PIC Microcontroller
PIC Microcontroller (MCU)

**Widely used device from Microchip Technology**

- Sold > 10 billion PIC controllers
- Several device families
  - Many devices per family
  - Common development environment
- Widely available
  - Large user base
  - Extensive application notes
- Low cost
- Free / low cost development tools

**Basic architectural features**

- Pipelined RISC microprocessor core
- Accumulator execution model
- Data width — 8 / 16 / 32 bits
- Instruction width — 12 / 14 / 16 / 32 bits
## PIC Families

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Family</th>
<th>Data Width</th>
<th>Instruction Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit MCU</td>
<td>PIC10 / PIC12 / PIC16 Baseline</td>
<td>8 bits</td>
<td>12-bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mid-Range</td>
<td>14-bits</td>
</tr>
<tr>
<td></td>
<td>PIC18</td>
<td></td>
<td>16-bits</td>
</tr>
<tr>
<td>16-bit MCU</td>
<td>PIC24</td>
<td>16 bits</td>
<td>16 bits</td>
</tr>
<tr>
<td></td>
<td>dsPIC30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32-bit MCU</td>
<td></td>
<td>32 bits</td>
<td>32 bits</td>
</tr>
</tbody>
</table>

### Data width

- 8 / 16/ 32 bits
- Wider integer ⇒ higher precision arithmetic

### Instruction width

- 12 / 14 / 16 / 32 bits
- Wider instruction ⇒ more complex instructions + higher precision arithmetic
Typical Applications

**Baseline**

Replace discrete logic functions

- Gates, simple state machines, encoders/decoders, etc.

Disposable electronics

- Drug / pregnancy testers, dialysis monitor, etc

**Mid-Range**

Digital sensors, displays, controllers, telecom equipment

- Glucose / blood pressure set

**PIC18**

Integration with peripherals + networks

- USB, Ethernet, MCU-to-MCU, etc

Higher level analog peripherals, industrial control, major appliances

**PIC24 / dsPIC30**

16-bit ALU with integrated DSP

- Portable EGK

**PIC32**

General purpose RISC microprocessor + controller

- MRI
Learning PIC Architecture

Some general observations

Variety

Hundreds of PIC devices in 3 families and several sub-families

Updates

Microchip Technology upgrades devices frequently
Familiar devices replaced with new model

Instruction Set Architecture

8 and 16 bit devices share approximately uniform instruction set
PIC32 implements MIPS ISA

Caveats

Course takes general pedagogical approach to PIC as typical MCU
Focus on 8-bit devices — Mid-Range + PIC18
Many books + websites on PIC with general-sounding titles
Each device is unique
Few statements are precisely true about each device
## 8-Bit PIC MCUs

| **Data memory**                          | Organized as 8-bit registers  
Some devices also store data on EEPROM  
16 B to 4 KB |
|----------------------------------------|--------------------------------|
| **Program memory**                     | Addressable unit = instruction word = 12 / 14 / 16 bits  
Smallest: 2 Kword (3 KB of 12-bit instructions)  
Largest: 64 Kword (128 KB of 16-bit instructions) |
| **Architecture**                       | Pipelined RISC  
33 to 77 instructions |
| **Stack**                              | Stores 0 (no stack) to 31 instruction addresses  
Used for function calls |
| **I/O devices**                        | 8-bit parallel ports  
Synchronous / asynchronous serial ports  
Timers + watchdog timer  
A/D + D/A converters  
Pulse width modulators |
8-Bit PIC Operation Model

**ALU sources**
- Special **WORKING** register W
- Data register or immediate

**ALU destination**
- Data register or W

**Transfer operations**
- Data register ↔ W

**Status register**
- Flags produced by ALU operations
Pipeline Operation
Instruction cycles (CY)

Instruction Cycles

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( I_1 )</td>
<td>fetch</td>
<td>execute</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( I_2 )</td>
<td>fetch</td>
<td></td>
<td>execute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( I_3 )</td>
<td></td>
<td>fetch</td>
<td></td>
<td>execute</td>
<td></td>
</tr>
<tr>
<td>( I_4 )</td>
<td></td>
<td></td>
<td>fetch</td>
<td></td>
<td>execute</td>
</tr>
</tbody>
</table>

