



LUNDS  
UNIVERSITET

# Datorteknik

---

ERIK LARSSON



# Personal

---

- Kursansvar
  - Erik Larsson: [Erik.Larsson@eit.lth.se](mailto:Erik.Larsson@eit.lth.se)
- Lärare:
  - Babak Mohammadi, Breeta SenGupta,  
Dimitar Nikolov, Farrokh Ghani Zadegan,  
Oskar Andersson, Anders Ardö,
- Kurssekreterare
  - Doris Glöck: [doris.glock@eit.lth.se](mailto:doris.glock@eit.lth.se)
- Forskningsingenjör
  - Bertil Lindvall: [bertil@eit.lth.se](mailto:bertil@eit.lth.se)



LUNDS  
UNIVERSITET

# Kursmoment

---

- Laborationer
- Tentamen - slutbetyg sätts på tentamenspoäng



LUNDS  
UNIVERSITET

# Laborationer

---

- Laboration 1: Maskininstruktioner
- Laboration 2: Aritmetiska funktioner
- Laboration 3: Assemblyprogrammering
- Laboration 4: Realtids schemaläggning



LUNDS  
UNIVERSITET

# Laborationer: Regler

---

- För att göra laborationer måste du följa följande regler:
  - Laborationsuppgifter ska lösas självständigt av varje laborationsgrupp
  - Det är inte tillåtet att ge laborationsresultat eller laborationsrapport till en annan grupp
  - Det är inte tillåtet att ta, kopiera eller på något annat sätt efterlikna en annan grups resultat eller rapport
  - Alla gruppmedlemmar måste ta aktiv del i alla delar av laborationen, det inkluderar att skriva programkod, testa och felsöka, genomföra experiment, skriva laborationsrapport och demonstrera (examination). Examinationen är alltid baserad på individuella resultat.



LUND  
UNIVERSITET

# Tentamen

---

- Skriftlig tentamen. Inga hjälpmmedel
- Första tentamen: 2015-03-18 klockan 14:00-19:00



LUNDS  
UNIVERSITET

# Litteratur

---

- Rekommenderad kursbok
  - » D. A. Patterson and J. L. Hennessy, Computer Organization and Design - the Hardware/Software Interface, Morgan Kaufmann.
- Alternativa kursböcker
  - » Brorsson, M: Datorsystem: program- och maskinvara. Studentlitteratur AB
  - » William Stallings: Computer Organization and Architecture, Prentice Hall International, Inc..
  - » Sven Eklund, Avancerad datorarkitektur, Studentlitteratur, 1994.
  - » Andrew S. Tanenbaum, Structured Computer Organization, 4th edition, Prentice Hall International, Inc., 1999.



LUNDS  
UNIVERSITET

# Upplägg

---

- Föreläsningar:
  - Syfte: Ge överblick. Underlätta läsning av kursbok.
  - Förberedelser: Inga. Kan skriva ut PDF för att underlätta anteckningar
- Lektioner:
  - Syfte: Introducerar laborationer
  - Förberedelser: Ha läst kursboken rörande laboration.
- Laborationer:
  - Syfte: Ge praktisk kunskap och förståelse
  - Förberedelser: Ha studerat laborationsmaterial



LUND  
UNIVERSITET



LUNDS  
UNIVERSITET

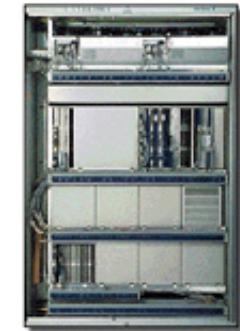
# Datorteknik

---

ERIK LARSSON

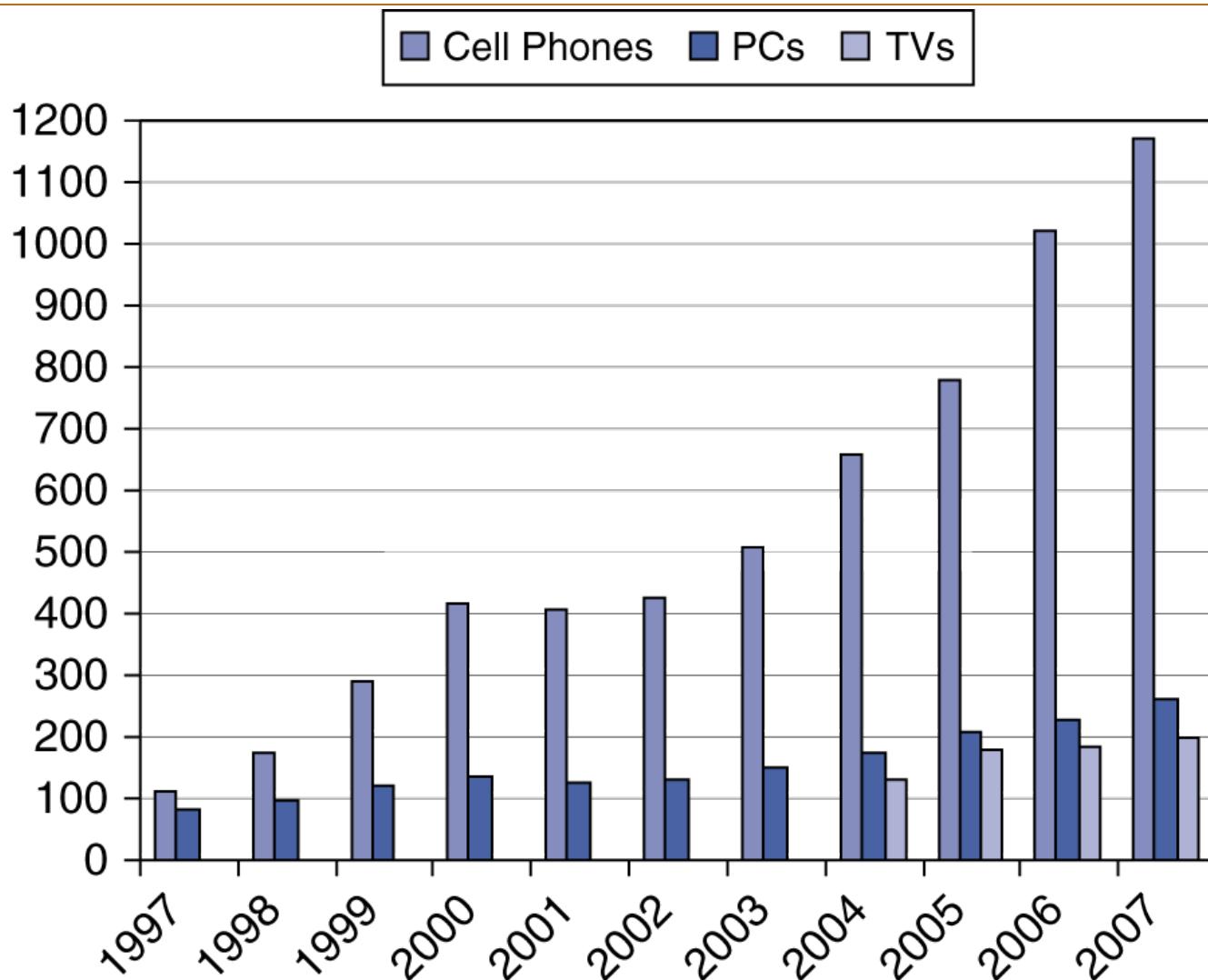


# Datorer



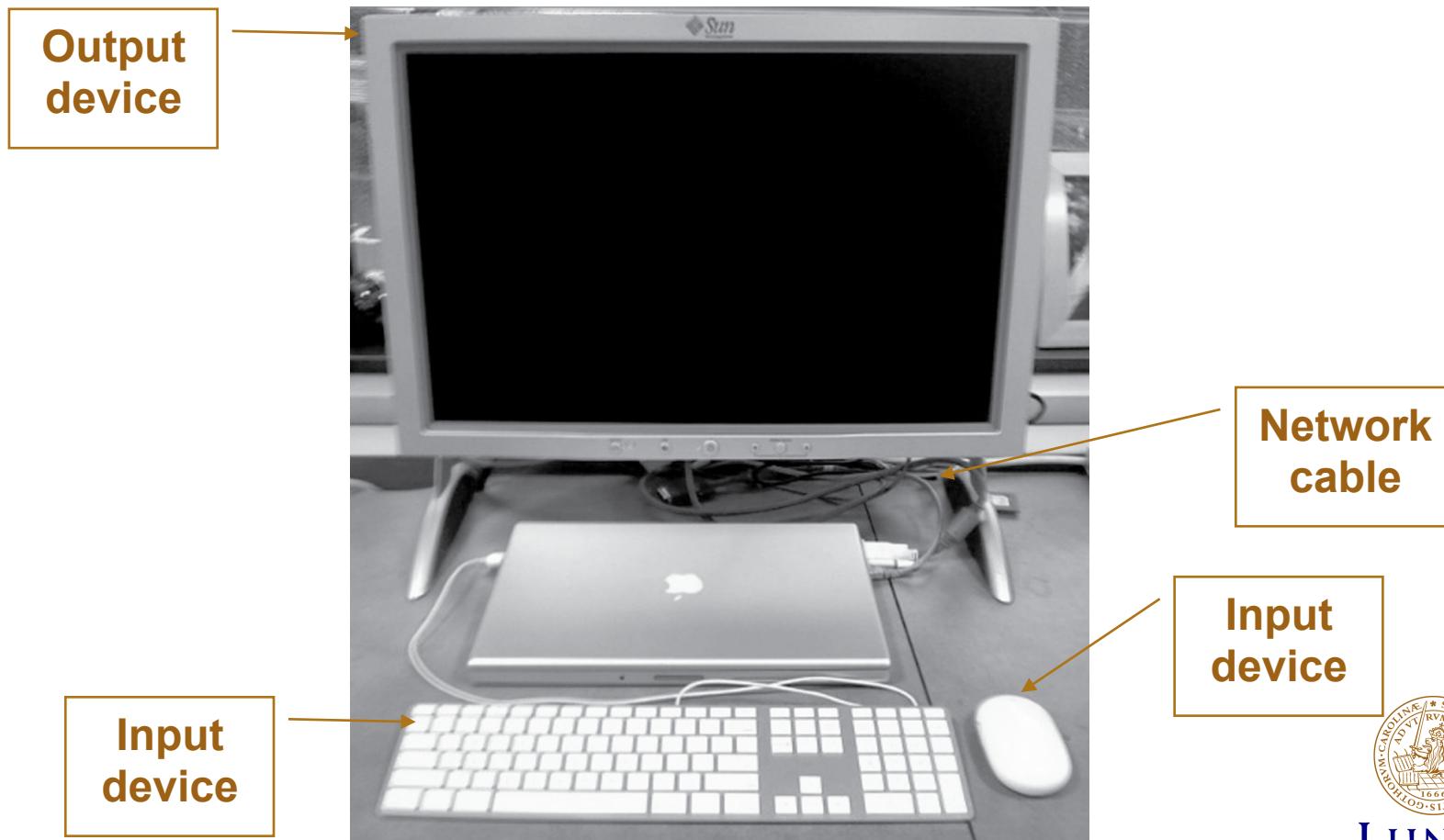
LUNDS  
UNIVERSITET

# Marknad för processorer



# Datorns delar

---

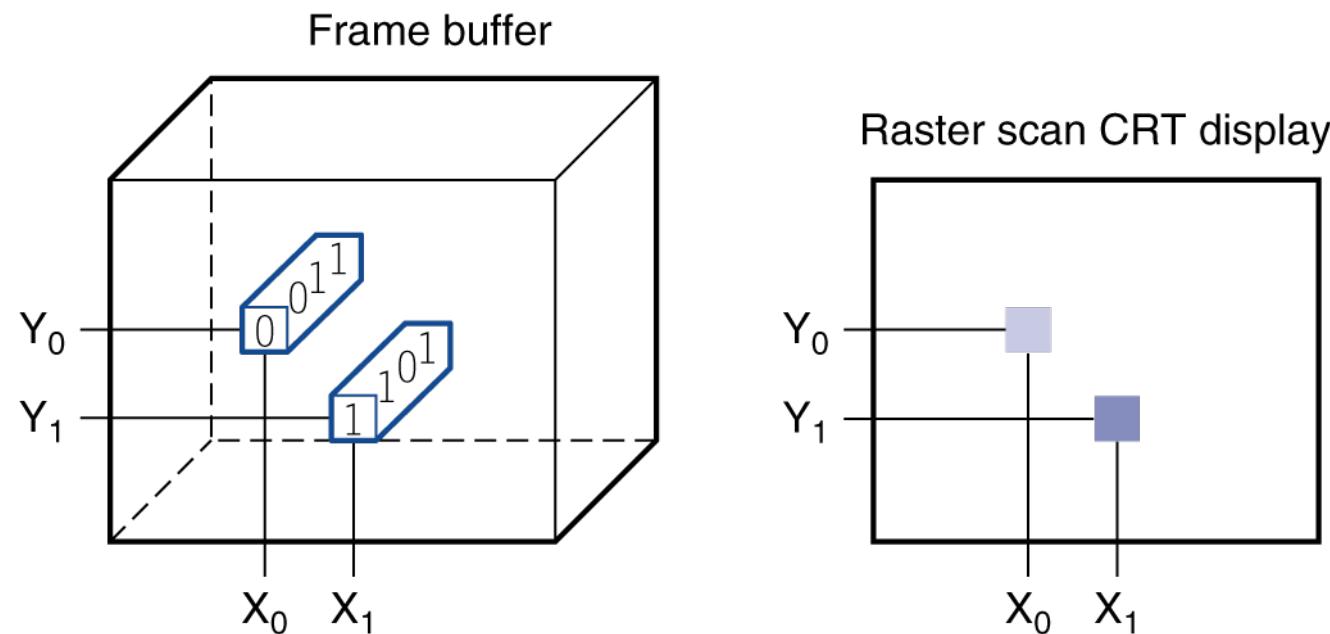


LUNDS  
UNIVERSITET

# Output device

---

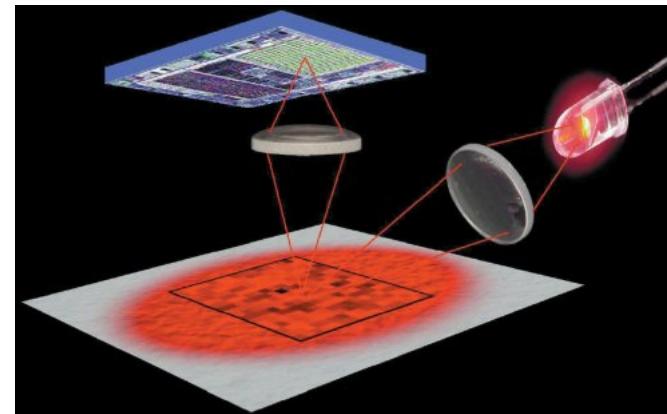
- LCD skärm: bildelement (pixels)
- Speglar innehållet i "frame buffer memory"



