

Övning 4-VT2022

Cacheminnen

Poängen med övningen är dels att se hur man räknar ut adresseringen av ett cacheminne med olika mappningstekniker, men också att visa vilka konsekvenser de olika teknikerna kan få för hit raten.

Direktmappade

Ett direktmappat cacheminne använder minnesadressen från primärminnet som adress direkt i cacheminnet. Beroende på storleken på cacheminnet och längden på raderna används en viss mängd av de minst signifikanta bitarna för att hitta rätt plats i minnet.

Exempel

Cachestorlek: 128 byte

Radlängd: 16 byte

Associativitet: 1-vägs, med andra ord ingen associativitet.

För att identifiera en byte i en rad behöver vi fyra bitar: 0000 - 1111.

För att få ut antalet rader tar vi cachestorleken/radlängden: $128/16=8$

För att identifiera en rad behöver vi därmed tre bitar: 000 - 111

Det ger följande:

Tag	Rad	Byte
$32 - 4 - 3 = 25$ bitar	3 bitar	4 bitar

Rad	Bytes
0000	00
0001	
0010	
0011	
0100	
0101	
0110	
0111	

Om vi tar en kodsnuitt som exempel för att koppla från assembler till cacheminnet:

```
rad 1 movia r8, 0x803C00 # Påverkar inte cacheminnet
rad 2 ldw    r12,0(r8)    # Läser från 0x803C00
rad 3 ldw    r13,56(r8)   # Läser från 0x803C38 (5610 blir 3816) -> 0x803C00 + 38 = 0x803C38
rad 4 stw    r14,32(r8)   # Skriver till 0x803C20 (3210 blir 2016)
rad 5 ldw    r15,128(r8)  # Läser från 0x803C80 (12810 blir 8016)
rad 6 ldw    r16,384(r8)  # Läser från 0x803D80 (38410 blir 18016)
rad 7 stw    r17,0(r8)    # Skriver till 0x803C00
rad 8 ldw    r18,48(r8)   # Läser från 0x803C30 (4810 blir 3016)-> 0x803C00+30= 0x803C30
```

För rad 2 är cacheminnet tomt, vi får en **miss** och läser in från adressen **0x803C00**.

Vi är intresserade av de sju sista bitarna -> $00_{16} = 0 \mid 000 \mid 0000_2$ -> rad = 000 och byte= 0000

Vi läser in en hel rad, 803C00 till 803C0F, till rad 000.

För rad 3 kollar vi **0x803C38** -> $38_{16} = 0 \mid 011 \mid 1000$ -> rad = 011

Rad 011 är tom, vi får en **cachemiss** och läser in ett block till rad 011

På rad 011 finns nu 0x803C30 - 0x803C3F.

För rad 4 kollar vi **0x803C20** -> $20_{16} = 0 \mid 010 \mid 0000$ -> rad = 010

Rad 010 är tom, vi får en **cachemiss** och läser in ett block till rad 010

På rad 010 finns nu 0x803C20 - 0x803C2F.

För rad 5 kollar vi **0x803C80** -> $80_{16} = 1 \mid 000 \mid 0000$ -> rad = 000

På rad 000 med byte 0000 finns ett värde, vars tag inte stämmer överens (**notera ettan** i den åttonde biten, vi hade en nolla där tidigare). Vi får återigen en **cachemiss** och byter ut innehållet i rad 000. På rad 000 finns nu 0x803C80 - 0x803C8F.

För rad 6 kollar vi **0x803D80** -> $80_{16} = 1 \mid 000 \mid 0000$ -> rad = 000

Här ser vi att vi behöver kolla resten av taggen också, eftersom vi återigen är på rad 000.

Vi jämför **0x803C80** med **0x803D80** och ser att de skiljer sig. Vi har alltså **cachemiss** igen, och har nu 0x803D80 - 0x803D8F på rad 000.

För rad 7 kollar vi **0x803C00** -> $00_{16} = 0 \mid 000 \mid 0000$ -> rad 000

Återigen ser vi att taggen inte stämmer, vi får **cachemiss** och läser återigen in ett nytt block till rad 000, där vi nu har 803C00 till 803C0F (igen).

(här ska jämföras innehållet av rad 000 med vad som vi räknat fram för rad 7. Raden (000) ändrads efter rad 6 så att taggen var 0x803D.

För rad 8 kollar vi **0x803C30** -> $30_{16} = 0 \mid 011 \mid 0000$ -> rad 011

På rad 011 läste vi tidigare in 0x803C30 - 0x803C3F, och vi får därmed en **träff** i cachén.

Det här minnet, med direktnöppning av adresser, ger alltså **en enda träff**. Därmed har vi egentligen heller inte mycket nytta av cacheminnet. Vi kan se att vi har kastat ut saker som hamnat på samma rad. Kan man göra något åt det?

