Université de Jijel.
Faculté des sciences et de la technologie
Département d'électrotechnique
1ière Année Master électrotechnique

Travaux pratiques de Microprocesseur

Le jeu d'instructions du PIC 16F877

I. Introduction

Chaque microprocesseur possède son propre **langage** représenté par un ensemble d'instructions binaires. C'est ce que l'on appelle le jeu d'instructions. Chaque instruction sous forme binaire possède alors une **longueur** (nombre de bit) bien déterminé et obéit à un **format spécial** propre au microprocesseur : **C'est son langage machine.**

l'instruction est divisées en deux champs :

Le champ **code opération** qui désigne le type d'opération à effectuer

Le champ **opérandes** qui détermine les données sur lequelles l'opération doit être réalisée

Par exemple l'instruction suivante du pic 16F877,:

00 0111 1001 1010

Est divisée en deux parties :

Les six bit de poids le plus fort (00 0111) représentent une **opération d'adition**.

Les huit bit de poids le plus faible (1001 1010) représente deux opérandes

Il est bien évident, dans ce cas, que c'est difficile de manipuler le langage machine et d'écire des programmes avec ce langage. Une simple erreur sur un seul bit n'est pas facile à détecter.

II. Le langage assembleur

Afin d'éviter de travailler avec le langage machine, les programmateurs ont eu l'idée d'inventer un langage indépendant du microprocesseur où chaque **opération** est représentée par un **mnémonique** ; une sorte de mot désignant l'opération et que l'on peut facilement retenir.

Par exemple :

Une opération d'**addition** est désignée par le mnémonique **ADD**, Une opération de **soustraction** par **SUB**, une opération de **multiplication** par **MULT**…

On peut alors écrire par exemple :

ADD X1,X2 si l'on veut additionner les opérandes X1 et X2

SUB R1,R2 si l'on veut soustraire le registre R2 du registre R1

Malheureusement, chaque famille de programmeurs a développé son propre langage assembleur, et on se retrouve avec autant de langages assembleur qu'il existe de famille de microprocesseur, avec des différences plus ou moins importantes.

III. Le jeu d'instruction du PIC 16F877

Les instructions du PIC 16F877 s'effectuent soit :

- Entre registre de travail **w** (work) et un registre mémoire **f** (File)
- Entre le registre de travail **w** et une valeur donnée **k** (Literal)
- Sur le seul registre **f**
- Sur le seul registre **w**

La table 13-2 présente le jeu d'instructions en assembleur du PIC 16F877.

- La première colonne présente l'instruction en assembleur avec ses deux champs, code opération et opérandes
- La seconde colonne présente une description sommaire de l'opération réalisé par chaque instruction
- La troisième colonne donne le temps d'exécution de chaque instruction en nombre de cycles machines
- La quatrième colonne présente l'instruction machine en binaire. On peut voir q'une instruction possède une **longueur de 14 bit**.
- La cinquième colonne présente les bit d'état affectés par l'opération

Concernant les opérandes, il sont divisés en **quatre types**

f (File): qui représente l'un des registres mémoire indique dans la figure 2-3. C'est une valeur codée sur 7 bit qui peut varier entre 0x00 et 0x7F. il s'agit d'une **adresse** en **mémoire données.**

d (destination) : désigne l'emplacement de stockage du résultat de l'opération. Cest une valeur codée sur un bit. Il peut être soit à 0 soit à 1

d = 0 : le résultat de l'opération est mis dans le registre w

d = 1 : le résultat de l'opération est mis dans le registre f indiqué

b (bit) : désigne le numéro d'un bit du registre f indiqué dans l'instruction. Il est codé sur trois bit, il peut prendre donc les valeurs entre 0 et 7

k (?) : désigne une valeur codée sur 8 ou sur 11 bit selon les instructions concernées.

Il existe aussi six instructions qui ne nécessitent pas d'opérandes (CLRW, NOP, CLRWDT, RETFIE, RETURN, SLEEP)

Les insructions sont divisé en trois types

a. Les instructions portant sur les registres (byte-oriented file register operations)

Elles sont au nombre de dix-huit. Ces instructions manipulent des **données de 8 bit** (octet) stockées dans une **adresse mémoire** (file). Leur code-opération débute toujours par (00), puis un code sur quatre ou cinq bit selon l'opération.

Par exemple :

ADDWF	correspond au code	00 0111
CLRF	correspond au code	00 0001 1

b. Les instruction portant sur les bit (bit-oriented file register instructions)

Elle sont au nombre de quatre. Elle manipule **un bit** d'une donnée stocké dans une **adresse mémoire** (file). Leur code-opération débute toujours par (01), puis un code sur deux bit.

