NTNU
Kunnskap for en bedre verden

Institutt for informatikk og e-læring, NTNU

Datamaskinens virkemåte

Geir Ove Rosvold
16. august 2016

Datamaskinens virkemåte

Resymé: I denne leksjonen ser vi på den grunnleggende virkemåten til en datamaskin. Vi ser på de forskjellige delene - blant annet CPU, I/O-utstyr og minne. Vi ser på den skjematiske virkemåten til datamaskinen.

Innhold

1.1. Leksjonens oppbygging

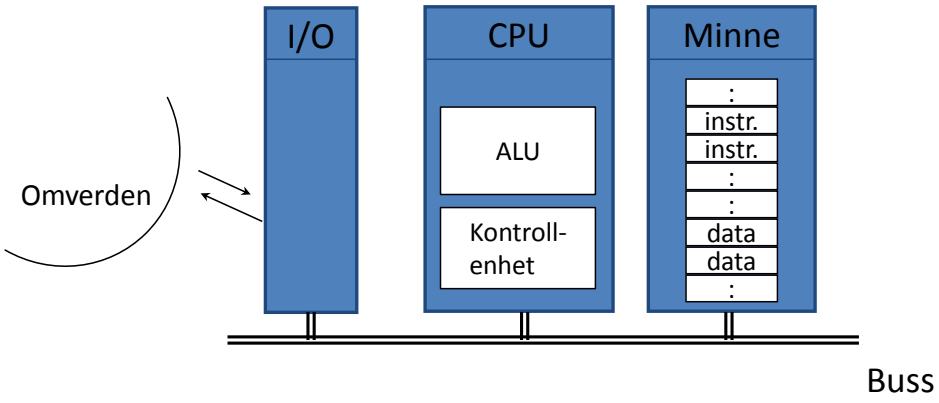
Nå skal vi se på den grunnleggende virkemåten til en datamaskin. Vi begynner med å se på de viktigste komponentene, og hvordan disse kan kommunisere. Deretter ser vi på prosessorens oppbygging og virkemåte.

1.2. De enkelte delene av datamaskinen

Den ungarsk-amerikanske forskeren *von Neumann* har fått kreditt for å være den første som beskrev konstruksjonsprinsippene for en digital regnemaskin av den type som brukes i dag. Et viktig kjennetegn for en slik maskin er at den bruker *binær elektronikk*. Det vil si at all datakommunikasjon og dataprosessering er basert på elektroniske signaler som bare kan innta to ulike verdier, nemlig enten *av* eller *på*. Oftest brukes de matematiske tegnene *0* og *1* for å betegne de to tilstandene, eller vi bruker de logiske begrepene *sant* og *usant*. Et annet kjennetegn på von Neumans design er at ett og samme binære minne inneholder både data og instruksjoner. Tidligere maskiner hadde gjerne et minne for instruksjoner og et annet for data. Von Neumans arkitektur brukes på "alle" dagens datamaskiner, og består av følgende deler:

- Et arbeidslager som inneholder både data og instruksjoner på *binær form*. Dette arbeidslageret kalles *primærminnet*, eller bare *minnet*.
- En *aritmetisk/logisk enhet* (ALU - Arithmetic Logic Unit) som kan utføre en del matematiske og logiske operasjoner.
- En *kontrollenhet* (CU - Control Unit) som tolket instruksjonene i minnet og sørger for at de blir utført.
- En *buss* som disse komponentene bruker til å kommunisere med hverandre. Bussen er en samling ledninger som overfører elektroniske signaler. Disse signalene er alltid binære – de er enten av eller på.
- Inn/ut-enheter (Input/Output, eller I/O på engelsk) som sørger for kommunikasjon med omverdenen.

På moderne maskiner er ALU og CU samlet på en enhet som kalles CPU (Central Processing Unit). Når vi snakker om en *prosessor* er det CPUen vi mener. Figuren nedenfor viser en skjematisk fremstilling av en datamaskin. I resten av leksjonen skal vi se på hver av disse hoveddelene.



Figur 1. Figuren viser en skjematisk fremstilling av de viktigste delene av en datamaskin.

1.2.1. CPU (prosessor)

CPUen - eller prosessoren som vi som regel kaller den - er en elektronisk hjerne som er generell (general purpose), det vil si at den kan utføre en del generelle operasjoner. Disse operasjonene er svært grunnleggende og enkle av natur. For å få inntrykk av hvilket nivå instruksjonene ligger på, kan vi se på noen eksempler:

- Legg sammen to tall.
- Sammenlign to tall.
- Sjekk om et tall er positivt (dvs større enn null).

Det finnes en lang rekke andre også, men som vi ser; det er enkle operasjoner det er snakk om.

