Μικροελεγκτές

Έναν ορισμό που θα μπορούσαμε να δώσουμε για τους μικροελεγκτές είναι ο εξής:
Μικροελεγκτής είναι ένα προγραμματιζόμενο ολοκληρωμένο κύκλωμα το οποίο διαθέτει επεξεργαστή, μνήμη, διάφορα περιφερειακά κυκλώματα καθώς επίσης και θύρες εισόδου/εξόδου για επικοινωνία με εξωτερικές συσκευές.
Θα μπορούσε να παρομοιαστεί με έναν μικροϋπολογιστή. Όπως ακριβώς ένας μικροϋπολογιστής έχει επεξεργαστή, μνήμη, περιφερειακές συσκευές και εκτελεί προγράμματα έτσι κι ένας μικροελεγκτής διαθέτει τα παραπάνω χαρακτηριστικά και μάλιστα ολοκληρωμένα σε ένα μόνο chip.
Το πρόγραμμα που εκτελεί ο μικροελεγκτής αποθηκεύεται μόνιμα στη μνήμη προγράμματος.

Εφαρμογές μικροελεγκτών

Οι μικροελεγκτές βρίσκουν εφαρμογή στα παρακάτω πεδία:

– Σε συστήματα αυτοματισμών
– Σε κυκλώματα τηλεπικοινωνιών
– Στις ηλεκτρονικές συσκευές
– Στις ηλεκτρικές συσκευές
– Σε συστήματα τηλεματικής
– Σε συστήματα συλλογής δεδομένων (Data Acquisition)
– Σε εφαρμογές ηλεκτρονικών ισχύος
– Σε συστήματα διασύνδεσης
– Σε εφαρμογές δικτύων

Γενικότερα οι μικροελεγκτές χρησιμοποιούνται οπουδήποτε απαιτείται έλεγχος συστημάτων.

Όταν λέμε ενσωματωμένα συστήματα (**Embedded Systems**) εννοούμε συστήματα τα οποία είναι βασισμένα σε μικροεπεξεργαστή (ή επίσης FPGA ή DSP)

Κατασκευαστές μικροελεγκτών

Υπάρχουν δεκάδες εταιρείες παγκοσμίως που κατασκευάζουν μικροελεγκτές.
Οι πιο διαδεδομένες είναι:
– Microchip, www.microchip.com
– Atmel, www.atmel.com
– Texas Instruments, www.ti.com
– Freescale (πρώην Motorolla), www.freescale.com
– Intel, www.intel.com
– Analog Devices, www.analog.com
Οι περισσότερες εταιρείες παράγουν μεγάλη γκάμα μικροελεγκτών. Από πολύ μικρούς και φθηνούς για απλές εφαρμογές έως ιδιαίτερα προηγμένους για πολύ απαιτητικές εφαρμογές.

Γλώσσα προγραμματισμού μικροελεγκτών

Οι μικροελεγκτές γενικά προγραμματίζονται σε γλώσσες χαμηλού επιπέδου. Τελευταία όλο και περισσότεροι προγραμματιστές επιλέγουν γλώσσες υψηλότερο επιπέδου.

Ως γλώσσα χαμηλού επιπέδου ονομάζεται μια γλώσσα η οποία βρίσκεται πιο κοντά στο υλικό (γλώσσα μηχανής, assembly)

Ως γλώσσα υψηλού επιπέδου ονομάζεται μια γλώσσα η οποία είναι αυστηρά δομημένη και υπάρχει συγκεκριμένος compiler ο οποίος μετατρέπει το πρόγραμμα σε γλώσσα μηχανής για το συγκεκριμένο μικροελεγκτή.

Γλώσσα προγραμματισμού μικροελεγκτών

Πλεονεκτήματα γλωσσών χαμηλού επιπέδου:

– Ο προγραμματιστής έχει τον απόλυτο έλεγχο της συμπεριφοράς του μικροελεγκτή

– Μπορεί να επιτύχει με απόλυτη ακρίβεια διάφορους χρονισμούς

– Δεν απαιτείται η δαπάνη για την αγορά assembler καθώς συνήθως διατίθεται δωρεάν από την κατασκευάστρια εταιρεία

Μειονεκτήματα γλωσσών χαμηλού επιπέδου:

