bne loop
```

1. Normal instruktion
1 klockcykel
Sammanlagt: 1 klockcykel

```
-> mov r0,#50
    mov r1,#0x5000
    mov r2,#0
loop: ldrsh r3,[r1],#2
      add r2,r2,r3
      subs r0,#1
      bne loop
```

2. Normal instruktion
1 klockcykel
Sammanlagt: 2 klockcykler

```
-> mov r0,#50
    mov r1,#0x5000
    mov r2,#0
loop: ldrsh r3,[r1],#2
      add r2,r2,r3
      subs r0,#1
      bne loop
```

3. Normal instruktion
1 klockcykel
Sammanlagt: 3 klockcykler

```
mov r0,#50
mov r1,#0x5000
mov r2,#0
loop:
-> ldrsh r3,[r1],#2
   add r2,r2,r3
   subs r0,#1
   bne loop
```

4. Läsning i primärminnet
Adress 0x5000
50 klockcykler
Sammanlagt: 53 klockcykler

Exempel utan datacache, forts.

- Räkna samman antal klockcykler som krävs för alla instruktioner (steg 5 till 8)

```
mov r0,#50
mov r1,#0x5000
mov r2,#0
loop: ldrsh r3,[r1],#2
-> add r2,r2,r3
    subs r0,#1
    bne loop
```

5. Normal instruktion
1 klockcykel
Sammanlagt: 54 klockcykler

```
mov r0,#50
mov r1,#0x5000
mov r2,#0
loop: ldrsh r3,[r1],#2
-> add r2,r2,r3
    subs r0,#1
    bne loop
```

6. Normal instruktion
1 klockcykel
Sammanlagt: 55 klockcykler

```
mov r0,#50
mov r1,#0x5000
mov r2,#0
loop: ldrsh r3,[r1],#2
-> add r2,r2,r3
    subs r0,#1
    bne loop
```

7. Normal instruktion
1 klockcykel
Sammanlagt: 56 klockcykler

```
mov r0,#50
mov r1,#0x5000
mov r2,#0
loop:
-> ldrsh r3,[r1],#2
   add r2,r2,r3
   subs r0,#1
   bne loop
```

8. Läsning i primärminnet
Adress 0x5002
50 klockcykler
Sammanlagt: 106 klockcykler

Exempel utan datacache, totalt

- Fortsätt med alla 50 additionerna
 - Totalt $3+50*4$ instruktioner utförs
 - 203 instruktioner
 - Varje instruktion tar 1 klockcykel att hämta om ingen dataminnesaccess
 - Totalt 50 dataminnesläsningar
 - Dessa instruktioner tar 50 klockcykler var
 - Totalt antal klockcykler (utan cache för data)
 - $3*1+50(3*1+1*50) = 2653$ klockcykler
 - 2500 klockcykler i ldrsh instruktionen, 153 klockcykler för övriga instruktioner

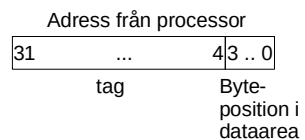
```

mov r0,#50
mov r1,#0x5000
mov r2,#0
loop:
ldrsh r3,[r1],#2
add r2,r2,r3
subs r0,#1
bne loop

```

Exempel med datacache

- Antagande om cacheegenskaper
 - Cacheline 16 bytes lång (antal byte data per cacheline)
 - Cache är 4096 bytes stor
 - $4096/16 = 256$ cachelines i cacheminnet
 - Fullt associativt
 - Vilken address som helst kan ligga i varje cacheline
 - Address från processorn 32 bitar lång
 - Tag måste vara $32-4 = 28$ bitar
- Antagande om minne
 - När 16 värden (en cacheline) hämtas som ett block behövs 60 klockcykler (burstaccess)



```

mov r0,#50
mov r1,#0x5000
mov r2,#0
loop:
ldrsh r3,[r1],#2
add r2,r2,r3
subs r2,#1
bne loop

```

index	status	tag	dataarea
0	tom	0x0000	0x0000 0x0000 0x0000 ...
1	tom	0x0000	0x0000 0x0000 0x0000 ...
2	tom	0x0000	0x0000 0x0000 0x0000 ...
3	tom	0x0000	0x0000 0x0000 0x0000 ...

Exempel med datacache, forts.

- Räkna samman antal klockcykler som krävs för alla instruktioner (steg 1-3 precis som förra gången)
 - Steg 4 lite annorlunda

