

CPUer og maskinkode

DM573

Rolf Fagerberg

Mål

Målet for disse slides er at beskrive, hvordan maskinkode ser ud og hvordan CPUen udfører et program i maskinkode.

CPUs opbygning

En CPU er bygget op af elektriske kredsløb (jvf. sidste forelæsning), som kan manipulere bits.

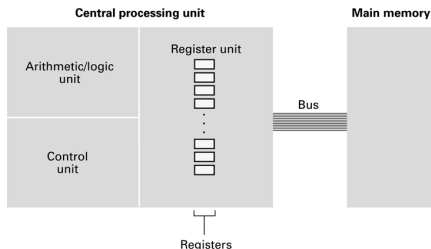
En CPU manipulerer flere bits ad gangen. Deres antal kaldes CPUens *ordlængde* (typisk 32 eller 64), og en sådan gruppe bits kaldes et *ord*.

CPUers opbygning

En CPU er bygget op af elektriske kredsløb (jvf. sidste forelæsning), som kan manipulere bits.

En CPU manipulerer flere bits ad gangen. Deres antal kaldes CPUens *ordlængde* (typisk 32 eller 64), og en sådan gruppe bits kaldes et *ord*.

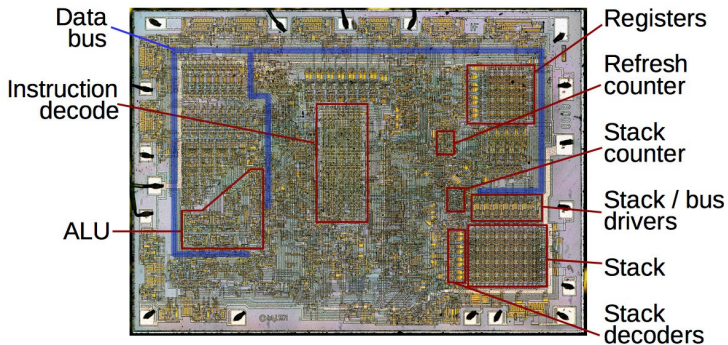
Typisk er en CPU struktureret således:



Registre er et lille antal hukommelsesceller på CPU'en. Main memory (RAM) er en masse hukommelsesceller uden for CPU'en. Hver hukommelsescelle kan gemme et ord.

CPUs opbygning

Et virkeligt eksempel:



Intel 8008, 1972 (3.500 transistorer)

Instruktionssæt

En CPU tilbyder en række ret simple kommandoer til at manipulere ord. Disse kaldes CPUens *instruktionssæt*.

Der er normalt kommandoer af følgende typer:

- ▶ **Moves:** Flytter ord mellem RAM (main memory) og CPUens registre.
- ▶ **Calculations:** Lave simple beregninger på ord i CPUens registre (plus, minus, gange, dividere, AND, OR, . . .).
- ▶ **Jumps:** Ændre hvorhenne i programmet den næste kommando læses fra, ofte baseret på sammenligning af indhold i registre.

Program

Et eksekverbart program er en række af sådanne kommandoer.

Programmet ligger i RAM. Hver kommando er, som alt andet i en computer, repræsenteret ved en række bits.

En CPUs maskinsprog angiver derfor (endnu) et fortolkningssystem for bitmønstre.

Dette fortolkningssystem er en del af designvalget, når CPUer designes (men det meste af designprocessen går naturligvis med at få lavet elektronik, som kan *udføre* kommandoerne i maskinsproget).

Maskinkode vs. programmeringsprog

Man programmerer sjældent i maskinsprog. Det er svært at bevare overblikket og nemt at lave fejl.

Derfor har man opfundet *programmeringsprog* såsom Python, Java, C++ og C#.

Programmer skrevet i et programmeringsprog oversættes til et eksekverbart program i maskinsprog før [eller mens] det kører. Dette sker af et fast program kaldet en *compiler* [eller *fortolker*]. Det er derfor, at programmeringsprog er nødt til at have så stiv en sprogstruktur (syntaks).

Maskinkode vs. programmeringsprog

Man programmerer sjældent i maskinsprog. Det er svært at bevare overblikket og nemt at lave fejl.

Derfor har man opfundet *programmeringsprog* såsom Python, Java, C++ og C#.

Programmer skrevet i et programmeringsprog oversættes til et eksekverbart program i maskinsprog før [eller mens] det kører. Dette sker af et fast program kaldet en *compiler* [eller *fortolker*]. Det er derfor, at programmeringsprog er nødt til at have så stiv en sprogstruktur (syntaks).

Vi skal i dag prøve at programmere *direkte* i maskinsprog.