Exemple :

BCF	correspond au code	01 00
BTFSC	correspond au code	01 10

c. Les instructions de contrôle et portant sur des valeurs (Literal and control operation)

Ici nous avons deux types d'instructions ; celle qui réalisent un contrôle (branchement, appel/retour de sous programme). Ces instruction commensent par l'un des codes (00, 10), et les instructions qui manipulent des **valeurs immédiates** ; c'est-à-dire des valeurs que l'on fournie dans l'instruction après le code opération. Ces instructions commensent avec le code (11).

IV. La mémoire données

Le PIC16F877 possède dex types de mémoires :

- Une **mémoire programme** où sont stockées **les instructions** (programme)
- Et une **mémoire données** où sont stockées **les données** traitées par le programme

Pour programmer le PIC16F877 il est important de connaitre la structure de la mémoire données.

La figure 2-3 présente la mémoire données. Elle se divise en **quatre banques** de **128 adresses** chacune.

Certains emplacements portent un nom particulier ce qui veut dire qu'il ont **un effet sur le comportement du PIC16F877 ce sont des registres spéciaux (Special Function Register)**. Ces emplacement sont à eviter sauf si on connait leur effet. Les emplacements qui sont **en gris** n'existent pas en réalité ou sont résérvés, il ne faut donc **jamais les utiliser**.

Le reste de la mémoire peut être librement utilisée (General Purpose Register) pour le stockage des données à traiter et pour les résultats des traitements.

La plage d'adressses entre 70h et 7Fh est partagée avec les autres banques. Par exemple les adresses 70h, F0h,170h et 1F0h représentent en réalité la même case mémoire.

FIGURE 2-3: PIC16F877/876 REGISTER FILE MAP

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

Unimplemented data memory locations, read as '0'.
* Not a physical register.

Note 1: These registers are not implemented on the PIC16F876.
2: These registers are reserved, maintain these registers clear.

TABLE 13-2: PIC16F87X INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
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	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 module.

3: If Program Counter (PC) is modified, or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

ADDLW **Add Literal and W**

Syntax: [*label*] ADDLW k
Operands: $0 \le k \le 255$
Operation: $(W) + k \rightarrow (W)$
Status Affected: C, DC, Z
Description: The contents of the W register are added to the eight bit literal 'k' and the result is placed in the W register.

ADDWF **Add W and f**

Syntax: [*label*] ADDWF f,d
Operands: $0 \le f \le 127$
$d \in [0,1]$
Operation: $(W) + (f) \rightarrow$ (destination)
Status Affected: C, DC, Z
Description: Add the contents of the W register with register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'.

ANDLW **AND Literal with W**

Syntax: [*label*] ANDLW k
Operands: $0 < k < 255$
Operation: (W) .AND. (k) → (W)
Status Affected: Z
Description: The ontents of W register are AND'ed with the eight bit literal 'k'. The result is placed in the W register.

ANDWF **AND W with f**

Syntax: [*label*] ANDWF f,d
Operands: $0 \le f \le 127$
$d \in [0,1]$
Operation: (W) .AND. (f) →(destination)
Status Affected: Z
Description: AND the W register with register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'.

BCF **Bit Clear f**

Syntax: [*label*] BCF f,b
Operands: $0 \le f \le 127$
$0 \le b \le 7$
Operation: 0 ≤ (f< b >)
Status Affected: None
Description: Bit 'b' in register 'f' is cleared

BSF **Bit Set f**

Syntax: [*label*] BSF f,b
Operands: $0 \le f \le 127$
$0 \le b \le 7$
Operation: 1 → (f<b>)
Status Affected: None
Description: Bit 'b' in register 'f' is set.

BTFSS **Bit Test f, Skip if Set**

Syntax: [*label*] BTFSS f,b
Operands: $0 \le f \le 127$
$0 \le b < 7$
Operation: skip if (f<b>) = 1
Status Affected: None
Description: If bit 'b' in register 'f' is '0', the next instruction is executed. If bit 'b' is '1', then the next instruction is discarded and a NOP is executed instead, making this a 2TCY instruction.