```
-> mov r0,#50
    mov r1,#0x5000
    mov r2,#0
loop: ldrsh r3,[r1],#2
      add r2,r2,r3
      subs r0,#1
      bne loop
```

1. Normal instruktion
1 klockcykel
Sammanlagt: 1 klockcykel

```
-> mov r0,#50
    mov r1,#0x5000
    mov r2,#0
loop: ldrsh r3,[r1],#2
      add r2,r2,r3
      subs r0,#1
      bne loop
```

2. Normal instruktion
1 klockcykel
Sammanlagt: 2 klockcykler

```
-> mov r0,#50
    mov r1,#0x5000
    mov r2,#0
loop: ldrsh r3,[r1],#2
      add r2,r2,r3
      subs r0,#1
      bne loop
```

3. Normal instruktion
1 klockcykel
Sammanlagt: 3 klockcykler

```
mov r0,#50
mov r1,#0x5000
mov r2,#0
loop:
-> ldrsh r3,[r1],#2
   add r2,r2,r3
   subs r0,#1
   bne loop
```

4. Läsning i primärminnet
Adress 0x5000
Finns den i cacheminnet?

Exempel med datacache, forts.

- Läsning address 0x5000
 - Finns inte i cacheminnet
 - Kräver läsning av primärminnet, 60 klockcykler
 - Kopia läggs i cache
 - Inklusive efterföljande 15 byte
 - Tag är address minus bitar som indexerar i dataarean
 - 4 sista bitarna väljer byte i dataarea

Primärminne	
0x5000	0x1234 0x5678 0xabcd ...
0x5010	0xbeef 0xbeef 0xbeef ...
0x5020	0x0000 0x1111 0x2222 ...

```
mov r0,#50
mov r1,#0x5000
mov r2,#0
loop:
-> ldrsh r3,[r1],#2
   add r2,r2,r3
   subs r0,#1
   bne loop
```

4. Cachemiss,
60 klockcykler
Sammanlagt: 63 klockcykler

index	status	tag	dataarea
0	valid	0x500	0x1234 0x5678 0xabcd ...
1	tom	0x0000	0x0000 0x0000 0x0000 ...
2	tom	0x0000	0x0000 0x0000 0x0000 ...
3	tom	0x0000	0x0000 0x0000 0x0000 ...

Exempel med datacache, forts.

```

mov r0,#50
mov r1,#0x5000
mov r2,#0
loop:
-> ldrsh r3,[r1],#2
    add r2,r2,r3
    subs r0,#1
    bne loop

```

5. Normal instruktion
1 klockcykel
Sammanlagt: 64 klockcykler

```

mov r0,#50
mov r1,#0x5000
mov r2,#0
loop:
-> ldrsh r3,[r1],#2
    add r2,r2,r3
    subs r0,#1
    bne loop

```

6. Normal instruktion
1 klockcykel
Sammanlagt: 65 klockcykler

```

mov r0,#50
mov r1,#0x5000
mov r2,#0
loop:
-> ldrsh r3,[r1],#2
    add r2,r2,r3
    subs r0,#1
    bne loop

```

7. Normal instruktion
1 klockcykel
Sammanlagt: 66 klockcykler

index	status	tag	dataarea
0	valid	0x500	0x1234 0x5678 0xabcd ...
1	tom	0x0000	0x0000 0x0000 0x0000 ...
2	tom	0x0000	0x0000 0x0000 0x0000 ...
3	tom	0x0000	0x0000 0x0000 0x0000 ...

Exempel med datacache, forts.

- Läsning address 0x5002
 - Finns i cacheminnet
 - Kräver ingen läsning av primärminnet
 - Kostnad 1 klockcykel
 - 4 lägsta bitarna väljer byte i cacheline (2)

Primärminne	
0x5000	0x1234 0x5678 0xabcd ...
0x5010	0xbeef 0xbeef 0xbeef ...
0x5020	0x0000 0x1111 0x2222 ...