– Απαιτείται μεγαλύτερος κόπος για την εκμάθηση της συμβολικής γλώσσας του εκάστοτε μικροελεγκτή

– Τα προγράμματα που δημιουργούνται σε συμβολική γλώσσα δεν είναι ευανάγνωστα και ο προγραμματιστής δυσκολεύεται να θυμηθεί τη λογική που έχει εφαρμόσει όταν χρειάζεται να κάνει τροποποιήσεις εκ των υστέρων

– Είναι δυσκολότερο να δουλέψουν πολλοί προγραμματιστές στο ίδιο πρόγραμμα

Γλώσσα προγραμματισμού μικροελεγκτών

Πλεονεκτήματα γλωσσών υψηλού επιπέδου:

– Είναι ευκολότερη η ανάπτυξη μεγάλων και σύνθετων προγραμμάτων

– Μπορούν να δουλέψουν πιο εύκολα πολλοί προγραμματιστές στο ίδιο πρόγραμμα

Μειονεκτήματα γλωσσών υψηλού επιπέδου:

– Σε εφαρμογές με κρίσιμους χρονισμούς είναι δυσκολότερη η συγγραφή κώδικα που ανταποκρίνεται στους χρονισμούς αυτούς

– Μερικές φορές η δαπάνη για την αγορά compiler δεν αποτελεί αμελητέο μέγεθος

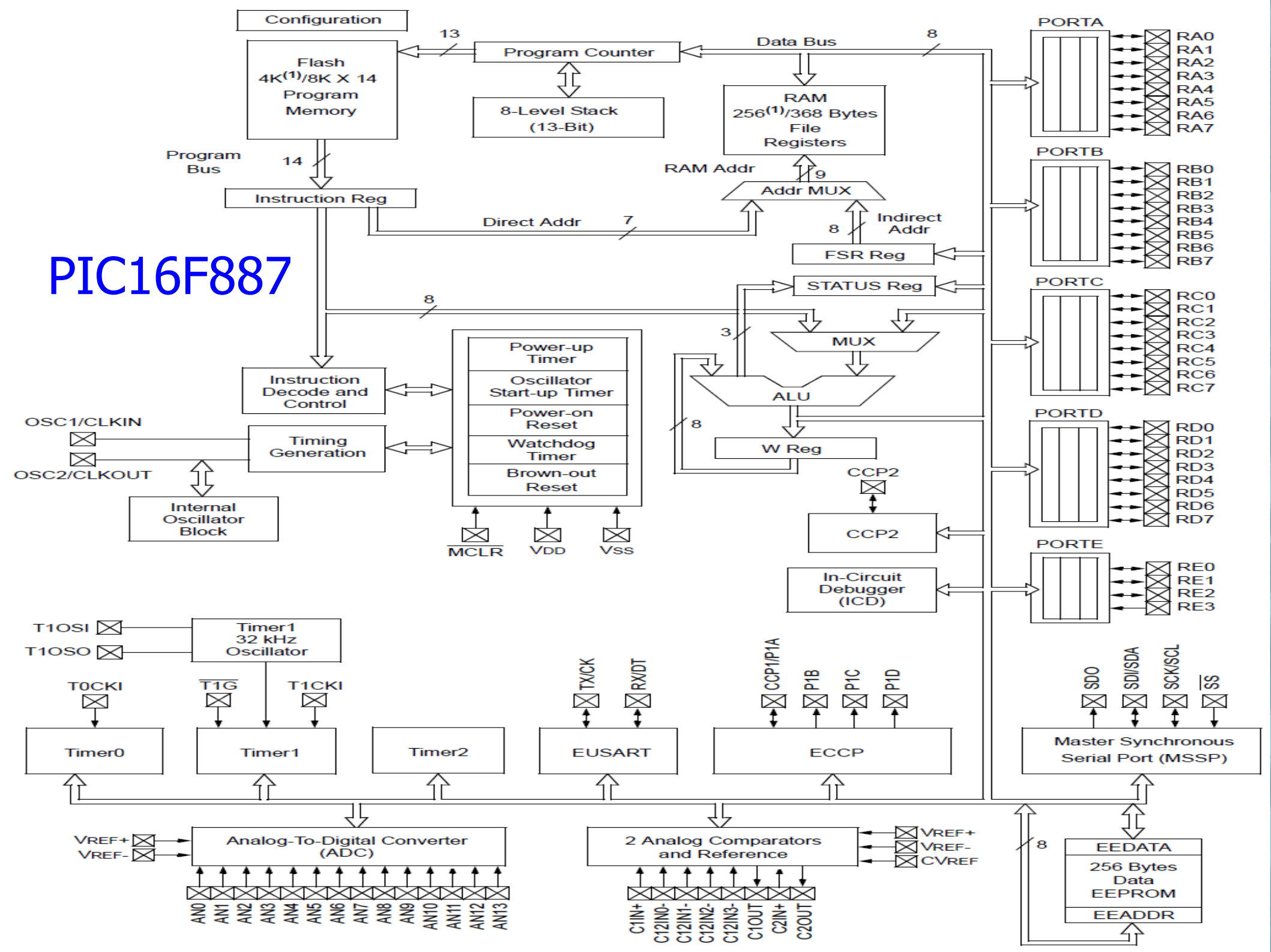
– Σε παλιότερους compilers ο κώδικας μηχανής που παραγόταν δεν ήταν βελτιστοποιημένος με αποτέλεσμα να απαιτείται μικροελεγκτής με πολύ περισσότερη μνήμη. Οι compilers που κυκλοφορούν σήμερα διαθέτουν εξελιγμένα εργαλεία για βελτιστοποίηση (optimization) του κώδικα και έχουν κερδίσει την εμπιστοσύνη ακόμα και των πιο δύσπιστων προγραμαμτιστών.

