Institutt for informatikk og e-læring, NTNU

Primærminne

Geir Ove Rosvold

6. januar 2016

Primærminne

Resymé: I denne leksjonen diskuteres primærminne. Vi ser på hovedtypene ROM og RAM. Minneorganisering, både internt på brikkene og eksternt (hvordan vi setter sammen flere brikker til et fullstendig primærminne) presenteres. Vi ser at det finnes to viktige RAM-teknologier, nemlig DRAM og SRAM. Styrker og svakheter med hver av dem diskuteres.

Innhold

1.1. Primærminne

I en datamaskin finnes det mange forskjellige lagertyper. Så langt i kurset har vi i hvert fall nevnt registrene, primærminnet (eller bare minnet) og harddisk.

De lagertypene som kan aksesseres direkte av CPUen kalles internt minne. Dette er registrene, cache (som vi skal se på i en annen leksjon), og primærminnet. Andre lagertyper kan ikke aksesseres direkte fra CPU, men må aksesseres via en I/O-modul. Slike lagertyper kalles eksternt minne eller sekundærlager. Eksempler på dette er disker og CD-ROM.

I denne leksjonen skal vi se på primærminnet. I senere leksjoner ser vi på de øvrige lagertyper.

Primærminne er en svært viktig del av datamaskinen. Ytelsen og størrelsen på primærminnet bidrar sterkt til å bestemme ytelsen til hele datamaskinen. En av grunnene til dette er at når et program utføres så ligger instruksjonene i minnet, og instruksjonene blir overført til CPU og utført en etter en.

Utføringen av enhver instruksjon medfører altså minst en minneaksess; nemlig for å hente instruksjonen. Mange instruksjoner innebærer dessuten enten at data skal hentes fra minne og brukes i instruksjonens beregninger, eller at resultatet av beregningen skal skrives til minnet. Derfor får vi ofte flere minneaksesser pr instruksjon.

Dette er grunnen til at det er vitalt at primærminnet greier å levere instruksjoner og data tilstrekkelig kjapt. Vi kaller tiden det tar å hente data fra minnet for minnets aksesstid. Det er altså viktig at minnet har en aksesstid som er så kort at det greier å holde tritt med CPUens behov. Hvis dette ikke er tilfellet vil CPU kaste bort tid på å vente på minnet.

I denne leksjonen skal vi se at det på moderne maskiner forholder seg slik at CPUen er altfor rask til at minnet greier å holde følge. Den vanligste måten å takle denne situasjonen på skal vi se på i neste leksjon; nemlig å bruke cache.

1.1.1. Bakgrunn

Opp gjennom årene har det vært brukt mange forskjellige minneteknologier. Nå er halvlederminne enerådende.

En av de teknologiene som tidligere var mye brukt, var såkalte ferittkjernelager eller *core*. Slike lager var bygget opp av små metalloksyd-ringer med diameter på rundt 1 mm. En slik ring kan magnetiseres i en av to retninger, og den ene veien ble brukt som 0 den andre som 1. Hver ring lagret altså verdien til en bit. Gjennom hver ring ble det tredd tynne ledninger som ble brukt til lesing og skriving.

Core var altså bygget opp av enheter (små ringer) som lagret en bit hver. Moderne halvlederminne er også bygget opp av små enheter - vi kaller dem celler - som lagrer hver sin bit. Men halvlederminne er fremstilt som integrerte kretser, ICer, og vi kan derfor ikke se hver enkelt celle. Kretsene inneholder en stor mengde celler - antallet celler måles i megabits eller i gigabits. Det vil si millioner eller milliarder av bits i hver minnebrikke.

Vi skiller mellom to hovedtyper minne, nemlig ROM og RAM. ROM er en forkortelse for *Read Only Memory*, og er altså minne som kun kan leses og ikke skrives. RAM står for *Random Access Memory* og er minne som både kan leses og skrives. En datamaskin inneholder begge typer minne.

1.2. ROM

ROM er en fellesbetegnelse for minne som det normalt bare kan leses fra. Innholdet av ROM-minne bevares hvis strømmen forsvinner. Program som ligger i ROM trenger derfor ikke lastes fra disk.