Branch instructions require 2 instruction cycles
Pipeline Operation
Clock cycles (OSC)

Instruction Cycle

4 cycles of external clock (oscillator)

\[ CY = Q_1 \rightarrow Q_2 \rightarrow Q_3 \rightarrow Q_4 \]

Instruction fetch

<table>
<thead>
<tr>
<th>Q1</th>
<th>Update Program Pointer</th>
<th>PC ← PC + 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q2-Q3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td>Fetch</td>
<td>IR ← [PC]</td>
</tr>
</tbody>
</table>

Execution

<table>
<thead>
<tr>
<th>Q1-Q4</th>
<th>Decode and Execute</th>
<th>Operation dependent</th>
</tr>
</thead>
</table>

Instruction Cycles

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1Q2Q3Q4</td>
<td>Q1Q2Q3Q4</td>
<td>Q1Q2Q3Q4</td>
<td>Q1Q2Q3Q4</td>
<td>Q1Q2Q3Q4</td>
</tr>
<tr>
<td>I_1</td>
<td>fetch</td>
<td>execute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I_2</td>
<td>fetch</td>
<td></td>
<td>execute</td>
<td></td>
</tr>
<tr>
<td>I_3</td>
<td></td>
<td>fetch</td>
<td>execute</td>
<td></td>
</tr>
<tr>
<td>I_4</td>
<td></td>
<td>fetch</td>
<td></td>
<td>execute</td>
</tr>
</tbody>
</table>
Clock Types

**RC oscillator**

Least expensive

Can be used for non-critical frequency accuracy and stability

Some devices have internal RC oscillator at 4 MHz

**Crystal oscillator**

Most stable

**External clock**

Provided by external digital system

**Specific modes**

LP mode — frequencies between 32 kHz and 200 kHz

XT mode — frequencies between 100 kHz and 4 MHz

HS mode — frequencies between 8 MHz and 20 MHz
Sleep Mode

Low-power mode

- Main oscillator stopped
- Most MCU functions stopped
- Watchdog time continues
- Power consumed < 1 mA for some models

Instruction SLEEP

- MCU → sleep mode
- Data register values stable

Pipeline locked

- Sleep instruction executes ⇒ next instruction already fetched

On wake up

- Next instruction executes
- Recommendation — instruction after sleep = NOP
- Watchdog timer counter reset
Wake Up Events

**Reset**

Fetch instruction from address 0

**Watchdog timer overflow**

Normal execution of instruction following sleep

**Interrupt**

Interrupt not enabled $\Rightarrow$ ignore interrupt

Enabled

Normal execution of instruction following sleep

PC jumps to address 4 in program memory

Finds interrupt routine
Watchdog Timer

**WDT oscillator (clock)**
- Independent from main clock
- Continues in low power mode
- May be disabled

**WDT timeout**
- Timeout = 18 ms
- Non-sleep mode
  - MCU resets
- Sleep mode
  - MCU wakes up → executes instruction following sleep

**Reset WDT**
- CLRWDT resets timeout = 18 ms

**Prescaler**
- Divide time-base by $2^k$, $k = 0, \ldots, 7$
- Extend timeout up to 2300 ms
# Some Typical 8-bit PIC Device Families