For å sette i gang en av disse operasjonene overføres det et bitmønster fra minnet og til prosessoren. Dette bitmønsteret kalles en instruksjon, og er egentlig en beskjed til CPU om hva den skal gjøre. Instruksjonen beskriver både hvilken operasjon som skal utføres, og hvilke data som skal brukes i denne operasjonen.

Et dataprogram er en sekvens av slike instruksjoner som skal utføres etter hverandre. Instruksjonene ligger i minnet, og det normale er at de skal utføres i den rekkefølgen de er lagret. Vi sier at instruksjonene skal utføres sekvensielt. Det betyr at instruksjonene som ligger i minnet skal overføres - en for en - til CPU og utføres der.

1.2.2. Minnet

Datamaskinens minne er en enhet som kan lagre et antall *ord*. Hvert slikt ord er en binær kode bestående av et antall bits. Det er vanlig med 16, 32, 64 eller (på noen få maskiner) 128 bits *ordbredde*.

Hver plass i minnet kalles en *lokasjon*. Lokasjonene nummereres fortløpende, og dette nummeret kalles lokasjonens *adresse*. På denne måten har hver lokasjon i minnet sin egen adresse, og man kan lese eller skrive nøyaktig det ordet man ønsker.

Vi skal komme mye tilbake til minnet senere, men det er viktig å legge merke til følgende:

- Data og instruksjoner befinner seg i samme fysiske minne.
- Lokasjonene i minnet kan adresseres uavhengig av om det er data eller instruksjoner som er lagret der.
- Det eneste som ligger i en minnelokasjon er et bitmønster. Det er opp til CPU å tolke dette bitmønsteret; enten som instruksjon eller som data.

1.2.3. Bussen

En buss er et sett av parallelle ledere. Disse lederne er enten ledninger, eller de er kobberbaner i et printkort. Poenget er at hver enkelt leder kan overføre verdien til en bit. En buss kan overføre mange bits samtidig - en bit på hver av lederne sine.

CPU, minne og IU-enheter er koblet til denne bussen slik at de kan utveksle informasjon. Informasjon utveksles ved at en komponent legger et bitmønster ut på bussen, og de øvrige komponenter som er tilknyttet bussen kan lese dette bitmønsteret.

1.2.4. Prosessor, minne og buss samarbeider

CPU, minne og buss utgjør kjernen i datamaskinen. Med utgangspunkt i disse tre kan vi skissere den grunnleggende virkemåten til datamaskinen. Utgangspunktet er at vi har et minne

der instruksjonene ligger lagret i form av en sekvens bitmønstre. Etter tur skal hver enkelt instruksjon først hentes fra minnet og overføres til prosessoren via bussen. Deretter skal den utføres på prosessoren. Disse handlingene gjentas for hver eneste instruksjon.

Sekvensiell utføring – med noen unntak

Instruksjonene som utgjør et program ligger fortløpende i minnet, og instruksjonene utføres i den rekkefølgen de ligger der. Vi sier at instruksjonene *utføres sekvensielt* At instruksjonene utføres sekvensielt betyr altså at instruksjonene hentes inn til prosessoren og utføres i den rekkefølgen de finnes i minnet.

Det viktigste unntaket fra denne regelen er ved hopp i programmet. Et hopp vil vil si at prosessoren begynner å lese instruksjoner fra en annen minnelokasjon enn den påfølgende sekvensielle adressen.

La oss se på noen eksempler på slike hopp. De som har programmert litt, vet at det i alle programmeringsspråk finnes *if*-setninger, *while*-setninger, *for*-setninger og lignende strukturer. Dette er eksempler på at programmet vil hoppe til en ny plass i minnet. Ved en *if*-setning vil programmet utføre en del instruksjoner kun dersom en betingelse er oppfylt – dersom betingelsen ikke er oppfylt vil disse instruksjonene ikke utføres; da vil prosessoren hoppe over dem. *While*- og *for*-setninger er eksempler på at en del instruksjoner utføres flere ganger – altså en løkkestruktur der prosessoren vil hoppe tilbake til starten av løkka hver gang instruksjonene i løkka skal utføres på nytt.

1.2.5. I/O-utstyr og I/O-moduler

I/O-utstyr en samlebetegnelse for et vidt spekter av utstyr som tar seg av kommunikasjonen mellom datamaskinen og utenomverdenen. Eksemplene på slikt utstyr er tallrike. Vi kan nevne tastatur, skjerm, skrivere, mus, scanner, modem og så videre. Det er også vanlig å regne såkalte *sekundærlagere* - for eksempel harddisker og båndstasjoner - som I/O-utstyr.

