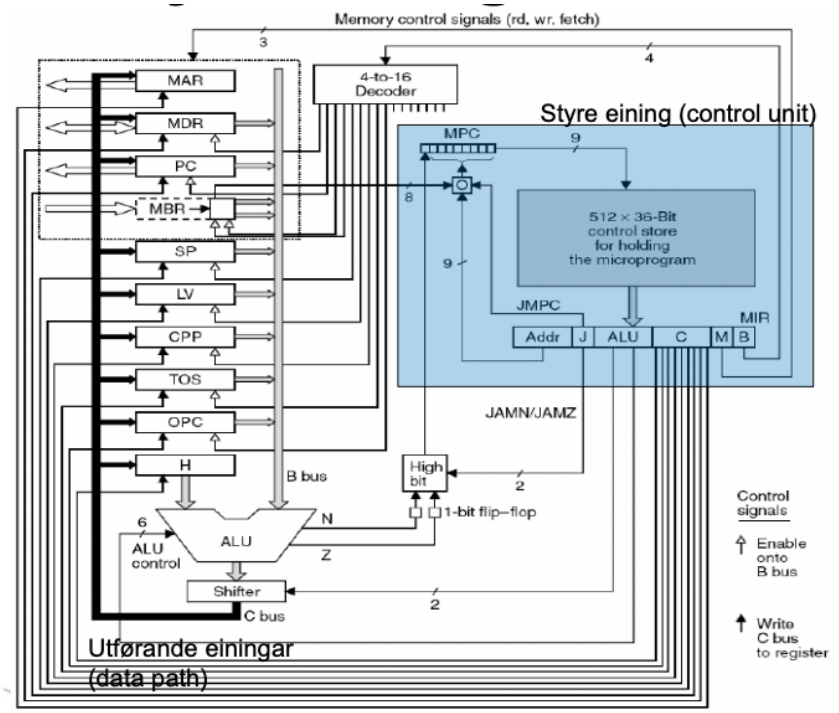
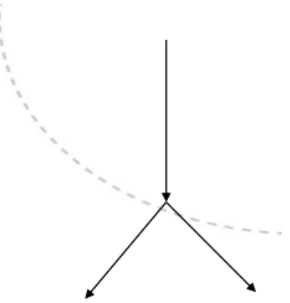
Week 10

Kontroll og programflyt
Betyr i realiteten "hvilken adresse skal velges".
Vi trenger kun å gå gjennom betinget hopp instruksjoner igjen for å ha IJVM som en generell datamaskin.¨

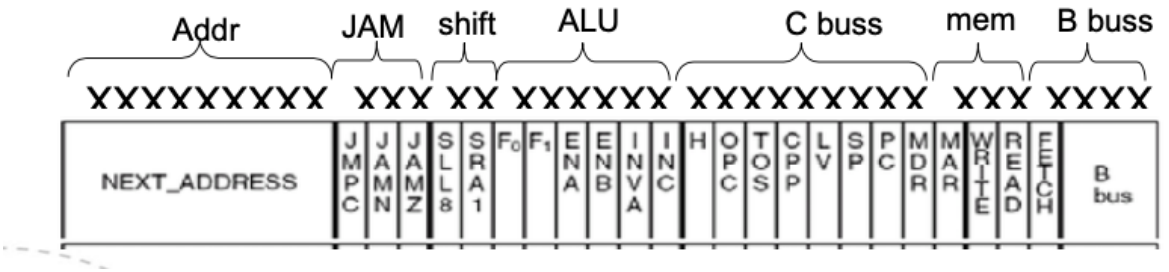
Hvordan kan et program velge forskjellige sier i utførelsen sin? Vi har if/else i høynivå kode, men datamaskinen må ha mulighet til å gjøre noe tilsvarende.





I IJVM så er det control unit som kommer til å støtte dette. Det vi si at control unit må være i stand til å velge mellom to forskjellige programmstier. Nå må kunne gjøre en test og få control unit må sette opp datapathen slik at vi kan velge mellom forskjellige programstier.