```

mov r0,#50
mov r1,#0x5000
mov r2,#0
loop:
-> ldrsh r3,[r1],#2
    add r2,r2,r3
    subs r0,#1
    bne loop

```

8. Cacheträff,
1 klockcykler
Sammanlagt: 67 klockcykler

index	status	tag	dataarea
0	valid	0x500	0x1234 0x5678 0xabcd ...
1	tom	0x0000	0x0000 0x0000 0x0000 ...
2	tom	0x0000	0x0000 0x0000 0x0000 ...
3	tom	0x0000	0x0000 0x0000 0x0000 ...

Exempel med datacache, forts.

- Cacheträffar fås för adresser 0x5004 - 0x500E
 - Kräver ingen läsning av primärminnet
 - Sammanlagt $3*1+1*60+7*1+8*3 = 94$ klockcykler
- Ny cachemiss för läsning av adress 0x5010
 - Läs 16 bytes med start från adress 0x5010 till dataarea rad 2 i cache
 - Uppdatera tag till adress (borträknat 4 LSB-bitar)
 - Tar 60 klockcykler

Primärminne	
0x5000	0x1234 0x5678 0xabcd ...
0x5010	0xbeef 0xbeef 0xbeef ...
0x5020	0x0000 0x1111 0x2222 ...

```

mov r0,#50
mov r1,#0x5000
mov r2,#0
loop:
-> ldrsh r3,[r1],#2
   add r2,r2,r3
   subs r0,#1
   bne loop

```

index	status	tag	dataarea
0	valid	0x500	0x1234 0x5678 0xabcd ...
1	valid	0x501	0xbeef 0xbeef 0xbeef ...
2	tom	0x0000	0x0000 0x0000 0x0000 ...
3	tom	0x0000	0x0000 0x0000 0x0000 ...

36. Cacheträff,
1 klockcykler
Sammanlagt: 154 klockcykler

Exempel med datacache, totalt

- Efter 203 instruktioner (samma som tidigare)
 - 7 cachemissar ($7*16 = 112$ byte = 56 ord)
 - Sammanlagt $3*1+50*3*1+(50-7)*1+7*60 = 616$ klockcykler
 - Jämför utan cache: 2653 klockcykler
 - Cache ger 4 gånger snabbare exekvering!

```

mov r0,#50
mov r1,#0x5000
mov r2,#0
loop:
ldrsh r3,[r1],#2
add r2,r2,r3
subs r0,#1
bne loop

```

Viktig kommentar ang. ordlängder

- I exemplet råkar bytepositionen i cacheline vara 4 bitar lång
 - Orsak: 16 byte per cacheline
 - Stämmer bra med antal bitar i en hexadecimal siffra
 - Valt för att lättare se hur tag och minnesadress hänger ihop
- Kan vara andra värden! Ex: 5 bitar byteposition (32 byte/cacheline)
 - Tänk binär representation! $0x5034 = 0101\ 0000\ 0011\ 0100$
 - Adressen $0x5034$ delas då upp:
 - Tag: $0101\ 0000\ 001 = 010\ 1000\ 0001 = 0x281$
 - Byteposition: $1\ 0100 = 0x14$

Mer komplicerat exempel

- Summera värden två vektorer placerade på adress $0x5000$ och $0x6000$, placera svar i vektor på adress $0x6000$
 - Vektorlängd: 25, elementstorlek halvord (2 byte)
 - Ungefärligt program


```

mov r0,#0x5000
mov r1,#0x6000
mov r2,#0
loop:
ldrshr3,[r0,r2]
ldrshr4,[r1,r2]
add r4,r4,r3
strh r4,[r1,r2]
add r2,r2,#2
cmp r2,#50
bne loop
          
```

Primärminne	
0x5000	0x1234 0x5678 0xabcd ...
0x5010	0xbeef 0xbeef 0xbeef ...
0x5020	0x0000 0x1111 0x2222 ...
0x6000	0x4321 0x1111 0x0001 ...
0x6010	0x1100 0x0000 0x0115 ...
0x6020	0xffff 0xffff 0xffff ...

Cacheminne			
index	status	tag	dataarea
0	tom
1	tom
2	tom
3	tom
4	tom

Mer komplicerat exempel, forts.

- Första minnesåtkomst, cachemiss på adress 0x5000
- Ladda in adress 0x5000 till första lediga cacheline (index 0)
 - Läs 16 byte till dataarea
 - Sätt tag till adress förutom de sista 4 bitarna