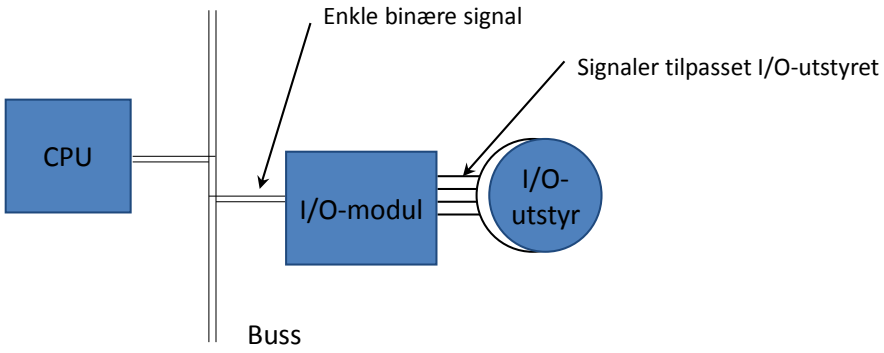
Det er enorm forskjell i overføringshastighet og fysisk oppbygging til de forskjellige typene I/O-utstyr. En habil sekretær taster kanskje 2-3 tastetrykk i sekundet på et tastatur. Tastaturet overfører derfor 2-3 byte pr sekund. En harddisk derimot "spyr ut" flere MB i sekundet.

Siden det finnes så mange forskjellige typer I/O-utstyr kan ikke en datamaskin inneholde elektronikk for å styre og kontrollere absolutt alle typene som finnes. Vi trenger et opplegg som er så fleksibelt at vi kan utstyre datamaskinen med styre-elektronikk for akkurat det I/O-utstyret vi faktisk har. Det skjer på følgende måte:

1. Elektronikk som kan styre en bestemt type IO-utstyr monteres på et lite kort. Dette kortet kalles en IO-modul. På PC brukes ofte andre navn på slike kort, for eksempel *adapter*, *kontroller*[1], eller rett og slett bare *kort* (for eksempel skjermadapter eller grafikk-kort).
2. På en datamaskin installerer vi akkurat de kontrollerne som trengs for det IO-utstyret vi skal bruke. Dersom vi senere skal ta i bruk flere typer IO-utstyr må vi installere flere kontrollere.

[1] IO-modul er et generelt navn som brukes i forbindelse med alle datamaskintyper. PCer har gjerne vært forholdsvis enkle og billige maskiner med relativt enkle IO-moduler. Slike enkle IO-moduler kalles ofte *kontrollere*. På moderne PCer har enkelte kontrollere blitt svært avanserte (spesielt gjelder det skjermkortet), men navnet kontroller brukes ofte likevel.

3. Å intallere en kontroller vil si å koble kontrolleren til bussen. Kontrolleren utgjør med andre ord grensesnittet mellom et I/O-utstyr og bussen. På den ene siden kommuniserer kontrolleren med I/O-utstyret, mens det på den andre siden kommuniserer med CPU (via bussen). Se Figur 2.



Figur 2. I/O-moduler utgjør grensesnittet mellom CPU - som bare sender og mottar enkle bitmønstre - og I/O-utstyret som krever helt andre typer kontrollsignaler.

Den tekniske virkemåten til en kontroller er at CPUen og kontrolleren utveksler bitmønstre via bussen. Bitmønstrene vil representere en av følgende typer informasjon:

- Kommandoer. Dette er bitmønstre som prosessoren sender til kontrolleren. Kontrolleren tolker dem som kommandoer. Kontrolleren inneholder elektronikk som styrer IU-utstyret slik at kommandoen utføres.
- Statusrapport. Dette er bitmønstre som prosessoren leser fra kontrolleren. Bitmønstrene gir informasjon om feilsituasjoner og annen statusinformasjon om kontrollerens aktivitet.
- Data. For ut-utstyr er dette informasjon som prosessoren skriver til kontrolleren. På inn-utstyr leser prosessoren informasjonen fra kontrolleren.

CPUen kommuniserer altså med kontrolleren ved at den leser og skriver bitmønstre over bussen. Dette er helt analogt med hvordan CPU kommuniserer med de ulike lokasjonene i minnet. En datamaskin inneholder mange kontrollere, og CPU må kunne skille mellom dem slik at kommandoene blir sendt til riktig kontroller. Litt forenklet kan vi si at de forskjellige kontrollerne får hver sin adresse på samme måte som lokasjonene i minnet får hver sin adresse.

Eksempler på slike I/O-moduler, eller kontrollere, er blant annet:

- Diskkontrollere som styrer harddisker.
- Grafikkort (eller skjermkort) som styrer skjermer.
- Nettverkskort som tar seg av kommunikasjonen med nettverket.

Det finnes også tallrike andre eksempler.

1.3. CPUens innhold og virkemåte

En CPU er altså elektronikk som kan utføre en del generelle og enkle operasjoner. Eksempler på slike operasjoner er aritmetiske operasjoner og logiske operasjoner

Aritmetiske operasjoner er regneoperasjoner som addisjon, subtraksjon, multiplikasjon, o.a. Logiske operasjoner er boolske operasjoner som AND, OR, NOT, o.a. Det vil si opersjoner som tester eller sammenligner bitmønstre og som enten gir svaret SANT eller svaret USANT.

