Οργάνωση μνήμης
Instruction Set

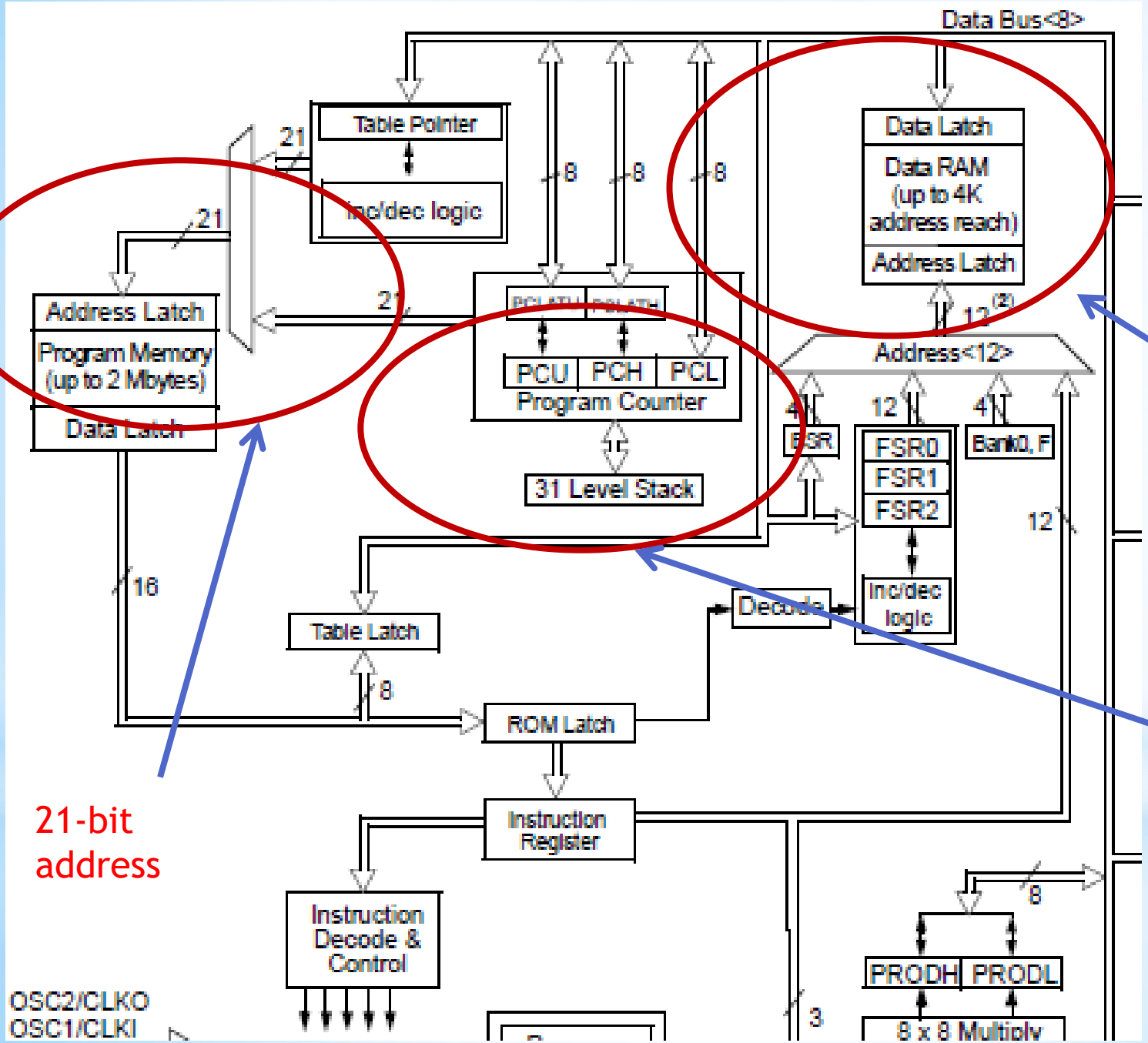
18F452

Οργάνωση Μνήμης

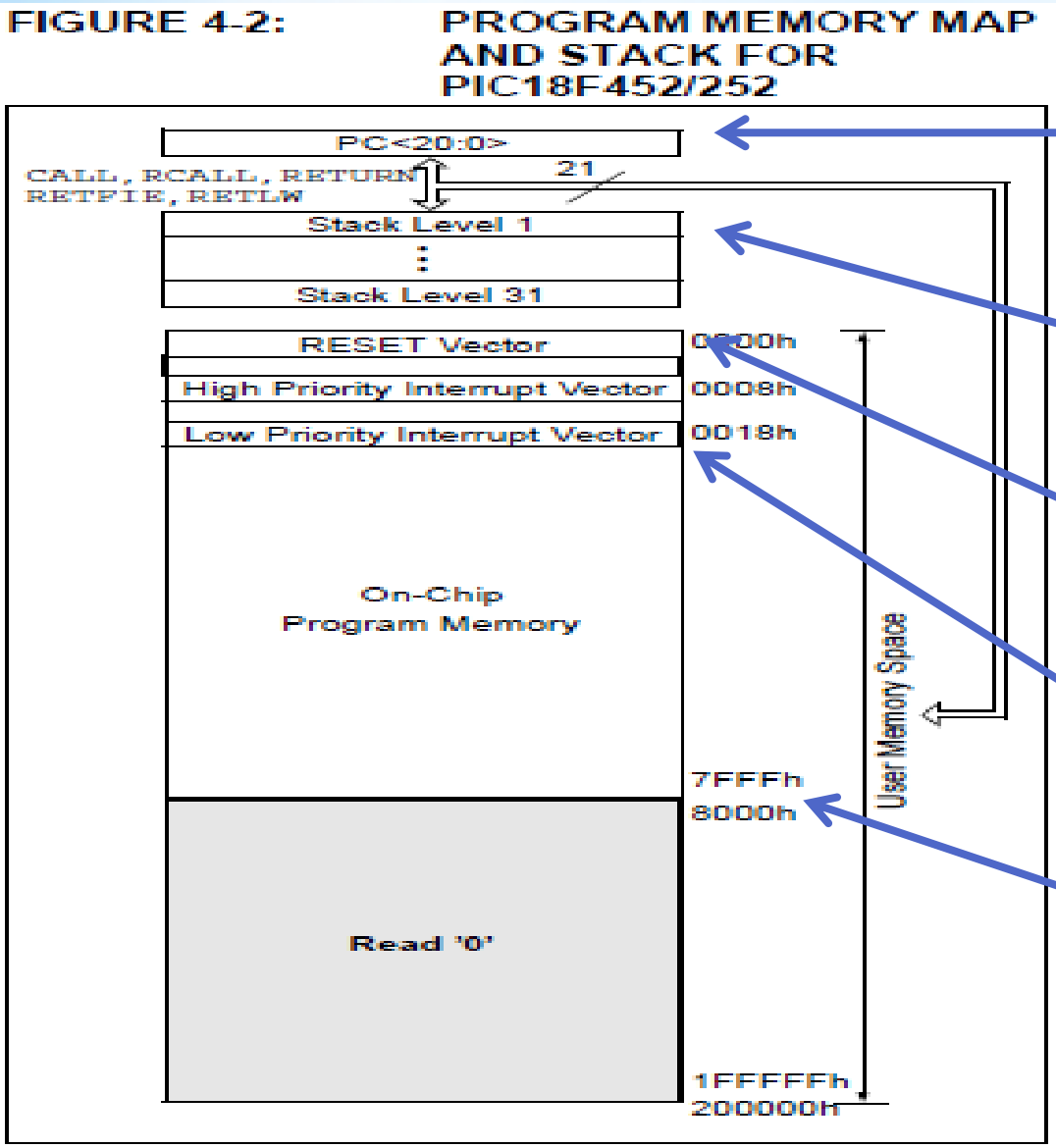
Τρία Μπλοκ Εσωτερικής Μνήμης

- Μνήμη Προγράμματος (Program Memory)
- Μνήμη Δεδομένων (Data RAM)
- Μη πτητική Μνήμη Δεδομένων (Data EEPROM)