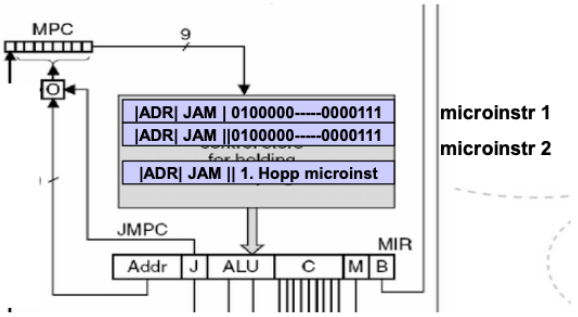
- **Status register**
  - **Register der bit gir status til einingar eller av resultat av instruksjon**
  - **Bit i status register: flagg**
- **IJVM status**
  - JAMZ: Sjekkar Z-flagget (zero) til ALU
  - JAMN: Sjekkar N-flagget (negativ) til ALU
  - JMPC: Last MBR inn i MPC ved JMPC = 1, elles MPC = Next_Address
- No mogleg å detektera N og Z for å kunne bestemme hopp
  - Kan sjekke resultat frå beregning utført av ALU
  - Kan bruke resultatet til betingahopp

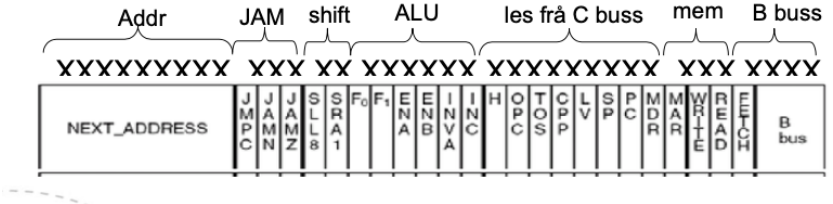


I IJVM så har bitene JAM i MIR. Disse forteller om vi enabler et kontrollhopp

- JMPC: bestemmer om vi skal kopiere innholdet i MBR inni MPC, eller om vi skal kopiere next_adress fra MIR i MPC
- JAMZ: et flag fra ALU. Blir satt dersom resultatet fra ALU blir 0 (Z-flagget)
- JAMN: et flag fra ALU. Blir satt dersom resultatet fra ALU blir negativt (N-flagget)

- MPC
  - MicroProgram Counter
  - JAM kan påvirke MPC
- Betingahopp
  - viss (hoppinstr og Z) eller (hoppinstr og N)
    - MPC set til hopp microinstr.
  - Hopp microinstr.
    - microinsstr som utfører eit hopp
    - Betinga hopp i mikroinst. prog. flyt





NTNU

La oss se hvordan vi kan bruke dette ved hopp.

Vi kan endre på MPC basert på hva vi får ut fra ALU-en. Selve endringen blir gjort helt fremst i MPC. Dette vises ved en egen pil. Vi kan sette dette bitet til 1 eller 0.

