NTNU

KANDIDAT

10513

PRØVE

TDT4160 1 Datamaskiner

Emnekode	TDT4160
Vurderingsform	Skriftlig eksamen
Starttid	11.12.2025 14:00
Sluttid	11.12.2025 18:00
Sensurfrist	08.01.2026 23:59
PDF opprettet	11.12.2025 19:23

Forside

Oppgave	Tittel	Oppgavetype
i	Forside	Informasjon eller ressurser

Instruksjonssett

Oppgave	Tittel	Oppgavetype
1.1	Instruksjonssett	Fyll inn tekst

Enkeltsykelprosessor

Oppgave	Tittel	Oppgavetype
2.1	Kontrollsignaler 1	Fyll inn tekst
2.2	Forklare kontrollsignaler 1	Langsvar
2.3	Kontrollsignaler 2	Fyll inn tekst
2.4	Forklare kontrollsignaler 2	Langsvar

Aritmetisk-logisk enhet

Oppgave	Tittel	Oppgavetype
3.1	ALU kontrollsignaler	Fyll inn tekst
3.2	Forklare ALU kontrollsignaler	Langsvar

Multiplikator

Oppgave	Tittel	Oppgavetype
4.1	Verdier i produktregisteret	Fyll inn tekst
4.2	Forklare multiplikatoren	Langsvar

Flyttall

Oppgave	Tittel	Oppgavetype
5.1	Flyttall til desimaltall	Fyll inn tall
5.2	Forklare flyttall	Langsvar

Samlebånd

Oppgave	Tittel	Oppgavetype
6.1	Samlebånd, kontrollsignaler	Fyll inn tekst
6.2	Forklare samlebånd	Langsvar
6.3	Korrekt utføring	Langsvar

Hurtigbuffer og virtuelt minne

Oppgave	Tittel	Oppgavetype
7.1	Adresseformat	Fyll inn tekst
7.2	Forklare adresseformat	Langsvar
7.3	Aksesstid	Fyll inn tall
7.4	Forklare aksesstid	Langsvar
7.5	TLB	Langsvar

Prosessorer med høyere ytelse og parallelle datamaskiner

Oppgave	Tittel	Oppgavetype
8.1	Register renaming	Langsvar
8.2	SISD og SIMD	Langsvar

1 Instruksjonssett

Instruksjon	Opcode	Funct3	Funct7
add	0110011	000	0000000
lw	0000011	010	—
sll	0110011	001	0000000
xor	0110011	100	0000000
andi	0010011	111	—
sw	0100011	010	—
slt	0110011	010	0000000
sra	0110011	101	0100000
sub	0110011	000	0110000
addi	0010011	000	—

I denne oppgaven gir vi 1 poeng for hvert riktige svar. Feil svar/ubesvart gir 0 poeng.

Angi maskinkode for instruksjonen *slt x10, x15, x20:*

- opcode: 0110011
- rs1: 01111
- rs2: 10100
- rd: 01010
- func3: 010
- func7: 0000000

