KANDIDAT

10095

PRØVE

TDAT1003 A Datateknikk og operativsystem

Emnekode	TDAT1003
Vurderingsform	Skriftlig eksamen
Starttid	03.12.2018 08:00
Sluttid	03.12.2018 11:00
Sensurfrist	03.01.2019 23:59
PDF opprettet	14.12.2019 15:38

Forside

Oppgave	Tittel	Oppgavetype
i	Forside	Dokument

Operativsystem - flerevalg (20%)

Oppgave	Tittel	Oppgavetype
1	OS - main function	Flervalg
2	OS - virtualisation	Flervalg
3	OS - security	Flervalg
4	OS - processes	Flervalg
5	OS - dual-mode operation	Flervalg
6	OS - process memory	Flervalg
7	OS - mode transfer	Flervalg
8	OS - exceptions	Flervalg
9	OS - mode transfer	Flervalg
10	OS - interrupts	Flervalg

Operativsystem - kernel (15%)

Oppgave	Tittel	Oppgavetype
11	Kernel types	Langsvar
12	Fork, exec, wait	Langsvar

Operativsystem - minneadministrasjon (15%)

Oppgave	Tittel	Oppgavetype
13	OS - paged memory	Langsvar

Datateknikk - Tallsystemer og instruksjonsformat

Oppgave	Tittel	Oppgavetype

14	Tallsystemer	Langsvar
15	Instruksjonsformat	Langsvar
16	Instruksjoner	Langsvar

Prinsippet om lokalitet

Oppgave	Tittel	Oppgavetype
17	Aksessmønster	Tekstfelt
18	Prinsippet om lokalitet	Tekstfelt
19	Instruksjoner og data	Langsvar

Datateknikk - moderne primærminne

Oppgave	Tittel	Oppgavetype
20	Flernivå caching	Langsvar
21	Flerkjerneprosessorer og cache	Langsvar
22	Splittet cache	Langsvar

1

OS - main function

Hva beskriver best hovedfunksjonen til et operativsystem?
Velg ett alternativ

- Tillat at brukerprogrammer direkte styrer CPUen
- Start systemet og overlater kontrollen av tastaturet og musen til brukerprogrammer
- Tillat brukerprogrammer å administrere systemressurser direkte
- Administrer systemressurser og leverer et sett av tjenester til brukerprogrammer

2 **OS - virtualisation**

Virtualisering i forbindelse med et operativsystem er definert som:
Velg ett alternativ

- Gir en illusjon av ressurser som kanskje ikke er fysisk tilstede.
- Et sett av felles tjenester som tilbys til brukerprogrammer.
- Basissettet for instruksjoner som er felles for alle instruksjonset-arkitekturer.
- Begrensning av brukerprogrammer til å utføre bare ikke-privilegerte instruksjoner.

3 **OS - security**

Applikasjonsisolering er en viktig egenskap for et operativsystem fordi:
Velg ett alternativ

- Begrensninger på fysisk minne krever begrenset kommunikasjon mellom applikasjoner.
- Bare ett program kan utføre instruksjoner på CPUen på et gitt tidspunkt.
- En feil i en applikasjon bør ikke forstyrre andre applikasjoner.
- Kommunikasjon mellom applikasjoner er risikabelt og unødvendig.

4 **OS - processes**

For å kunne kjøre flere kopier av det samme programmet, så må operativsystemet:
Velg ett alternativ

- Opprette flere kopier av programmets instruksjoner, statiske data og stack, og gjenbruker den første eksisterende kopien av programmets heap for å spare minne
- Opprette en peker til første eksisterende kopi av programmets instruksjoner, statiske data, heap og stack
- Opprette flere kopier av programmets instruksjoner, statiske data, heap og stack
- Lage flere kopier av programmets statiske data, heap og stack, og gjenbruker den første eksisterende kopien av programmets instruksjoner for å spare minne

