PIC16F877A overview

The PIC16F877A is an 8-bit microcontroller developed by Microchip. The chip is of PIC's midrange MCUs (Microcontroller unit) following the PIC architecture and a RISC instruction set. RISC stands for Reduced Instruction set architecture, were, all the operation codes have a fixed length. The PIC16F877A comes in various packages. The package used is a 40 pin DIP (Dual inline package). The MCU has four I/O ports, A through E. Port B has the only interrupt pin at RB0. Moreover, port B is the only port to have internal pull up. Port A can be used as analogue input. The MCU has an 8-level stack meaning a maximum of eight nested sub-routines can be used. The PIC16F877A has 35 instructions.

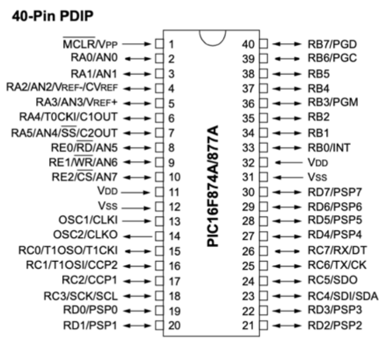


Figure 1 PIC16F877A DIP out

All operations are a single cycle except program branches like the CALL function. An operation cycle is the clock speed / 4. As each operation is Fetched, Decoded, Executed and Returned to memory. Each of these operations takes one clock cycle. The clock speed can be set in the configuration bits. Each instruction is detailed in the data sheet, with the required operation cycles.

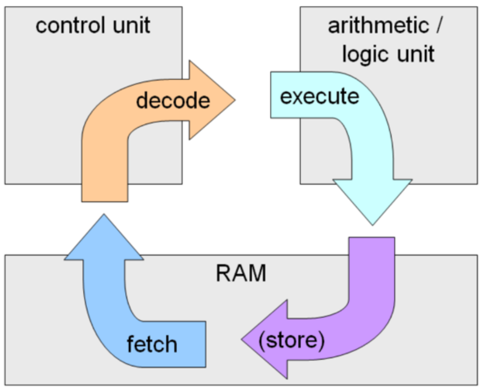
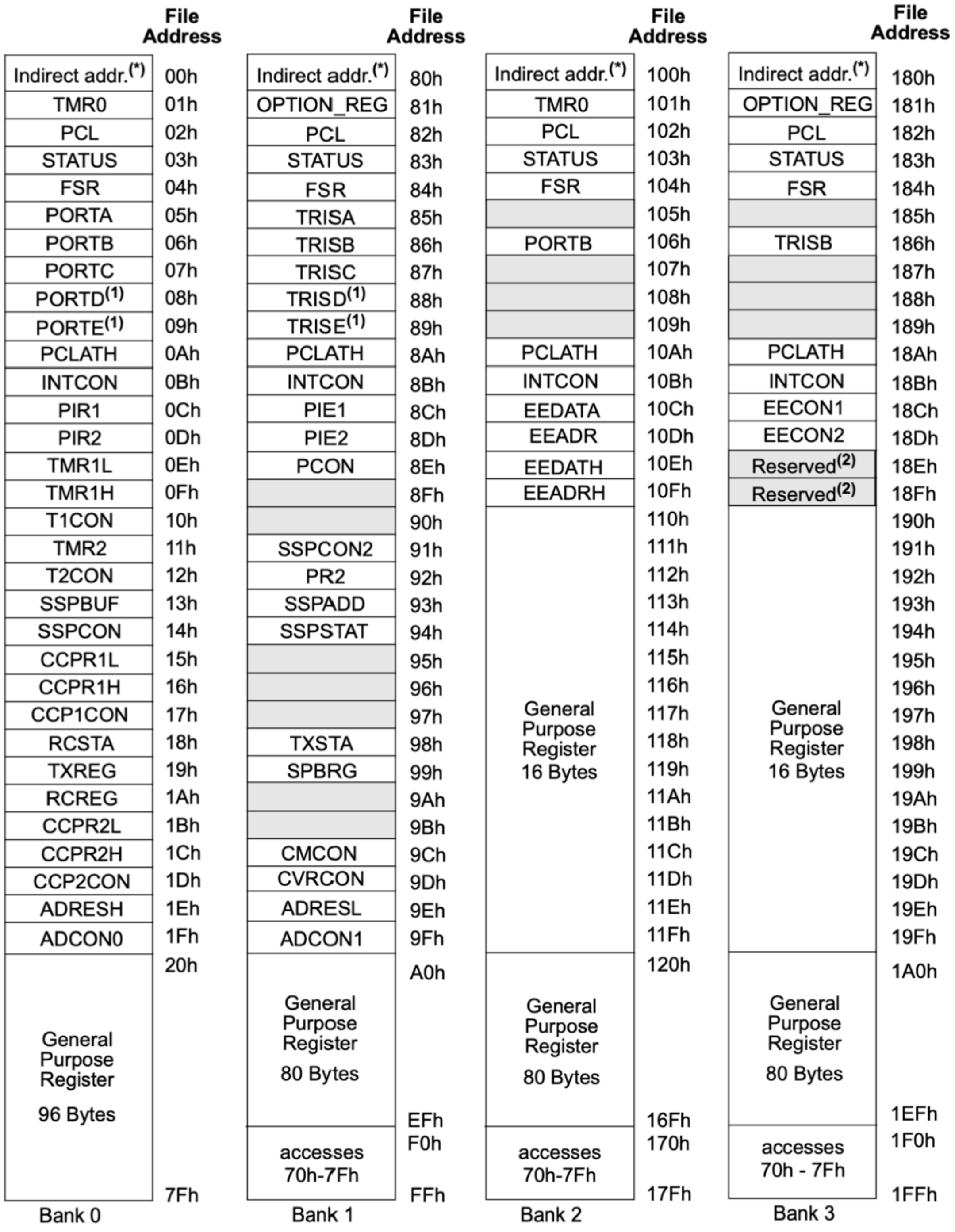


