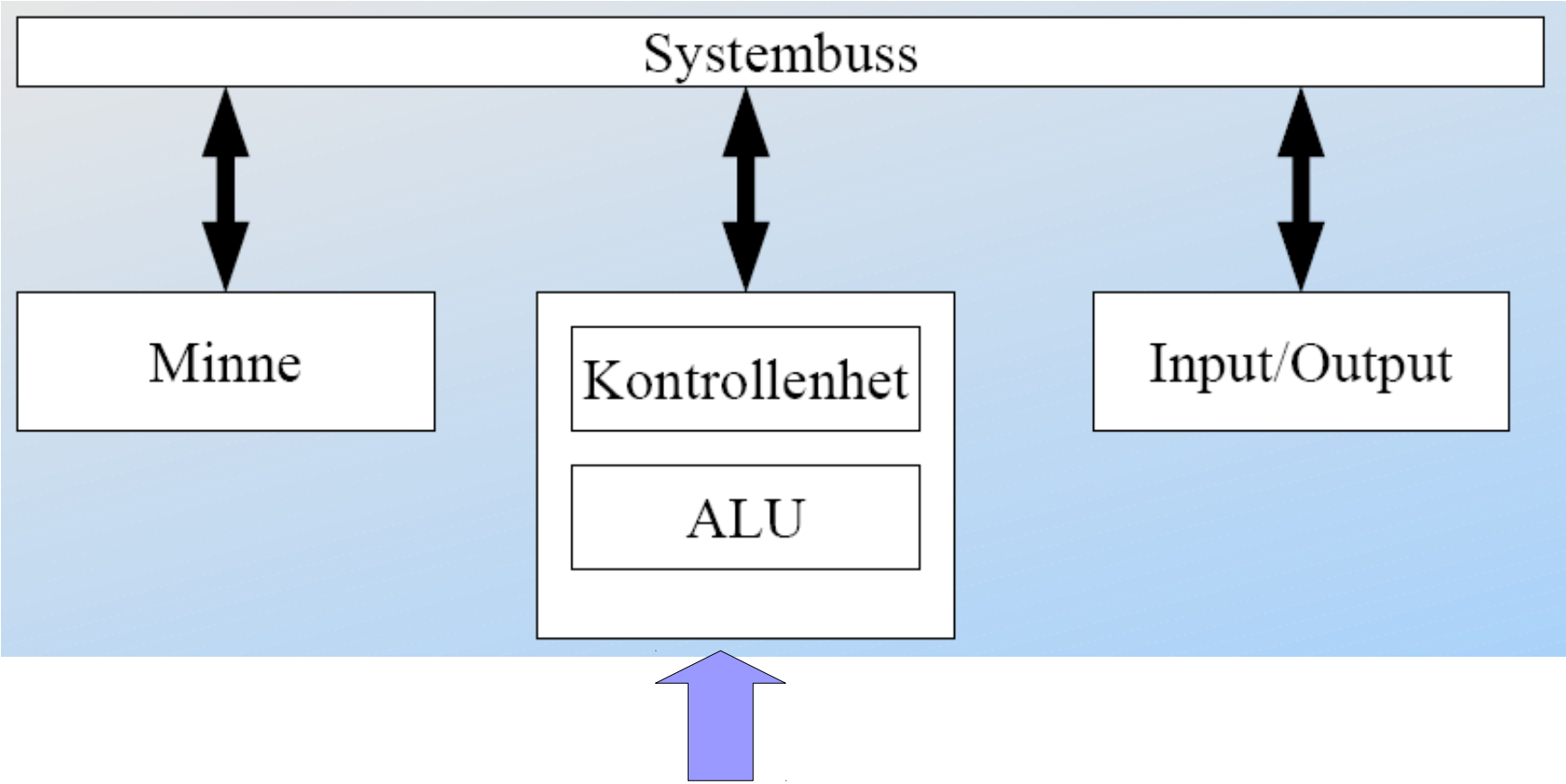
Grundläggande definitioner

- *Dator*: En programmerbar maskin som tar emot indata, utför operationer på datan och presenterar resultatet (=utdata) för användaren.
- *Algoritm*: En allmän procedur eller formel för att lösa ett problem: Sortera en lista, lös ett matematiskt problem...
- *Program*: En serie instruktioner som berättar för en dator vad som ska göras med indatan. Nära relaterat till algoritmbegreppet, kan ses som en specifik *implementation* av en algoritm

Hårdvaruarkitektur

- En typisk dator består av
  - centralenheter (processor, CPU)
    - innehåller (mycket förenklat) en kontrollenhet samt en aritmetisk-logisk enhet (ALU)
  - minne
  - enheter för in- och utmatning av data (input/output)
  - delsystemen förenas genom *systembussar*