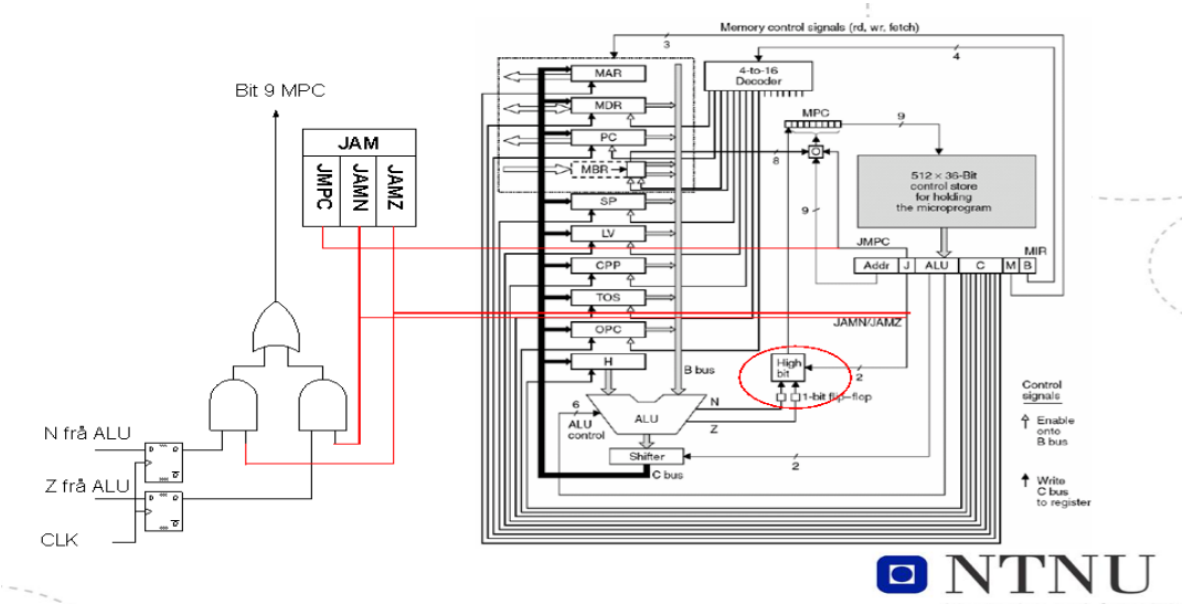
Dersom vi ønsker å sjekke om svaret vårt er negativt så setter vi JAMN = 1. Om vi ønsker å sjekke om svaret er null så setter vi JAMZ = 1. Vi kan sette begge til 1 som betyr at vi får utslag dersom resultatet er negativ eller 0.

Hva som er poenget i IJVM her er at vi har 8 bit operander. MPC derimot er 9 bit. Det vil si at vi kan bruke det 9-ende bitet til å manipulere et hopp i control store. Dette bitet kan bestemme om i skal hoppe til den laveste eller høyeste adressen i control store.

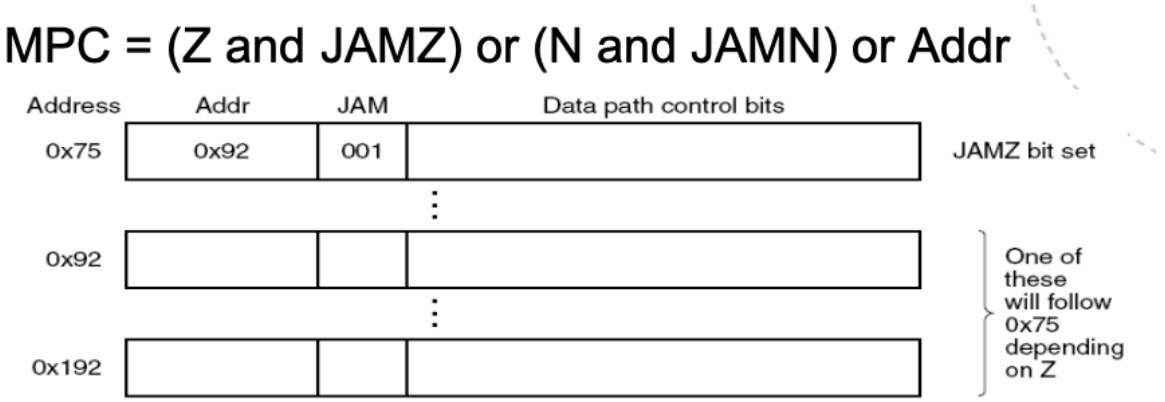
Korleis hopp

- MPC: 9 bit
  - 8 frå **Addr** i micro Instruksjon
  - Viss JAMZ eller JAMN =”0”:
    - MPC = **Addr**
    - Neste microInst **Addr**
  - Viss JAMZ eller JAMN = ”1”:
    - MPC = **Adr** or 100000000
    - Neste microInst 1XXXXXXXX

Siden JAMZ og JAMN bestemmer det første bitet så kan dette styre mye av plasseringen av adressen. Dersom JAMZ eller JAMN = 0 så vil vi bare finne den neste adressen (vi befinner oss i de laveste 256 adressene). Dersom JAMZ eller JAMN = 1 så vil befinne oss i de høyeste 256 adressene.