Figure 2 Fetch - Execute cycle

Memory Organization and mapping

There is only, however, one general register called W register (Working register). In spite, that variables can be declared in memory and used. The registers are organized in four banks due to the limited ability of 8-bit MCUs to access memory addresses larger than 255. To that end, multiple banks are used to access registers.

To access the required bank RP1 and RP0 bits in the status register needs to be accessed and set accordingly or simply use '*BANKSEL*' instruction followed by the required register. There are common registers like the status register and the variables found at the end of the memory starting from address 70h. In order to for instance, set PORTB as output it is required to go to bank 1 to access the TRISB which is the data direction register and set it as output then go to bank 0 to access PORTB and control the bit values, whether high or low. To declare a variable the address needs to be accessed from the memory using the required instruction.

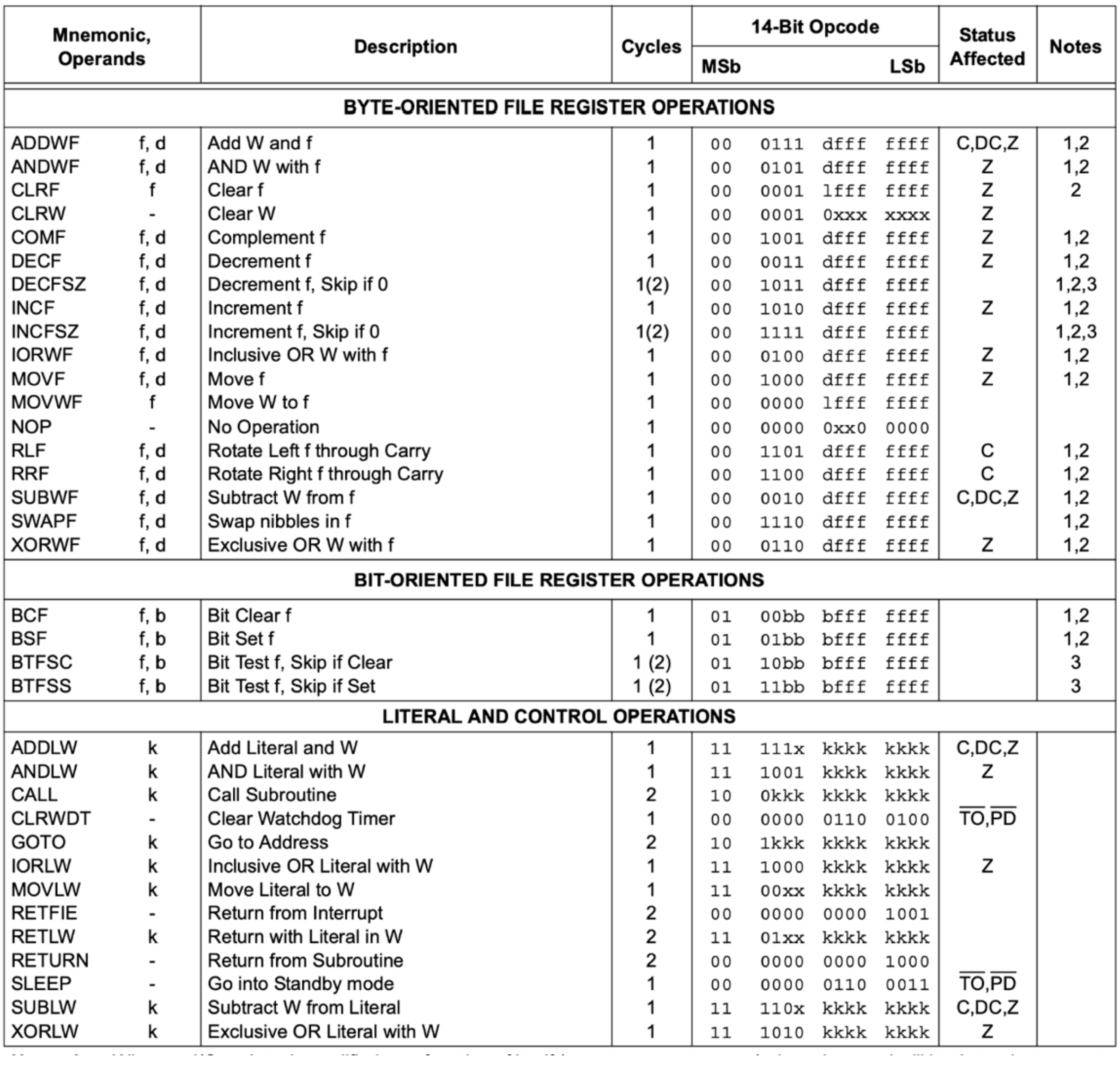


Bank 0	File Address	Bank 1	File Address	Bank 2	File Address	Bank 3	File Address
Indirect addr.(*)	00h	Indirect addr.(*)	80h	Indirect addr.(*)	100h	Indirect addr.(*)	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR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

Figure 3 Register mapping.