# Input device

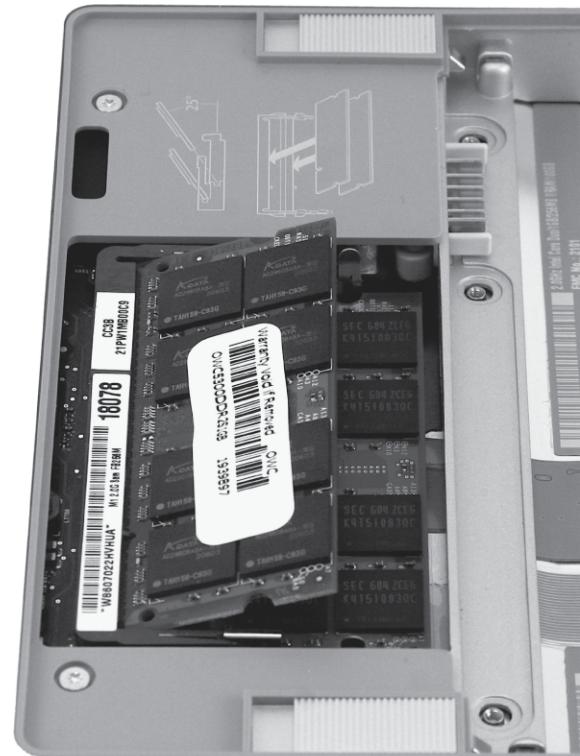
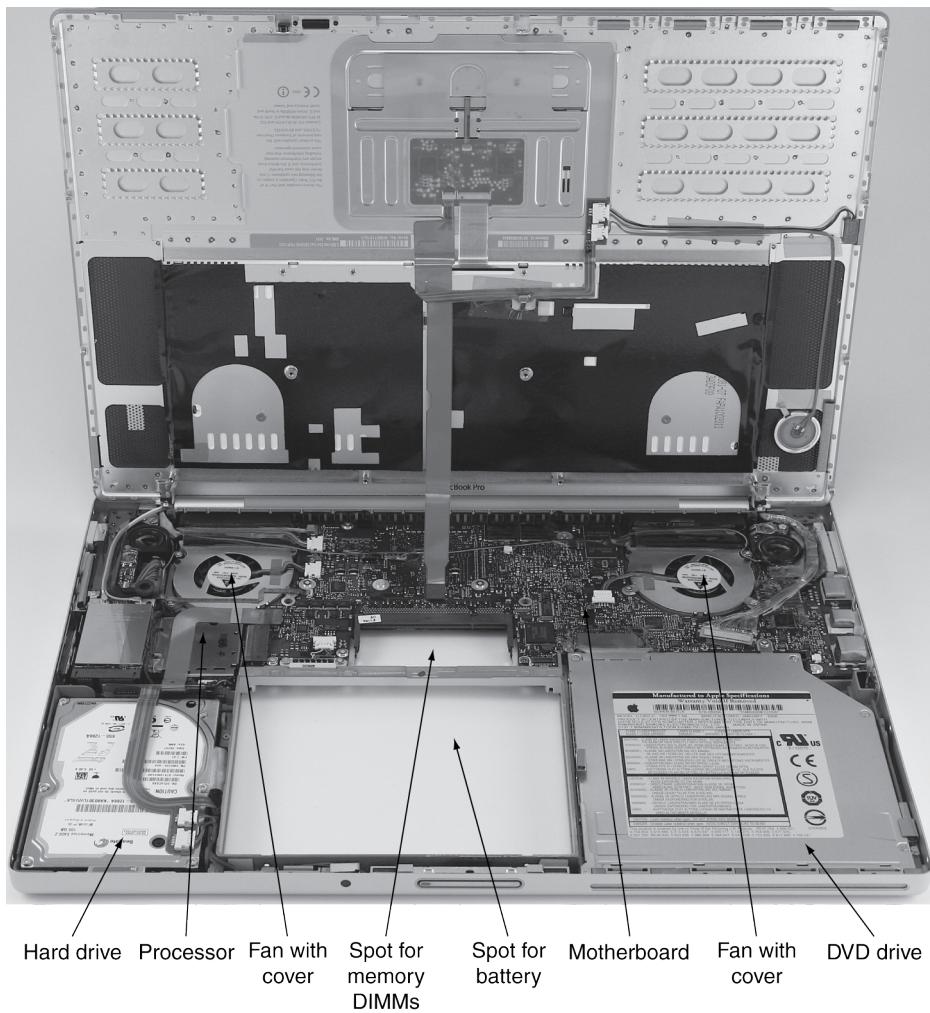
---

- Optisk mus
  - LED lyser upp underlag
  - Liten kamera
  - Enkel bildbehandlingsprocessor
    - » Söker x,y förflyttningar
  - Knappar och hjul



LUND  
UNIVERSITET

# Datorns insida



LUND  
UNIVERSITET

# Minne

---

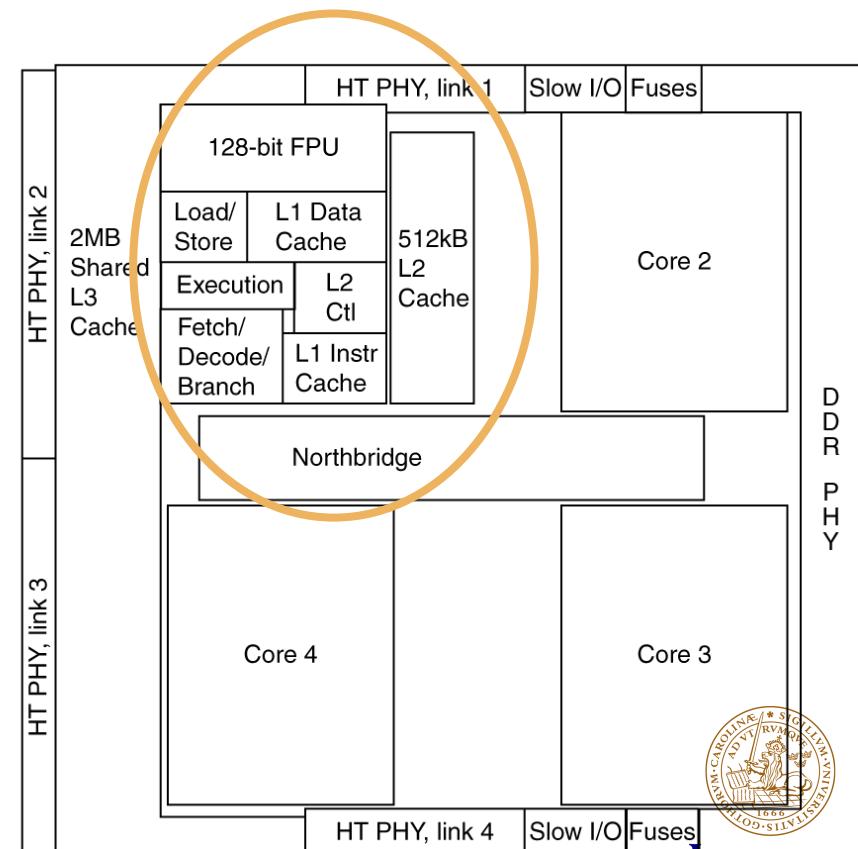
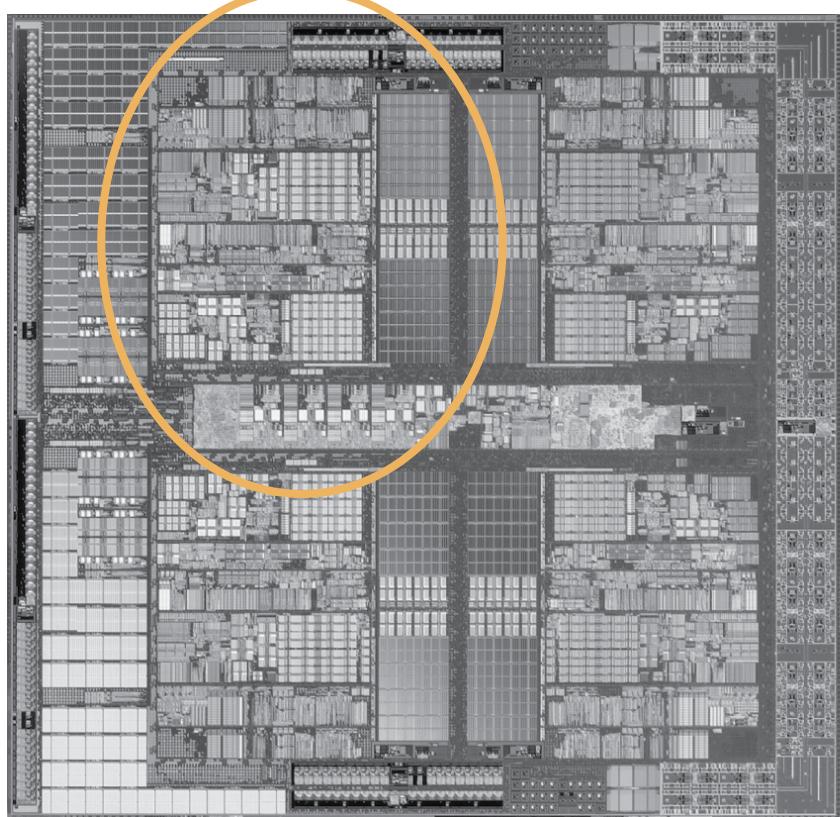
- Volatile main memory
  - Förlorar information (instruktioner och data) när strömmen stängs av
- Non-volatile secondary memory
  - Magnetic disk (hårddisk)
  - Flash memory
  - Optical disk (CDROM, DVD)