Noen eksempler på hva ROM brukes til er:

- BIOS (Basic I/O System) på PC. Dette er maskinnær programvare som operativ-systemet kan bruke. BIOS inneholder også oppstartsrutiner som utføres når PCen startes.
- Lagring av *mikroprogram*. Dette skal vi diskutere i senere leksjoner.

ROM brukes langt mindre nå enn tidligere. Tidligere var ROM-minne raskere enn RAM. Dette har nå endret seg, og moderne RAM er raskere enn moderne ROM. ROM brukes stadig i stor utstrekning i ikke-generelle datamaskiner som kontrollere, måleutstyr, leketøy og spill, og hundrevis av andre anvendelser som verken er PC, RISC, mini- eller stormaskin.

Den opprinnelige ROM-brikken var slik at minneinnholdet ble lagt inn under fremstilling av brikken, og innholdet kunne ikke endres senere. Dette hadde blant annet de svakhetene at de måtte bestilles i store serier og at innholdet burde være riktig ved første forsøk.

Det var selvfølgelig ønskelig at minneinnholdet kunne legges inn på et senere tidspunkt, og derfor ble såkalte *Programmerbare ROM* (PROM)-brikker utviklet. Med riktig utstyr kan brukeren programmere slike brikker én, og bare én, gang. En ny PROM-brikke har 0-ere i alle lokasjoner, og med hjelp av en "PROM-brenner" kan man legge inn 1-ere der man ønsker. Dette gjøres ved å tilføre en spenning som er høyere enn de som brukes i datamaskinen slik at en tynt isolasjonslag brennes av i de ønskede lokasjonene.

Neste trinn var en brikke hvor innholdet kunne slettes og skrives på nytt. Intel løste dette problemet i 1971, og startet produksjon av EPROM-brikker (*Erasable PROM*). På disse brikkene finnes det et gjennomsiktig vindu på toppen, og innholdet kan nullstilles ved å belyse brikken med UV-lys gjennom dette vinduet. Senere kan nytt innhold skrives på samme måte som for PROM. Når vi sletter en slik brikke fjernes altså all informasjon på brikken. En EPROM kan slettes og skrives på nytt mange ganger, men den er dyrere enn en PROM.

Senere har Texas Instrument utviklet EEPROM-brikken, eller *Electrically EPROM*, altså en brikke som kan slettes elektronisk. Denne har mange fordeler. Man kan slette deler av innholdet, det vil si at bare de adresserte områdene slettes og at resten ikke endres. Dessuten kan både sletting og skriving utføres uten bruk av spesialutstyr. Det betyr at både lesing, sletting og skriving kan skje uten å ta ut brikken fra maskinen - man bruker de ordinære busslinjene og den ordinære spenningen i maskinen. Det tar imidlertid mye lengre tid å slette enn enn å lese fra brikken. Sletting kan ta flere hundre millisekund, mens lesing tar et par hundre nanosekund. EEPROM er dyrere enn EPROM og hver celle tar større plass slik at man ikke får lagret like mange bit på hver brikke.

Den nyeste ROM-teknologien er *Flash-memory*. Denne kom på midten av 80-tallet, og her benyttes også elektrisk sletting. Slettingen går mye fortere enn på en EEPROM, men man kan ikke slette på byte-nivå slik som på EEPROM. Isteden må man slette blokker av data. Størrelsen på disse blokkene varierer fra brikketype til brikketype. Hver celle er liten, slik at en flash-memory-brikke lagrer mer enn en EEPROM med samme fysiske størrelse.

Etter hvert har prisen på Flash sunket så mye at vi kan bruke ganske mye av det. Først kom såkalte minnepenner. Disse har vanligvis et USB-grensesnitt, og operativsystemet kan presentere disse som en liten harddisk. I de senere årene har det også kommet SSD-disker, der

Flash-minne erstatter det magnetiske lagringsmediet i en harddisk. Disse har samme grensesnitt mot datamaskinen som de tradisjonelle harddiskene, for eksempel SATA. Foreløpig kan ikke lagringskapasiteten konkurrere med de magnetiske harddiskene, men aksesstiden er mye bedre på SSD-disker enn på magnetiske disker.

For EPROM, EEPROM, flash-memory er betegnelsen ROM egentlig misvisende, siden de også kan skrives. Et bedre navn har kanskje vært Read Mostly Memory. Navnet ROM er likevel i vanlig bruk.