Οι Μνήμες Προγράμματος και Δεδομένων χρησιμοποιούν διαφορετικούς διαύλους για να υπάρχει δυνατότητα ταυτόχρονης πρόσβασης


Data Bus<8>
Table Pointer
inc/dec logic
21
8
8
8
Data Latch
Data RAM
(up to 4K
address reach)
Address Latch
12
(2)
Address<12>
21
Address Latch
Program Memory
(up to 2 Mbytes)
Data Latch
PCU
PCH
PCL
Program Counter
31 Level Stack
4
12
4
BSR
FSR0
FSR1
FSR2
Bank0, F
12
16
Table Latch
Decode
Inc/dec
logic
8
ROM Latch
Instruction
Register
Instruction
Decode &
Control
8
PRODH
PRODL
8 x 8 Multiply
3
OSC2/CLKO
OSC1/CLKI
12-bit
address
21-bit
address
Program
counter
Stack
Pointer

Χάρτης Μνήμης Προγράμματος και Στοίβας



21-bit Program counter μπορεί να προσπελάσει μέχρι 2 Mbyte θέσεις Μνήμης

Η Στοίβα έχει βάθος 31 επίπεδα.(PUSH -POP)

Αρχική Διεύθυνση μετά από RESET
00000H

Δύο INTERRUPT vectors
00008H
00018H

On Chip FLASH Memory
32 KByte

Δείκτης Στοίβας (STACK POINTER)

STKPTR REGISTER

R/C-0	R/C-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
STKOVF	STKUNF	—	SP4	SP3	SP2	SP1	SP0
bit 7							bit 0