LUND  
UNIVERSITET

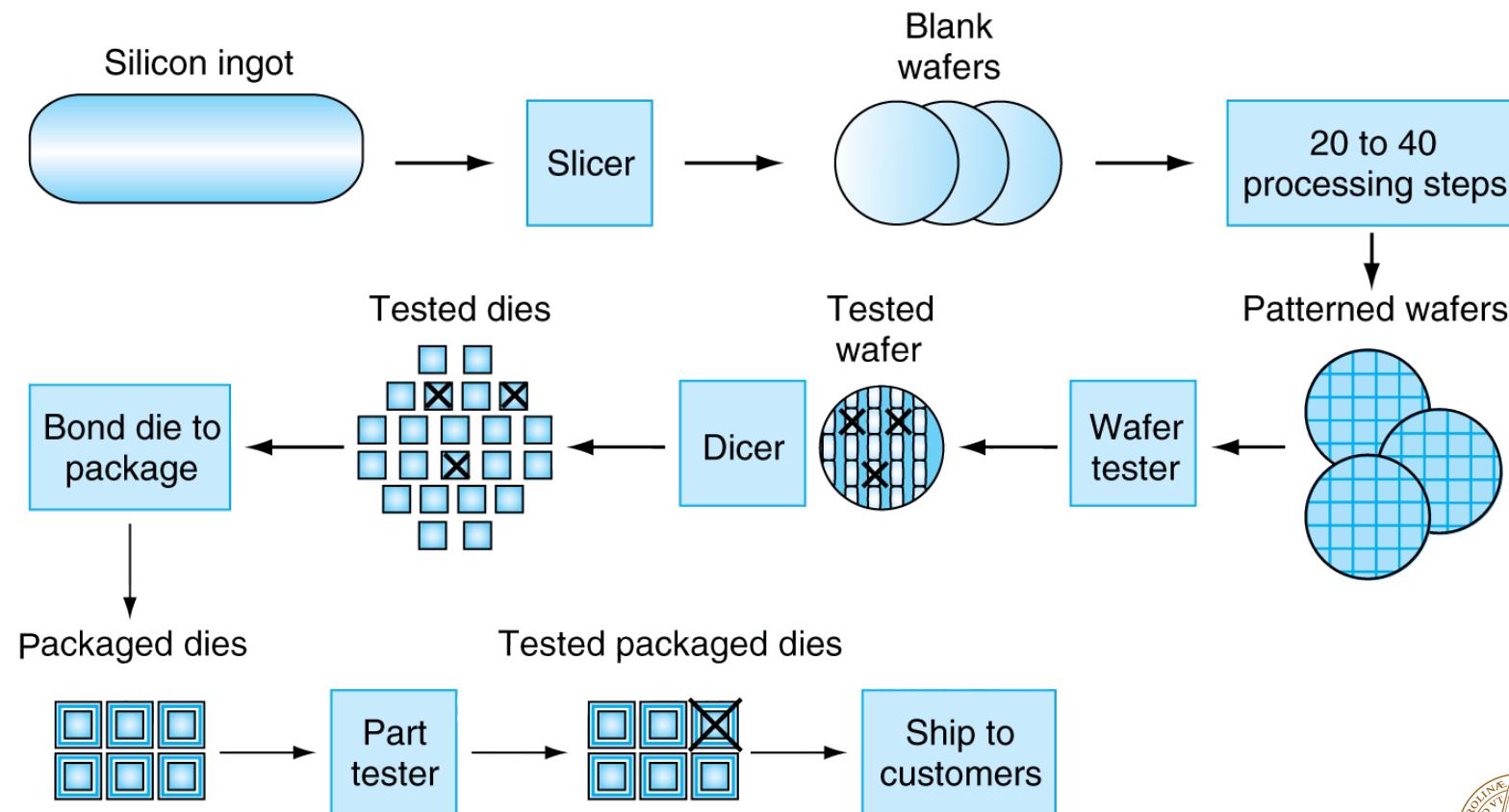
# Processor

- AMD Barcelona: 4 processor cores



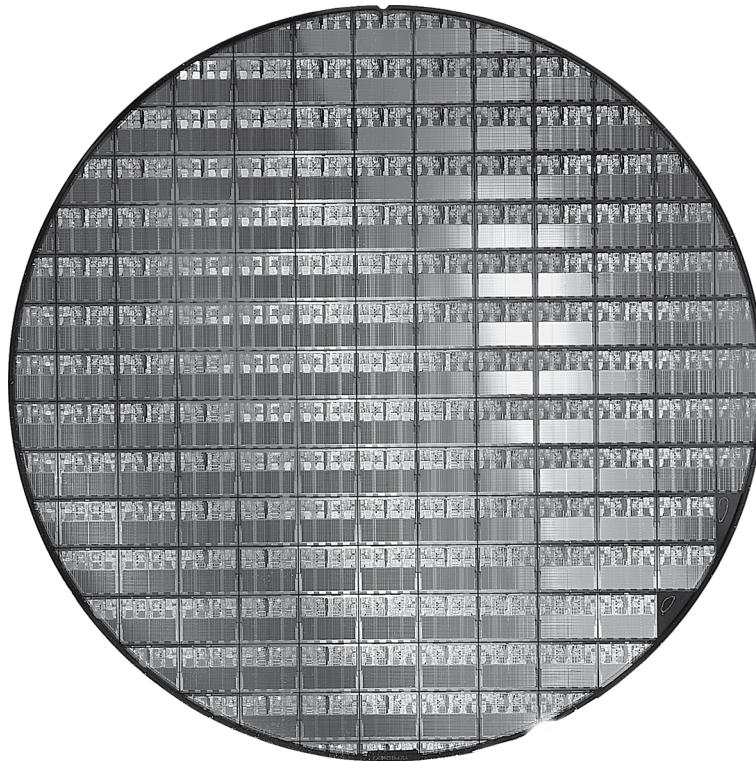
LUND  
UNIVERSITET

# IC tillverkning



# IC, die och wafer

---



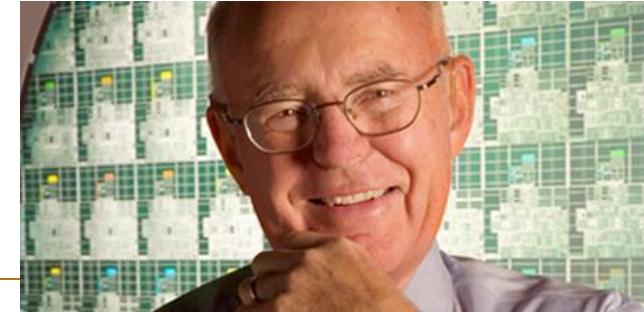
- AMD Opteron X2: 300mm wafer, 117 chips, 90nm technology
- X4: 45nm technology



LUNDS  
UNIVERSITET

# Intel

---

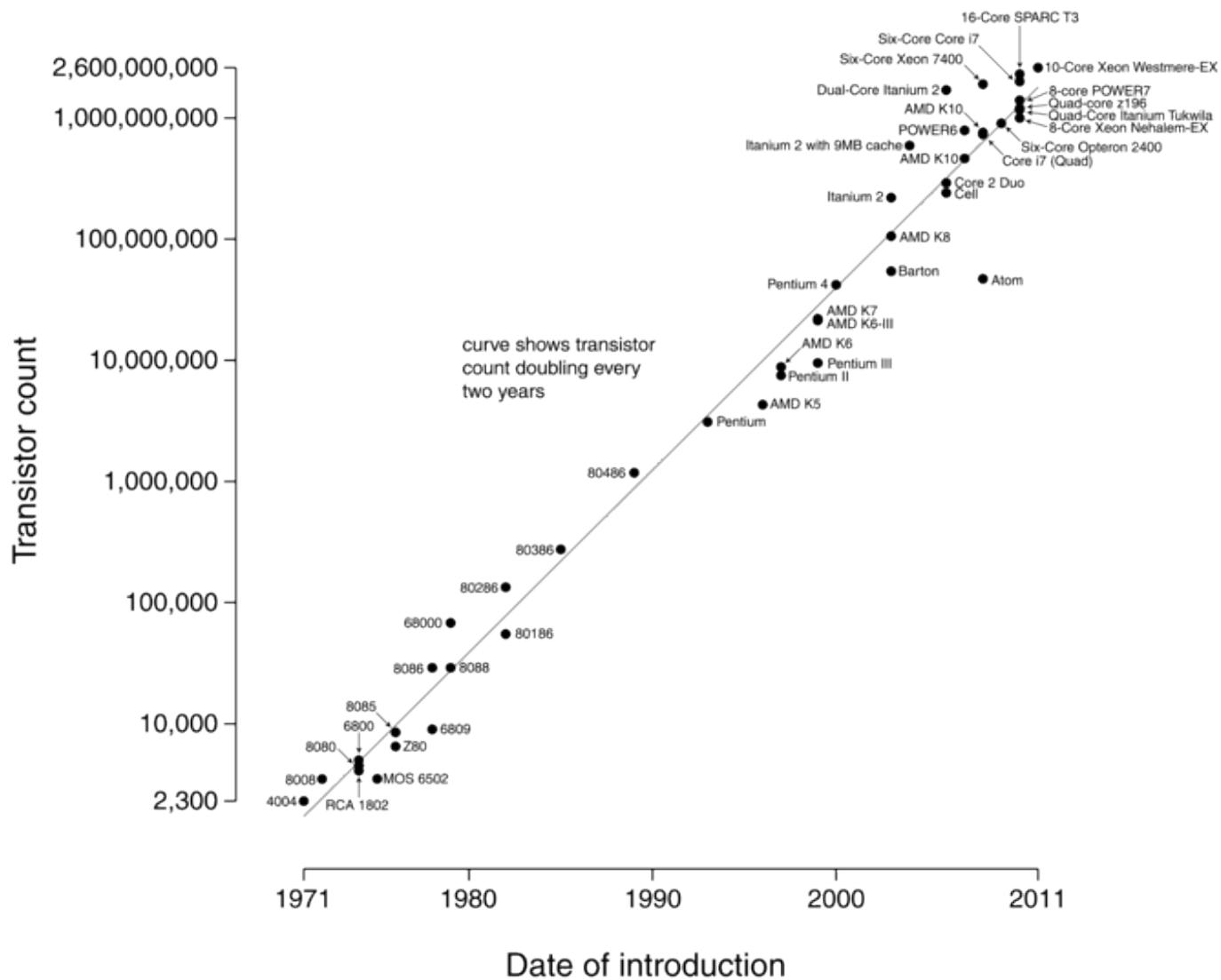


- Gordon E. Moore är en av grundarna av Intel (Integrated Electronics) (tillsammans med Robert Noyce)
- Första tanken på namn var: Moore & Noyce Electronics
- Moores lag (1965): Antalet transistorer på ett chip växer exponentiellt; det dubblas var 24:e månad.



LUND  
UNIVERSITET

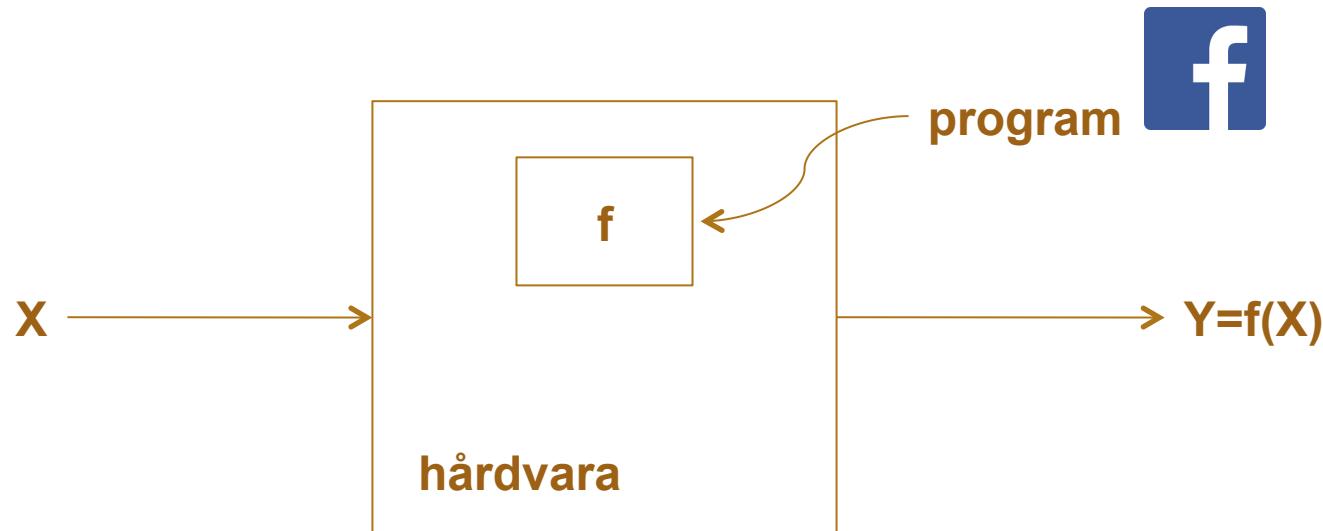
# Moores lag - Trender



LUND  
UNIVERSITET

# Dator

---



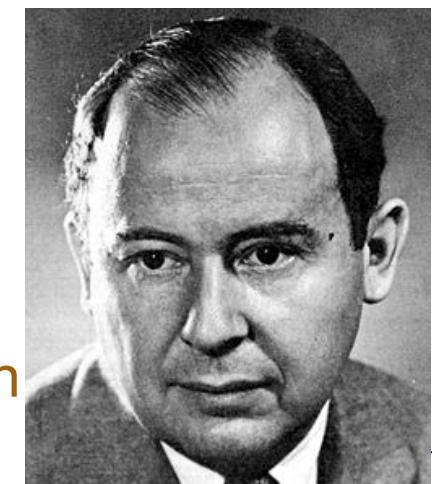
LUNDS  
UNIVERSITET

# Dator

---

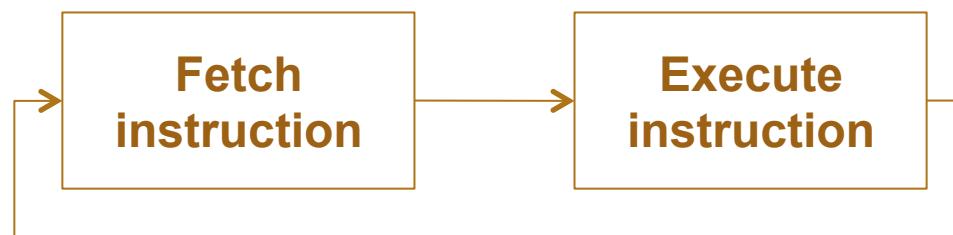
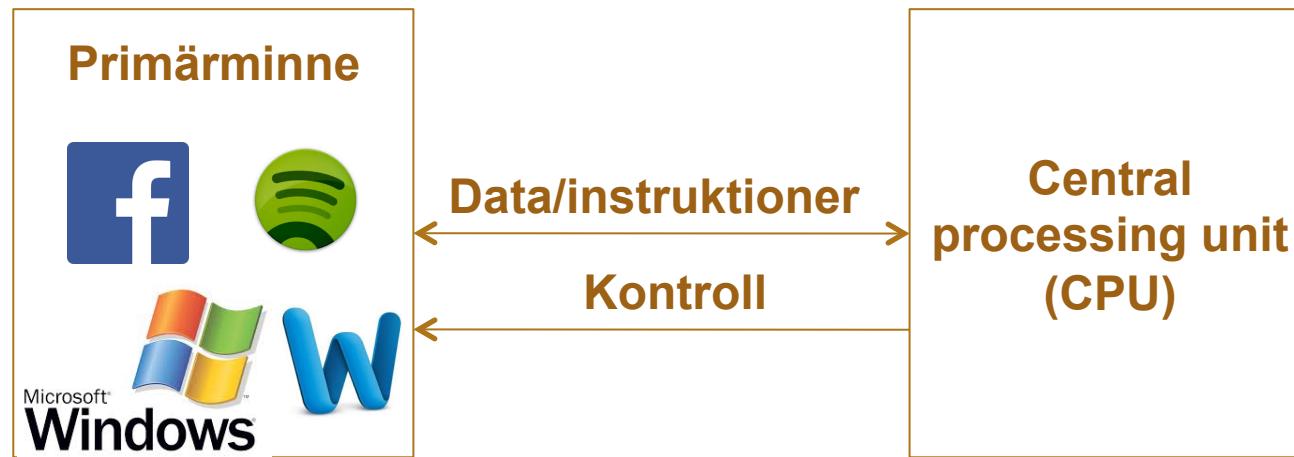
- John von Neumann arkitektur
  - Gemensamt minne för instruktioner och data (vad som är vad bestäms av kontext)
  - Aritmetisk enhet för logiska och aritmetiska operationer
  - Kontrollenhet
- Instruktioner exekveras sekventiellt

- John von Neumann  
(1903-1957)