En slik operasjon igangsettes ved at det overføres en instruksjon til CPU. CPU vil tolke instruksjonen og utføre den rette operasjonen.

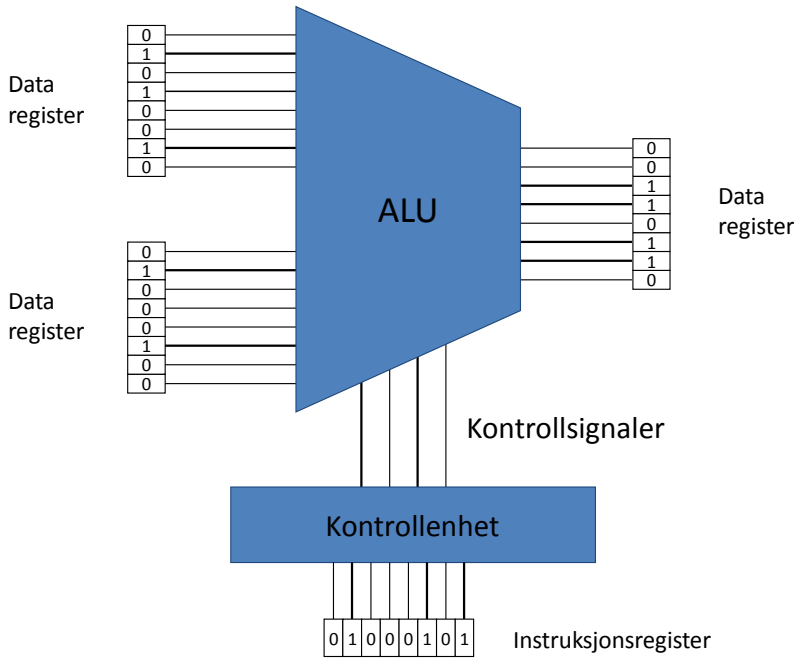
Instruksjonene ligger i minnet og skal normalt utføres sekvensielt - altså i den rekkefølgen de ligger der. Av og til utføres hopp i programmet (for eksempel er en if-setning et slikt hopp). Men det normale er altså at instruksjonene utføres fortløpende. Det betyr at instruksjonene som ligger i minnet skal overføres - en for en - til CPU og utføres der.

1.3.1. Kontrollenhet, ALU og registre

Inne i CPUen er det binære kontrollsignaler som til en hver tid bestemmer hva som skjer. Enkelt forklart består en CPU av en kontrollenhet som tolker instruksjonene og setter opp de riktige kontrollsignalene. ALUen utfører den ønskede operasjonen på de ønskede data. Disse data ligger i et lite minne inne i CPU. Siden dette minnet er innebygget i CPUen, har prosessoren direkte adgang til det. Dette lille interne minnet kalles et *register*.

Hvert *register* inneholder et *ord.* Et ord er et bitmønster som består av et fast antall bits. De fleste moderne prosessorer har en ordbredde på 32 eller 64 bits. De fleste prosessortyper har noen ti-talls slike registre. Alle registrene sett under ett kalles *registerblokken*.

De viktigste delene av en CPU er altså ALUen, kontrollenheten og registerblokken. Dette er skissert i Figur 3. I figuren ser vi en ALU som kan utføre et antall aritmetiske og logiske operasjoner. Det er kontrollsignaler som bestemmer hvilken av de disse operasjonene som skal foretas. Kontrollenheten tolker instruksjonen i instruksjonsregisteret og setter opp de riktige kontrollsignaler til ALU. ALUen har to innganger og en utgang. Både innganger og utganger er bitmønstre. Bitmønstrene på inngangen kombineres og det resulterende bitmønsteret presenteres på utgangen. Bitmønstrene som utgjør inngangsdata og resultatdata ligger i dataregistre.



Figur 3. Enkel fremstilling av CPUens oppbygging.

1.3.2. Instruksjonstyper

Instruksjonene som en CPU utfører er svært enkle og fundamentale. Hver instruksjon utfører en meget begrenset oppgave, og et *dataprogram* består av mange slike instruksjoner som utføres etter hverandre.

Det finnes tre forskjellige kategorier instruksjoner:

- Instruksjoner for dataprosessering. Instruksjoner som gjør at CPUen utfører en aritmetisk eller en logisk operasjon.
- Instruksjoner for dataoverføring. Dette er instruksjoner for overføring av data mellom et CPU-register og en minnelokasjon, eller mellom et CPU-register og en I/O-modul. Slike dataoverføringer kan gå begge veier: fra et CPU-register til en minnelokasjon eller fra en minnelokasjon til et CPU-register, eventuelt fra et CPU-register til en I/O-modul eller fra en I/O-modul til et CPU-register.
- Instruksjoner for programkontroll. Instruksjoner som endrer instruksjonsrekkefølgen bort fra den normale sekvensielle rekkefølgen, det vil si at de fører til et hopp i programmet. Altså at programutføringen starter et nytt sted i minnet, istedenfor for i den neste sekvensielle minnelokasjonen. Et eksempel er en if-setning i et program.

