NTNU

Kunnskap for en bedre verden

Institutt for informatikk og e-læring, NTNU

Avbruddsystemet

Geir Ove Rosvold

6. januar 2016

Avbruddsystemet

Resymé: I denne leksjonen ser vi på avbruddsmekanismen til datamaskinen. Først ser vi på programmert IO. Deretter på avbrudd i forbindelse med I/O-utstyr. Vi er også på de andre anvendelsene av avbruddsmekanismen. Til sist ser vi på direkte minneaksess (DMA)

Innhold

1.1. Introduksjon

Vi har sett at datamaskinen grovt sett består av CPU, minne og I/O-utstyr. Videre vet vi at disse delene kommuniserer over en buss. Vi har sett på virkemåten til CPUen. Og vi har også sett at elektronikken som styrer I/O-enhetene ligger på såkalte I/O-moduler (som gjerne kalles kontrollere på en PC).

Alle som har brukt en datamaskin en stund vet at når vi starter et program – for eksempel ved å dobbelklikke på programmets symbol i Windows – så tar det en stund før programmet kommer i gang. Det er fordi programmet må hentes inn fra harddisken og startes opp. Når programmet først har kommet i gang går det imidlertid ganske radig. Vi kan altså grovt sett skissere virkemåten til datamaskinen slik:

- Når program og data ikke er i bruk så lagres de på en disk. Der er de organisert som filer. Programmene ligger som kjørbare filer. Kjørbare filer inneholder de instruksjonene som et program består av. Alle andre filer inneholder data, og kalles datafiler. Datafiler inneholder data som brukes av programmene. Data kan være hva som helst, for eksempel dokumenter med tekst til en tekstbehandler, data til et regneark og mye annet.

  Vi kan oppfatte disken som et arkiv der det ligger program og data som ikke er i bruk.

- Når et program skal starte vil operativsystemet hente programmets kjørbare fil fra disken og laste instruksjonene inn i primærminnet. Deretter vil operativsystemet la programmet kjøre på CPUen. Det er dette som skjer når vi dobbelklikker på et program i Windows. Operativsystemet inneholder også systemrutiner som gjør at programmet senere kan be operativsystemet hente data fra datafilene og legge dem også i minnet. Det er dette som skjer når vi for eksempel åpner et dokument i Word.

  I minnet ligger altså det programmet som i øyeblikket kjører samt de data som dette programmet trenger. Instruksjonene til programmet ligger normalt fortløpende (sekvensielt) i minnet.

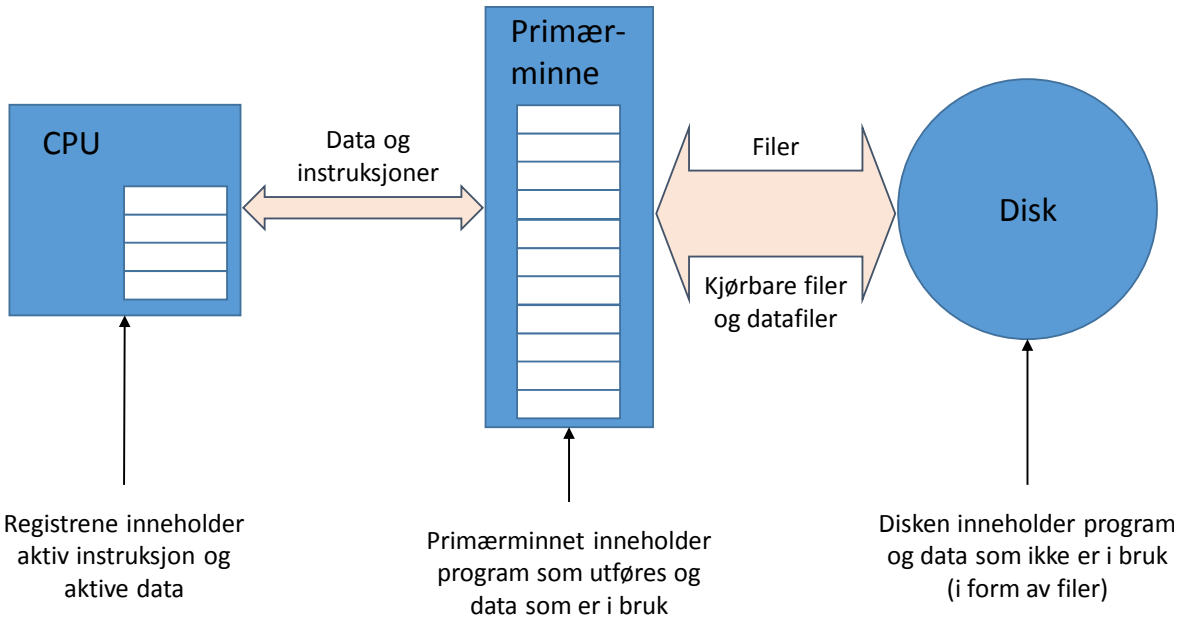
- Når programmet utføres, betyr det at instruksjonene - en for en - hentes fra minnet og overføres over bussen til CPUen. I CPU legges instruksjonen som skal utføres inn i et register. I tillegg til dette registeret finnes det også et lite antall registre som inneholder data som instruksjonen kan bruke.

  Instruksjonene hentes inn en for en til CPUen. Inne i CPUen ligger bare instruksjonen som er under utføring, samt litt av de data som programmet skal behandle.

Dette er skissert i Figur 1.

1.2. Bruk av I /O-utstyr

Det er liten vits i datamaskinen dersom den ikke kan kommunisere med omverden. Slik kommunikasjon skjer ved hjelp av ulike typer I/O-utstyr. I/O står for Input/Output; Utstyr for Input gir data til datamaskinen, mens utstyr for Output presenterer data som kommer fra datamaskinen. Tastatur og mus er eksempler på utstyr for input, mens skjerm og skriver er eksempler på utstyr for Output. Det finnes også I/O-utstyr som gir både input og output. En disk er et eksempel på det.



