

**MPOR**

# **Překladač Cx51 - II**

**Pavel Kučera**



# Jazyk C – shrnutí základních pojmů

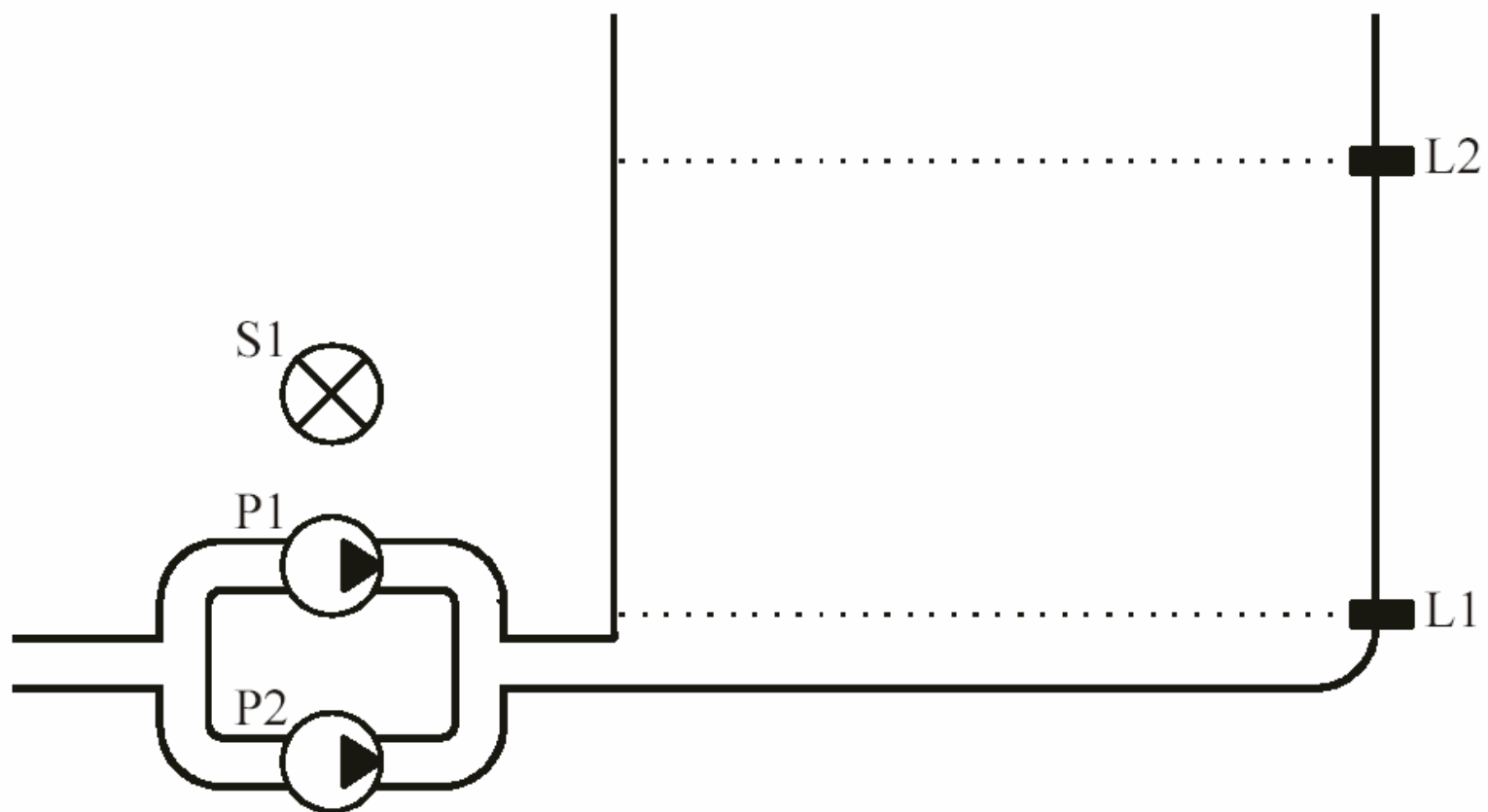
## switch

```
switch (int_expression) {  
    case value1 : statement_1; break;  
    ...  
    case value1 : statement_N; break;  
    default : statement_def; break;  
}
```