Angi maskinkode for instruksjonen *andi x5, x4, 0x7fb:*

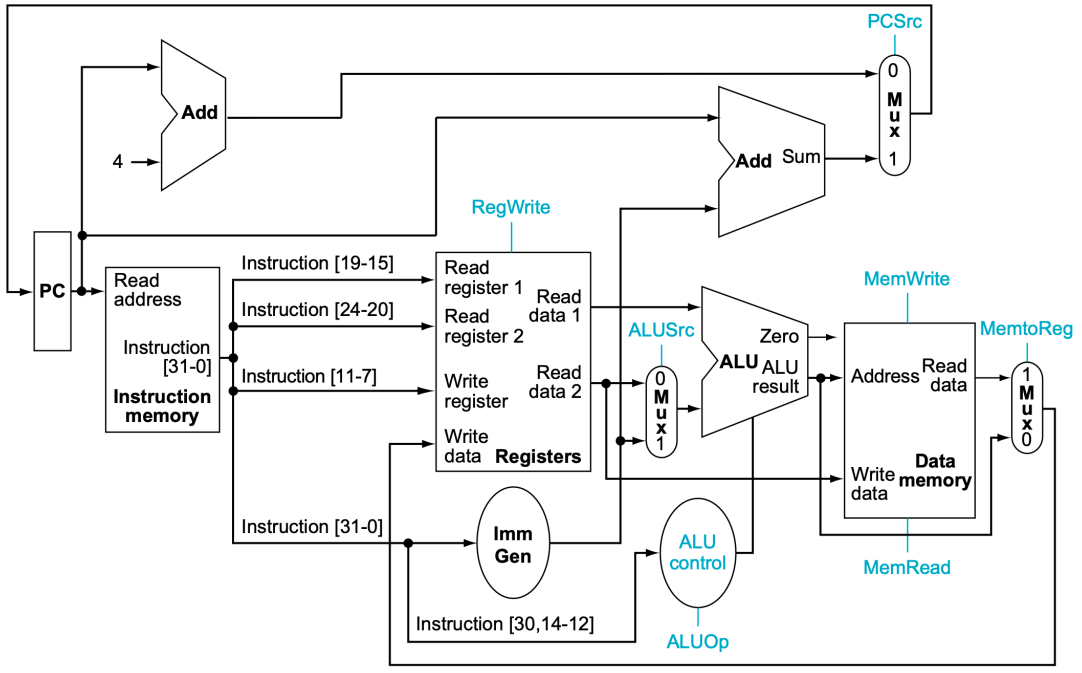
- opcode: 0010011
- rs1: 00100
- rd: 00101
- imm: 011111111011
- func3: 111

Angi maskinkode for instruksjonen *sw x30, -16(x14):*

- opcode: 0100011
- rs1: 01110
- r2: 11110
- imm: 111111110000
- func3: 010

Maks poeng: 16

2



Denne seksjonen omhandler prosessoren vist over. Du må bruke "don't care" (X) når du kan. I oppgavene som er automatisk rettet, gir vi 0,5 poeng for riktige svar og -0,25 poeng for feil svar. Ubesvarte oppgaver gir 0 poeng.

2.1 Kontrollsignaler 1

Prosessoren skal utføre instruksjonen *slt*. Hvilke verdier skal kontrollsignalene under ha?

RegWrite: 1

ALUSrc: 0

PCSrc: 0

MemWrite: 0

MemRead: 0

MemtoReg: 0

Maks poeng: 3

2.2 Forklare kontrollsignaler 1

Forklar hvordan kontrollsignalene du oppga i oppgave 2.1 gjør at *slt*-instruksjonen utføres korrekt.

Skriv svaret ditt her. Endringer blir lagret automatisk.

Instruksjonen hentes og dekodes først fra PC. Deretter sendes rs1 inn fra read data 1 og rs2 fra read data 2, da må ALUSrc være 0 for at rs2 skal velges og ikke immediate. ALU-en regner deretter ut resultatet av operasjonen, 1 hvis rs1 < rs2, 0 ellers. Det skal ikke skrives eller leses til minne så MemWrite og MemRead settes til 0. MemToReg må settes til 0 slik at resultatet fra ALU-en sendes tilbake til registerfilen for skriving, og RegWrite må være 1 slik at resultatet blir skrevet til rd. Neste instruksjon må også få adressen sin inn i PC, så PCSrc må være 0 slik at adressen blir inkrementert med 4 som er neste instruksjons adresse.

Ord: 118

Maks poeng: 5

2.3 Kontrollsignaler 2

Prosessoren skal utføre instruksjonen *sb*. Hvilke verdier skal kontrollsignalene under ha?

RegWrite: 0

ALUSrc: 1

PCSrc: 0

MemWrite: 1

MemRead: 0

MemtoReg: X

Maks poeng: 3

2.4 Forklare kontrollsignaler 2

Forklar hvordan kontrollsignalene du oppga i oppgave 2.3 gjør at *sb*-instruksjonen utføres korrekt.