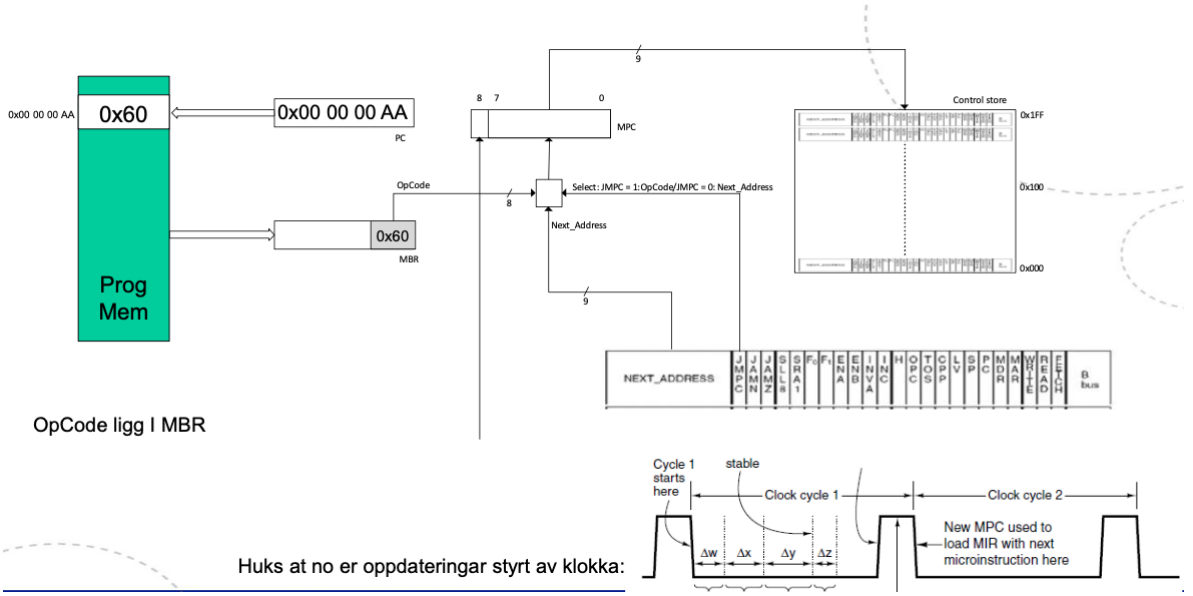


Her ser vi hvordan dette blir implementert i praksis. De mulige test-tilfellene enabler N og Z fra ALU-en. Vi brukker flip-floper for N og Z for å forsikre oss om at verdiene er gyldig når den neste klokkepulsen kommer. Når vi begynner å endre på datapath så er det mulig at vi får ut feil verdi.



Vi kan se litt mer visuelt hva som skjer over tid her. Vi har en adresse 0x75 i control store. Denne har data path control bits, next_adress (0x92) og vi setter JAMZ = 1. Hvilken instruksjon som kommer etter er ikke helt sikkert selv om vi har satt next_address. Pga JAMZ så vil den neste adressen enten bli 0x92 (dersom ALU != 0) eller 0x192 (dersom ALU = 0).
Vi ser at vi legger til 1 i 0x92 og får dermed 0x192

Instruksjonsuftøring



Husk at vi en klokke! Oppdateringen av MIR skjer på hver klokkeflanke.

Vi har en adressen i PC som peker på en instruksjon i programminne. PC peker på AA. I programminne så ligger 0x60 på AA. Vi har kjørt en fetch for å hente hva som er der. Det vil ligge en kopi av 0x60 på MBR-registeret. OpCode vil altså ligge i MBR.