```
loop:
  ldrsh r3,[r0,r2]
  ldrsh r4,[r1,r2]
  add r4,r4,r3
  strh r4,[r1,r2]
  add r2,r2,#2
  cmp r2,#50
  bne loop
```

Primärminne	
0x5000	0x1234 0x5678 0xabcd ...
0x5010	0xbeef 0xbeef 0xbeef ...
0x5020	0x0000 0x1111 0x2222 ...
0x6000	0x4321 0x1111 0x0001 ...
0x6010	0x1100 0x0000 0x0115 ...
0x6020	0xffff 0xffff 0xffff ...

Cacheminne			
index	status	tag	dataarea
0	valid	0x500	0x1234 0x5678 0xabcd ...
1	tom
2	tom
3	tom
4	tom

Mer komplicerat exempel, forts.

- Andra minnesåtkomst, cachemiss på adress 0x6000
 - 2 cachemissar totalt
- Ladda in adress 0x6000 till första lediga cacheline (index 1)
 - Läs 16 byte till dataarea
 - Sätt tag till adress förutom de sista 4 bitarna

```
loop:
  ldrsh r3,[r0,r2]
  ldrsh r4,[r1,r2]
  add r4,r4,r3
  strh r4,[r1,r2]
  add r2,r2,#2
  cmp r2,#50
  bne loop
```

Primärminne	
0x5000	0x1234 0x5678 0xabcd ...
0x5010	0xbeef 0xbeef 0xbeef ...
0x5020	0x0000 0x1111 0x2222 ...
0x6000	0x4321 0x1111 0x0001 ...
0x6010	0x1100 0x0000 0x0115 ...
0x6020	0xffff 0xffff 0xffff ...

Cacheminne			
index	status	tag	dataarea
0	valid	0x500	0x1234 0x5678 0xabcd ...
1	valid	0x600	0x4321 0x1111 0x0001 ...
2	tom
3	tom
4	tom

Mer komplicerat exempel, forts.

- Tredje minnesåtkomst, skrivning på adress 0x6000
 - Ingen cachemiss (uppdatera inte primärminnet)
 - 2 cachemissar totalt
- Spara nya datat till motsvarande plats i cacheminnet
 - Värdet i d1 = $0x1234 + 0x4321 = 0x5555$, indikera ändrat värde i cacheline (dirty)
 - Index 0, 1:a ordet i dataarean (byte 0 och 1)

loop:

```
ldrsh r3,[r0,r2]
ldrsh r4,[r1,r2]
add r4,r4,r3
strh r4,[r1,r2]
add r2,r2,#2
cmp r2,#50
bne loop
```

Primärminne

0x5000	0x1234 0x5678 0xabcd ...
0x5010	0xbeef 0xbeef 0xbeef ...
0x5020	0x0000 0x1111 0x2222 ...
0x6000	0x4321 0x1111 0x0001 ...
0x6010	0x1100 0x0000 0x0115 ...
0x6020	0xffff 0xffff 0xffff ...

Cacheminne

index	status	tag	dataarea
0	valid	0x500	0x1234 0x5678 0xabcd ...
1	dirty	0x600	0x5555 0x1111 0x0001 ...
2	tom
3	tom
4	tom

Mer komplicerat exempel, forts.

- Fjärde minnesåtkomst, läsning på adress 0x5002
 - Cacheträff, data finns i cache index 0, 2:a ordet
 - 2 cachemissar totalt
- Femte minnesåtkomst, läsning på adress 0x6002
 - Cacheträff, data finns i cache index 1, 2:a ordet
 - 2 cachemissar totalt

loop:

```
ldrsh r3,[r0,r2]
ldrsh r4,[r1,r2]
add r4,r4,r3
strh r4,[r1,r2]
add r2,r2,#2
cmp r2,#50
bne loop
```

Primärminne

0x5000	0x1234 0x5678 0xabcd ...
0x5010	0xbeef 0xbeef 0xbeef ...
0x5020	0x0000 0x1111 0x2222 ...
0x6000	0x4321 0x1111 0x0001 ...
0x6010	0x1100 0x0000 0x0115 ...
0x6020	0xffff 0xffff 0xffff ...

Cacheminne

index	status	tag	dataarea
0	valid	0x500	0x1234 0x5678 0xabcd ...
1	dirty	0x600	0x5555 0x1111 0x0001 ...
2	tom
3	tom
4	tom

Mer komplicerat exempel, forts.

- Sjätte minnesåtkomst, skrivning på adress 0x6002
 - Cacheträff, data skrivs till index 1, 3:e ordet
 - 2 cachemissar totalt
- Inga cachemissar för adresser till och med 0x500e respektive 0x600e
 - 2 cachemissar totalt