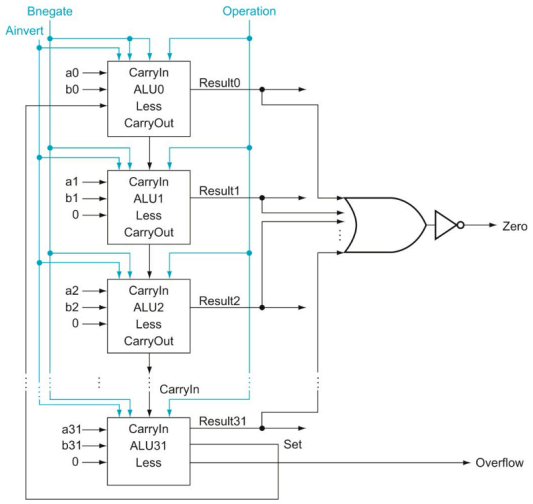
Skriv svaret ditt her. Endringer blir lagret automatisk.

Instruksjonen hentes og dekodes først fra PC. Deretter blir rs1 og immediate hentet og sendt til ALU-en, da må ALUSrc være 1 for at immediate skal velges. ALU-en regner deretter ut adressen som skal skrives til som er rs1 + immediate. Deretter skrives det til denne adressen med data fra rs2, så MemWrite må være 1. Det skal ikke leses til minne så MemRead må være 0. Det skal ikke skrives til noe register så RegWrite er 0, og det har derfor ingenting å si hvilke data som sendes til write data så MemToReg er dont care. Neste instruksjon må også få adressen sin inn i PC, så PCSrc må være 0 slik at adressen blir inkrementert med 4 som er neste instruksjons adresse.

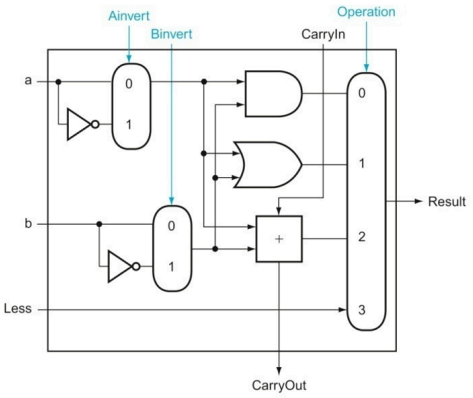
Ord: 124

Maks poeng: 5

3



32-bit ALU



1-bit ALU

Denne oppgaven omhandler den 32-bit brede aritmetisk-logiske enheten (ALUen) vist over (til venstre) som igjen er bygget opp av 1-bit ALUene vist over til høyre. *ALU31* inneholder også logikk for å detektere overflyt, men dette er ikke vist i figuren til høyre.

3.1 ALU kontrollsignaler

ALUen skal utføre operasjonen a - b. Angi korrekte kontrollsignaler.

Angi signalverdier i binær med rett antall bit. Vi gir 1 poeng for riktige svar og -0,25 poeng for feil svar. Ubesvarte oppgaver gir 0 poeng.

Bnegate: 1

Ainvert: 0

Operation: 10

Maks poeng: 3

3.2 Forklare ALU kontrollsignaler

Forklar hvordan kontrollsignalene du har oppgitt i oppgave 3.1 fører til at operasjonen a - b utføres.