Figur 1. Figuren viser CPU, primærminne og disken. Program og data som ikke er i bruk ligger på disken. Programmet som kjører, og data som dette programmet bruker, ligger i primærminnet. Den instruksjonen som er under utføring ligger i en av prosessorens register. Data som prosessoren trenger ligger også i registrene.

For å kunne bruke et I/O-utstyr må datamaskinen inneholde elektronikk som kan styre utstyret. Slik elektronikk kalles for I/O-moduler eller kontrollere. Når man kobler en disk til datamaskinen, må man også installere en diskkontroller (hvis man ikke har en fra før), når man kobler en skjerm til maskinen må man ha en skjermkontroller (kalles ofte skjermkort, grafikkort eller skjermadapter), når man kobler til et digitalt videokamera må man installere et kort som tar seg av kommunikasjonen med kameraet (ofte et såkalt Firewire-kort).

En del kontrollere finnes som standard på vanlige PCer (for eksempel tastaturkontroller, disk-kontroller, et enkelt skjermkort, kontrollere for serieporter, printerport, usb-port,…). Andre kontrollere må installeres ved behov (avanserte skjermkort, Firewirekort og nettverkskort kan være eksempler på det).

1.3. Hastighet og behov for synkronisering

Det aller meste av I /O-utstyr er svært langsomt i forhold til prosessoren, og programmer bruker mye tid på å vente på at I/O-utstyr skal bli ferdig. En moderne prosessor utfører millioner av instruksjoner hvert sekund – men selv en habil sekretær greier ikke å taste mer enn noen få tegn i sekundet på et tastatur. Et tekstbehandlingsprogram bruker altså meste-parten av tiden til å vente på at neste tast skal trykkes, og mellom hvert tastetrykk kan den utføre hundretusenvis av instruksjoner.

Et eksempel på dette kjenner vi fra Word. Hver gang vi har tastet et ord i Word, slår programmet opp i ordlista og sjekker om ordet er skrevet rett. Programmet rekker å gjøre ferdig denne oppgaven, før vi begynner å taste inn neste ord. Feilstavede ord markeres med en rød bølge under ordet. Etter at vi har tastet inn en hel setning, vil Word foreta en grammatikalsk sjekk av setningen. Setninger med grammatikalske feil markeres med en grønn bølgelinje. Disse oppgavene krever mange instruksjoner. Inntasting av tegn skjer imidlertid

svært langsomt i forhold til tiden det tar å utføre en instruksjon. Derfor rekker programmet å utføre oppgavene samtidig som brukeren skriver.

1.3.1. Stadig spørring

En skriver er et eksempel på I/O-utstyr som er mye langsommere enn prosessoren.

For enkelhets skyld kan vi se på en såkalt matriseskriver. Dette er en relativt enkel type skriver som bare kan skrive ut tegn – altså bokstaver, tall og lignende. Matriseskriveren kan ikke skrive ut grafikk (som bilder og tegninger). Den forholder seg bare til tegn.

Slike skrivere virker på den måten at det kommer en strøm av tegn fra datamaskinen. Denne strømmen overføres via en kabel fra datamaskinen til skriveren. Skriverens oppgave er å skrive ut hvert av disse tegnene. Problemet er imidlertid at det tar flere tusen ganger lengre tid å skrive et tegn på skriveren enn å utføre en instruksjon på prosessoren.

Et program som skal bruke skriveren må ta hensyn til dette. For eksempel må ikke skriveren mates med tegn fortere enn den klarer å skrive. Programmet som skriver ut må derfor synkroniseres med skriveren.

Hver gang skriveren mottar et tegn må den flytte papiret til rett linje, flytte skrivehodet til rett posisjon på linjen, og så skrive tegnet. Prosessoren må vente med å sende neste tegn til skriveren er ferdig.

På en eller annen måte må skriveren kunne varsle datamaskinen om at den er klar til å motta et nytt tegn. Dette er ikke så vanskelig; vi bruker bare en av ledningene inne i kabelen mellom datamaskin og skriver til å overføre et signal. Hver gang skriveren har skrevet ferdig et tegn, så sender den et signal til I/O-modulen om at den er klar til å motta neste tegn.

Prosessoren kan når som helst sende en forespørsel til I/O-modulen for å få vite om skriveren er klar til å skrive et nytt tegn.

Hvordan ser så programmet som kjører på datamaskinen ut? Jo, den delen av programmet som skal skrive ut et enkelt tegn vil bestå av tre deler:

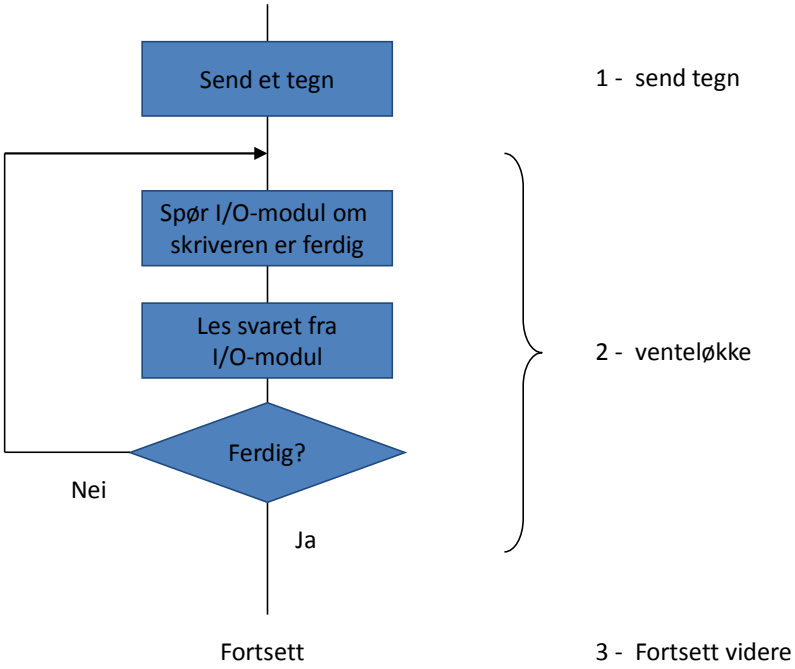
1. Send tegn → Her sendes tegnet over kabelen til skriveren.
2. Venteløkke → Mens skriveren holder på med å skrive et tegn, må CPU vente til skriveren er ferdig. Denne delen av programmet består av en løkke som hele tiden sjekker om skriveren er ferdig. I denne perioden gjør ikke programmet noe produktivt i det hele tatt. Det bare står og sjekker om det har kommet noe signal fra skriveren om at tegnet er ferdigskrevet.
3. Fortsett → Først når skriveren har meldt fra at den er ferdig med å skrive tegnet, kan programmet fortsette. Dersom det skal skrives ut flere tegn hopper programmet til 1 igjen, for å skrive ut neste tegn.

1.3.2. Flytskjema

Et flytskjema er en skjematisk fremstilling av programflyten til et dataprogram. I Figur 2 ser vi et flytskjema for de tre punktene ovenfor.

Først sendes et tegn til skriveren (via I/O-modulen). Deretter går prosessoren inn i venteløkka. Venteløkka består i å spørre I/O-modulen om skriveren er ferdig, så lese svaret – og ut fra svaret bestemmes det om løkka må gjennomføres en gang til, eller om man kan fortsette i programmet. Poenget er at siden skriveren er langsom i forhold til instruksjonsutføringen, så vil CPU spørre I/O-modulen mange ganger før svaret endelig er positivt. Av de tre delene i

flytskjemaet er det venteløkka som tar desidert mest tid. Det å sende et tegn over kabelen til skriveren tar bare noen få instruksjoner – gjerne bare èn på en moderne CPU.



Figur 2. Flytskjema over programutføring ved I/O med venteløkke (programmert I/O eller polling).

Venteløkka er derimot et problem. Det at skriveren er så langsom gjør at venteløkka gjerne gjennomføres tusenvis av ganger før skriveren endelig er ferdig. Det er altså der tidsforbruket ligger.

Flytskjemaet illustrerer en ting til, nemlig at I/O-modulen kun kommuniserer med CPU når den utrykkelig blir bedt om det. Den tar ikke selv initiativ – den venter pent til CPU ber om å få tilsendt en statusrapport.

Dette er jo ikke noe problem dersom prosessoren likevel ikke har noe annet å gjøre. Men på en datamaskin – selv på en vanlig ordinær PC – kjører som regel mange programmer samtidig. Da er ikke dette noen god løsning. Vi burde brukt tiden til å utføre instruksjoner til de andre programmene istedenfor å kaste bort tiden med å gjentagne ganger sjekke om skriveren er klar.

Avbrudd (Interrupt på engelsk) er en mekanisme som lar programmet drive med nyttige ting i ventetiden.

La oss tenke oss et firma med en sekretær som bare har følgende to oppgaver: Hun (eller han) skal skrive brev og passe telefonen.

Tenk på arbeidssituasjonen til denne sekretæren dersom telefonen ikke har noen ringelyd. Da må sekretæren sjekke om noen ringer ved å ta opp røret og lytte på telefonen.

Dette vil selvsagt gå kraftig utover effektiviteten. Siden vi aldri kan vite når det kommer en henvendelse, må jo sekretæren med jevne mellomrom sjekke om det er noen på telefonen. Og hvor ofte må hun gjøre dette? Minst to problemer er åpenbare: dersom hun sjekker telefonen veldig ofte, vil det gå mye tid til spille fordi det tross alt er relativt sjelden at det

kommer en henvendelse. Dersom hun sjekker for sjelden vil hun kunne miste henvendelser (fordi folk blir lei av å vente, og gir opp).

Det er en mye bedre løsning at telefonen ringer. Da vil jo sekretæren kunne sitte med skrivearbeidet helt til det faktisk kommer en henvendelse. Når henvendelsen kommer kan hun betjene henvendelsen, og deretter gjenoppta skrivearbeidet.

På samme måte er det med matriseskriveren vår: I stedet for at prosessoren kaster bort tiden med å gå i en venteløkke bør den utføre en annen og mer nyttig oppgave. Dette krever imidlertid at skriveren kan "ringe" og fange CPUens oppmerksomhet. En slik mekanisme finnes på alle prosessorer, og kalles avbruddsmekanismen.

1.4. Avbruddsmekanismen

Avbruddsmekanismen er en viktig del av moderne datamaskiner. Vi skal først se på avbruddsmekanismen i forbindelse med bruk av I/O-utstyr, og etterpå skal vi se at mekanismen brukes i andre forbindelser også.

1.4.1. Uten avbrudd: Polling

I forrige kapittel så vi hvordan et program synkroniserte hendelsene på en skriver ved hjelp av ei venteløkke. Når et program inneholder en slik venteløkke – som ikke har noen annen funksjon enn at den venter på at I/O-utstyret skal bli ferdig – så sier vi at programmet benytter *polling* eller *stadig spørring*. Et annet navn som ofte brukes på slik I/O er *Programmert I/O*. Grunnen til dette navnet er nok at CPU i detalj kontrollerer alle detaljer i I/O-overføringen.

Ulempen med programmert I/O eller polling er selvfølgelig at tiden som går med til denne stadige spørringen kunne vært brukt til å utføre mer matnyttige instruksjoner.

