Datorer och datoranvändning
Maskinkod

Mattias Nordahl

mattias.nordahl@cs.lth.se

Föreläsning 4 — Maskinkod

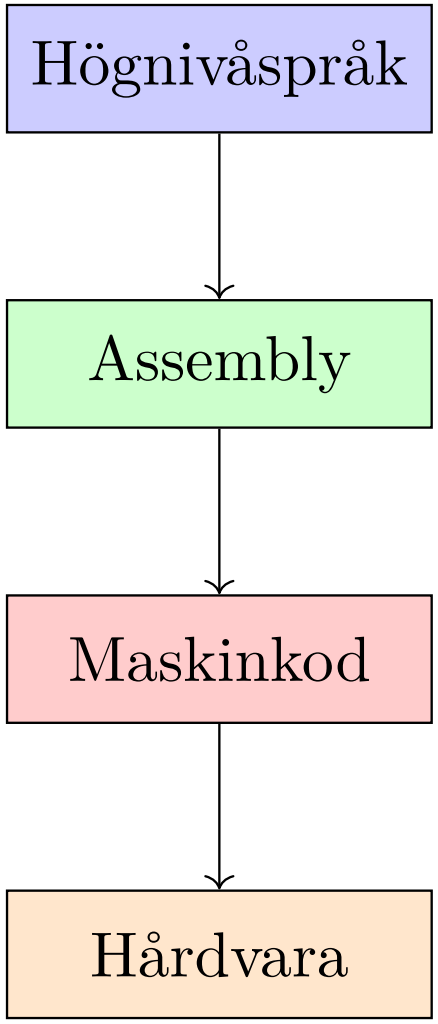
Förberedelse inför laboration 4

- Vad är en dator?
- Binära tal
- CPU och Minne
- Maskinkod och instruktioner
- Assembly-språk
- Kompilerare och tolkar
- *c3pu* – används på laborationen

Bakgrund

Varför lära sig Maskinkod?

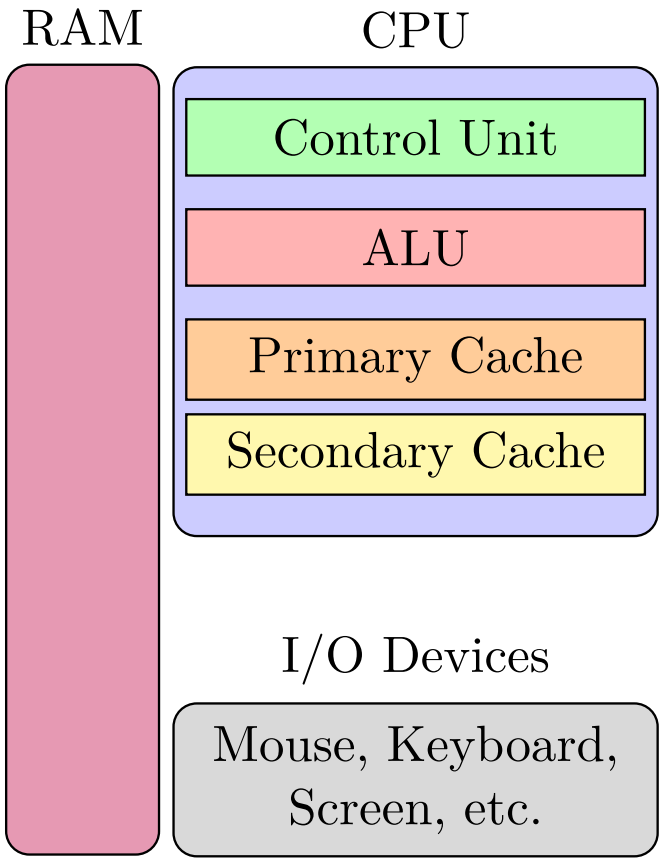
- Djupare förståelse – insikt i hur datorer fungerar på en grundläggande nivå.
  - *Vad händer i datorn med kod som vi skrivit?*
- Effektiv problemlösning och innovativt tänkande.
  - *Hur kan vi skriva snabb och effektiv kod?*
  - *Hur kan vi hitta på innovativa lösningar?*



Vad är en dator?

Komponenter

- **CPU:** ALU, kontrollenhet, cache
- **Minne:** RAM och lagring (hårddisk)
- **I/O-enheter:** In- och utmatning (mus, tangentbord, skärm)
- **Moderkort:** Kopplar ihop alla komponenter



Programvara vs. Hårdvara

- **Hårdvara:** Fysiska delarna av datorn
- **Mjukvara (programvara):** Instruktioner för att utföra uppgifter

Binära tal - Introduktion

Olika talsystem

- Talsystem används för att representera tal med symboler.
- Decimal (bas 10) – vanligast, använder siffror 0-9.
- Siffrors position bestämmer 10-potensvärdet. Exempel:

$$274 = 2 \cdot 100 + 7 \cdot 10 + 4 \cdot 1$$
$$274 = 2 \cdot 10^2 + 7 \cdot 10^1 + 4 \cdot 10^0$$

Binärt talsystem

- Binärt (bas 2) – används av datorer, bara två siffror, 0 och 1.
- Varje siffra kallas en *bit*.
- Siffrors position bestämmer 2-potensvärdet. Exempel:

$$1011_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$
$$1011_2 = 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = 11$$

Binära tal - Varför?