**break** vždy ukončuje nejvnitřnější smyčku !

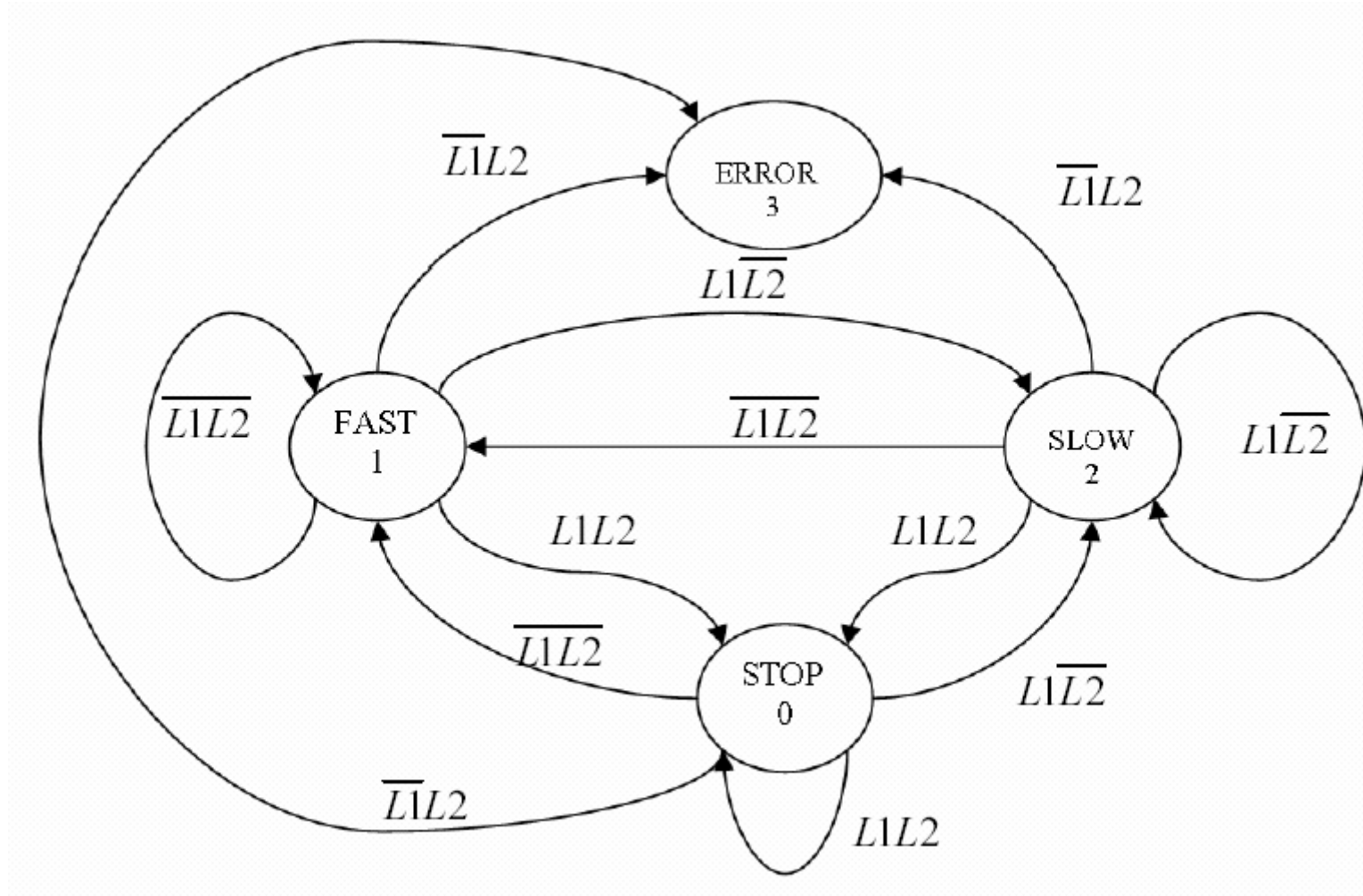


# Příklad





# Příklad





# Příklad

<i>No.</i>	<i>Name</i>	<i>Action</i>		
		<i>P1</i>	<i>P2</i>	<i>S1</i>
0	STOP	0	0	0
1	FAST	1	1	0
2	SLOW	1	0	0
3	ERROR	0	0	1

```
int main() {  
  
    state = 0;  
    do {  
        ReadSensors();  
        switch (state) {  
            case 0: ActionsInState_0(); ConditionsInState_0(); break;  
            case 1: ActionsInState_1(); ConditionsInState_1(); break;  
            case 2: ActionsInState_2(); ConditionsInState_2(); break;  
            case 3: ActionsInState_3(); ConditionsInState_3(); break;  
            default: ActionsDefault(); ConditionsDefault(); break;  
        }  
        WriteActuators();  
    }  
    while (1);  
    return 0;  
}
```



## Příklad

```
void ActionsInState_0() {
    P1 = 0;
    P2 = 0;
    S1 = 0;
}
void ActionsInState_1() {
    P1 = 1;
    P2 = 1;
    S1 = 0;
}
void ActionsInState_2() {
    P1 = 1;
    P2 = 0;
    S1 = 0;
}
void ActionsInState_3() {
    P1 = 0;
    P2 = 0;
    S1 = 1;
}
void ActionDefault() {
    P1 = 0;
    P2 = 0;
    S1 = 1;
}
```

```
void ConditionsInState_0() {
    if (L1 && L2) state = 0;
    if (!L1 && L2) state = 3;
    if (L1 && !L2) state = 2;
    if (!L1 && !L2) state = 1;
}
void ConditionsInState_1() {
    if (L1 && L2) state = 0;
    if (!L1 && L2) state = 3;
    if (L1 && !L2) state = 2;
    if (!L1 && !L2) state = 1;
}
void ConditionsInState_2() {
    if (L1 && L2) state = 0;
    if (!L1 && L2) state = 3;
    if (L1 && !L2) state = 2;
    if (!L1 && !L2) state = 1;
}
void ConditionsInState_3() {
    if (L1 && L2) state = 3;
    if (!L1 && L2) state = 3;
    if (L1 && !L2) state = 3;
    if (!L1 && !L2) state = 3;
}
void ConditionsDefault() {
    state = 3;
}
```