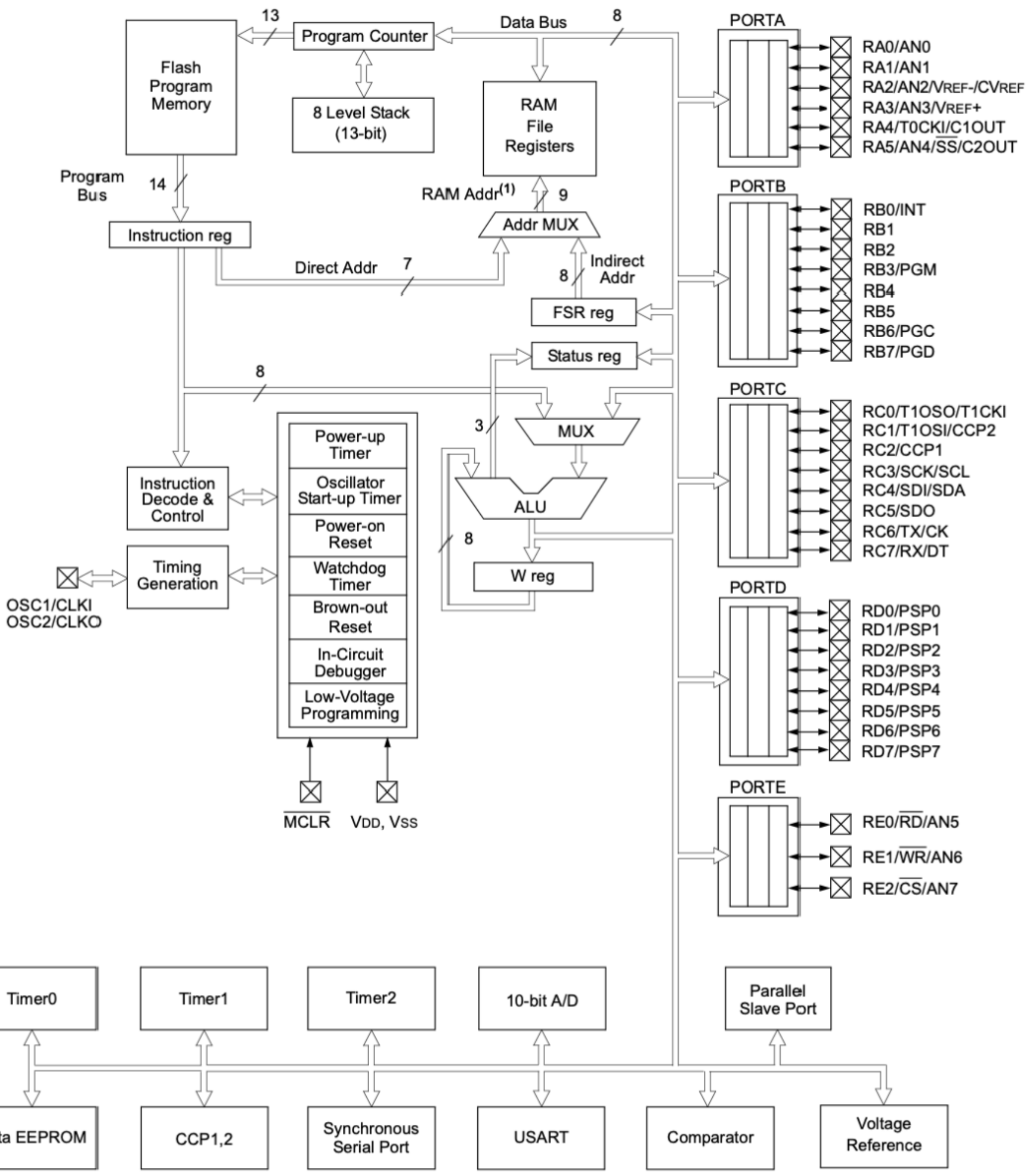
PIC16F877A instruction set architecture

The PIC16F877A has a total of 35 instructions. Divided into three sections, Byte, Bit and control operations. The BSF and BCF are used to test a specific Bit in a Port for its state, whether high or low. Depending on the outcome, it would skip the next instruction. Call function executes a sub-routine were the W register and flags are pushed to the stack coupled with the program counter after incrementing by one. Then the program jumps to the address of the sub-routine. At the end of the subroutine a return instruction indicates the end of the subroutine were the values of the W register and flags are popped from the stack with the program counter is popped to execute the next instruction. Most instructions are once operation cycles, meaning four clock cycles. However, branching instructions are two operation cycle instructions meaning eight clock cycles, for instance, the '*CALL*' and '*GOTO*'.



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

Figure 4 Instruction set



Device	Program Flash	Data Memory	Data EEPROM
PIC16F874A	4K words	192 Bytes	128 Bytes
PIC16F877A	8K words	368 Bytes	256 Bytes

Figure 5 PIC16F877A block diagram

PIC16F877A interrupts

To use any internal peripheral like the interrupt the necessary registers are needed to be set accordingly. There are two types of interrupts, peripheral interrupts like the timer interrupt or overflow interrupt. The second type is external interrupts, there is one external interrupt attached to PORTB pin 0. The interrupt enables the system upon a change of pin state, either falling or rising edge, to execute the Interrupt Service Routine. Upon the state change, the microprocessor interrupts the current process, pushes the register, flags, and program counter to the stack. Then executes the ISR. After completion, the register, flags, and program counter are popped from the stack and the microprocessor continues the current operation before the interrupt flag was raised. To enable the interrupt a few registers, need to be manipulated. Registers are set in the initialization phase, then in the execution and after the execution. Steps to enable and use hardware interrupts:

1- From the OPTION register choose falling or rising edge from INTEDG bit, rising edge = 1, falling edge = 0.

