Norges teknisk–naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

Faglig kontakt under eksamen:
Magnus Jahre (952 22 309)

EKSAMEN I TDT4160 DATAMASKINER GRUNNKURS

Tirsdag 7. Desember
Tid: 09:00 – 13:00
BOKMÅL

Hjelpemidler:
D.
Ingen trykte eller håndskrevne hjelpemidler tillatt.
Bestemt, enkel kalkulator tillatt.

Bruk den angitte plassen til å svare på oppgavene. Hvis du trenger mer plass, er det satt av ekstra plass på den siste siden i oppgavesettet. Eksamen teller 100% av karakteren i faget, og de oppgitte prosenttallene angir det maksimale antall poeng man kan oppnå på hver oppgave. Les oppgavene grundig.

Kandidatnummer:

Oppgave 1 Multiple Choice (30%)

Svar ved å ringe rundt ønsket svaralternativ. Du får 3 poeng for riktig svar og 0 poeng hvis du avstår fra å svare. Hvis du svarer feil eller ringer rundt mer enn ett alternativ, får du -1 poeng.

a) (3%) Hvilken påstand om en prosessor er *ikke* riktig?

1. En generell prosessor trenger bare instruksjoner for aritmetiske operasjoner og betingede hopp
2. En prosessor er ofte delt inn i en utførende enhet og en kontrollenhet
3. Hovedoperasjonen til en prosessor kan beskrives som en fetch-decode-execute løkke
4. Prosessorer bruker ofte parallellitet for å øke ytelsen

Riktig svar: Alternativ 1

b) (3%) Hvilken av disse påstandene om RISC-maskiner er *ikke* korrekt?

1. RISC-maskiner har mange generelle registre
2. RISC-maskiner har få og enkle instruksjonsformater
3. RISC-maskiner aksesserer kun minne gjennom Load- og Store-instruksjoner
4. RISC-maskiner ble oppfunnet for å tette gapet mellom høynivåspråk og maskinkode

Riktig svar: Alternativ 4

c) (3%) Hva er det viktigste kjennetegnet ved en Von Neumann-arkitektur?

1. Arkitekturen mangler flyttallsenhet fordi programmereren bør kunne holde styr på komma
2. Arkitekturen har et hurtigbuffer til data og et til instruksjoner
3. Både program og data er lagret i samme minne
4. Von Neumann-maskiner er alltid CISC

Riktig svar: Alternativ 3

Kandidatnummer:

d) (3%) Hvilken av disse påstandene er *ikke* en grunn til at man har begynt å bygge flerkjerneprosessorer (eng: Chip Multiprocessor, multi-core architecture)?

1. Det er vanskelig å øke ytelsen ytterligere ved hjelp av teknikker som utnytter ILP
2. Avanserte enkjerneprosessorer er så kompliserte at det å designe og verifisere dem utgjør en betydelig kostnad
3. Det er vanskelig å designe enkjerneprosessorer med høy ytelse og et akseptabelt effektforbruk
4. Når man plasserer flere prosessorer på samme brikke, senker man behovet for minnebåndbredde

Riktig svar: Alternativ 4

e) (3%) Hvilken av disse påstandene om flerkjerneprosessorgenerasjoner (eng: Chip Multiprocessor generations) er *ikke* korrekt?

1. Første generasjon med flerkjerneprosessorer er det beste valget hvis du trenger rask kommunikasjon mellom prosessorkjernene
2. Andre generasjon med flerkjerneprosessorer deler hurtigbuffer mellom prosessorkjernene
3. Tredje generasjon med flerkjerneprosessorer prioriterer throughput fremfor høy ytelse per tråd
4. Tredje generasjon med flerkjerneprosessorer benytter multithreading i prosessorkjernene

Riktig svar: Alternativ 1

f) (3%) Hvilken påstand om et set-assosiativt hurtigbuffer (eng: cache) er riktig?