# Dator

---



# Användarprogram, OS och hårdvara

---

- Applikationsprogram, t ex Word, Facebook, är ofta skrivna i ett högnivåspråk, t ex C, C++.
  - System programvara
    - Kompilatorer: Översätter program i högnivåspråk till maskinspråk
    - Operativsystem:
      - » Hantera I/O, minne och schemaläggning av jobb (tasks)
  - Hårdvara
    - Processor, minne, I/O-kontrollenhet
- 
- The slide features several logos arranged in a grid. In the top row, from left to right, are the Microsoft Word logo (blue 'W'), the Facebook logo (blue 'f'), and the Spotify logo (green circular design). In the middle row, from left to right, are the Linux logo (orange/red swoosh) and the Tux the Penguin icon. In the bottom row, from left to right, are the Android logo (green Android robot), the Microsoft Windows logo (blue, green, and yellow windows), and the Lunds University logo (a circular seal with Latin text and the university's name).
-

# Program

---

- För att bestämma vad som görs – ( $f(X)$ ) – behövs program, programmeringsspråk.
- Programmeringsspråk:
  - Högnivåspråk (C, C++)
  - Assemblyspråk (t ex ADD R1, R2)
  - Maskinspråk (t ex 001101....101)



LUNDS  
UNIVERSITET

# Program

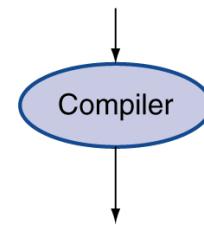
- Abstraktionsnivå:
    - Högnivåspråk
    - Assemblyspråk
    - Maskinspråk

High-level  
language  
program  
(in C)

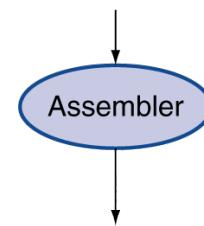
## Assembly language program (for MIPS)

# Binary machine language program (for MIPS)

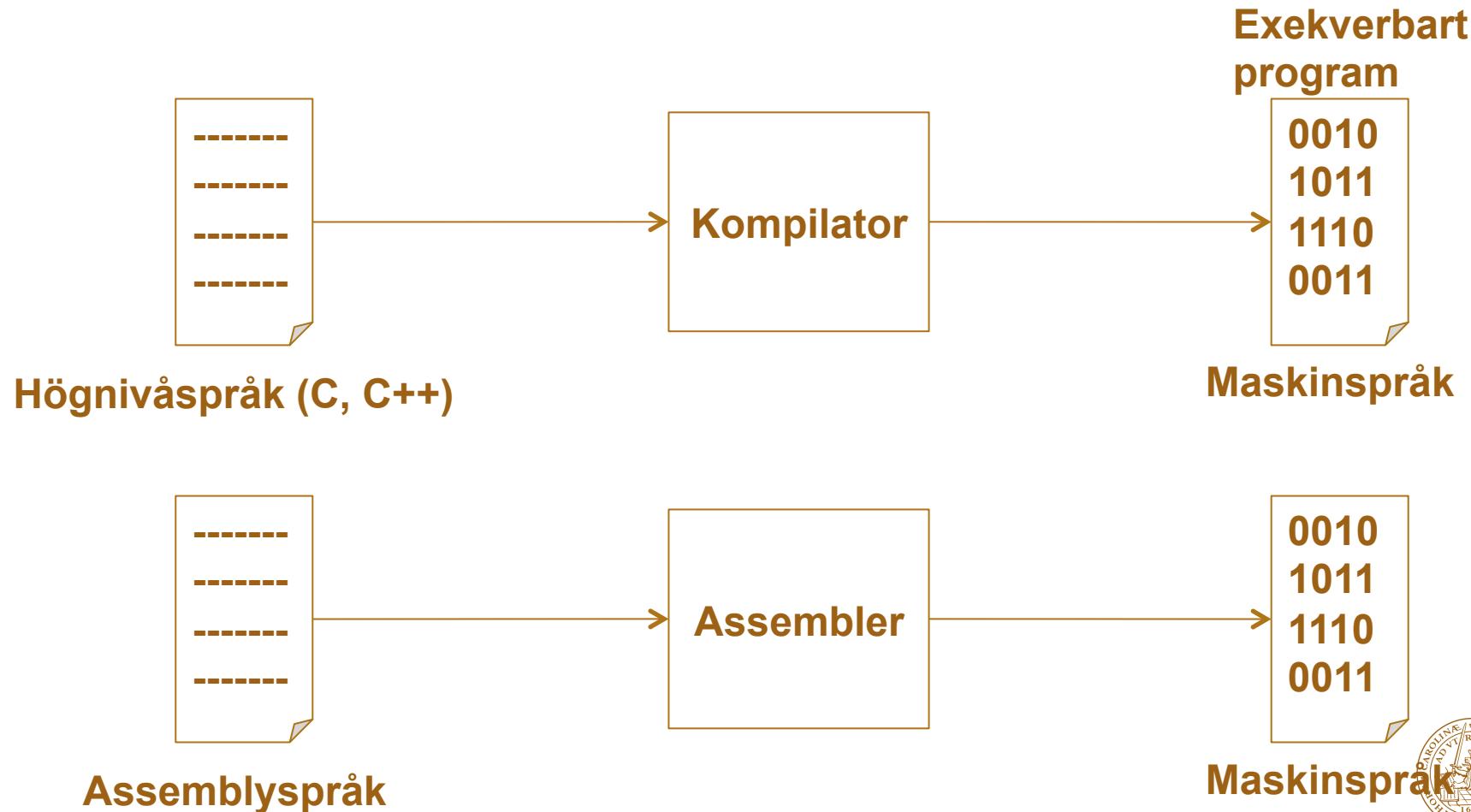
```
swap(int v[], int k)
{int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```



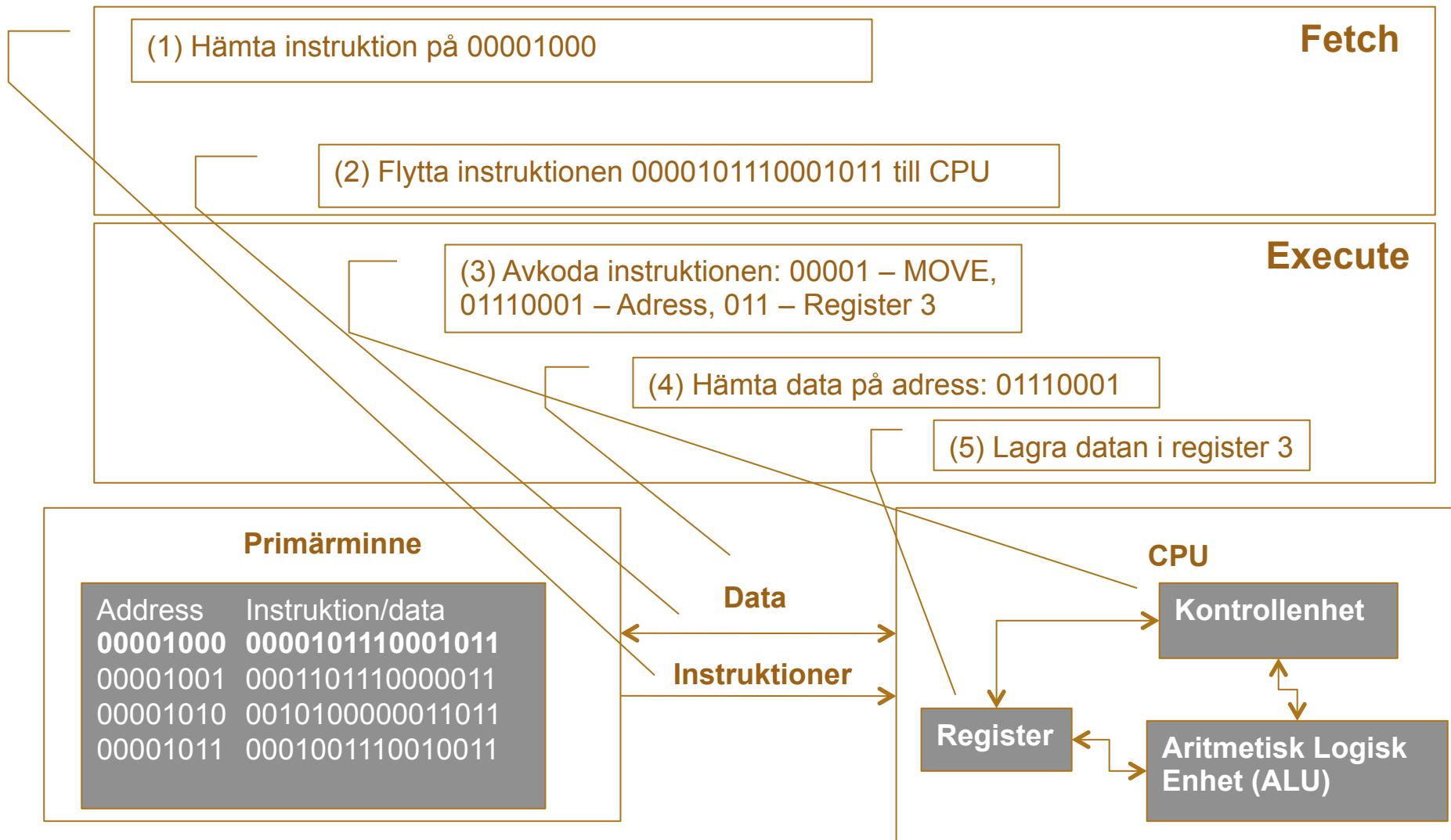
```
swap:  
    muli $2, $5,4  
    add  $2, $4,$2  
    lw   $15, 0($2)  
    lw   $16, 4($2)  
    sw   $16, 0($2)  
    sw   $15, 4($2)  
    jr   $31
```



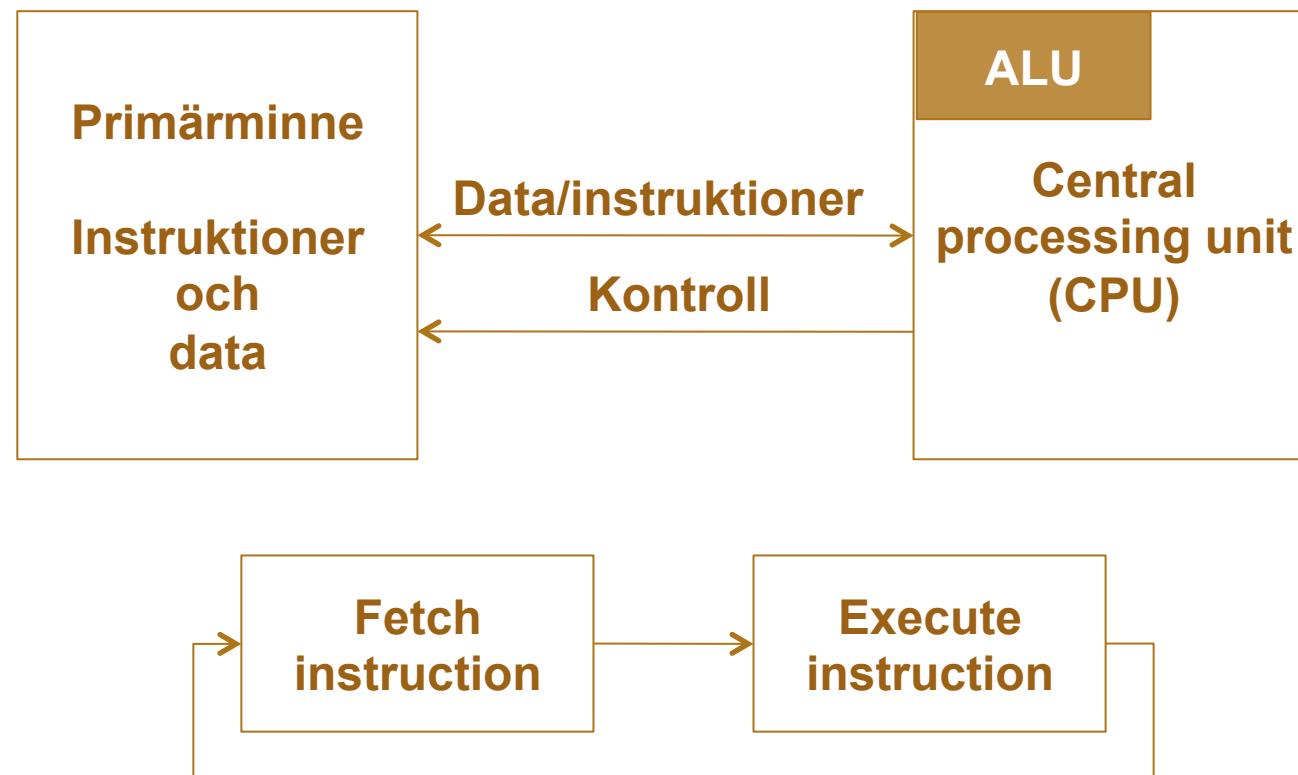
# Program



# Dator

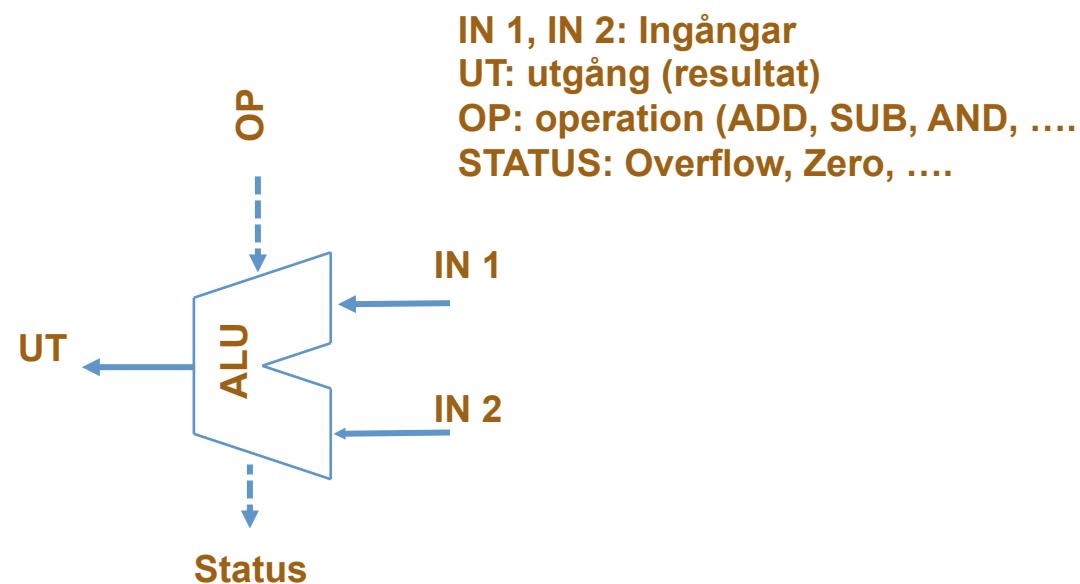


# ALU – arithmetic logic unit



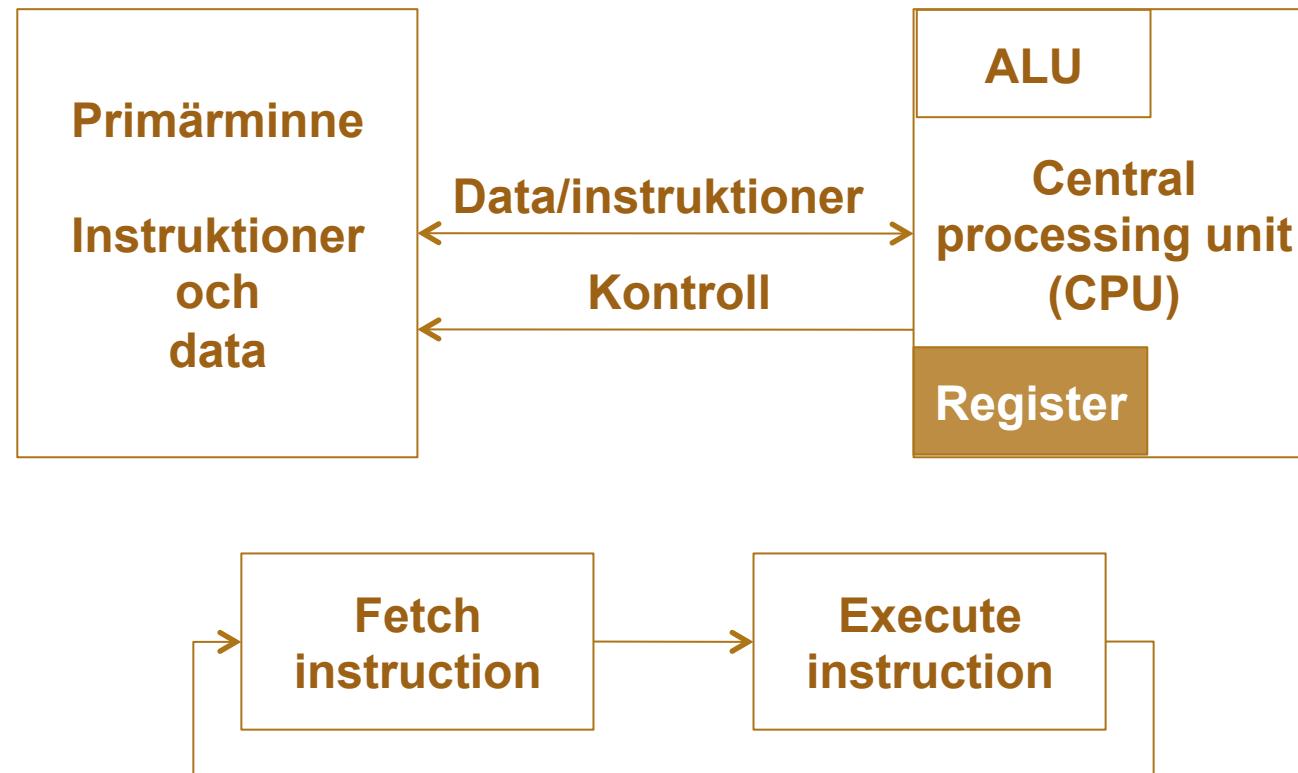
# ALU – arithmetic logic unit

---



LUND  
UNIVERSITET

# Register



LUNDS  
UNIVERSITET

# Register

---

- Alla processorer har någon form av register fil för temporär lagring
- MIPS har en 32x32-bitars registerfil
  - Används för data som används frekvent
  - Numreras 0 till 31
  - 32-bitars data kallas ett "word"
- Assemblerer namn
  - \$t0, \$t1, ..., \$t9 för temporära värden
  - \$s0, \$s1, ..., \$s7 för sparade (subrutin) värden