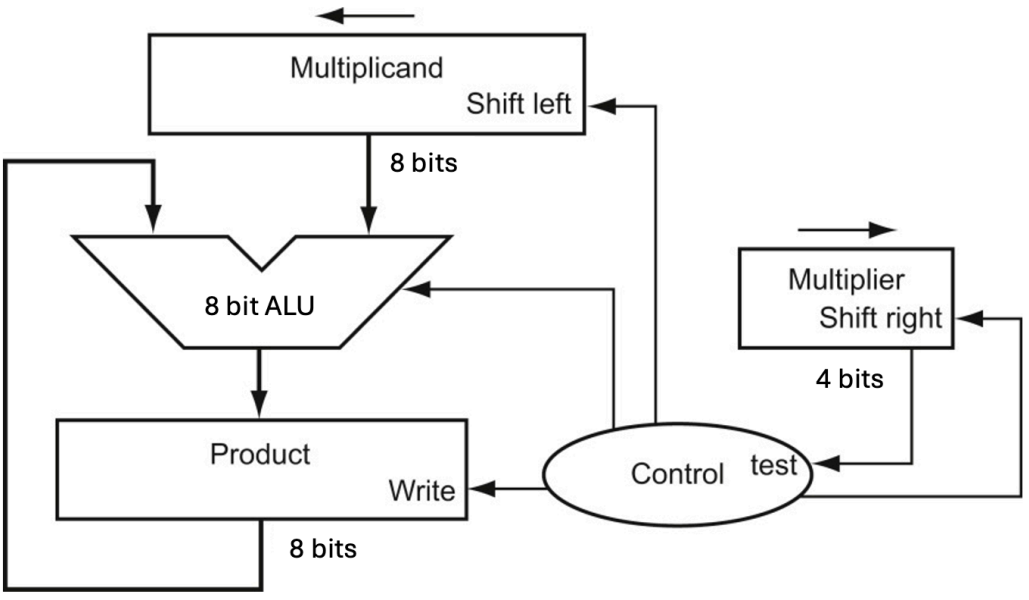
Skriv svaret ditt her. Endringer blir lagret automatisk.

Når jeg referer til ALU så mener jeg 1-bit ALU. Vi kan gjøre et triks. Siden -x = neg(x) + 1 på 2-ers komplement så er a - b = a + invert(b) + 1, så kan vi utføre subtraksjon med kun bruk av addisjon. Da setter vi Bnegate = 1, som gjør at Binvert = 1 inne i hver ALU og den vil da velge invert(b) som verdi for operasjonen. Den setter også carry biten for første ALU slik at vi også får +1. Vi skal ikke invertere a så vi setter Ainvert = 0 slik at vanlig a velges. Operasjonen vi skal utføre er en add, så vi setter Operation = 10 = 2 for å velge outputen fra adderen. Da ender vi opp med at hver bit fra de to tallene sendes inn i hver sin ALU som outputer hver sin bit som en del av svaret. Aluene er også koblet sammen men en carry bit som er outputet av adderen hvis den overflyter, siden 0+1+1 = 10 som er større enn 1 bit og må derfor sendes til neste. Dette gjelder alle utenom første som bruker Bnegate som input og siste som outputer carry bit som global overflyt, så overflyt for 32-bits operasjonen.

Ord: 207

Maks poeng: 4

4



Denne seksjonen omhandler multiplikatoren over.

4.1 Verdier i produktregisteret

Multiplikatoren skal gange de binære tallene 1010 og 1001 med hverandre, altså utføre operasjonen 1010 x 1001. Angi verdien som *Product*-registeret inneholder etter hver iterasjon.

Angi registerverdiene i binær med rett antall bit. Vi gir 1 poeng for hvert riktige svar. Feil svar/ubesvart gir 0 poeng.

Verdien i *Product* etter første iterasjon: 00001010

Verdien i *Product* etter andre iterasjon: 00001010

Verdien i *Product* etter tredje iterasjon: 00001010

Verdien i *Product* etter fjerde iterasjon: 01011010

Maks poeng: 4

4.2 Forklare multiplikatoren

Forklar hvordan du kom frem til svaret i oppgave 4.1.

Skriv svaret ditt her. Endringer blir lagret automatisk.

For hver iterasjon sjekker vi om minst signifikante bit er 1 i multiplier, hvis den er det så legger vi multiplikand til produktet med ALU. I tillegg shifter vi multiplicand 1 bit til venstre og multiplier en bit til høyre for hver iterasjon, uavhengig om hva multiplier er. Da ender vi opp med dette:

Første iterasjon: legg til siden multiplier = 100[1], produkt = 0000 0000 + 0000 1010

Andre iterasjon: gjør ingenting siden multiplier = 010[0]