1.4.2. Med avbrudd: avbruddsdrevet IO

Isteden for at CPU hele tiden skal sjekke om skriveren er ferdig, bør det være slik at skriveren (eller rettere sagt I/O-modulen) selv kan varsle CPU når den er ferdig.

Istedenfor at prosessoren går inn i en venteløkke kan den da fortsette med en annen og mer produktiv oppgave. Og så, når skriveren er ferdig, kan I/O-modulen varsle CPU om dette ved å sende et signal til CPU. Dette signalet kalles et avbruddssignal.

Dette bryter med datamaskinens virkemåte slik vi har sett på den frem til nå. Tidligere har vi sagt at prosessoren er sjef på datamaskinen, og suverent skal bestemme og styre alt som skjer på maskinen.

Nå går vi litt bort fra dette. Nå sier vi at også I/O-modulen selv skal få lov å ta initiativ ovenfor prosessoren. I/O-modulen skal få "ringe" og varsle CPU om at den trenger oppmerksomhet.

Forskjellen fra tidligere er at nå kan I/O-modulen komme til å "ringe" når som helst. Prosessoren må være laget slik at den kan ta imot avbruddssignalet til en hver tid – og helt uavhengig av de andre oppgavene til CPU.

1.4.3. Endringer i CPUens virkemåte

Det at prosessoren ikke bare kan ture frem som den vil, men også må ta hensyn til at andre enheter kan utføre handlinger på egen hånd, gjør at CPUen må være laget for å ta hensyn til avbrudd. Vi skal se hvilke endringer det er snakk om.

1.4.4. Avbruddsrutiner

Sekretæren vår vil ha faste rutiner for hva hun gjør når telefonen ringer. Vi kan for eksempel godt tenke oss at rutinen ved telefonhenvendelser vil bestå i å løfte telefonrøret, presentere firmaet og seg selv, hjelpe kunden, deretter si farvel og til siste legge på røret igjen.

Når telefonen ringer vil hun skrive ferdig ordet hun har begynt på, og deretter ta telefonen. Så gjennomfører hun rutinen for telefonhenvendelser. Etter at hun har gjort ferdig samtalen vender hun tilbake til skrivearbeidet, og fortsetter akkurat der hun slapp.

Legg merke til at den eneste måten skrivearbeidet blir påvirket på, er ved at det oppstår en liten forsinkelse – ellers skjer det ingen endring i skrivearbeidet.

På en prosessor vil avbruddsmekanismen virke på en lignende måte. Når et avbruddssignal kommer, vil prosessoren legge til side det arbeidet den holder på med, og isteden betjene avbruddet.

På samme måte som sekretæren må vite hva hun skal gjøre når telefonen ringer, må også CPUen vite hva som skal gjøres ved et avbrudd. Dette skjer ved at instruksjonene som skal utføres etter et avbrudd samles i en egen rutine. En rutine som skal startes ved avbrudd kalles avbruddsrutine. På engelsk kalles en avbruddsrutine for ISR (Interrupt Service Routine). Når et avbruddsignal kommer vil CPU legge til side det programmet den kjører, og starte avbruddsrutinen isteden.

1.5. Hvordan virker avbruddsmekanismen?

Avbrudd er en mekanisme som virker på følgende måte:

Når det kommer et avbruddssignal, legger prosessoren til side det kjørende programmet og kjører et annet program i stedet. Dette andre programmet kalles en avbruddsrutine. Etter at avbruddsrutinen er ferdig utført, gjenopptar CPU arbeidet som ble avbrutt.

Avbruddsrutiner er ganske korte program. Det er kort fordi vi ikke vil at datamaskinen skal bruke mye tid på å betjene hvert avbrudd.

CPU starter avbruddsrutinen når den får avbruddssignal fra en I/O-modul. Etter at avbrudds-rutinen har kjørt ferdig vil CPU fortsette utføringen av programmet som ble avbrutt. Programmet som ble avbrutt merker ingenting (bortsett fra en liten forsinkelse).

1.5.1. Avbruddssignalet

Hvordan skal så I/O-modulen varsle CPU? Jo, ved at den sender et avbruddssignal til CPU. Avbruddssignalet sendes over bussen. På kontrollbussen finnes det en egen linje for å varsle avbrudd. Via denne busslinjen sendes avbruddssignalet til CPU.

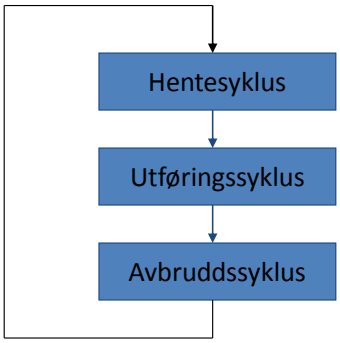
Hver gang det kommer et avbruddssignal på denne linjen vil CPU legge til side kjørende program, sjekke hvem som har sendt avbruddssignalet og sette i gang riktig avbruddsrutine (ISR).

1.5.2. Når sjekker CPU om det har oppstått avbrudd?

Sekretæren vår vil alltid skrive ferdig ordet hun holder på med, selv om hun blir avbrutt av telefonen eller av at det banker på døra.

På en prosessor er det tilsvarende: et avbrudd kan ikke avbryte en instruksjon som er under utføring. Avbrudd sjekkes først etter at instruksjonen er ferdig utført. Dette er gjort ved å utvide instruksjonssyklusen med en delsyklus til. I en tidligere leksjon så vi at CPUen instruksjonssyklus består av to delsykluser; nemlig hente- og utføringssyklus. Nå skal vi utvide instruksjonssyklusen med en delsyklus til, nemlig en såkalt avbruddssyklus.