Stack full bit

Stack underflow bit

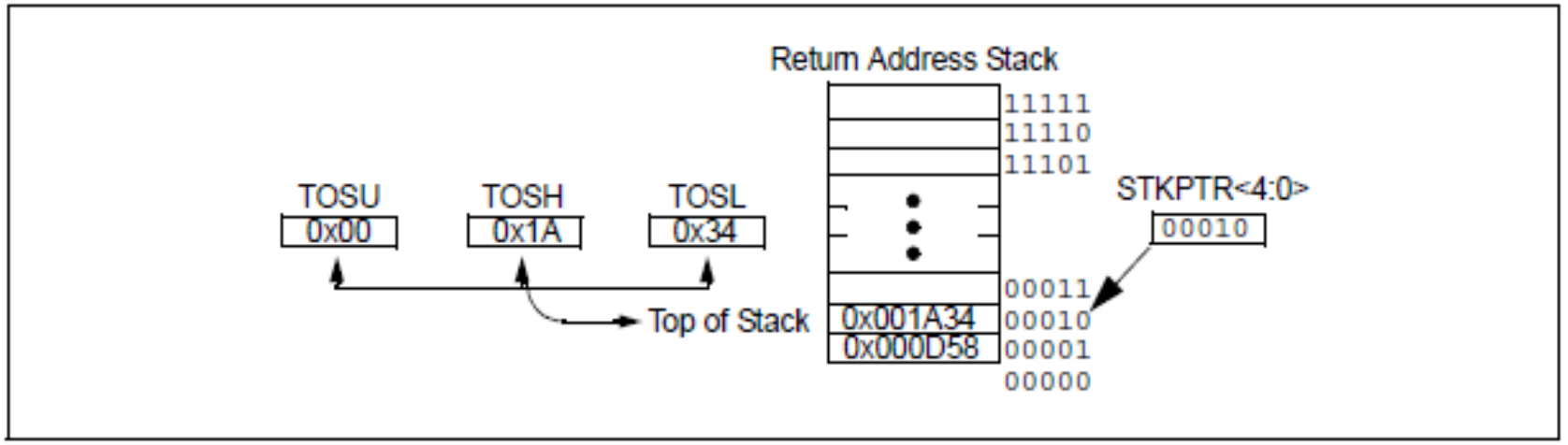
Τρέχουσα τιμή στοίβας

Αυξάνει με PUSH

Μειώνεται με POP

Στοίβα και σχετιζόμενοι καταχωρητές

FIGURE 4-3: RETURN ADDRESS STACK AND ASSOCIATED REGISTERS



Η Στοίβα έχει βαθος 31 επίπεδα και εύρος 21 bit.

Τρεις καταχωρητές TOSU, TOSH, TOSL έχουν τα περιεχόμενα της κορυφής της στοίβας.

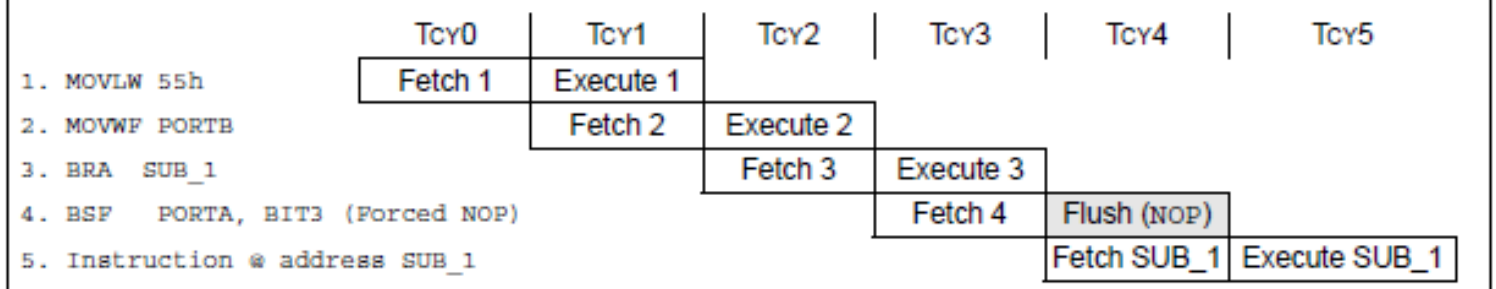
Μορφή και εκτέλεση εντολών

Οι Εντολές αποθηκεύονται στη μνήμη Προγράμματος σε 2 ή 4 συνεχόμενα Bytes.

Ο μετρητής προγράμματος (PC) αυξάνει πάντα κατά δυο.

Για την αποδοτικότερη λειτουργία του επεξεργαστή οι εντολές εκτελούνται σε pipeline.

EXAMPLE 4-2: INSTRUCTION PIPELINE FLOW



All instructions are single cycle, except for any program branches. These take two cycles since the fetch instruction is "flushed" from the pipeline while the new instruction is being fetched and then executed.

Παράδειγμα αποθήκευσης εντολών στην Μνήμη Προγράμματος

FIGURE 4-5: INSTRUCTIONS IN PROGRAM MEMORY

			Program Memory Byte Locations → LSB = 1	LSB = 0	Word Address ↓
					000000h
					000002h
					000004h
					000006h
Instruction 1:	MOVLW	055h	0Fh	55h	000008h
Instruction 2:	GOTO	000006h	EFh	03h	00000Ah
			F0h	00h	00000Ch
Instruction 3:	MOVFF	123h, 456h	C1h	23h	00000Eh
			F4h	56h	000010h
					000012h
					000014h

EXAMPLE 4-3: TWO-WORD INSTRUCTIONS

CASE 1:

Object Code	Source Code		
0110 0110 0000 0000	TSTFSZ	REG1	; is RAM location 0?
1100 0001 0010 0011	MOVFF	REG1, REG2	; No, execute 2-word instruction
1111 0100 0101 0110			; 2nd operand holds address of REG2
0010 0100 0000 0000	ADDWF	REG3	; continue code

CASE 2:

Object Code	Source Code		
0110 0110 0000 0000	TSTFSZ	REG1	; is RAM location 0?
1100 0001 0010 0011	MOVFF	REG1, REG2	; Yes
1111 0100 0101 0110			; 2nd operand becomes NOP
0010 0100 0000 0000	ADDWF	REG3	; continue code

Παράδειγμα (πηγαίος κώδικας)

```
char temp, temp1;

 void main(){
      TRISB =0xFF;
      TRISC =0x00;

      temp=18;
      temp1 = temp+PORTB;

 }
```

Παράδειγμα (assembly και μνήμη προγράμματος)

```
; ADDRESS OPCODE     ASM
; ------------------------------------------
$0000     $EF04     F000                GOTO      _main
$0008     $         _main:
;memory_map.c,3 ::            void main(){
;memory_map.c,5 ::            TRISB =0xFF;
$0008     $0EFF                         MOVLW     255
$000A     $6E93                         MOVWF     TRISB, 0
;memory_map.c,6 ::            TRISC =0x00;
$000C     $6A94                         CLRF      TRISC, 0
;memory_map.c,8 ::            temp=18;
$000E     $0E12                         MOVLW     18
$0010     $6E15                         MOVWF     _temp, 0
;memory_map.c,9 ::            temp1 = temp+PORTB;
$0012     $5081                         MOVF      PORTB, 0, 0
$0014     $0F12                         ADDLW     18
$0016     $6E16                         MOVWF     _temp1, 0
;memory_map.c,13 ::           }
$0018     $D7FF                         BRA       $

//** Procedures locations **
//ADDRESS           PROCEDURE
//------------------------------------------
$0008               main
```

GOTO	Unconditional Branch			
Syntax:	[*label*] GOTO k			
Operands:	$0 \le k \le 1048575$			
Operation:	$k \rightarrow PC<20:1>$			
Status Affected:	None			
Encoding:				
1st word (k<7:0>)	1110	1111	k_7kkk	$kkkk_0$
2nd word(k<19:8>)	1111	$k_{19}kkk$	kkkk	$kkkk_8$
Description:	GOTO allows an unconditional branch anywhere within entire 2 Mbyte memory range. The 20-bit value 'k' is loaded into PC<20:1>. GOTO is always a two-cycle instruction.			

MOVLW	Move literal to W
Syntax:	[*label*] MOVLW k
Operands:	$0 \le k \le 255$
Operation:	$k \rightarrow W$
Status Affected:	None
Encoding:	0000 1110 kkkk kkkk
Description:	The eight-bit literal 'k' is loaded into W.

MOVWF	Move W to f
Syntax:	[*label*] MOVWF f [,a]
Operands:	$0 \le f \le 255$ $a \in [0,1]$
Operation:	(W) $\rightarrow$ f
Status Affected:	None
Encoding:	0110 111a ffff ffff
Description:	Move data from W to register 'f'. Location 'f' can be anywhere in the 256 byte bank. If 'a' is 0, the Access Bank will be selected, over-riding the BSR value. If 'a' = 1, then the bank will be selected as per the BSR value (default).