1. Et dataelement kan lagres på kun en lokasjon i hurtigbufferet
2. Et dataelement kan lagres på alle lokasjoner i hurtigbufferet
3. Et dataelement kan lagres på noen bestemte lokasjoner i hurtigbufferet
4. Set-assosiative hurtigbuffere er lite brukt i moderne prosessorer

Riktig svar: Alternativ 3

Kandidatnummer:

g) (3%) Anta en prosessor med ett nivå hurtigbuffer (eng: cache), en hurtigbufferaksesstid på 3 klokkesykler og en minnelatens på 400 klokkesykler. Hva blir den gjennomsnittlige minneaksesstiden når 95% av forespørslene treffer i hurtigbufferet?

1. 3 klokkesykler
2. 23 klokkesykler
3. 380 klokkesykler
4. 400 klokkesykler

Riktig svar: Alternativ 2

Løsning: $a = 0.95 \cdot 3 + 0.05 \cdot (3 + 400) = 23$

Mystisk Dark stakkmaskinkode:

```
    push 4
a:  dup
    push 2
    lt
    jtrue e
    push 1
    sub
    jmp a
e:
```

Husk at:

- lt: resultat $\leftarrow$ nest øverst $<$ øverst
- sub: resultat $\leftarrow$ nest øverst $-$ øverst

h) (3%) Dark stakkmaskinprogrammet over kjøres på en tom stakk. Hvilken verdi ligger igjen på stakken når programmet har kjørt ferdig?

1. 1
2. 2
3. 3
4. 4

Kandidatnummer:

Riktig svar: Alternativ 1

Mystisk mikrokode:

```
MAR = SP - 1; rd
MAR = SP
H = MDR; wr
MDR = TOS
MAR = SP - 1; wr
TOS = H; goto Main1
```

i) (3%) Koden over implementerer en IJVM-instruksjon for Mic-1. Hvilken?

1. Invokevirtual
2. Swap
3. Bipush
4. Istore

Riktig svar: Alternativ 2

Flyttall på formen $n = f \cdot 2^e$ er kodet med følgende flyttallsformat:

Sign (1 bit)	e (Exponent, 4 bit)	f (Fraction, 4 bit)

I dette formatet er exponent kodet som "excess 8", og alle bit i fraction er kodet eksplisitt.

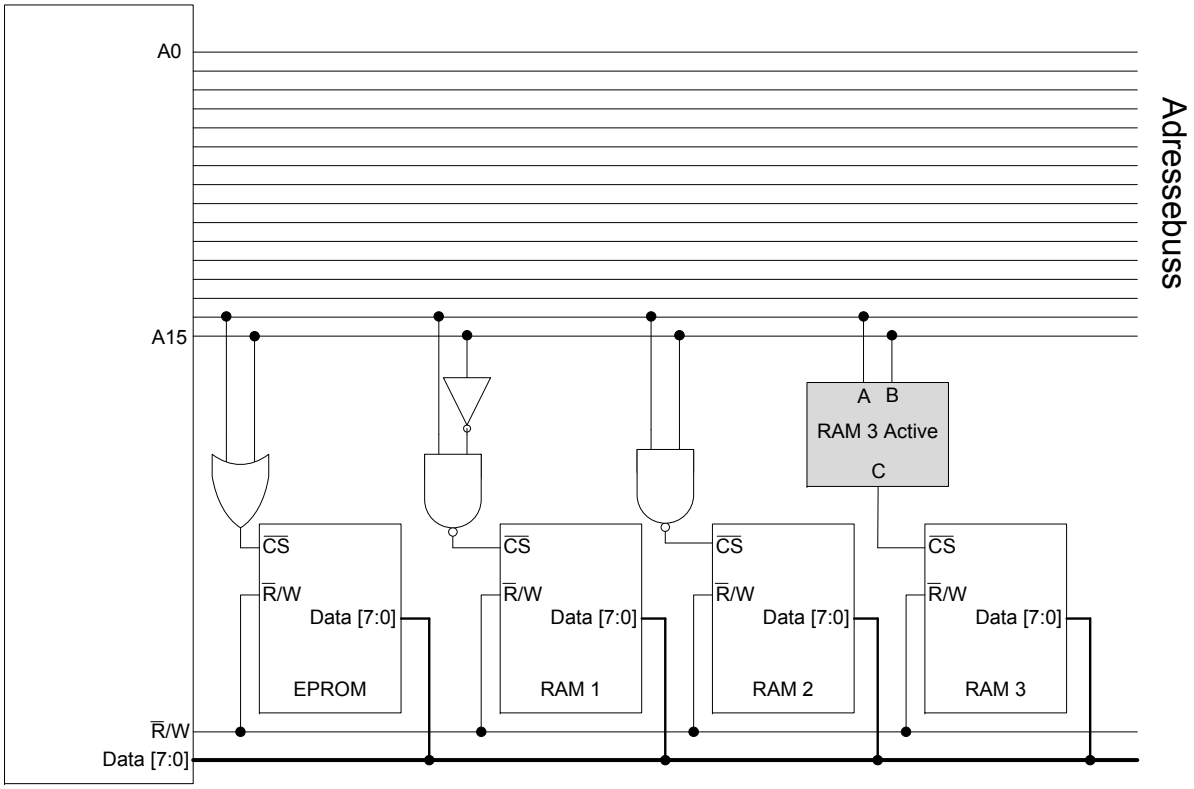
j) (3%) Hva er verdien til flyttallet 001101100 i 10-tallsystemet?

