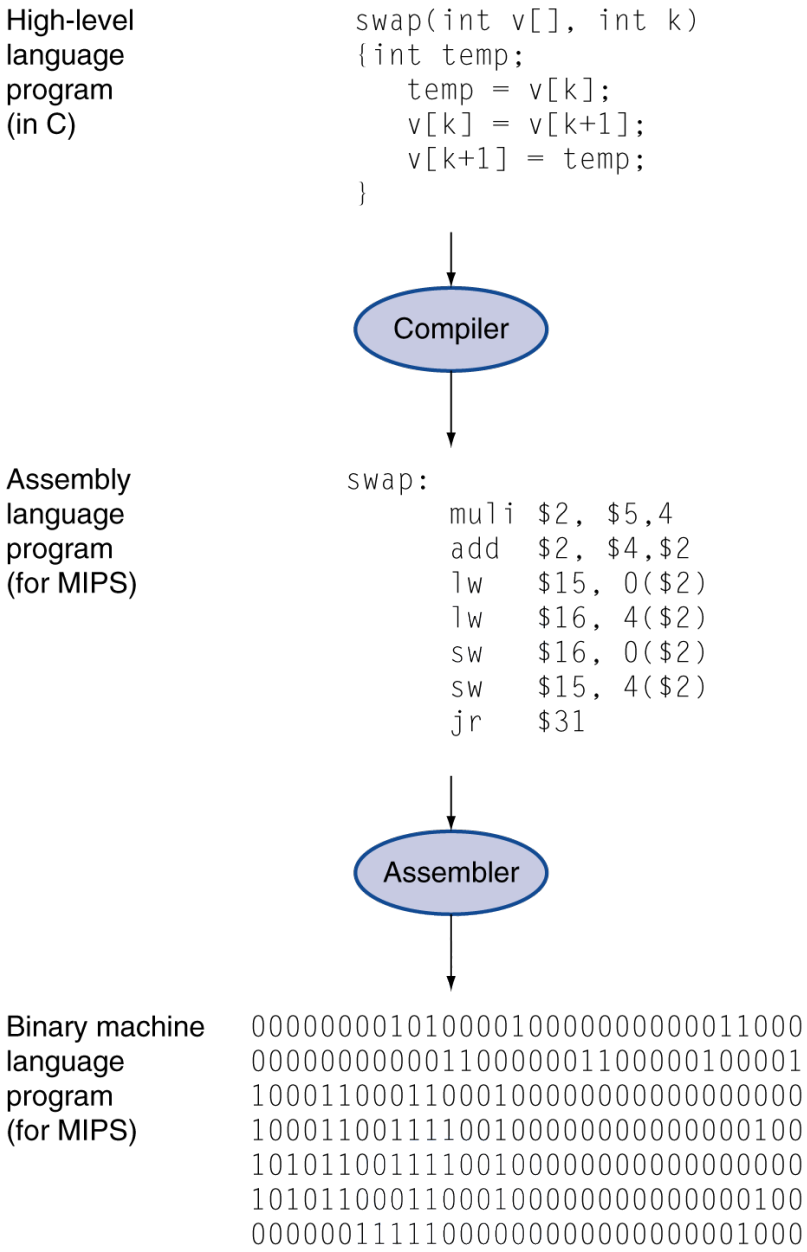
LUNDS UNIVERSITET

Datorteknik

ERIK LARSSON

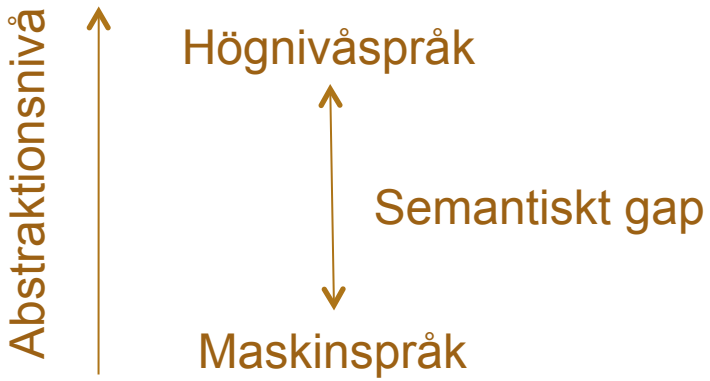
Program

- Abstraktionsnivå:
  - Högnivåspråk
    - » t ex C, C++
  - Assemblyspråk
    - » t ex ADD R1, R2
  - Maskinspråk
    - » t ex 001101….101



Semantiskt gap

- Alltmer avancerade programmeringsspråk tas fram för att göra programvaruutveckling mer kraftfull
- Dessa programmeringsspråk (Ada, C++, Java) ger högre abstraktionsnivå, konsistens och kraft
- Det semantiska gapet ökar (högnivåspråk-maskinspråk)



LUNDS UNIVERSITET

Datatyper

Datatyp	Antal bytes	Talområde
unsigned char	1	0 — 255
signed char	1	-128 — 127
unsigned int	2	0 — 65535
signed int	2	-32768 — 32767
unsigned long int	4	0 — 4294967295
signed long int	4	-2147483648 — 2147483647
float	4	±1,18 E-38 — 3,39 E+38

LUNDS UNIVERSITET

Datatyper

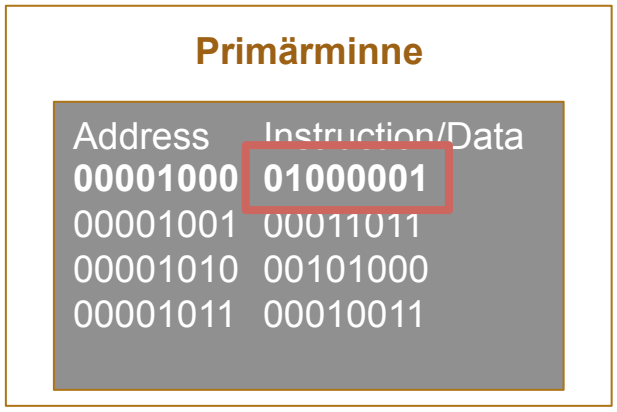
- `char Tal, Max, Min;`
  - `Exempel: Tal=5;`
- `unsigned int Adress;`
  - `Exempel: Adress=512;`
-
```
const char Tabell [][3] = {{ 23, 30, 64 } ,
                           { 12, 31, 16 } ,
                           { 42, 54, 86 } ,
                           { 29, 32, 64 }};
```
  - `Exempel: Tabell[1][1]=23;`
- `const char String [] = "ABC";`

LUNDS UNIVERSITET

Datatyper

- Samma sak lagras:

```
char a = 65;          decimalt
char a = 0x41;        hexadecimalt
char a = 0b01000001;  binärt
char a = 'A';         ASCII-kod
```



LUNDS UNIVERSITET

Tilldelningssatser

- Deklarera variabel:

```
unsigned char a;
```

- Tilldela variablen ett värde:

```
a = 5;
```

- Addera 2 till värdet i a:

```
a = a + 2;
```

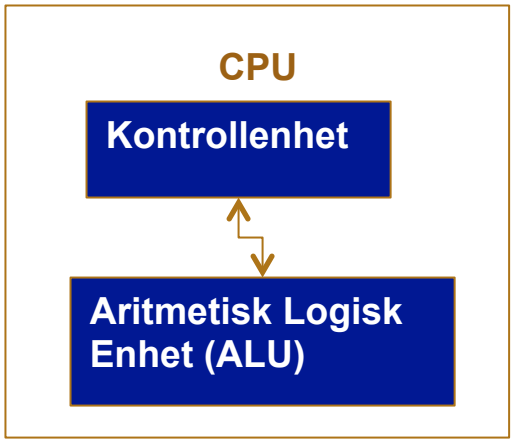
Primärminne

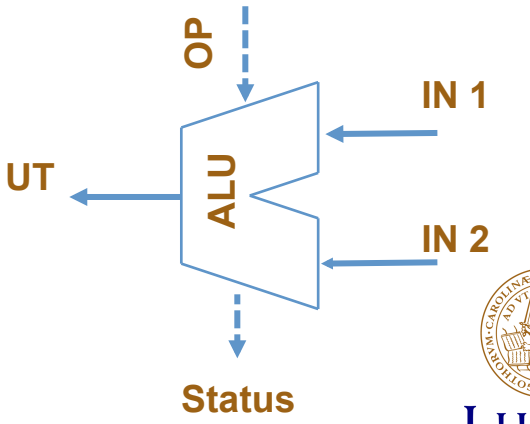
Address	Instruction/Data
00001000	**01000001**
00001001	00011011
00001010	00101000
00001011	00010011

LUNDS UNIVERSITET

Aritmetiska operationer

- Operationer:
  - + addition
  - - subtraktion
  - * multiplikation
  - / division
  - % Modulodivision





LUNDS UNIVERSITET

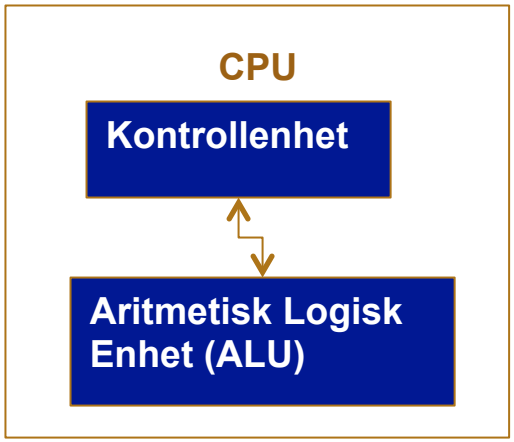
Beräkningar

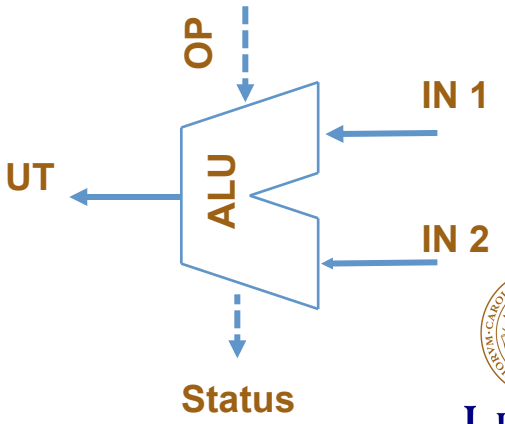
- `unsigned char a, b, c;`
- Exempel:

```
a=5;
b=10+a-3;
c=a*b;
a=b|20;          // bitvis eller (or)
b++;             //samma som b=b+1
```

Bithantering

- Och (AND): &
- Eller (OR): |
- Exklusivt eller (XOR): ^
- Invertering (NOT): ~
- Vänstershift: <<
- Högershift: >>





