



Kunnskap for en bedre verden

TDT4110 - ITGK - Hardware

Kapittel 9: Følge Instruksjoner - Prinsipper for
Datamaskinens Virkemåte

Terje Rydland - IDI/NTNU

Læringsmål

- Vite om å forstå hvordan
 - prosessoren er oppbygd (5 deler, PC)
 - maskinens primærminne er oppbygd og virker
 - maskinens «Fetch-Execute» kretsløp virker
 - samspillet med eksterne enheter foregår
 - operativsystemet programmeres
- Husk å stille spørsmål på Piazza

Instruksjons-Utførings-Maskin

- Hva datamaskiner kan
 - Deterministisk utføre (eksekvere) instruksjoner for å behandle informasjon
 - Datamaskinen må ha instruksjoner å følge
- Hva datamaskiner ikke kan
 - Ingen fantasi eller kreativitet
 - Ingen intuisjon
 - Forstår ikke ironi, tvetydighet, forhold, sømmelighet, humor
 - Er ikke hevngjerrige eller onde
 - Er ikke målbevisst
 - Har ikke fri vilje
 - Er ikke som i Hollywoodfilmer: Terminator, Matrix, AI

Oppbygningen av en datamaskin

- 5 grunnleggende deler/subsystemer
 - Minne
 - Kontrollenhet
 - Aritmetisk/logisk enhet (ALU)
 - Input-enhet
 - Output-enhet
- Bundet sammen av
 - Busser

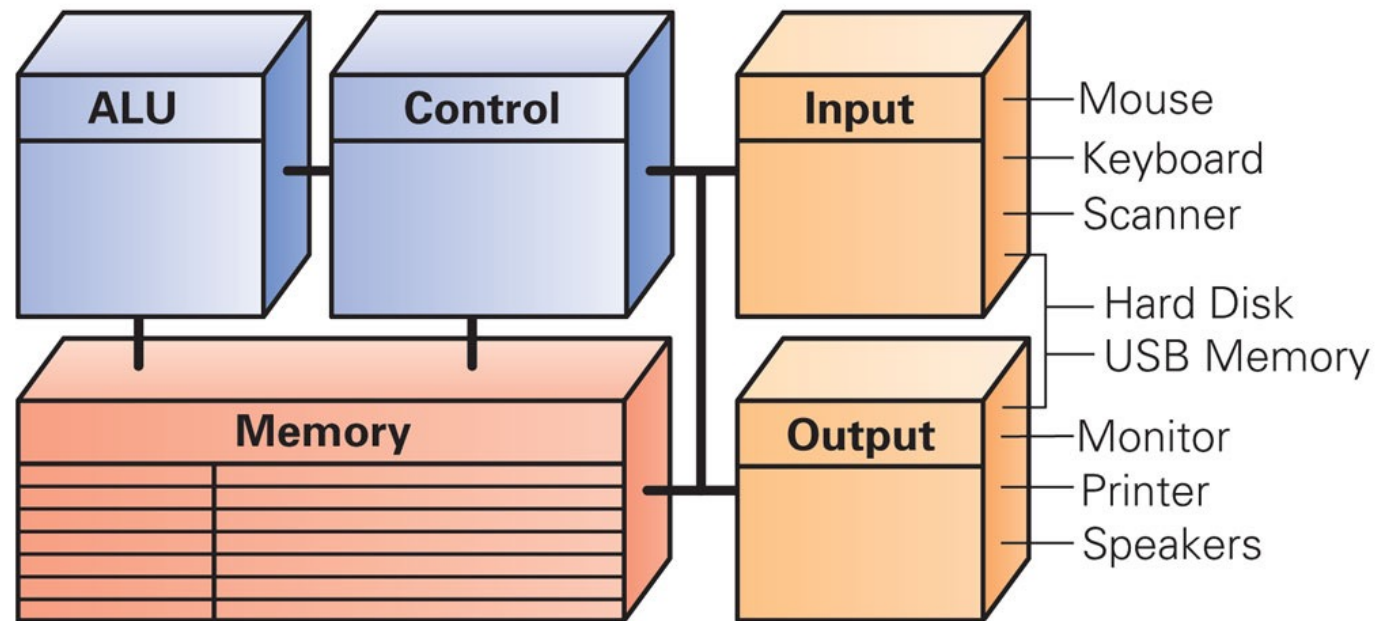


Figure 9.2. The principal subsystems of a computer.

Minne

- **Minnet (RAM)** lagrer programmet som kjører og dataene som programmet behandler
 - **Atskilte lagerplasser/lokasjoner.**
 - Hver plass består av 1 byte
 - **Adresser.**
 - Hver minnelokasjon (byte) har en adresse (hele tall som starter på null).
 - **Verdier.**
 - Minnelokasjoner tar opp eller lagrer verdier
 - **Begrenset kapasitet.**
 - En gitt størrelse—programmerere må huske at dataene kanskje ikke «får plass» i minnelokasjonen

Byte-store Minnelokasjoner

- En vanlig måte å representere de atskilte lokasjonene i minnet er ved et boks-diagram (hver boks inneholder en byte).
- Adressen til minnelokasjonen vises over boksen
- Verdien eller innholdet i lokasjonen vises i boksen

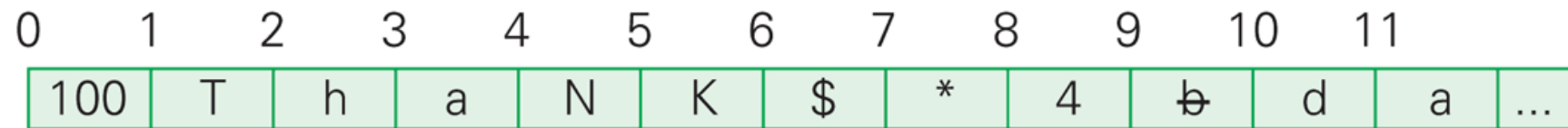


Figure 9.3. Diagram of computer memory illustrating its key properties.

Minne (2)

- 1-bytes minnelokasjoner kan lagre ett ASCII-tegn, eller ett nummer <256 (0-255)
- Bruker ofte en sammenhengende rekke av minnelokasjoner, uten å bry seg om at hver enkeltlokasjon har en egen adresse
 - Word
 - Blokk på fire bytes blir så ofte brukt som en enhet at de kalles for (minne-)ord («Words»)
- Forskjellige datatyper tar opp forskjellig plass
 - char (tar vanligvis opp 1 byte (ASCII) eller 2-4 byte (UNICODE))
 - float (8 eller 10 byte)
 - etc...
- Stort minne fører til mindre I/O

Kontrollenheten

- Maskinvare-implementasjon av **Hente/Utføre-kretsen**
 - Henter en instruksjon fra minnet
 - dekoder instruksjonen
 - og henter operandene som trengs
- En typisk instruksjon kan være på formen
 - ADD 4000, 2000, 2080 Op Dest, src1, src2
 - Numrene lagret i lokasjon 2000 og 2080 adderes. Resultatet lagres i lokasjon 4000
$$[4000] = [2000] + [2080]$$
- «Data/Operand Fetch»-steget må hente disse to verdiene og etter at de er lagt sammen må «Returner Resultat»/lagre-steget sende svaret til minnelokasjon 4000

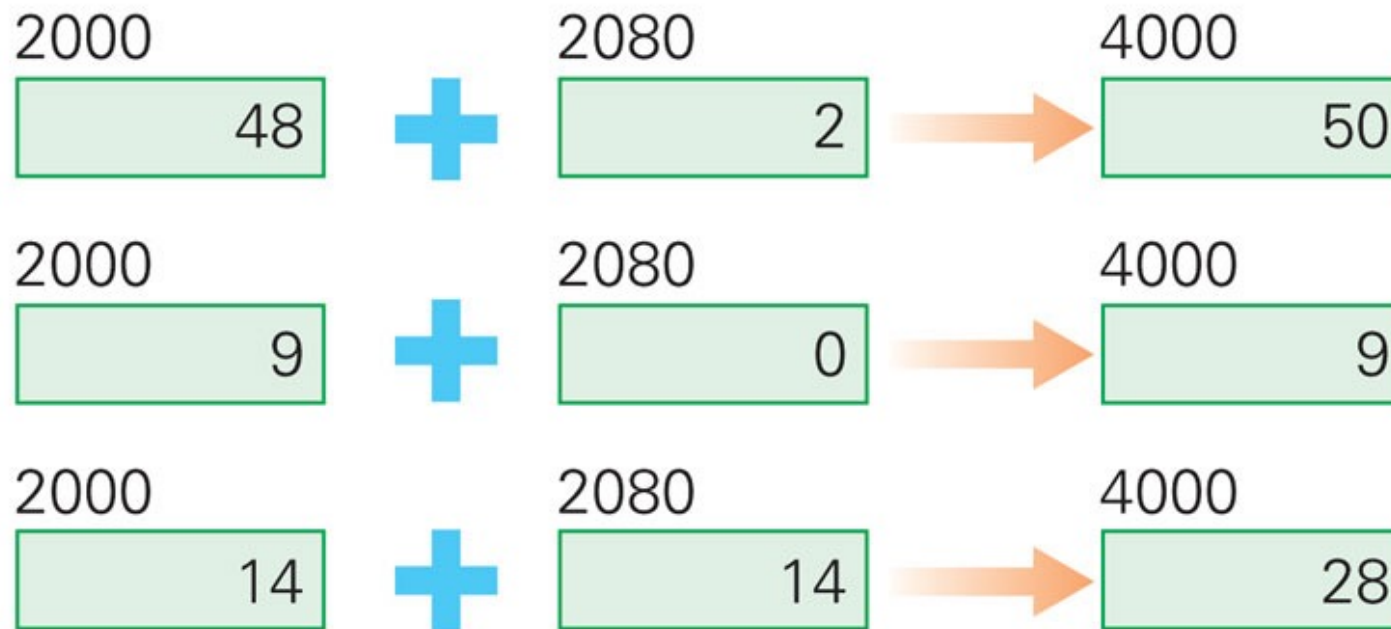


Figure 9.4. Illustration of a single ADD instruction producing different results depending on the contents of the memory locations referenced in the instruction.

Input Enhet og Output Enhet (I/O)

- De kablene og kretsene som informasjon flyttes inn og ut av datamaskinen gjennom.
- **De ytre enhetene**
 - Kobles til maskinens input/output porter
 - Regnes ikke som en del av maskinen, men som spesialiserte enheter som koder eller dekoder informasjonsflyten mellom datamaskina og den fysiske verden.

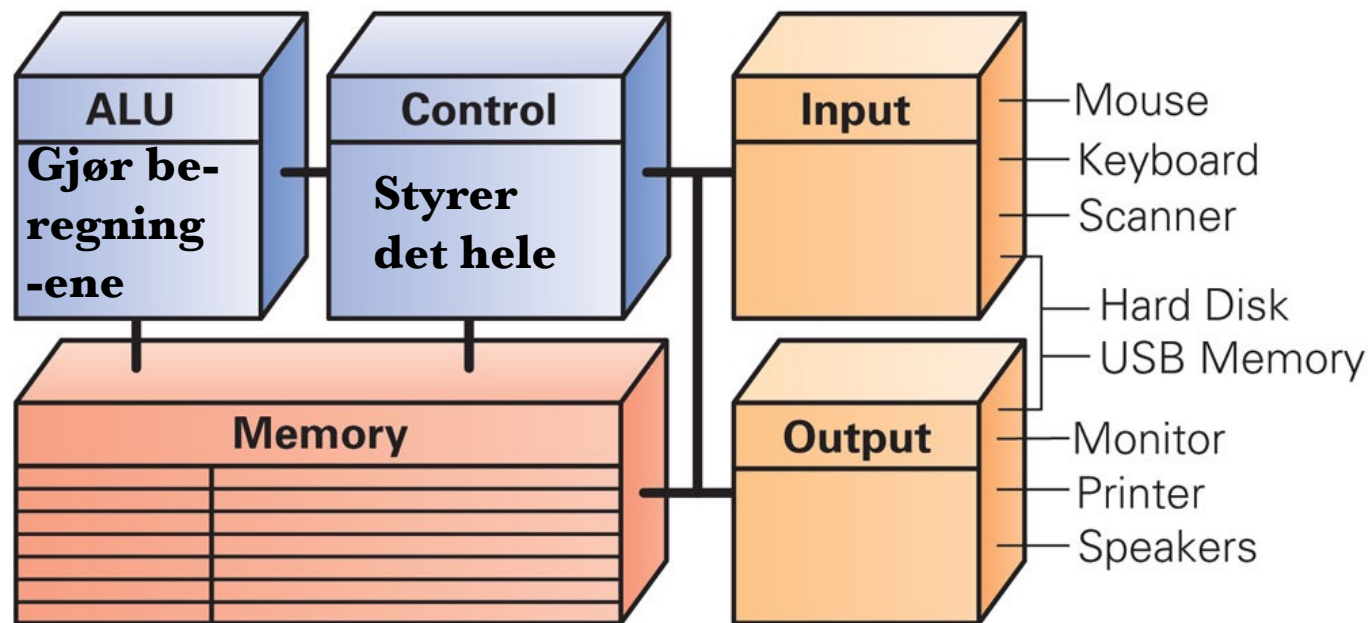


Figure 9.2. The principal subsystems of a computer.

De ytre enhetene («Peripherals»)

- Tastatur
 - koder tastetrykkene våre til binær form for datamaskina (sender en tallkode)
- Skjermen
 - dekoder informasjon fra maskinens minne og viser det på en opplyst, farget skjerm
- Harddiskene brukes både til input og output
 - lagringsenheter hvor datamaskinen legger informasjon mens den ikke trenges, og hvor den kan hentes fram igjen når den trenges igjen.
- En Driver for hver ytre enhet («peripheral»)

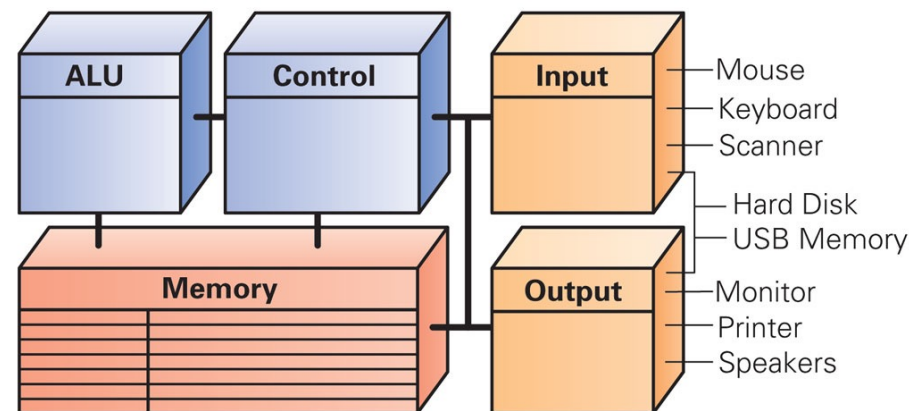


Figure 9.2. The principal subsystems of a computer.

Hente / Utføre - kretsløpet

- En fem-skrittets syklus
 - Instruksjons-henting (IF)
 - Instruksjons-dekodning (ID)
 - Data-henting (DF) / Operand-henting (OF)
 - Instruksjons-eksekvering (EX)
 - Resultat-retur (RR) / Lagre (ST = «Store»)

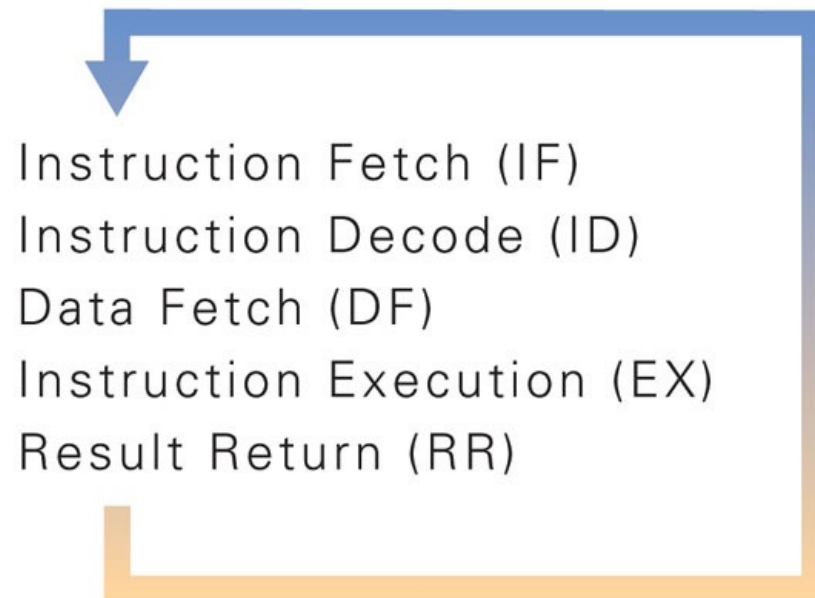


Figure 9.1. The Fetch/Execute Cycle.

Aritmetisk/Logisk Enhet (ALU)

- Utfører regneoperasjonene
- En krets i ALU kan legge sammen to tall
 - Det finnes også kretser for å multiplisere, sammenligne, osv.
- Gjør vanligvis jobben i Instruksjonsutføringssteget (EX) i syklusen, men instruksjoner som bare flytter på data bruker ikke ALU
- Data/Operand Hente-steget (DF) i syklusen henter de verdiene som ALUen trenger å jobbe med (operandene til operatoren)
- Når ALUen fullfører operasjonen sørger Resultat Returnerings/Lagrings-steget for å flytte svaret fra ALUen til den minnelokasjonen som er spesifisert i instruksjonen

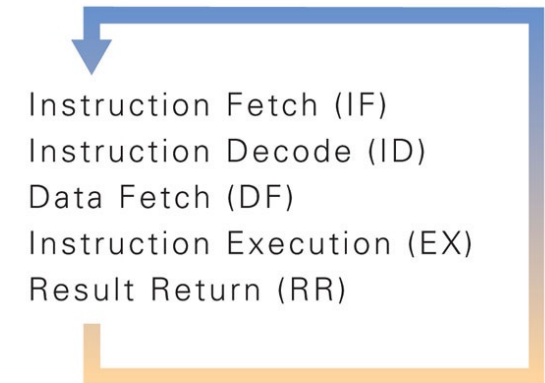


Figure 9.1. The Fetch/Execute Cycle.

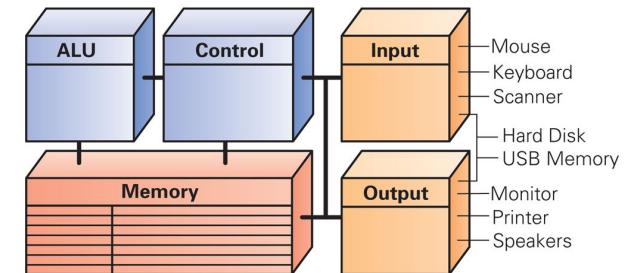
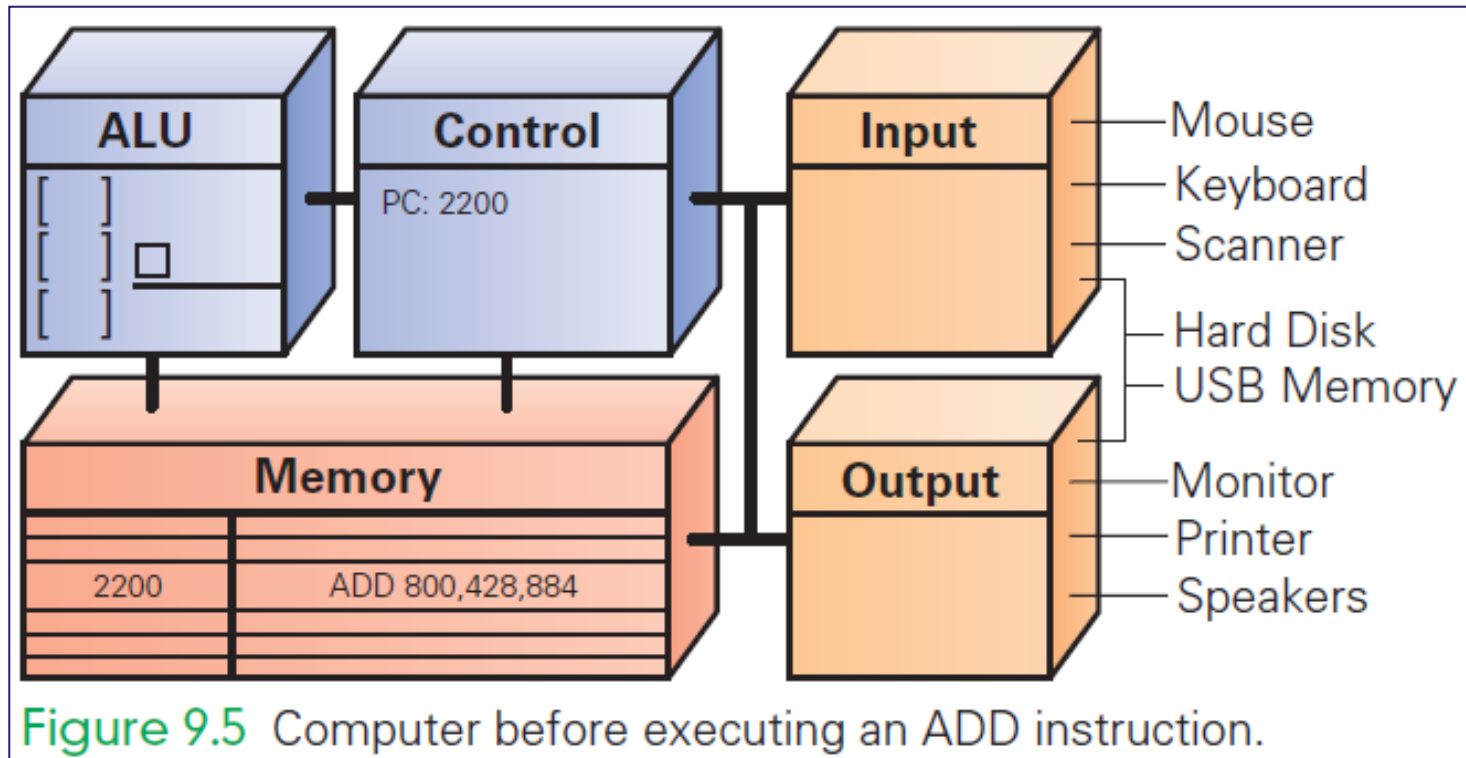


Figure 9.2. The principal subsystems of a computer.

Instruksjonstolkning

- Programkjørings-prosessen
 - Maskinen tolker våre kommandoer, men i sitt eget språk.
- Eksempel fra boka:
 - ADD 800, 428, 884
 - Legg sammen verdiene fra minne 428 og 884 og lagre resultat in minneposisjon 800
 - Før F/E-syklusen begynner er bare noen minnelokasjoner og PC-en synlige i kontroll-kretsen

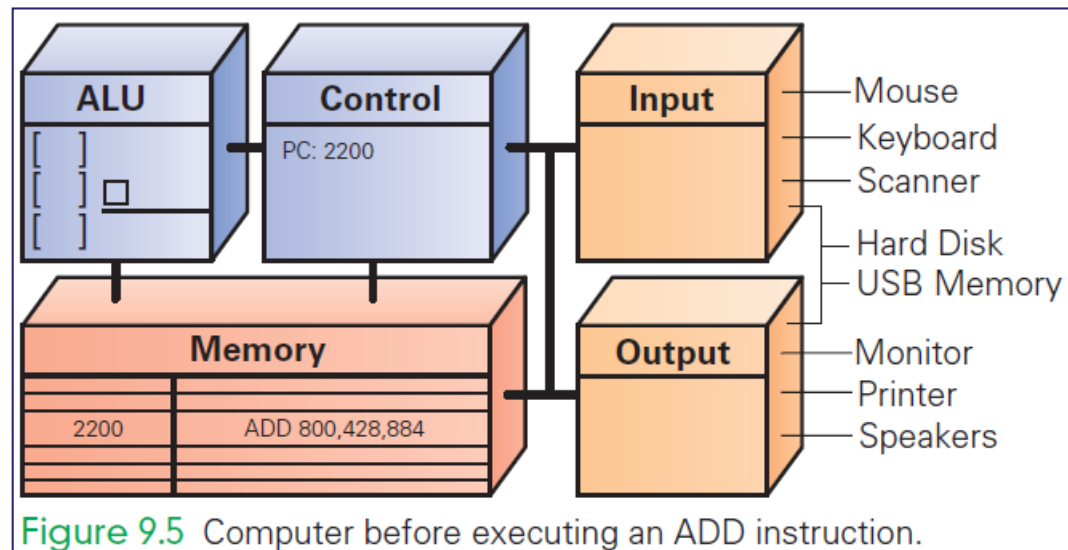
ADD 800, 428, 884



ADD the values found in memory locations 428 and 884 and store the result in location 800

Programtelleren: PC-ens PC (Counter)

- Adressen til den neste instruksjonen lagres i kontroll-delen til maskinen. Den kalles programteller (PC)
- Hvis instruksjoner bruker 4 byte minne, må neste instruksjon være på lokasjon $PC + 4$, 4 bytes lenger avgårde i sekvensen/programmet (generelt sett)
- Maskinen plusser på fire i PC, så når F/E-syklusen kommer tilbake til InstruksjonsHente-steget neste gang, så «peker» PC på den neste instruksjonen



Forgrening og Hopp-Instruksjoner

- Enkelte instruksjoner kan inkludere en adresse for neste steg. Det vil endre PCen, slik at istedenfor å gå automatisk til PC+4, så «hopper» den eller «bryter av» («brancher») til den spesifiserte minnelokasjonen for neste instruksjon

IF: Kjøringen begynner med å flytte instruksjonen i adressen gitt i PC-en fra minnet til kontrollenheten

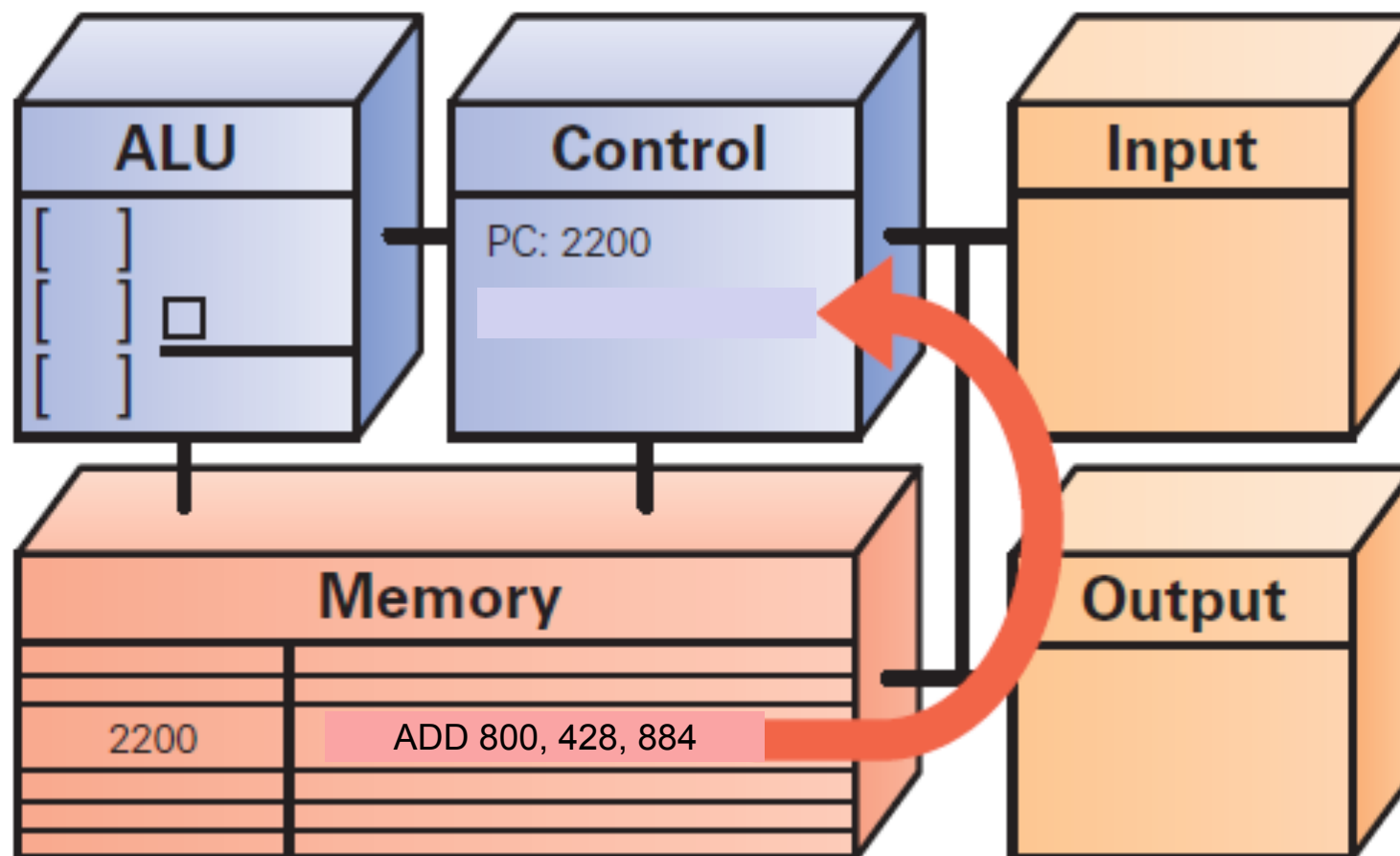


Figure 9.6 Instruction Fetch: Move instruction from memory to the control unit.

Så snart instruksjonen er hentet, kan PC-en bli gjort klar til å hente neste instruksjon

ID: Instruksjonstolkning (2)

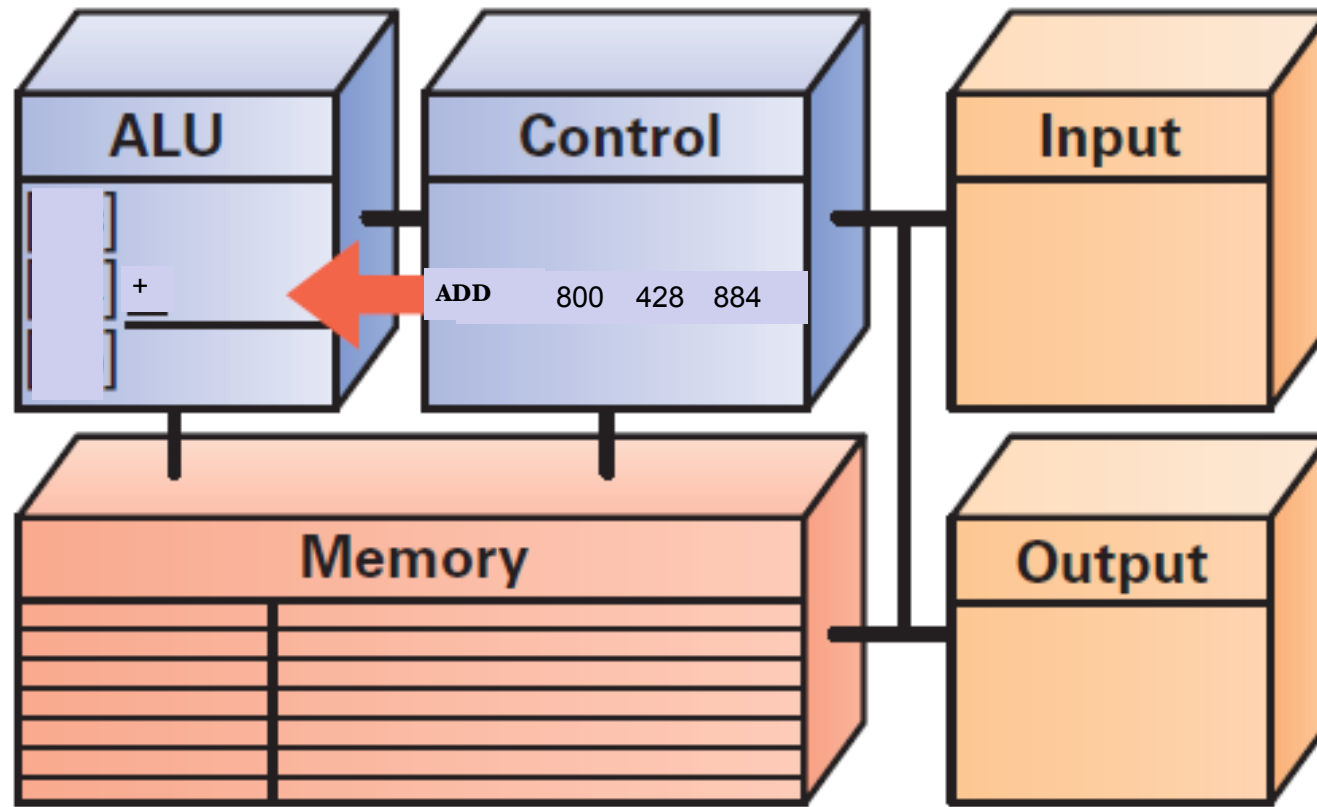


Figure 9.7 Instruction Decode: Pull apart the instruction, set up the operation in the ALU, and compute the source and destination operand addresses.

DF: Instruksjonstolkning (3)

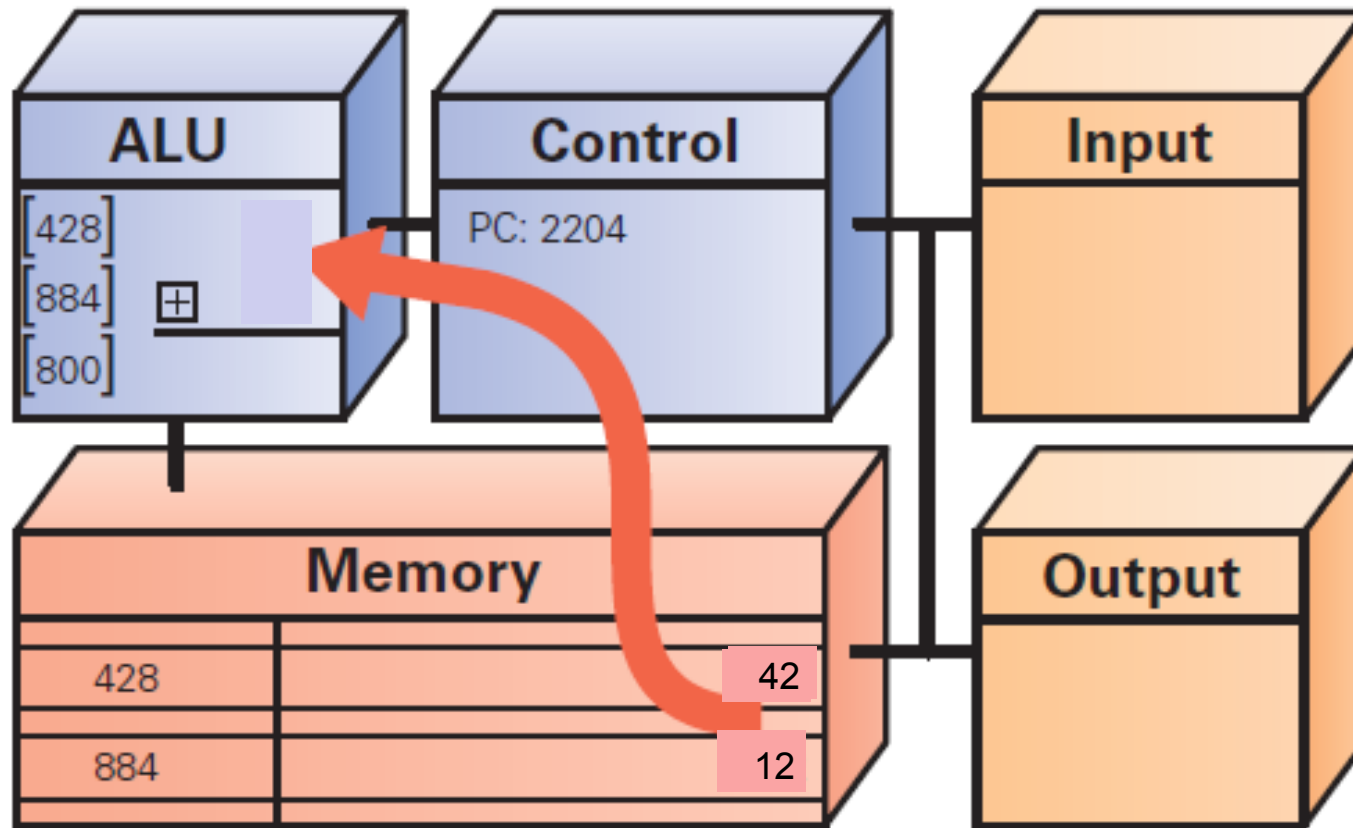


Figure 9.8 Data Fetch: Move the operands from memory to the ALU.

EX: Instruksjonstolkning (4)

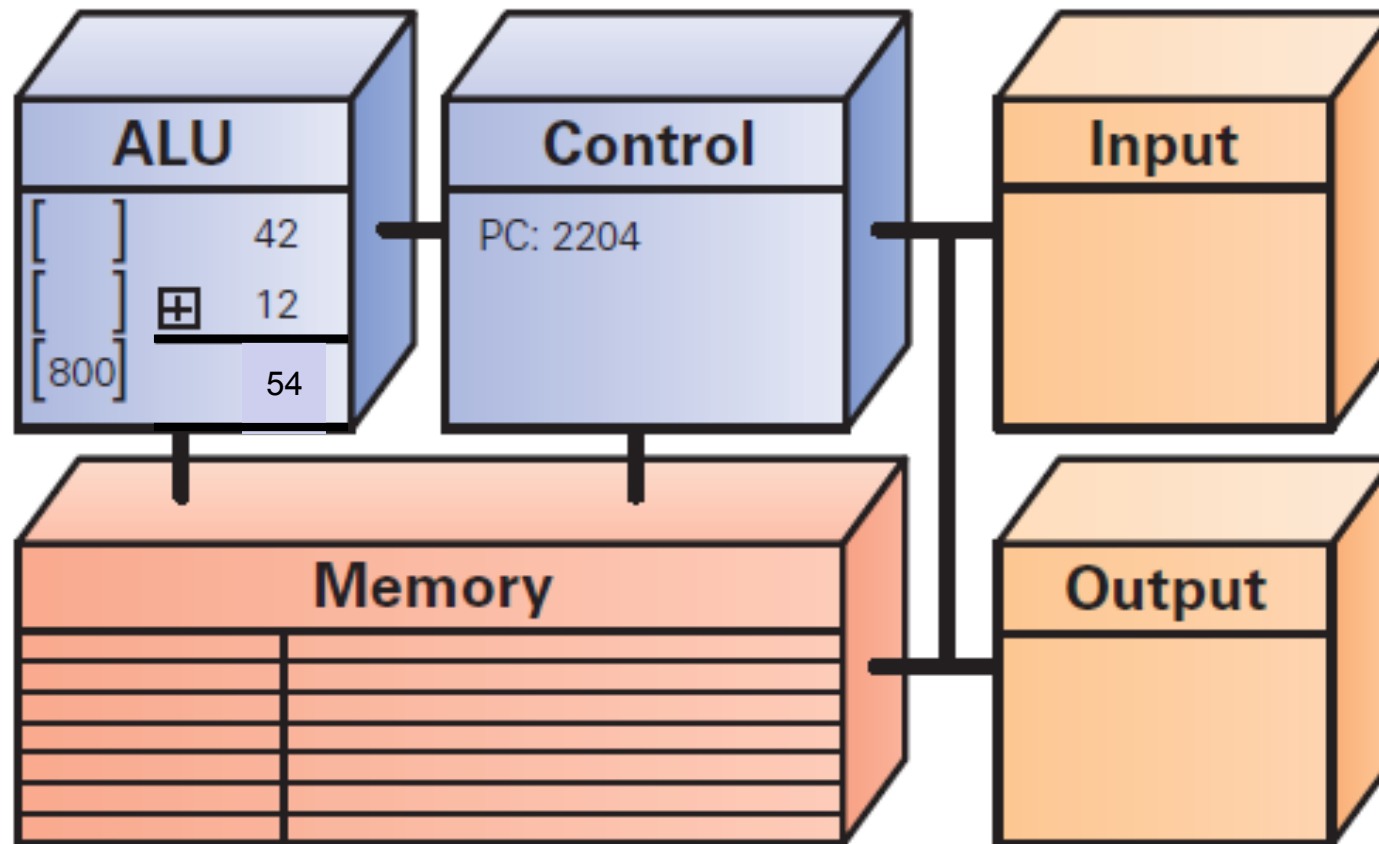


Figure 9.9 Instruction Execute: Compute the result of the operation in the ALU.

RR

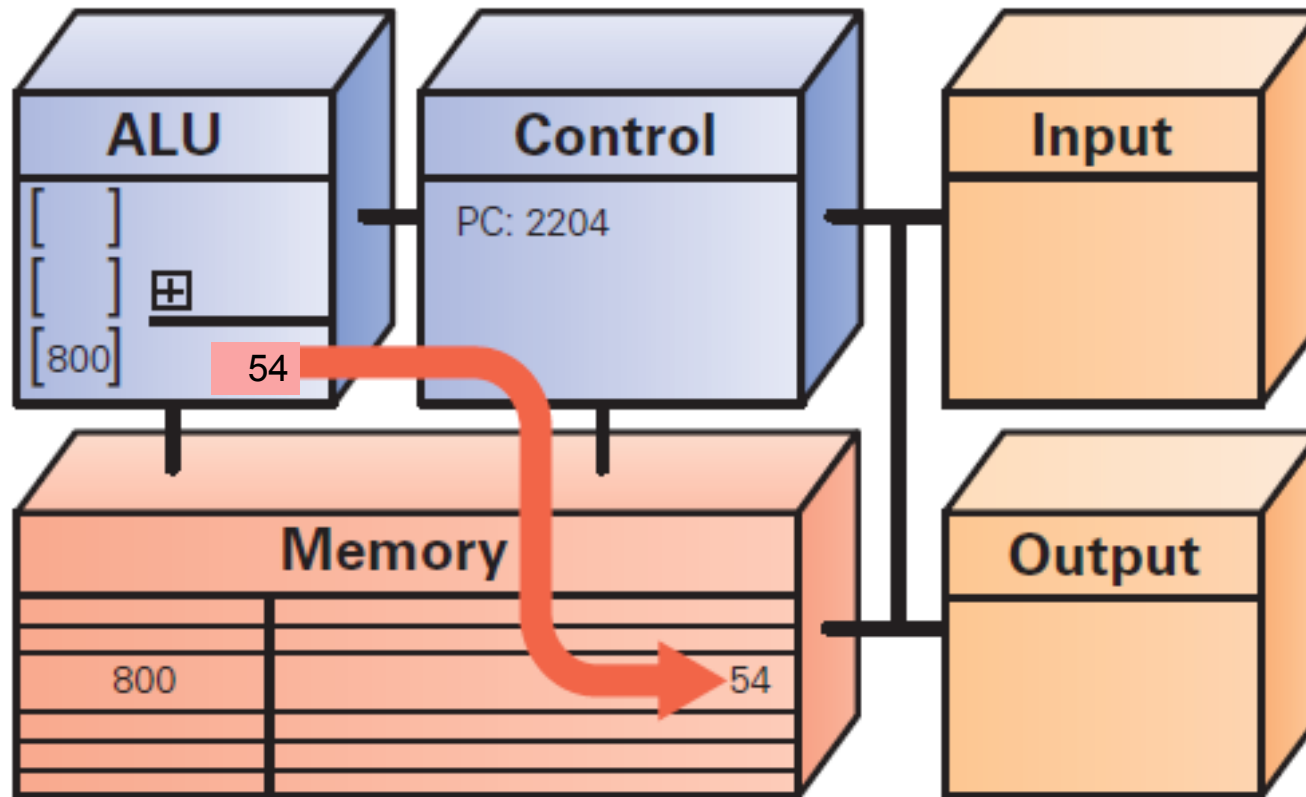


Figure 9.10 Result Return: Store the result from the ALU into the memory at the destination address.

Mange, mange enkle operasjoner

- Datamaskiner kan bare utføre omkring 100 forskjellige instruksjoner
 - Omtrent 20 forskjellige typer operasjoner (Det trengs forskjellige instruksjoner for å legge sammen enkelt-bytes, «ord», desimaltall, osv.)
- Alt datamaskinene gjør må reduseres til en kombinasjon av disse primitive, hardkodete instruksjonene

Eksempler på andre instruksjoner

- Foruten **ADD**, **MULT** (Multipliser) og **DIV** (Dividere), så finnes også følgende andre instruksjoner
 - Flytt bit-ene i et «ord» til venstre eller høyre, fyll de tømte plassene med null, og kast bort bit-er som detter utenfor enden
 - Beregn logisk **AND** (test om par av bit-er begge er «sann»), og logisk **OR** (test om minst en bit i paret er «sann»)
 - Test om en bit er null eller ikke-null, og hopp til et nytt sett instruksjoner avhengig av utfallet
 - Flytt informasjon i minnet
 - Sjekk signaler fra input/output-enhetene

Drive F/E-syklusen

- Datamaskiner får sine imponerende evner ved å kjøre mange av disse enkle instruksjonene hvert sekund.
- Maskin-klokka: Bestemmer hastigheten på F/E-syklusen
 - Måles i gigahertz (GHz), antall milliarder sykluser per sekund

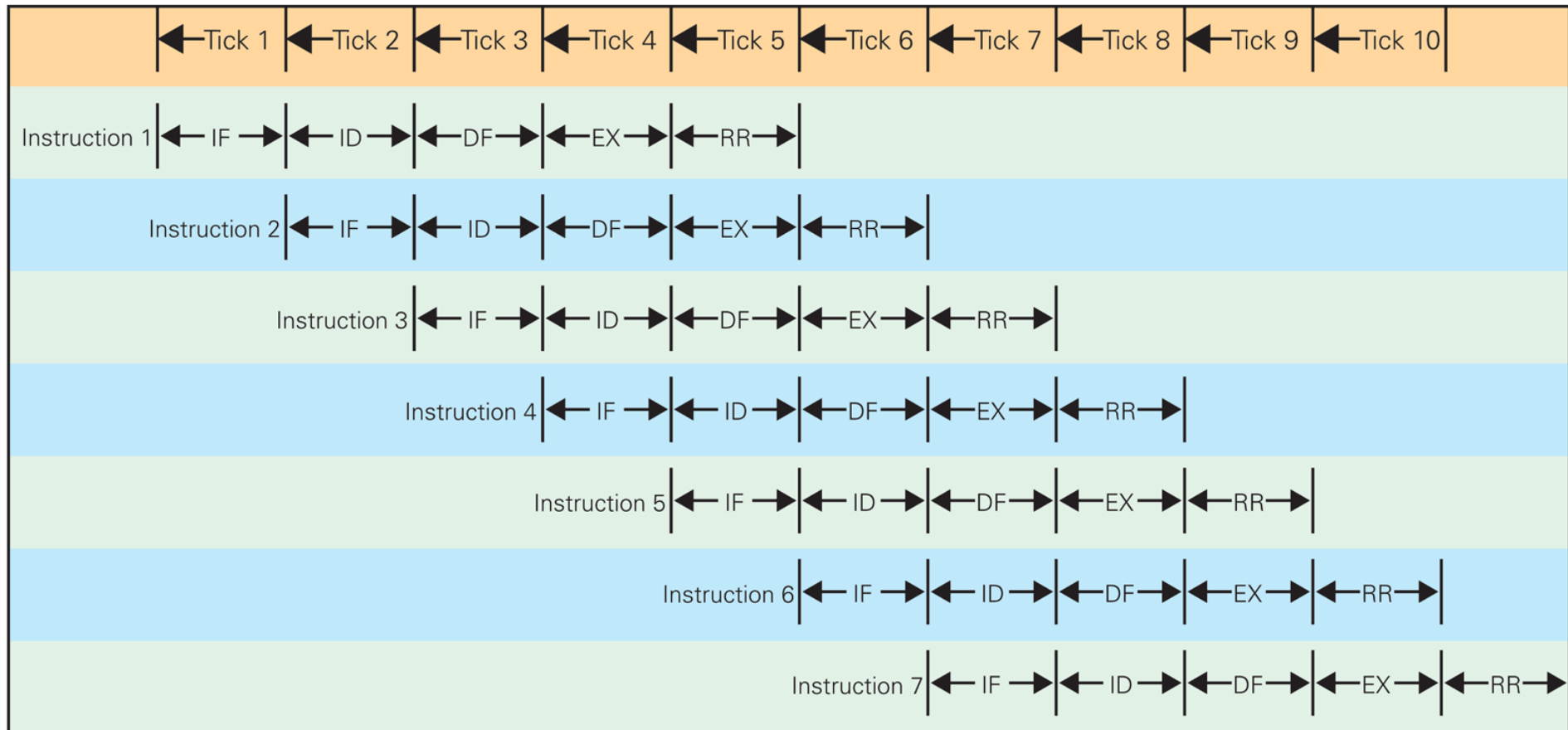
1000^1	kilo-	$1024^1 = 2^{10} = 1,024$	milli-	1000^{-1}
1000^2	mega-	$1024^2 = 2^{20} = 1,048,576$	micro-	1000^{-2}
1000^3	giga-	$1024^3 = 2^{30} = 1,073,741,824$	nano-	1000^{-3}
1000^4	tera-	$1024^4 = 2^{40} = 1,099,511,627,776$	pico-	1000^{-4}
1000^5	peta-	$1024^5 = 2^{50} = 1,125,899,906,842,624$	femto-	1000^{-5}
1000^6	exa-	$1024^6 = 2^{60} = 1,152,921,504,606,876,976$	atto-	1000^{-6}
1000^7	zetta-	$1024^7 = 2^{70} = 1,180,591,620,717,411,303,424$	zepto-	1000^{-7}
1000^8	yotta-	$1024^8 = 2^{80} = 1,208,925,819,614,629,174,706,176$	yocto-	1000^{-8}

Figur 9.11 Standard prefikser fra Systéme International (SI) konvensjonen for vitenskapelige målinger. Generelt så refererer et prefiks til en potens av 1000, unntatt når mengden (for eksempel minne) telles binært; For binære mengder referer prefikset til en potens av 1024, som igjen tilsvarer 2^{10} .

1024, which is 2^{10} .

Hvordan forbedre klokke-hastigheten?

- Prøver å starte en instruksjon på hvert klokke-tikk
- Overlater avslutning av instruksjonen til andre kretser (dette kalles «pipelining», som er en slags parallell-utføring. Forskjellig fra flerkjerne parallel-prosessering)
 - Fem instruksjoner kan behandles samtidig
- Kan en datamaskin med 1 GHz klokke virkelig utføre en milliard instruksjoner per sekund?
 - Ikke helt presist mål. Datamaskinen klarer ikke alltid å starte en ny instruksjon på hvert tikk, men klarer noen ganger å starte mer enn en instruksjon samtidig



Figur 9.12 Skjematisk diagram for en parallell-prosessor «**pipeline**» Hente/Eksekvere «F/E»-syklus. På hvert tikk starter IF en ny instruksjon, og sender den deretter videre til ID (Instruksjons-Dekoding) enheten; ID-enheten arbeider med instruksjonen den fikk, og når den er ferdig sendes instruksjonen videre til DF (Datahentning «Fetch»)-kretsen, og så videre. Mens «røret» er fullt utføres fem instruksjoner samtidig, og en instruksjon blir ferdig på hvert klokke-tikk, som gjør at det ser ut som om maskinen kjører en instruksjon per tikk.

Programvare (Software)

- En maskins syn på programvare
 - Maskinen ser **binære objekt-filer**, som er en lang sekvens av 4-bytes «ord» (0'ere og 1'ere)
- **Assemblerspråk**
 - En alternativ form for maskinspråk som bruker bokstaver og normale tall, slik at mennesker kan forstå det
 - Maskinen leser **assemblerkode**, og etter hvert som den finner ord slår den opp i en tabell for å konvertere ordene til **binærkoder**, tall konverteres til binær form, og så settes de binære kodene sammen («assembles») til en instruksjon
 - De forskjellige prosessorene har forskjellig assemblerspråk.

Programvare (Software, 2)

- Høynivå programmerings språk
 - De fleste moderne program er skrevet med høynivå notasjon, som deretter **kompileres** (oversettes) til assemblerspråk, som igjen assembles til binærkode
 - Har spesielle uttrykksformer som hjelper programmerere å gi kompliserte instruksjoner
 - Eksempel: Tre-delt «if-statement» (hvis)
 - Ja/Nei spørsmål som skal testes
 - Instruksjon som skal utføres hvis testen er «true»
 - Instruksjon som skal utføres hvis testen er «false»

Oversetting

- Høynivå
- $\text{Sum} = a + b + c$

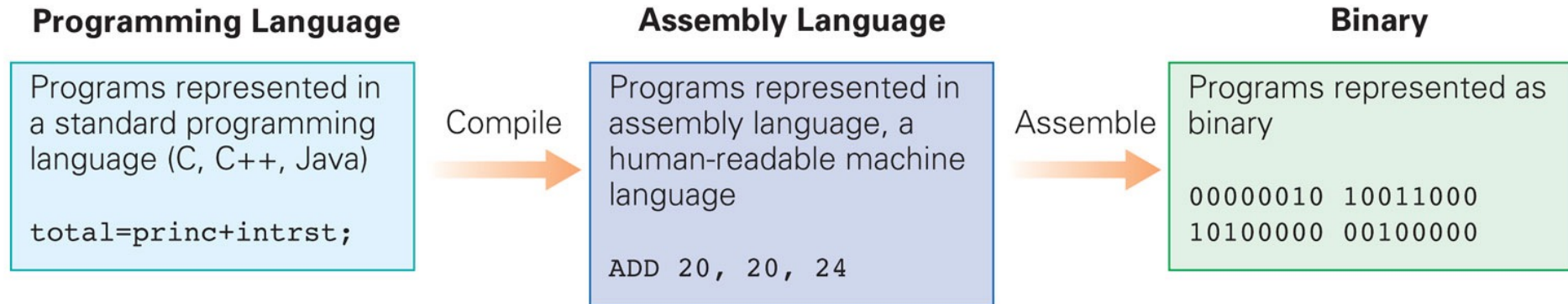


Figure 9.13. The three primary forms of encoding software: programming language, assembly language, and binary machine language.

- Assembler
- `ADD 100, 20, 24`
- `ADD 100, 100, 28`

```

1 factorial:
2  bgtz $a0, doit          # Argument > 0
3  li   $v0, 1             # Base case, 0! = 1
4  jr   $ra                # Return
5  doit:
6  sub  $sp,8              # Allocate stack frame
7  sw   $s0,($sp)          # Position for argument n
8  sw   $ra,4($sp)         # Remember return address

9  move $s0, $a0           # Push argument
10 sub  $a0, 1             # Pass n-1
11 jal  factorial          # Figure v0 = (n-1)!
12 mul  $v0,$s0,$v0        # Now multiply by n, v0 = n*(n-1)!

13 lw   $s0,($sp)          # Restore registers from stack
14 lw   $ra,4($sp)         # Get return address
15 add  $sp,8              # Pop
16 jr   $ra                # Return

```

Figure 9.14 An assembly language program to compute $n!$ for the MIPS computer.


```
1 var j, frame = -1, duration = 150, timeout_id = null;
2 var images = new Array(20);
3 function advance() {
4   for (j = 0; j < 19; j++) {
5     document.images[j].src = document.images[j+1].src;
6   }
7   if (frame == -1) test
8     document.images[19].src = pics[randNum(8)].src; true instructions
9   else
10    document.images[19].src = pics[frame].src; false instructions
11   timeout_id = setTimeout("animate()", duration);
12 }
```

Figure 9.15 A fragment of program text written in a high-level language.

Operativsystem (OS)

- Grunnleggende operasjoner som er nødvendig for å kunne bruke maskinen effektivt, men som ikke er bygd inn i maskinvaren
- De tre mest brukte Operativsystemene:
 - Microsoft Windows
 - Apple's Mac OS X
 - Unix / Linux
 - Mobiltelefonoperativssystemer øker (Android, iOS,...)
- OS-et tar seg av «booting» (oppstart), minnehåndtering, enhetshåndtering, Internettforbindelse, filhåndtering

Programmering

- Programmerere bygger på tidligere utviklet programvare for å gjøre jobbene sine lettere
- Eksempel: GUI Programvare
 - Ramme rundt vindu, skyveknapper (slide-bars), knapper, pekere, osv. er ferdig pakket inn i OS-et og tilgjengelig for programmerere

Kombiner alle idéene

- Slå alle idéene fra forrige og denne uken sammen
 - Start med en informasjons-prosesserings oppgave
 - Oppgaven utføres av en applikasjon, implementert som en del av et stort program, skrevet i et høynivå-språk som C eller Java
 - Programmet utfører spesifikke operasjoner; standard-operasjoner som «print» eller «lagre» utføres av Operativ-Systemet (OS'et).
 - Programs kommandoer kompiles til assembler-språk instruksjoner
 - Assembler-instruksjonene oversettes videre til binær kode
 - Binære instruksjoner lagres på hard-disken (sekundær-minne)
 - Applikasjonens instruksjoner flyttes til RAM (primær-minne)

Kombiner alle idéene (2)

- Hent/Eksekver (F/E)-kretsløpet utfører instruksjonene fra minnet (RAM)
- Alle maskinens instruksjoner utføres av ALU-kretser, som bygger på transistor-modellen vi introduserte i forrige uke, under kontroll av Kontroll-Enheten