Ett associativt cacheminne är betydligt simplare än de direktmappade. Det finns inte längre en bindning mellan adressen och en specifik rad, utan ett block från primärminnet läggs in direkt i nästa lediga rad i cacheminnet.

Tag	Byte
32 - 4 = 28 bitar	4 bitar

Återanvänder det tidigare exemplet, för enkelhetens skull

Associativitet: Full.

rad 8 ldw r18,48(r8) # Läser från 0x803C30₁₆ = 1000 0000 0011 1100 0011 0000₂

Här kommer vi att kunna lägga in allt data i minnet, så inget behöver kastas ut bara för att raden redan var upptagen.

Mängdassociativa

För att kombinera snabbheten med direktmappade minnen och de färre cachemissarna från helt associativa minnen finns mängdassociativa cacheminnen. De använder rader, precis som de direktmappade minnena, men varje (logisk)rad är en mängd av rader. Exempelvis har ett tvåvägsassociativt minne två block för varje (logisk)rad.

Exempel

Återanvänder det tidigare exemplet, för enkelhetens skull

Cachestorlek: 384 byte (Större eftersom vi ska få in tre block per rad)

Radlängd: 16 byte

För att få ut antalet rader tar vi cachestorleken/radlängden: $384/(16 \cdot 3) = 8$

För att identifiera en rad behöver vi därmed tre bitar: 000 - 111

Associativitet: 3-vägs.

Tag																								Rad			Byte																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																
32 - 4 - 3 = 25 bitar																								3 bitar			4 bitar																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																
																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											</

Samma kodsnuitt som ovan:

```
rad 1 movia r8, 0x803C00 # Påverkar inte cacheminnet
rad 2 ldw    r12,0(r8)    # Läser från 0x803C00
rad 3 ldw    r13,56(r8)   # Läser från 0x803C38
rad 4 stw    r14,32(r8)   # Skriver till 0x803C20
rad 5 ldw    r15,128(r8)  # Läser från 0x803C80
rad 6 ldw    r16,384(r8)  # Läser från 0x803D80
rad 7 stw    r17,0(r8)    # Skriver till 0x803C00
rad 8 ldw    r18,48(r8)   # Läser från 0x803C30
```

För rad 2 är cacheminnet tomt, vi får en miss och läser in från adressen **0x803C00**.

Vi är intresserade av de sju sista bitarna -> $00_{16} = 0 \mid 000 \mid 0000_2$ -> rad = 000

Vi läser in en hel rad, 803C00 till 803C0F, till rad 000.

För rad 3 kollar vi **0x803C38** -> $38_{16} = 0 \mid 011 \mid 1000$ -> rad = 011

Rad 011 är tom, vi får en cachemiss och läser in ett block till rad 011

På rad 011 finns nu 0x803C30 - 0x803C3F.

För rad 4 kollar vi **0x803C20** -> $20_{16} = 0 \mid 010 \mid 0000$ -> rad = 010
Rad 010 är tom, vi får en cachemiss och läser in ett block till rad 010
På rad 010 finns nu 0x803C20 - 0x803C2F.

För rad 5 kollar vi **0x803C80** -> $80_{16} = 1 \mid 000 \mid 0000$ -> rad = 000
På rad 000 med byte 0000 finns ett värde, vars tag inte stämmer överens (**notera ettan** i den åttonde biten, vi hade en nolla där tidigare) och vi får en cachemiss. Då **lägger vi in vårt nya block på andra platsen** på rad 000. Där finns nu också 0x803C80 - 0x803C8F.

För rad 6 kollar vi **0x803D80** -> $80_{16} = 1 \mid 000 \mid 0000$ -> rad = 000
Här ser vi att vi behöver kolla resten av taggen också, eftersom vi återigen är på rad 000.
Vi jämför **0x803C80** med **0x803D80** och ser att de skiljer sig. Vi har alltså cachemiss igen, och får sätta in 0x803D80 - 0x803D8F i platsen till **höger (tredje platsen)**.

För rad 7 kollar vi **0x803C00** -> $00_{16} = 0 \mid 000 \mid 0000$ -> rad 000
På första platsen i rad 000 finns 803C00 till 803C0F, och vi har därmed en träff i cachén.

För rad 8 kollar vi **0x803C30** -> $30_{16} = 0 \mid 011 \mid 0000$ -> rad 011
På rad 011 läste vi tidigare in 0x803C30 - 0x803C3F, och vi får därmed en träff i cachén.

Vi behövde inte kasta lika mycket data, och fick också en till träff. Sökningen går fortare jämfört med helt associativa minnen.