LUNDS UNIVERSITET

Bithantering

- Exempel:

```
c1=5;       /* c1 har bitmönstret 00000101 */
c2=6;       /* c2 har bitmönstret 00000110 */


c9=~c1;     /* c9 får bitmönstret 11111010 */
c10=c1<<3; /* c10 får bitmönstret 00101000 */
c11=c1<<6; /* c11 får bitmönstret 01000000 */
c12=c1>>2   /* c12 får bitmönstret 00000001 */
c13=c1&c2   /* c13 får bitmönstret 00000100 */
c14=c1|c2   /* c14 får bitmönstret 00000111 */
```

LUNDS UNIVERSITET

Villkor

- `if ( villkor ) sats;`
- Exempel 1:

```
int n
if ( n == 27 ) { din kod här }
```

- Exempel 2:

```
int n;
if ( n == 27 ) { din kod om talet var 27 }
else { din kod om talet inte var 27 }
```

LUNDS UNIVERSITET

Villkorsuttryck

- Om a=5 så öka a med 1:

```
if (a == 5) a = a +1;      // if (villkor) sats;
```

- Villkor:

  == lika med, > större än, < mindre än, != inte lika med

- Om a=10 så öka a med 2 i annat fall minska a med 3:

```
if (a == 10)
   a = a + 2;
else
   a = a - 3;
```

Villkorsuttryck

- Så länge a<5 öka b med 3:

```
while (a < 5){
   a = a + 2;
   b = b + 3;
}
```

Loopar

- Alternativ:

```
while (uttryck) sats;

do sats; while (uttryck);

for (initiering; styruttryck; stegning)
sats;
```

- Exempel 1:

```
int n=1;
while ( n++ <= 10 ){ din kod }
```

- Exempel 2:

```
int n;
for ( n=1; n <= 10; n++ ) { din kod här }
```

LUNDS UNIVERSITET

Funktioner

- All kod paketeras i funktioner.
- Huvudprogrammet:

```
void main(void) {
      b = 5 + my_funktion(3);
}
```

Funktionsanrop

Inparameter

- En funktion deklareras:

```
int my_funktion(x)
      int x
{
      return (x+2);
}
```

Datatyp som returneras

Gör/skapar retur värdet

Funktioner

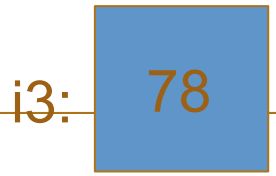
- Exempel: Funktionen kvadrat beräknar: y=x*x kan se ut:

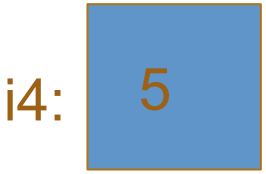
```
int kvadrat(x)
int x
{
      return x*x;
}
```

- Anrop: `y=kvadrat(3);  //y blir 9`
- Värdet av x skickas in och funktionen returnerar kvadraten.

LUNDS UNIVERSITET

Pekare (Intro)





- Deklarationen:

```
int i3, i4
```

- Ger att:

  i3 och i4 är heltalsvaribler

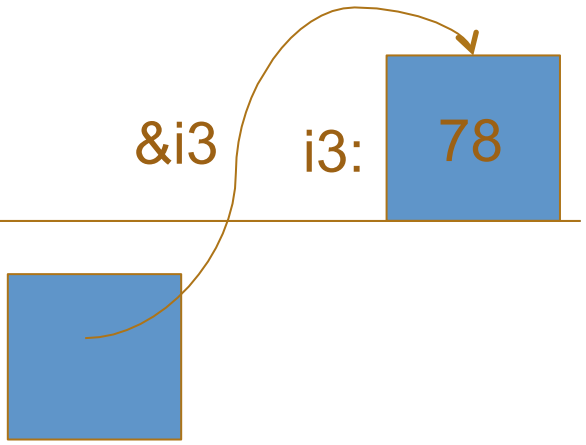
- Exempel:

```
i3=78;  //sätter i3 till 78
i4=5    //sätter i4 till 5
```

- Låt i3 vara lagrat på adress 0
- Låt i4 vara lagrat på adress 1

Adress	Data
0 (i3)	78
1 (i4)	5
2	
3	
4	
5	
6	

Pekare



- Deklarationen:

```
int i3, i4, *ip1, *ip2;
```

- Ger att:

  *ip1 och *ip2 är heltalspekarvariabler
  i3 och i4 är heltalsvariabel

- Exempel:

```
i3=78;    //sätter i3 till 78
ip1=&i3   //sätter ip1 att peka på
            adress där i3 finns
```

  & ger adressen till något

- Notera: ip1 har plats för en pil (adress) och i3 har plats för ett heltal

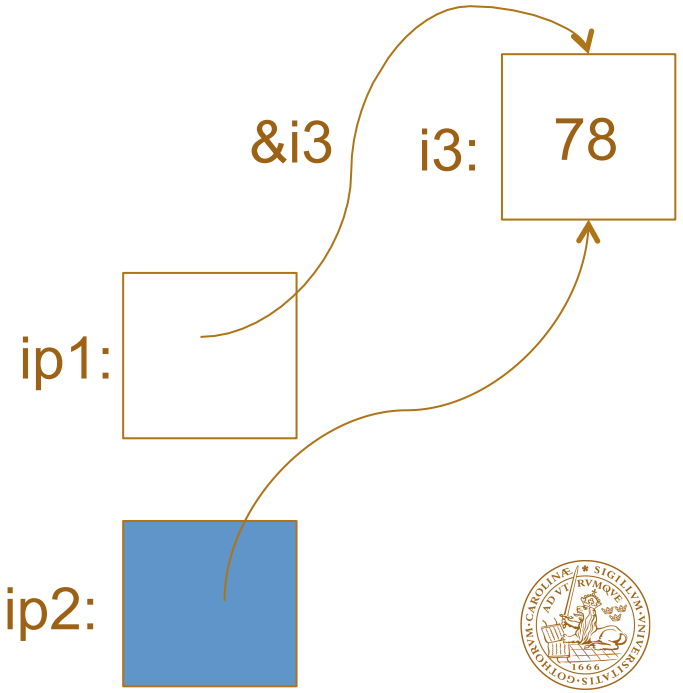
Adress	Data
0 (i3)	78
1 (i4)	5
2 (ip1)	
3 (ip2)	
4	
5	
6	

Pekare

- Exempel: Deklarationen (samma som innan):

```
int *ip1, *ip2, i3, i4
i3=78
ip1=&i3

ip2=ip1  //ip2 sätts att peka
         på samma som ip1
```



LUNDS UNIVERSITET

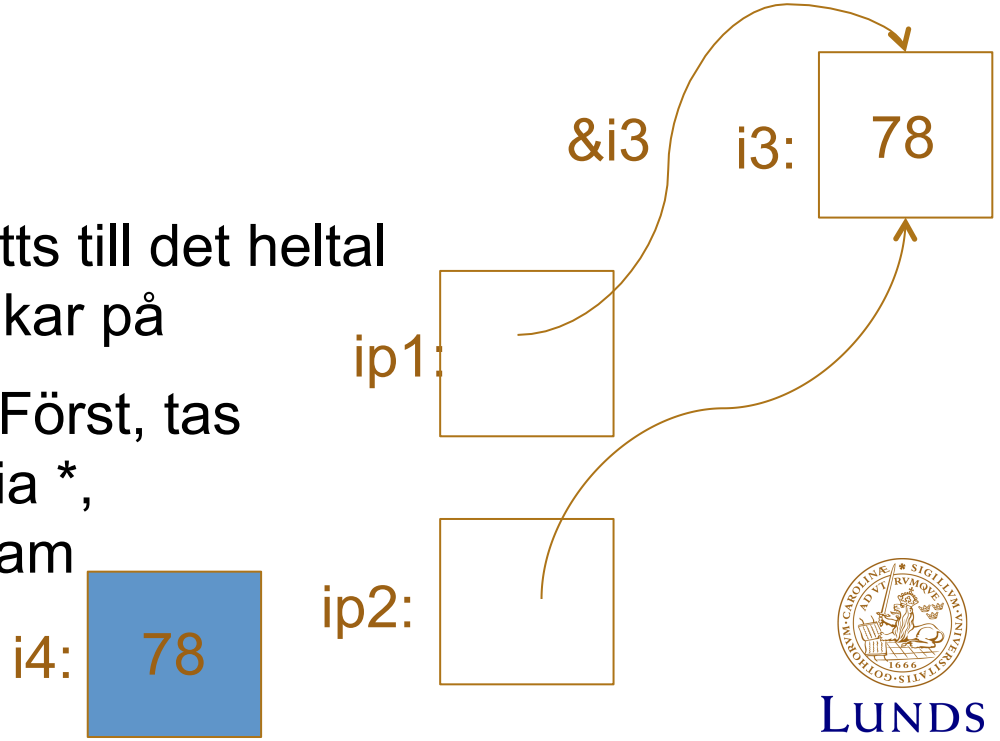
Pekare

- Exempel: Deklarationen (samma som innan):

```
int *ip1, *ip2, i3, i4
i3=78;
ip1=&i3;
ip2=ip1;

i4=*ip1;  //i4 sätts till det heltal
          som ip1 pekar på
```

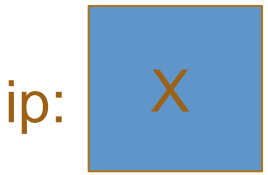
- *ip1 består av två steg. Först, tas pekaren fram. Sedan, via *, tas värdet till pekaren fram



Pekare

- Precis som andra variabler, blir pekare inte automatiskt tilldelade ett värde vid deklaration.
- För att sätta en pekare att peka på ingenting:

  ip=NULL;
- Sätts inte en pekare att peka på ingenting kan den peka på vad som helst – det som råkar ligga på den minnesplatsen.