Μικροελεγκτές PIC

Η Microchip διαθέτει πολύ μεγάλη ποικιλία μικροελεγκτών.
– 8-bit μικροελεγκτές (οικογένειες PIC10, PIC12, PIC16, PIC18)
– 16-bit μικροελεγκτές (οικογένειες PIC24, dsPIC)
– 32-bit μικροελεγκτές (οικογένειες PIC32)

Η επιλογή ενός μικροελεγκτή για μια εφαρμογή γίνεται με την εξής διαδικασία.
Καθορίζονται οι προδιαγραφές που πρέπει να πληρεί ο μικροελεγκτής και επιλέγεται ο φθηνότερος που ανταποκρίνεται στις προδιαγραφές αυτές.

Ο μικροελεγκτής που θα μελετηθεί διεξοδικά στα πλαίσια του μαθήματος θα είναι ο PIC16F887.


PIC16F887
Configuration
Flash
4K(1)/8K X 14
Program
Memory
13
Program Counter
8-Level Stack
(13-Bit)
Data Bus
8
RAM
256(1)/368 Bytes
File
Registers
Program
Bus
14
Instruction Reg
RAM Addr
9
Addr MUX
Direct Addr
7
8
Indirect
Addr
FSR Reg
STATUS Reg
8
3
MUX
ALU
8
W Reg
Instruction
Decode and
Control
Timing
Generation
Internal
Oscillator
Block
OSC1/CLKIN
OSC2/CLKOUT
Power-up
Timer
Oscillator
Start-up Timer
Power-on
Reset
Watchdog
Timer
Brown-out
Reset
MCLR
VDD
VSS
CCP2
CCP2
In-Circuit
Debugger
(ICD)
PORTA
RA0
RA1
RA2
RA3
RA4
RA5
RA6
RA7
PORTB
RB0
RB1
RB2
RB3
RB4
RB5
RB6
RB7
PORTC
RC0
RC1
RC2
RC3
RC4
RC5
RC6
RC7
PORTD
RD0
RD1
RD2
RD3
RD4
RD5
RD6
RD7
PORTE
RE0
RE1
RE2
RE3
T1OSI
T1OSO
Timer1
32 kHz
Oscillator
T0CKI
T1G
T1CKI
Timer0
Timer1
Timer2
TX/CK
RX/DT
EUSART
CCP1/P1A
P1B
P1C
P1D
ECCP
SDO
SDI/SDA
SCK/SCL
SS
Master Synchronous
Serial Port (MSSP)
VREF+
VREF-
Analog-To-Digital Converter
(ADC)
AN0
AN1
AN2
AN3
AN4
AN5
AN6
AN7
AN8
AN9
AN10
AN11
AN12
AN13
2 Analog Comparators
and Reference
VREF+
VREF-
CVREF
C1IN+
C12IN0-
C12IN1-
C12IN2-
C12IN3-
C1OUT
C2IN+
C2OUT
8
EEDATA
256 Bytes
Data
EEPROM
EEADDR