5

OS - dual-mode operation

Dual-mode-operasjon, som tillater separasjon mellom bruker- og kjernemodus, oppnås ved:
Velg ett alternativ

- Deler den fysiske prosessoren i to virtuelle prosessorer: en for brukerprosesser og en for kjernen
- Deler fysisk minne i to regioner: en for brukerprosesser og en for kjernen
- En tolk i kjernen som kontrollerer om prosessen har tillatelse til å utføre den aktuelle instruksjonen
- Et flagg i prosessoren som angir om nåværende utførelsesmodus er bruker- eller kjernemodus

6

OS - process memory

Program (eller text) -segmentet i prosessminneoppsettet inneholder:

Velg ett alternativ

- Dynamisk allokert minne
- Utførbare instruksjoner
- Statiske og globale variabler
- Lokale variabler

7

OS - mode transfer

Hvilken av følgende trenger normalt ikke en modusoverføring?
Velg ett alternativ

- Prosessor-unntak (exception)
- Systemkall
- Henter neste programinstruksjon
- Avbrudd

8

OS - exceptions

Alle de følgende er eksempler på prosessor-unntak (exception) unntatt:
Velg ett alternativ

- Forsøk å utføre en privilegert instruksjon i brukermodus
- Timeravbrudd (timer interrupt)
- Dele-med-nul (divide-by-zero)
- Minne sidefeil

9

OS - mode transfer

En overføring fra kjerne- til brukermodus skjer i alle følgende tilfeller, unntatt:
Velg ett alternativ

- Kontekstbryter (bytt til en annen brukerprosess)
- Fortsett prosessen etter avbrudd
- Ny prosess
- Prosessavslutning

10

OS - interrupts

Avbrudd er gunstige sammenlignet med polling (også kalt "busy waiting") fordi:
Velg ett alternativ

- Avbrudd er enklere å implementere i programvare.
- Polling kaster bort mange CPU sykluser til ingen nytte.
- Avbrudd krever mindre maskinvare.
- Polling er strengt synkron.

11 Kernel types

Forklar forskjellen mellom en monolittisk kjerne og en mikrokjerne. Til hver, nevn en fordel og en ulempe.

Skriv ditt svar her...

En monolittisk kjerne er en type kjerne som kjører mest mulig av de nødvendige tjenestene i kjernemodus. En fordel med monolittisk kjerne er at kjernen hele tiden har kontroll. Det er den som styrer tjenestene, og på den måten har til enhver tid oversikt ettersom den ikke overlater tjenestene til noen andre enn seg selv.
En ulempe med en monolittisk kjerne er at dersom en av tjenestene skulle feile, ville det få katastrofale følger for datamaskinen ettersom det potensielt kan ødelegge hele PC'en. Den er derfor mer sårbar for feil, og er helt avhengig av at alt fungerer nøyaktig som det skal.

En mikrokjerne er en type kjerne der man ønsker å flytte flest mulig av de mest nødvendige tjenestene ut av kjernen, og kjøre dem i brukermodus. En fordel med en mikrokjerne er at dette isolerer de ulike tjenestene fra hverandre. Det vil si at en krasj eller feil i en tjeneste ikke vil få konsekvenser for noen av de andre. En ulempe med en mikrokjerne er at kjernen ikke har den kontrollen den nødvendigvis ønsker. At tjenestene kjører i brukermodus gjør at de fungerer uavhengig av hverandre, og kjernen kan ikke styre de slik den kan i en monolittisk kjerne.

12 Fork, exec, wait

Du har i oppgave å skrive kildekoden til et enkelt skall-program.

- Forklar hvordan fork(), exec(), and wait() blir brukt sammen til å kjøre andre programmer.
- Beskriv hva hvert enkelt systemkall gjøre, og hvordan det blir brukt i skallprogrammet.