Παράδειγμα (μνήμη δεδομένων)

```
//** Variables locations **
//ADDRESS                 VARIABLE
//-----------------------------------------
$0000                     STACK_0
$0001                     STACK_1
$0002                     STACK_2
$0003                     STACK_3
.............
$0012                     STACK_18
$0013                     STACK_19
$0014                     STACK_20
$0015                     _temp
$0016                     _temp1
$0F81                     PORTB
$0F93                     TRISB
$0F94                     TRISC
```

Αρχείο HEX με εκτελεσιμο κώδικα

```
:10000000 04EF00F0FFFFFFFFFF0E936E946A120EE5
:10001000 156E8150120F166EFFD7FFFFFFFFFFFF17
:020000040030CA
:0E000000FFFAFFFEFFFFFBFFFFFFFFFFFFFF0A
:00000001FF
```

Οργάνωση Μνήμης Δεδομένων

* Η μνήμη δεδομένων είναι υλοποιημένη ως στατική RAM.

* Κάθε θέση μνήμης προσδιορίζεται με 12-bit διεύθυνση επιτρέποντας μέχρι 4096 byte μνήμης.

* Η μνήμη είναι διαιρεμένη σε 16 τμήματα (banks) των 256 bytes. Τέσσερα bit ενός καταχωρητή (BSR) καθορίζουν το τμήμα.

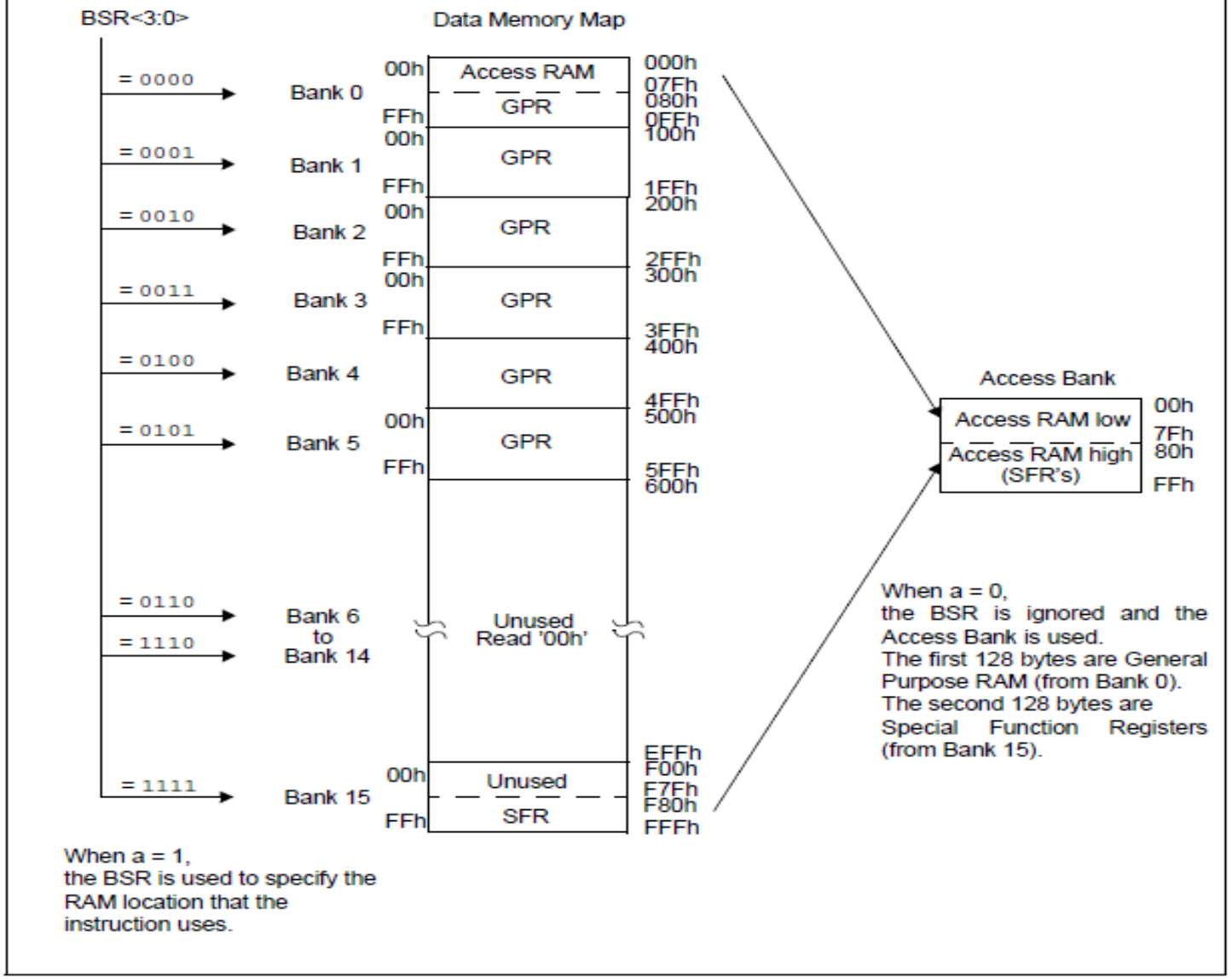
* Η Μνήμη δεδομένων περιλαμβάνει Καταχωρητές ειδικής Λειτουργίας (Special Function Registers) που χρησιμεύουν για έλεγχο του μικροελεγτή και περιφερειακών λειτουργιών.