PIC16F887

– Είναι μικροελεγκτής με δίαυλο δεδομένων 8-bit
– Έχει δίαυλο διευθύνσεων 9-bit
– Λειτουργεί με ταλαντωτή χρονισμού μέγιστης συχνότητας 20MHz (εσωτερικά διαιρεί με το 4 για να παράγει τελικά 5MHz κύκλο ρολογιού)
– Διαθέτει μνήμη προγράμματος 8K λέξεων
– Διαθέτει RAM 368 bytes
– Διαθέτει στοίβα 8 θέσεων
– Διαθέτει EEPROM δεδομένων 256 θέσεων
– Διαθέτει 5 θύρες I/O (PORTA,PORTB,PORTC,PORTD,PORTE)
– Διαθέτει ενσωματωμένα διάφορα περιφερειακά κυκλώματα όπως Timers, ADC, MSSP (SPI/I^2C), USART, CCP κλπ.

Ο επεξεργαστής είναι τύπου RISC και υποστηρίζει σύνολο 35 εντολών. Κάθε εντολή εκτελείται σε ένα παλμό ρολογιού πλην των εντολών διακλάδωσης που εκτελούνται σε δύο παλμούς.

Χάρτης μνήμης PIC16F887

Bank 0	File Address	Bank 1	File Address	Bank 2	File Address	Bank 3	File Address
Indirect addr. (1)	00h	Indirect addr. (1)	80h	Indirect addr. (1)	100h	Indirect addr. (1)	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h	WDTCON	105h	SRCON	185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h	CM1CON0	107h	BAUDCTL	187h
PORTD(2)	08h	TRISD(2)	88h	CM2CON0	108h	ANSEL	188h
PORTE	09h	TRISE	89h	CM2CON1	109h	ANSELH	189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDAT	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2(1)	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Reserved	18Eh
TMR1H	0Fh	OSCCON	8Fh	EEADRH	10Fh	Reserved	18Fh
T1CON	10h	OSCTUNE	90h		110h		190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h	WPUB	95h		115h		195h
CCPR1H	16h	IOCB	96h	General Purpose Registers 16 Bytes	116h	General Purpose Registers 16 Bytes	196h
CCP1CON	17h	VRCON	97h		117h		197h
RCSTA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h		119h		199h
RCREG	1Ah	SPBRGH	9Ah		11Ah		19Ah
CCPR2L	1Bh	PWM1CON	9Bh		11Bh		19Bh
CCPR2H	1Ch	ECCPAS	9Ch		11Ch		19Ch
CCP2CON	1Dh	PSTRCON	9Dh		11Dh		19Dh
ADRESH	1Eh	ADRESL	9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
General Purpose Registers 96 Bytes	20h – 3Fh, 40h – 6Fh	General Purpose Registers 80 Bytes	A0h – EFh	General Purpose Registers 80 Bytes	120h – 16Fh	General Purpose Registers 80 Bytes	1A0h – 1EFh
accesses 70h-7Fh	70h – 7Fh	accesses 70h-7Fh	F0h – FFh	accesses 70h-7Fh	170h – 17Fh	accesses 70h-7Fh	1F0h – 1FFh

Χάρτης μνήμης PIC16F887

– Περιλαμβάνει 4 μπλοκ (ομάδες) καταχωρητών όπου το κάθε ένα από αυτά ονομάζεται Bank.

Bank0: h'00' – h'7F'
Bank1: h'80' – h'FF'
Bank2: h'100' – h'17F'
Bank3: h'180' – h'1FF'

Κάποιοι καταχωρητές της RAM αποτελούν τους καταχωρητές ειδικής χρήσης (Special Function Registers - SFRs) και οι τιμές τους καθορίζουν τη συμπεριφορά περιφερειακών του μικροελεγκτή.

Οι υπόλοιποι καταχωρητές αποτελούν τους καταχωρητές γενικής χρήσης (General Purpose Registers – GPRs) και μπορούν να χρησιμοποιηθούν από τον προγραμματιστή για την αποθήκευση δεδομένων.