Svaret kan være i form av setninger eller pseudo-kode (syntaksen er ikke viktig).
Skriv ditt svar her...

```
pid = fork()
if (pid == 0) { //Hvis det er barneprosesen
exec() //Barneprosessen kjører
} else { //Hvis det er foreldreprosessen
wait() //Foreldreprosessen venter på at barnet skal kjøre ferdig.
}
```

Slik pseudo-koden viser vil vi først opprette en ny barneprosess med fork(). Deretter vil vi kjøre exec() eller wait() avhengig av hva fork() returnerte. Barneprosessen vil returnere 0. Dersom det er barnet som returnerer ønsker vi at det skal kjøre og utføre sine oppgaver (exec). Ellers (fork() returnerer barneprosessend id) så ønsker vi at foreldreprosessen skal vente på barneprosessen kjører ferdig. Disse fungerer sammen på den måten at foreldreprosessen oppretter barn for å gjøre arbeid for seg. Foreldren ønsker ikke å gjøre noe arbeid før barneprosessen har blitt ferdig utført (wait).

Systemkallet fork() vil lage en kopi av en prosess. Man kaller prosessen som kaller fork() for **foreldreprossen**, mens den nye prosessen kalles for **barneprosessen**. Barnet vil være en identisk kopi av foreldreprossen. Det er kun et unntak som gjør at vi kan skille mellom de to. Barneprossen vil returnere 0, mens foreldreprosessen vil returnere barneprosessens id. I skallprogrammet vil fork() gjøre det mulig å opprette nye prosesser.

Systemkallet exec() vil bytte ut den gjeldende prosessen som kjører med prosessen som blir kalt med exec(). I tilfellet over vil barneprosessen kalle exec(), da vi ønsker at barneprosessen skal kjøre på prosessoren. I et skalprogram blir dette nyttig ettersom det gjør at barneprosessen vil kunne kjøre etter at den har blitt opprettet.

Systemkallet wait() blir kalt av foreldreprosessen og indikerer at foreldreprosessen ikke skal kjøre før barneprosessen er utført/terminert/har krasjet. Wait blir viktig i skallprogrammet ettersom den gjør at den venter på at barneprosessen skal utføres. I tillegg er den viktig for å unngå at vi får såkalte zombie-prosesser. Med wait vil nemlig barneprosessen, og dens minne, bli fullstendig fjernet etter at den har blitt utført.

13 OS - paged memory

Anta en sidetabell med fire spor og sekvensen av minnehenvisninger som er gitt i tabellen nedenfor.

1. For hver av de følgende sideutskiftningsalgoritmer, fyll ut tabellen med en '+' hvis referansen resulterer i et treff, eller referert bokstav (f.eks. 'A') hvis referansen resulterer i en feil.
    1. Optimal (MIN)
    2. Først inn først ut (FIFO)
    3. Minst nylig brukt (LRU)
2. For hver algoritme, hva er treff-frekvensen for referansesekvensen?

Merk: Kopier og lim inn følgende tabell for hver algoritme for å gjøre det lettere å fylle ut.

	A	B	C	A	B	C	E	F	E	F	E	F	A	B	A	C	A	F	E	F	C	D	B	A	F
1																									
2																									
3																									
4																									

Skriv ditt svar her...

1. Optimal (MIN)

	A	B	C	A	B	C	E	F	E	F	E	F	A	B	A	C	A	F	E	F	C	D	B	A	F
1	A			+									+		+		+							+	
2		B			+									+		C					+	D			
3			C			+		F		+		+						+		+					+
4							E		+		+								+				B		

2. Først inn først ut (FIFO)

	A	B	C	A	B	C	E	F	E	F	E	F	A	B	A	C	A	F	E	F	C	D	B	A	F
1	A			+				F		+		+						+	E					A	
2		B			+								A		+		+			F					+
3			C			+								B								D			
4							E		+		+					C					+		B		