# Jazyk C – shrnutí základních pojmů

## goto

```
goto label;
```

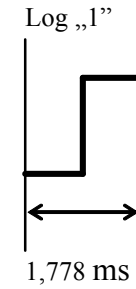
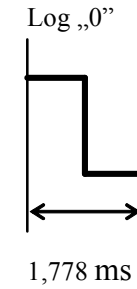
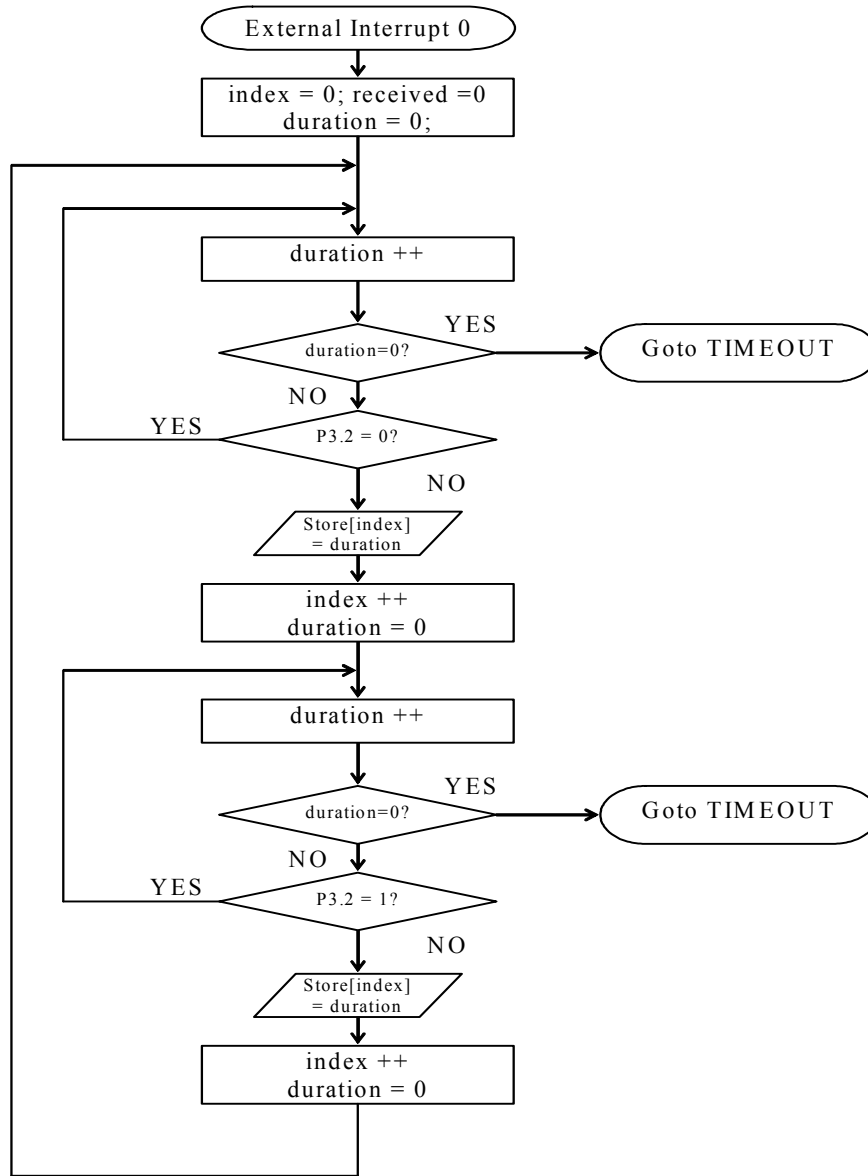
```
...
```

```
label:
```

Příkaz `goto` se ve správně strukturovaných programech vyskytuje velmi zřídka. Nicméně ve zvláštních případech umožní rychlý odskok do jiné části dekódovací rutiny - zejména v obsluhách přerušení a ovladačích zařízení.



# MPOR – II





# Překladač Cx51

`bit`

## Může být využit:

- při deklaraci proměnné,
- jako argument funkce,
- jako návratový parametr funkce (krom některých případů, viz dále).

Př.:

```
static bit flag = 0;
```

```
bit bit_or(bit a, bit b) {  
    bit flag;  
    flag = a  
    return flag;  
}
```



# Překladač Cx51

**bit**

Proměnné typu **bit** jsou uloženy v bitovém segmentu interní paměti procesoru - paměť **bdata** (20-2F)h.

Rozsah je tedy  $16 \times 8 = 128$  bitů.

Funkce, které zakazují přerušení a funkce, které jsou explicitně deklarovány pro použití s určitou registrovou sadou NEMOHOU vracet typ **bit**.