1. 0.09375
2. 0.1875
3. 0.375
4. 0.75

Riktig svar: Alternativ 2

Løsning: $n = (\frac{1}{2} + \frac{1}{4}) \cdot 2^{6-8} = \frac{3}{4} \cdot 2^{-2} = \frac{3}{16} = 0.1875$

Kandidatnummer:



Figur 1: Adressedekoding

Oppgave 2 Digitalteknikk (15%)

Figur 1 viser et blokkdiagram av et system der en EPROM enhet og tre RAM enheter er koblet til en felles adresse- og databuss. EPROM og RAM enhetene har et aktivt lavt (logisk "0") CS (Chip Select) signal.

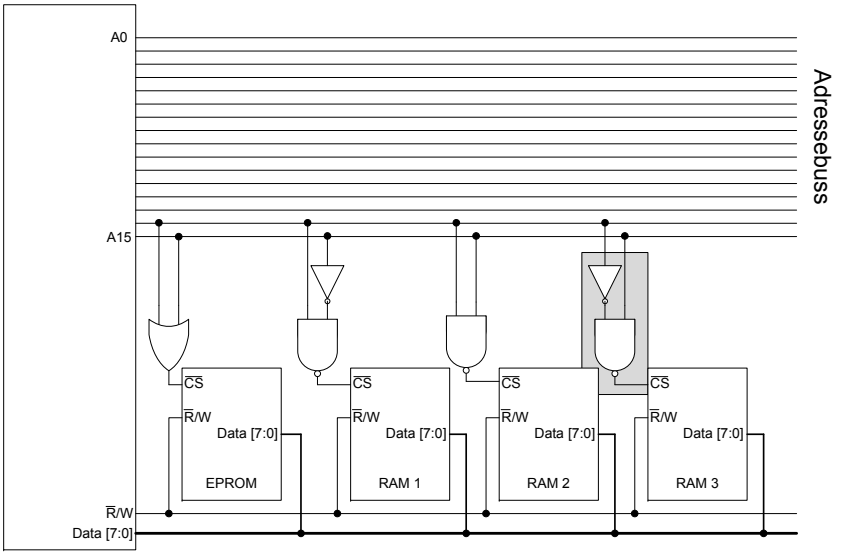
a) (5%) Angi adresserommet der EPROM har kontroll over databussen. Svaret skal oppgis heksadesimalt.

Løsning: EPROM har kontroll over bussen for adresse a hvis 0x0000 $\leq a <$ 0x4000

b) (5%) Angi adresserommet der RAM 1 har kontroll over databussen. Svaret skal oppgis heksadesimalt.

Løsning: RAM 1 har kontroll over bussen for adresse a hvis 0x4000 $\leq a <$ 0x8000

Kandidatnummer:



Figur 2: Løsningsforslag for adressedekoding

c) (5%) Systemet i figur 1 er ikke helt ferdig da logikk for å aktivere RAM 3 mangler. Denne logikken skal inn i komponenten "RAM 3 Active" i figuren. Tegn kombinatorisk logikk for "RAM 3 Active". Systemet skal benytte hele adresserommet.