CPU cyklus

En CPU arbejder ved at gentage følgende cyklus:

- ▶ **Fetch:** Hent næste kommando (en samling bits) fra programmet i RAM til CPU.
- ▶ **Decode:** Konverter kommandoen (en samling bits) til kontrolsignaler internt i CPUen.
- ▶ **Execute:** Disse kontrolsignaler aktiverer de relevante dele af CPUen som udfører kommandoen.

Der er ca. 10^9 cykler i sekundet i en typisk CPU (Ghz clock-frekvens).

CPU cyklus

En CPU arbejder ved at gentage følgende cyklus:

- ▶ **Fetch:** Hent næste kommando (en samling bits) fra programmet i RAM til CPU.
- ▶ **Decode:** Konverter kommandoen (en samling bits) til kontrolsignaler internt i CPUen.
- ▶ **Execute:** Disse kontrolsignaler aktiverer de relevante dele af CPUen som udfører kommandoen.

Der er ca. 10^9 cykler i sekundet i en typisk CPU (Ghz clock-frekvens).

Adressen i RAM på den næste kommando som skal hentes, angives af et specielt register kaldet *program counter*. Normalt tælles program counter én op efter hver cyklus (*sekventiel programudførelse*).

CPU cyklus

En CPU arbejder ved at gentage følgende cyklus:

- ▶ **Fetch:** Hent næste kommando (en samling bits) fra programmet i RAM til CPU.
- ▶ **Decode:** Konverter kommandoen (en samling bits) til kontrolsignaler internt i CPUen.
- ▶ **Execute:** Disse kontrolsignaler aktiverer de relevante dele af CPUen som udfører kommandoen.

Der er ca. 10^9 cykler i sekundet i en typisk CPU (Ghz clock-frekvens).

Adressen i RAM på den næste kommando som skal hentes, angives af et specielt register kaldet *program counter*. Normalt tælles program counter én op efter hver cyklus (*sekventiel programudførelse*).

En jump kommando ændrer på indholdet af dette register, og man kan dermed styre, hvilke kommandoer fra programmet der udføres som det næste. Sådan laves *metodekald*, *forgreninger* (if/then/else) og *løkker*.

Kommandoer

Et programs kommandoer er, som alt andet i en computer, repræsenteret ved en række bits. Disse er ofte struktureret således:

| | | | |
|----------|----------|----------|----------|
| 01101011 | 01111101 | 00001101 | 11101001 |
| opcode | operand | operand | operand |

Opcode angiver kommandotypen (flyt ord, plus, gange, jump, ...).

Operand angiver input til kommandoen. Det kan være f.eks. et registernummer, en hukommelsesadresse, eller et stykke data, alt efter typen af kommando.

Kommandoer

Et programs kommandoer er, som alt andet i en computer, repræsenteret ved en række bits. Disse er ofte struktureret således:

| | | | |
|----------|----------|----------|----------|
| 01101011 | 01111101 | 00001101 | 11101001 |
| opcode | operand | operand | operand |

Opcode angiver kommandotypen (flyt ord, plus, gange, jump, . . .).

Operand angiver input til kommandoen. Det kan være f.eks. et registernummer, en hukommelsesadresse, eller et stykke data, alt efter typen af kommando.

Tænk på det som funktioner:

opcode(operand, operand, operand)

F.eks.:

plus(fra register 1, fra register 2, til register 3)

flyt(fra RAM celle 14, til register 4)

En CPU simulator

The Brookshear Machine: et program som simulerer en simpel CPU udstyret med:

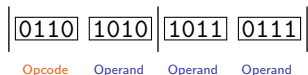
- ▶ 13 kommandotyper.
- ▶ 16 registre (plus en program counter).
- ▶ Ordlængde 8 bits.
- ▶ En RAM med 256 ord. Indeholder både program og data.

En CPU simulator

The Brookshear Machine: et program som simulerer en simpel CPU udstyret med:

- ▶ 13 kommandotyper.
- ▶ 16 registre (plus en program counter).
- ▶ Ordlængde 8 bits.
- ▶ En RAM med 256 ord. Indeholder både program og data.

En kommando fylder to ord (16 bits), som er opdelt således:

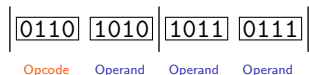


En CPU simulator

The Brooks Machine: et program som simulerer en simpel CPU udstyret med:

- ▶ 13 kommandotyper.
- ▶ 16 registre (plus en program counter).
- ▶ Ordlængde 8 bits.
- ▶ En RAM med 256 ord. Indeholder både program og data.

En kommando fylder to ord (16 bits), som er opdelt således:



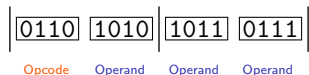
I simulatoren angives bits fire ad gangen via et hexadecimalt ciffer (0, 1, 2, 3, ..., 9, A [= 10], B [= 11], C [= 12], ..., F [= 15]).