IRAM Address									Description
00h	R0	R1	R2	R3	R4	R5	R6	R7	Reg. bank 0
08h	R0	R1	R2	R3	R4	R5	R6	R7	Reg. bank 1
10h	R0	R1	R2	R3	R4	R5	R6	R7	Reg. bank 2
18h	R0	R1	R2	R3	R4	R5	R6	R7	Reg. bank 3
20h	00h	08h	10h	18h	20h	28h	30h	38h	Bits 00h-3Fh
28h	40h	48h	50h	58h	60h	68h	70h	78h	Bits 40h-7Fh
30h	GENERAL PURPOSE RAM 80 Bytes								GENERAL IRAM
.									
.									
7Fh									
80h	SPECIAL FUNCTION REGISTERS 128 Bytes								SFR
.									
.									
FFh									

**Nelze** deklarovat pointer na bit a **nelze** deklarovat pole bitů:

```
bit *b_pointer;      /*CHYBA!*/  
bit pole[10];       /*CHYBA!*/
```



# Překladač Cx51

## Bitově adresovatelné proměnné

Proměnné, které leží v interní paměti procesoru v oblasti **bdata**, jsou bitově přístupné.

Nelze deklarovat funkci vracející typ **bdata**.

Př.:

```
int bdata store; /*bitově přístupný int*/  
char bdata field[4]; /*bitově přístupné pole*/
```

**/\*CHYBA!\*/**

```
bdata sum(int a, int b) {  
    . . .  
}
```



# Překladač Cx51

**sbit**

```
int bdata store;
```

```
char bdata field[4];
```

```
sbit nejnizsi_bit_store = store ^ 0;
```

```
sbit nejvyssi_bit_store = store ^ 15;
```

```
sbit field05 = field[0] ^ 5;
```



# Překladač Cx51

`sbit`

Výraz za operátorem **^ musí** být konstanta ve správném rozsahu.  
**POZOR!** Mimo deklaraci znamená operátor **^ XOR!**

Platné rozsahy bitového přístupu:

<i>typ</i>	<i>^ min</i>	<i>^ max</i>
<code>char</code>	0	7
<code>unsigned char</code>	0	7
<code>int</code>	0	15
<code>unsigned int</code>	0	15
<code>short</code>	0	15
<code>unsigned short</code>	0	15
<code>long</code>	0	31
<code>unsigned long</code>	0	31



# Překladač Cx51

**sbit**

```
int bdata store;  
char bdata field[4];  
sbit field05 = field[0] ^ 5;  
sbit nejnizsi_bit_store = store ^ 0;  
sbit nejvyssi_bit_store = store ^ 15;
```

```
field05 = 0;           /*nastaví 5. bit ve field[0] do nuly*/  
field[5] = 88;        /*bajtový přístup*/  
field[3] = 'A';       /*bajtový přístup*/  
store = 0xFFAA        /*dvoubajtový přístup*/  
nejvyssi_bit_store = 1; /*nastaví 15. bit ve store do jednický*/
```







# Překladač Cx51

sfr

Speciální funkční registry okupují oblast 0x80 až 0xFF v interní paměti procesoru - SFR.

Address	Bit access							Address	
	0	1	2	3	4	5	6		7
F8h	PI2	CH	CCAP0H	CCAP1H	CCAP2H	CCAP3H	CCAP4H		FFh
F0h	B								F7h
E8h	P5	CL	CCAP0L	CCAP1L	CCAP2L	CCAP3L	CCAP4L		EFh
E0h	ACC								E7h
D8h	CCON	CMOD	CCAPM0	CCAPM1	CCAPM2	CCAPM3	CCAPM4		DFh
D0h	PSW	FCON	EECON						D7h
C8h	T2CON	T2MOD	RCAP2L	RCAP2H	TL2	TH2			CFh
C0	P4			SPCON	SPSTA	SPDAT		P5 (BYTE)	C7h
B8h	IPL0	SADEN							BFh
B0h	P3	IEN1	IPL1	IPH1				IPH0	B7h
A8h	IEN0	SADDR						CKCON1	AFh
A0h	P2		AUXR1				WDTRST	WDTPRG	A7h
98h	SCON	SBUF	BRL	BDRCON	KBLS	KBE	KBF		9Fh
90h	P1							CKRL	97h
88h	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	CKCON0	8Fh
80h	P0	SP	DPL	DPH				PCON	87h

Key:

	8051 series
	8052 series
	AT89C51ED2/RD2 series
	not in use



# Překladač Cx51

**sfr/sfr16**

Př.:

```
sfr P0      = 0x80;      /*Timer 2: T2L:T2H, 0xCC:0xCD*/
sfr P1      = 0x90;      sfr16 TL2 = 0xCC;
sfr P2      = 0xA0;
sfr P3      = 0xB0;      /*R2CAPL:R2CAPH, 0xCA:0xCB*/
sfr PSW     = 0xD0;      sfr16 RCAP2= 0xCA;
sfr ACC     = 0xE0;
sfr B       = 0xF0;
sfr SP      = 0x81;
```

reg51.h           - standard 8051  
at89x51.h         - série 89C51  
89c51rd2.h        - pro typ AT89C51 RD2 & ED2