* Οι SFR στις τελευταίες διευθύνσεις της 15^{ης} ομάδας μνήμης.

* Οι υπόλοιπες θεσεις μνήμης μπορούν να χρησιμοποιηθούν ως Γενικοί Καταχωρητές (GPR)

Οργάνωση Μνήμης Δεδομένων

FIGURE 4-7: DATA MEMORY MAP FOR PIC18F252/452



Πίνακας καταχωρητών ειδικού σκοπού

TABLE 4-1: SPECIAL FUNCTION REGISTER MAP

Address	Name	Address	Name	Address	Name	Address	Name
FFFh	TOSU	FDFh	INDF2(3)	FBFh	CCPR1H	F9Fh	IPR1
FFEh	TOSH	FDEh	POSTINC2(3)	FBEh	CCPR1L	F9Eh	PIR1
FFDh	TOSL	FDDh	POSTDEC2(3)	FBDh	CCP1CON	F9Dh	PIE1
FFCh	STKPTR	FDCh	PREINC2(3)	FBCh	CCPR2H	F9Ch	—
FFBh	PCLATU	FDBh	PLUSW2(3)	FBBh	CCPR2L	F9Bh	—
FFAh	PCLATH	FDAh	FSR2H	FBAh	CCP2CON	F9Ah	—
FF9h	PCL	FD9h	FSR2L	FB9h	—	F99h	—
FF8h	TBLPTRU	FD8h	STATUS	FB8h	—	F98h	—
FF7h	TBLPTRH	FD7h	TMR0H	FB7h	—	F97h	—
FF6h	TBLPTRL	FD6h	TMR0L	FB6h	—	F96h	TRISE(2)
FF5h	TABLAT	FD5h	T0CON	FB5h	—	F95h	TRISD(2)
FF4h	PRODH	FD4h	—	FB4h	—	F94h	TRISC
FF3h	PRODL	FD3h	OSCCON	FB3h	TMR3H	F93h	TRISB
FF2h	INTCON	FD2h	LVDCON	FB2h	TMR3L	F92h	TRISA
FF1h	INTCON2	FD1h	WDTCON	FB1h	T3CON	F91h	—
FF0h	INTCON3	FD0h	RCON	FB0h	—	F90h	—
FEFh	INDF0(3)	FCFh	TMR1H	FAFh	SPBRG	F8Fh	—
FEEh	POSTINC0(3)	FCEh	TMR1L	FAEh	RCREG	F8Eh	—
FEDh	POSTDEC0(3)	FCDh	T1CON	FADh	TXREG	F8Dh	LATE(2)
FECh	PREINC0(3)	FCCh	TMR2	FACh	TXSTA	F8Ch	LATD(2)
FEBh	PLUSW0(3)	FCBh	PR2	FABh	RCSTA	F8Bh	LATC
FEAh	FSR0H	FCAh	T2CON	FAAh	—	F8Ah	LATB
FE9h	FSR0L	FC9h	SSPBUF	FA9h	EEADR	F89h	LATA
FE8h	WREG	FC8h	SSPADD	FA8h	EEDATA	F88h	—
FE7h	INDF1(3)	FC7h	SSPSTAT	FA7h	EECON2	F87h	—
FE6h	POSTINC1(3)	FC6h	SSPCON1	FA6h	EECON1	F86h	—
FE5h	POSTDEC1(3)	FC5h	SSPCON2	FA5h	—	F85h	—
FE4h	PREINC1(3)	FC4h	ADRESH	FA4h	—	F84h	PORTE(2)
FE3h	PLUSW1(3)	FC3h	ADRESL	FA3h	—	F83h	PORTD(2)
FE2h	FSR1H	FC2h	ADCON0	FA2h	IPR2	F82h	PORTC
FE1h	FSR1L	FC1h	ADCON1	FA1h	PIR2	F81h	PORTB
FE0h	BSR	FC0h	—	FA0h	PIE2	F80h	PORTA