<table>
<thead>
<tr>
<th></th>
<th>PIC10F2xx</th>
<th>PIC12F5xx</th>
<th>PIC16F5xx</th>
<th>PIC10F3xx</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>PIC12F6xx</td>
<td>PIC16F6xx</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PIC18F5xx</td>
</tr>
<tr>
<td>Instruction word</td>
<td>12 bits</td>
<td>12 bits</td>
<td>12 bits</td>
<td>14 bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>16 bits</td>
</tr>
<tr>
<td>Instructions</td>
<td>33</td>
<td>33</td>
<td>33</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>83</td>
</tr>
<tr>
<td>Program memory</td>
<td>256 – 512 words</td>
<td>512 – 1024 words</td>
<td>1024 – 2048 words</td>
<td>256 – 8192 words</td>
</tr>
<tr>
<td></td>
<td>Flash</td>
<td>Flash</td>
<td>Flash</td>
<td>Flash</td>
</tr>
<tr>
<td>ROM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data memory</td>
<td>16 – 24</td>
<td>25 – 41</td>
<td>25 – 134</td>
<td>56 – 368</td>
</tr>
<tr>
<td></td>
<td>256 – 4K</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(bytes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interrupts</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>int / ext</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>int / ext</td>
</tr>
<tr>
<td>Pins</td>
<td>6</td>
<td>8</td>
<td>14 – 40</td>
<td>6 – 64</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>16 – 70</td>
<td></td>
</tr>
<tr>
<td>I/O pins</td>
<td>4</td>
<td>6</td>
<td>12 – 32</td>
<td>4 – 54</td>
</tr>
<tr>
<td>Stack</td>
<td>2 levels</td>
<td>2 levels</td>
<td>2 levels</td>
<td>8 levels</td>
</tr>
<tr>
<td>Timers</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2 – 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2 – 5</td>
</tr>
<tr>
<td>Bulk price</td>
<td>$0.35</td>
<td>$0.50</td>
<td>$0.50 – 0.85</td>
<td>$0.35 – 2.50</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$1.20 - $8.50</td>
</tr>
</tbody>
</table>
Typical Baseline MCU — PIC16X5xx Family

12-bit instruction

8-bit data

Port A → RA3-RA0
Port B → RB7-RB0
Port C → RC7-RC0

General I/O ports

W+ ALU

timer

Instruction decode and control

Program memory

Data memory

PC

Data Bus

2-level stack

IR

Direct address

Indirect address

Direct data

FSR

MUX

MUX

ALU

Reset

WDT

Configuration bits

OSC1

OSC2

MCLR#

VDD

VSS

W

STATUS

Timer0

OSC

Oscillator
Typical Mid-Range MCU — PIC16F873

14-bit instruction

8-bit data

General I/O ports

External Interrupt

Timers

A/D

UART

Compare-Capture-Pulsewidth (CCP)

Synchronous Serial Port (SSP)
Typical PIC18 MCU

16-bit instruction

Timers
A/D
UART
USB
Compare-Capture-Pulsewidth (CCP)
Controller Area Network (CAN)

8-bit data

Port A
Port B
Port C
Port D
Port E

General I/O ports
External Interrupts
Mid-Range PIC MCUs
Data Memory / Registers

**Register**

Addressable location in data memory
8-bit word (byte)

**Data address space**

9 bit address \(\Rightarrow\) memory \(\leq 2^9 = 512\) bytes = 0.5 KB

**Memory partitioned into banks**

Bank = \(2^7 = 128 = 80\)h registers (1/8 KB)
7 bit file address
Displacement in bank = 00h ... 7Fh

**Banks in address space**

\(\leq 2^{9-7} = 4\) banks
2 to 4 banks implemented in device

Unimplemented banks
Read as 0
Write as NOP

![Diagram of data address structure]

- **data address**: 9 bits
- **bank**: 2 bits
- **file address**: 7 bits

- \(00\) to \(11\)
- \(00\) to \(7F\)
# Special / General Registers

## GPR

General Purpose Registers
User program data

## SFR

Special Function Registers
Reserved for
- Control / configuration
- Peripheral access
- Indirect addressing
- Program counter