Figur 3 viser den nye instruksjonssyklusen etter at avbruddssyklusen har kommet til.



Figur 3. Figuren viser instruksjonssyklusen når vi har lagt inn en egen syklus som sjekker for avbrudd (avbruddssyklus).

1.5.3. Hvordan startes avbruddsrutinen?

Når det kommer et avbrudd skal CPU legge til side kjørende program og kjøre avbrudds-rutinen isteden. Hvordan gjøres dette i praksis?

For å forstå det, må vi huske litt fra en tidligere leksjon: Der lærte vi at et program er en sekvens av instruksjoner som ligger i minnet. Instruksjonene skal utføres en for en, og det finnes et eget register i CPU som alltid peker på neste instruksjon. Dette registeret kalles PC (Program Counter). Hver gang vi har utført en instruksjon, peker PC på neste instruksjon. Et eller annet sted i minnet ligger programmet som er under utføring, og hver gang CPU er ferdig med en instruksjon peker PC på den neste instruksjonen.

Avbruddsrutinen er også et program. Derfor ligger instruksjonene som utgjør avbruddsrutinen i minnet. Avbruddsrutinen ligger en annen plass i minnet enn programmet som utføres. Både kjørende program og avbruddsrutinen ligger altså i minnet samtidig. Så lenge vi ikke får et avbrudd utfører CPU instruksjonene i det kjørende programmet. Men så kommer det et avbrudd, og da må CPU sørge for å starte utføringen av avbruddsrutinen isteden.

Avbruddet betyr at CPU skal begynne å utføre instruksjonene i avbruddsrutinen - isteden for å utføre neste instruksjon i kjørende program. For å starte utføringen av avbruddsrutinen må CPU sørge for å endre innholdet av PC-registeret slik at det peker på startadressen til avbruddsrutinen.

Men nå holdt vi på å glemme noe. Vi må jo også huske at når avbruddsrutinen er ferdig, så skal CPU gjenoppta arbeidet med kjørende program – dette programmet skulle jo ikke merke noe til avbruddet. Det betyr at CPU må lagre verdien til PC før den skriver inn startadressen til avbruddsrutinen.

Når det kommer et avbrudd, består altså avbruddssyklusen av følgende handlinger:

- *Lagre innholdet av PC-registeret*. Det vil si at vi legger bort adressen til det som er neste instruksjon i kjørende program. Vi legger den et sted der vi finner den igjen senere.

- *Sett PC til startadressen for avbruddsrutinen*. Det vil si at vi sørger for at CPU begynner å kjøre avbruddsrutinen.

Etter at avbruddsrutinen er ferdig vil CPU finne tilbake den gamle verdien av PC, og fortsette utføringen av programmet som ble avbrudt.

1.5.4. Oppsummering

En god ting kan ikke sies for ofte, så vi tar det grunnleggende en gang til:

Vi har lært at en instruksjonssyklus består av tre delsykluser: hente-, utførings-, og avbrudds-syklus. Når et avbrudd inntreffer vil CPU gjøre ferdig instruksjonen som den har begynt på. Det betyr at CPUen gjør ferdig hente- og utføringssyklusen for instruksjonen. Deretter utføres avbruddssyklusen. I avbruddssyklusen blir avbruddet registrert og avbruddsrutinen startes.

Avbruddsrutinen startes ved at PC-registeret - som alltid inneholder adressen til neste instruksjon - settes til avbruddsrutinens startadresse. Før dette skjer må imidlertid den gamle verdien til PC-registeret tas vare på slik at det kjørende programmet senere kan fortsette der det ble avbrutt.

Senere - når alle instruksjonene i avbruddsrutinen er utført - settes PC-registeret tilbake til sin gamle verdi. Dermed kan det programmet som ble avbrutt fortsette "som om ingenting hadde hendt".

1.6. Avbruddstyper

Så langt har vi konsentrert oss om I/O-utstyr som avbruddskilde. Avbruddsmekanismen brukes imidlertid i mange andre sammenhenger også. De viktigste er beskrevet nedenfor.

1.6.1. I/O-utstyr.

Slike har vi allerede sett på.

1.6.2. Klokkeavbrudd (timer interrupt)

En klokkekrets gir avbrudd (med svært høy prioritet) med eksakte tidsintervall. Ofte mellom 15 og 50 ganger pr sekund. Operativsystemet benytter gjerne anledningen til å utføre administrasjon når avbruddet kommer.

1.6.3. Utstyrsfeil

Både CPU og I /O-utstyr har ofte elektronikk som detekterer feil. Eventuelle feil varsles ved å gi et avbrudd. Eksempel på slike feil er

- bortfall av spenning
- paritetsfeil i minnet

1.6.4. Programfeil

Mange forskjellige programfeil kan føre til avbrudd. Noen eksempler:

- forsøk på divisjon med 0
- forsøk på å utføre ulovlige instruksjoner
- ulovlig operand
- forsøk på å lese eller skrive til/fra beskyttet del av minnet

1.6.5. Feilfinning (debuging)

Mange operativsystemer har egne spesialavbrudd til bruk for debuging av program. Disse brukes blant annet til å stanse programutføringen på bestemte steder i programmet (break-points). Deretter kan brukeren se på verdien til variable, innholdet av registre og så videre.

1.6.6. Kall til operativsystemet – programmert avbrudd

Alle program har av og til behov for tjenester fra operativsystemet. Eksempler på slike tjenester er innlesing fra tastatur, skriving på skjermen eller skriver, samt skriving og lesing på disk. Hvert enkelt program inneholder ikke egen kode for slike ting. Isteden ber programmet operativsystemet om å utføre tjenesten; vi sier at programmet starter en *systemrutine*.