Løsning: Det ferdige systemet er beskrevet i figur 2. Vi trekker 1p per slurvefeil (f.eks. AND i steden for NAND).

Kandidatnummer:

Oppgave 3 Mikroarkitektur (10%)

Dr. Tufte har sluttet med fotball og ønsker å slå seg opp som mikroprogrammerer for IJVM og Mic-1 (se figur 7). Hans første forsøk er å implementere IJVM ISA-instruksjonen Pop. Pop skal i følge spesifikasjonen fjerne det øverste elementet på stakken og oppdatere TOS registeret. Dr. Tufte har foreslått følgende mikrokode:

```
1. MAR = SP = SP - 1; rd
2. TOS = MDR; goto Main1
```

Figur 4, 5, 6 og 7 i vedlegget kan være nyttige for å løse denne oppgaven.

a) (5%) Fyll ut tabellen under med styreordet for instruksjon nummer 1 i Dr. Tuftes kode. Du trenger ikke å fylle ut NEXT_ADDRESS og JAM feltene.

Løsning:

Next Address	JAM	ALU	C	Mem	B
–	–	00 110110	000001001	010	0100

Her gir vi 1p for riktige shifter bits (to mest signifikante ALU bit), 1p for riktig resten av ALU, 1p for riktig C, 1p for riktig MEM og 1p for riktig B

b) (5%) Koden til Dr. Tufte inneholder en feil. Hva er feilen, og hvordan kan man rette den?

Løsning: Minneoperasjoner tar en ekstra klokkesykel i Mic-1. Derfor er ikke minneaksessen i instruksjon 1 ferdig før instruksjon 2 kjører. Løsningen er å sette inn en NOP mellom instruksjon 1 og 2. Alternativt kan man dele instruksjon 1 opp i to: MAR = SP-1; rd og så SP = SP-1. Begge løsningene gir full pott.

Kandidatnummer:

Oppgave 4 Instruksjonsnivåparallelitet (10%)

Anta et instruksjonssett med 3-adresseinstruksjoner der den første operanden angir destinasjonsregister og de to andre angir inputregistre. For eksempel vil instruksjonen *ADD R1, R2, R3* utføre operasjonen *R1 = R2 + R3*.

```
1. MUL R3, R0, R1
2. ADD R4, R3, R2
3. ADD R3, R0, R4
4. SUB R4, R5, R6
```

a) (5%) Hvilke avhengigheter finnes i assemblykodesnutten over, og hva er navnet på disse avhengighetene?

Løsning: Koden inneholder følgende avhengigheter:
Instruksjon 1 –> 2: R3: Sann dataavhengighet (RAW)
Instruksjon 1 –> 3: R3: Ut-dataavhengighet (WAW)
Instruksjon 2 –> 3: R4: Sann dataavhengighet (RAW)
Instruksjon 2 –> 3: R3: Anti-avhengighet (WAR)
Instruksjon 2 –> 4: R4: Ut-avhengighet (WAW)
Instruksjon 3 –> 4: R4: Anti-avhengighet (WAR)

Vi godkjenner også de tilsvarende hazards WAR, WAW og RAW som riktige svar siden skillet mellom avhengighet og hazard ikke er tydelig i læreboka. Vi gir 1p for hver avhengighet som er korrekt navngitt, men maks 5 poeng (det er 6 avhengigheter). Vi gir inntil 3p for å identifisere avhengigheter men ikke navngi disse korrekt.

b) (5%) Fjern så mange som mulig av avhengighetene ved hjelp av teknikken "register renaming".

Løsning: Ut- og anti-avhengigheter kan fjernes med register renaming. Koden blir da:
1. MUL R3, R0, R1
2. ADD R4, R3, R2
3. ADD S1, R0, R4
4. SUB S2, R5, R6

Oppgave 5 Instruksjonssett (10%)

a) (5%) Hva er forskjellen på en trap og et avbrudd (eng: interrupt)?