Effekten av olika baser

- Basen bestämmer antalet symboler (siffror) som används för att representera tal.
- Större bas kräver färre symboler för att representera samma tal.
- Exempel:
  - Talet 999_{10} skrivs med 3 siffror
  - Motsvarande i binärt: 1111100111_2, 10 siffror.

Varför binära tal i datorer?

- Enkelt att representera elektroniskt (ström finns/ström saknas).
- Pålitligt att tolka signaler som hög (1) eller låg (0) spänning.

Binära tal - I Datorer

Bits, Bytes och Ord

- **Bit:** Grundläggande enhet av data i datorer (0 eller 1)
- **Byte:** Grupp av 8 bitar
- **Ord:** Processor-specifik grupp av bitar (t.ex., 32-bit eller 64-bit)

Användning av binära tal

- Lagring och bearbetning av all data och instruktioner
- Varje byte kan t.ex. representera ett litet heltal (0–255) eller ett tecken (ASCII-kod)
- Större grupper (ord) hanterar mer komplexa data som heltal och flyttal

Hexadecimala Tal

Hexadecimala talsystemet

- Hexadecimal (bas 16) - använder siffror 0-9 och bokstäverna A-F.

Varför Hexadecimala tal?

- Förenkling!
  - Varje tecken representerar fyra binära siffror (bitar).
  - Lättare att läsa och skriva stora binära tal.
- Används ofta för att representera *minnesadresser*

Exempel

- Binärt: `1011 1010` = Hex: `BA`
- Större binärtal: `1011 1010 0101 1110` = Hex: `BA5E`

Central Processing Unit (CPU)

Struktur och funktion

- ALU (Aritmetisk-logisk enhet) – utför beräkningar och logiska operationer
- Register – lagrar tillfällig data under exekvering
- Kontrollenhet – styr och koordinerar CPU:ns aktiviteter
- Cache – snabbt minne för att optimera åtkomst till data

CPU:n gör:

- Läser instruktioner från minnet
- Avkodar (tolkar) dem
- Utför instruktionerna (exekverar)

Minne i datorer

Typer av minne

- Hårddisk, SSD, etc. – permanent lagring av data
- RAM – arbetsminne som lagrar data och instruktioner temporärt
- Cache – snabbare minne nära CPU:n för att optimera åtkomst
- ROM – lagrar permanent data, t.ex. firmware

Var används vilket minne?

- Moderkortet har litet ROM, för att starta datorn
- CPU:n kan komma åt RAM via moderkortet
- Cache används för att minska åtkomsttiden till ofta använd data
  (Finns ofta flera nivåer av cache)

Maskinkod – definition och egenskaper

Vad är maskinkod?

- Binära instruktioner direkt förstådda av CPU:n
- Utför enkla operationer som aritmetik, logik, och flyttning av data
- CPU:n använder en specifik instruktionsuppsättning (ISA – *Instruction Set Architecture*)

Exempel på maskininstruktioner

- `MOV`, `ADD`, `SUB`, `JMP`

Maskininstruktioner

Komponenter av en instruktion

- Opcode – specificerar operationen (t.ex. `ADD`)
- Operand(er) – data eller adresser för operationen (t.ex. register)
- Exempel: `ADD R1, R2, R3`

Exekvering av instruktioner (fetch-decode-execute)

- CPU:n hämtar instruktionen från minnet
- Avkodar opcode och identifierar operand(er)
- Utför operationen

Assembly-språk

Vad är Assembly-språk?

- Lågnivåspråk nära maskinkod, använder textrepresentation av instruktioner
- Varje assembly-instruktion motsvaras direkt av en maskininstruktion
- CPU-specifikt, beroende på arkitektur

Översättning till maskinkod

- Assembler konverterar assembly till maskinkod som CPU:n kan exekvera

Exempel på en assembly-instruktion

Instruktion: `ADD R1, R2, R2`

- Adderar värdet i `R1` och `R2`, lagrar resultatet i `R2`
- R1 och R2 är register som används för temporär datalagring i CPU:n

Relation till maskinkod

- Opcode för `ADD` + binär representation av register (`R1`, `R2`)
- Exempel: `ADD R1, R2, R2` → `1001 0001 0010 0010`

Instruktionsuppsättningsarkitektur (ISA)

Vad är ISA?

- ISA specificerar alla maskininstruktioner som en CPU kan förstå och utföra
- Olika CPU-arkitekturer har olika uppsättningar
- Påverkar hur assembly-språk och maskinkod ser ut för olika processorer

Exempel på ISA:er

- x86 – Används i de flesta persondatorer
- ARM – Vanlig i mobila enheter
- RISC-V – Öppen arkitektur, växande i popularitet

Maskinkod vs. Högnivåspråk

Utmaningar med maskinkod

- Maskinkod är lågnivå och svår att läsa och skriva för människor
- Svårt att hantera tillstånd och strukturer, och att debugga
- Inga if/else och loopar – ersätts med jämförelser och hopp
- Funktioner är manuellt hanterade med stack- och registeroperationer
- Ingen objektorientering (klasser, arv, etc.)

Varför högnivåspråk?

- Abstraktioner för vanliga strukturer, vilket gör kod enklare att skriva, läsa och underhålla
- Kompilatorer hanterar översättning av komplexa strukturer till maskinkod

Laboration – c3pu

Så, vad ska ni göra på laborationen?

- Översätta kod till assembly?
- Nej, värre än så ...
- Skriva egen assembly-kod?
- Nej, ännu värre än så!
- **Skriva maskinkod för hand!**
  `01010111 01010100 01000110 00100001 00111111`

DEMO

c3pu – en enkel CPU-simulator