- Designregel: Mindre minne är snabbare



# Register

---

- Register synliga för användare, som kan vara:
  - Helt generella, eller
  - Specifika för data och instruktioner och/eller
  - Specifika för heltal och flyttal
  - Speciella för t ex multiplikation/division för att kunna hantera resultat eller för adressberäkningar:  
basregister och stackpekare



LUNDS  
UNIVERSITET

# Register

---

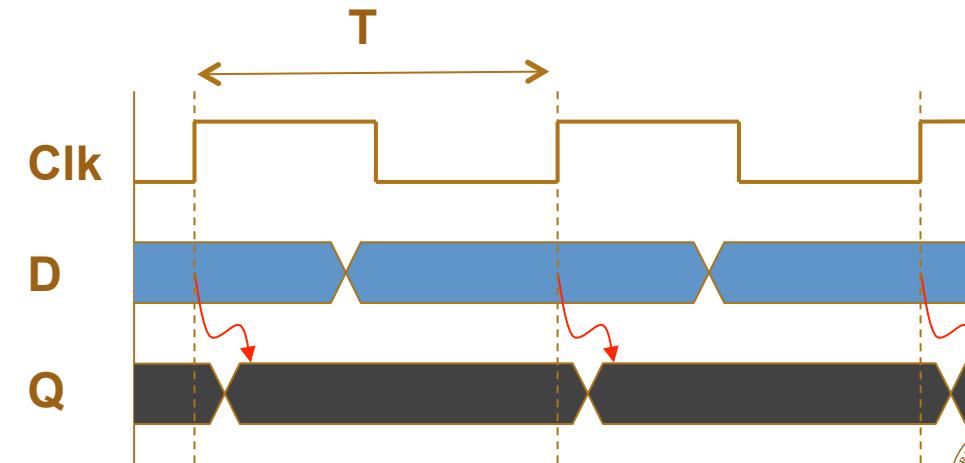
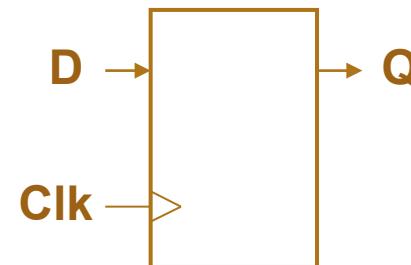
- Kontroll och statusregister
  - Inte direkt kontrollerbara av programmeraren, t ex:
    - » Programräknare (program counter (PC)): adress till den instruktion som ska hämtas.
    - » Instruktions register (IR): senaste instruktionen som hämtats
    - » Memory address register (MAR): adress som ska läsas/skrivas
    - » Memory buffer register (MBR): data som ska skrivas eller data som lästs från minnet
    - » Program status word (PSW): Olika flaggor (är avbrott på eller av, supervisor mode)



LUNDS  
UNIVERSITET

# Sekventiell logik

- Register: lagrar data
- Använder klocka för att bestämma när data ska uppdateras
- Flank-triggat (Edge-triggered): uppdaterar när Clk går från 0 till 1



# Register och primärminne

---

- Högnivåspråk:

$a=b+c;$

- Assemblyspråk:

ADD a, b, c

- Kompilatorn hittar en maskininstruktion som passar

- 3 minnesaccesser:

- Hämta b
  - Hämta c
  - Skriv a



LUND  
UNIVERSITET

# Register och primärminne

---

- Högnivåspråk:  $a=b+c+d$ ;
- Alternativ 1:
  - ADD a, b, c //addrar b och c, resultat i a
  - ADD a, a, d //addrar a (som är  $b+c$ ) med d,  
resultat i a
- Antal minnesaccesser:  
$$6=3+3$$
- Tid för en minnesaccess: 100 ns gör att minnesläsning/  
skrivning tar 600 ns



# Register och primärminne

---

- Högnivåspråk:  $a=b+c+d$ ;

- Alternativ 2:

ADD r1, b, c //addrar b och c, resultat i register 1

ADD a, r1, d //addrar a (som är i r1) med d,  
resultat i a

- Antal minnesaccesser:

$$4=2+2$$

- Tid för en:

- minnesaccess: 100 ns

- Registeraccess: 1 ns:

- Minnesläsning/skrivning tar  $400 + 2$  ns = 402 ns

30% sparad tid  
 $(600-402)/600$



# Primärminne

---

LOAD – läs data i minne

data läses från en given  
minnesplats (adress) och  
hamnar i processorn

STORE – skriv data i minne

data skrivs in i minnet på  
en given adress

Adress	Instruktion/Data
0	1111 0000
1	1010 0101
2	1100 0011
3	0011 0011
4	1111 1111
5	0000 1111
6	1111 0000
7	1010 1010

# Variabler och minne

---

- Exempel, C deklaration:

```
x unsigned char; // värde mellan 0 och 255
```

1 byte – 8 bitar ger att dessa tal kan representeras:

Binärt	Decimalt
0000 0000	0
0000 0001	1
0000 0010	2
0000 0011	3
.....	
1111 1101	253
1111 1110	254
1111 1111	255



LUNDS  
UNIVERSITET

# Variabler och minne

---

- Exempel, C deklaration

x signed char; // värde mellan -128 och 127

1 byte – 8 bitar: Hur representera “-” talen?

Binärt	Decimalt
0000 0000	0
0000 0001	1
0000 0010	2
0000 0011	3
.....	
1111 1101	?
1111 1110	?
1111 1111	?



LUNDS  
UNIVERSITET

# Variabler och minne

---

- Alternativ 1: MSB är teckenbit //Most Significant Bit
- Får 2 nollor (-0 och 0)

Binärt	Decimalt
0000 0000	0
0000 0001	1
0000 0010	2
...	....
1000 0000	-0
....	....
1111 1110	-126
1111 1111	-127



# Variabler och minne

---

- Alternativ 2: Två-kompliment

Binärt	Decimalt
0000 0000	0
0000 0001	1
0000 0010	2
...	....
1000 0000	-127
.....	....
1111 1110	-2
1111 1111	-1

- Exempel:  $-1 (1111 1111) + 1 = 0$

# Minne

---

- Exempel:

C deklaration x char;  
// char behöver 1 byte

LOAD r1, 3

// ladda register r1 med data  
som finns på plats 3

STORE r1, 3

/ spara det som finns i  
register r1 på plats 3

Adress	Instruktion/Data
0	1111 0000
1	1010 0101
2	1100 0011
3	0011 0011
4	1111 1111
5	0000 1111
6	1111 0000
7	1010 1010

# Minne

---

- Exempel:

C deklaration x int;  
// int behöver 2 bytes

LOAD r1, 3

// ladda register r1 med data  
som finns på plats 3 och ?

STORE r1, 3

/ spara det som finns i  
register r1 på plats 3 och ?

Adress	Instruktion/Data
0	1111 0000
1	1010 0101
2	1100 0011
3	0011 0011
4	1111 1111
5	0000 1111
6	1111 0000
7	1010 1010

# Byte eller wordadresserat minne

Adress	Byte	Data
0	0	1111 0000
1	1	1010 0101
2	2	1100 0011
3	3	0011 0011
4	4	1111 1111
5	5	0000 1111
6	6	1111 0000
7	7	1010 1010

Adress	Byte	Data
0	0	1111 0000
	1	1010 0101
	2	1100 0011
	3	0011 0011
1	4	1111 1111
	5	0000 1111
	6	1111 0000
	7	1010 1010

# Byte eller word adresserat minne?

---

- Byteadresserat
  - Måste hålla koll på vad som ska anses vara ord
  - Exempel: Läs adress 4, som är en del i ett ord.
- Wordadresserat:
  - Läs word (4 byte) per gång.
  - Fragmentering om en byte ska lagras (tar 4 bytes).

Adress	Byte	Data
0	0	1111 0000
1	1	1010 0101
2	2	1100 0011
3	3	0011 0011
4	4	1111 1111
5	5	0000 1111
6	6	1111 0000
7	7	1010 1010

# Hur sätts byte ihop till word?

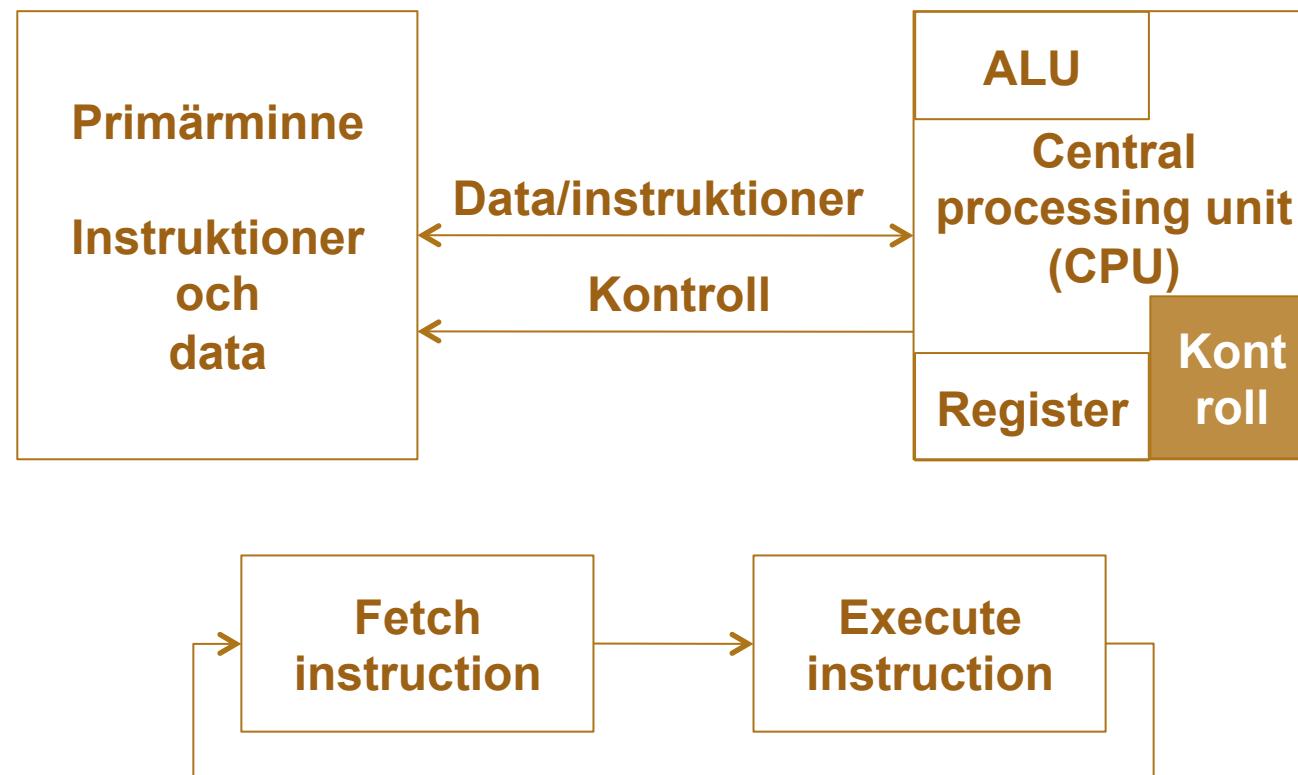
---

