Fält, pekare och portar

Ur innehållet

- Fält
- Pekarvariabler, pekarkonstanter och portar
- Pekarreferenser

Läsanvisningar:

- Arbetsbok kap 2
- Quick-guide, instruktionslistan

Målsättningar:

- Använd printf för testutskrifter på värddator
- Generera ARM/Thumb assemblerkod för fältreferenser
- Konstruera pekarreferenser för portar

Vad är ett "fält"?

Ett fält ("array") är en datastruktur som mångfaldigar förekomsten av element med samma typ.

Exempel:
Deklarationen:

```
int   intvec[N];
```

skapar ett fält med N st. element av typen `int`.

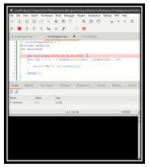
Innehållet är initialt odefinierat men ett fält kan också deklarares och initieras samtidigt:

```
int   initintvec[]={10,20,30,40,50};
```

skapar ett fält med 5 element och initierar samtidigt fältets element .

Undersök ett fält:

```
/* PrintIntegerArray.c */
#include <stdio.h>
int main(void)
{
  int   initintvec[]={10,20,30,40,50};
  for( int i = 0; i < sizeof(initintvec) / sizeof(int); i++)
  {
    printf("%d ", initintvec[i]);
  }
  return 0;
}
```



Fält och textsträngar

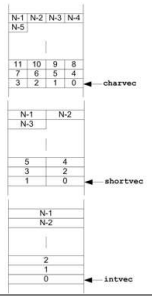
En textsträng är ett fält av `char`, som avslutas med 0 ('\0')

Exempel:

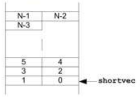
```
#include <stdio.h>

int main(void)
{
  char animal1[] = "Monkey";
  char animal2[] = {'M','o','n','k','e','y',0};
  char animal3[] = {77,111,110,107,101,121,0};
  printf("%s\n", animal1);
  printf("%s\n", animal2);
  printf("%s\n", animal3);
  return 0;
}
```

Fält – "vektorer" – N element, indexeras 0..N-1

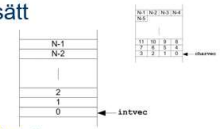
```
char  charvec[N];
sizeof (charvec) = N bytes


short shortvec[N];
sizeof(shortvec)= N*2 bytes


int   intvec[N];
sizeof (intvec) = N*4 bytes
```



Adressberäkning och adresseringssätt




```
int   i;
short shortvec[N];
shortvec[i];

=shortvec + i*sizeof(short)

LDR   R0,=shortvec
LDR   R1,i
LSL   R1,R1,#1    @R1←R1×2
LDRSH R0,[R0,R1]
```




```
int  i;
int  intvec[N];
intvec[i];

=intvec + i*sizeof(int)

LDR   R0,=intvec
LDR   R1,i
LSL   R1,R1,#2    @R1←R1×4
LDR   R0,[R0,R1]
```

Flerdimensionella fält "matriser" – N × M element, indexeras [0..N-1] [0..M-1]

Exempel:

```
char  mc [N][M];
int   i1,i2;
```

Referens: `mc[i1][i2]`
Minnesadress: `=mc + ((i1*M)+i2)*sizeof(char)`

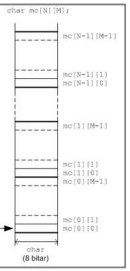
Kompilatorn ersätter multiplikationer med skift/add kombinationer eftersom `mul`-instruktionen tar betydligt fler klockcykler att utföra.

M = $2^x + y$

Exempelvis då M=10:

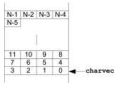
10 = 2^3+2

i1*10 =
i1*(2^3+2) =
i1<<3 + i1 + i1



Adressberäkning och adresseringssätt

```
const  k; (0 ≤ k ≤ N-1)
int    i;
char   charvec[N];
```



Exempel:

```
charvec[5];

LDR   R0,=charvec
LDRB  R0,[R0,#5]
```

Adress: `charvec+k`
Konstant index:

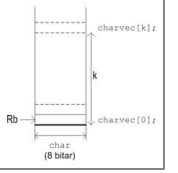
```
LDRB  Rd,[Rb,#k]
@ Rd←M(Rb+k)
```

Exempel:

```
charvec[i];

LDR   R0,=charvec
LDR   R1,i
LDRB  R0,[R0,R1]
```

Adress: `charvec+i`
Register index:

```
LDRB  Rd,[Rb,Ri]
@ Rd←M(Rb+Ri)
```



Pekare

En pekare är en datatyp för en minnesadress.
En pekarvariabel innehåller minnesadressen till en variabel, port etc.
snarare än variabelns (portens) värde.

Exempel:
En *pekare* till värdet `2000`,
dvs. värdets position som är en minnesadress

0x20046670	0x20030104
...	...
0x20030108	...
0x20030104	2000
0x20030100	...
...	...
0x00000001	...
0x00000000	...

Pekaroperatorer

1. Pekarens värde är en adress (&)
2. Pekarens typ anger hur vi ska tolka bitarna hos innehållet på adressen
3. '*' (stjärna) används för att referera *innehållet på adressen* (dereferera) .

Exempel:

```
int salaryLevel1 = 1000;
int salaryLevel2 = 2000;
int salaryLevel3 = 3000;

int* mySalary = &salaryLevel2;
```

```
&mySalary är 0x20046670
 mySalary är 0x20030104
*mySalary är 2000
```

0x20046670	0x20030108	mySalary
...	...	
0x20030108	3000	salaryLevel3
0x20030104	2000	salaryLevel2
0x20030100	1000	salaryLevel1
...	...	
0x00000001		
0x00000000		

Vad betyder stjärnan..? "*"

I en deklaration anger stjärnan en pekartyp

Exempel:

```
char  *cp;
float *fp;
unsigned int *ip;
void f(int *ip);
char *g(void);
```

I ett uttryck, dvs. som operator anger stjärnan en dereferens.

Exempel:

```
char a = *cp;
*cp = 'b';
printf( "%X\n", *ip );
a = 5 * *cp;
```

Pekare: dereferens

- När vi derefererar en pekare får vi objektet som finns på pekarens adress.
- *Antal bytes* beror då av pekarens typ
- *Tolkningen* av bitarna beror av pekarens typ

Exempel:

```
char buffer[] = {0xFF,1,0,4};

unsigned char * ucp = &buffer[0];
signed   char * scp = &buffer[0];
...
int b;
   b = (int) *ucp;
   ...
   b = (int) *scp;
```

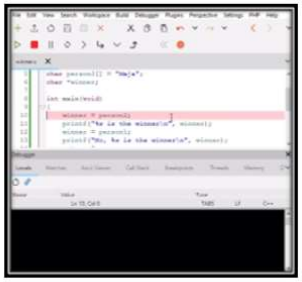
Varför behöver vi pekare?

En pekarvariabel innehåller minnesadressen till en variabel, port etc. snarare än variabelns (portens) värde.

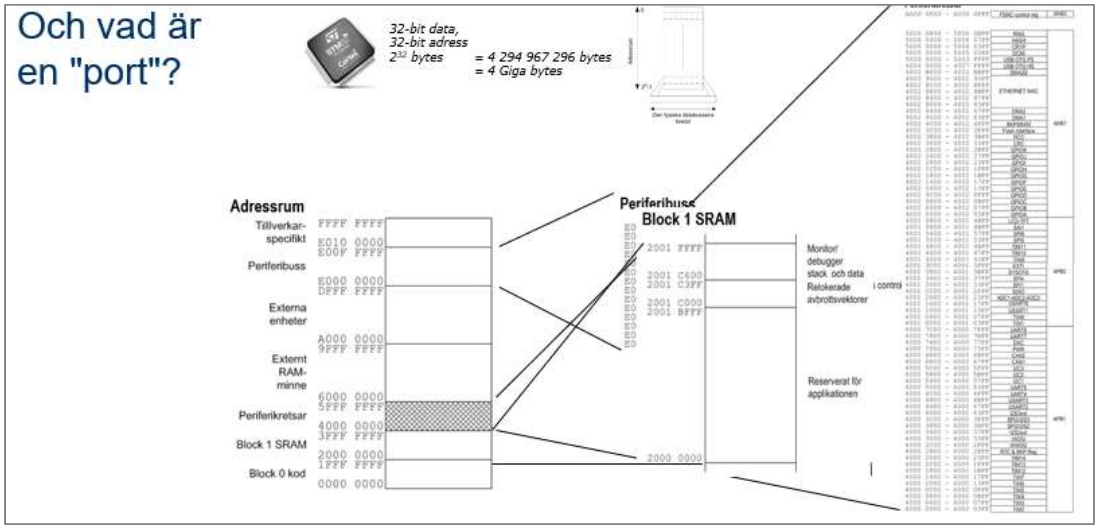
Pekare tillåter oss att referera en variabel (eller ett objekt) utan att först skapa en kopia

Exempel:

```
#include <stdio.h>

char person1[] = "Elsa";
char person2[] = "Alice";
char person3[] = "Maja";
char *winner;

int main(void)
{
    winner = person2;
    printf("%s is the winner\n", winner);
    winner = person1;
    printf("No, %s is the winner\n", winner);
    return 0;
}
```



Och vad är en "port"?



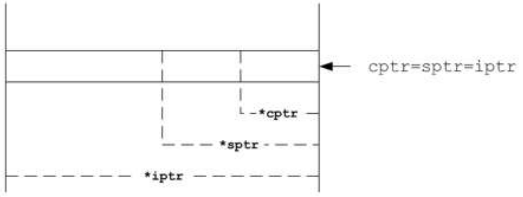
Grundläggande pekartyper, ARM/Thumb

```
char   *cptr; /* pekar på 8-bitars datatyp  */
short  *sptr; /* pekar på 16-bitars datatyp */
int    *iptr; /* pekar på 32-bitars datatyp */
```

Adressutrymmet är 32 bitar. PC, SP och ALLA pekartyper är 32 bitar

Exempel:

```
cptr = ( char  *) 0x40021010
sptr = ( short *) 0x40021010
iptr = ( int   *) 0x40021010
```



Användning av konstant pekare