Løsning: En trap skyldes en hendelse som er forårsaket av programmet, mens et avbrudd forårsakes av en hendelse utenfor programmet. 2.5 poeng for hver korrekte forklaring, mindre poengsummer kan deles ut etter skjønn.

b) (5%) Forklar baseindeksert adressering (eng: based-indexed addressing). Tegn gjerne en figur.

Løsning: Baseindeksert adressering oppgir to registre og bruker summen av innholdet i disse registrene til å generere minneadressen. Det er også mulig å bruke spesifisere en immediate offset, men vi gir full pott uavhengig av om studenten har tatt med dette eller ikke. Mange har kommet med en flott forklaring av indeksert adressering, men dette gir 0 poeng da det ikke er det vi spør etter.

Oppgave 6 Minnesystemer (15%)

a) (5%) Forklar begrepene "lokalitet i tid" (eng: temporal locality) og "lokalitet i rom" (eng: spatial locality)

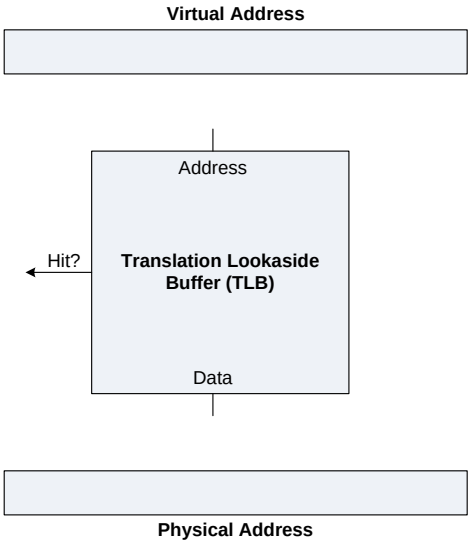
Løsning: Se læreboka s. 294. Maks 2.5p per begrep, tildeles etter skjønn.

b) (5%) Anta et system med 256 MB byteadresserbart hovedminne og 4 KB sidestørrelse. Hvor mange sider (eng: pages) er det plass til i hovedminnet?

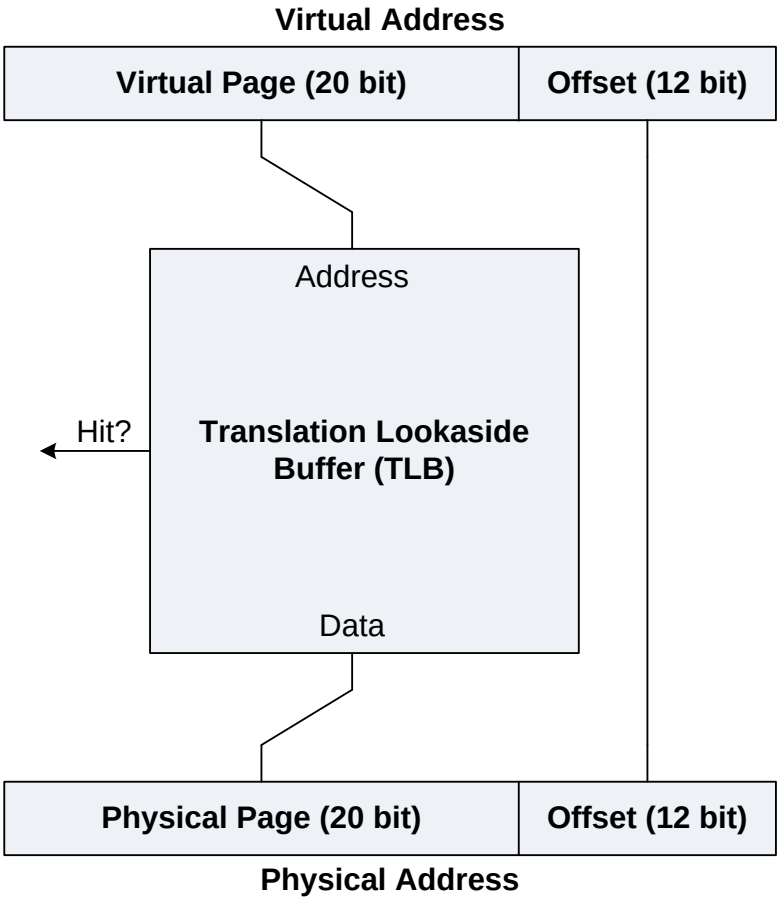
Løsning: $\frac{256 \cdot 2^{10} \text{ KB}}{4 \text{ KB}} = 64 \cdot 2^{10}$ sider. Vi godkjenner også utregning med tierpotenser.