## Core SFRs

Appear in every bank at same file address

<table>
<thead>
<tr>
<th>Typical SFRs</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS</td>
</tr>
<tr>
<td>STATUS word + flags</td>
</tr>
<tr>
<td>OPTION</td>
</tr>
<tr>
<td>Timer options</td>
</tr>
<tr>
<td>PCLATH</td>
</tr>
<tr>
<td>Components of program counter (PC)</td>
</tr>
<tr>
<td>PCL</td>
</tr>
<tr>
<td>FSR</td>
</tr>
<tr>
<td>File Select for indirect data addressing</td>
</tr>
<tr>
<td>INTCON, PIR1, PIE1, PIR2, PIE2</td>
</tr>
<tr>
<td>Components of interrupt handling</td>
</tr>
<tr>
<td>PORTA, TRISA, ...</td>
</tr>
<tr>
<td>Access to parallel ports</td>
</tr>
<tr>
<td>TMR0, OPTION, INTCON, ...</td>
</tr>
<tr>
<td>Timer0</td>
</tr>
<tr>
<td>TXREG, TXSTA, RCREG, RCSTA, ...</td>
</tr>
<tr>
<td>Access to serial port</td>
</tr>
<tr>
<td>ADRESH, ADRESL, ADCON0, ...</td>
</tr>
<tr>
<td>Access to A/D converter</td>
</tr>
<tr>
<td>EEADRH, EEDATA, ...</td>
</tr>
<tr>
<td>Access to EEPROM and Flash memory</td>
</tr>
</tbody>
</table>
Notes

(2,3) Not all locations implemented on all devices

(4) Common RAM — accessible in all banks (on applicable devices)

(5) Not implemented on smaller devices
# Status Register

Core SFR accessible at file address 03h in every bank

<table>
<thead>
<tr>
<th>Name</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRP</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>RO</td>
<td>RO</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Writable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reset value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IRP</th>
<th>Bank Select</th>
<th>Indirect Register Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>RP1, RP0</td>
<td>State of WDT</td>
<td>Direct Register Pointer</td>
</tr>
<tr>
<td>TO#</td>
<td>State of WDT</td>
<td>TO# ← 0 on WDT overflow</td>
</tr>
<tr>
<td>PD#</td>
<td>State of WDT</td>
<td>TO# ← 1 on power-on reset, CLRWDT, SLEEP</td>
</tr>
<tr>
<td>Z</td>
<td>State of WDT</td>
<td>PD# ← 0 on SLEEP</td>
</tr>
<tr>
<td>DC</td>
<td>State of WDT</td>
<td>PD# ← 1 on CLRWDT and power-on reset</td>
</tr>
<tr>
<td>C</td>
<td>State of WDT</td>
<td>Z ← 1 on ALU zero</td>
</tr>
<tr>
<td>C</td>
<td>State of WDT</td>
<td>Z ← 0 on non-zero</td>
</tr>
</tbody>
</table>

C ← 1 on carry (Addition)
C ← 0 on borrow (Subtraction)
# Addressing Data Memory

## Notation

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REG&lt;(b)&gt;</td>
<td>Bit (b) in register REG</td>
</tr>
<tr>
<td>REG&lt;(a:b)&gt;</td>
<td>Bits (a) to (b) in register REG</td>
</tr>
<tr>
<td>A (\cdot) B</td>
<td>Concatenation of (A) and (B) ((A) bits followed by (B) bits)</td>
</tr>
</tbody>
</table>

## Direct addressing

Program specifies data address

Bank selection

\(\text{STATUS} \) bits \(\text{RP1} \) and \(\text{RP0}\)

On reset

\(\text{RP1} = \text{RP0} = 0 \Rightarrow \) bank 0 selected

Bank switching

Write to \(\text{STATUS}<6:5>\)

File Address

Literal field in instruction

<table>
<thead>
<tr>
<th>RP1</th>
<th>RP0</th>
<th>7 bits from instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>bank</td>
<td>file address</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>
Addressing Data Memory

Indirect addressing

Program writes to Special Function Registers (SFRs)
Address formed from SFRs
Instructions can increment/decrement SFR values
Similar to pointer arithmetic

File Select Register (FSR)

Core SFR accessible at file address 08h in all banks

File Address
FSR<6:0>

Bank
IRP.FSR<7>

STATUS bit IRP (Indirect Register Pointer)

On small devices

1 or 2 banks = 128 or 256 bytes of data memory
8 bit FSR address covers 2 banks
IRP not implemented (read 0 / write = NOP)
INDF Register

