

When using code bank switching the address space is effectively extended from 16-bits to 24-bits with the high 8-bits specifying the code bank. A problem arises, however, when using function pointers. As function pointers are only 16-bits wide it is not possible to pass a full 24-bits function address. To solve this the compiler will generate an extra indirection level for every function address. This indirection is actually a 3 bytes area in the common area specifying the full 24-bit function address. Since the common area itself is limited to the first 64kb, pointers to these entries can be 16-bits. Now when calling a function through a function pointer the 16-bits address of the entry in the common area is passed instead of the 24-bits address of the function itself. A special runtime routine `__IICALL` is then called to handle the extra indirection.

The `__IICALL` runtime routine has to be adapted to handle target specific bank switching. The default implementation, using port P1 to switch code banks, is present in the 'iicall.src' routine in the C library, and is shown below.

Default calling functions through function pointers is handled completely by the compiler. However, in certain situations it is required to handle function pointers differently, e.g. when using function pointers in context switching.

The following example shows how to deal with the extra indirection level to get the full 3-bytes destination address of a function pointer.

```
__IICALL:
    PUSH    P1                ; store the current bank
    CALL    _bankswitch
    POP     P1                ; switch back to original bank
    RET

_bankswitch:
    CLR     A
    MOVC    A,@@A+DPTR       ; get low 8-bits of function
    PUSH    ACC

    INC     DPTR
    CLR     A
    MOVC    A,@@A+DPTR       ; get middle 8-bits of function
    PUSH    ACC

    INC     DPTR
    CLR     A
    MOVC    A,@@A+DPTR       ; get bank (highest 8-bits) of function
    MOV     P1,A              ; switch to new bank
    RET
```

```
#include <stdio.h>

typedef void (* t_fptr)(void);    /* function pointer typedef */

void task1( void ) {}
void task2( void ) {}
void task3( void ) {}

_rom t_fptr fparr[3] = { task1, task2, task3 }; /* declare function pointer array */

/*
 * This function prints the full address of the passed function pointer.
 */
void printfp( t_fptr fp )
{
    unsigned char adreslow = ((_rom unsigned char *)fp)[0]; /* Read low address byte of fp */
    unsigned char adresmid = ((_rom unsigned char *)fp)[1]; /* Read mid address byte of fp */
    unsigned char bank     = ((_rom unsigned char *)fp)[2]; /* Read high address byte of fp */

    printf( "fp points to function at address 0x0%x%x\n", (int) count, (unsigned int)bank,
(unsigned int)(adresmid << 8 + adreslow) );
}

void main(void)
{
    printfp( fparr[0] );
    printfp( fparr[1] );
    printfp( fparr[2] );
}
```

Example