# Překladač Cx51

## sbit a sfr

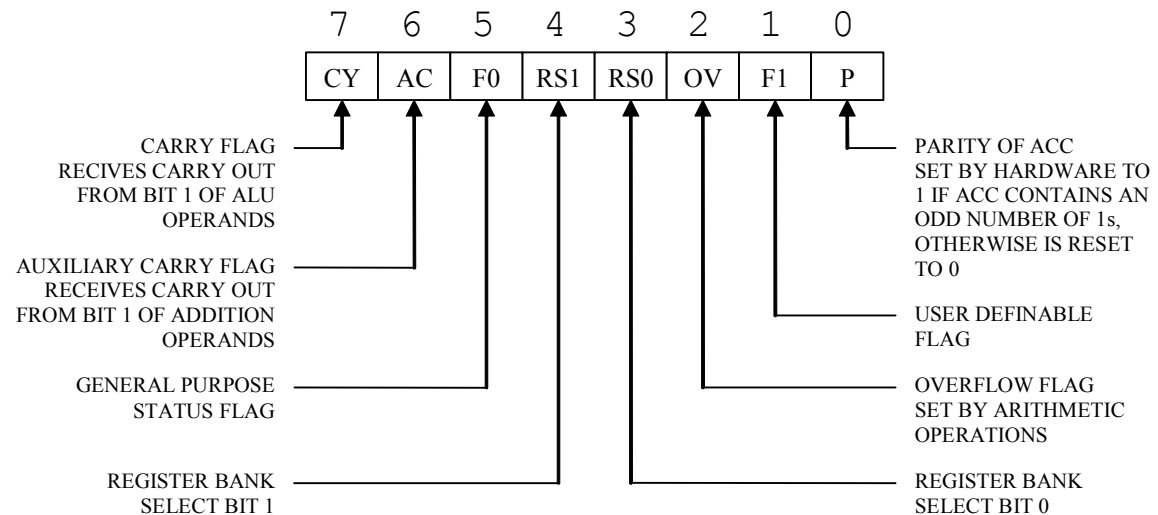
SFR které leží na adrese dělitelné 8 jsou bitově přístupné pomocí typu **sbit**.

Pokud chceme získat adresu bitu, který leží ve SFR, který je bitově přístupný, stačí přičíst k bázové adrese příslušného SFR bitový offset.

Př.:

PSW: 0xD0

Carry:  $0xD0+7=0xD7$





# Překladač Cx51

**sbit a sfr**

Varianta 1:

**sfr\_name ^ int\_constant**

Př.:

```
sfr PSW = 0xD0;  
sfr IE = 0xA8;  
sbit OV = PSW ^ 2;  
sbit CY = PSW ^ 7;  
sbit EA = IE ^ 7;
```



MPOR – II



# Překladač Cx51

**sbit a sfr**

Varianta 2:

**int\_constant ^ int\_constant**

Př.:

```
sbit OV   = 0xD0 ^ 2;  
sbit CY   = 0xD0 ^ 7;  
sbit EA   = 0xA8 ^ 7;
```



MPOR – II



# Překladač Cx51

**sbit a sfr**

Varianta 3:

**int\_constant**

Př.:

```
sbit OV   = 0xD2;  
sbit CY   = 0xD7;  
sbit EA   = 0xAF;
```



# Překladač Cx51

## Deklarace funkcí

```
[return_type] funcname([args])  [{small|compact|large}]  
                                [reentrant][interrupt n][using n]
```

Př.:

```
int calc(char i, int b) large reentrant;  
int plus(int i, float f) large;  
void service(void) using 3;  
void timer0(void) interrupt 1 using 3;
```



# Překladač Cx51

## Předávání parametrů přes registry

```
#pragma REGPARMS  
#pragma NOREGPARMS
```

Standardně se předávají přes fixní paměť (nikoliv zásobník).  
Až tři parametry mohou být předány pomocí registrů R0...R7.

No.	char, 1-byte ptr	int, 2-byte ptr	long, float	generic ptr
1	R7	R6&R7	R4-R7	R1-R3
2	R5	R4&R5	R4-R7	R1-R3
3	R3	R2&R3		R1-R3

Pozn.:

Pokud je ve funkci použit jako první parametr typ `bit`, pak jsou všechny parametry předány přes paměť! Typ `bit` je proto vhodné psát jako poslední.



# Překladač Cx51

## Návratový parametr funkce

Vždy jsou využity registry R0...R7.