1.3. RAM

RAM er en forkortelse for *Random Access Memory*. RAM er minne som både kan leses og skrives. Minnet er avhengig av kontinuerlig drivspenning, og hvis strømmen forsvinner går all informasjon i minnet tapt.

Random betyr vilkårlig, og navnet sier egentlig ikke annet enn at man uten videre kan lese eller skrive en hvilken som helst plass i minnet (uten å for eksempel starte forfra slik man må på et bånd). Igjen har vi altså et misvisende navn - ROM kan jo også leses vilkårlig - men navnet er så innarbeidet at alle bruker det.

1.3.1. DRAM og SRAM

Vi har to hovedtyper av RAM, nemlig Statisk RAM (eller SRAM) og Dynamisk RAM (DRAM). I Statisk RAM er hver enkelt celle en stabil transistorkrets - en såkalt FlipFlop eller bistabil vippe. At den er stabil betyr at den er konstruert slik at informasjonen huskes så lenge drivspenningen ikke forsvinner.

I DRAM er hver celle bygget opp rundt en kondensator isteden. Når man skriver til cellen vil kondensatoren utsettes for en spenning. Kondensatoren lades opp, og når spenningen forsvinner vil kondensatoren "huske" spenningen en liten stund - inntil kondensatoren lades ut. Siden kondensatoren husker spenningen i svært kort tid, må informasjonen stadig vekk oppdateres. Oppdateringen skjer med en egen gjennoppfrisknings-elektronikk som lader opp cellene et visst antall gang i sekundet. Man skulle tro at en slik gjennoppfrisknings-krets ville fordyre minnet betraktelig, og kanskje også være en potensiell feilkilde. Dette er imidlertid i liten grad tilfellet. Man har produsert slike kretser i mange år, så teknikken er både billig og med svært lav feilrate.

Til tross for at den må utstyres med elektronikk for gjennoppfriskning er DRAM langt billigere enn SRAM, men det er også tregere. Hver enkelt celle er mindre på DRAM enn på SRAM slik at hver brikke kan inneholde flere celler og dermed ha større lagringskapasitet. På de fleste datamaskiner benyttes DRAM i primærlageret, mens SRAM brukes til cache.

Aksesstider for DRAM og SRAM

Aksesstiden, eller tilgangstiden, er den tiden minnet bruker på å fremskaffe ønskede data.

For DRAM er som regel aksesstiden ca 35-70 ns.

For SRAM er som regel tiden mellom 1 og 10 ns.

1.4. RAM kontra ROM i dagens maskiner

Moderne datamaskiner benytter sjelden ROM for systemrutiner og biblioteksrutiner. Dette var vanlig tidligere, men siden RAM er raskere enn ROM vil det sinke systemet. På ordinære datamaskiner, for eksempel PC, er ROM mest brukt i forbindelse med booting; altså oppstart av datamaskinen. Datamaskinen må jo lese oppstartsprogrammet fra ROM siden RAM-minnet er tomt når strømmen skrus på.

På en PC ligger disse oppstartsrutinene i en del av minnet som kalles ROM BIOS. BIOS står for Basic I/O System. I ROM BIOS ligger det også mange systemrutiner som kan brukes av operativsystemet. DOS brukte disse BIOS-rutinene svært mye. Moderne operativsystem, som Windows og Linux, har sine egne systemrutiner og bruker ikke BIOS-rutinene.

Som tidligere nevnt brukes ROM i stor utstrekning i ikke-generelle datamaskiner som digitale kamera, mp3-spillere, kontrollere, måleutstyr, leketøy, spill og mange andre anvendelser. Det brukes også til mer utradisjonelle lagringsmedia, som usb-penner.

1.5. Minnebrikkene

Vanligvis ser vi på primærminnet som en enhet. Denne enheten er organisert slik at den består av mange adresserbare lokasjoner. Hver lokasjon kan lagre et ord. Det er vanlig å oppgi lager-kapasiteten som det antall *bytes* primærminnet kan lagre - selv om ordbredden kan være en annen enn 8 bits.

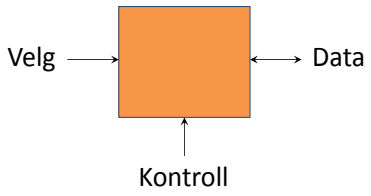
I de fleste sammenhenger er det greit å se på primærminnet som en enhet, men i denne leksjonen skal vi ”åpne minnet” og se hvordan enheten er bygget opp. Da vil vi se at primærminnet er bygget opp av mange integrerte kretser. Denne typen integrerte kretser kaller vi minnebrikker.