c) (5%) Blokkdiagrammet under viser hvordan en TLB (Translation Lookaside Buffer) kan brukes til å øke hastigheten på oversettelsen fra virtuelle til fysiske adresser. Fullfør tegningen med å velge riktig format på fysisk og virtuell adresse samt rute deladressene til riktige porter. Anta 32 bit fysiske og virtuelle adresser, 4 KB sidestørrelse og byteadresserbart lager.

Løsning: $4\text{KB} = 4 \cdot 2^{10}\text{B} = 2^{12}\text{B}$. Følgelig er page offset 12 bit og fysisk og virtuell page 20 bit. Figur 3 viser det ferdige blokkdiagrammet.



Kandidatnummer:



Figur 3: Løsningsforslag for TLB-oppgaven

Kandidatnummer:

Oppgave 7 Multiprosessorer (10%)

a) (5%) Hvilke kategorier inngår i Flynns taksonomi for parallelle datamaskiner?

Løsning: SISD, SIMD, MISD, MIMD. Se læreboka s. 587.

b) (5%) Fire prosessorer utfører minneoperasjoner og ser operasjonene i følgende rekkefølge:

t	CPU 0	CPU 1	CPU 2	CPU 3
1	Write A	Write A	Write A	Write A
2	Write B	Write B	Write B	Write B
3	Read A	Read B	Read B	Read A
4	Read B	Read A	Read A	Read B

Er maskinvaren "sequentially consistent"? Begrunn svaret.

Løsning: Denne oppgaven ble litt mer finurlig enn det som var planen. Følgelig gir vi full pott for to forskjellige svar:

Løsning 1 (mest riktig): Utføringen er "sequentially consistent" fordi alle prosessorene ser skriveoperasjonene til adresse A og B i samme rekkefølge. Følgelig vil resultatet av utføringen være den samme selv om CPUene ser leseoperasjonene til A og B i forskjellig rekkefølge.

Løsning 2 (riktig ut fra forenklet forklaring gitt i forelesning): Utføringen er ikke "sequentially consistent" fordi CPU 0 og CPU 3 ser leseoperasjonen til A før leseoperasjonen til B mens CPU 1 og CPU 2 ser leseoperasjonen til B før leseoperasjonen til A.

Hvis ikke konklusjonen følger naturlig fra begrunnelsen, gir vi ingen poeng.

Ekstra svarplass

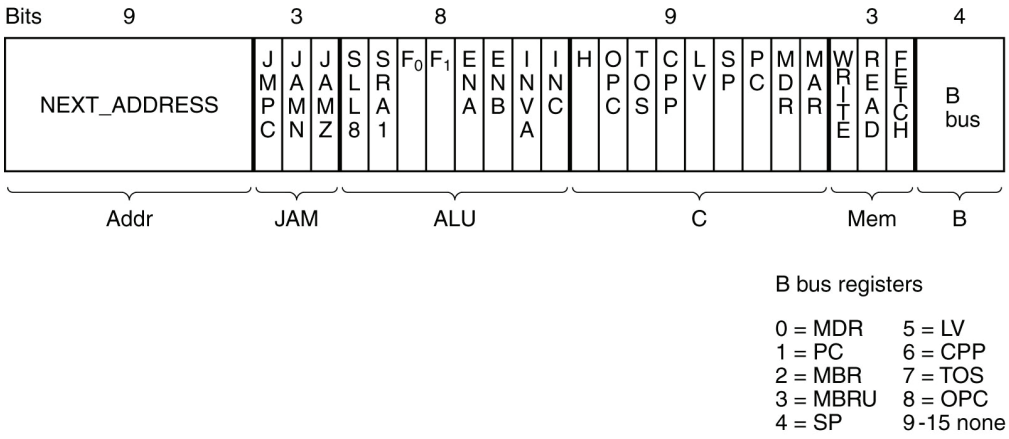
Kandidatnummer:

Vedlegg

Kandidatnummer:

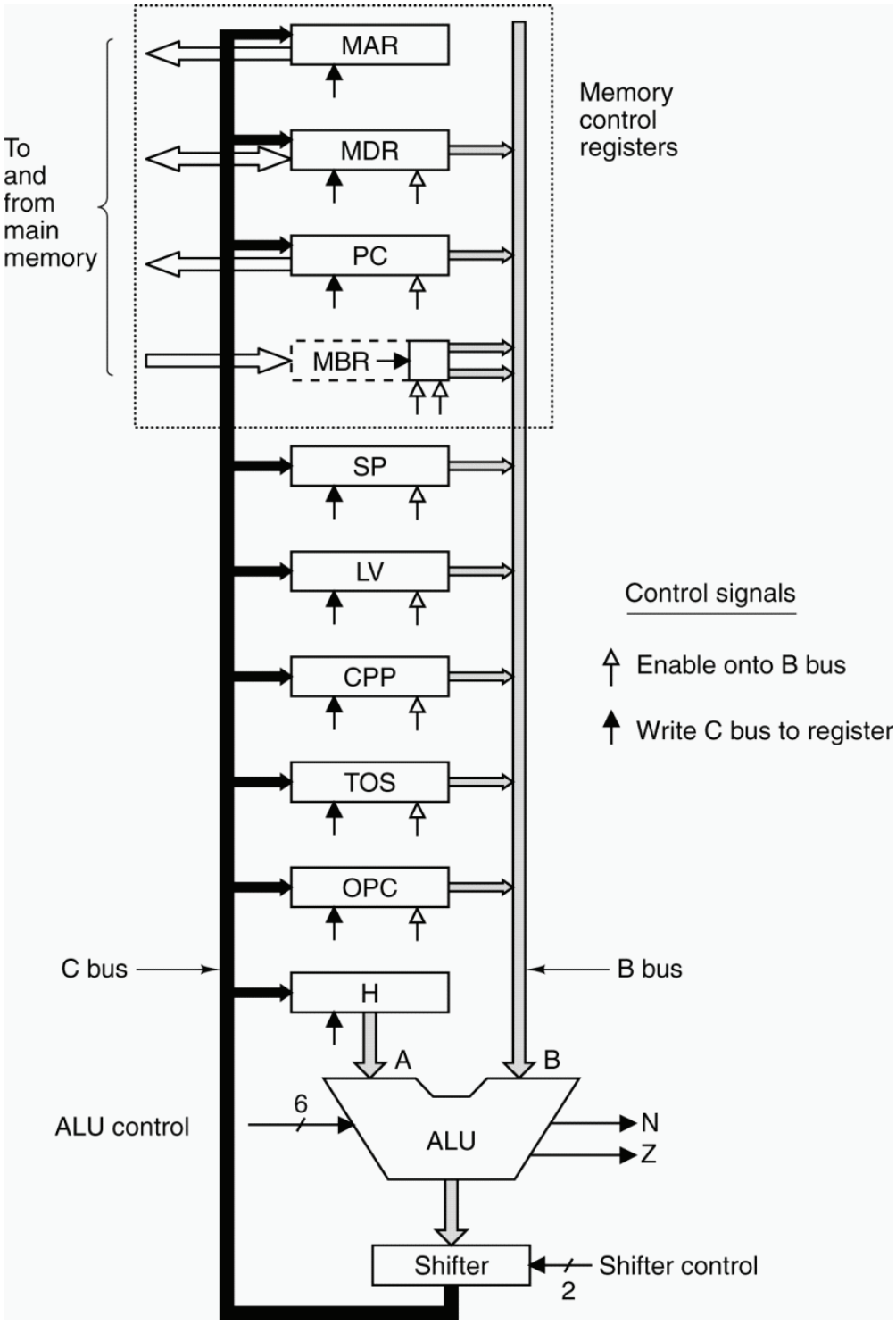
F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\overline{A}$
1	0	1	1	0	0	$\overline{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1

Figur 4: Funksjonstabell for ALU (Mic-1)