En CPU simulator

The Brookshear Machine: et program som simulerer en simpel CPU udstyret med:

- ▶ 13 kommandotyper.
- ▶ 16 registre (plus en program counter).
- ▶ Ordlængde 8 bits.
- ▶ En RAM med 256 ord. Indeholder både program og data.

En kommando fylder to ord (16 bits), som er opdelt således:



I simulatoren angives bits fire ad gangen via et hexadecimalt ciffer (0, 1, 2, 3, ..., 9, A [= 10], B [= 11], C [= 12], ..., F [= 15]).

En kommando (16 bits) kan beskrives med fire hexadecimale cifre (fire bits hver). Heraf kan en kommando- og et registernummer hver angives med ét ciffer (16 muligheder), og en adresse i RAM kan angives med to cifre ($16 \cdot 16 = 256$ muligheder).

Instruktionssæt

Kommandoer: (hvor R , X og Y angiver ét hexadecimalt ciffer).

| | <i>Opcode</i> | <i>Operands</i> | <i>Effekt</i> |
|---|---------------|-----------------|--|
| 1 | RXY | | Kopier indholdet af RAM celle XY til register R . |
| 2 | RXY | | Læg bitmønstret angivet af XY ind i register R . |
| 3 | RXY | | Kopier indholdet af register R til RAM celle XY . |
| 4 | xRS | | Kopier indholdet af register R til register S . |
| 5 | RST | | Lav addition af indholdet af register S og T og læg resultatet i register R . Indhold fortolkes som heltal i two's complement. |
| 6 | RST | | Lav addition af indholdet af register S og T og læg resultatet i register R . Indhold fortolkes som flydende kommatall. |

Instruktionssæt

Kommandoer: (hvor R , X og Y angiver ét hexadecimalt ciffer).

| | <i>Opcode</i> | <i>Operands</i> | <i>Effekt</i> |
|---|---------------|-----------------|--|
| 1 | RXY | | Kopier indholdet af RAM celle XY til register R . |
| 2 | RXY | | Læg bitmønstret angivet af XY ind i register R . |
| 3 | RXY | | Kopier indholdet af register R til RAM celle XY . |
| 4 | xRS | | Kopier indholdet af register R til register S . |
| 5 | RST | | Lav addition af indholdet af register S og T og læg resultatet i register R . Indhold fortolkes som heltal i two's complement. |
| 6 | RST | | Lav addition af indholdet af register S og T og læg resultatet i register R . Indhold fortolkes som flydende kommatall. |

Eksempel: $3A2F$ = kopier indholdet af register A_{16} [10] til RAM celle $2F_{16}$ [47].

Instruktionssæt

Kommandoer: (hvor R , X og Y angiver ét hexadecimalt ciffer).

| | <i>Opcode</i> | <i>Operands</i> | <i>Effekt</i> |
|---|---------------|-----------------|--|
| 1 | RXY | | Kopier indholdet af RAM celle XY til register R . |
| 2 | RXY | | Læg bitmønstret angivet af XY ind i register R . |
| 3 | RXY | | Kopier indholdet af register R til RAM celle XY . |
| 4 | xRS | | Kopier indholdet af register R til register S . |
| 5 | RST | | Lav addition af indholdet af register S og T og læg resultatet i register R . Indhold fortolkes som heltal i two's complement. |
| 6 | RST | | Lav addition af indholdet af register S og T og læg resultatet i register R . Indhold fortolkes som flydende kommatall. |

Eksempel: $3A2F$ = kopier indholdet af register A_{16} [10] til RAM celle $2F_{16}$ [47].

Eksempel: $58BC$ = ?

Instruktionssæt

Kommandoer: (hvor R , X og Y angiver ét hexadecimalt ciffer).

| | <i>Opcode</i> | <i>Operands</i> | <i>Effekt</i> |
|---|---------------|-----------------|--|
| 1 | RXY | | Kopier indholdet af RAM celle XY til register R . |
| 2 | RXY | | Læg bitmønstret angivet af XY ind i register R . |
| 3 | RXY | | Kopier indholdet af register R til RAM celle XY . |
| 4 | xRS | | Kopier indholdet af register R til register S . |
| 5 | RST | | Lav addition af indholdet af register S og T og læg resultatet i register R . Indhold fortolkes som heltal i two's complement. |
| 6 | RST | | Lav addition af indholdet af register S og T og læg resultatet i register R . Indhold fortolkes som flydende kommatall. |

Eksempel: $3A2F$ = kopier indholdet af register A_{16} [10] til RAM celle $2F_{16}$ [47].

Eksempel: $58BC$ = lav addition af indholdet (set som two's complement heltal) af register B_{16} [11] og C_{16} [12] og læg resultatet i register 8_{16} [8].