Kanskje kunne man tenke seg at systemrutinene var vanlige funksjoner som programmet kunne kalle på samme måte som det kaller sine egne funksjoner. Dette er imidlertid ikke tilfellet. Isteden for å implementere systemrutinene som vanlige funksjoner, er det gjort slik at systemrutinene faktisk er avbruddsrutiner. Når et program vil starte en systemrutine så skjer dette ved at programmet selv foretar et avbrudd. Dette avbruddet fører til at avbruddsrutinen (som altså er systemrutinen) utføres.

Hva mener vi med at programmet selv foretar et avbrudd? Jo, dette realiseres ved at det finnes en egen instruksjon som får CPU til å oppføre seg som om det hadde skjedd et avbrudd. Når programmet utfører denne instruksjonen så "simuleres" det altså et avbrudd. Prosessoren oppfører seg akkurat som om det hadde kommet et avbrudd fra en hvilken som helst annen avbruddskilde: det starter en avbruddsrutine (som i dette tilfellet er en systemrutine). Et slikt "imitert" avbrudd kalles et *programmert avbrudd.*

Hvorfor ikke bruke en helt vanlig funksjon isteden for slike avbruddsrutiner? Den viktigste grunnen til at det er fornuftig å bruke avbruddssystemet er at avanserte operativsystem har beskyttelsesmekanismer hvor det kreves privilegier for å lese eller skrive til/fra I/O-utstyr. Vanlige brukerprogram har **ikke** slike privilegier, og en funksjon får alltid samme privilegier som det kallende programmet. En avbruddsrutine har derimot andre privilegier enn vanlige brukerprogrammer. Når et brukerprogram foretar et slikt programmert avbrudd oppnår vi altså at et brukerprogram kan *starte* en avbruddsrutine (systemrutine med tilstrekkelige privilegier), men *ikke endre* innholdet i den.

La oss se litt nærmere på hvordan dette er implementert i praksis. Når CPU behandler et avbruddssignal, skjer to vesentlige ting: En avbruddsrutine blir startet, og CPUen settes i *priviligert modus*. Et vanlig program har ikke tilgang til **alle** CPUens instruksjoner. For eksempel er det normalt ikke tillatt å utføre I/O-instruksjoner for et vanlig program; et vanlig program må be operativsystemet om å utføre jobben. Operativsystemet sjekker om operasjonen er lovlig - for eksempel om bruker har lov til å skrive fila - før operasjonen utføres. For å utføre slike spesielle instruksjoner må CPUen settes i det som kalles priviligert modus. Altså en modus med flere rettigheter enn den modus CPUen befinner seg i når den kjører vanlige program.

1.7. Tre typer IO

Så langt har vi sett på IO mest fra maskinvaren sitt ståsted. Vi kan forsøke å se det fra programvaren sitt ståsted også. Vi skal da se på tre måter å gjennomføre IO-opersjoner på. De tre metodene belaster CPU i ulik grad:

1. Programmert IO. Vi har sett på det tidligere.
2. Avbruddsdrevet IO.
3. Direkte minneaksess. Dette skal vi se på lenger ned.

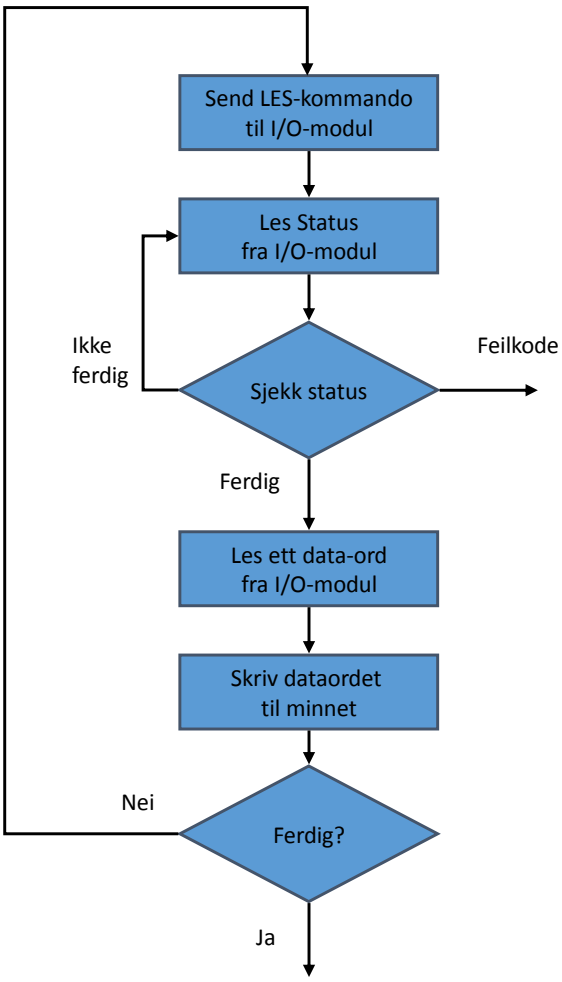
1.7.1. Programmert I/O

I enklere systemer, med lavkost I/O-moduler, er ofte hele I/O-operasjonen under kontroll av CPU. Dette belaster CPUen, men til gjengjeld har den full kontroll over hvert steg i operasjonen. I mange dedikerte anvendelser - styrings- og overvåkingssystemer, spill, fjernkontroller og lignende - har likevel ikke CPU andre oppgaver å gjøre, og direkte CPU-styring av overføringen kan være både effektivt og billig. I/O-modulene kan gjøres meget enkle og rimelige hvis programmert I/O er akseptabelt i systemet.

Generelt følger kommunikasjonen mellom CPU og I/O-modul følgende mønster:

- CPU sender en kommando til I/O-modulen.
- Når I/O-modulen har utført kommandoen, varsler den ved å sette en bit i kontroll/statusregisteret.
- CPU "poller" kontroll/statusregisteret inntil kommandoen er utført.