1.5.1. Minneceller

Minnebrikkene er integrerte kretser som består av mange celler som hver kan lagre en bit. Hver celle har tre tilkoblingspunkter hvor signaler sendes til eller leses fra cellen. Disse tre tilkoblingspunktene er:

- *Velg* er et binært signal som settes når akkurat denne cellen skal leses eller skrives
- *Kontroll* er et binært signal som avgjør om det skal skrives til cella eller om innholdet av cella skal leses
- *Data* er det siste signalet. Ved skriving på cella kommer data inn til cella på denne linjen. Skal cella leses, så legges data ut på denne linjen.

Figur 1 viser en skjematisk fremstilling av en celle med tilhørende signaler. Retningen på signalene fremgår også av figuren.



Figur 1. Figuren viser en celle med tilhørende signaler. Cellen kan lagre en bit.

1.5.2. Virkemåte

Når vi skal forstå virkemåten til minnet er det viktig å huske at:

1. Utad er minnet bygget opp som en lang rekke lokasjoner. Hver lokasjon har plass til *ett ord*. Det vil si at hver lokasjon består av like mange minneceller som det er bits i ordet. For eksempel kan vi ha en ordbredde på 64 bits. På en moderne PC består primærminnet av flere hundre millioner slike 64-bits lokasjoner.
2. Vi kan lese eller skrive hver enkelt minnelokasjon fordi hver lokasjon har en adresse. Når vi skal aksessere (det vil si lese eller skrive) en minnelokasjon settes Velg-signalet til cellene i denne lokasjonen. Velg-signalet til alle andre celler settes til null.
3. Samtidig settes også Kontroll-signalet til cellene i lokasjonen. Dette forteller cellen om det er lesing eller skriving som skal skje.
4. Dersom vi vil skrive til cella, legges den binære verdien som skal skrives på cellens data-signal. Dersom vi skal lese data fra cellen, legger cellen sin binære verdi ut på data-signalet sitt.

Sett fra CPU foregår en overføring på følgende måte (ref: leksjonen om busser):

- Når CPU skal lese fra en bestemt lokasjon i minnet, så legger CPU adressen til lokasjonen ut på adressebussen. Minnet vil lete frem rett lokasjon og legge den på databussen.
- Når CPU skal skrive til en lokasjon i minnet, så legger CPU adressen til lokasjonen ut på adressebussen, og data som skal skrives på databussen. Minnet vil legge innholdet fra databussen på den adresserte lokasjonen.

1.5.3. Dekoder

Hvordan vet vi akkurat hvilke celler som skal aksesseres? Jo, det ser vi av adressen til minnelokasjonen. Adressen på adressebussen skal altså bestemme hvilke celler vi skal sette *Velg*-signalet på.

Adressen på adressebussen er et bitmønster som består av et visst antall bits, og ut fra dette må vi få laget et signal som setter *Velg*-signalet på de riktige cellene i lagerbrikken. Til dette brukes en dekoder.

En dekoder er en elektronisk krets som har en adresse som innganger og et antall binære signaler som utganger. En eneste utgang er til enhver tid høy, og alle andre er lave. Hvilken utgang som er høy er avhengig av hvilken adresse som ligger på inngangene.

En en-bits dekoder har én inngang og to utganger, fordi vi med en bit kan velge mellom to utganger. En to-bits dekoder har to innganger og fire utganger, fordi vi med to bits kan velge mellom fire utganger. En trebits dekoder har 8 utganger. Og så videre.

Adressen på inngangen er altså et bitmønster som består av en blanding av 0-er og 1-ere. På utgangssiden er bare ett av signalene høyt; nemlig akkurat det utgangssignalet som er adressert.

Som eksempel ser vi på en åttebits dekoder. Den har en åttebits adresse som input, og 256 utgangssignaler nummerert fra 0 til 255. Det adresserte utgangssignalet er høyt, de øvrige er lav. Hvis for eksempel bitmønsteret $0101011_2 = 43_{10}$ ligger på inngangen, vil utgangssignal nummer 43_{10} være høyt og alle de andre vil være lav.