De fleste CPUer har instruksjoner som er kombinasjoner av disse kategoriene.

1.3.3. De viktigste registrene

En moderne CPU inneholder mange registre. Hvert register er tildelt et navn, og de fleste registre har også sin egen helt bestemte funksjon. For å forstå virkemåten til en datamaskin må vi først og fremst se på følgende registre:

- Program Counter, PC eller *programtelleren*, inneholder adressen til neste instruksjon. På Intel-prosessorer kalles dette registeret IP eller instruksjonspeker.
- Instruction register; IR eller *instruksjonsregisteret*, inneholder instruksjonen som skal utføres
- Accumulator, AC eller *akkumulatorregisteret*, som er et register for korttidslagring av data.

1.4. Instruksjonssyklusen

Instruksjonene ligger altså i minnet og utføres etter tur. Om ikke annet er sagt ligger disse instruksjonene fortløpende etter hverandre i minnet.

Når en CPU har utført en instruksjon sier vi at den har gjennomløpt en instruksjonssyklus. For hver instruksjon som utføres må CPUen først hente instruksjonen fra minnet og deretter utføre den. Vi sier at CPUen gjennomløper to delsykluser, nemlig hentesyklus (fetch) og utførings-syklus (execute) slik Figur 4 viser.

Hentesyklus (fetch)

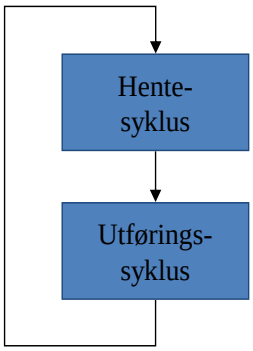
I starten av hver instruksjonssyklus henter CPU inn en instruksjon fra minnet. Et register, PC, holder rede på hvor i minnet instruksjonen skal hentes.

Instruksjonen hentes fra minnelokasjonen og legges i IR-registeret. Instruksjonen er et binært bitmønster som spesifiserer hva CPU skal gjøre i utføringssyklusen.

Etterpå økes PC-registeret med én dersom instruksjonen ikke ber om noe annet. Som regel sier vi at PC-registeret inkrementeres.

Utføringssyklus (execute)

Utføringssyklusen består i at prosessoren utfører instruksjonen i IR-registeret. Utføringene skjer under kontroll av kontrollenheten, som setter opp de riktige kontrollsignalene.



Figur 4 For hver instruksjonssyklus gjennomløper CPU to delsykluser. Disse kalles hentesyklus og utføringssyklus. I hentesyklusen hentes en ny instruksjon fra minnet, og i utføringssyklusen utfører CPU denne instruksjonen.

1.5. En hypotetisk maskin

For å illustrere virkemåten tar vi utgangspunkt i en hypotetisk maskin. Denne maskinen er langt enklere enn reelle maskiner, men viser likevel prinsippene. Vår hypotetiske maskin er en enkel 16-bits maskin. Det vil si at hvert register består av 16 bits, og hver lokasjon i minnet kan også lagre 16 bits.

Minnet

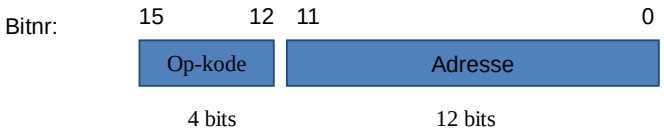
Minnet på den hypotetiske maskinen er organisert som 16 bits ord. Disse ordene kan CPUen tolke enten som instruksjoner eller som heltall.

Instruksjoner

Når et bitmønster skal tolkes som instruksjon er det gitt på formen som er vist i Figur 5. Alle instruksjoner består av fire bits op-kode og en 12-bits adresse.

Op-kode, eller operasjonskode, er et bitmønster som forteller hvilken operasjon CPU skal utføre. For eksempel kan opkoden bety at det skal utføres en addisjon (eller en subtraksjon, en logisk operasjon eller noe annet). Vi ser at det brukes 4 bits til op-koden. Det betyr at vår hypotetiske maskin kan forstå inntil 16 forskjellige instruksjoner.

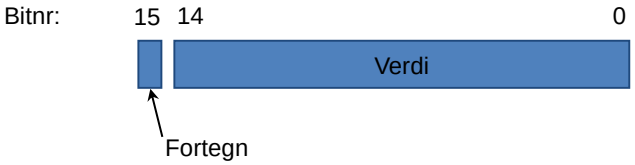
Adresse-feltet er et bitmønster som forteller hvor data som instruksjonen skal bruke ligger. Data ligger i minnet, og adressefeltet forteller hvilken minnelokasjon data ligger på. Vi ser at det brukes 12 bits til adressefeltet. Dermed kan det adresseres $2^{12} = 4096$ forskjellige minne-lokasjoner.



