 INDIRECT ADDRESSING
INDIRECT ADDRESSING

In Access Bank, the memory is divided into 128 pages of 1k bytes each. This method is known as "paging" in Data Memory and "BSR" in 1/4k memory. Each page is associated with 12 bytes of data memory.

FSR0, 1, 2 are used to indicate the current page. The FSR register is used to load the page number into FSR0, 1, 2. Data Memory is considered to be 4k bytes in size. Each page is 4k bytes in size.

Page 50, 51

- AN1 - INDFn
- AN2 - POSTDECn
- AN3 - POSTINCn
- AN4 - PREINCn
- AN5 - WREG
- AN6 - PLUSWn

For more information, please refer to page 51.
FIGURE 4-9: INDIRECT ADDRESSING OPERATION

Instruction Executed

OpCode Address

12

File Address = access of an indirect addressing register

Instruction Packed

OpCode File

BSR<3:0> 12

4 8

RAM 0

FFFh

FIGURE 4-10: INDIRECT ADDRESSING

Indirect Addressing

11

FSR Register 0

Location Select

0000h

Data Memory(1)

0FFFh

Note 1: For register file map detail, see Table 4-1.
למי شيئסוק בפקודת המעון העכיקס נסבי 2 פקודת בחר אנ נפנשיס ליאשונא:

.movff, deecfsz
DECFSZ

Decrement f, skip if 0

Syntax:  
\[[label]\] DECFSZ f|d[.a]]

Operands:

\[0 \leq f \leq 255\]
\[d \in [0, 1]\]
\[a \in [0, 1]\]

Operation:

\((f) - 1 \rightarrow \text{dest}, \skip\text{if result} = 0\)

Status Affected: None

Encoding:

| 0010 | 11da | eeee | eeee |

Description:
The contents of register \(f\) are decremented. If \(d\) is 0, the result is placed in \(W\). If \(d\) is 1, the result is placed back in register \(f\) (default).

If the result is 0, the next instruction, which is already fetched, is discarded, and a NOE is executed instead, making it a two-cycle instruction. If \(a\) is 0, the Access Bank will be selected, overriding the BSR value. If \(a\) is 1, then the bank will be selected as per the BSR value (default).

Words: 1

Cycles: 1(2)

Note: 3 cycles if skip and followed by a 2-word instruction.

Q Cycle Activity:

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decode</td>
<td>Read register (f)</td>
<td>Process Data</td>
<td>Write to destination</td>
</tr>
</tbody>
</table>

If skip:

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
</tr>
</thead>
<tbody>
<tr>
<td>No operation</td>
<td>No operation</td>
<td>No operation</td>
<td>No operation</td>
</tr>
</tbody>
</table>

If skip and followed by 2-word instruction:

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
</tr>
</thead>
<tbody>
<tr>
<td>No operation</td>
<td>No operation</td>
<td>No operation</td>
<td>No operation</td>
</tr>
<tr>
<td>No operation</td>
<td>No operation</td>
<td>No operation</td>
<td>No operation</td>
</tr>
</tbody>
</table>

Example:  HERE DECFSZ CNT, 1, 1  
CONTINUE

Before Instruction

PC = Address (HERE)

After Instruction

CNT = CNT - 1
If CNT = 0;  
PC = Address (CONTINUE)
If CNT = 0;  
PC = Address (HERE+2)
MOVFF

Move f to f

Syntax:  \[ [\text{label}] \text{ MOVFF } f_s, f_d \]

Operands:  
0 ≤ \( f_s \) ≤ 4095 
0 ≤ \( f_d \) ≤ 4095

Operation:  \((f_s) \rightarrow f_d\)

Status Affected:  None

Encoding:

<table>
<thead>
<tr>
<th>1st word (source)</th>
<th>2nd word (destin.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1100</td>
<td>( tttt )</td>
</tr>
<tr>
<td>1111</td>
<td>( tttt )</td>
</tr>
</tbody>
</table>

Description:
The contents of source register \( f_s \) are moved to destination register \( f_d \). Location of source \( f_s \) can be anywhere in the 4096 byte data space (000h to FFFh), and location of destination \( f_d \) can also be anywhere from 000h to FFFh. Either source or destination can be W (a useful special situation). MOVFF is particularly useful for transferring a data memory location to a peripheral register (such as the transmit buffer or an I/O port).

The MOVFF instruction cannot use the PCL, TOSU, TOSH or TOSL as the destination register.

Note:  The MOVFF instruction should not be used to modify interrupt settings while any interrupt is enabled. See Section 8.0 for more information.