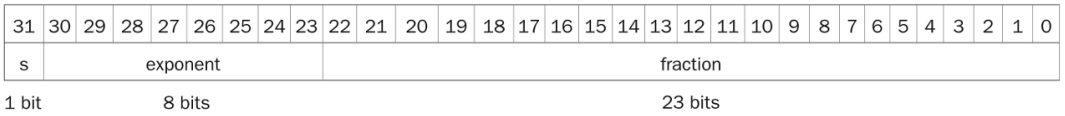
Tredje iterasjon: gjør ingenting siden multiplier = 001[0]

Fjerde iterasjon: legg til siden multiplier = 000[1], produkt = 0000 1010 + 0101 0000

Ord: 100

Maks poeng: 4

5



$$(-1)^{s} \cdot (1+\text{Fraction}) \cdot 2^{(\text{Exponent}-127)}$$

Denne seksjonen omhandler flyttall som følger formatet definert over.

5.1 Flyttall til desimaltall

Hvilket desimaltall representerer flyttallet *0x40980000*? Feil svar/ubesvart gir 0 poeng.

Svar: 4.75

Maks poeng: 2

5.2 Forklare flyttall

Forklar hvordan du kom frem til svaret i oppgave 5.1

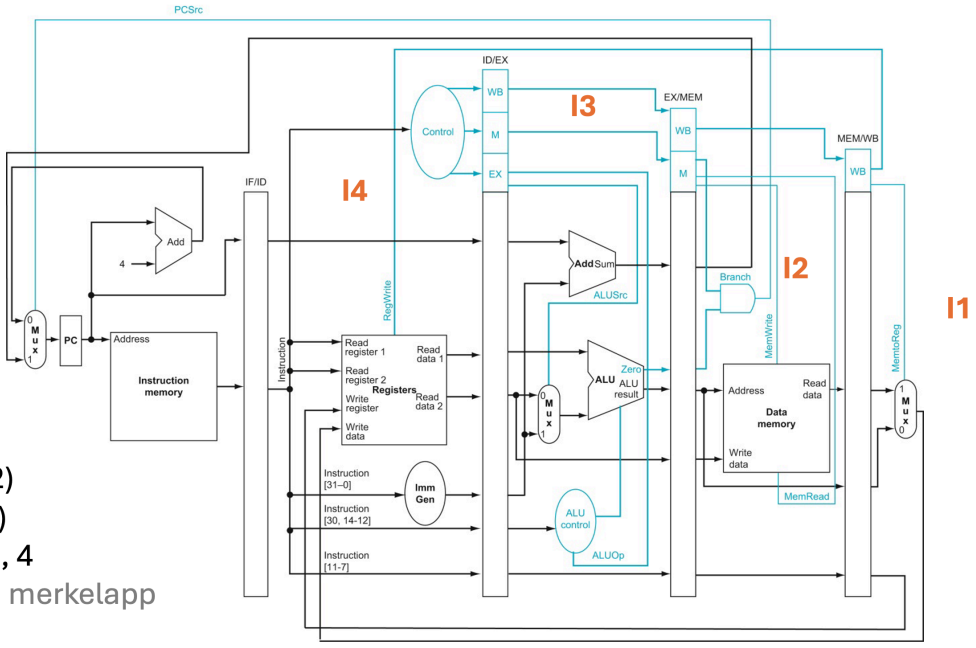
Skriv svaret ditt her. Endringer blir lagret automatisk.

Først konverterte jeg til binær, som ble: 0100 0000 1001 1000 0000 0000 0000 0000. Deretter regnet jeg ut sign, fraction og exponent. Sign = 0, Exponent = 129. Fraction = 1/8 + 1/16 = 3/16. Til slutt er det bare å plugge inn i formelen: -1^0 * (1+(3/16) * 2^(129-127) = 19/16 * 2^2 = 19/4 = 4,75

Ord: 59

Maks poeng: 6

6.1 Samlebånd, kontrollsignaler



merkelapp:

I1: sw x1, 0(x2)
I2: lw x1, 4(x2)
I3: addi x3, x2, 4
I4: beq x4, x5, merkelapp

Figuren over viser en spesifikk klokkesykel i utføringen av et program på en samlebåndsprosessor. Hva er verdiene til følgende kontrollsignaler i denne klokkesykelen?