3. Minst nylig brukt (LRU)

	A	B	C	A	B	C	E	F	E	F	E	F	A	B	A	C	A	F	E	F	C	D	B	A	F
1	A			+				F		+		+						+		+					+
2		B			+								A		+		+							+	
3			C			+								B		C					+	D	B		
4							E		+		+								+						

Oppgave 2)

For den optimale vil det totalt være 25 referanser. Vi får totalt 17 treff. Dette tilsvarer en treffrate på

$$\frac{17}{25} = 68\ \%$$

For FIFO vil det totalt være 25 referanser. Vi får totalt 12 treff. Dette tilsvarer en treffrate på

$$\frac{12}{25} = 48\ \%$$

For LRU vil det totalt være 25 referanser. Vi får totalt 15 treff. Dette tilsvarer en treffrate på

$$\frac{15}{25} = 60\ \%$$

14 Tallsystemer

Skriv følgende bitmønster på heksadesimal og desimal form:

`111100011`

Vis fremgangsmåten nøye, og forklar hva du gjør.

Skriv ditt svar her...

111100011_2

Det første jeg gjør er å sette opp tabellen der man ser hvilke grupper á fire bits som tilsvarer de ulike heksadesimalverdiene og desimalverdiene:
(De fire første utgjør den binære verdien, **uthevet** er heksadesimal og *kursiv* desimalverdien.

0	0	0	0	**0**	*0*
0	0	0	1	**1**	1
0	0	1	0	**2**	2
0	0	1	1	**3**	3
0	1	0	0	**4**	4
0	1	0	1	**5**	5
0	1	1	0	**6**	6
0	1	1	1	**7**	7
1	0	0	0	**8**	8
1	0	0	1	**9**	9
1	0	1	0	**A**	10
1	0	1	1	**B**	11
1	1	0	0	**C**	12
1	1	0	1	**D**	13
1	1	1	0	**E**	14
1	1	1	1	**F**	15

Deretter deler jeg opp bitmønsteret i grupper á fire bits:
$0001\ 1110\ 0011_2$ (Legger til tre nuller foran slik at vi får grupper á fire bits)

Binær til heksadesimal:
Bruker tabellen jeg har laget ovenfor og erstatter hver gruppe med tilsvarende heksadesimal-verdi:
$0001\ 1110\ 0011_2 = \mathbf{1E3_{16}}$

Binær til desimal:
Tar utgangspunktet i heksadesimalverdien jeg fant ovenfor og regner ut desimalverdien på følgende måte:

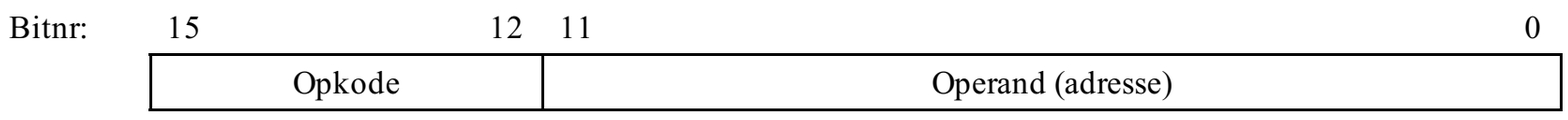
$1 * 16^2 + \mathbf{14 * 16^1} + 3 * 16^0 = 256 + 224 + 3 = \mathbf{483_{10}}$

NB: Ser i tabellen at E tilsvarer 14 i desimaltall.
(Kunne også ha tatt utgangspunkt i hvert enkelt binære tall og regnet ut, men velger å bruke heksadesimalverdien da jeg finner den metoden enklere og mindre risikofylt).

15

Instruksjonsformat

Nedenfor ser du oppbyggingen av instruksjonene til den hypotetiske maskinen vi har brukt i kurset:



Forklar følgende tre begreper:

- Instruksjonsformat
- opkode
- operand.

Vær relativt detaljert.

Skriv ditt svar her...