Words:  2

Cycles:  2 (3)

Q Cycle Activity:

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decode</td>
<td>Read register ( f_s ) (src)</td>
<td>Process Data</td>
<td>No operation</td>
</tr>
<tr>
<td>Decode</td>
<td>No operation</td>
<td>No operation</td>
<td>Write register ( f_d ) (dest)</td>
</tr>
</tbody>
</table>

Example:  MOVFF REG1, REG2

Before Instruction:
- REG1 = 0x33
- REG2 = 0x11

After Instruction:
- REG1 = 0x33, REG2 = 0x33
נפתוח חלון \texttt{watch} \quad \texttt{WREG} \quad \texttt{FSR} - \quad \texttt{COUNTER}

뷰 \texttt{File Registers} - \quad \texttt{View} \\

מעבר \texttt{רגיסטרים} - \quad \texttt{FSR}
Enhanced devices have two memory spaces. The program memory space and the data memory space. The program memory space is 16 bits wide, while the data memory space is 8 bits wide. Table Reads and Table Writes have been provided to move data between these two memory spaces through an 8-bit register (TABLAT).

The operations that allow the processor to move data between the data and program memory spaces are:

- Table Read (TBLRD)
- Table Write (TBLWT)

Table Read operations retrieve data from program memory and place it into the data memory space. Figure 8-1 shows the operation of a Table Read with program and data memory.

Table Write operations store data from the data memory space into program memory. Figure 8-2 shows the operation of a Table Write with program and data memory.

Table operations work with byte entities. A table block containing data is not required to be word aligned, so a table block can start and end at any byte address. If Enhanced MCU instructions are being written to program memory, these instructions must be word aligned.

**Figure 8-1: Table Read Operation**

**Figure 8-2: Table Write Operation**
### TBLRD Table Read

**Syntax:**

```
[ label ] TBLRD (*; *++; *--; ++)
```

**Operands:** None

**Operation:**

- If TBLRD *:
  - (Prog Mem (TBLPTR)) $\rightarrow$ TABLAT;
  - TBLPTR - No Change;

- If TBLRD *++:
  - (Prog Mem (TBLPTR)) $\rightarrow$ TABLAT;
  - (TBLPTR) +1 $\rightarrow$ TBLPTR;

- If TBLRD *-:
  - (Prog Mem (TBLPTR)) $\rightarrow$ TABLAT;
  - (TBLPTR) -1 $\rightarrow$ TBLPTR;

- If TBLRD ++:
  - (TBLPTR) +1 $\rightarrow$ TBLPTR;
  - (Prog Mem (TBLPTR)) $\rightarrow$ TABLAT;

**Status Affected:** None

**Encoding:**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>10nm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Description:** This instruction is used to read the contents of Program Memory (P.M.). To address the program memory, a pointer called Table Pointer (TBLPTR) is used. The TBLPTR (a 21-bit pointer) points to each byte in the program memory. TBLPTR has a 2 Mbyte address range.

- TBLPTR[0] = 0: Least Significant Byte of Program Memory Word
- TBLPTR[0] = 1: Most Significant Byte of Program Memory Word

The TBLRD instruction can modify the value of TBLPTR as follows:

- no change
- post-increment
- post-decrement
- pre-increment

**Words:** 1

**Cycles:** 2

**Q Cycle Activity:**

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
</tr>
</thead>
<tbody>
<tr>
<td>No operation</td>
<td>No operation</td>
<td>No operation</td>
<td>No operation</td>
</tr>
</tbody>
</table>

**Example 1:**

Before Instruction

- TABLAT = 0x55
- TBLPTR = 0x00A356
- MEMORY(0x00A356) = 0x34

After Instruction

- TABLAT = 0x34
- TBLPTR = 0x00A357

**Example 2:**

Before Instruction

- TABLAT = 0xAA
- TBLPTR = 0x01A357
- MEMORY(0x01A357) = 0x12
- MEMORY(0x01A358) = 0x34

After Instruction

- TABLAT = 0x34
- TBLPTR = 0x01A358
addressing

300000B00000000 NOP
; For use with 4 MHz clock osc.

; Watchdog Timer disabled.
300002B00000000 NOP
PART_ON_2L
300006B00000000 NOP
; Low Voltage in-circuit ...

000000 D021 BRA 0x44
000002 0000 NOP

00002C 3130 RRCF 0x30, W, BANKED
00002E 3332 RRCF 0x32, F, BANKED
000030 3534 RLCF 0x34, W, BANKED
000032 3736 RLCF 0x36, F, BANKED
000034 3938 SWAPF 0x38, W, BANKED
000036 4241 BRNCF 0x41, F, ACCESS
000038 4443 RLNCF 0x43, W, ACCESS
00003A 4645 RLNCF 0x45, F, ACCESS
00003C 4847 INFSN 0x47, W, ACCESS
00003E 4A49 INFSN 0x49, F, ACCESS
000040 FF90 NOP

000044 EF29 GOTO 0x52
000046 F000 NOP

000048 ED3C CALL 0x78, 0x1
00004A F000 NOP
00004C EC58 CALL 0xb0, 0
00004E F000 NOP
000050 D7FF BRA 0x50

Purpose Register) bank,

BSR.