Figur 5. Figuren viser hvordan bitmønsteret i en minnelokasjon på vår hypotetiske maskin skal tolkes som en instruksjon.

Data

Vår hypotetiske maskin er en svært enkel maskin som bare forstår en datatype, nemlig heltall med fortegn. Når et bitmønsteret skal tolkes som data, så er heltallet gitt på den formen som er vist i Figur 6.



Figur 6. Figuren viser hvordan bitmønsteret i en minnelokasjon på vår hypotetiske maskin skal tolkes som et heltall

CPU og registrene

Vi antar at maskinen har tre registre:

- PC (programteller)
- IR (instruksjonsregister)
- AC (akkumulator).

Noen instruksjoner

Op-kode forteller hvilken instruksjon som skal utføres, og siden det brukes 4 bits til op-koden finnes det 16 mulige instruksjoner. Vi antar at tre av disse op-kodene er:

Bitmønster	Hex-verdi	Symbolsk navn	Funksjon
0001_2	1_{16}	LOAD	Les data fra minnet til AC. Adressefeltet angir hvilken minnelokasjon som skal leses.
0010_2	2_{16}	STORE	Skriv til minnet fra AC. Adressefeltet angir til hvilken minnelokasjon data skal skrives.
0101_2	5_{16}	ADD	Adder et tall i minnet med innholdet i AC og legg resultatet i AC. Adressefeltet angir i hvilken minnelokasjon tallet finnes.

Tabell 1. Op-koden til tre av instruksjonene som kan utføres på den hypotetiske maskinen.

1.5.1. Instruksjoner for å addere to tall i minnet

Vi skal nå se hvordan prosessoren kan legge sammen to tall i minnet, og deretter skrive dette resultatet tilbake til minnet. Altså utføre beregningen B=A+B, hvor A og B er tall som ligger i minnet. For å gjøre dette utføres følgende tre instruksjoner på vår hypotetiske maskin:

Instruksjonsrekkefølge for å legge sammen to tall

Bruk instruksjon 1_{16} til å lese fra minnet og legg tallet i AC

Bruk instruksjon 5_{16} til å addere et tall i minnet med tallet i AC. Resultatet legges i AC.

Bruk instruksjon 2_{16} til å skrive innholdet av AC til minnet.

Anta nå at tallene som skal adderes ligger i minnelokasjon nr 940_{16} og 941_{16}. (Her kunne vi ha valgt andre adresser, men vi har valgt akkurat disse denne gangen).

Hvordan blir bitmønstrene som spesifiserer disse instruksjonene?

Vi begynner med den første instruksjonen. Fra Figur 5 ser vi at de fire bitene lengst til venstre i instruksjonene er op-koden, og fra Tabell 1 ser vi at op-koden for å lese fra minnet er $1_{16} = 0001_2$. De fire første bitene blir dermed 1_{16}. Så må vi se på adressefeltet til instruksjonen. Dette feltet angir jo hvilken minnelokasjon som skal leses inn i AC. Bitmønsteret i adressefeltet må være $940_{16} = 1001\ 0100\ 0000_2$. Vi setter sammen disse, og finner at bitmønstret for hele instruksjonen blir $0001\ 1001\ 0100\ 0000_2 = 1940_{16}$.

For den andre instruksjonen ser vi at op-koden er 5_{16}. Tallet som skal adderes til AC ligger i adressen 941_{16}, som dermed utgjør adressefeltet til instruksjon 2. Bitmønsteret for den andre instruksjonen blir derfor 5941_{16}. Den tredje instruksjonen skal skrive innholdet av AC tilbake til lokasjon 941_{16}, og bitmønsteret blir 2941_{16}.

Vi antar nå at instruksjonene legges fortløpende fra lokasjon 300. Igjen kunne vi valgt andre adresser, men nå bruker vi lokasjonene fra 300. På ordentlige maskiner er det operativsystemet som bestemmer hvor i minnet programmet skal ligge.

Figur 7 viser hvordan data flyter mellom minnet og CPU når dette programmet utføres. Vi starter med at PC inneholder verdien 300, som er startadressen til programmet. I trinn 1 leses innholdet av minnelokasjon 300 inn i IR (hentesyklus). I trinn to utføres instruksjonen (utføringssyklus) - det vil si at innholdet av lokasjon 940 leses inn i AC. Når vi kommer til trinn 3 er PC automatisk inkrementert slik at den peker på neste instruksjon. I trinn 3 (hentesyklus) hentes innholdet av lokasjon 301 til IR. I trinn 4 (utføringssyklus) adderes AC med innholdet av lokasjon 941. Resultatet legges i AC. I trinn 5 hentes instruksjonen i lokasjon 302, og i trinn 6 utføres instruksjonen som skriver resultatet til lokasjon 941.