BTFSC **Bit Test, Skip if Clear**

Syntax: [*label*] BTFSC f,b
Operands: $0 \le f \le 127$
$0 \le b \le 7$
Operation: skip if (f<b>) = 0
Status Affected: None
Description: If bit 'b' in register 'f' is '1', the next instruction is executed. If bit 'b', in register 'f', is '0', the next instruction is discarded, and a NOP is executed instead, making

CALL **Call Subroutine**

Syntax: [*label*] CALL k
Operands: $0 \le k \le 2047$
Operation: (PC)+ 1→ TOS,k→ PC<10:0>
(PCLATH<4:3>)→PC<12:11>
Status Affected: None
Description: Call Subroutine. First, return address (PC+1) is pushed onto the stack. The eleven-bit immediate address is loaded into PC bits <10:0>. The upper bits of the PC are loaded from PCLATH.

CLRWDT **Clear Watchdog Timer**

Syntax: [*label*] CLRWDT
Operands: None
Operation: 00h → WDT
0 →WDT prescaler,
1→ $\overline{TO}$,1→ $\overline{PD}$
Status Affected: $\overline{TO}$, $\overline{PD}$
Description: CLRWDT instruction resets the Watchdog Timer. It also resets the prescaler of the WDT. Status bits $\overline{TO}$ and $\overline{PD}$ are set.

CLRW **Clear W**

Syntax: [*label*] CLRW
Operands: None
Operation: 00h → (W)
1 → Z
Status Affected: Z
Description: W register is cleared. Zero bit (Z) is set.

DECFSZ **Decrement f, Skip if 0**

Syntax: [*label*] DECFSZ f,d
Operands: 0 ≤ f ≤ 127
d ∈ [0,1]
Operation: (f) - 1 → (destination); skip if result = 0
Status Affected: None
Description: The contents of register 'f' are decremented. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed back in register 'f'. If the result is 1, the next instruction is executed. If the result is 0, then a NOP is executed instead making it a 2TCY instruction.

INCF **Increment f**

Syntax: [*label*] INCF f,d
Operands: 0 ≤ f ≤ 127
d ∈ [0,1]
Operation: (f) + 1 → (destination)
Status Affected: Z
Description: The contents of register 'f' are incremented. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed back in register 'f'

CLRF **Clear f**

Syntax: [*label*] CLRF f
Operands: 0 ≤ f ≤ 127
Operation: 00h → (f)
1 → Z
Status Affected: Z
Description: The contents of register 'f' are cleared and the Z bit is set.

DECF **Decrement f**

Syntax: [*label*] DECF f,d
Operands: 0 ≤ f ≤ 127
d ∈ [0,1]
Operation: (f) - 1 → (destination)
Status Affected: Z
Description: Decrement register 'f'. If 'd' is 0, the result is stored in W . If 'd' is 1, the result is stored back in register 'f'.

INCFSZ **Increment f, Skip if 0**

Syntax: [*label*] INCFSZ f,d
Operands: 0 ≤ f ≤ 127
d ∈ [0,1]
Operation: (f) + 1 → (destination), skip if result = 0
Status Affected: None
Description: The contents of register 'f' are incremented. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed back in register 'f'.If the result is 1, the next instruction is executed. If the result is 0, a NOP is executed instead, making it a 2TCY instruction.

IORWF **Inclusive OR W with f**

Syntax: [*label*] IORWF f,d
Operands: 0 ≤ f ≤ 127
d ∈ [0,1]
Operation: (W).OR.(f)→ (destination)
Status Affected: Z
Description: Inclusive OR the W register with register 'f'. If 'd' is 0 the result is placed in the W register. If 'd' is 1 the result is placed back in register 'f'.

COMF **Complement f**

Syntax: [*label*] COMF f,d
Operands: 0 ≤ f ≤ 127
d ∈ [0,1]
Operation: (f) → (destination)
Status Affected: Z
Description: The contents of register 'f' are complemented. If 'd' is 0, the result is stored in W. If 'd' is 1, the result is stored back in register 'f'.

IORLW **Inclusive OR Literal with W**

Syntax: [*label*] IORLW k
Operands: 0 ≤k ≤ 255
Operation: (W) .OR. k → (W)
Status Affected: Z
Description: The contents of the W register are OR'ed with the eight bit literal 'k'. The result is placed in the W register.

GOTO **Unconditional Branch**

Syntax: [*label*] GOTO k
Operands: 0 ≤ k ≤ 2047
Operation: k → PC<10:0>
PCLATH<4:3> → PC<12:11>
Status Affected: None
Description: GOTO is an unconditional branch. The eleven-bit immediate value is loaded into PC bits <10:0>. The upper bits of PC are loaded from PCLATH<4:3>. GOTO is a two-cycle instruction.

NOP **No Operation**

Syntax: [*label*] NOP
Operands: None
Operation: No operation
Status Affected: None
Description: No operation.