Du må bruke "don't care" (X) når du kan. Vi gir 0,5 poeng for riktige svar og -0,25 poeng for feil svar. Ubesvarte oppgaver gir 0 poeng.

ID/EX ALUSrc: 1

ID/EX MemWrite: 0

ID/EX MemRead: 0

ID/EX RegWrite: 1

EX/MEM MemWrite: 0

EX/MEM MemRead: 1

EX/MEM RegWrite: 1

EX/MEM MemtoReg: 1

MEM/WB RegWrite: 0

MEM/WB MemtoReg: X

Maks poeng: 5

6.2 Forklare samlebånd

Hvordan sørger kontrollsignalene for at programmet i oppgave 6.1 utføres korrekt? Forklaringen skal dekke alle kontrollsignaler unntatt *ALUOp* for instruksjonene I1, I2, og I3.

Skriv svaret ditt her. Endringer blir lagret automatisk.

Antar at jeg bare skal forklare de resterende stegene for I1, I2 og I3 og ikke de stegene som tidligere er utført før tilstanden som er i 6.1

Ingen av instruksjonene skal branche (ser ikke på I4), så Branch=0 slik at PCSrc alltid blir 0, og pc inkrementeres da med 4 for hver sykel slik at vi får neste instruksjon. Egentlig bare relevant for I2 og I3, siden I1 er forbi branching logikken.

I1 er egentlig ferdig når den har kommet til WB og skal ikke skrive noe RegWrite=0, det har derfor ingenting å si hva vi velger som input for write data siden det ikke blir skrevet, og MemToReg er derfor don't care.

I2 skal lese fra minne i MEM så MemRead = 1, men ikke skrive så MemWrite=0. Adressen kommer fra ALU i forrige steg. I WB skal verdien som ble lest skrives til et register, så RegWrite = 1 og MemToReg må være 1 for at verdien som ble lest fra minne brukes og ikke resultat fra ALU, som er adressen til dataen som ble hentet.

I3 skal legge til et register med immidiate i EX. Første register rs1 sendes til ALU uansett men ALUSrc må være 1 for å velge immidiate verdien som andre operand for ALU. I MEM skal instruksjonen ikke gjøre noe, siden den verken skal lese eller skrive til minne så MemWrite=0 og MemRead=0. Til slutt i WB skal verdien skrives til register, da må RegWrite være 1 og MemToReg=0 for at resultatet fra ALU i EX skal velges som det som skal skrives til register.

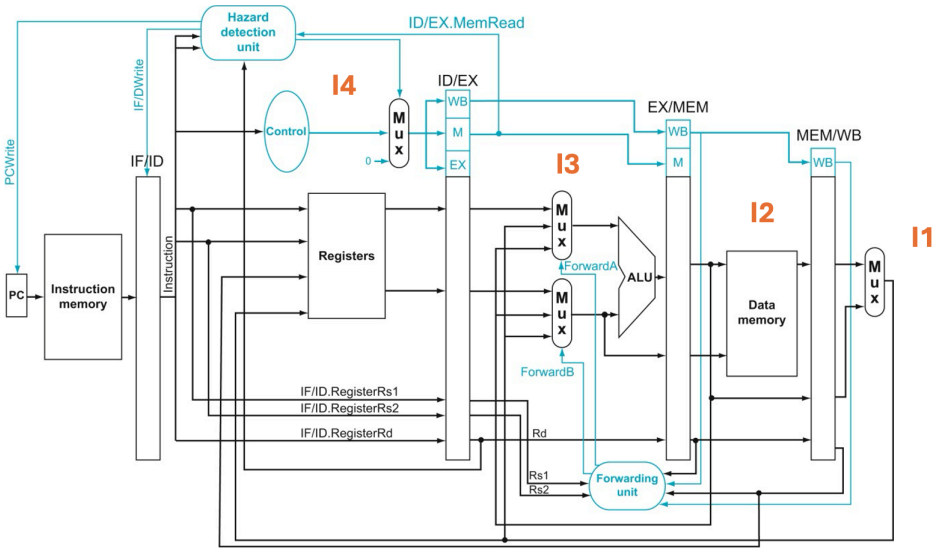
Det er ingen RAW-farer mellom I1, I2 og I3, så alle vil utføres korrekt selv om det ikke er videresending