EEPROM ΜΝΗΜΗ ΔΕΔΟΜΕΝΩΝ

Η EERPOM δεδομένων είναι αναγνώσιμη και εγγράψιμη κατά την λειτουργία του μικροελεγκτή.

Δεν είναι προσβάσιμη στον χώρο της RAM αλλά μπορεί να προσπελαστεί έμμεσα μέσω καταχωρητών ειδικού σκοπού:

EECON1

EECON2

EEDATA → 8-bit δεδομένα EEPROM

EEADR -> 8-bit διεύθυνση EEPROM

EEPROM ΜΝΗΜΗ ΔΕΔΟΜΕΝΩΝ

EXAMPLE 6-1: DATA EEPROM READ

```
        MOVLW   DATA_EE_ADDR    ;
        MOVWF   EEADR           ; Data Memory Address to read
        BCF     EECON1, EEPGD   ; Point to DATA memory
        BCF     EECON1, CFGS    ; Access program FLASH or Data EEPROM memory
        BSF     EECON1, RD      ; EEPROM Read
        MOVF    EEDATA, W       ; W = EEDATA
```

EXAMPLE 6-2: DATA EEPROM WRITE

```
                MOVLW   DATA_EE_ADDR    ;
                MOVWF   EEADR           ; Data Memory Address to read
                MOVLW   DATA_EE_DATA    ;
                MOVWF   EEDATA          ; Data Memory Value to write
                BCF     EECON1, EEPGD   ; Point to DATA memory
                BCF     EECON1, CFGS    ; Access program FLASH or Data EEPROM memory
                BSF     EECON1, WREN    ; Enable writes

                BCF     INTCON, GIE     ; Disable interrupts
  Required      MOVLW   55h             ;
  Sequence      MOVWF   EECON2          ; Write 55h
                MOVLW   AAh             ;
                MOVWF   EECON2          ; Write AAh
                BSF     EECON1, WR      ; Set WR bit to begin write
                BSF     INTCON, GIE     ; Enable interrupts

                .                       ; user code execution
                .
                .
                BCF     EECON1, WREN    ; Disable writes on write complete (EEIF set)
```

ΠΑΡΑΔΕΙΓΜΑ ΕΓΓΡΑΦΗΣ ΚΑΙ ΑΝΑΓΝΩΣΗΣ ΑΠΌ EEPROM

```
unsigned short i = 0, j = 0;

void main() {
  PORTB = 0;
  TRISB = 0;

  j = 4;
  for (i = 0; i < 20u; i++)
    EEprom_Write(i, j++);

  for (i = 0; i < 20u; i++) {
    PORTB = EEprom_Read(i);
    Delay_ms(500);
  }
}//~!
```