Figur 2 viser virkemåten ved lesing av primærminnet.

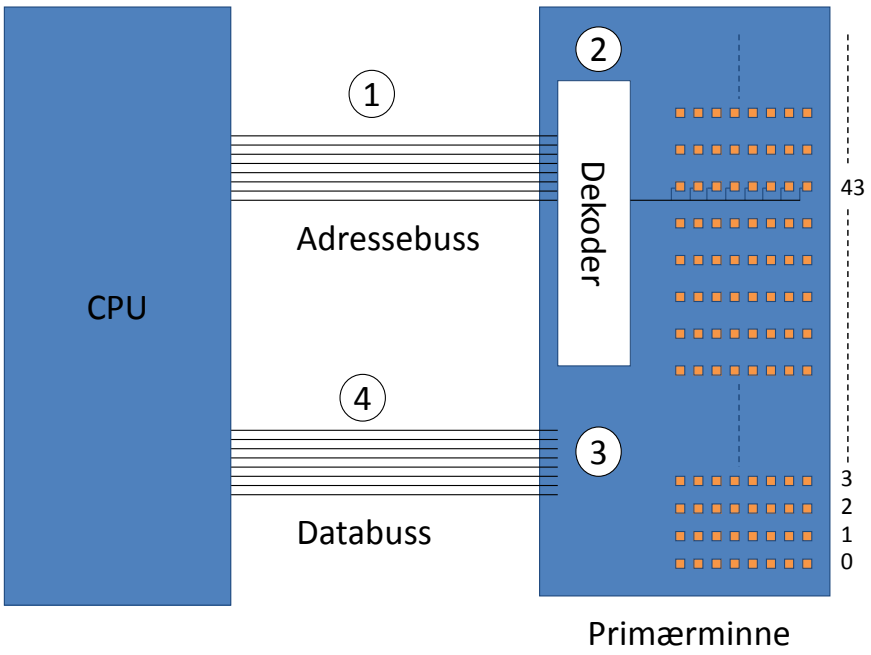
1.5.4. Ordbreddeorganisering

Hvis vi lager et primærminne på den måten som er vist i Figur 2 så får hele primærminnet plass i en eneste brikke.

Dette er en vanlig måte å organisere ROM-brikker på. Fordelen er at alle bitene som inngår i et ord ligger i en og samme brikke.

Teknikken har imidlertid en alvorlig ulempe. Den krever en svært avansert dekoder hvis minnebrikken skal inneholde mange lokasjoner. For eksempel: Antar vi en ordbredde på en byte, og at minnebrikken skal kunne lagre 1 KByte, må vi ha en 10 bits dekoder fordi 1 K = 2^{10} = 1024. Skal vi lagre 4 KByte på brikken må vi ha en 12 bits dekoder.

Denne måten å organisere cellene på kalles ordbreddeorganisering. Dette fordi hele ordet får plass i en og samme brikke.



Figur 2. Lesing fra minnet. Figuren viser et primærminne med en rekke adresserbare lokasjoner a 8 bits. Cellene er vist som små orange firkanter. Figuren viser hvordan dekoderen brukes til å plukke ut en bestemt lokasjon i minnet. 1: Overføringen begynner med at CPU legger en adresse på adressebussen. 2: Dekoderen bruker adressen til å plukke ut en bestemt lokasjon i minnet. I dette tilfellet velges lokasjon nr 43. Dekoderen virker slik at den setter Velg-signalet til alle celler i dette ordet til 1 (og velg-signalet til alle de andre cellene i primærminnet til 0). 3: Innholdet av cellene i den utvalgte lokasjonen legges på databussen. 4: CPU leser verdien fra databussen.

1.5.5. Kvadratisk organisering

Produksjonsteknisk er det faktisk ganske enkelt å pakke sammen svært mange minneceller i en integrert krets. Det som er vanskelig er å lage store dekodere, fordi dekoderen er en relativt komplisert krets hvis den skal skille mellom svært mange lokasjoner

Isteden for å lage én minnebrikke som inneholder hele primærminnet, er det vanlig å spre minnecellene utover flere minnebrikker.

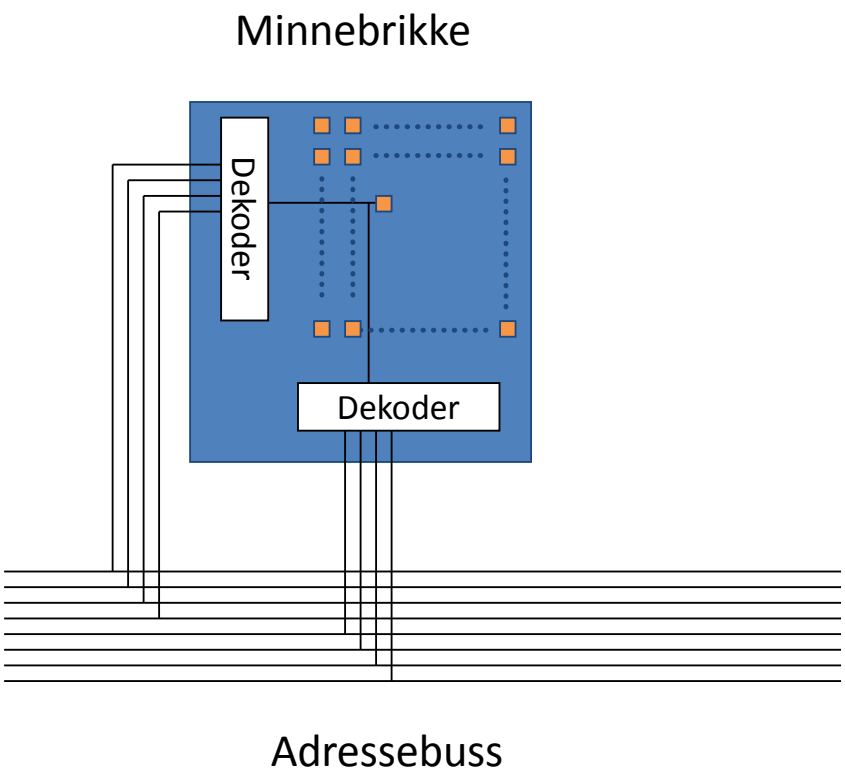
Ytterpunktet blir å bruke en brikke for hver bit i ordet. Da lar vi en brikke inneholde bit 0 til alle lokasjoner, la en annen brikke inneholder bit 1, en tredje brikke inneholder bit 2 osv. Vi trenger like mange brikker som vi har bits i ordet.

Da får vi mange flere brikker å holde styr på, men nå skal vi se at vi til gjengjeld kan greie oss med enklere dekodere.

Det vi gjør er at vi organiserer minnecellene i hver av brikkene i form av en kvadratisk matrise. Vi setter cellene i like mange rekker og søyler. Å velge en av dem blir dermed omtrent likedan som å velge et punkt i et xy-koordinatsystem. Vi velger ut en bestemt celle ved å angi hvilken rekke og hvilken søyle cellen befinner seg.

Fordelen med dette er at vi kan bruke halvparten av bitene i adressen til å velge rekke, og den andre halvparten til å velge søyle. Dekoderne blir enklere nå, fordi vi bare bruker halve adressen til hver av dem.

Oppbyggingen til en minnebrikke som bruker slik kvadratisk organisering er vist i Figur 3.



Figur 3. Minnebrikke med kvadratisk organisering. Cellene er organisert som en matrise, og man velger ut en bestemt celle med å angi hvilken rekke og søyle cellen ligger i. Halvparten av adresselinjene brukes til å velge rekke. Den andre halvparten brukes til å velge søyle.

Nå må vi altså bruke flere brikker slik Figur 4 viser. Vi må bruke like mange brikker som vi har bits i ordene. Hvis minnet er organisert som bytes trenger vi altså 8 brikker[1]. Med 16-bits ordbredde trengs 16 brikker.

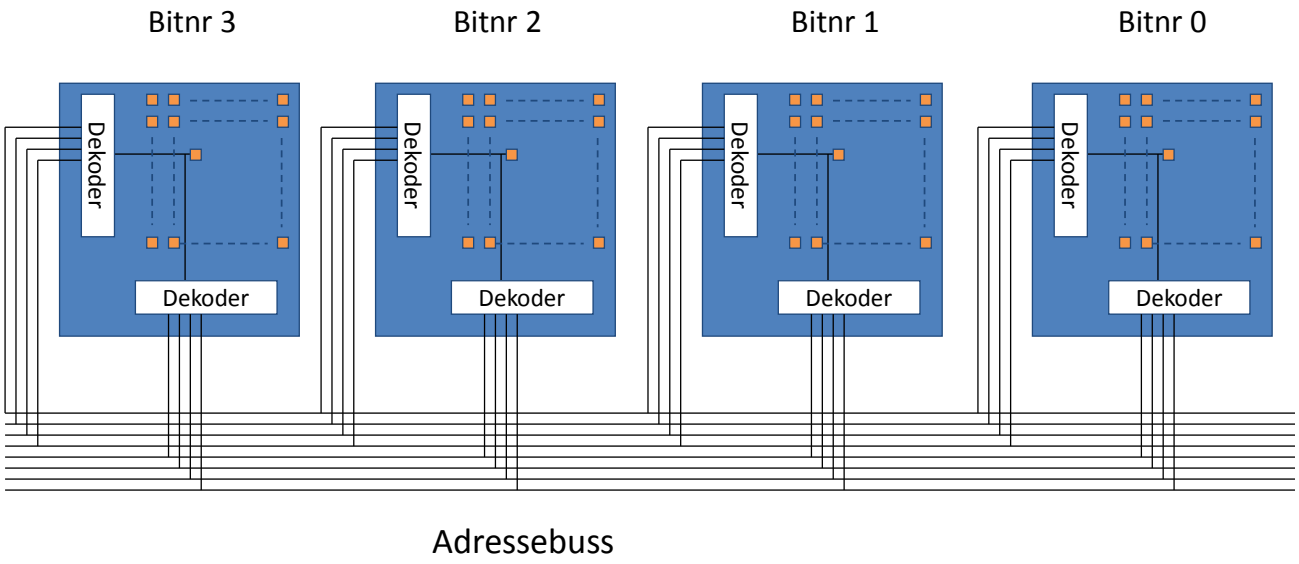
Dette er en svært vanlig måte å organisere minne på - spesielt RAM-brikker. Lagrings-kapasiteten til en slik brikke angis i antall *bits* som kan lagres på brikken. Typiske størrelser er 16 Mb, 64, 256 Mb, … (Mb er megabits). Etter som teknologien har blitt bedre har

[1] Ofte brukes det en bit i tillegg til de bitene som inngår i ordet. Denne ekstra biten kalles paritetsbit og er en enkel form for feilsjekking. Hvis det brukes paritetsbit vil man trenge 9 brikker ved 8-bits ordbredde. I denne leksjonen ser vi bort fra paritetsbiten og antar at det brukes like mange brikker som det finnes bits i et ord.

brikkene fått stadig større kapasitet. Typisk dobles antall rekker og søyler - noe som fører til firedobling av kapasiteten - ca hvert 3. år.

Legg merke til at lagerkapasiteten til hver enkelt lagerbrikke måles i antall *bits* som lagres på dem, og ikke antall bytes.

Fordelene med kvadratisk organisering er at hver dekoder blir enklere fordi de bare behandler halvparten av adresseinformasjonen. Dersom vi er i stand til å lage 10-bits dekodere, så kan vi altså bruke 20 bits adresse. 20 bits adresse skiller mellom 1M forskjellige celler ($1M = 2^{20} \approx 1$ million).



Figur 4. Bruk av kvadratisk organisering. Her brukes fire brikker for å lage et primærminne med firebitsord. Vi trenger like mange brikker som vi har bits i ordene.

1.5.6. Andre organiseringer av minnebrikken

Nå har vi sett at vi kan organisere brikkene med ordbreddeorganisering eller med kvadratisk organisering.

Men man kan også organisere primærminnet med brikker som bruker en mellomting mellom disse to.

For eksempel kan man bygge opp minnet med brikker som lagrer fire og fire bits.

Det har i liten grad festet seg noen standard, så man finner brikker med de forskjelligste organiseringer.

1.6. Moduler og minnekapasitet

Det trengs altså mange lagerbrikker for å bygge opp primærminnet til en datamaskin. For eksempel har vi sett at man trenger 8 brikker med kvadratisk organisering når hver lokasjon består av en byte. Disse brikkene kaller vi *en modul*. Dersom vi ønsker å utvide minnet må vi bruke flere slike moduler. Husk at minnekapasitet for det totale primærminne oppgis som antall bytes i minnet.

1.6.1. Organisering av minne på PC; SIMM og DIMM

Heldigvis er det slik at vi sjelden trenger å bry oss om hva slags minnebrikker som brukes på en PC. På moderne PCer brukes enten noe som kalles SIMM-kort, eller de bruker såkalte DIMM-kort. SIMM står for *Single Inline Memory Module*, og DIMM står for *Dual Inline Memory Module*.

En SIMM og en DIMM er ganske enkelt et lite printkort. På dette printkortet står alle brikkene som inngår i en modul. Produsenten setter sammen brikker på printkortet og selger det hele som en enhet. Kjøperen trenger følgelig ikke tenke på hva slags brikker som brukes.

En SIMM er organisert som 32 bits minne. På Pentium-maskiner leses eller skrives 64 bits samtidig. På gamle maskiner ble derfor SIMM-kort alltid brukt parvis[2].

Dette gjaldt helt til en lyst hode innså at printkorter har både en over- og en underside, og at vi kan sette innholdet fra en SIMM på oversiden, og innholdet fra en annen SIMM på under-siden. Dette ble kalt DIMM.

På DIMM er altså de to modulene som inngår i en bank samlet på ett kort. Man ser enkelt forskjell på SIMM og DIMM ved at SIMM har lagerbrikker montert bare på en side av print-kortet, mens DIMM har lagerbrikker på begge sider.

SIMM og DIMM er altså ikke betegnelsen på noen RAM-typer, men forteller hvordan minnet er organisert.

1.6.2. Forskjellige DRAM-typer

I annonser for PCer florerer det av betegnelser på forskjellige minnetyper. SDRAM og DDR SDRAM er de vanligste. Alle disse er DRAM, og i en senere leksjon skal vi se nærmere på disse typene. Foreløpig holder det å vite at selve lagercelle-teknologien er akkurat den samme på alle disse DRAM-typene, men teknikken for å hente ut data er noe forskjellige.

1.7. Hvor ofte ønsker vi å lese fra minnet?

Alle instruksjoner trenger minst en minneaksess. Nemlig for å hente instruksjonen fra minnet. Dessuten skal det ofte leses eller skrives operander i tillegg. Det vil si at prosessorens effektive hastighet i hvert fall ikke er større enn lese/skrive-hastigheten mot minnet.

Hvor mange instruksjoner kan en moderne prosessor utføre hvert sekund? En Pentium-prosessor utfører rundt 100MIPS (millioner instruksjoner pr sekund). Nyere prosessorer greier enda flere instruksjoner i sekundet. Som vi husker fra leksjon 1 blir utføringshastigheten fordoblet omtrent annet hvert år.

Moderne prosessorer kan utføre flere hundre millioner beregninger hvert sekund. Med en beregningskapasitet på 100 millioner instruksjoner pr sekund vil hver instruksjon ta 10ns. Som vi har sett i leksjonen er det bare SRAM som har så kort aksesstid at den matcher denne hastigheten. Hvis hele minnet skal bygges opp med SRAM vil minnet på maskinen bli svært dyrt, og det er ikke økonomisk forsvarlig. Vi kan bruke DRAM isteden, men det har jo mye dårligere aksesstid. Derfor vil DRAM sinke utføringshastigheten fordi det ikke greier å fôre CPU med instruksjoner og data tilstrekkelig hurtig.

---

[2] Et slikt SIMM-par kalles ofte en *bank*, fordi man alltid må sette inn et helt antall *par* med SIMM-kort ved minneutvidelser. (Vær klar over at begrepet bank også brukes i andre sammenhenger innenfor minneteknologi).

Hvordan skal vi løse dette problemet? Jo, vi bruker en teknikk som kalles cache. Teknikken ble opprinnelig utviklet for stormaskiner, og inntil for noen år siden var det utenkelig å bruke den på mikromaskiner - nå har det altså blitt helt vanlig.

Cache medfører at vi bruker både SRAM og DRAM. Vi legger de instruksjoner og data som CPU har mest bruk for i SRAM. Så lar vi det som vi sjeldnere har bruk for ligge i DRAM. CPU vil vanligvis finne det den trenger i det hurtige SRAM-minnet, slik at hastigheten blir opprettholdt. Så godtar vi at CPU av og til må vente på det som ligger i DRAM.

Cache vil bli beskrevet i en annen leksjon.