Type	Register	Note
<b>bit</b>	Carry Falg	
<b>char, unsigned char, 1-byte ptr</b>	R7	
<b>int, unsigned int, 2-byte ptr</b>	R6&R7	MSB in R6, LSB in R7
<b>long, unsigned long</b>	R4-R7	MSB in R4, LSB in R7
<b>float</b>	R4-R7	IEEE 754
generic ptr	R1-R3	R3 – type, MSB in R2, LSB in R1



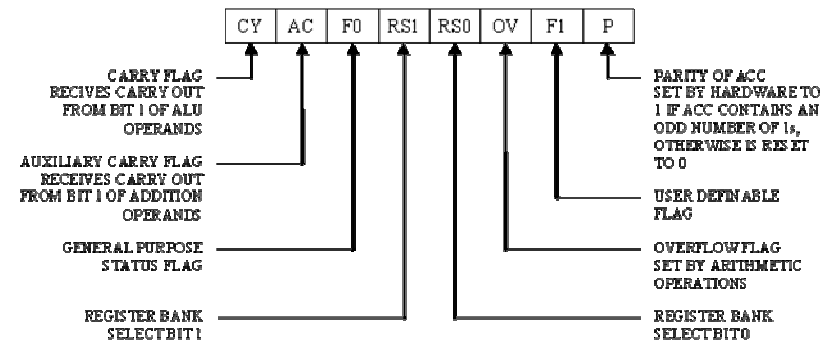
# Překladač Cx51

## Registrové banky & funkce

Př.:

```
void int0(void) using 3  
{  
.  
.  
.  
}
```

IRAM Address									Description
00h	R0	R1	R2	R3	R4	R5	R6	R7	Reg. bank 0
08h	R0	R1	R2	R3	R4	R5	R6	R7	Reg. bank 1
10h	R0	R1	R2	R3	R4	R5	R6	R7	Reg. bank 2
18h	R0	R1	R2	R3	R4	R5	R6	R7	Reg. bank 3



- Volitelné přes PSW.
- Parametr banky musí být konstanta v rozmezí 0...3.
- Není povoleno v prototypu funkce.
- Cx51 nekontroluje kolizi při volání funkcí se stejnými reg. bankami!



# Překladač Cx51

## Volání funkcí - příklad

```
bit alarm;
int n_alarm;
extern void func1(bit b0);
extern void func2(char c, int i);
extern void func3(bit b0, char c, int i);

void falarm(void) using 3 {
    n_alarm++;
    func1(alarm = 0);
    func2('a', 0x1155);
    func3(0, 'a', 0xAABB);
}
```



# Překladač Cx51

## Volání funkcí - příklad

```
1      #pragma REGPARMS
2
3      bit alarm;
4      int n_alarm;
5      extern void func1(bit b0);
6      extern void func2(char c, int i);
7      extern void func3(bit b0, char c, int i);
8
9      void falarm(void) using 3 {
10     1          n_alarm++;
11     1          func1(alarm = 0);
12     1          func2('a', 0x1155);
13     1          func3(0, 'a', 0xAABB);
14     1      }
```

□C51 COMPILER V7.00 Beta 6 REGBANK

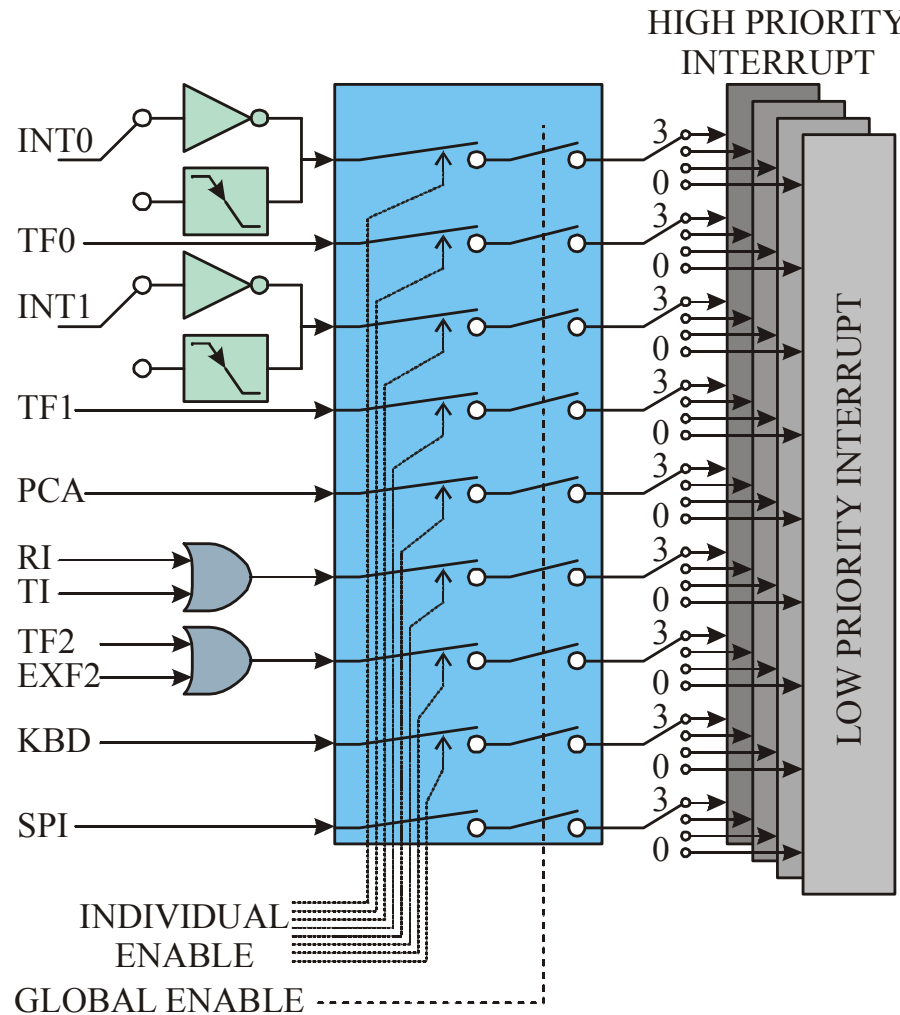
```
ASSEMBLY LISTING OF GENERATED OBJECT CODE

; FUNCTION falarm (BEGIN)
0000 C0D0          PUSH    PSW
0002 75D018       MOV     PSW,#018H
; SOURCE LINE # 9
; SOURCE LINE # 10
0005 0500         R      INC     n_alarm+01H
0007 E500         R      MOV     A,n_alarm+01H
0009 7002         JNZ    ?C0002
000B 0500         R      INC     n_alarm
000D              ?C0002:
; SOURCE LINE # 11
000D C3           CLR     C
000E 9200         R      MOV     alarm,C
0010 9200         E      MOV     ?func1?BIT,C
0012 120000       E      LCALL   func1
; SOURCE LINE # 12
0015 7D55         MOV     R5,#055H
0017 7C11         MOV     R4,#011H
0019 7F61         MOV     R7,#061H
001B 120000       E      LCALL   _func2
; SOURCE LINE # 13
001E C200         E      CLR     ?func3?BIT
0020 750061       E      MOV     ?func3?BYTE,#061H
0023 7500AA       E      MOV     ?func3?BYTE+01H,#0AAH
0026 7500BB       E      MOV     ?func3?BYTE+02H,#0BBH
0029 120000       E      LCALL   func3
; SOURCE LINE # 14
002C D0D0         POP     PSW
002E 22           RET
; FUNCTION falarm (END)
```