Bank0 GPRs: h'20' – h'7F'
Bank1 GPRs: h'A0' – h'EF'
Bank2 GPRs: h'110' – h'16F'
Bank3 GPRs: h'190' – h'1EF'

Ο καταχωρητής STATUS

REGISTER 2-1: STATUS: STATUS REGISTER

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC(1)	C(1)
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **IRP:** Register Bank Select bit (used for indirect addressing)
1 = Bank 2, 3 (100h-1FFh)
0 = Bank 0, 1 (00h-FFh)

bit 6-5 **RP<1:0>:** Register Bank Select bits (used for direct addressing)
00 = Bank 0 (00h-7Fh)
01 = Bank 1 (80h-FFh)
10 = Bank 2 (100h-17Fh)
11 = Bank 3 (180h-1FFh)

bit 4 **$\overline{TO}$:** Time-out bit
1 = After power-up, CLRWDT instruction or SLEEP instruction
0 = A WDT time-out occurred

bit 3 **$\overline{PD}$:** Power-down bit
1 = After power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction

bit 2 **Z:** Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1 **DC:** Digit Carry/$\overline{Borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)(1)
1 = A carry-out from the 4th low-order bit of the result occurred
0 = No carry-out from the 4th low-order bit of the result

bit 0 **C:** Carry/$\overline{Borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)(1)
1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

Σύνολο εντολών PIC16F887 (Instruction Set)

BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C, DC, Z	1, 2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1, 2
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z	2
CLRW	–	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1, 2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1, 2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1, 2, 3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1, 2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1, 2, 3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1, 2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1, 2
MOVWF	f	Move W to f	1	00	0000	lfff	ffff		
NOP	–	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1, 2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1, 2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C, DC, Z	1, 2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1, 2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1, 2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1, 2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1, 2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C, DC, Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call Subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	–	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$, $\overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	–	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	–	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	–	Go into Standby mode	1	00	0000	0110	0011	$\overline{TO}$, $\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C, DC, Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

Εντολές PIC16F887

1. clrf <όνομα καταχωρητή>
Η εντολή πραγματοποιεί μηδενισμό του καταχωρητή που ακολουθεί. Επηρεάζει το Zero flag.
π.χ.: clrf Register1
Μετά την εκτέλεση ο Register1 θα έχει την τιμή μηδέν.

2. movlw <αριθμητική τιμή>
Η εντολή φορτώνει στον συσσωρευτή W την αριθμητική τιμή που ακολουθεί.
π.χ.: movlw h'64'
Μετά την εκτέλεση ο W θα έχει την τιμή
$64_{16}=100_{10}$

Εντολές PIC16F887

3. movwf <όνομα καταχωρητή>
Η εντολή αποθηκεύει τα περιεχόμενα του συσσωρευτή W στον καταχωρητή που ακολουθεί
π.χ.: movwf Register1
Μετά την εκτέλεση ο Register1 θα έχει όποια τιμή είχε προηγουμένως ο W.

4. movf <όνομα καταχωρητή>,a
a=f ή a=w
Η εντολή διαβάζει τα περιεχόμενα του καταχωρητή που ακολουθεί και αν a=f τα αποθηκεύει στον εαυτό του ενώ αν a=w τα αποθηκεύει στον συσσωρευτή W.
Επηρεάζει το Zero flag.
π.χ.: movf Register1,w
Μετά την εκτέλεση ο W θα έχει όποια δεδομένα είχε ο Register1 προηγουμένος.

Εντολές PIC16F887

5. goto <ετικέτα>
Η εκτολή πραγματοποιεί άλμα του PC στη διεύθυνση μνήμης που αναπαριστάνει η ετικέτα.

6. addlw <αριθμητική τιμή>
Η εντολή προσθέτει τα περιεχόμενα του W με την αριθμητική τιμή που ακολουθεί και αποθηκεύει το άθροισμα στον W. Επηρεάζει το Carry flag και το Zero flag.
π.χ.: addlw d'25'
Μετά την εκτέλεση ο W θα έχει το άθροισμα ανάμεσα στο 25 και στην τιμή που είχε προηγουμένως.
W = d'20'
addlw d'25'
W = d'45'