LUNDS UNIVERSITET

Pekare till funktioner

Adress	Data
0 (a)	10
1 (b)	5
2 xref	
3	
4	
5	
6	

- Om, parametern är en pekare…..
- Exempel:

```
void kvadrat(xref)
int *xref
{
        *xref= *xref * *xref;
}
```

- Anrop: `kvadrat(&x);`
- x är ett pekarvärde. Funktionen tar värdet av pekaren, gör kvadrat och updaterar pekaren

LUNDS UNIVERSITET

Exempel 1:Fråga

- Komplettera koden nedan så att värdet i variabel a och b byter värde.

```
void main (void){
     int a, b;  // deklaration av värde
     a = 10;    // a tilldelas värde
     b = 5;     // b tilldelas värde
     ?          // kod för att a och b
                // byter värde
}
```

Adress	Data
0 (a)	10
1 (b)	5
2	
3	
4	
5	
6	

Exempel 1:Lösning

- Kod där värdet i variabel a och b byter värde.

```
void main (void){
    int a, b, tmp;   // deklaration av värde
    a=10;            // a tilldelas värde
    b=5;             // b tilldelas värde
    tmp=a;           //kod för att a och b
    a=b;             //byter värde
    b=tmp;
}
```

LUNDS UNIVERSITET

Illustration av lösning

```
a=10;
b=5;
```

Adress	Data
0 (a)	10
1 (b)	5
2 (tmp)	
3	
4	
5	
6	

```
tmp=a;
```

Adress	Data
0 (a)	10
1 (b)	5
2 (tmp)	10
3	
4	
5	
6	

```
a=b;
```

Adress	Data
0 (a)	5
1 (b)	5
2 (tmp)	10
3	
4	
5	
6	

```
b=tmp;
```

Adress	Data
0 (a)	5
1 (b)	10
2 (tmp)	10
3	
4	
5	
6	

Exempel 2:Fråga

- Skriv en funktion swap som byter värden på två variabler

```
void main (void){
     int a, b;          // deklaration av värde
     a = 10;            // a tilldelas värde
     b = 5;             // b tilldelas värde
     swap(a,b);
}
```

LUNDS UNIVERSITET

Exempel 2:Fråga+problem

- Skriv en funktion swap som byter värden på två variabler

```
void main (void){
      int a, b;  // deklaration av värde
      a = 10;           // a tilldelas värde
      b = 5;            // b tilldelas värde
      a=swap(a,b);
}

int swap (int c, d){
      int temp;
      temp=c;
      c=d;
      d=temp;
      return ?????
}
```

LUNDS UNIVERSITET

Exempel 2: Lösning

- Skriv en funktion swap som byter värden på två variabler

```
void swap (int *a, *b){
     int temp;  //vanlig variabel
     temp=*a;   // * ger värdet som a pekar på
     *a=*b;
     *b=temp;
 }
```

LUNDS UNIVERSITET

Illustration av lösning

"a=10";
"b=5;"

Adress	Data
0 (a)	
1 (b)	
2 (tmp)	
3	
4	10
5	5
6	

tmp=*a;

(tilldelar tmp det som *a pekar på)

Adress	Data
0 (a)	
1 (b)	
2 (tmp)	10
3	
4	10
5	5
6	

*a=*b;
(tilldelar det a pekar på värdet som finns i b)

Adress	Data
0 (a)	
1 (b)	
2 (tmp)	10
3	
4	5
5	5
6	

*b=tmp;
(tilldelar den plats b pekar på värdet i tmp)

Adress	Data
0 (a)	
1 (b)	
2 (tmp)	10
3	
4	5
5	10
6	

Variablers synlighet

```
#include <stdio.h>
unsigned char n;

void display (unsigned char number){
      static int a;
      int c;
      c=4+a; }


int main(void){
      int b;
      n=5;
      b=10;
      display(b);}
```

Global variabel

Global variabel

Lokal variabel

Lokal variabel

LUNDS UNIVERSITET

Include

- Includeringsbara bibliotek:

  `#include <stdio.h>` standardfunktioner för I/O

LUNDS UNIVERSITET

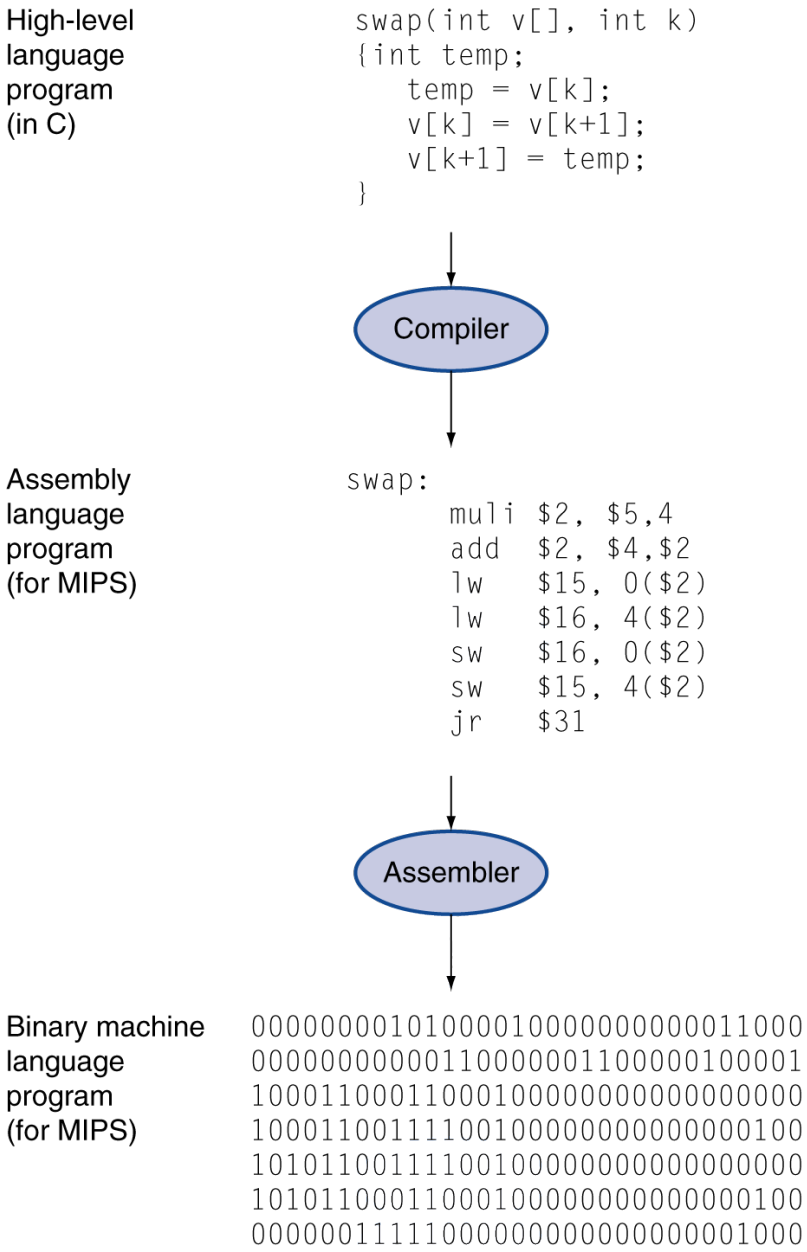
LUNDS UNIVERSITET

Datorteknik

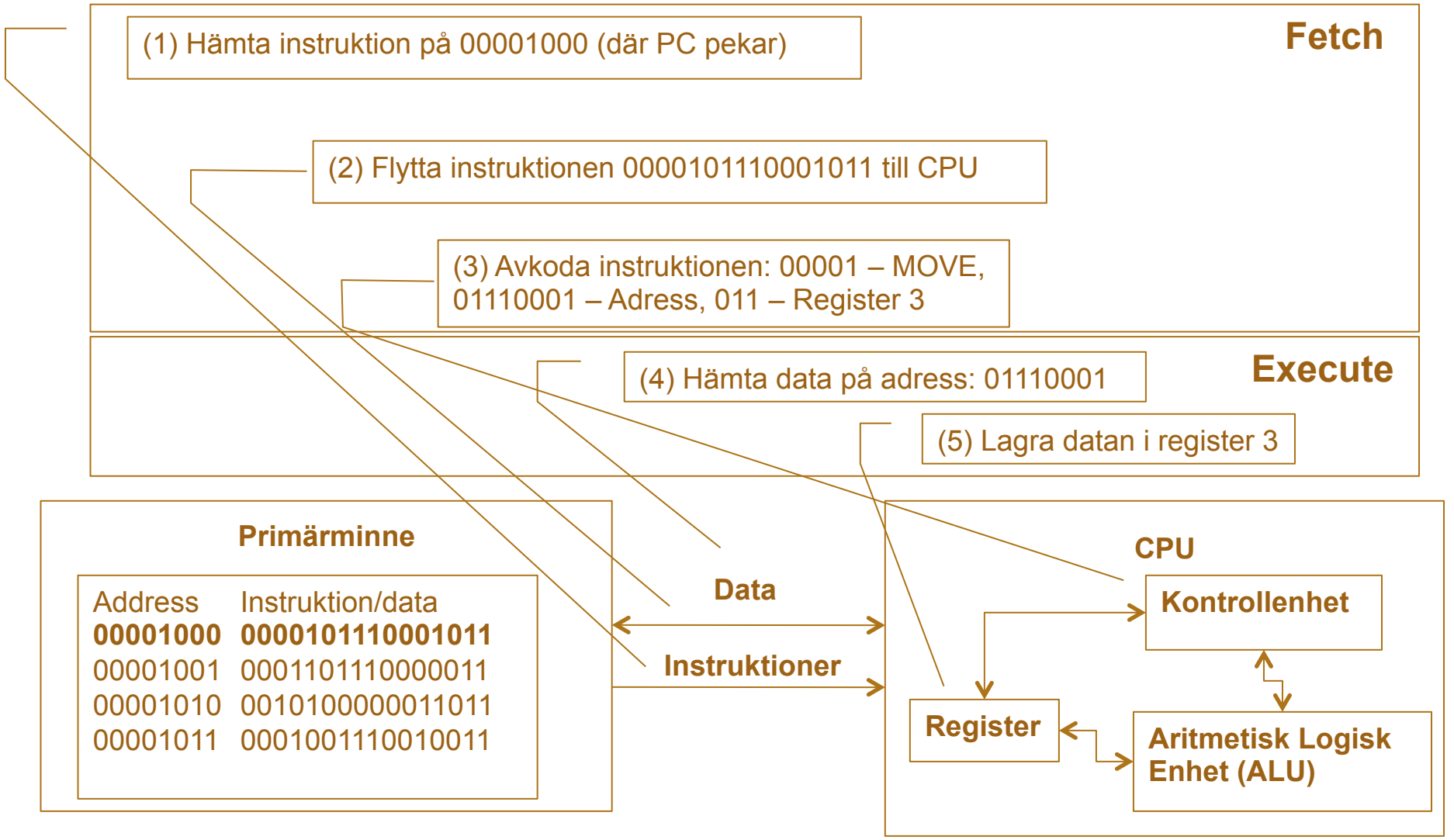
ERIK LARSSON

Program

- Abstraktionsnivå:
  - Högnivåspråk
    - » t ex C, C++
  - Assemblyspråk
    - » t ex ADD R1, R2
  - Maskinspråk
    - » t ex 001101….101



Maskininstruktioner



Maskininstruktioner

- Definitioner:
  - Vad ska göras (operationskod)?
  - Vem är inblandad (source operander)?
  - Vart ska resultatet (destination operand)?
  - Hur fortsätta efter instruktionen?

LUNDS UNIVERSITET

Maskininstruktioner

- Att bestämma:
  - Typ av operander och operationer
  - Antal adresser och adresseringsformat
  - Registeraccess
  - Instruktionsformat
    - » Fixed eller flexibelt

LUNDS UNIVERSITET

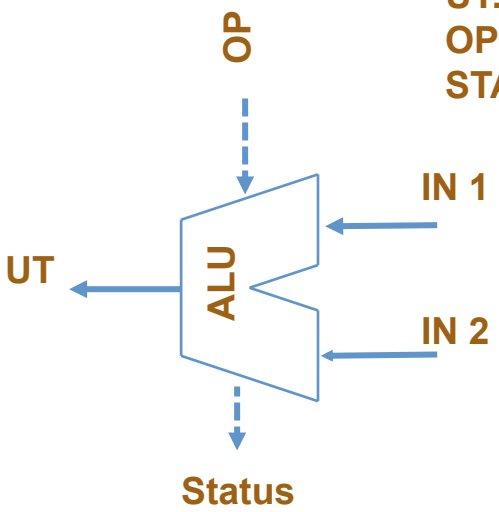
Maskininstruktioner

- Typer av instruktioner:
  - Aritmetiska och logiska (ALU)
  - Dataöverföring
  - Hopp
  - In- och utmatning

LUNDS UNIVERSITET

Aritmetiska operationer

IN 1, IN 2: Ingångar
UT: utgång (resultat)
OP: operation (ADD, SUB, AND, ….
STATUS: Overflow, Zero, ….