MOVF **Move f**

Syntax: [*label*] MOVF f,d
Operands: $0 \le f \le 127$
$d \in [0,1]$
Operation: (f) → (destination)
Status Affected: Z
Description: The contents of register f are moved to a destination dependant upon the status of d. If d = 0, destination is W register. If d = 1, the destination is file register f itself. d = 1 is useful to test a file register, since status flag Z is affected.

SLEEP

Syntax: [*label*] SLEEP
Operands: None
Operation: 00h → WDT,
0 → WDT prescaler,
1 → $\overline{TO}$,0 → $\overline{PD}$
Status Affected: $\overline{TO}$, $\overline{PD}$
Description: The power-down status bit, PD is cleared. Time-out status bit, TO is set. Watchdog Timer and its prescaler are cleared.The processor is put into SLEEP mode with the oscillator stopped.

RETFIE **Return from Interrupt**

Syntax: [*label*] RETFIE
Operands: None
Operation: TOS → PC,
1 → GIE
Status Affected: None

MOVLW **Move Literal to W**

Syntax: [*label*] MOVLW k
Operands: $0 \le k \le 255$
Operation: k → (W)
Status Affected: None
Description: The eight bit literal 'k' is loaded into W register. The don't cares will assemble as 0's.

RETLW **Return with Literal in W**

Syntax: [*label*] RETLW k
Operands: $0 \le k \le 255$
Operation: k → (W); TOS → PC
Status Affected: None
Description: The W register is loaded with the eight bit literal 'k'. The program counter is loaded from the top of the stack (the return address). This is a two-cycle instruction.

RETURN **Return from Subroutine**

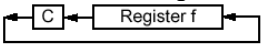
Syntax: [*label*] RETURN
Operands: None
Operation: TOS → PC
Status Affected: None
Description: Return from subroutine. The stack is POPed and the top of the stack (TOS) is loaded into the program counter. This is a two-cycle instruction.

MOVWF **Move W to f**

Syntax: [*label*] MOVWF f
Operands: $0 \le f \le 127$
Operation: (W) → (f)
Status Affected: None
Description: Move data from W register to register 'f'.

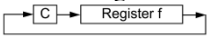
RLF **Rotate Left f through Carry**

Syntax: [*label*] RLF f,d
Operands: $0 \le f \le 127$
$d \in [0,1]$
Operation: See description below
Status Affected: C
Description: The contents of register 'f' are rotated one bit to the left through the Carry Flag. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is stored back in register 'f'.



RRF **Rotate Right f through Carry**

Syntax: [*label*] RRF f,d
Operands: $0 \le f \le 127$
$d \in [0,1]$
Operation: See description below
Status Affected: C
Description: The contents of register 'f' are rotated one bit to the right through the Carry Flag. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed back in register 'f'.



SUBLW **Subtract W from Literal**

Syntax: [*label*] SUBLW k
Operands: 0 ≤ k ≤ 255
Operation: k - (W) → W)
Status Affected: C, DC, Z
Description: The W register is subtracted (2's complement method) from the eight-bit literal 'k'. The result is placed in the W register.

SUBWF **Subtract W from f**

Syntax: [*label*] SUBWF f,d
Operands: 0 ≤ f ≤ 127
d ∈ [0,1]
Operation: (f) - (W) destination)
Status Affected: C, DC, Z
Description: Subtract (2's complement method) W register from register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'.

SWAPF **Swap Nibbles in f**

Syntax: [*label*] SWAPF f,d
Operands: 0 ≤ f ≤ 127
d ∈ [0,1]
Operation: (f<3:0>) →(destination<7:4>), (f<7:4>) → (destination<3:0>)
Status Affected: None
Description: The upper and lower nibbles of register 'f' are exchanged. If 'd' is 0, the result is placed in the W register. If 'd' is 1, the result is placed in register 'f'.

XORLW **Exclusive OR Literal with W**

Syntax: [*label*] XORLW k
Operands: 0 ≤ k ≤ 255
Operation: (W) .XOR. k → (W)
Status Affected: Z
Description: The contents of W are XOR'ed with the eight-bit literal 'k'. The result is placed in the W register.

XORWF **Exclusive OR W with f**

Syntax: [*label*] XORWF f,d
Operands: 0 ≤ f ≤ 127
d ∈ [0,1]
Operation: (W) .XOR.
(f) → destination)
Status Affected: Z
Description: Exclusive OR the contents of the W register with register 'f'. If 'd' is 0, the result is stored in the W register. If 'd' is 1, the result is stored back in register 'f'.