Ord: 283

Maks poeng: 6

6.3 Korrekt utføring

I1: add x5, x1, x2
I2: add x6, x5, x2
I3: add x7, x5, x6
I4: add x8, x7, x2



Figuren over viser en ny klokkesykel for et annet program på en annen prosessor. Hvordan sørger prosessoren for at instruksjon I3 utføres korrekt?

Skriv svaret ditt her. Endringer blir lagret automatisk.

Det er RAW-farer mellom I3 og både I2 og I1, siden I3 trenger både x5 og x6. Det er heldigvis ikke noen les-bruk farer, så det holder med videresending for å løse problemet og Hazard detection unit trenger ikke å sette inn noen hull. Instruksjon I2 har allerede fått videresendt svaret fra I1 når den var i MEM for å kunne regne ut sitt svar. Når I3 skal regnes i EX så sendes svaret fra I2 for register x6 når den er i MEM og svaret fra I1 for register x5 når den er i WB. Forwarding unit er ansvarlig for videresending og sjekker om rs1 eller rs2 for instruksjonen som er i EX matcher rd (skriveregister) for en av instruksjonene i MEM eller WB. Så den oppdager da at rs1 (x5) for I3 matcher rd for I1 som er i WB, den setter da ForwardA = 1 slik at muxen velger verdien som kommer fra WB som første input for ALU og ikke den gamle verdien som ligger i registerfilen. Den oppdager også at rs2 (x6) matcher rd for I2 som er i MEM, den setter da ForwardB=1, slik at muxen velger verdien som kommer fra MEM som andre input for ALU.

Ord: 203

Maks poeng: 5

7.1 Adresseformat

Anta et 2-veis settassosiativt hurtigbuffer med en kapasitet på 256 KiB og en blokkstørrelse på 128 byte. Minneadressene er 32 bit lange, og minnet er byte-adresserbart. Hvilke bit i adressen brukes til offset, indeks, og tag?

Bitposisjonen til den minst signifikante biten er 0. Vi gir 0,5 poeng for rett svar og 0 poeng for feil svar/ubesvart.

	Minst signifikante bit	Mest signifikante bit
Offset	0	6
Indeks	7	16
Tag	17	31

Maks poeng: 3

7.2 Forklare adresseformat

Forklar hvordan du kom frem til adresseformatet i oppgave 7.1.

Skriv svaret ditt her. Endringer blir lagret automatisk.

For å finne antall bits du trenger for offset tar du log2(128) = 7. Offset er alltid minst signifikante biter. Deretter regner du ut hvor mange sets basert på størrelsen til minne, siden total størrelse / størrelse per set = antall sets: 256*2^10 / (128 * 2) = 2^10. Det er 2^10 sets, og man trenger da 10 bits for å skille mellom disse. Indeks vil utgjøre de minst signifikante bitene som ikke er brukt for offset. Resten av adressen brukes til tag.

Ord: 83

Maks poeng: 6

7.3 Aksesstid

Enhet	Latens (klokkesykler)	Bomrate
Nivå 1 hurtigbuffer	3	15%
Nivå 2 hurtigbuffer	10	30%
Minnet	75	-

Hva blir gjennomsnittlig minneaksesstid for systemet beskrevet i tabellen over?

Svar: 7.875

Maks poeng: 2

7.4 Forklare aksesstid

Forklar hvordan du kom frem til svaret i oppgave 7.3.

Skriv svaret ditt her. Endringer blir lagret automatisk.