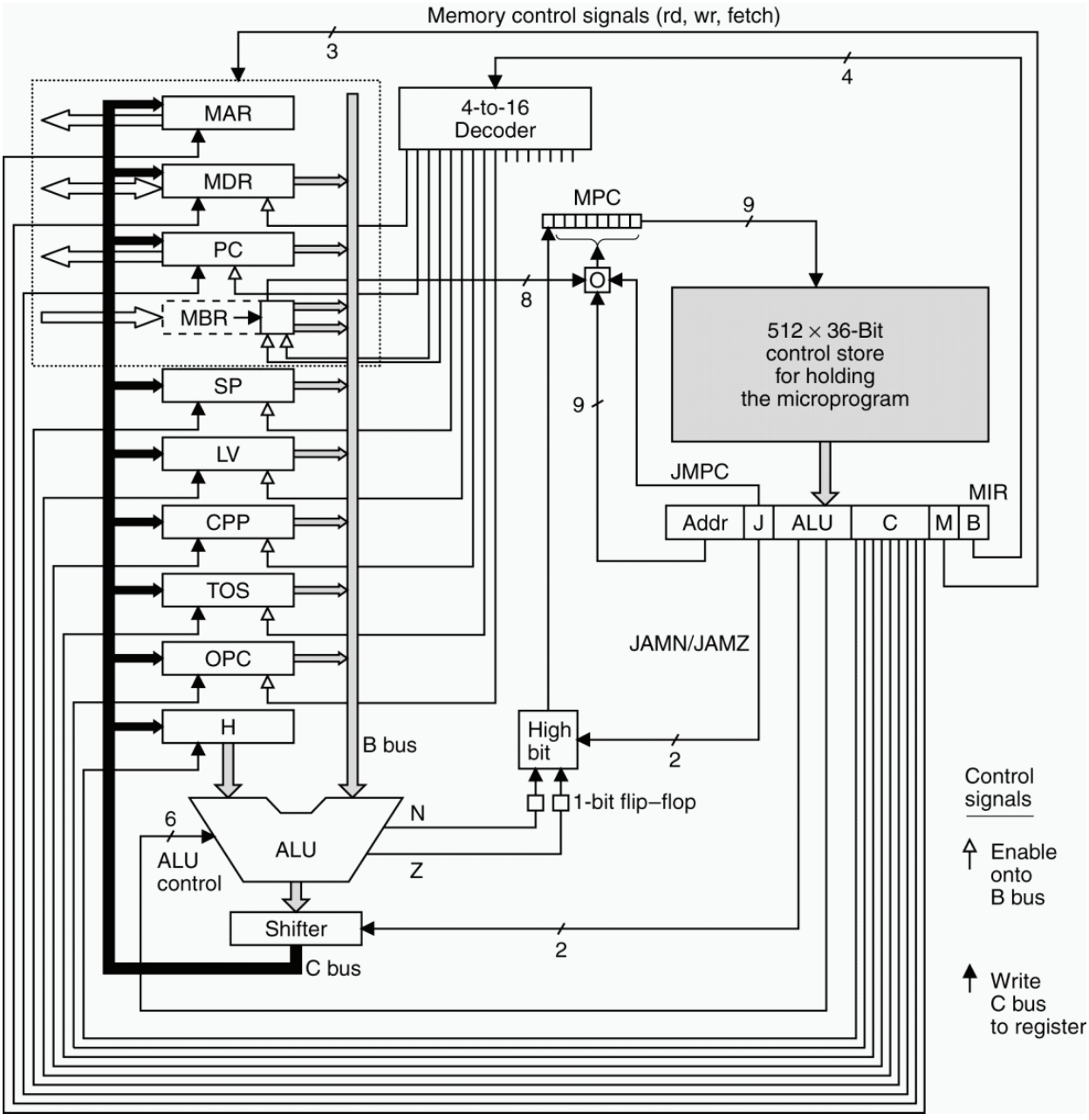
Figur 5: Mikroinstruksjonsformat (Mic-1)

Kandidatnummer:



Figur 6: Utførende enhet (Mic-1)

Kandidatnummer:



Figur 7: IJVM mikroarkitektur (Mic-1)

Kandidatnummer: