

Hantering av textsträngar och talsträngar

ABCDEFGHIJKLMNO
PQRSTUVWXYZØabc
defghijklmnopqr
stuvwxyzø&12345
67890(\$.,!?)

54

Strängen "Hello world!"

PIC-processorerna lagrar strängkonstanter med bokstäverna inbakade i en följd av instruktioner (en tabell).

Man hämtar en bokstav genom att först ladda **W**-registret med bokstavens ordningsnummer och därefter göra ett subrutinanrop, tex **CALL text1**.

Tabellen **text1** börjar med instruktionen **ADDWF** som innebär att hoppet går vidare till önskat ordningsnummer i tabellen (enligt talet i **W**-registret). Där finner man instruktionen **RETLW** som innebär återhopp, men nu med den önskade bokstaven i **W**-registret.

```
Assemblerkod:  → text1 addwfp      ; jump indirect W
                 retlw 0x48    ; 'H'
                 retlw 0x65    ; 'e'
                 ← retlw 0x6c    ; 'l'
                 retlw 0x6c    ; 'l'
                 retlw 0x6f    ; 'o'
```

Detta kallas för computed goto

Cc5x C-sträng ”computed goto”

- Som funktion med ”computed goto”.

→ `char text1(char W)` *Skip-funktionen hoppar W stycken*
`{` *C-instruktioner (special för Cc5x)*
 `skip(W);`
 `return 'h'; return 'e'; return 'l'; return 'l';`
← `return 'o'; return 'w'; return 'o'; return 'r';`
 `return 'l'; return 'd'; return '\r';`
 `return '\n'; return '\0';`
`}`

- Eller samma sak med ett förenklat skrivsätt (pragma = special).

```
char text1( char W)
{
    skip(W);
    #pragma return[] = "hello world" '\r' '\n' '\0'
}
```

Skriva ut en sträng

```
char i, k;
for(i = 0 ; ; i++)
{
    k = text1(i);
    if( k == '\0') break;    // found end of string
    putchar(k);
}
```

hämta bokstav

tills strängen är slut

skriva ut bokstav

```
char text1( char W)
{
    skip(W);
    #pragma return[] = "hello world" '\r' '\n' '\0'
}
```

William Sandqvist william@kth.se

(C pekare * adress & och avreferering *)

Kort om C-språkets pekare – kommer Du ihåg?

`int b;` Deklaration av heltalsvariabeln `b`. Plats reserveras. 

`b = 18;` Definition av variabeln `b`. Nu innehåller den talet 18. 

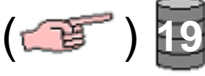
`&b` Adressoperatorn. Adressen till variabeln `b`. 

`int * c;` Deklaration av int-pekarevariabeln `c`. 

`c = &b;` Nu pekar `c` på `b`. 

`*c` Avrefereringsoperatorn. Det som `c` pekar på. 

`*c = 19;` Talet 19 lagras på den plats som `c` pekar ut.

Nu innehåller `b = 19`. 

C-språkets ”strängar”

```
const char text1[]="hello world\r\n";
```

En sträng är en char-array som avslutats med '`\0`'

```
&text1[0]=text1
```




```
text1[0]='h'
```



Index börjar med 0

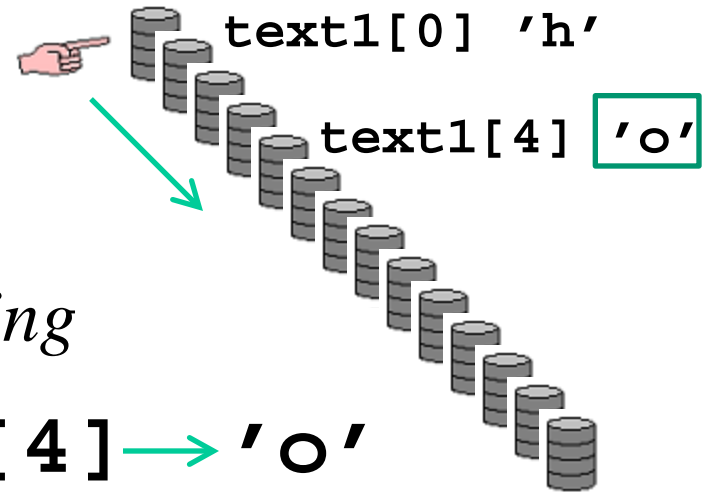
Arrayens **namn** är också
strängens startadress!

```
text1[13]='\0'
```

Adresser lagras i pekarvariabler

`char * a;` 

 `a` = `&text1[0];` 



Indexstegning eller Pekarstegning

Avreferering med Index: `a[4]` → `'o'`

Avreferering av Pekare: `*(a+4)` → `'o'`

Adresser som funktionsparametrar

Om en funktion ska kunna skriva ut *olika* strängar behöver man kunna skicka med strängens startadress

Funktionsdeklaration:



```
void string_out( const char * );
```

Plats för startadressen

Adresser som funktionsparametrar

Funktionsdefinition:



```
void string_out( const char * s )  
{  
    char i,k;  
    for(i = 0 ; ; i++)  
    {  
        k = s[i];  
        if( k == '\0' ) return;  
        putchar(k);  
    }  
}
```

*Här sker avreferering
med index*



Adresser som funktionsparametrar

Funktionsanrop:

skicka med startadressen



```
string_out( &text1[0] );
```

eller så också så här

```
string_out( "hello world\r\n" );
```

*Med detta skrivsätt lagras först strängkonstanten i minnet (som computed goto-tabell) men sedan är det **bara** ”startadressen” som skickas som parameter till funktionen.*

Inte mycket att peka på ?

```
const char text1[]="hello world\r\n";
```

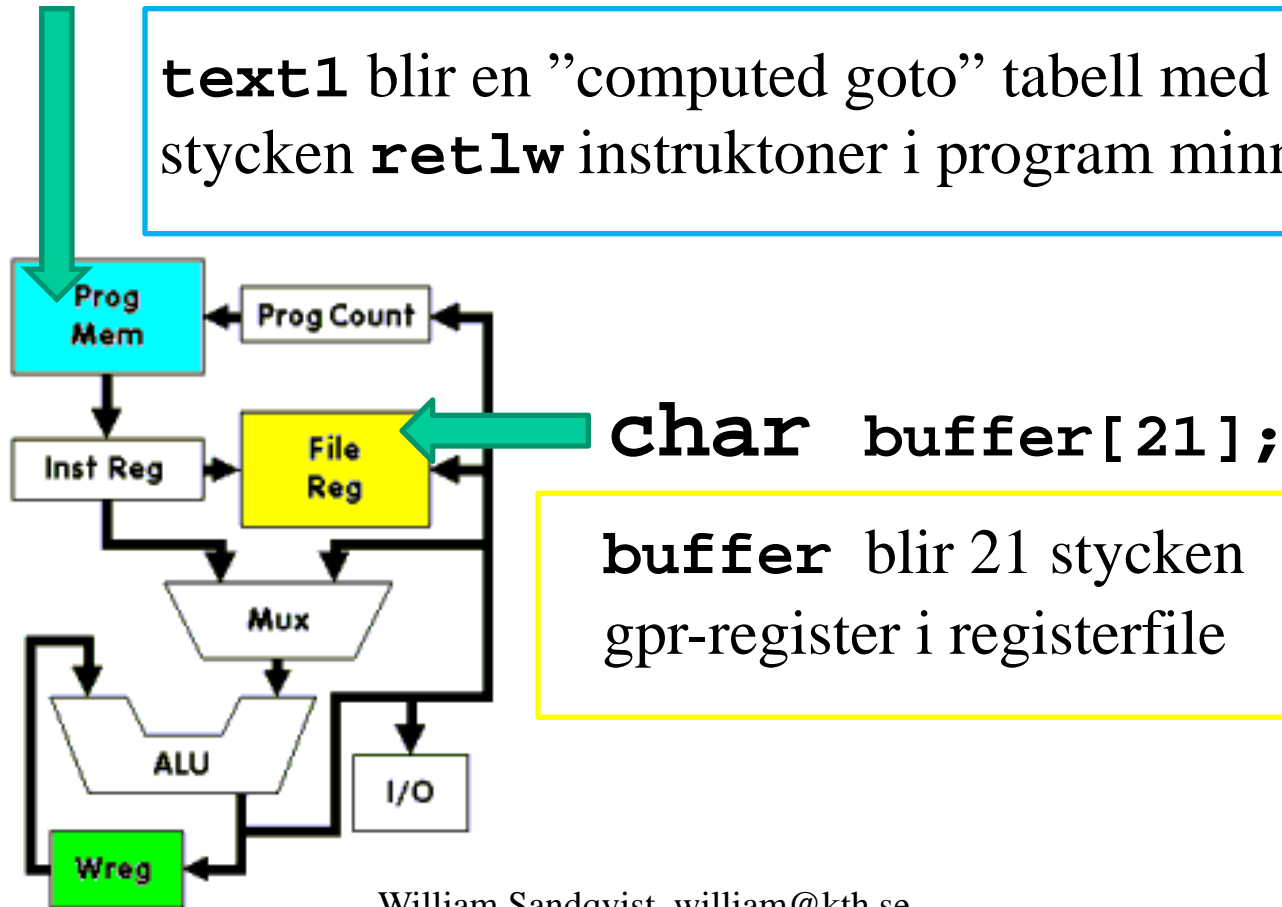
Det finns *inte* mycket att peka på i en midrange PIC-processor. Kompilatorn Cc5x tillåter trots detta att man använder pekare och viss C-syntax för pekare.

Detta är bara ”**för syns skull**”. Konstanta strängar lagras som ”computed goto” i programminnet, och strängvariabler som arrayer i registerarean.

Inte mycket att peka på ?

```
const char text1[]="hello world\r\n";
```

`text1` blir en "computed goto" tabell med 14 stycken `retlw` instruktioner i program minnet



```
char buffer[21];
```

`buffer` blir 21 stycken gpr-register i registerfile

hello.c

```
void main(void)
{
    char i;
    initserial(); /* Initialize serial port */
    string_out("hello world\r\n");
    while(1) nop();
}
```

```
void string_out( const char * s )
{
    char i,k;
    for(i = 0 ; ; i++)
    {
        k = s[i];
        if( k == '\0') return;
        putchar(k);
    }
}
```

Mata in en sträng

```
char s[11]; /* room for 10 characters and '\0' */
```

```
Anrop: string_in( &s[0] );
```

```
void string_in( char * str )
{
    char n, c;
    for(n = 0; ; n++ )
    {
        c = getchar( ); /* input 1 character */
        str[n] = c; /* store the character */
        putchar( c ); /* echo the character */
        if( (n == 10) || (c == '\r' ) ) /* end of input */
        {
            str[n] = '\0'; /* add "end of string" */
            return;
        }
    }
}
```

`string_in()` matar in bokstäver till buffern `s` tills "enter" (eller max antal tecken, **10** st).

comp_str ()

```
bit comp_str( char * in_str, const char * ref_str )
{
    char i, c, d;
    for(i=0; ; i++)
    {
        c = in_str[i];
        d = ref_str[i];
        if(d != c ) return 0;          /* no match      */
        if( d == '\0' ) return 1;    /* exact match */
    }
}
```

comp_str () jämför två strängar med varandra, om de är lika returneras en bitvariabel = 1

Lösenordskontroll!

printf() -lookalike

Standard ANSI-C har in och utmatningsfunktionerna **printf()** och **scanf()** i standardbiblioteket **stdio.h**. Detta är alldeles för omfattande och komplicerade funktioner för en PIC-processor. För den som är van vid C vore åtminstone en funktion som *liknar printf()* bra att ha. Speciellt vid avlusningen av programkoden.

```
void printf(const char *str, char var);
```

```
printf("Hello World!\n\r", 0);  
printf("Number 234 as unsigned: %u\n\r", 234);  
printf("Number 234 as signed: %d\n\r", 234);  
printf("Number 234 as binary: %b\n\r", 234);  
printf("Print a char: %C\n\r", 'W');
```

Hur ser utskriften ut?

`printf ()` -lookalike

Så här blev utskriften!

```
Hello World
Number 234 as unsigned: 234
Number 234 as signed: -022
Number 234 as binary: 11101010
Print a char: W
```

Från binärt till ASCII

Ex talet 123

$s[3] = '\backslash 0'$;

$s[2] = 123 \% 10 + '0' = '3'$

$123 / 10 = 12$

$s[1] = 12 \% 10 + '0' = '2'$

$12 / 10 = 1$

$s[0] = 1 \% 10 + '0' = '1'$

En konverterings-
algoritm.

Efter tre steg
innehåller $s[]$ de
ASCII-tecken som
ska skrivas ut.

$s[] = \{ '1', '2', '3', '\backslash 0' \}$

```

void printf(const char *str, char var)
{
    char i, k, m, a, b;
    for(i = 0 ; ; i++)
    {
        k = str[i];
        if( k == '\0') break; // at end of string
        if( k == '%') // insert variable in string
        {
            i++;
            k = str[i];
            switch(k)
            {
                case 'd': // %d signed 8bit
                    if( variable.7 ==1) putchar('-');
                    else putchar(' ');
                    if( variable > 127) variable = -variable; // no break!
                case 'u': // %u unsigned 8bit
                    a = variable/100;
                    putchar('0'+a); // print 100's
                    b = variable%100;
                    a = b/10;
                    putchar('0'+a); // print 10's
                    a = b%10;
                    putchar('0'+a); // print 1's
                    break;
                ...
            }
        }
    }
}

```



Omvandling till decimaltal
är krävande.

Att skriva ut
variabler binärt
tar inte speciellt
mycket resurser



```
...  
case 'b': // %b BINARY 8bit  
    for( m = 0 ; m < 8 ; m++ )  
    {  
        if (variable.7 == 1) putchar('1');  
        else putchar('0');  
        variable = rl(variable);  
    }  
    break;
```

```
case 'c': // %c 'char'  
    putchar(variable);  
    break;  
case '%':  
    putchar('%');  
    break;  
default: // not implemented  
    putchar('!');
```

```
    }  
    }  
else putchar(k);
```

```
}  
}
```

I praktiken tar
man bara med de
talomvandlingar
man verkligen
behöver – allt tar
plats och tar
exekveringstid.

William Sandqvist william@kth.se