```
loop:
  ldrsh r3,[r0,r2]
  ldrsh r4,[r1,r2]
  add r4,r4,r3
  strh r4,[r1,r2]
  add r2,r2,#2
  cmp r2,#50
  bne loop
```

Primärminne	
0x5000	0x1234 0x5678 0xabcd ...
0x5010	0xbeef 0xbeef 0xbeef ...
0x5020	0x0000 0x1111 0x2222 ...
0x6000	0x4321 0x1111 0x0001 ...
0x6010	0x1100 0x0000 0x0115 ...
0x6020	0xffff 0xffff 0xffff ...

Cacheminne			
index	status	tag	dataarea
0	valid	0x500	0x1234 0x5678 0xabcd ...
1	dirty	0x600	0x5555 0x6789 0xabce ...
2	tom
3	tom
4	tom

Mer komplicerat exempel, forts.

- 25:e minnesåtkomsten, Cachemiss på adress 0x5010
 - 3 cachemissar totalt
- Ladda in adress 0x5010 till första lediga cacheline (index 2)
 - Läs 16 byte till dataarea
 - Sätt tag till adress förutom de sista 4 bitarna

```
loop:
  ldrsh r3,[r0,r2]
  ldrsh r4,[r1,r2]
  add r4,r4,r3
  strh r4,[r1,r2]
  add r2,r2,#2
  cmp r2,#50
  bne loop
```

Primärminne	
0x5000	0x1234 0x5678 0xabcd ...
0x5010	0xbeef 0xbeef 0xbeef ...
0x5020	0x0000 0x1111 0x2222 ...
0x6000	0x4321 0x1111 0x0001 ...
0x6010	0x1100 0x0000 0x0115 ...
0x6020	0xffff 0xffff 0xffff ...

Cacheminne			
index	status	tag	dataarea
0	valid	0x500	0x1234 0x5678 0xabcd ...
1	dirty	0x600	0x5555 0x6789 0xabce ...
2	valid	0x501	0xbeef 0xbeef 0xbeef ...
3	tom
4	tom

Mer komplicerat exempel, forts.

- Fortsätter tills alla 25 halvord lästs från adress 0x5000-0x5064 respektive 0x6000-0x6064
- Totalt 14 cachemissar
- Alla skrivningar hamnar i cache
 - Vid någon senare tidpunkt töms cacheline och skrivs till primärminnet
 - Tex pga full cache eller speciell signal från processorn

loop:

```
ldrsh r3,[r0,r2]
ldrsh r4,[r1,r2]
add r4,r4,r3
strh r4,[r1,r2]
add r2,r2,#2
cmp r2,r2,#50
bne loop
```

Primärminne

0x5000	0x1234 0x5678 0xabcd ...
0x5010	0xbeef 0xbeef 0xbeef ...
0x5020	0x0000 0x1111 0x2222 ...
0x6000	0x4321 0x1111 0x0001 ...
0x6010	0x1100 0x0000 0x0115 ...
0x6020	0xffff 0xffff 0xffff ...

Cacheminne

index	status	tag	dataarea
0	valid	0x500	0x1234 0x5678 0xabcd ...
1	dirty	0x600	0x5555 0x6789 0xabce ...
2	valid	0x501	0xbeef 0xbeef 0xbeef ...
3	dirty	0x601	0xcfef 0xbeef 0xbef0 ...
4	valid	0x502	0x0000 0x1111 0x2222 ...

Fullt associativ cachestruktur

- Fördel
 - Enkel att använda (och förstå?)
 - Alla cachelines kan placeras var som helst i cacheminnet
 - Använder hela cacheminnet
- Nackdel
 - Dyr
 - Varje cacheline innehåller jämförelsehårdvara för att jämföra tag och adressens tag-del
 - Tag är lång (antal bitar i adress - 4 i exemplet ovan)
 - Långsam
 - Resultat från alla jämförelser ska samlas ihop innan beslut om cache-hit eller cache-miss

Adress från processor

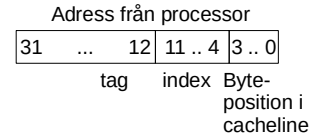
31	...	4	3 .. 0
		tag	Byte- position i cacheline

Cacheminne

index	st	tag	Dataarea
0	st	tag	Dataarea
1	st	tag	Dataarea
2	st	tag	Dataarea

Direktadresserad cache

- Varje cacheline från minnet kan bara placeras på en plats i cache
 - Cache < Minne => flera olika minnesadresser placeras på samma plats i cache
- Del av adressen används för att bestämma index (dvs rad) i cache
 - Bitmönstret sparas inte i cache
- Resten adressen sparas i tag för att ange vilket minnesblock som finns i cache
 - Endast en jämförelse per minnesaccess: Adressens högsta bitar jämfört med lagrad tag
 - Enklare implementera, kan använda vanliga minnen
 - 1 jämförare för hela cache istället för 1 jämförare för varje cacheline



index		Cacheminne	
0	st	tag	Dataarea
1	st	tag	Dataarea
2	st	tag	Dataarea

Direktadresserad cache, exempel

- Samma uppgift som tidigare
 - 32 bitar adress
 - 20 bitar tag indikerar vilket minnesblock data kommer ifrån
 - 8 bitar index för att välja cacheline i cacheminnet
 - 4 KB minne, 16 byte/cacheline => 256 cacheline
 - 4 bitar indexering inom dataarean i varje cacheline
 - 16 byte per cacheline (dataarea)

loop:

```
ldrsh r3,[r0,r2]
ldrsh r4,[r1,r2]
add r4,r4,r3
strh r4,[r1,r2]
add r2,r2,#2
cmp r2,#50
bne loop
```

Primärminne

0x5000	0x1234 0x5678 0xabcd ...
0x5010	0xbeef 0xbeef 0xbeef ...
0x5020	0x0000 0x1111 0x2222 ...
0x6000	0x4321 0x1111 0x0001 ...
0x6010	0x1100 0x0000 0x0115 ...
0x6020	0xffff 0xffff 0xffff ...

Cacheminne

index	status	tag	dataarea
0	tom
1	tom
2	tom
3	tom
4	tom

Direktadresserad cache, exempel

- Minnesåtkomst 1, cachemiss på adress 0x5000
- Endast index 0 får lagra data från adress 0x5000 (indexbitar i adress = 00000000)
- Läs 16 byte från adress 0x5000, sätt tag till resten av bitarna i adress (bit 31 till 12)

```
loop:
  ldrsh r3,[r0,r2]
  ldrsh r4,[r1,r2]
  add r4,r4,r3
  strh r4,[r1,r2]
  add r2,r2,#2
  cmp r2,#50
  bne loop
```

Primärminne			
0x5000	0x1234	0x5678	0xabcd ...
0x5010	0xbeef	0xbeef	0xbeef ...
0x5020	0x0000	0x1111	0x2222 ...
0x6000	0x4321	0x1111	0x0001 ...
0x6010	0x1100	0x0000	0x0115 ...
0x6020	0xffff	0xffff	0xffff ...

Cacheminne			
indexstatus	tag	dataarea	
0	valid	0x5	0x1234 0x5678 0xabcd ...
1	tom
2	tom
3	tom
4	tom

Direktadresserad cache, exempel

- Minnesåtkomst 2, läsning på adress 0x6000
 - Endast index 0 får lagra data från adress 0x6000 (indexbitar i adress = 00000000)
 - Adress har tag = 0x6, cacheminne har tag = 0x5 => cachemiss!
 - Måste ersätta innehåll i cacheline index 0
- Läs 16 byte från adress 0x6000, sätt tag till resten av bitarna i adress (bit 31 till 12)
 - Totalt 2 cachemissar

```
loop:
  ldrsh r3,[r0,r2]
  ldrsh r4,[r1,r2]
  add r4,r4,r3
  strh r4,[r1,r2]
  add r2,r2,#2
  cmp r2,#50
  bne loop
```

Primärminne			
0x5000	0x1234	0x5678	0xabcd ...
0x5010	0xbeef	0xbeef	0xbeef ...
0x5020	0x0000	0x1111	0x2222 ...
0x6000	0x4321	0x1111	0x0001 ...
0x6010	0x1100	0x0000	0x0115 ...
0x6020	0xffff	0xffff	0xffff ...

Cacheminne			
indexstatus	tag	dataarea	
0	valid	0x6	0x4321 0x1111 0x0001 ...
1	tom
2	tom
3	tom
4	tom

Direktadresserad cache, exempel

- Minnesåtkomst 3, skrivning på adress 0x6000.
 - Adress har tag = 0x6, cacheminne har tag = 0x6
=> cacheträff
 - Markera data ändrat (dirty)
 - Totalt 2 cachemissar