**INDF**

Core SFR accessible at file address 00h in all banks
Virtual pointer — not physical register
Tracks contents of FSR
Simplifies pointer arithmetic

**Example**

In register file,

\[
\begin{align*}
[05] &= 10h \\
[06] &= 0Ah
\end{align*}
\]

Load FSR ← 05 ; FSR points to file address 05
[INDF] = 10h ; INDF points to file address 05
FSR++ ; increment FSR ⇒ FSR = 06
[INDF] = 0Ah ; INDF points to file address 06
# Instruction Memory Space

## All 8-bit MCUs

### Instruction address
- **n** bit location address
- Location = instruction
- $\leq 2^n$ instructions

### Instruction width
- 12 / 14 / 16 bits

### Page
- Partition of instruction memory space
- $2^k$ instructions / page
- **k** bit offset

---

<table>
<thead>
<tr>
<th>Page</th>
<th>Instruction</th>
<th>Memory Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>page offset</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>instruction</td>
<td>1...1 1...11</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>instruction</td>
<td>1...1 0...00</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>instruction</td>
<td>0...1 1...11</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>instruction</td>
<td>0...1 0...10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>instruction</td>
<td>0...1 0...01</td>
</tr>
<tr>
<td></td>
<td>instruction</td>
<td>0...1 0...00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>instruction</td>
<td>0...0 1...11</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>instruction</td>
<td>0...0 0...11</td>
</tr>
<tr>
<td></td>
<td>instruction</td>
<td>0...0 0...10</td>
</tr>
<tr>
<td></td>
<td>instruction</td>
<td>0...0 0...01</td>
</tr>
<tr>
<td></td>
<td>instruction</td>
<td>0...0 0...00</td>
</tr>
</tbody>
</table>

---

The diagram illustrates the partitioning of instruction memory space into pages and the offset within each page.
### Instruction Memory Space

#### Mid-Range instruction memory

- 14-bit instruction word
  - \( n = 13 \)
  - \( 2^{13} = 8192 \) instruction words
- \( k = 11 \)
  - Page = \( 2^{11} = 2048 = 800h \) words

#### Program counter (PC)

- \( \text{PC}<12:11> = \text{page number} \Rightarrow 4 \) pages
- \( \text{PC}<10:0> = \text{offset} \)

#### Reserved addresses

- Address 0h
  - Reset vector — pointer to reset routine
- Address 4h
  - Interrupt vector — pointer to interrupt service routine
PC Access

PC register details

PC low (PCL) = PC<7:0>
Accessible by instruction reads/writes

PC high (PCH) = PC<12:8>
Not directly accessible to instructions

PC latch high (PCLATH)
Core SFR accessible at file address 0Ah in all banks

PCH = PC<12:8> = PCLATH<4:0>
PCLATH<7:5> not implemented

---

<table>
<thead>
<tr>
<th>PCLATH</th>
<th>PCH</th>
<th>PCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>
PC Updates

**Reset**
- \( \text{PC} \leftarrow 0 \)

**Non-branch instruction**
- \( \text{PC} \leftarrow \text{PC} + 1 \)

**Branch types**

**Direct branch**
- GOTO instruction
- \( \text{PCH}_{12:11} \leftarrow \text{PCLATH}_{4:3} \)
- Offset = \( \text{PC}_{10:0} \leftarrow \text{literal}_{10:0} \) from instruction

**Indirect branch**
- Computed GOTO
- Write to PCL as register
- Copies \( \text{PCL} \leftarrow \text{ALU result} \)
- Forces \( \text{PCH} \leftarrow \text{PCLATH}_{4:0} \)
Call / Return

Stack

8 level FILO buffer
Holds 13 bit instruction addresses on CALL/RETURN

Function entry

CALL instruction
STACK ← PC<12:0>
PCL ← literal<10:0> from instruction
PCH<12:11> ← PCLATH<4:3>

Function exit

RETURN instruction
PC<12:0> ← STACK
PCLATH not updated
May be different from PCH after RETURN
### Instruction Format