I Figur 4 er det fremstilt et eksempel der det foretas en lesing fra et IO-utstyr.



Figur 4 Lesing med hjelp av programmert I/O (Stadig spørring eller polling på engelsk). Mens I/O-modulen henter data går programmet i en løkke som stadig sjekker om I/O-modulen har hentet data fra I/O-utstyret. Denne "spørreløkka" utgjør den innerste løkka i flytdiagrammet.

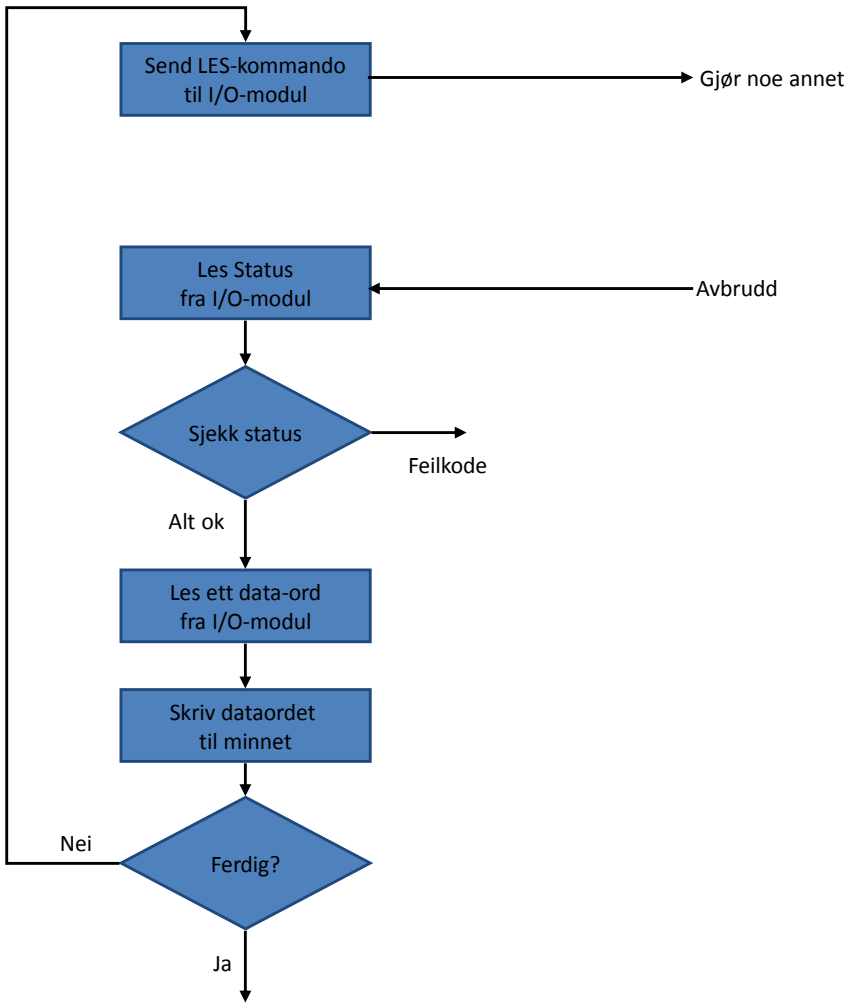
Den kontinuerlige pollingen (lesingen) fra statusregisteret kalles av og til for "busy waiting". Legg merke til at ingen overføring starter uten at CPUen tar initiativ til det; dette gjelder også for data *inn* til CPU. I et system med flere inndata-kilder vil CPU polle de ulike I/O-modulene en etter en, som regel i en fast rekkefølge. Så snart en av modulene svarer positivt på pollingen, blir dataene fra den modulen lest av CPU.

1.7.2. Avbruddsdrevet I /O

I en generell maskin, eller en maskin som skal utføre flere oppgaver samtidig, er det lite ønskelig at CPUens kapasitet skal brukes til å igjen og igjen spørre om en I/O-operasjon er ferdig. Vi vil gjerne starte operasjonen, og så la CPU gjøre andre oppgaver; I/O-modulen kan rapportere til CPUen når overføringen er fullført, og i mellomtiden kan CPU utnytte tiden til "fornuftige" oppgaver. Dette er vist i Figur 5.

For å få noen gevinst med denne løsningen er det nødvendig at I/O-modulene er i stand til å "arbeide på egenhånd" over en viss tid, og varsle CPUen når de er ferdig; de blir derfor mer komplekse/kostbare, men de kan avlaste CPU betydelig.

I/O-modulene bruker avbruddsmekanismen for å varsle CPU om at de er ferdige, og trenger oppmerksomhet.



Figur 5. Avbruddsdrevet I/O. Isteden for at CPU hele tiden sjekker om I/O-modulen er klar, så varsler I/O-modulen selv at den er ferdig. Dette skjer ved at I/O-modulen sender et avbruddssignal til CPU. CPU kan gjøre andre – og mer fornuftige ting – enn å gå i en venteløkke.

1.7.3. Ulemper med både programmert IO og avbruddsdrevet IO

Både programmert I/O og avbruddsdrevet I/O har en del ulemper:

- Begge medfører at prosessoren utfører mange instruksjoner
- Alle data går via prosessoren

La oss kort se nærmere på hver av disse ulempene:

Flytdiagrammene for programmert I/O og på avbruddsdrevet I/O (se **Feil! Fant ikke eferansekilden.** og Figur 5) viser at både programmert I/O og avbruddsdrevet I/O består av mange trinn – mange instruksjoner skal utføres.

Med programmert I/O blir CPU helt bundet opp av I/O-håndtering og kan ikke utføre andre oppgaver.