1.6. Symbolsk program

Isteden for å sette opp programmet slik vi gjorde i rammen ovenfor, kunne vi ha skrevet ett *symbolsk program*. Da setter vi beskrivende navn, eller symbolske navn, på hver op-kode. Dette symbolske navnet fremgår av Tabell 1. Isteden for å si "utfør instruksjonen som henter data fra minnet og legger inn i AC" kan vi si utfør instruksjonen LOAD. Ved siden av det symbolske navnet setter vi opp adressen som skal brukes. Da blir programmet mer lettlest.

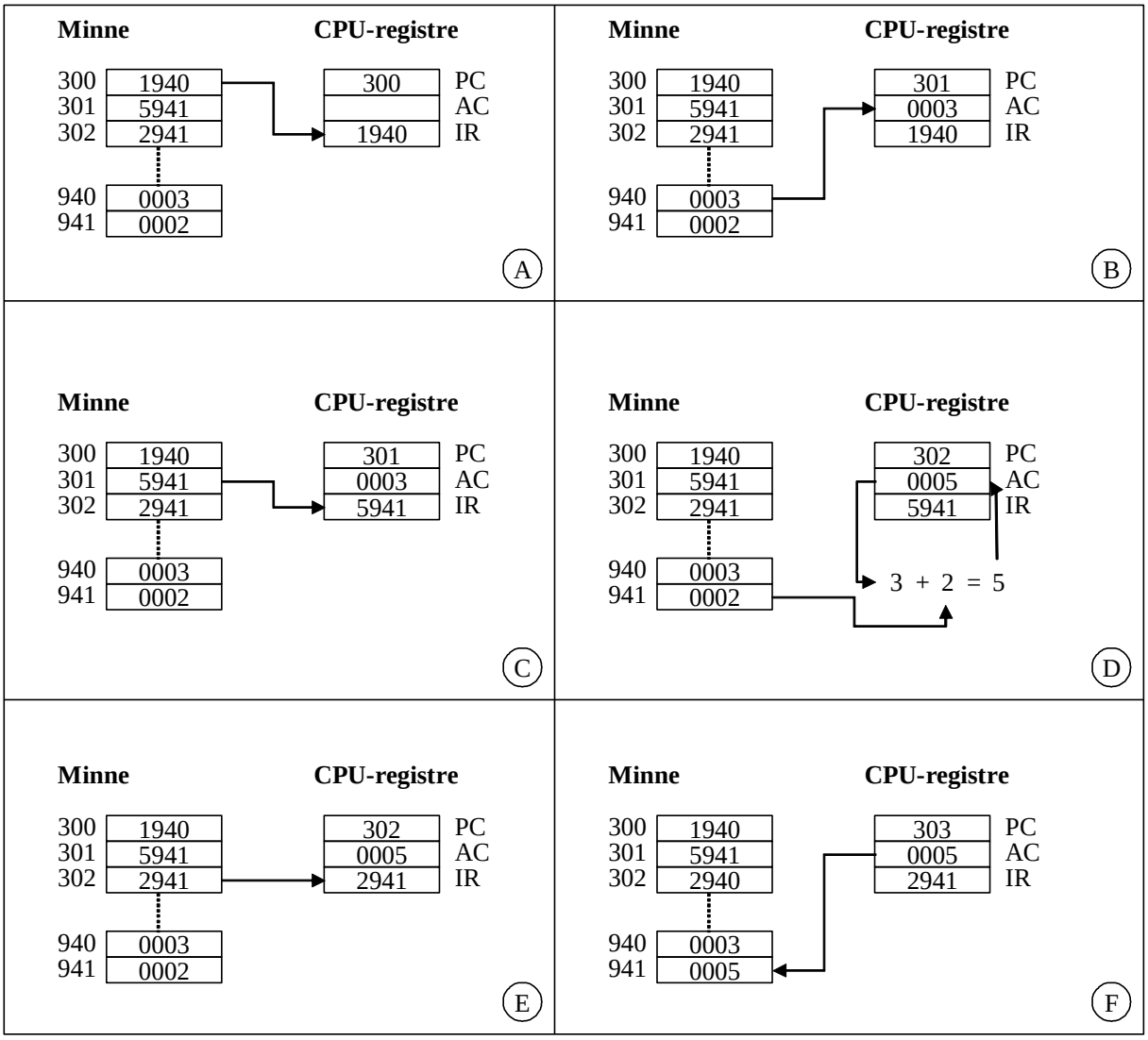
Eksempel på symbolsk program

```
LOAD    940
ADD     941
STORE   941
```

1.7. Opkoder og operander

I det symbolske programmet i rammen ovenfor består hver linje av en opkode og en adresse. Adressen forteller hvor data som skal brukes befinner seg. Så langt har vi antatt at data befinner seg i minnet. Men på en reell maskin kan jo data også befinne seg andre steder, nemlig i et CPU-register eller på en I/O-modul.