Et **instruksjonsformat** beskriver hele instruksjonen. Dette inkluderer opkoden og operanden. Det er med andre den fulle instruksjonen som beskriver det som skjer i sin helhet.

Opkoden beskriver hva slags instruksjon som skal utføres. I den hypotetiske maskinen vi har brukt i kurset består den av fire bits. Eksempler på instruksjoner som opkoden kan fortelle er:

- ADD
- READ
- WRITE

Operanden er den siste delen. Den består av 12 bits og forteller hvor dataen som skal behandles ligger. Det er med andre ord adressen hvor dataen som skal behandles ligger.

Et eksempel på en fullstendig instruksjon (bitmønster) kunne vært:
1101 100101011001
På symbolsk form kunne den ha sagt
READ 480. Altså les fra adressen 480.

Bitnr:
15
12
11
0
Opkode
Operand (adresse)

16

Instruksjoner

Tabellen nedenfor er også hentet fra leksjonene i kurset. Den viser oppbyggingen av instruksjonene på den hypotetiske maskinen.

Opkode (binær)	Symbolsk navn	Funksjon
0001	LOAD	Les data fra minnet til AC. Adressefeltet angir hvilken minne-lokasjon som skal leses.
0010	STORE	Skriv til minnet fra AC. Adressefeltet angir til hvilken minne-lokasjon data skal skrives.

0101	ADD	Adder et tall i minnet med innholdet i AC og legg resultatet i AC. Adressefeltet angir i hvilken minnelokasjon tallet er.

Anta at den heksadesimale verdien nedenfor er en instruksjon på den hypotetiske maskinen:

515E

Svar på følgende to spørsmål:

1. Hvilken instruksjon er det? Forklar kort hva denne instruksjonen gjør.
2. Hvor ligger data som instruksjonen bruker? Angi adressen på desimal form.

Begrunn godt, og vis alle utregninger nøye

Skriv ditt svar her...

Det første jeg gjør er å gjøre om heksadesimalverdien 515E til binær form. Her bruker jeg lignende metode som jeg brukte i en tidligere oppgave.

0	0		0	**0**	*0*
0	0	0	1	**1**	1
0	0	1	0	**2**	2
0	0	1	1	**3**	3
0	1	0	0	**4**	4
0	1	0	1	**5**	5
0	1	1	0	**6**	6
0	1	1	1	**7**	7
1	0	0	0	**8**	8
1	0	0	1	**9**	9
1	0	1	0	**A**	10
1	0	1	1	**B**	11
1	1	0	0	**C**	12
1	1	0	1	**D**	13
1	1	1	0	**E**	14
1	1	1	1	**F**	15

Tar hvert siffer i den heksadesimale verdien hver for seg og finner binærverdien ved bruk av tabellen ovenfor:

515E$_{16}$ = 0101 0001 0101 1110$_2$

1)

For å finne ut hvilken instruksjon dette er tar jeg utgangspunkt i den binære verdien jeg har funnet ovenfor.

5 på heksadesimal form tilsvarer 0101 på binær form. Dermed vil instruksen være ADD. ADD vil addere tallet vi finner på adressen med et tall som allerede ligger i akkumulator-registeret. Deretter vil resultatet av disse i akkumulator-registeret.

2)

Ettersom de fire første bitsene forteller oss hvilken instruksjon som skal utføres vil de 12 siste fortelle adressen.

Adressen er altså: 0001 0101 1110$_2$

For å gjøre om dette til desimalform gjør jeg følgende:

Fjerner de tre første nullene da disse er urelevante for desimalverdien

1 0101 1110$_2$

Deretter regner jeg ut slik:

$(1 * 2^8) + (0 * 2^7) + (1 * 2^6) + (0 * 2^5) + (1 * 2^4) + (1 * 2^3) + (1 * 2^2) + (1 * 2^1) + (1 * 2^0)$

= 256 + 64 + 16 + 8 + 4 + 2 = **350$_{10}$**

Hele instruksjonen vil dermed se slik ut på symbolsk form: ADD 350

Opkode (binær)

Symbolsk navn

Funksjon

0001

LOAD

Les data fra minnet til AC. Adressefeltet angir hvilken minnelokasjon som skal leses.