2- Clear flag bit from INTCON register, INTF bit, which is the first bit.

3- Enable global interrupt bit from the INTCON register. GIE is the seventh bit. GIE = 1.

4- Enable interrupt bit from INTCON register. INTE is the fourth bit. INTE = 1.

5- Create an ISR sub-routine and interrupt vector table starting at 004h.

6- Within the sub-routine, clear the INTF, interrupt flag.

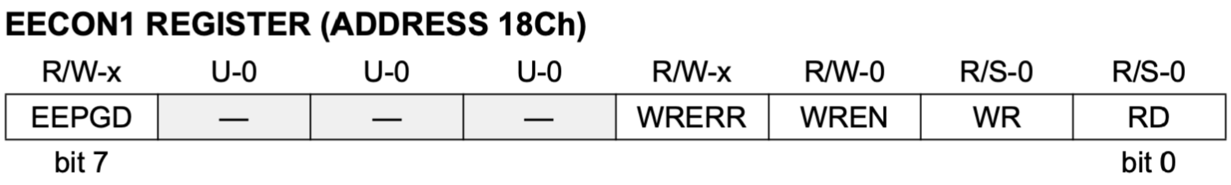
7- At the end of the ISR '*RETFIE*' is used to enable back the global interrupt and return to main program.

EEPROM

Electrically Erasable Programmable Read Only Memory, is used to store data as it is non-volatile storage. It can store data without power for up to 40 years, It has 1,000,000 read and write cycles. It is used to store the user password and safe flag states. The EEPROM has two operations, read and write. It consists of 256 bytes mapped from address 0X00 to 0XFF. To use the EEPROM in either mode proper flags need to be set. The EEPROM has a total of 6 registers.

- EECON1
- EECON2
- EEDATA
- EEDATH
- EEADR
- EEADRH

The EECON1 is used to control read and write mode, while EECON2 is only readable by the EEPROM not the user and would return an empty data if read. The EEDATA and EEDATH are the data registers together they store 14-bit data. The EEADR and EEADRH store the address to be written or read and together they store 13-bit addresses.


EECON1 REGISTER (ADDRESS 18Ch)

R/W-x	U-0	U-0	U-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	—	—	—	WRERR	WREN	WR	RD
bit 7							bit 0

Figure 6 EECON1 register

Steps to read from EEPROM:

1- Address is written to EEADR.
2- EEPGD bit in the EECON1 register is cleared to point to data memory.
3- RD bit in the EECON1 register is set to start the reading operation.
4- Data is read from the EEDATA register and stored in W register.

Steps to write to EEPROM:

1- Check the WR bit in the EECON1 register to check for any other reading operations.
2- Address is written to EEADR.
3- Data is written to the EEDATA register.
4- EEPGD bit in the EECON1 register is clear to point to data memory.
5- WREN bit in the EECON1 register is set to enable operations.
6- Interrupts are disabled.

7- Execute special sequence.
    a. 0X55 written to EECON2.
    b. 0XAA written to EECON2
    c. WR bit is set in the EECON1 register.
8- Interrupts are enabled.
9- WREN bit in the EECON1 register is cleared to disable operations.
10- At the end the writing operation, the WR bit is cleared and EEIF interrupt flag is set.