Von Neumann-arkitekturen



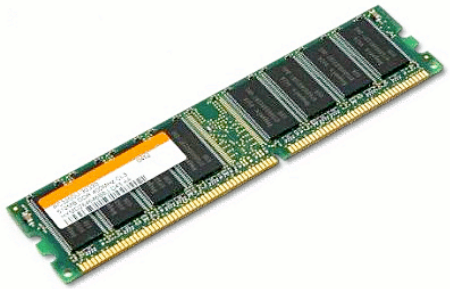
(Mycket förenklad) processor

Minnet

- Innehåller programinstruktioner och lagrad data
- Olika typer av minnen

Massminnen
(ex. hårdskiva)

RAM (Centralminne)



Cacheminne

Långsammare/billigare

Snabbare/dyrare

Programkörning

- Ett program finns lagrat i ett massminne
  - Består av instruktioner samt lagrad data
- Före körningen måste programinstruktionerna kopieras till centralminnet
- Under körningen hämtas instruktionerna, en i taget, till processorn

Processorn

- Hämtar, dekodar och utför programinstruktioner
  - Flyttar data mellan olika typer av minnen samt från och till I/O enheterna
  - ALU:n utför aritmetiska och logiska operationer
- Ett program för en viss processor (eller processorfamilj) kan endast använda sådana instruktioner som just denna processor förstår - “Instruction set”
- Instruktioner för Intel x86-processorer: http://en.wikipedia.org/wiki/X86_instruction_listings

Maskinkod och portabilitet

- För processorn är ett program endast en lång följd av siffror (egentligen endast ettor och nollor, men programmet nedan är skrivet i “hexadecimalt” format för att spara utrymme)
- Detta program skriver ut en rad text på skärmen, och kan endast köras på den processor (x86) som det är avsett för
- En instruktion består av en operationskod samt parametrar / operander för denna operation

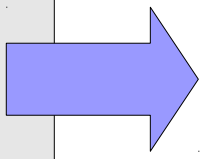
```
Data segment:
00000000  48 65 6c 6c 6f 20 77 6f
00000008  72 6c 64 21 0a

Code segment:
00000000  b8 04 00 00 00
00000005  bb 01 00 00 00
0000000a  b9 00 00 00 00
0000000f  ba 0d 00 00 00
00000014  cd 80
0000000a  b8 01 00 00 00
0000000f  bb 00 00 00 00
00000014  cd 80
```

Assembler

- Genom att använda överenskomna förkortningar för de olika operationskoderna samt “etiketter” för att namnge vissa dataområden kan maskinkoden göras mera lättläst
  => *Assemblerkod*
- Assemblerinstruktionerna måste omvandlas till maskinkod innan programmet kan köras

```
section .data
    string: db 'Hello world!',10
    length: equ $-string
section .text
    global _start
_start:
    mov eax,4
    mov ebx,1
    mov ecx,string
    mov edx,length
    int 80h
    mov eax,1
    mov ebx,0
    int 80h
```



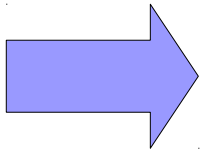
```
Data segment:
00000000  48 65 6c 6c 6f 20 77 6f
00000008  72 6c 64 21 0a

Code segment:
00000000  b8 04 00 00 00
00000005  bb 01 00 00 00
0000000a  b9 00 00 00 00
0000000f  ba 0d 00 00 00
00000014  cd 80
0000000a  b8 01 00 00 00
0000000f  bb 00 00 00 00
00000014  cd 80
```

Högnivåspråk och abstraktion

- Assemblerkod ger total kontroll över processorn, och (i teorin) den bästa prestandan, men att skriva komplexa program i assembler är mycket tidskrävande, och samma portabilitetsproblem som för den rena maskinkoden kvarstår
- Med ett *högnivåspråk* kan vi beskriva vad vårt program skall utföra på en högre *abstraktionsnivå*, utan att behöva ta ställning till hårdvarudetaljerna

```
string = 'Hello World!'
length = 13
output (string, length)
```



```
section .data
    string: db 'Hello world!',10
    length: equ $-string
section .text
    global _start
_start:
    mov eax,4
    mov ebx,1
    mov ecx,string
    mov edx,length
    int 80h
    mov eax,1
    mov ebx,0
    int 80h
```

Tolkade och kompilerade språk

- För en processor är ett program skrivet i ett högnivåspråk obegripligt
- En *tolk* översätter programmet till maskinkod *varje gång* det körs
  - Exempel på tolkade språk: Python, Perl
- En *kompilator* omvandlar programmet till maskinkod *en gång*. Resultatet av kompileringen är en binärfil som kan köras utan kompilatorns hjälp
  - Exempel på kompilerade språk: C, C++, Java
- Genom att använda tolkar och kompilatorer för olika processorer kan samma högnivåkod omvandlas till maskinkod för olika hårdvaruomgivningar