LUNDS UNIVERSITET

Aritmetiska operationer

- Addition (+) och subtraktion (-)
  - Två källor (sources) och en destination (destination)

```
add a, b, c // a = b + c
```

- Alla aritmetiska funktioner följer detta mönster
- Design regel:
  - Regelbundenhet gör implementation enklare
  - Enkelhet ökar möjligheten till högre prestanda till lägre kostnad

LUNDS UNIVERSITET

Aritmetiska operationer

- C kod:

```
f = ( g + h ) - ( i + j );
```

- Kompileras till assemblykod (MIPS):

```
add t0, g, h        # temp t0 = g + h
add t1, i, j        # temp t1 = i + j
sub f, t0, t1       # f = t0 - t1
```

LUNDS UNIVERSITET

Adressering

- Immediate adressering

```
ADD R4,#3       //R4<-R4+3
```

Operanden finns direkt i instruktionen

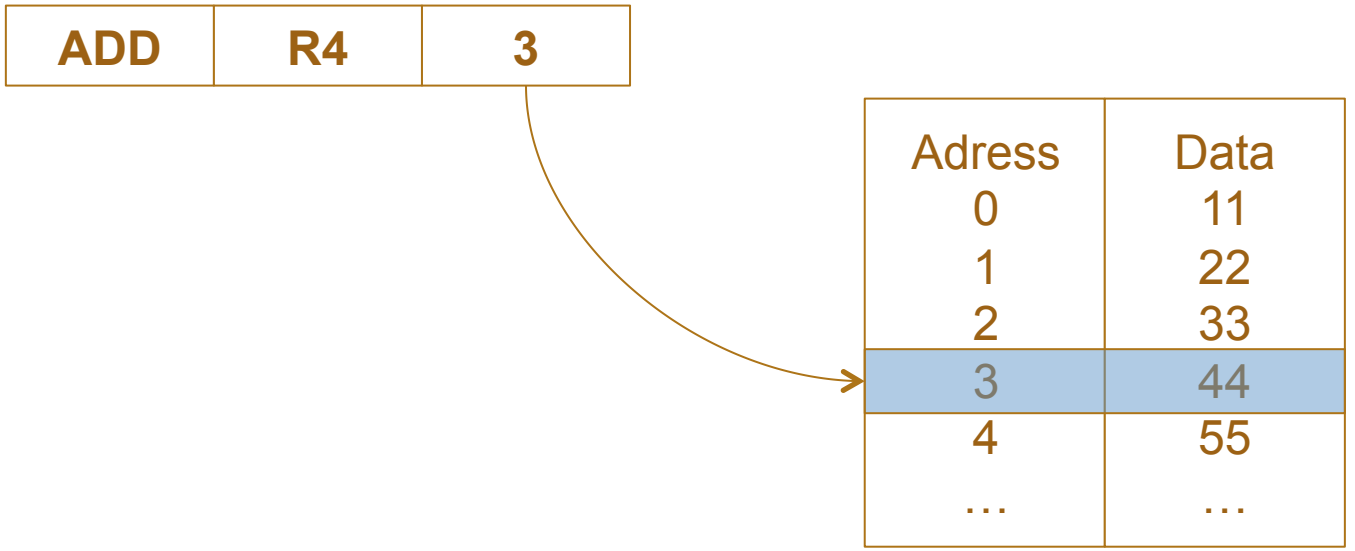
ADD	R4	3

Adressering

- Direkt adressering

```
ADD R4, 3      //R4<-R4+[3]
```

Adressen till operanden ligger i instruktionen



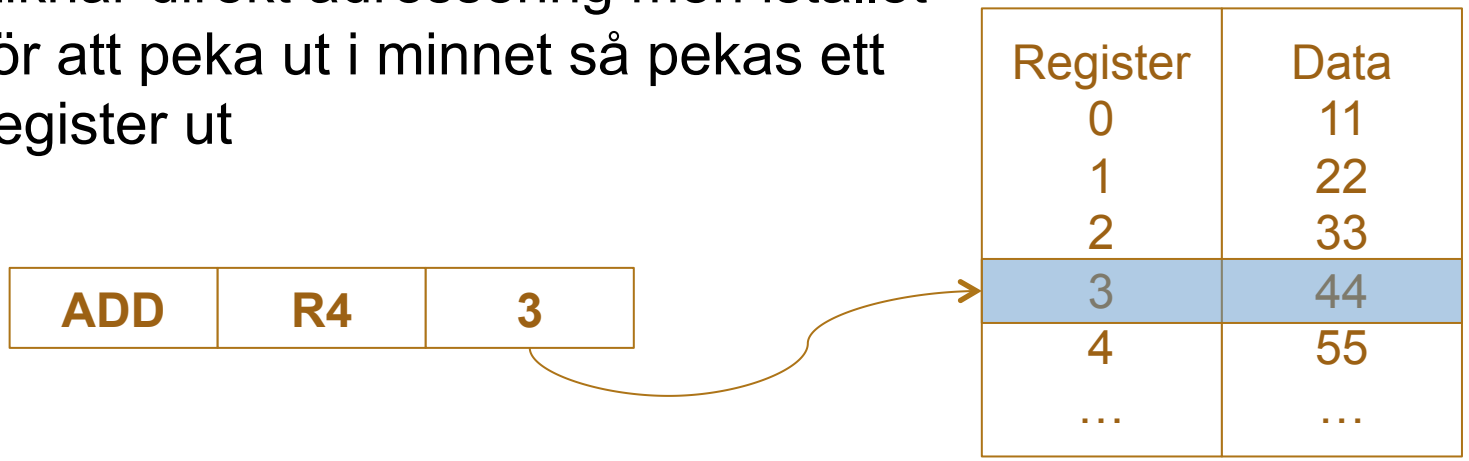
LUNDS UNIVERSITET

Adressering

- Register adressering

```
ADD R4,R3      //R4<-R4+R3
```

Liknar direkt adressering men istället för att peka ut i minnet så pekas ett register ut

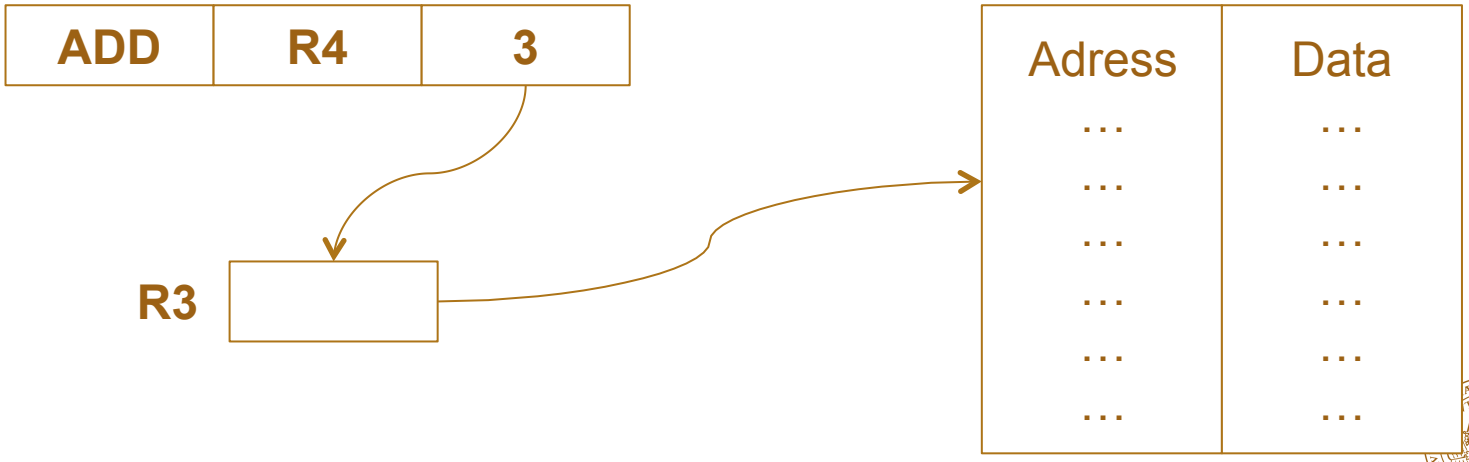


Adressering

- Register indirect

```
ADD R4,(R3)    R4<-R4+[R3]
```

Liknar indirekt adressering men instruktion pekar ut register istället för minnesplats



LUNDS UNIVERSITET

Hoppinstruktioner

Exempel 1

```
a=a+b;
c=a-b;
d=a*f;
```

Instruktionerna utförs i ordning

Instruktionen här görs flera gånger

Exempel 2

```
a=0;
while (a<10)
      a=a+1;
if (b==5)
   c=10;
else
   c=0;
d=a*f;
```

Beroende på villkor görs olika saker (olika instruktioner exekveras)

LUNDS UNIVERSITET

Hoppinstruktion

- Nästa instruktion är normalt nästa i programmet.

  ```
  PC=PC+1;
  ```

  Programräknaren (program counter) räknas upp och nästa instruktion hämtas

- För att göra hopp till annan del av kod, ladda programräknaren med nytt värde

  ```
  PC=dit hopp ska sek
  ```

- Två typer av hopp
  - Ovillkorliga hopp
  - Villkorliga hopp

LUNDS UNIVERSITET

Ovillkorliga hopp

- Exempel:

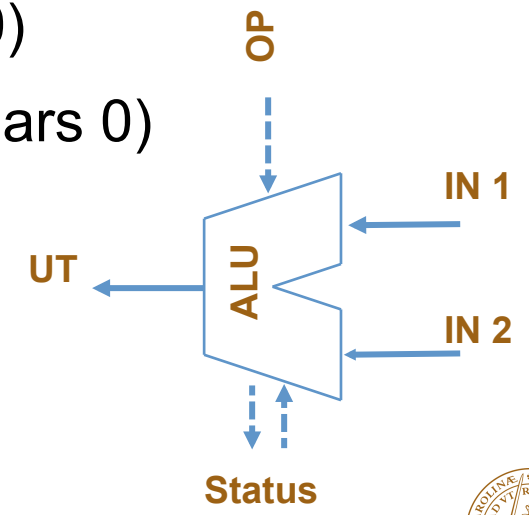
Programräknaren får nytt värde och hämtar nästa instruktion på nytt ställe

Adress	Instruktion	Kommentar
.....	.....	
01011	Instruktion 29	//Instruktion 29,PC=PC+1
01100	BR 10101	// PC = 10101, PC=PC+1
01101	Instruktion 31	//Instruktion 31, PC=PC+1
.....	.....	
10101	Instruktion 89	//Instruktion 89, PC=PC+1
.....	.....	

LUNDS UNIVERSITET

Villkorliga hopp

- Villkor bestäms av flaggor i statusregister
- De vanligaste flaggorna är:
  - N: 1 om resultatet är negativt (annars 0)
  - Z: 1 om resultatet är noll (annars 0)
  - V: 1 om aritmetiskt “overflow” (annars 0)
  - C: 1 om “carry” (annars 0)



LUNDS UNIVERSITET

Villkorliga hopp

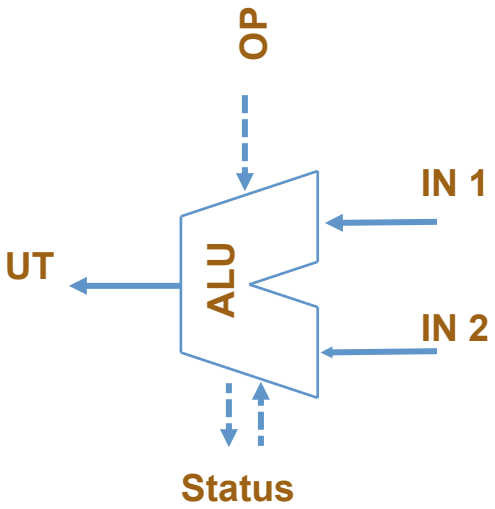
- Exempel:

```
SUB R1, #1        // R1=R1-1
BEZ TARGET        // IF Z=1 THEN PC=TARGET
```

Kan påverka statusflagga

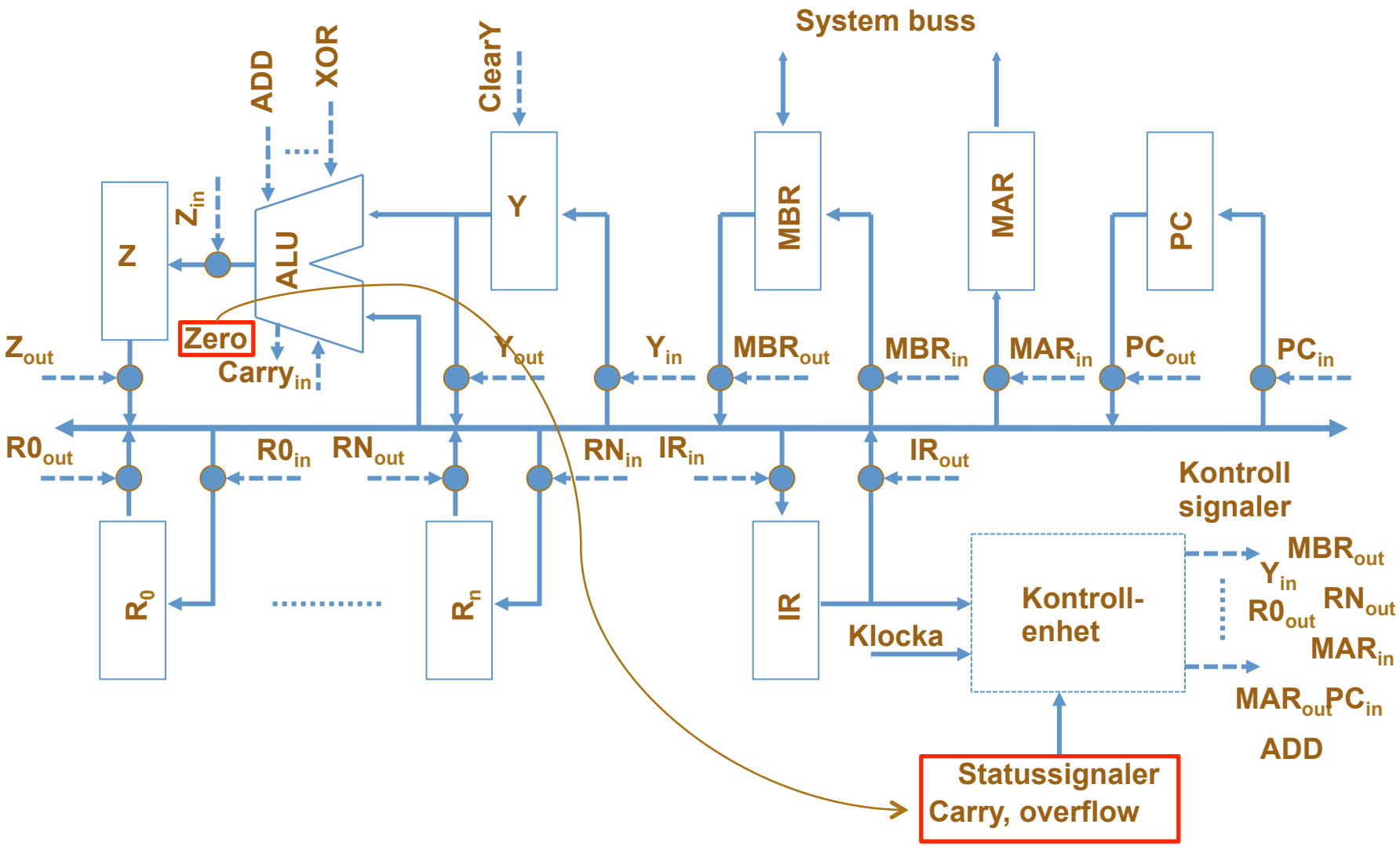
Kollar status-flagga

Statusregister



LUNDS UNIVERSITET

Villkorliga hopp



Subrutiner och funktioner

Program:

```
void main(void){
int a, b, c;
  a=5;
  b=6;
  c=my_add(a,b);
}


int my_add(int x, int y)
{
  return x+y;
}
```

Exekvering:

a=5 //instruktion 1
b=6 //instruktion 2

funktionsanrop
+parameteröverföring

c=my_add(a,b); instruktion 11

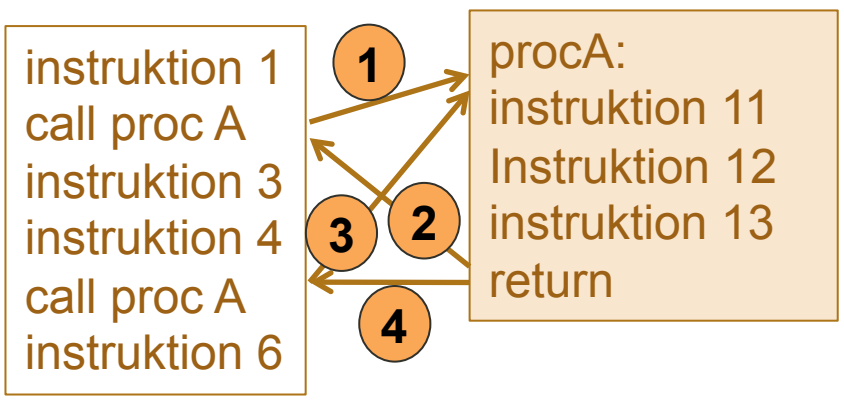
återhopp+parameteröverföring

LUNDS UNIVERSITET

Subrutiner och funktioner

- Problem
  - Hopp till subrutin
  - Återhopp från subrutin
  - Parameteröverföring
  - Nästlade subrutinanrop

	Exekvering:	
	instruktion 1	
1	instruktion 11	//Proc A
	Instruktion 12	//Proc A
	Instruktion 13	//Proc A
2	instruktion 3	
	instruktion 4	
3	instruktion 11	//Proc A
	Instruktion 12	//Proc A
	instruktion 13	//Proc A
4	instruktion 6	

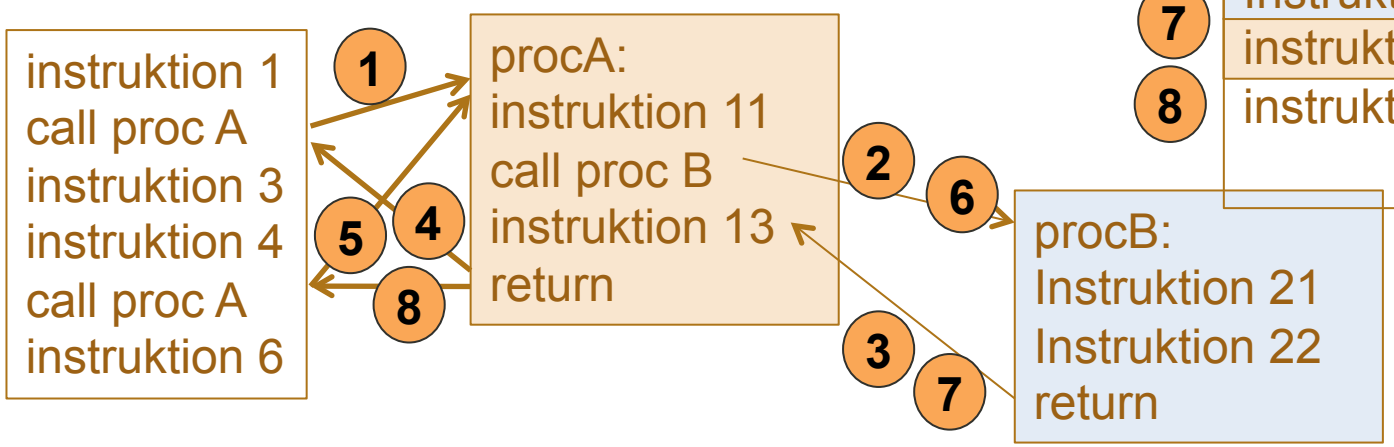


LUNDS UNIVERSITET

Subrutiner och funktioner

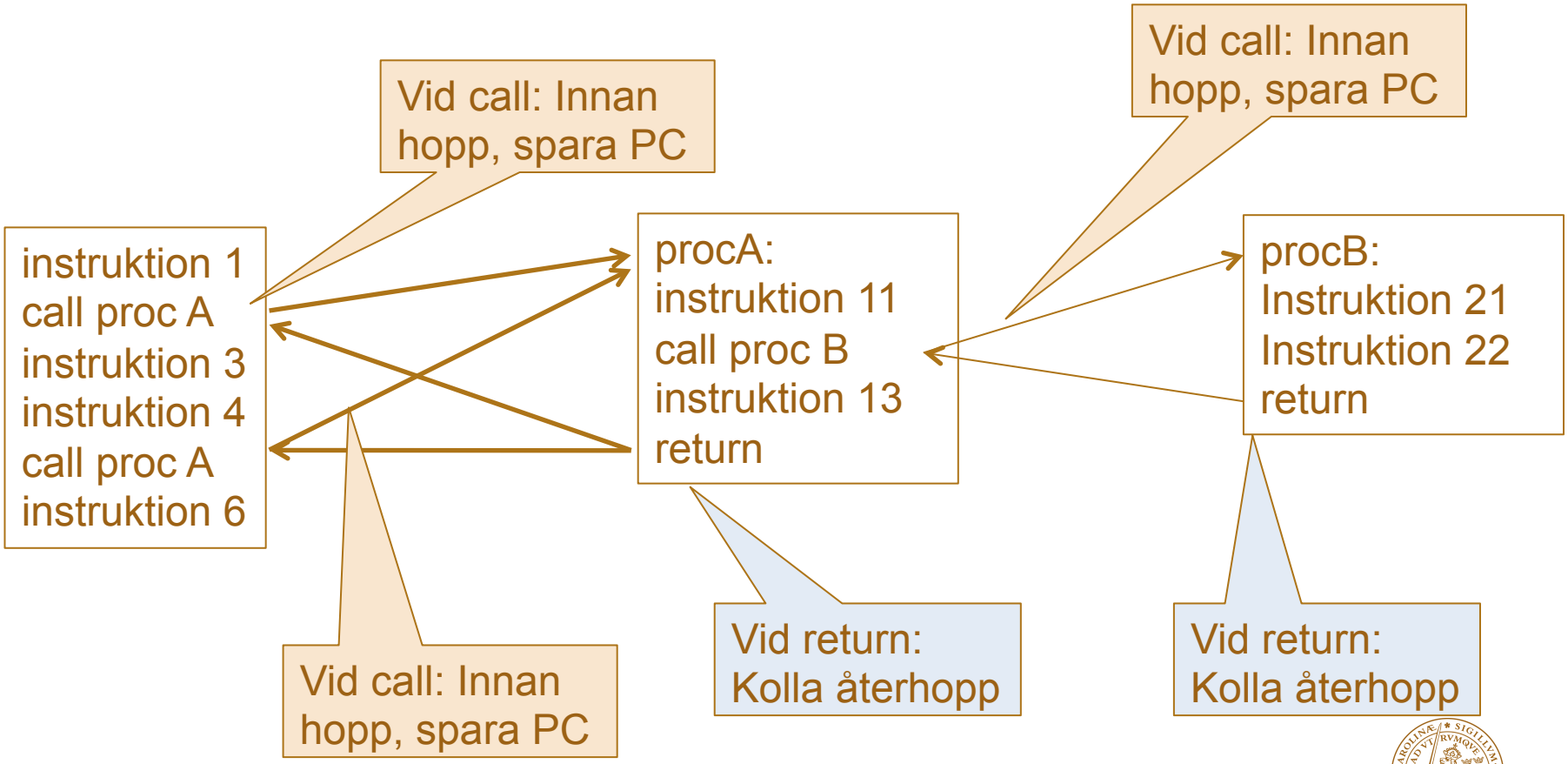
- Problem
  - Hopp till subrutin
  - Återhopp från subrutin
  - Parameteröverföring
  - Nästlade subrutinanrop

	Exekvering:	
	instruktion 1	
1	instruktion 11	//Proc A
2	Instruktion 21	//Proc B
	Instruktion 22	
3	instruktion 13	//Proc A
4	instruktion 3	
	instruktion 4	
5	instruktion 11	//Proc A
6	Instruktion 21	//Proc B
	Instruktion 22	
7	instruktion 13	//Proc A
8	instruktion 6	



LUNDS UNIVERSITET

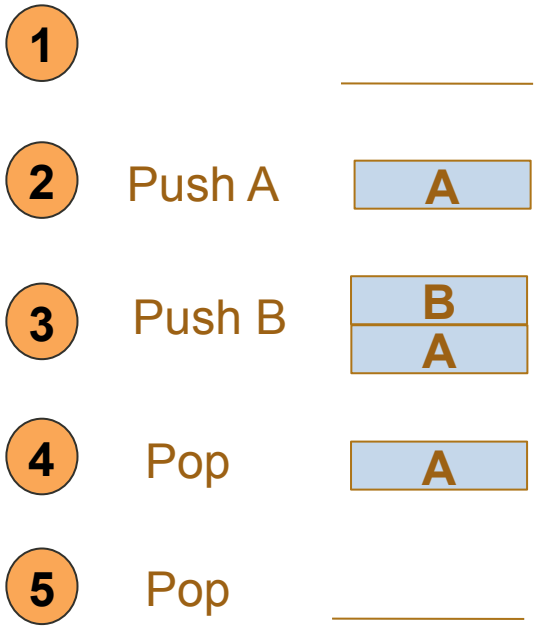
Subrutiner och funktioner

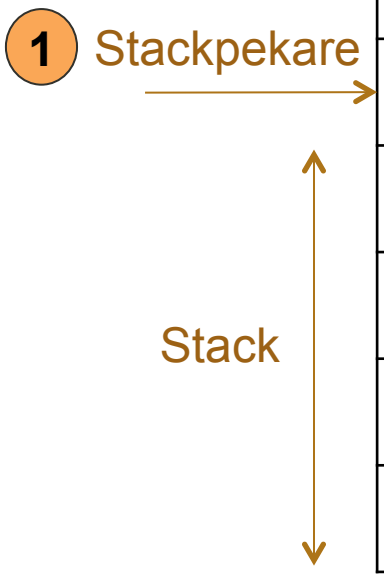


LUNDS UNIVERSITET

Stack

- Två operationer: push och pop





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

Stack

- Två operationer: push och pop

1

2 Push A

3 Push B

4 Pop

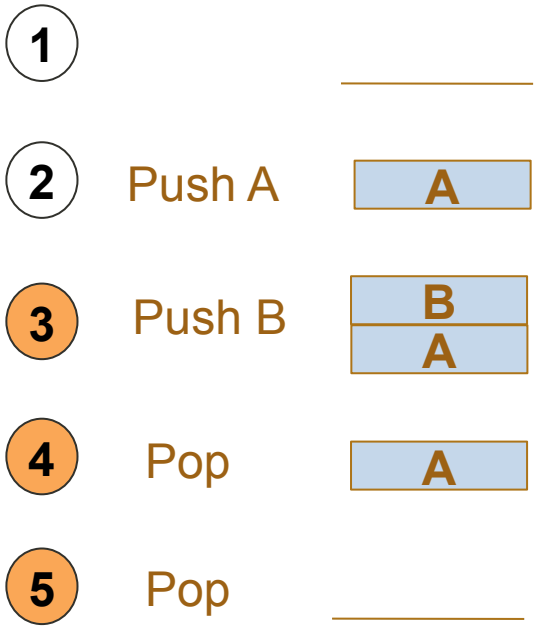
5 Pop

2 Stackpekare

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

Stack

- Två operationer: push och pop

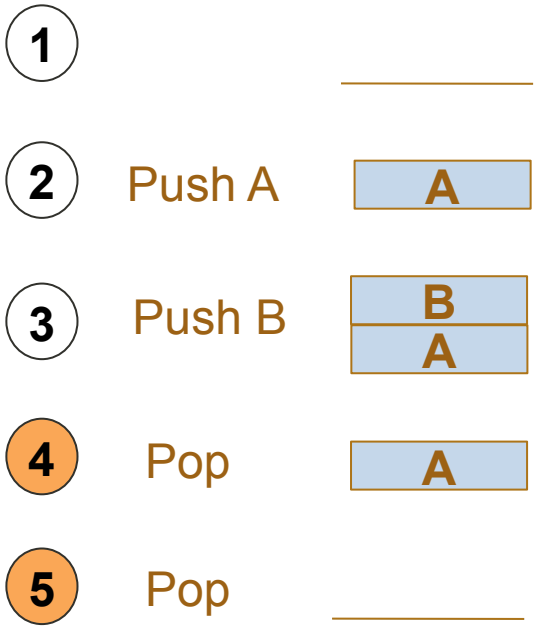


Adress	Byte	Data
0	0	1111 0000
1	1	1010 0101
2	2	1100 0011
3	3	0011 0011
4	4	A
5	5	B
6	6	1111 0000
7	7	1010 1010