Εντολές PIC16F887

7. addwf <όνομα καταχωρητή>,a
a=f ή a=w
Η εντολή προσθέτει τα περιεχόμενα του καταχωρητή που ακολουθεί με τα περιεχόμενα του W και αν a=f αποθηκεύει το άθροισμα στον καταχωρητή ενώ αν a=w το αποθηκεύει στο συσσωρευτή W. Επηρεάζει το Carry και το Zero flag.
π.χ.: addwf Register1,w
Μετά την εκτέλεση ο W θα έχει το άθροισμα ανάμεσα στα δεδομένα που είχε προηγουμένως και στα δεδομένα του Register1.

```
W = d'25'
Register1 = d'20'
addwf Register1,w
W = d'45'
Register1 = d'20'
```

```
W = d'25'
Register1 = d'20'
addwf Register1,f
W = d'25'
Register1 = d'45'
```

Παραδείγματα

1. Να φορτώσετε την τιμή d'200' στον W.

```
movlw d'200'
```

2. Να αποθηκεύσετε τα περιεχόμενα του W στον καταχωρητή Reg1.

```
movwf Reg1
```

3. Να προσθέσετε το d'50' στα περιεχόμενα του καταχωρητή W και να αποθηκεύσετε το αποτέλεσμα στον καταχωρητή Reg1.

```
addlw d'50'
movwf Reg1
```

4. Να μηδενίσετε τα περιεχόμενα το καταχωρητή Reg1.

```
clrf Reg1
```

Παραδείγματα

5. Να φορτώσετε το d'100' στον καταχωρητή Reg1, το d'150' στον W και να προσθέσετε Reg1 και W. Το άθροισμα να αποθηκευτεί στον Reg1.

```
movlw d'100'
movwf Reg1
movlw d'50'
addwf Reg1,f
```

6. Να πραγματοποιήσετε την πράξη της πρόσθεσης 20+30+40+50 και να αποθηκεύσετε το τελικό άθροισμα στον καταχωρητή Reg1.

```
movlw d'20'
addlw d'30'
addlw d'40'
addlw d'50'
movwf Reg1
```

Παραδείγματα

7. Να φορτώσετε το d'44' στον καταχωρητή Reg1, το d'88' στον Reg2, το d'22' στον Reg3 και να προσθέσετε Reg1, Reg2 και Reg3. Το άθροισμα να αποθηκευτεί στον Reg1.

```
movlw d'44'       ;W       <= 44
movwf Reg1        ;Reg1    <= 44
movlw d'88'       ;W       <= 88
movwf Reg2        ;Reg2    <= 88
movlw d'22'       ;W       <= 22
movwf Reg3        ;Reg3    <= 22
movf Reg2,w       ;W       <= 88
addwf Reg3,w      ;W       <= 110
addwf Reg1,f      ;Reg1    <= 154
```

Σημ.: Κάθε πράξη στην αριθμητική λογική μονάδα υποστηρίζεται ανάμεσα στο συσσωρευτή W και σε κάποιο καταχωρητή μνήμης. Δεν υποστηρίζεται απευθείας πράξη ανάμεσα σε περιεχόμενα δύο θέσεων μνήμης.

Παράδειγμα προγράμματος

Να γίνει ένα πρόγραμμα που ορίζει δύο καταχωρητές Reg1, Reg2 σε ελεύθερες θέσεις μνήμης, φορτώνει στους καταχωρητές αυτούς τις τιμές 100 και 150 αντίστοιχα και πραγματοποιεί πρόσθεση στα περιεχόμενά τους. Το αποτέλεσμα αποθηκεύεται στον W.

```
  include <p16f887.inc>
Reg1  equ   h'20'
Reg2  equ   h'21'
  org h'00'
  movlw d'100'
  movwf Reg1
  movlw d'100'
  movwf Reg2
  addwf Reg1,w
LOOP
  goto LOOP
  end
```

Επεξήγηση οδηγιών προς τον assembler

```
include <p16f887.inc>
```

Δήλωση αρχείου συμβολικών ονομάτων καταχωρητών ειδικής χρήσης του μικροελεγκτή.

```
Reg1  equ   h'20'
Reg2  equ   h'21'
```

Δήλωση συμβολικών ονομάτων καταχωρητών γενικής χρήσης στις διευθύνσεις της μνήμης RAM h'20' και h'21'.

```
org h'00'
```

Οδηγία προς τον assembler να τοποθετήσει τον κώδικα που ακολουθεί από τη θέση μνήμης h'00' και μετά.

```
end
```

Οδηγία προς τον assembler ότι τελειώνει το αρχείο πηγαίου κώδικα.