Ved avbruddsdrevet I/O kan CPU riktignok gjøre en del arbeid mellom hvert avbrudd. Men alle avbrudd medfører en del administrasjon, og totalt sett blir denne administrasjonen tid-krevende hvis avbruddene kommer svært tett. Anta for eksempel at det skal overføres 2048 bytes fra disken til primærminnet over en 8 bits buss. Da vil avbruddsrutinen bli utført 2048 ganger, en for hver byte (8 bits) som overføres. For hvert avbrudd blir en rekke verdier i registre lagret unna, adresser og tellere hentet fram, dataverdien hentet fra I/O-modulen til CPU og derfra sendt ut til primærminnet, før det gamle registerinnholdet endelig kan hentes fram igjen.

Det vi trenger – i hvert fall for I/O-utstyr som skal overføre mye data – er en mekanisme som overfører data direkte mellom I/O-modulen og minnet, uten å gå via CPU. Dette vil i vesentlig grad avlaste CPU som dermed kan gjøre annet fornuftig arbeid enn å utføre I/O. En slik mekanisme er DMA, eller Direkte MinneAksess (*Direct Memory Access* på engelsk).

1.7.4. Direkte minneaksess, DMA

DMA brukes først og fremst på I/O-moduler som overfører store datamengder. Typiske eksempler kan være diskkontrollere, høyhastighets nettverk, grafiske skjermer med høy ytelse og så videre. For å oppnå en mer effektiv dataoverføring kan vi gi slike I/O-moduler ansvaret for å styre overføringen av en større *blokk* med data. Denne blokka overføres direkte mellom I/O-modulen og minnet - uten å gå gjennom CPU, og uten at CPU styrer med detaljene i overføringen.

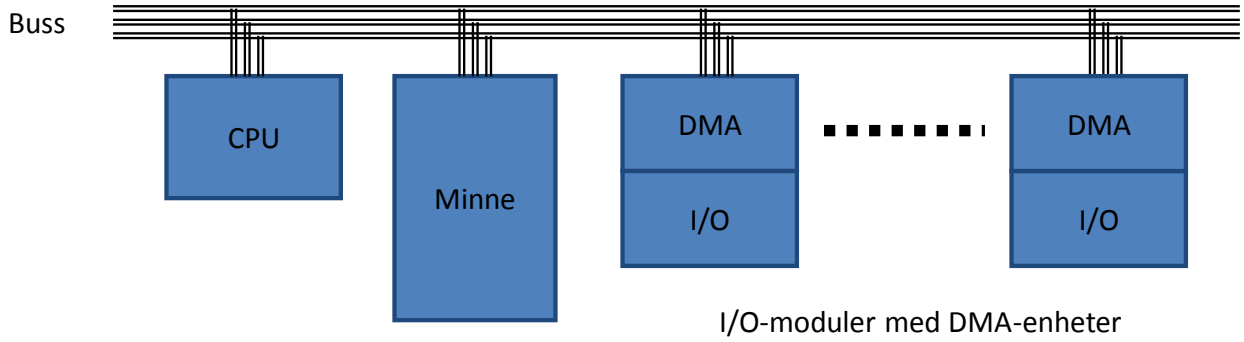
I/O-modulen får overført fra CPU de verdiene den trenger for å starte overføringen. Det kan være:

- Skal det leses eller skrives.
- Hvilket eksternt utstyr skal brukes.
- Hvor skal det leses eller skrives (for eksempel sektor- og spornummer på en disk).
- Hvor mye skal leses eller skrives.
- Startadresse i minnet.
- Antall ord som skal overføres

Men med en gang operasjonen er startet, tar CPU til med andre oppgaver og I/O-modulen arbeider på egenhånd. Den skriver (eller leser) data i primærminnet over bussen, administrerer tellere og lageradresser. Når hele overføringen er fullført, sender den et avbruddssignal til CPU for å signalere at overføringen er ferdig.

Betegnelsen DMA - Direct Memory Access - henspiller på at DMA-modulen selv adresserer primærminnet; den benytter seg ikke av CPUen for å få tak i eller lagre dataverdier i minnet.

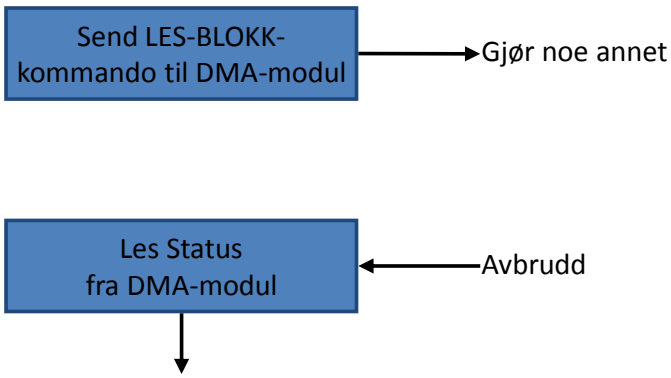
Siden I/O-modulen nå får flere oppgaver, kompliserer selvsagt I/O-modulen; den blir mer komplisert og dermed dyrere. Den delen av I/O-modulen som har ansvaret for DMA kalles en *DMA-enhet*. En I/O-modul som har støtte for DMA har altså en innebygget DMA-enhet slik Figur 6 viser.



Figur 6. DMA er en mekanisme som gjør at I/O-modulene kan kommunisere direkte med minnet uten at data går via CPU. I/O-modulen blir mer komplisert, men får ansvaret for å overføre en større blokk med data.

Flytdiagram ved DMA

DMA gir en vesentlig avlastning for CPU. Som vi har sett, medfører både programmert I/O og avbruddsdrevet I/O at I/O-aktivitetene er relativt arbeidskrevende for CPU. DMA reduserer belastningen i stor grad, slik Figur 7 viser.



Figur 7. Direkte minneaksess (DMA). Flytdiagram over CPUens aktivitet ved DMA.