3 Stackpekare

Stack

- Två operationer: push och pop

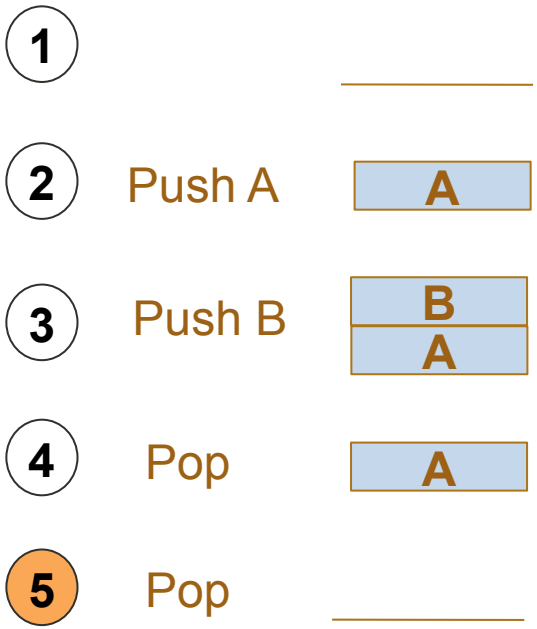


4 Stackpekare

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

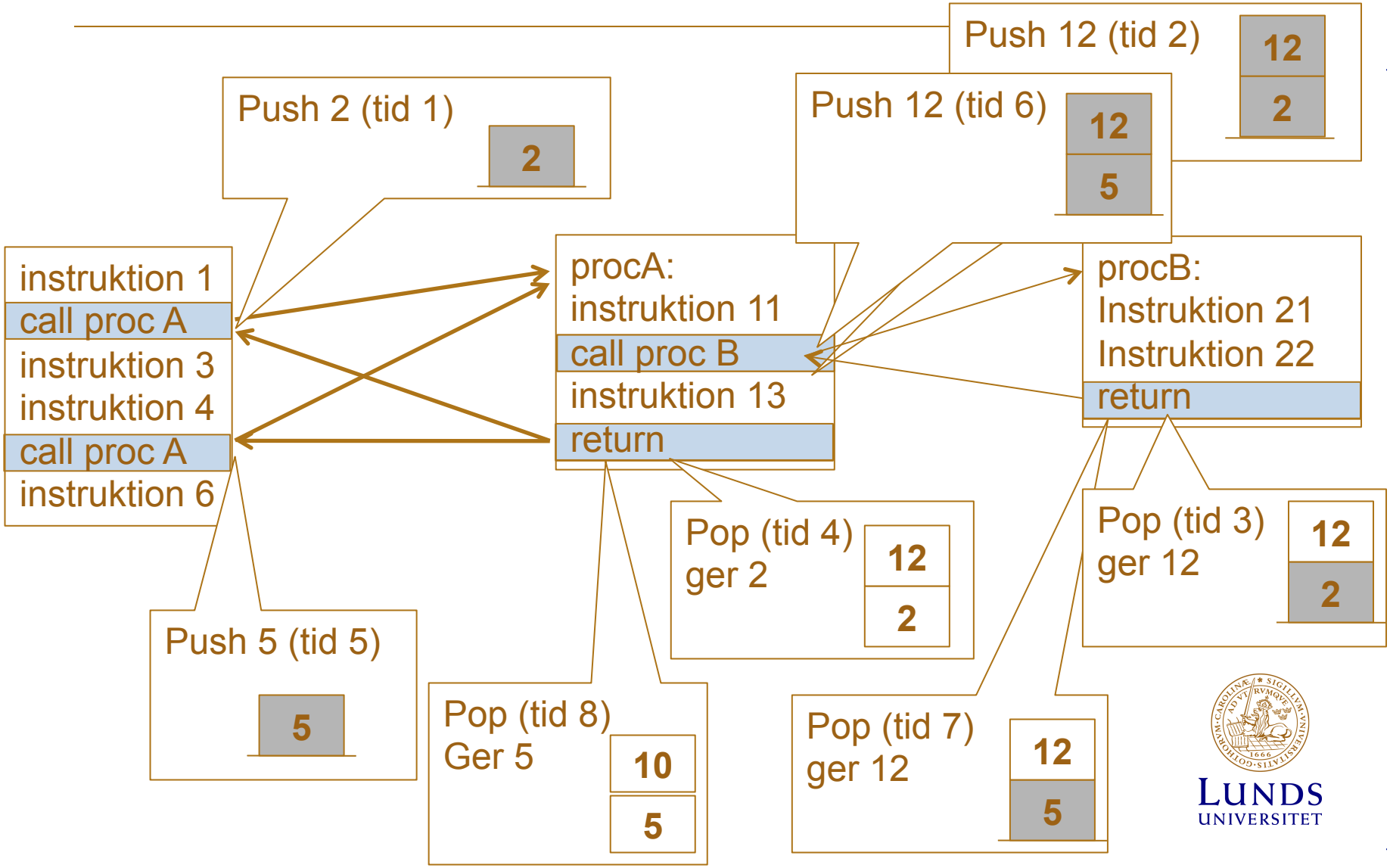
Stack

- Två operationer: push och pop



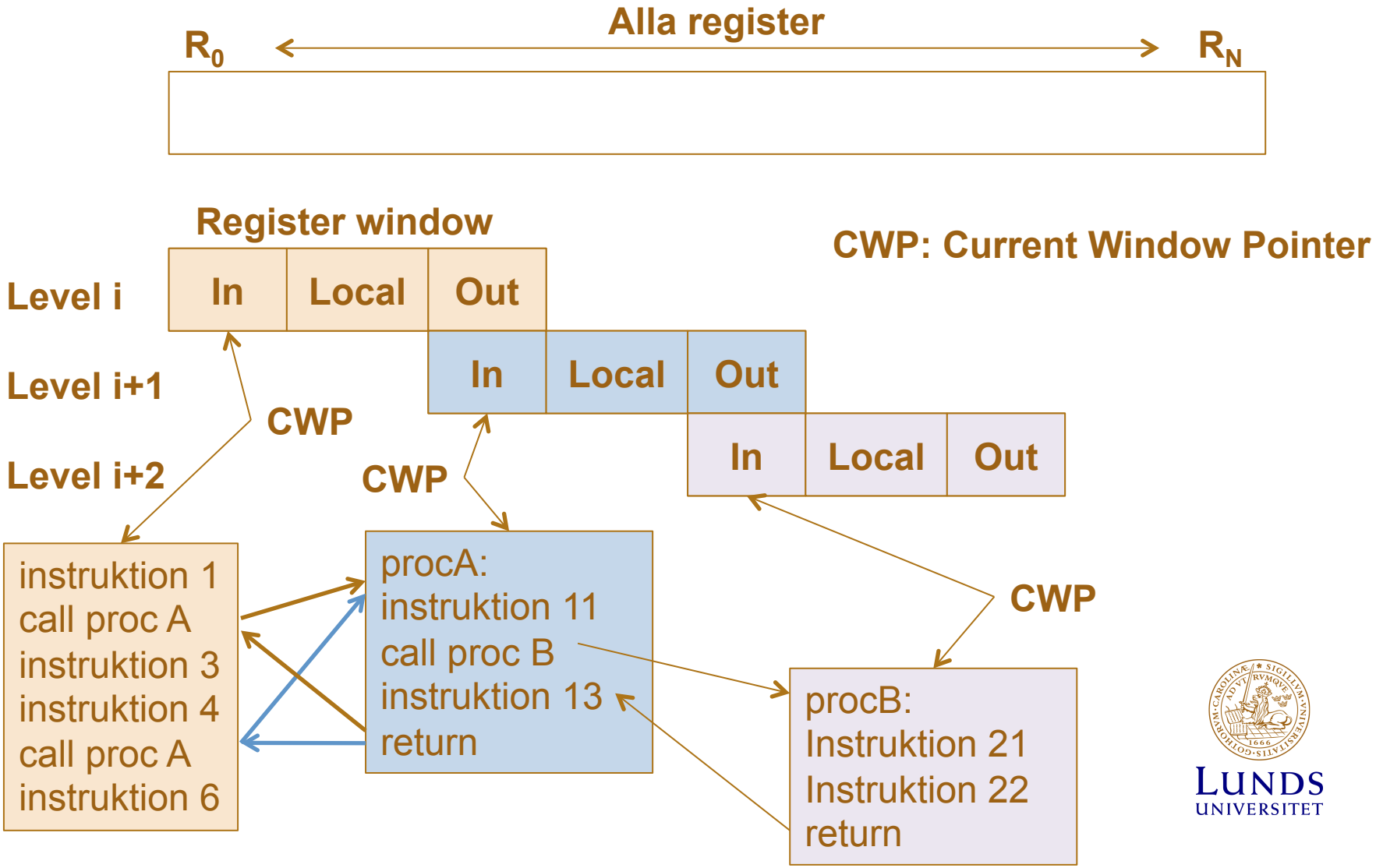
Adress	Byte	Data
0	0	1111 0000
1	1	1010 0101
2	2	1100 0011
3	3	0011 0011
4	4	A
5	5	B
6	6	1111 0000
7	7	1010 1010

