CPUer og maskinkode

DM573

Rolf Fagerberg

Mål

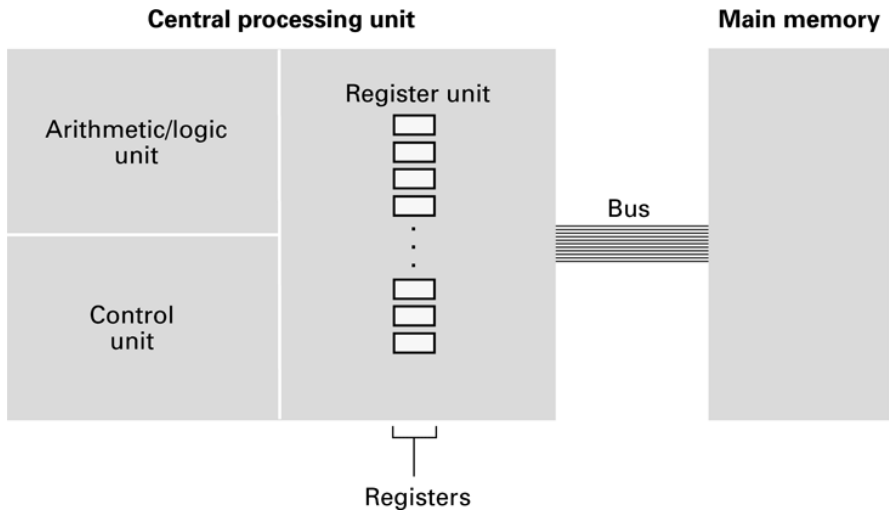
Målet for disse slides er at beskrive, hvordan maskinkode ser ud og hvordan CPUen udfører et program i maskinkode.

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 *ordlænge* (typisk 32 eller 64), og en sådan gruppe bits kaldest et *ord*.

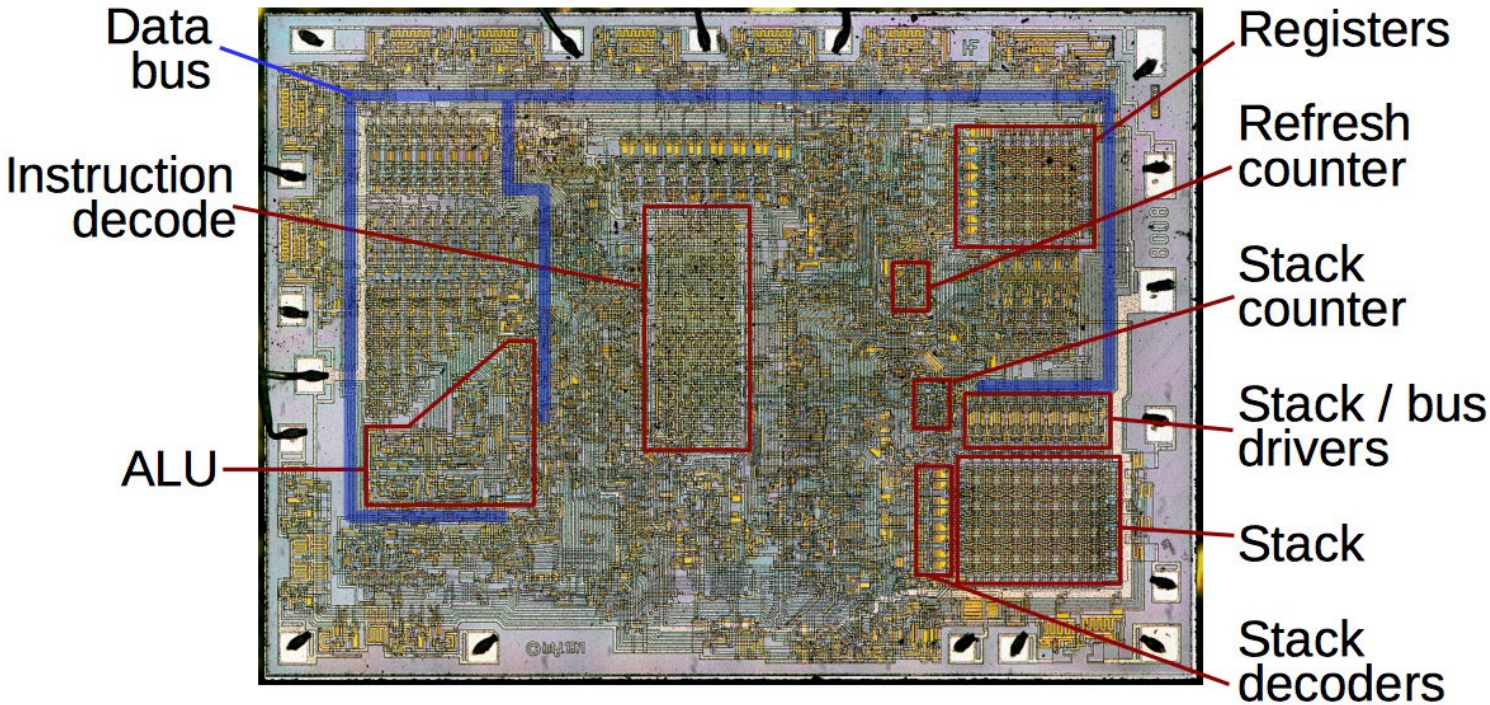
Typisk er en CPU struktureret således:



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.

CPUers 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, f.eks. 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. programmeringssprog

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

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

Programmer skrevet i et programmeringssprog 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 programmeringssprog 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).

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 programudførsel*).

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 *funktionskald*, *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.

Tænk på det som funktioner:

opcode(operand, operand, operand)

F.eks.:

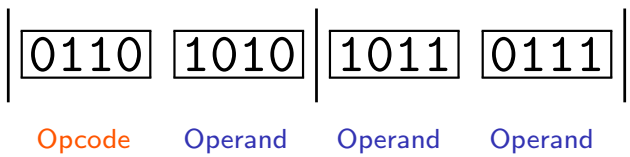
plus(fra register 1, fra register 2, til register 3)
flyt(fra RAM celle 14, til register 4)

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 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 hver angiver ét hexadecimalt ciffer).

Opcode	*Operands*	*Effekt*
1	$r\ x\ y$	Kopier indholdet af RAM celle xy til register r.
2	$r\ x\ y$	Læg bitmønstret angivet af xy ind i register r.
3	$r\ x\ y$	Kopier indholdet af register r til RAM celle xy.
4	- $r\ s$	Kopier indholdet af register r til register s.
5	$r\ s\ t$	Lav addition af indholdet af registre s og t og læg resultatet i register r. Indhold fortolkes som heltal i two's complement.
6	$r\ s\ t$	Lav addition af indholdet af registre s og t og læg resultatet i register r. Indhold fortolkes som flydende kommatal.

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

Opcode	*Operands*	*Effekt*
7	*r s t*	Lav bit-wise OR af indholdet af register *s* og *t* og læg resultatet i register *r*.
8	*r s t*	Lav bit-wise AND af indholdet af register *s* og *t* og læg resultatet i register *r*.
9	*r s t*	Lav bit-wise XOR af indholdet af register *s* og *t* og læg resultatet i register *r*.
A	*r* - *x*	Rotér indholdet af register *r* cyclisk mod højre *x* skridt.
B	*r x y*	Jump til instruktionen i RAM celle *xy* hvis indholdet i register *r* er lig (=) indholdet i register 0.
C	- - -	Stop programmet.
D	*r x y*	Jump til instruktionen i RAM celle *xy* hvis indholdet i register *r* 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).

- *Cyclisk 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:

$$01011001 \rightarrow 00101011$$

Eksempelprogram 1

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.)

Eksempelprogram 2

Følgende program illustrerer en løkke. Det skriver efter tur tallene $0, 1, 2, 3, \ldots, 6$ (dvs. indhold $00, 01, 02, 03, \ldots, 06$) i RAM celle $1C_{16}$ [28], hvis RAM celle $1A_{16}$ [26] indeholder 07 til at starte med.

```
2000
2101
121A
301C
5001
D206
C000
```

Eksempelprogram 2 med hver linie forklaret

2000	Læg bitmønstret angivet (hexadecimalt) af 00 ind i register 0
	Register 0 er en tæller, til start tallet 0 (i two's complement)
2101	Læg bitmønstret angivet (hexadecimalt) af 01 ind i register 1
	Register 1 er tallet 1 (i two's complement)
121A	Kopier indholdet af RAM celle 1A til register 2
	Indlæs den ønskede max-grænse fra RAM celle 1A
301C	Kopier indholdet af register 0 til RAM celle 1C
	Udskriv tæller til RAM celle 1C
5001	Lav addition af register 0 og 1, læg resultat i register 0
	Opdater tæller (øg den med 1)
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.
	Gentag de to sidste kommandoer, HVIS tæller ikke har nået max-grænsen (ELLERS gå videre til næste kommando).
C000	Stop

(Husk først at fylde RAM celle 1A med en max-grænse (f.eks. 07).)

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