```
loop:
ldrsh r3,[r0,r2]
ldrsh r4,[r1,r2]
add r4,r4,r3
strh r4,[r1,r2]
add r2,r2,#2
cmp r2,#50
bne loop
```

Primärminne	
0x5000	0x1234 0x5678 0xabcd ...
0x5010	0xbeef 0xbeef 0xbeef ...
0x5020	0x0000 0x1111 0x2222 ...
0x6000	0x4321 0x1111 0x0001 ...
0x6010	0x1100 0x0000 0x0115 ...
0x6020	0xffff 0xffff 0xffff ...

Cacheminne			
index	status	tag	dataarea
0	dirty	0x6	0x5555 0x1111 0x0001 ...
1	tom
2	tom
3	tom
4	tom

Direktadresserad cache, exempel

- Minnesåtkomst 4, läsning på adress 0x5002
 - Adress har tag = 0x5, cacheminne har tag = 0x6
=> cachemiss!
 - Behöver ersätta innehåll på index 0 i cache
 - Dirty flaggan satt, skriv först data till primärminne (alla 16 byte)
 - Totalt 3 cachemissar

```
loop:
ldrsh r3,[r0,r2]
ldrsh r4,[r1,r2]
add r4,r4,r3
strh r4,[r1,r2]
add r2,r2,#2
cmp r2,#50
bne loop
```

Primärminne	
0x5000	0x1234 0x5678 0xabcd ...
0x5010	0xbeef 0xbeef 0xbeef ...
0x5020	0x0000 0x1111 0x2222 ...
0x6000	0x4321 0x1111 0x0001 ...
0x6010	0x1100 0x0000 0x0115 ...
0x6020	0xffff 0xffff 0xffff ...

Cacheminne			
index	status	tag	dataarea
0	valid	0x5	0x1234 0x5678 0xabcd ...
1	tom
2	tom
3	tom
4	tom

Skrivning till cache, alternativ

- Writeback
 - Skrivningar sker bara till cache
 - Flagga indikerar om data måste skrivas till primärminne när cacheline töms
 - Ändrad data skrivs tillbaks senare
- Writethrough
 - Skrivningar går alltid till primärminnet
 - Kopia av värdet lagras också i cache
 - Enklare att implementera, mindre effektivt

Val av cacheline att ersätta, alternativ

- LRU - Least recently used
 - Håll reda på vilken cacheline som inte använts på länge
 - Dyrt att hålla reda på vilken som senast använts
- Random
 - Slumpmässigt val. Billigare, dock större risk ta fel
- Round-Robin/FIFO
 - Kasta ut den cacheline som varit i cachen längst, oberoende om den använts nyligen etc.
 - Lite billigare

Val av cache parametrar

- Associativitet
 - Vanligen 2 till 8 idag
 - Störst vinst när man går från direktadresserad till 2-vägs associativ cache
 - Högre associativitet betalar sig dåligt
- Vanligt ha separat data och instruktionscache
 - Kallad nivå1 cache
 - Båda läser data från samma minne vid cachemissar
- Multipla nivåer av cache möjliga
 - T ex AMD multiprocessorer: Nivå1, nivå2, nivå3 cache innan primärminnet nås

Problem orsakade av cache

- Data från I/O
 - Läsning av t ex statussignaler får inte lagras i cache (missar ändringar från externa signaler)
 - Skrivning i rätt ordning?
- Cache-koherens
 - Om primärminnet kan skrivas från flera enheter (Multiprocessor) måste alla cache få reda på ändringen
 - Notering: variabler i C taggande med "volatile" kommer inte undan problemet med cache-koherens
 - Behöver få hårdvara och/eller mjukvara hantera tömning av cache i vissa fall

Praktiska kommentarer inför lab5

- Laboration 5 info
 - Använder hårdvara ansluten till windowmaskiner (lablokal Grinden)
 - Placerade i ett låst labb
 - Access på schemalagd tid samt efter att alla 1:a labbtillfällen genomförts
 - Bara en person inloggad per maskin
 - Ni måste kontrollera i schemaservern så inte undervisning pågår i labbet!
- Lab 5 använder rdp-protokoll för att prata med windows maskiner
 - Info på rdpklienter.edu.liu.se