Stack för subrutiner och funktioner



LUNDS UNIVERSITET

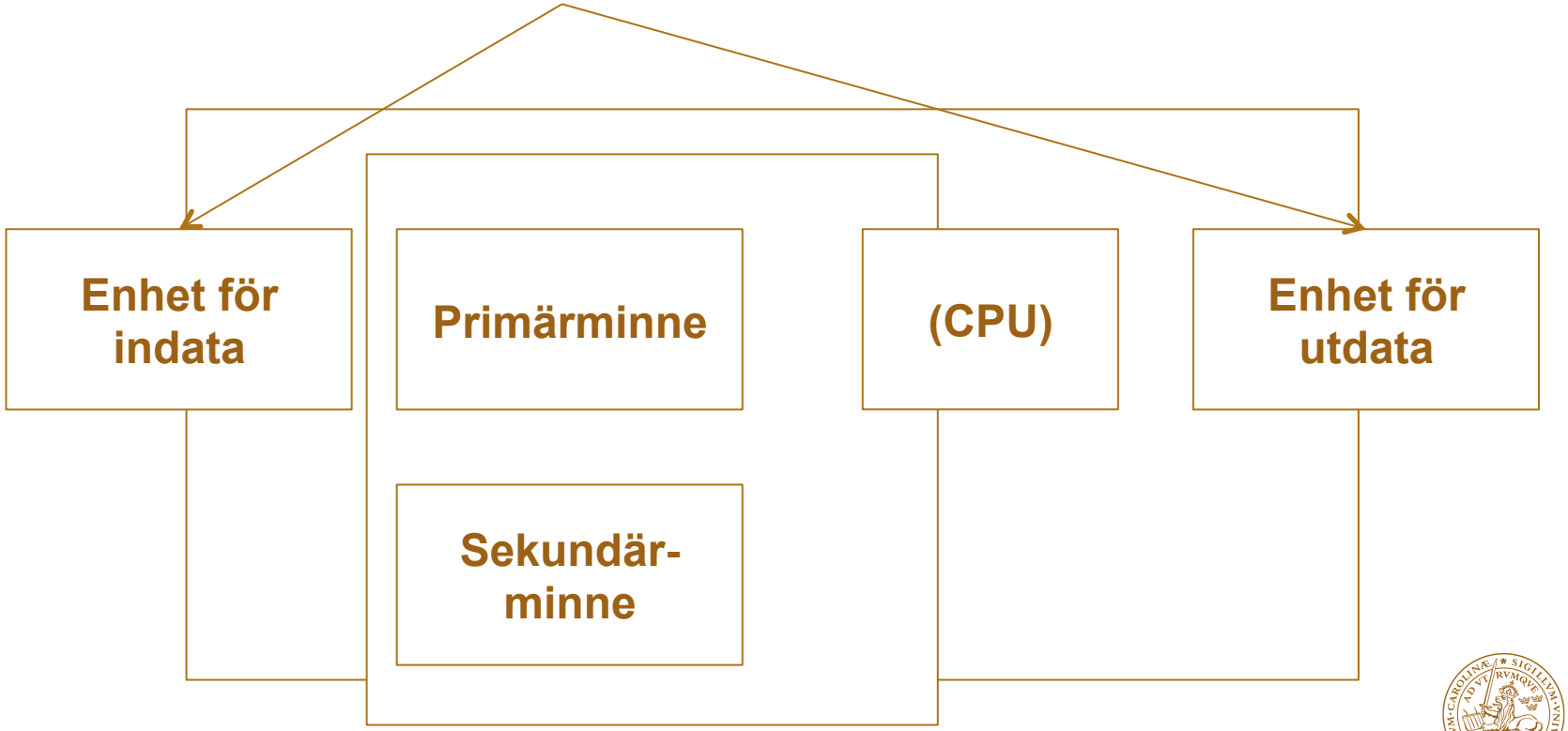
Register för subrutiner och funktioner



LUNDS UNIVERSITET

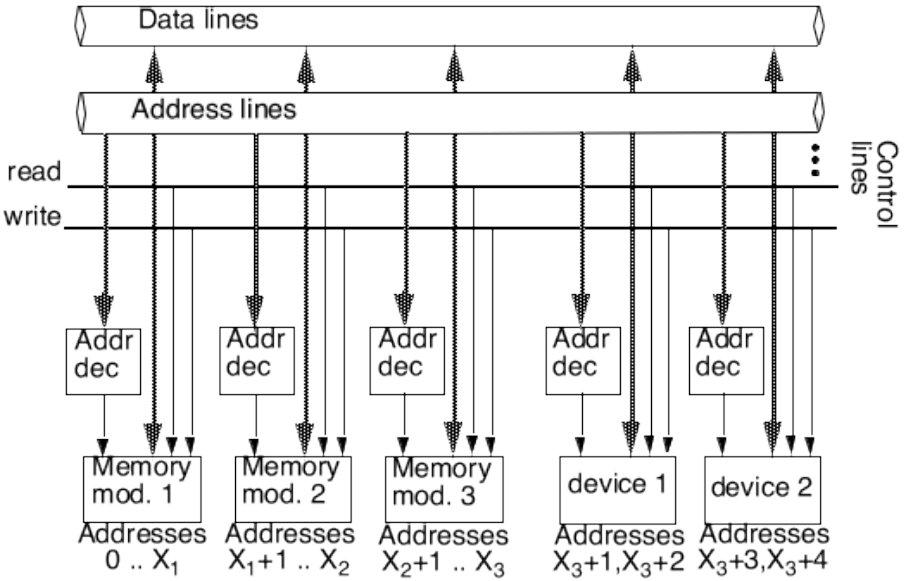
In- och utmatning

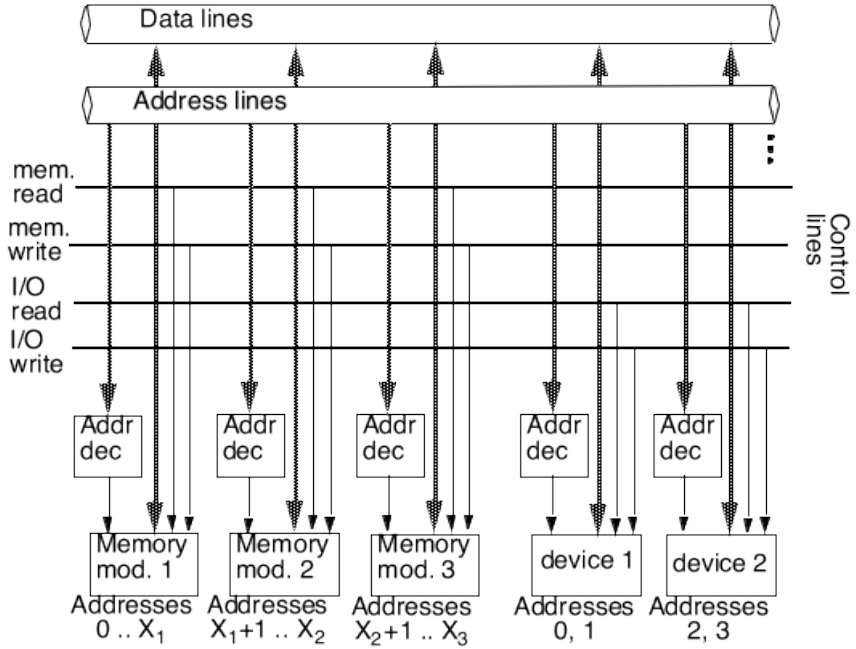
- Läsning och skrivning:



LUNDS UNIVERSITET

Minnesmappad och isolerad I/O



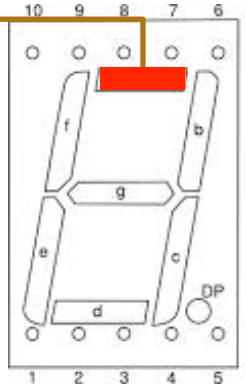


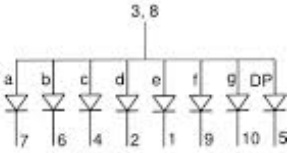
Exempel på minnesmappad I/O

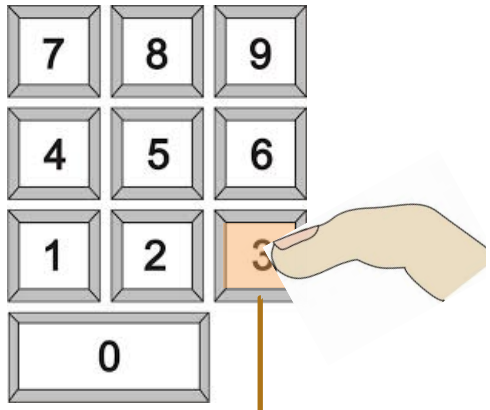
LOAD R1, 7 //Ladda R1 med värde på adress 7

STORE R1, 6 //Lagra värdet i R1 på adressplats 6

Adress	Instruktion/ Data
0	1111 0000
1	1010 0101
2	1100 0011
3	0011 0011
4	1111 1111
5	0000 1111
6	0010 0000
7	0010 0000







LUNDS UNIVERSITET

Maskininstruktioner

- Aritmetiska och logiska (ALU)
- Dataöverföring (adressering)
- Hopp och subrutiner
- Inmatning/utmating (Input/Output (I/O))

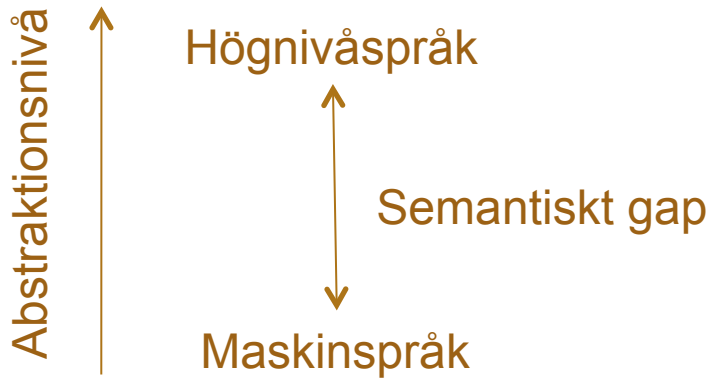
LUNDS UNIVERSITET

Instruktionsformat

- Hur definiera fälten i instruktioner?

Opcode	Operand1	Operand2

- Hur ska högnivåspråk kompileras och exekveras effektivt?



LUNDS UNIVERSITET

Antal adresser i instruktion

- För att beräkna: X = (A+B)*C
- 4-adress instruktioner:
  - Op A1, A2, A3, A4 //A1 <- [A2] op [A3], nästa instruktion = A4

```
I1: ADD X, A, B, I2
I2: MUL X, X, C, I3
I3:
```

- 3-adress instruktioner:
  - Op A1, A2, A3 //A1 <- [A2] op [A3]

```
I1: ADD X, A, B
I2: MUL X, X, C
I3:
```

LUNDS UNIVERSITET

Antal adresser i instruktion

- För att beräkna: X = (A+B)*C
- 2-adress instruktioner:
  - Op A1, A2 //A1 <- [A1] op [A2]

```
MOVE A, X
ADD X, B
MUL X, C
```

- 1-adress instruktioner:
  - Op A1 //Acc <- [Acc] op [A1]

```
LOAD A
ADD B
MUL C
STORE X
```

LUNDS UNIVERSITET

Antal adresser i instruktion

- Vanligast med 2 och 3 adresser i instruktioner
- 4 adress instruktioner är opraktiskt.
  - Nästa instruktion antas vara nästa instruktion. PC räknas upp “av sig själv”. Måste ha hopp-instruktioner
- 1 adress instruktion är ganska begränsande
- Exempel, antag 32 register (5 bitar behövs):
  - 3-adress format: 15 bitar
  - 2-adress format: 10 bitar
  - 1-adress format: 5 bitar

- Få adresser, få instruktioner förenklar processorn men gör den mer primitiv

LUNDS UNIVERSITET

Intel x86 ISA

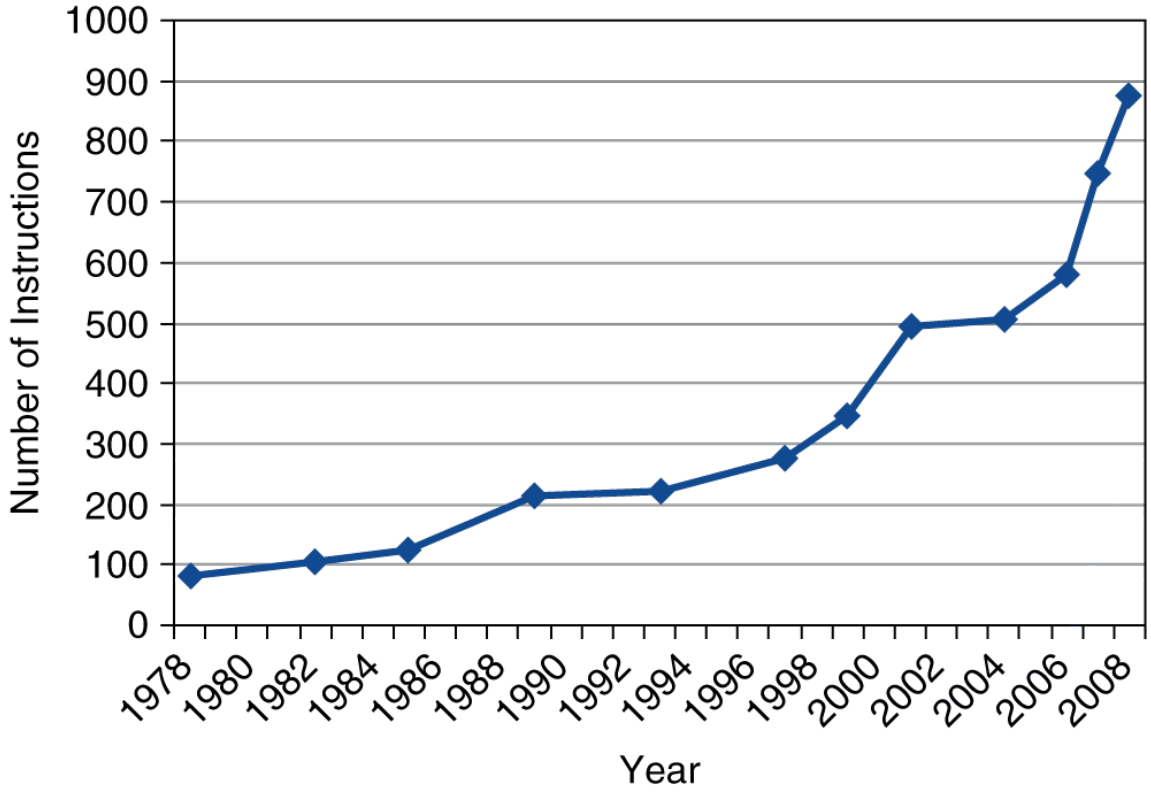
AMD och Intel bygger för x86 arkitektur men organiserar på olika sätt

- Utveckling med backward compatibility
  - 8080 (1974): 8-bit microprocessor
    - » Accumulator, plus 3 index-register pairs
  - 8086 (1978): 16-bit extension to 8080
    - » Complex instruction set (CISC)
  - 8087 (1980): floating-point coprocessor
    - » Adds FP instructions and register stack
  - 80286 (1982): 24-bit addresses, MMU
    - » Segmented memory mapping and protection
  - 80386 (1985): 32-bit extension (now IA-32)
    - » Additional addressing modes and operations
    - » Paged memory mapping as well as segments

LUNDS UNIVERSITET

Intel x86 ISA

- Bakåtkompabilitet (Backward compatibility)



LUNDS UNIVERSITET

LUNDS
UNIVERSITET