- Byte 4, 5, 6, och 7 bildar ett ord, men hur ser ordet ut?
- Två alternativ:
  - 1111 1111 0000 1111  
1111 0000 1010 1010  
(ordning 4, 5, 6, 7)  
Little Endian
  - 1010 1010 1111 0000  
0000 1111 1111 1111  
(ordning 7, 6, 5, 4)  
Big Endian

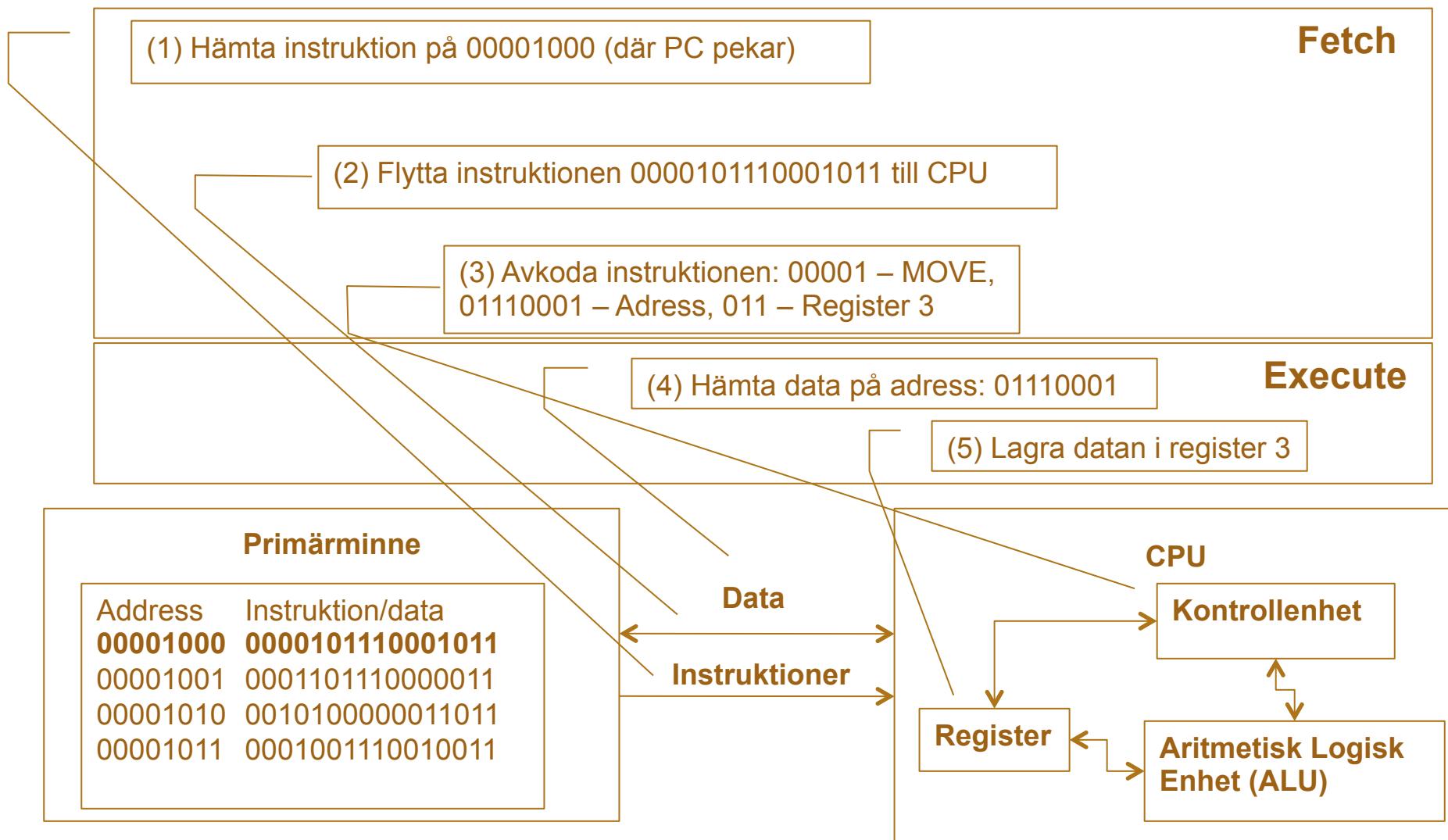
Adress	Byte	Data
0	0	1111 0000
1	1	1010 0101
2	2	1100 0011
3	3	0011 0011
4	4	1111 1111
5	5	0000 1111
6	6	1111 0000
7	7	1010 1010

# Kontrollenhet

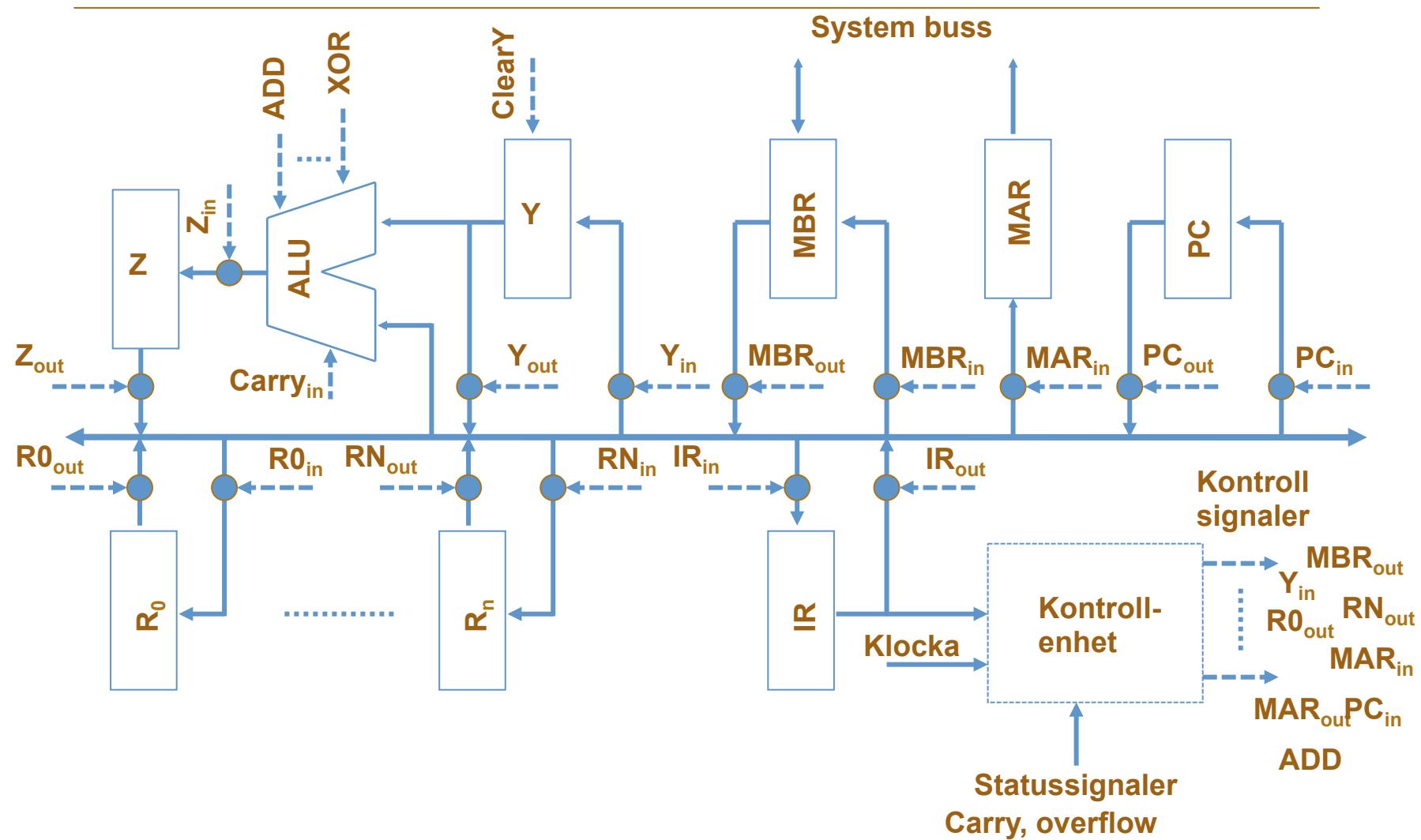
---



# Exekvering av en instruktion



# Kontrollenhet



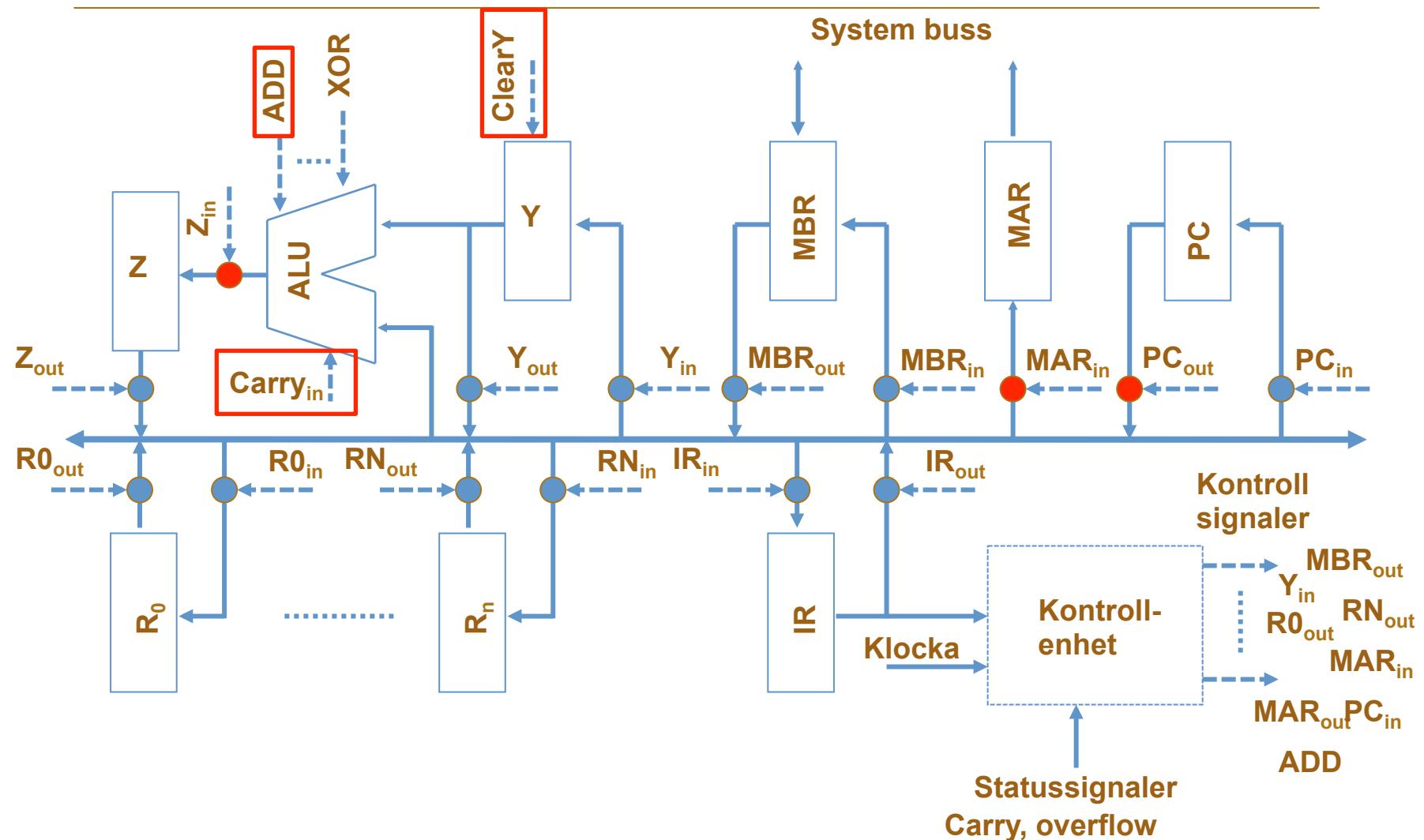
# Kontrollenhet

---

- Instruktion:
  - ADD R1, R3 //  $R1 \leftarrow R1 + R3$
- Kontrollsteg:
  1.  $PC_{out}$ ,  $MAR_{in}$ , Read, Clear Y, Carry-in, Add,  $Z_{in}$
  2.  $Z_{out}$ ,  $PC_{in}$
  3.  $MBR_{out}$ ,  $IR_{in}$
  4.  $R1_{out}$ ,  $Y_{in}$
  5.  $R3_{out}$ , Add,  $Z_{in}$
  6.  $Z_{out}$ ,  $R1_{in}$ , End



# Kontrollenhet



# Kontrollenhet

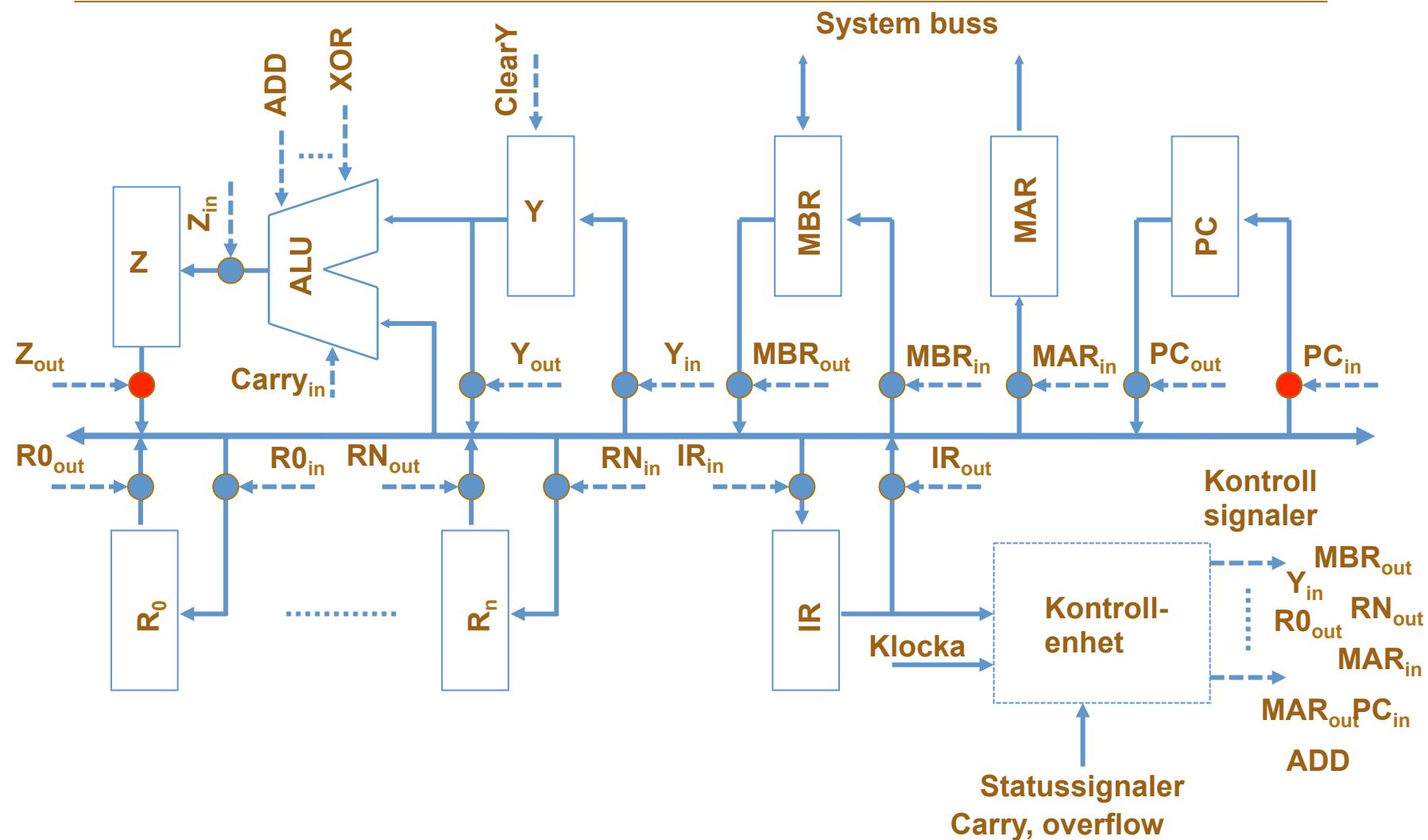
---

- Instruktion:
  - ADD R1, R3 //  $R1 \leftarrow R1 + R3$
- Kontrollsteg:
  1.  $PC_{out}$ ,  $MAR_{in}$ , Read, Clear Y, Carry-in, Add,  $Z_{in}$
  2.  $Z_{out}$ ,  $PC_{in}$
  3.  $MBR_{out}$ ,  $IR_{in}$
  4.  $R1_{out}$ ,  $Y_{in}$
  5.  $R3_{out}$ , Add,  $Z_{in}$
  6.  $Z_{out}$ ,  $R1_{in}$ , End



LUNDS  
UNIVERSITET

# Kontrollenhet



# Kontrollenhet

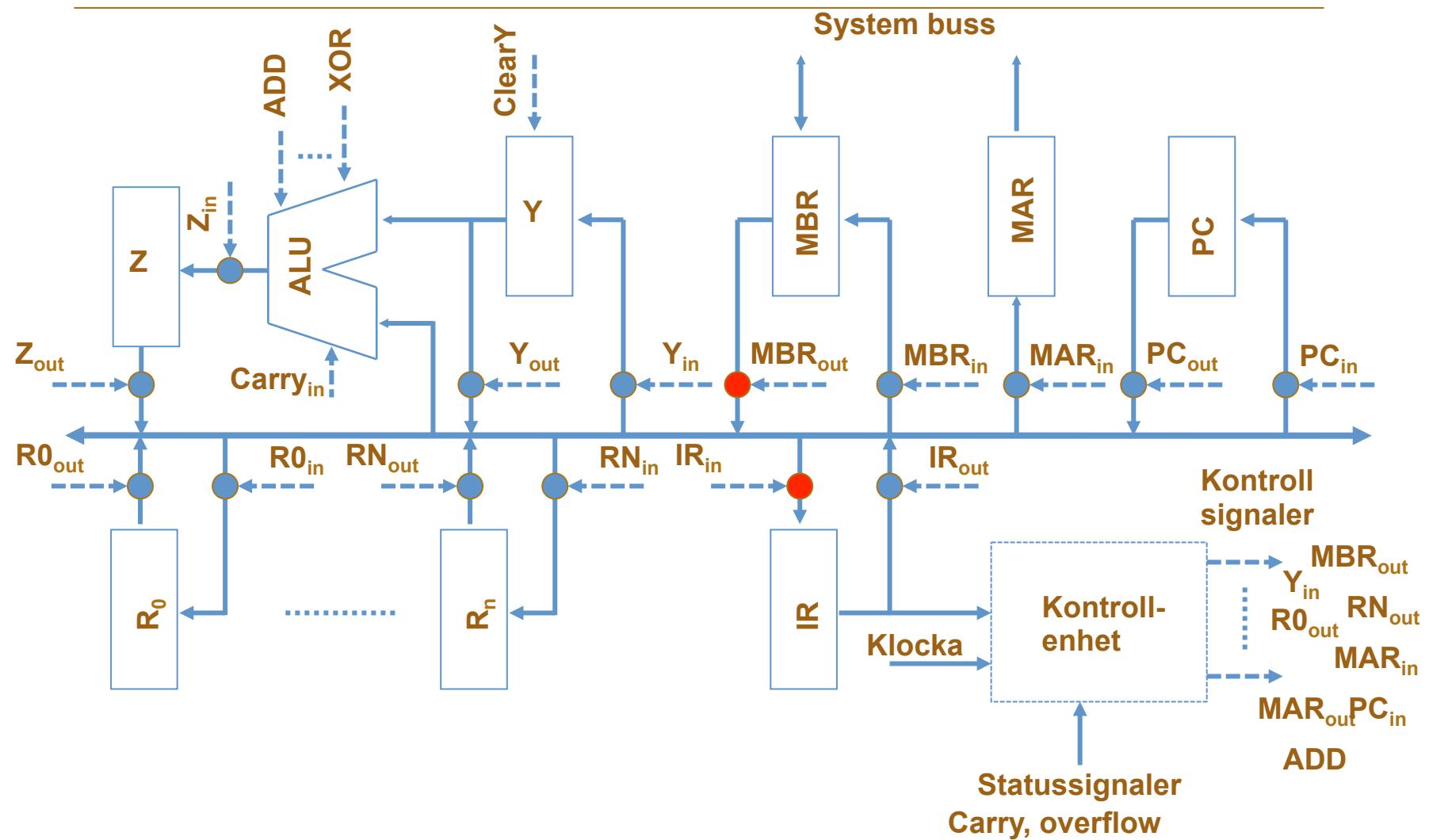
---

- Instruktion:
  - ADD R1, R3 //  $R1 \leftarrow R1 + R3$
- Kontrollsteg:
  1.  $PC_{out}$ ,  $MAR_{in}$ , Read, Clear Y, Carry-in, Add,  $Z_{in}$
  2.  $Z_{out}$ ,  $PC_{in}$
  3. MBR<sub>out</sub>, IR<sub>in</sub>
  4.  $R1_{out}$ , Y<sub>in</sub>
  5.  $R3_{out}$ , Add,  $Z_{in}$
  6.  $Z_{out}$ ,  $R1_{in}$ , End



LUNDS  
UNIVERSITET

# Kontrollenhet



# Kontrollenhet

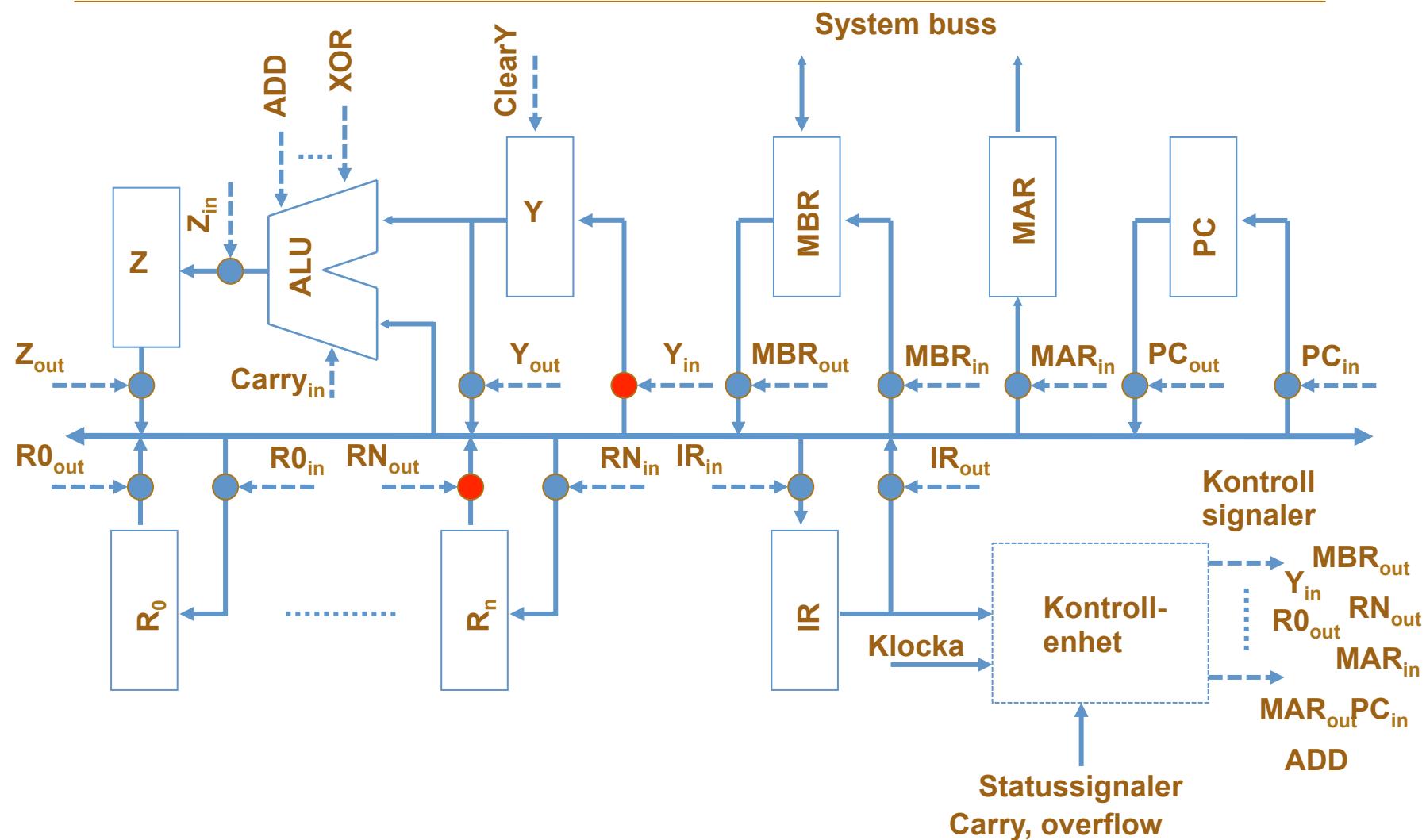
---

- Instruktion:
  - ADD R1, R3 //  $R1 \leftarrow R1 + R3$
- Kontrollsteg:
  1.  $PC_{out}$ ,  $MAR_{in}$ , Read, Clear Y, Carry-in, Add,  $Z_{in}$
  2.  $Z_{out}$ ,  $PC_{in}$
  3.  $MBR_{out}$ ,  $IR_{in}$
  4.  $R1_{out}$ ,  $Y_{in}$
  5.  $R3_{out}$ , Add,  $Z_{in}$
  6.  $Z_{out}$ ,  $R1_{in}$ , End



LUNDS  
UNIVERSITET

# Kontrollenhet



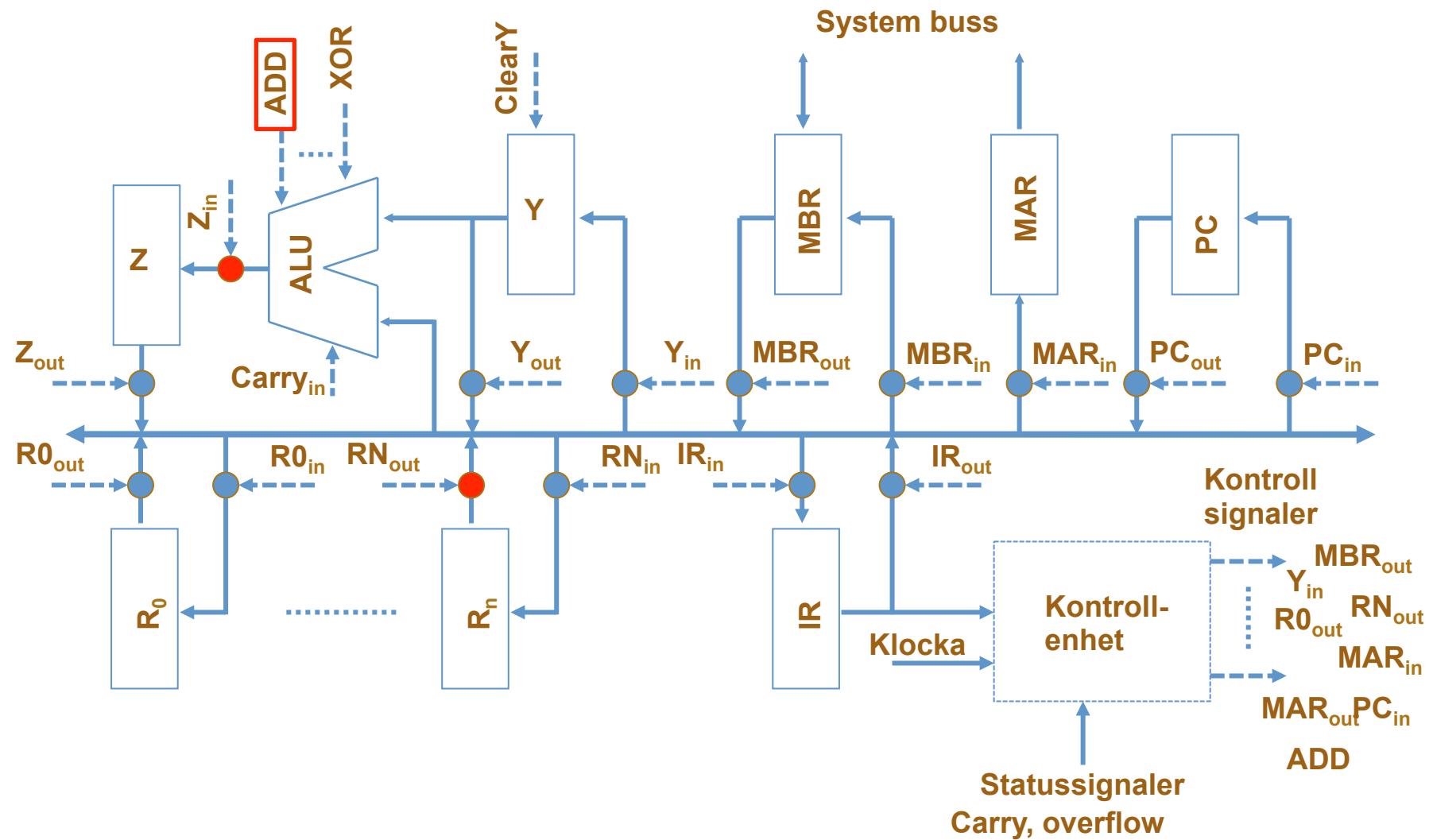
# Kontrollenhet

---

- Instruktion:
  - ADD R1, R3 //  $R1 \leftarrow R1 + R3$
- Kontrollsteg:
  1.  $PC_{out}$ ,  $MAR_{in}$ , Read, Clear Y, Carry-in, Add,  $Z_{in}$
  2.  $Z_{out}$ ,  $PC_{in}$