# Překladač Cx51

## Řerušovací systém



No.	Interrupt Source	Internal Peripheral Module	Interrupt Vector (Vector Address)	Note
0	RESET	Reset	0000h	NMI
1	INT0	External input signal	0003h	IRQ
2	TF0	Timer 0	000Bh	IRQ
3	INT1	External input signal	0013h	IRQ
4	TF1	Timer 1	001Bh	IRQ
5	RI/TI	UART	0023h	IRQ
6	TF2/EXF2	Timer 2	002Bh	IRQ
7	PCA	Programmable Counter Array	0033h	IRQ
8	KBD	Keyboard	003Bh	IRQ
9	-	-	0043h	-
10	SPI	SPI	004Bh	IRQ



# Překladač Cx51

## Zdroje přerušení

Interrupt Number	Interrupt Source	Internal Peripheral Module	Interrupt Vector (Vector Address)	Note
X	RESET	Reset	0000h	NMI
0	INT0	External input signal	0003h	IRQ
1	TF0	Timer 0	000Bh	IRQ
2	INT1	External input signal	0013h	IRQ
3	TF1	Timer 1	001Bh	IRQ



# Překladač Cx51

## Obsluha přerušení

```
void function(void) interrupt n {  
.  
.  
.  
}
```

Argument *n* může být v rozsahu 0...31.

Argument *n* musí být konstanta.

Činnost:

1. SFR ACC, B, DBH, DPL a PSW jsou uloženy do zásobníku (je-li to potřeba).
2. Všechny použité registry R jsou uloženy do zásobníku – pokud není stanovena registrová banka klíčovým slovem **using**.
3. Po ukončení obsluhy jsou SFR a Registry obnoveny ze zásobníku.
4. Je zavolána instrukce RETI.



```
void timer0_1(void) interrupt 1 {  
    int a,b;  
    long l;  
    n_alarm = l*a+b;  
}
```

```
; FUNCTION timer0_1 (BEGIN)  
0000 C0E0          PUSH    ACC  
0002 C0F0          PUSH    B  
0004 C083          PUSH    DPH  
0006 C082          PUSH    DPL  
0008 C0D0          PUSH    PSW  
000A 75D000        MOV     PSW,#00H  
000D C000          PUSH    AR0  
000F C001          PUSH    AR1  
0011 C002          PUSH    AR2  
0013 C003          PUSH    AR3  
0015 C004          PUSH    AR4  
0017 C005          PUSH    AR5  
0019 C006          PUSH    AR6  
001B C007          PUSH    AR7  
  
                                ; SOURCE LINE # 16  
001D AE00          R      MOV     R6,a  
001F AF00          R      MOV     R7,a+01H  
0021 EE           MOV     A,R6  
0022 33           RLC     A  
0023 95E0         SUBB   A,ACC  
0025 FD           MOV     R5,A  
0026 FC           MOV     R4,A  
0027 AB00         R      MOV     R3,l+03H  
0029 AA00         R      MOV     R2,l+02H  
002B A900         R      MOV     R1,l+01H  
002D A800         R      MOV     R0,l  
002F 120000       E      LCALL  ?C?LMUL  
0032 E500         R      MOV     A,b+01H  
0034 2F           ADD    A,R7  
0035 F500         R      MOV     n_alarm+01H,A  
0037 E500         R      MOV     A,b  
0039 3E           ADDC  A,R6  
003A F500         R      MOV     n_alarm,A  
  
                                ; SOURCE LINE # 21  
003C D007          POP     AR7  
003E D006          POP     AR6  
0040 D005          POP     AR5  
0042 D004          POP     AR4  
0044 D003          POP     AR3  
0046 D002          POP     AR2  
0048 D001          POP     AR1  
004A D000          POP     AR0  
004C D0D0          POP     PSW  
004E D082          POP     DPL  
0050 D083          POP     DPH  
0052 D0F0          POP     B  
0054 D0E0          POP     ACC  
0056 32           RETI  
  
; FUNCTION timer0_1 (END)
```