Instruktionssæt, fortsat

| <i>Opcode</i> | <i>Operands</i> | <i>Effekt</i> |
|---------------|-----------------|--|
| 7 | <i>RST</i> | Lav bit-wise OR af indholdet af register <i>S</i> og <i>T</i> og læg resultatet i register <i>R</i> . |
| 8 | <i>RST</i> | Lav bit-wise AND af indholdet af register <i>S</i> og <i>T</i> og læg resultatet i register <i>R</i> . |
| 9 | <i>RST</i> | Lav bit-wise XOR af indholdet af register <i>S</i> og <i>T</i> og læg resultatet i register <i>R</i> . |
| A | <i>RxX</i> | Rotér indholdet af register <i>R</i> cyklisk mod højre <i>X</i> skridt. |
| B | <i>RXY</i> | Jump til instruktionen i RAM celle <i>XY</i> hvis indholdet i register <i>R</i> er lig (=) indholdet i register 0. |
| C | <i>xxx</i> | Stop programmet. |
| D | <i>RXY</i> | Jump til instruktionen i RAM celle <i>XY</i> hvis indholdet i register <i>R</i> er større (>) end indholdet i register 0. Indhold fortolkes som heltal i two's complement. |

Uddybning

Uddybning af et par af operationerne ovenfor:

- ▶ *Bit-wise OR* (eller AND eller XOR): Brug OR (eller AND eller XOR) på hver plads. Et eksempel med OR:

$$\begin{array}{r} 10011001 \\ 01010001 \\ \hline = 11011001 \end{array}$$

På hver plads er den nye bit 1 hvis mindst én af de to gamle er 1 (jvf. definitionen af OR).

- ▶ *Cyklisk rotation mod højre*: Alle bits i ordet flyttes mod højre. Dem, som skubbes ud over højre ende af ordet, sættes ind igen i venstre ende i samme rækkefølge. Et eksempel på at rotere 3 skridt cyklisk mod højre:

$$01011001 \rightarrow 00101011$$

Eksempelprogrammer

Følgende program bytter om på indholdet af RAM celle 10_{16} [16] og 12_{16} [18]:

1110

1212

3112

3210

C000

Eksempelprogrammer

Følgende program bytter om på indholdet af RAM celle 10_{16} [16] og 12_{16} [18]:

```
1110  
1212  
3112  
3210  
C000
```

Her er programmet igen, med hver linie forklaret:

```
1110   Kopier indholdet af RAM celle 10 til register 1  
1212   Kopier indholdet af RAM celle 12 til register 2  
3112   Kopier indholdet af register 1 til RAM celle 12  
3210   Kopier indholdet af register 2 til RAM celle 10  
C000   Stop
```

(Hvis man skal se effekten når programmet kører, skal man først fylde RAM celler 10 og 12 med forskelligt indhold.)

Eksempelprogrammer

Følgende program illustrerer en løkke. Det skriver efter tur tallene 0, 1, 2, 3, ..., 6 (dvs. indhold 00, 01, 02, 03, ..., 06) i RAM celle 1C₁₆ [28], hvis RAM celle 1A₁₆ [26] indeholder 07 til at starte med.

2000

2101

121A

301C

5001

D206

C000

Eksempelprogrammer

Her er programmet igen, med hver linie forklaret:

```
2000  Læg bitmønstret angivet (hexadecimalt) af 00 ind i register 0
2101  Læg bitmønstret angivet (hexadecimalt) af 01 ind i register 1
121A  Kopier indholdet af RAM celle 1A til register 2
301C  Kopier indholdet af register 0 til RAM celle 1C
5001  Lav addition af register 0 og 1, læg resultat i register 0
D206  Jump til instruktionen i RAM celle 06 (og 07) hvis indholdet i
      register 2 er større end indholdet i register 0. Når programmet
      placeres i starten af hukommelsen er denne instruktion 301C.
C000  Stop
```

(Husk først at fylde RAM celle 1A med indhold (f.eks. 07) inden programmet kører.)

Eksempelprogrammer

Her er programmet igen, med hver linie forklaret:

```
2000  Læg bitmønstret angivet (hexadecimalt) af 00 ind i register 0
2101  Læg bitmønstret angivet (hexadecimalt) af 01 ind i register 1
121A  Kopier indholdet af RAM celle 1A til register 2
301C  Kopier indholdet af register 0 til RAM celle 1C
5001  Lav addition af register 0 og 1, læg resultat i register 0
D206  Jump til instruktionen i RAM celle 06 (og 07) hvis indholdet i
      register 2 er større end indholdet i register 0. Når programmet
      placeres i starten af hukommelsen er denne instruktion 301C.
C000  Stop
```

(Husk først at fylde RAM celle 1A med indhold (f.eks. 07) inden programmet kører.)

Opgave: Hvad skal ændres hvis man skal tælle *ned* fra 06 til 00 i stedet?