  3.  $MBR_{out}$ ,  $IR_{in}$
  4.  $R1_{out}$ ,  $Y_{in}$
  5.  $R3_{out}$ , Add,  $Z_{in}$
  6.  $Z_{out}$ ,  $R1_{in}$ , End



# Kontrollenhet



# Kontrollenhet

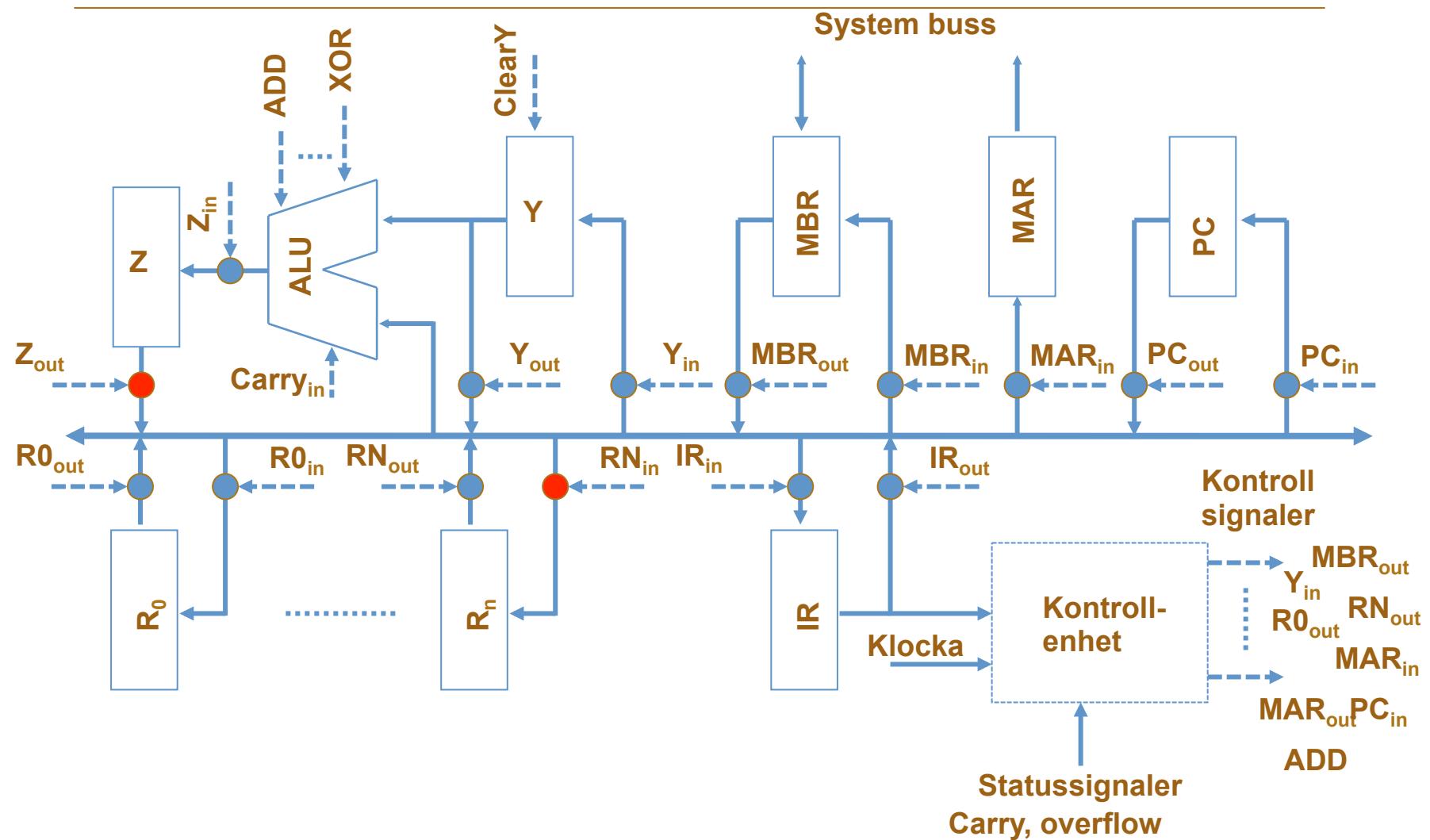
---

- Instruktion:
  - ADD R1, R3 //  $R1 \leftarrow R1 + R3$
- Kontrollsteg:
  1.  $PC_{out}$ ,  $MAR_{in}$ , Read, Clear Y, Carry-in, Add,  $Z_{in}$
  2.  $Z_{out}$ ,  $PC_{in}$
  3.  $MBR_{out}$ ,  $IR_{in}$
  4.  $R1_{out}$ ,  $Y_{in}$
  5.  $R3_{out}$ , Add,  $Z_{in}$
  6.  $Z_{out}$ ,  $R1_{in}$ , End



LUND  
UNIVERSITET

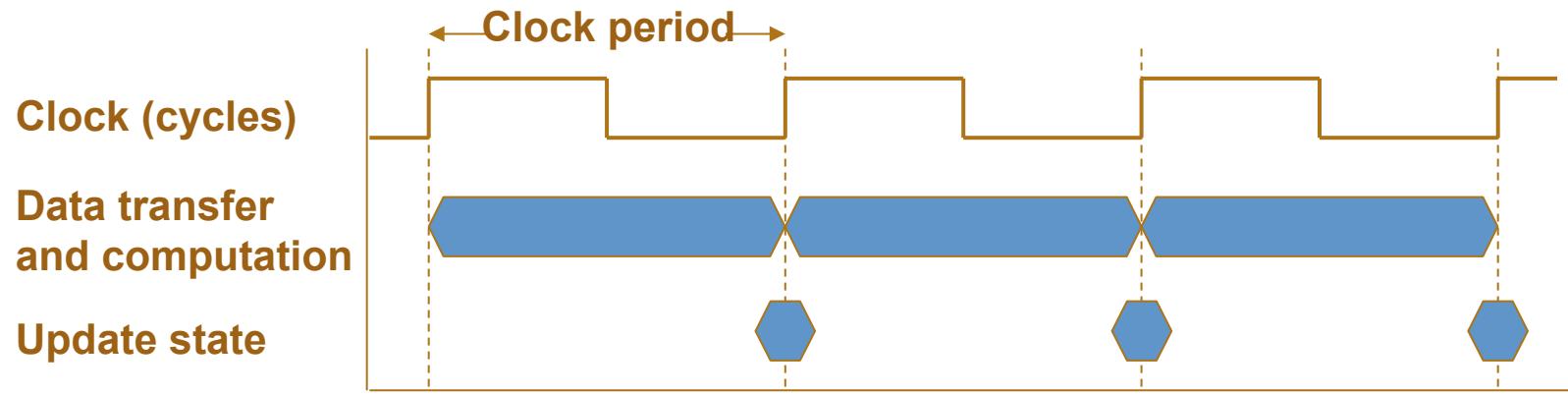
# Kontrollenhet



# Klockning

---

- En klocka styr hastigheten på digital hårdvara



- Klockperiod: tid för en klockcykel
  - e.g.,  $250\text{ps} = 0.25\text{ns} = 250 \times 10^{-12}\text{s}$
- Clock frequency (rate): cycles per second
  - e.g.,  $4.0\text{GHz} = 4000\text{MHz} = 4.0 \times 10^9\text{Hz}$



# CPU tid

---

CPU Time = CPU Clock Cycles × Clock Cycle Time

- Prestanda kan öka genom:
  - Minska antal klockcykler (clock cycles)
  - Öka klockfrekvensen (clock rate)



LUNDS  
UNIVERSITET

# Exekveringstid

---

- Instruktion:
  - ADD R1, R3 //  $R1 \leftarrow R1 + R3$
- Kontrollsteg = Klockcykler per instruktion (CPI)
  1.  $PC_{out}$ ,  $MAR_{in}$ , Read, Clear Y, Carry-in, Add,  $Z_{in}$
  2.  $Z_{out}$ ,  $PC_{in}$
  3.  $MBR_{out}$ ,  $IR_{in}$
  4.  $R1_{out}$ ,  $Y_{in}$
  5.  $R3_{out}$ , Add,  $Z_{in}$
  6.  $Z_{out}$ ,  $R1_{in}$ , End



# Exekveringstid

---

- Antal klockcykler för att exekvera en maskininstruktion
  - Clocks per instruction (CPI)
- Tid för en klockcykel
  - Tid för en klockperiod (T)
  - Frekvens (f) är:  $f=1/T$
- Antal maskininstruktioner
  - Instruction count (IC)
- Exekveringstid i klockcykler = CPI x T x IC



# Exekveringstid

---

- Prestanda kan ökas genom:
  - Öka klockfrekvensen ( $f$ ) (minskar  $T$ )
  - Minska antal instruktioner (IC)
  - Minska antal klockcykler per instruktion (CPI)



LUNDS  
UNIVERSITET

# Prestanda

---

- Algoritm
  - Bestämmer vilka och hur många *operationer* som ska utföras
- Programmeringsspråk, kompilator, arkitektur
  - Bestämmer hur många *maskininstruktioner* som ska utföras per *operation*
- Processor och minnessystem
  - Bestämmer hur snabbt instruktioner exekveras
- I/O (Input/Output) och operativsystem
  - Bestämmer hur snabbt I/O operationer ska exekveras





LUNDS  
UNIVERSITET