Siden data kan befinne seg andre steder enn i minnet går vi bort fra å kalle adresse-feltet i instruksjonen for adresse, isteden kaller vi feltet en *operand*. Hver instruksjon består altså av en op-kode og en operand. *Op-koden* beskriver hva som skal gjøres, og *operanden* forteller hvor data som skal behandles ligger.



Figur 7. Instruksjonssyklusen. A, C og E viser hentesykluser, og B, D og F viser utføringssykluser. Instruksjonene ligger sekvensielt fra lokasjon 300. I A leses den første instruksjonen inn i IR. Denne instruksjonen utføres i B hvor data leses fra lokasjon 940 og kopieres til AC. PC inkrementeres. I C leses neste instruksjon fra adresse 301. I D adderes verdien i adresse 941 med verdien i AC, og resultatet legges i AC. PC økes med en, og i E leses siste instruksjon fra adresse 302. I fig F skrives innholdet av AC til lokasjon 941.

En operand er vanligvis et register, en minnelokasjon eller en I/O-modul.

På en reell prosessor brukes ofte flere operander. På Intel-prosessorene i en PC har de fleste instruksjoner en eller to operander, men det finnes instruksjoner som har ingen operander og noen få som har tre. På en slik Intel-prosessor finnes det en instruksjon som heter *mov*. Denne instruksjonen kopierer data fra et sted til et annet. På slik prosessorer finnes det mange registre. for eksempel finnes det et register som heter *ax* og et annet heter *bx*. Instruksjonen som kopierer data fra bx-registeret til ax-registeret er:

mov ax bx

Dette er en instruksjon som har en opkode (*mov*) og to operander (*ax* og *bx*). I dette tilfellet er begge operandene *registre*.

1.8. Mer om utføringssyklusen

Nå vet vi nok til å se litt nærmere på utføringssyklusen. Vi har sett at utføringssyklusen ikke nødvendigvis bare består i å utføre en regneoperasjon på ALU, men at den også kan medføre at data må hentes fra minnet og/eller at resultater må skrives til minne.

Dermed kan vi spalte utføringssyklusen opp i følgende handlinger:

- Finn stedet der innoperand ligger
- Hent innoperand
- Utfør regneoperasjon
- Finn stedet der resultatet (utoperand) skal ligge
- Skriv utoperand

Dersom det brukes mer enn en innoperand må de første to linjene gjentas for hver av dem. Det samme gjelder de to siste linjene dersom instruksjonen har mer enn en utoperand (men det er ikke vanlig).

1.9. Instruksjoner og høynivåspråk

Dere er vant med å lese kildekoden til et program skrevet i et høynivåspråk, for eksempel java eller C++. Det er viktig å forstå at det er stor forskjell mellom *setninger i høynivåspråk* på den ene siden og *instruksjonene til CPU* på den andre siden.

Instruksjonene er tilpasset CPUen, og vi har sett at en CPU bare kan gjøre svært enkle handlinger. Hver instruksjon innebærer handlinger på et svært lavt nivå, som for eksempel flytte data mellom minne og register eller legge sammen to tall som finnes i registrene. Instruksjonene gjenspeiler CPUens oppbygging. Instruksjonene som finnes på èn CPU-type er helt andre instruksjoner enn de som finnes på andre CPU-typer. Dette er fordi forskjellige CPU-typer har forskjellig oppbygging; forskjellig antall register, forskjellige operasjoner på ALU og så videre.

Et høynivåspråk er derimot tilpasset oss mennesker. Det skal først og fremst være lett å forstå og å programmere. I tillegg skal det skjule særegenheter med den CPUen som programmet skal kjøre på. Vi sier at høynivåspråket er maskinuavhengig; kildekoden til et C-program ser

likedan ut enten det skal kjøre på en PC, en Mac eller en av IBMs stormaskiner – selv om CPUene i disse maskinene er vidt forskjellig, og forstår helt forskjellige instruksjoner.

Det er kompilatorens jobb å oversette et høynivåprogram til en sekvens av instruksjoner som CPUen kan utføre. For å legge sammen to tall, A og B, vil vi i alle språk skrive noe sånt som `b=a+b`. Dette gjelder f.eks java, C++, Visual Basic, … Som vi har sett, vil en slik linje i et høynivå-program oversettes til en sekvens av tre instruksjoner i den hypotetiske maskinen som ble beskrevet i kap. 1.5. Mer kompliserte høynivå-strukturer, som for eksempel switch-setninger, oversettes til en mye lenger sekvens av instruksjoner.