### 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.

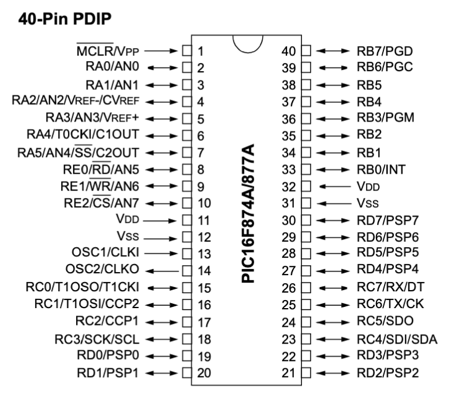


Figure 1 PIC16F877A DIP out

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.

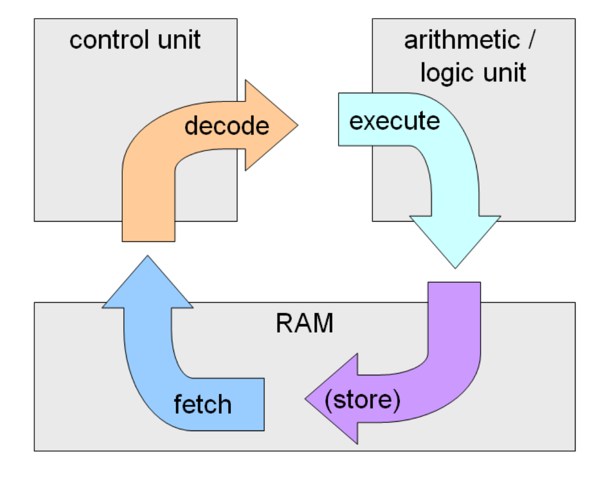


Figure 2 Fetch - Execute cycle

### 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.

A picture containing text, menu, document, font

Description automatically generated

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*’.

A picture containing text, screenshot, number, menu

Description automatically generated

Figure 4 Instruction set

A picture containing text, diagram, plan, schematic

Description automatically generated

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.

A picture containing text, font, screenshot, line

Description automatically generated

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.
   1. 0X55 written to EECON2.
   2. 0XAA written to EECON2
   3. 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

A picture containing text, diagram, plan, line

Description automatically generated



Figure 7 program flowchart