Finner først praktisk latens, siden du først bommer i hurtigbuffere på lavere nivå. Niv1 = 3. Niv2 = 13. Minnet = 88. Deretter må du gange med hvor ofte den skal aksessere noe fra det spesifikke minnet. Niv1=85% (treffraten). Niv2 = 15% (bomraten til Niv1) * 70% (treffraten) = 10,5%. Mem = 15%(bom i niv1) * 30% (bom i niv2) = 4,5%. Du kan legge sammen alle prosentene og sjekke at du får 100% som en dobbeltsjekk: 85 + 10,5 + 4,5 = 100. Det siste som gjenstår er å vekte alle aksesstidene og legge sammen: 3*0,85 + 13*0,105 + 88*0,045 = 7,875

Ord: 103

Maks poeng: 4

7.5 TLB

Hva gjør et "Translation Lookaside Buffer (TLB)", og hvorfor er de viktige i datamaskiner som støtter virtuelt minne?

Skriv svaret ditt her. Endringer blir lagret automatisk.

En TLB er en form for hurtigbuffer som ligger nærmt prossesoren. Den er ansvarlig for å oversette raskt fra virtuelle adresser til fysiske adresser. TLB er som oftest full-assosiativt for å øke treffraten, altså den har ingen indeks. Dette gjør det også lett å oversette adresser siden det da ikke er noe indeks i adresseformatet, så den trenger bare å oversette sideadressen/taggen i adressen.

TLB er viktige i datamaskiner som støtter virtuelt minne fordi minne i datamaskiner er tregt. Og hvis hver eneste operasjon på minne krevde at den først leste sidetabellen fra minnet, og så oversatte, så hadde det blitt ekstremt tregt

Ord: 103

Maks poeng: 4

8.1 Register renaming

Hva gjør "register renaming" og hvorfor brukes denne teknikken i prosessorer med høy ytelse?
Skriv svaret ditt her. Endringer blir lagret automatisk.

Register renaming endrer registernavnene i instruksjonene. Dette krever at vi har mange flere registere enn det som er synlig på programvarenivå. Prosessoren har en tabell med oversettelser og hver gang det skrives til et register så legges det til en ny oversettelse slik at alle fremtidige instruksjoner leser fra det registeret som instruksjonen skrev til (helt til det skrives på nytt til registeret). Grunnen til at vi gjør dette er for å kunne gjøre intruksjoner out of order (OOO). For eksempel kan vi kjøre minneinstruksjoner og ALU i parallell for å slippe å hele tiden vente på minnet, som er ønskelig i prosessorer med høy ytelse. Alle instruksjoner fullfører til slutt i den rekkefølgen de skal for å støtte presise unntak. Å kjøre instruksjoner i parallell gjør at vi må tenke på datafarer som WAW og WAR selv om de ikke er sanne datafarer. Å fullføre i riktig rekkefølge løser WAW farer. Register renaming løser RAW og WAR farer sammen med videresending. Uten register renaming kunne vi ha risikert å videresende feil verdi med OOO.

Ord: 175

Maks poeng: 4

8.2 SISD og SIMD

Hva er den viktigste forskjellen på en SISD og en SIMD maskin? Nevn en fordel og en utfordring med SIMD-maskiner.

Skriv svaret ditt her. Endringer blir lagret automatisk.

Den viktigste forskjellen er at SISD utfører operasjonen på ett dataelement, for eksempel å adde to registere. SIMD utfører en instruksjon på flere dataelement, som for eksempel 4 add samtidig på totalt 8 dataelement. SIMD er da en måte å kjøre flere like SISD instruksjoner i parallel. SIMD er derfor veldig nyttig hvis du skal gjøre mange like enkle operasjoner, som i grafikk prosessering, og er grunnen til at GPU bruker SIMD. Siden enkle instruksjoner ofte er raske på ALU og lignende, så en stor del av kjøretiden brukt på fetching, dekoding osv. SIMD trenger da bare å dekode/fetche en instruksjon som er en stor fordel. En ulempe med SIMD er at mye kontrollflyt skaper problemer, siden da er det vanskelig å gjøre mange instruksjoner i parallell siden du ikke helt vet hvilke instruksjoner du skal utføre, så programmer med mye sekvensiell logikk som OS egner seg dårlig for bruk av SIMD.

Ord: 152

Maks poeng: 6