#### Byte oriented

- \( d = 0 \Rightarrow \text{destination} = W \)
- \( d = 1 \Rightarrow \text{destination} = f \)
- \( f = 7 \text{ bit file address} \)

#### Bit oriented

- \( b = \text{bit position in register} \)
- \( f = 7 \text{ bit file address} \)

#### General literal

- \( k = 8 \text{ bit literal (immediate)} \)

#### CALL / GOTO

- \( k = 11 \text{ bit literal (immediate)} \)
# Instruction Set

## Data transfer

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operation</th>
<th>Comment</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVF f, 0</td>
<td>W ← f</td>
<td>d = W</td>
<td>Z</td>
</tr>
<tr>
<td>MOVF f, W</td>
<td>W ← f</td>
<td>d = W</td>
<td>Z</td>
</tr>
<tr>
<td>MOVF f, 1</td>
<td>f ← f</td>
<td>d = f</td>
<td>Z</td>
</tr>
<tr>
<td>MOVF f, f</td>
<td>f ← f</td>
<td>d = f</td>
<td>Z</td>
</tr>
<tr>
<td>MOVF f</td>
<td>f ← W</td>
<td>Move W to f</td>
<td>—</td>
</tr>
<tr>
<td>MOVWF f</td>
<td>f ← W</td>
<td>Move W to f</td>
<td>—</td>
</tr>
<tr>
<td>MOVVLW k</td>
<td>W ← k</td>
<td>Move literal to W</td>
<td>—</td>
</tr>
<tr>
<td>CLRF f</td>
<td>f ← 0</td>
<td>Clear f</td>
<td>Z</td>
</tr>
<tr>
<td>CLRW</td>
<td>W ← 0</td>
<td>Clear W</td>
<td>Z</td>
</tr>
</tbody>
</table>

### Operands

<table>
<thead>
<tr>
<th>f</th>
<th>name / address of register</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>destination</td>
</tr>
<tr>
<td>k</td>
<td>literal</td>
</tr>
</tbody>
</table>

destination = \[
\begin{cases} 
  W, & d = 0 \\
  f, & d = 1 
\end{cases} \]
# Instruction Set

## Arithmetic and Logic — 1

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operation</th>
<th>Comment</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDWF f, d</td>
<td>d ← f + W</td>
<td>Add w to f</td>
<td>C, DC, Z</td>
</tr>
<tr>
<td>ADDLW k</td>
<td>W ← k + W</td>
<td>Add k to W</td>
<td>C, DC, Z</td>
</tr>
<tr>
<td>SUBWF f, d</td>
<td>d ← f - W</td>
<td>Sub W from f</td>
<td>C, DC, Z</td>
</tr>
<tr>
<td>SUBLW k</td>
<td>W ← k - W</td>
<td>Sub W from k</td>
<td>C, DC, Z</td>
</tr>
<tr>
<td>INCF f, d</td>
<td>d ← f + 1</td>
<td>Inc f to d</td>
<td>Z</td>
</tr>
<tr>
<td>DECF f, d</td>
<td>d ← f - 1</td>
<td>Dec f to d</td>
<td>Z</td>
</tr>
<tr>
<td>ANDWF f, d</td>
<td>d ← f and W</td>
<td>And w with f</td>
<td>Z</td>
</tr>
<tr>
<td>ANDLW k</td>
<td>W ← k and W</td>
<td>And k with W</td>
<td>Z</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flags</th>
<th>destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>C, DC, Z</td>
<td>W, d = 0</td>
</tr>
<tr>
<td></td>
<td>f, d = 1</td>
</tr>
</tbody>
</table>

## Operands

<table>
<thead>
<tr>
<th>Operands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>name / address of register</td>
</tr>
<tr>
<td>d</td>
<td>destination</td>
</tr>
<tr>
<td>k</td>
<td>literal</td>
</tr>
</tbody>
</table>
### Instruction Set