000052 0E12 MOVlw 0x12
000054 6B62 MOVWF 0xf92, ACCESS

value
0. see page 44.
it means

-11-
that's the address of PORTC
function reg. window,)
to watch window.

; (for doing that we have to open special
; we'll initiate PORTC to output and add it

clrf TRISC ; PORTC data direction
movlw 0x33 ; PORTC data latch when read
movwf PORTC ; PORTC latch
addwf MYREG1,0 ;RESULT TO F, ACCESS
addwf MYREG1,1 ;RESULT TO F, BSR
; we'll open the file register (data memory)
movlw .7 ;Bank 0
movwf MYREG1,1
movlw .8 ;Bank 1
movib 1
movwf .256,1
movlw .9
movib 3 ;Bank 3
movwf FLAG,1
; goto Done

; -----------------------------

Two
I/2. Indirect addressing
Indirect access to data memory with
and not with BSR or access memory

movib 0 ;bank 0
movlw 5
movwf COUNTER
lfsw 0,256 ;load FSR0
lfsw 1,200h
lfsw 2,0x300
clrf WREG
FILL2
; fill fsc2 memory with array of num
movwf POSTINC2 ;mov wreg to

incf WREG
decfsz COUNTER,1 ;still counter=0
bra FILL2
movsf IND2F ;mov last num
movf COUNTER ;initiate counter
clrf WREG
TRANS20
;transfer num from fsc2 memory to fsc0
movff POSTDEC2,PLUSW0
incf WREG
decfsz COUNTER,1
bra TRANS20
movff IND2F,PLUSW0 ;mov of last

CFDD MOVFF Oxffd, 0xefb
CFDE MOVFF Oxff, 0xefb
CFDF MOVFF Oxff, 0xefb
DEF2 MOVFF Oxff, 0xefb

2's complement
0000A8 0EFE MOVHL 0xfe
0000AA CFEE MOVHL 0xffe, 0xffdb
Effective address?
0000AC FFDB NOP
0000AE 0013 RETURN 0x1

memory)

0000B0 0100 MOVHL 0
0000B2 0E00 MOVHL 0xa
0000B4 6F81 MOVHF 0x81, BANKED
0000B6 0E00 MOVHL 0
0000B8 6F88 MOVHF 0xff8, ACCESS
TBLPTR
0000BA 0E00 MOVHL 0
0000BC 6EF7 MOVHF 0xff7, ACCESS
TBLPTR
0000BE 0E2C MOVHL 0x2c
0000C0 6EF6 MOVHF 0xff6, ACCESS
TBLPTR
0000C2 0E0A MOVHL 0xa
0000C4 26F6 ADDWF 0xff6, F, ACCESS
0000C6 6AE8 CLRIF 0xfe8, ACCESS
0000C8 2EF7 ADDWF 0xff7, F, ACCESS
0000CA 2EF8 ADDWF 0xff8, F, ACCESS
0000CC 0029 TBLRD++
0000CE 5F85 MOVF 0xff5, W, ACCESS
0000D0 2F81 DECFSZ 0x81, F, BANKED
0000D2 D7FC BRA 0xccc
0000D4 0012 RETURN 0

121: TWOCCM
122: ; transfer from fso to fstr memory with
123: movlw 0x0f
124: movff POSTINC0,PLUSW2 ; where is the
125: ;
126: return 1 ; Fast
127: ;
128: ;
129: ;
130: Three
131: ; 3. Reading program memory. (Flash
132: ; (we'll write & erase later)
133: ;
134: movlb 0
135: movlw .10
136: movwf FLAG
137: movlw UPPER stan_table
138: movwf TBLPTRU ; Upper byte of
139: movlw HIGH stan_table
140: movwf TBLPTRH ; High byte of
141: movlw LOW stan_table
142: movwf TBLPLTR ; Low byte of
143: movlw .10
144: addwf TBLPTRL,1
145: clrf WREG
146: addwf TBLPTRH,1
147: addwf TBLPLTR,1
148: Next
149: thldr++
150: movf TABLAT,0
151: declsz FLAG,1
152: bra Next
153: ;
154: return

- 13 -
שימו.VarChar ב_gray

.adr.asm עם הקובץ Project
. nop

לאחר ה[label] שוחל אחר הפקודת nop.

שם покודה עצירה הפקודת ה-nop.

.PCL, STKPTR, TOS: וitere בטוח آخر.

Hardware Stack - הצג את ה- View

דרק ה- הצג את ה-

בריצת עוקב אחריך השיניים במכסה עוקב כנה סכין länger.

ויורח.
צרו תכנית בה תשלל את שמכס הפרט ב -
באמצוות פקודת מעון עקף העוברים את שמכס לאור שינהור.
באמצוות פקודת דורמת (מען עקף) העוברים את שמכס למוקט נסף בסדר.
הפר.
והינשו את התכנית בצורת החולנות על שמכס בכל המקומאות הנית.