Nå ser vi på dekodingen av instruksjonen. Alle OpCodene (8-bit) peker til første mikroinstruksjon. For å utføre en instruksjon så har vi OpCoden i MBR. Vi må få den opp til MPC (Micro Program Counter). MPC peker til adressen til MIR-innholdet der instruksjonen ligger. Den første instruksjonen ligger altså på der OpCoden peker. JMPC sørger for at MBR kommer til MPC. Dersom JMPC er 1 så går MBR til MPC, dersom JMPC er 0 så går next_address fra MIR til MPC.

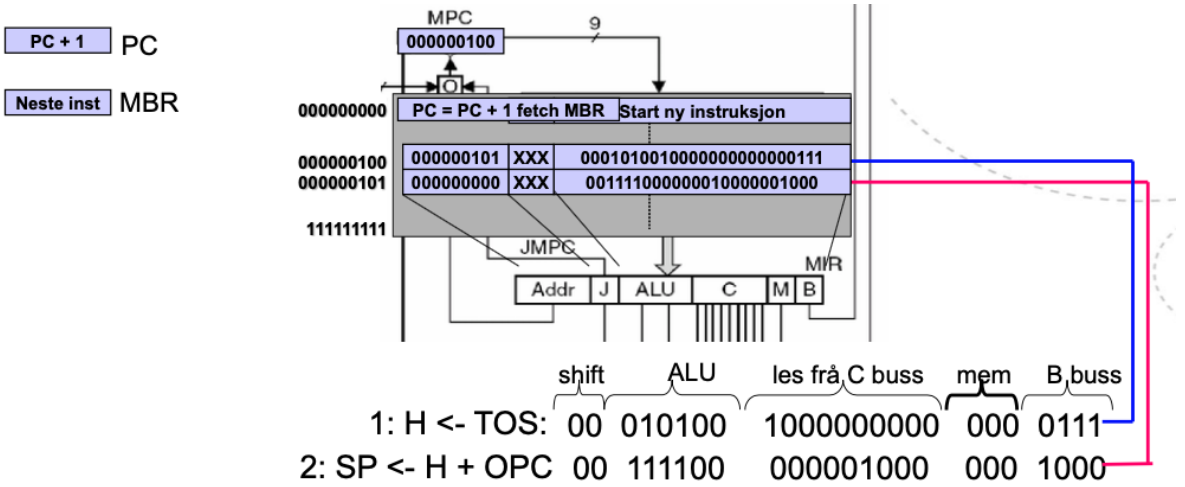
Nå har MPC adressen til den første instruksjonen. Denne blir sendt til control store som da legger ut denne instruksjonen i MIR.

Vi ser for oss at vi har en instruksjon til. MIR vil da inneholde next_address som inneholder adressen til den neste instruksjonen. Vi kan f.eks si at next_address = 0x61. JMPC vil da være 0 for at next_address = 0x61 skal bli kopiert over til MPC. Da vil control store hente ut instruksjon i 0x61 som da definerer bit-mønsteret til MIR for neste mikroinstruksjon.

La oss si at den siste instruksjonen har next_address = 0x00. Det betyr at MPC peker på 00 i control store. Dette betyr at vi kommer til å hente en helt ny instruksjon. Det som faktisk ligger å 0x00 er en mikroinstruksjon: **PC = PC + 1.** Vi

oppdaterer dermed PC med en for å få neste instruksjon. Samtidig tar den innholdet i MBR og kopierer det over i MPC.

Et lite eksempel



La oss si at vi ønsker å gjøre SP = TOS + OPC

Vi finner ut at OpCoden til den instruksjonen er 100. Vi peker altså på plass 100 i control store. Vi setter MIR slik at TOS blir kopiert over til H-registeret. Next_address feltet er nå 101 og JMPC vil være 0. MPC vil altså få 101 adressen som er neste instruksjon i control store. I 101 ligger bitmønsker til neste instruksjonen og er den siste instruksjonen. Dette betyr at next_adress vil være 0x00. Denne adressen betyr at vi skal kopiere det som ligger MBR til MPC, vi skal oppdatere PC = PC + 1 og vi skal få inn neste instruksjon i MBR-registeret.

Det som ligger i control store er definert ut ifra instruksjonssettet til IJVM. Vi definerer ikke disse på programminnet.