0010

STORE

Skriv til minnet fra AC. Adressefeltet angir til hvilken minnelokasjon data skal skrives.

0101

ADD

Adder et tall i minnet med innholdet i AC og legg resultatet i AC. Adressefeltet angir i hvilken minnelokasjon tallet er.

Innledning

Man skulle kanskje tro at når CPU aksesserer primærminnet, så hadde alle minnelokasjoner den samme sannsynligheten for å bli brukt til enhver tid.

I praksis viser det seg imidlertid at dersom man vet hvilken lokasjon som ble aksessert forrige gang, så kan man med stor sannsynlighet forutsi hvor neste aksess blir.

17 Aksessmønster

Hvor ligger sannsynligvis neste aksess?

Skriv ditt svar her...

Neste aksess vil sannynsligvis ligge i nærheten av den forrige aksesserte minnelokasjonen.

18 Prinsippet om lokalitet

Denne egenskapen ved programmene har gitt opphav til et viktig prinsipp i datateknikk, nemlig prinsippet om lokalitet. Hva sier prinsippet om lokalitet?

Skriv ditt svar her...

Prinsippet om lokalitet sier at vi med overveiende sannsynlighet kan si at neste aksess vil ligge i nærheten av den forrige aksesserte minnelokasjonen.
Vi skiller mellom romlig og temporal lokalitet. Romlig lokalitet forteller oss at minnelokasjonene ofte ligger sekvensielt. De ligger etterhverandre. Dette gjelder for eksempel ved utføring av et program; der man som regel vil utføre instruksjoenne sekvensielt, noe som fører til at instruksjonene ligger etter hverandre.
Temporal lokalitet sier at vi ofte bruker lokasjoner flere ganger. De blir brukt gjentatte ganger over et visst tidspunkt. Dette gjelder eksempelvis ved bruk av løkker der man igjen og igjen benytter seg av de samme minnelokasjonene.

19 Instruksjoner og data

Gjelder prinsippet både for instruksjoner og for data, eller gjelder det bare for en av dem?

Begrunn svaret nøye, og gi eksempler på at det gjelder / ikke gjelder.

Skriv ditt svar her...

Prinsippet om lokalitet gjelder for både instruksjoner og data.

Det gjelder for instruksjoner ettersom instruksjoner som regel blir utført sekvensielt. Det vil si at de blir utført etter hverandre. Dermed vil de også ligge sekvensielt i minne, noe som understreker at prinsippet om lokalitet gjelder for instruksjoner.

Samtidig gjelder prinsippet også for data. Data blir sjeldent lagret alene, men heller i sammenhengende datastrukturer. For eksempel i en tabell. Dermed vil de bli lagret sammenhengende, og prinsippet om lokalitet gjelder.

Innledning

Denne oppgaven omhandler sammenhengen mellom cachens oppbygging og oppbyggingen til moderne prosessorer.

20 Flernivå caching

Hva er flernivå caching?

(Hint: Hva er L1-, L2- og L3-cache?)

Skriv ditt svar her...

Flernivå caching går ut på at man har flere nivåer av cacher. Det vil si at vi har flere cacher til rådighet.
L1-cachen er en cache som inneholder kopier av minnelokasjoner som er mest brukt. Denne cachen er svært raskt, men til gjengjeld har den begrenset lagrinskapasitet (KB-området). Ettersom den er svært liten vil det være lite overhead, og man vil heller ikke bli straffet særlig hardt ved bom. Samtidig er gevinsten svært stor ved treff ettersom den er meget rask.