Flowchart

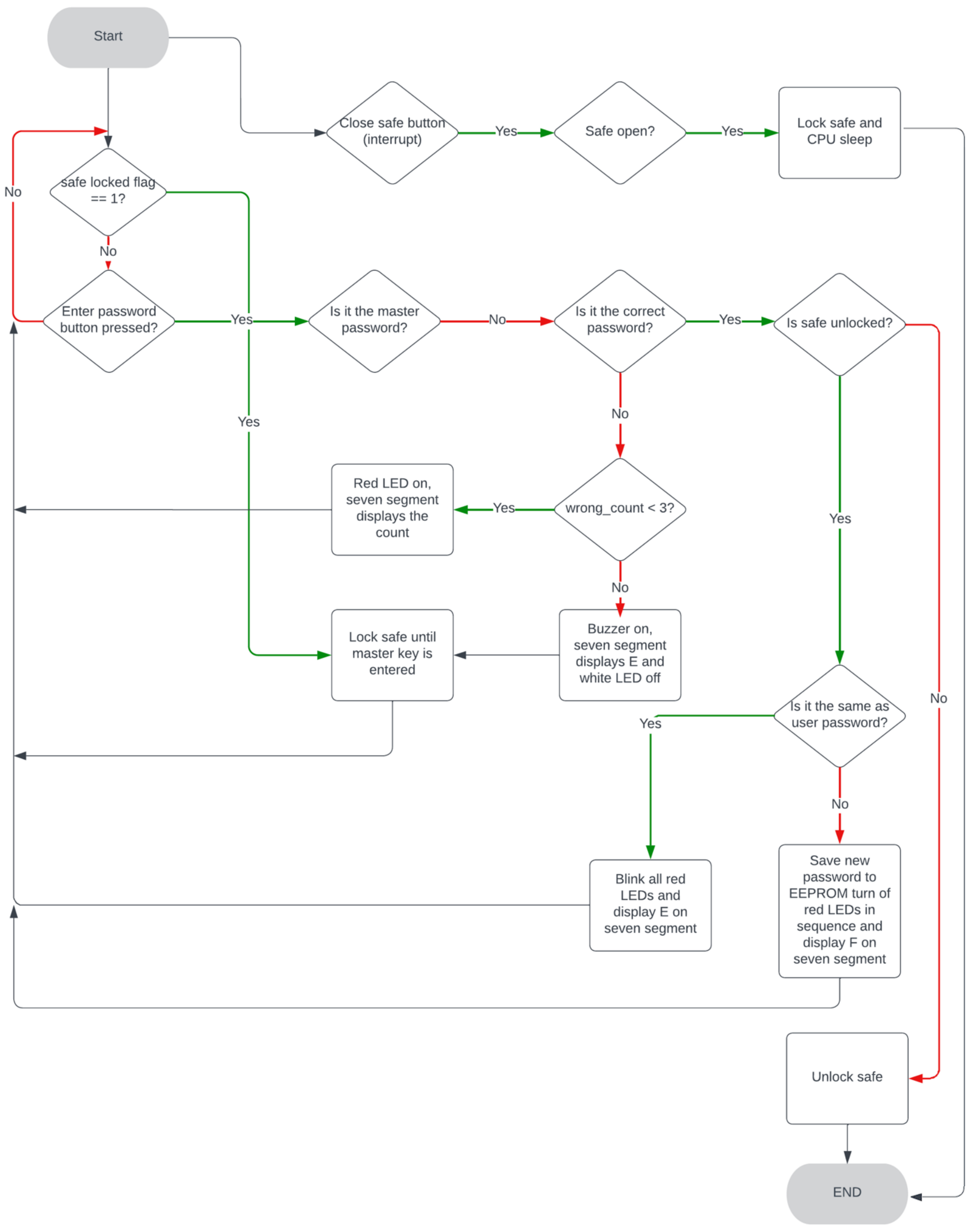

Start
Close safe button (interrupt)
Yes
Safe open?
Yes
Lock safe and CPU sleep
No
safe locked flag == 1?
No
Enter password button pressed?
Yes
Is it the master password?
No
Is it the correct password?
Yes
Is safe unlocked?
Yes
No
Red LED on, seven segment displays the count
Yes
wrong_count < 3?
Yes
No
Lock safe until master key is entered
Buzzer on, seven segment displays E and white LED off
No
Yes
Is it the same as user password?
No
Blink all red LEDs and display E on seven segment
No
Save new password to EEPROM turn of red LEDs in sequence and display F on seven segment
Unlock safe
END

Figure 7 program flowchart