

Oppgave 1

a)

i) Register: ALU

ii) Mikroinstruksjonsminne: Control Unit

iii) Hurtigbuffer: Minne

b)

i) Instruksjonsnivå parallelitet:

ILP

Fler instruksjoner skjer i parallel.
Pipeline. Dette skjer i en og samme tidsrom.

Vaskemaskinellimpelot.

ii)

Prosessornivå parallelitet:

Samme instruksjon skjer på en
fler mengder data samtidig.

c) Busy waiting:

Prosessoren vil lese status (feks. i en løp som man må vente på løpen) til VO-enheten.

Prosessoren vil være oppstatt med å vente, kun løpen som kan kjøre imens.

Interrupt Request (IRQ):

IRQ

VO-enheten kan gi et IRQ-signal til prosessoren for å si utført oppgaven. Prosesoren trenger ikke å overvåke i software.

SW

Koden som skal kjøres ved IRQ, utføres først når signalet gis. Prosesoren kan kjøre annen kode når den ikke håndterer I/O.

d) Lokalitet i tid og rom:

Tid: Data/inntut du nylig har brukt vil de sannsynligvis bruke igjen, så dette plasseres i cache.

Rom: Data/inntut i nærmestebaner av det du nylig har brukt vil sannsynligvis adresseres, - plasserer hel blokk med data i cache. F. dvs. lister.

e)

CMP : En chip med multiple prosessorer.

Homogen Multi Prosessor:

Alle prosessorme på chippen er like.

Heterogen Multi Prosessor

Ulike prosessorer på samme chip

Oppgave 2

F ₀	F ₁	F ₂	Out	Kommentar
0	0	0	0	Logisk 0
0	0	1	Full AND	
0	1	0	A \oplus B	
0	1	1	A AND B	
1	0	0	A OR B	
1	0	1	A NAND B	
1	1	0	\sim A	
1	1	1	A	

i) RAM:

001x xxxx xx xx xxxx
[2000, 3FFF]

ROM:

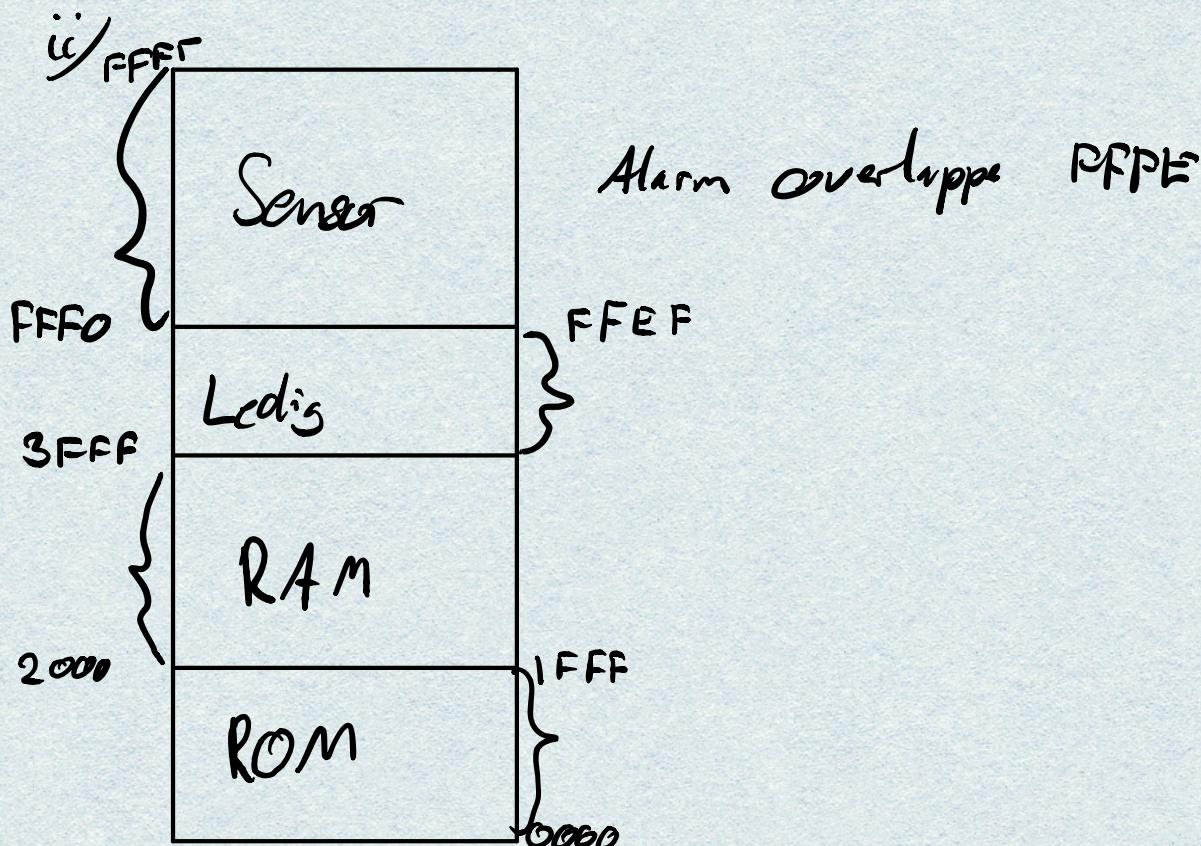
000x xx xx xx xx xx
[0000, 1FFF]

Sensor:

1111 1111 1111 xxxx
[FFFO, FFFF]

Alarm:

1111 1111 1111 1110
[FFFF]



iii)

Jn. Alarm og sensor brikker henholdsvis
w og r hver for seg, - ingen konflikt.

$$D_0 = (\overline{Q_2} \wedge \overline{Q_1} \wedge \overline{Q_0} \wedge OK) + (\overline{Q_2} \wedge Q_1 \wedge \overline{Q_0} \wedge OK)$$

$$D_1 = (\overline{Q_2} \wedge Q_1 \wedge \overline{Q_0} \wedge OK) + (\overline{Q_2} \wedge \overline{Q_1} \wedge Q_0 \wedge OK)$$

$$D_2 = (\overline{Q_2} \wedge Q_1 \wedge Q_0 \wedge OK)$$

$$Q_0 = D_0 \quad y = \overline{Q_0} \wedge \overline{Q_1} \wedge Q_2$$

$$Q_1 = D_1$$

$$Q_2 = D_2$$

State: Q_2, Q_1, Q_0

			NxtOK=1			NxtOK=0		
Q_0	Q_1	Q_2	Q_0	Q_1	Q_2	Q_0	Q_1	Q_2
0	0	0	0	0	1	0	0	0
0	0	1	0	1	0	0	0	0
0	1	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0



d) Dette er en Moore-maskin. State basert på current state.

Oppgave 3:

a)

MAR:

Memory Address Register. Her ligger adressen som peker på dataminne.

MDR:

Memory Data Register. Holder på data som skal tilgå fra minne.

PC:

Program Counter. Peker til instruksjonsminnet.

MBR:

Memory Buffer Register. OpCode fra instruksjonsminnet. (Fra pekkeren i PC).

b)

1 instruksjon:

ALU: A (lt-reg), C: OPC, TOS, CPP,

MEM: ade, B: DC

c) Velger å invertere og andre. Om resultatet
= 000...0, så er de like.

1. ALU: $\sim B$, C: H, MEM: 000, B: LV
2. ALU: A and B, C: 0, MEM: 000, B: MDR

Z-flagget blir sett på (I) dersom da
es like.

d) Operatordelen er 8 bit. Det 9.-ende littet
sier noe om branching.

Dette littet manipuleres også blant annet
N- og Z-flagg

Dette gir to alternative programstier.

e) 512 plasser for å holde på mikroprogrammer
med styrrelse 36-bit.

Oppgave 5

a)

Risc:

- Enkle instruksjoner
- Load/store- arkitektur
- Mange generelle register.

b)

MOV. Operand er gitt som verdi/konstant.

Rt og Nop har ingen adresse.

c) 32
2

d)

• Load R₁, R₈

Legger data R₈ peker på inn i R₁.

R₁ = 0x00 00 00 55

• Load R₂, R₉

Legger data R₉ peker på inn i R₂.

R₂ = 0x00 00 00 A4

- Store R₁, R₂

Legger data fra R₁ på lokasjonen R₂ peker på

0x00 00 00 55 legges på adresse 0x00 0000 A1

- ADD R₁, R₂, R₈

Adder data i R₁ og R₈, lagre i R₁

$$\begin{aligned}R_1 &= 0x00 00 00 55 + 0xFFFF 0000 \\&= 0xFFFF 0055\end{aligned}$$

- Stør R₁, R₈

Legger data fra R₁ på lokasjonen R₈ peker på

0xFFFF 00 55 legges på adresse 0xFFFF 0000

- Load R₁, R₂

Legger data R₂ peker på inn i R₁.

$$R_1 = 0x00 00 00 55$$

- MOVC R₁₆, 0x0055

Ligg konstanten 55 i R₁₆

$$R_{16} = 55$$

- ADD R₁, R₁, R_{1b}

Adder data i R₁ og R_{1b}, lagre i R₁

$$R_1 = 55 + 55 = AA$$

- CMP R₁, R₂

Sjekker om R₁ og R₂ er like. Setter z-flagg
om de er like.
z-flagg settes.

- BEZ R₁

Hopp til instruksjonsminne gitt i R₁

Hopp til 0x60 00 00 AA i PC.

$$R_1 = \underline{\underline{00 \ 00 \ 00 \ AA}}$$

Oppgave 5

a) $\overline{\text{AKSESSTID}} = c + (1-h) \cdot m$

c = cache access time

h = hit ratio

m = memory access time

$$\overline{\text{AKSESSTID}} = 1 \text{ ns} + (1 - 0,8) \cdot 100 \text{ ns} = \underline{\underline{36 \text{ ns}}}$$

b) 1) Trøgesk steg tar 15 ns.

$$1 / 15 \text{ ns} \cdot 1000 \text{ Hz} = \underline{\underline{66,67 \text{ MHz}}}$$

2.) Trøgeste steg tar ni 7,5 ns.

$$1 / 7,5 \text{ ns} \cdot 1000 \text{ Hz} = \underline{\underline{133,33 \text{ MHz}}}$$

3) Klokkefrekvensen uten pipeline blir summen av alle instruksjonene. tot: 37,5 ns

$$1 / 37,5 \text{ ns} \cdot 1000 \text{ Hz} = 26,667 \text{ MHz}$$

4)

Programflyt:

Pipelinen må tömmes om man hopper, om en prediction kommer fylles pipelinen med feil. programstid og operander.

Avhengigheter:

Jo flere avhengigheter, desto anklæs at pipelinen står.

C)

det ytelse:

- innfør A-Gus.
- innfør multimedie cache.
- innfør IFU.