L2-cachen er ulik L1-cachen ettersom den blir brukt for minnelokasjoner som er litt mindre brukt. L2-cachen har større lagrinskapasitet enn L1-cachen, men er til gjengjeld ikke like rask. Sammenligned med L1-cachen vil man bli straffet mer ved bom (pga. mer overhead), men sannsynligheten for treff er høyere.

Man ser mønsteret med cachene. L3-cachen har dermed større lagringskapasitet, men er også litt tregere enn de to andre. Hvis det også blir bom med L3-cachen vil man bli straffet relativt hardt da aksessering av minnelokasjonen vil ta relativt lang tid.

En annen vesentlig forskjell er at L1- og L2-cachen er knyttet opp mot hver sin kjerne. I en flerkjernet prosessor vil hver kjerne ha hver sin L1- og L2-cache, mens det på nyere prosessorer er vanlig med en felles L3-cache. Kjernene vil med andre ord ha L3-cachen som en delt cache.

21 Flerkjerneprosessorer og cache

Moderne prosessorer består av flere kjerner. Det vil si flere prosessorer som er bygget inn i samme integrerte krets.

Beskriv hvordan moderne prosessorers cache-arkitektur støtter flerkjerneprosessorer.

Skriv ditt svar her...

Når man sier at moderne prosessorer består av flere kjerner vil det si at de ulike kjernene fungere slik at de kan utføre oppgaver på egenhånd. De jobber som helst selvstendige enheter. Derfor er det viktig at det er tilrettelagt slik at de får utført deres oppgaver på en best mulig måte.
En viktig del av prosessoren og kjernene er at de utfører oppgavene raskt. Dette er grunnen til at man benytter seg av cache. På moderne prosessorer har man løst det slik at hver kjerne har sine egne cacher. Hver kjerne har sin egen L1- og L2-cache. Dette sikrer at de får utført instrukser i en svært høy hastighet da de ikke behøver å dele på cachene med andre kjerner. Dette ville gjort at ting ville tatt mer tid.
I tillegg har moderne prosessorer en felles (delt) L3-cache. Denne blir i tillegg til de allerede isolerte L1- og L2-cachene for hver kjerne. Denne lagrer de mindre brukte minnelokasjonene, men er fortsatt viktig for at kjernene skal opprettholde et høyt tempo.

22 Splittet cache

En splittet cache er en cache som består av to del-cacher. Den ene (del-)cachen brukes bare til instruksjoner, og den andre (del-)cachen brukes bare til data.

1. Hvilke fordeler og ulemper har en slik cache sammenliknet med en cache som lagrer både data og instruksjoner?
2. Er det egenskaper med instruksjoner som kan gjøre en instruksjons-cache enklere enn en data-cache?

Skriv ditt svar her...

1)

Fordelene med en splittet cache er at cachen slipper å ta hensyn til hvorvidt det er instruksjoner eller data ettersom cachen er splittet med tanke på de to. Dersom det blir kopiert innhold til data-cachen vet prosessoren hva slags informasjon dette er, og trenger dermed ikke ta hensyn slik den måtte ha gjort dersom cachen var enhetlig.

Ulempen er at cachen ikke kan avpasse de to i forhold til hverandre. Data- og instruksjons-cachen vil ha hver sin statiske størrelse, og dette gjør det ikke mulig å avpasse dersom det skulle vise seg at datainformasjonen trenger mer minne enn instruksjonene. Det blir derfor mindre fleksibelt i forhold til en enhetlig cache.

2)

En egenskap med instruksjonene som kan gjøre en instruksjons-cache enklere enn en data-cache er at instruksjonene alle er på samme form. De består av en op-kode og operanden. De har faste strukturer sånn sett, mens data kan komme i veldig mange ulike former.

Det er klart at alt blir lagret i form av bitmønster, men data kan fortsatt så mangt. Det kan være en integer, en tekststreng osv. Det er dermed mer hensyn å ta når det gjelder data enn instruksjoner. Dermed kan instruksjonscachen være enklere enn data-cachen.