# Překladač Cx51

```
void timer0_2(void) interrupt 1 using 2 {  
int a,b;  
long l;  
n_alarm = l*a+b;  
}
```

```
; FUNCTION timer0_2 (BEGIN)  
0000 C0E0          PUSH   ACC  
0002 C0F0          PUSH   B  
0004 C083          PUSH   DPH  
0006 C082          PUSH   DPL  
0008 C0D0          PUSH   PSW  
000A 75D010       MOV    PSW,#010H  
  
; SOURCE LINE # 16  
  
000D AE00          R      MOV    R6,a  
000F AF00          R      MOV    R7,a+01H  
0011 EE           MOV    A,R6  
0012 33           RLC   A  
0013 95E0          SUBB  A,ACC  
0015 FD           MOV    R5,A  
0016 FC           MOV    R4,A  
0017 AB00          R      MOV    R3,l+03H  
0019 AA00          R      MOV    R2,l+02H  
001B A900          R      MOV    R1,l+01H  
001D A800          R      MOV    R0,l  
001F 120000       E      LCALL  ?C?LMUL  
0022 E500          R      MOV    A,b+01H  
0024 2F           ADD   A,R7  
0025 F500          R      MOV    n_alarm+01H,A  
0027 E500          R      MOV    A,b  
0029 3E           ADDC  A,R6  
002A F500          R      MOV    n_alarm,A  
  
; SOURCE LINE # 21  
  
002C D0D0          POP   PSW  
002E D082          POP   DPL  
0030 D083          POP   DPH  
0032 D0F0          POP   B  
0034 D0E0          POP   ACC  
0036 32           RETI
```



# Překladač Cx51

## Obsluha přerušení

- Vstupní parametry funkce přerušení musí být prázdné.
- Návratový parametr z funkce přerušení musí být void. Implicitní návratový typ `int` je překladačem ignorován.
- Překladač nedovolí přímé volání funkce přerušení z programu.
- Překladač generuje vektor přerušení automaticky – lze zabránit z příkazové řádky direktivou `NOINTVECTOR`.
- Pozor na volání funkcí z přerušení, které nepoužívají stejnou registrovou banku.



# Překladač Cx51

## Reentrantní funkce

```
type function(arg) reentrant {  
    ...  
}
```

Př.:

```
int sum(char i, int b) reentrant {  
    int x;  
    x = table[i];  
    return (x*b);  
}
```



# Překladač Cx51

## Reentrantní funkce

- Nepodporují typ bit.
- Mohou být definovány v příslušném paměťovém režimu (**small**, **compact**, **large**)
- Mohou být **interrupt**.
- Mohou používat konkrétní registrovou banku – **using**.



# Překladač Cx51

## Reentrantní funkce

memchr  
memcmp  
memcpy  
memmove  
memset  
toascii  
toint  
tolower  
toupper  
abs  
cabs

labs  
fabs  
rand  
getchar  
puts  
strchr  
strcmp  
strcpy  
strlen  
strpos



# Překladač Cx51

## Optimalizace proměnných

```
20  char c;  
21  c = 0;  
22  while(!c) {  
23  
24  }
```

```
                                ; SOURCE LINE # 21  
;---- Variable 'c' assigned to Register 'R7' ----  
0000 E4                CLR    A  
0001 FF                MOV    R7,A  
0002                ?C0001:  
  
                                ; SOURCE LINE # 22  
0002 EF                MOV    A,R7  
0003 60FD             JZ     ?C0001  
  
                                ; SOURCE LINE # 24
```



# Překladač Cx51

`volatile`

```
20  volatile char c;  
21  c = 0;  
22  while(!c) {  
23  
24  }
```

```
                                ; SOURCE LINE # 21  
0000 E4          CLR      A  
0001 F500        R        MOV      c,A  
0003             ?C0001:  
                                ; SOURCE LINE # 22  
0003 E500        R        MOV      A,c  
0005 60FC        JZ       ?C0001  
                                ; SOURCE LINE # 24
```