#### Arithmetic and Logic — 2

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operation</th>
<th>Comment</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>IORWF f, d</td>
<td>( d \leftarrow f \text{ or } W )</td>
<td>OR ( w ) with ( f )</td>
<td>( Z )</td>
</tr>
<tr>
<td>IORLW k</td>
<td>( W \leftarrow k \text{ or } W )</td>
<td>OR ( k ) with ( W )</td>
<td>( Z )</td>
</tr>
<tr>
<td>XORWF f, d</td>
<td>( d \leftarrow f \text{ xor } W )</td>
<td>XOR ( W ) with ( f )</td>
<td>( Z )</td>
</tr>
<tr>
<td>XORLW k</td>
<td>( W \leftarrow k \text{ xor } W )</td>
<td>XOR ( W ) with ( k )</td>
<td>( Z )</td>
</tr>
<tr>
<td>RLF f, d</td>
<td>( d \leftarrow \text{ left rotate } f,C )</td>
<td></td>
<td>( C )</td>
</tr>
<tr>
<td>RRF f, d</td>
<td>( d \leftarrow \text{ right rotate } f,C )</td>
<td></td>
<td>( C )</td>
</tr>
<tr>
<td>COMF f, d</td>
<td>( d \leftarrow #f \text{ (not } f) )</td>
<td>compliment ( f ) to ( d )</td>
<td>( C )</td>
</tr>
<tr>
<td>SWAPF f, d</td>
<td>( d \leftarrow f_L \leftrightarrow f_H )</td>
<td>nibble swap (nibble = half byte)</td>
<td>—</td>
</tr>
</tbody>
</table>

### Operands

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>name / address of register</td>
<td></td>
</tr>
<tr>
<td>( d )</td>
<td>destination</td>
<td></td>
</tr>
<tr>
<td>( k )</td>
<td>literal</td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{destination} = \begin{cases} 
W, d = 0 \\
 f, d = 1 
\end{cases} \]
## Instruction Set

### Control

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOTO a</td>
<td>branch to address</td>
</tr>
<tr>
<td>BTFSC f, b</td>
<td>skip one instruction if f&lt;b&gt; = 0</td>
</tr>
<tr>
<td>BTFSS f, b</td>
<td>skip one instruction if f&lt;b&gt; = 1</td>
</tr>
<tr>
<td>INCFSZ f, d</td>
<td>d ← f + 1, skip one if result = 0</td>
</tr>
<tr>
<td>DECFSZ f, d</td>
<td>d ← f - 1, skip one if result = 0</td>
</tr>
<tr>
<td>CALL a</td>
<td>call subroutine in address a</td>
</tr>
<tr>
<td>RETURN</td>
<td>subroutine return</td>
</tr>
<tr>
<td>RETFIE</td>
<td>interrupt return</td>
</tr>
<tr>
<td>RETLW k</td>
<td>return from subroutine with k in W</td>
</tr>
</tbody>
</table>

### Operands

<table>
<thead>
<tr>
<th>f</th>
<th>name / address of register</th>
<th>a</th>
<th>11 bit address</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>destination</td>
<td>b</td>
<td>bit location</td>
</tr>
<tr>
<td>k</td>
<td>literal</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Instruction Set
### Other

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Operation</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCF f, b</td>
<td>( f_{&lt;b&gt;} \leftarrow 0 )</td>
<td>−</td>
</tr>
<tr>
<td>BSF f, b</td>
<td>( f_{&lt;b&gt;} \leftarrow 1 )</td>
<td>−</td>
</tr>
<tr>
<td>NOP</td>
<td>no operation</td>
<td>−</td>
</tr>
<tr>
<td>CLR WDT</td>
<td>WDT \leftarrow 0</td>
<td>TO#, PD#</td>
</tr>
<tr>
<td>SLEEP</td>
<td>go to low power consumption</td>
<td>TO#, PD#</td>
</tr>
</tbody>
</table>

### Operands

<table>
<thead>
<tr>
<th>f</th>
<th>name / address of register</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>bit location</td>
</tr>
</tbody>
</table>
Sample Program Fragments

RAM Initialization

```
CLRF STATUS ; STATUS ← 0
MOVLW 0x20 ; W ← 1st address in GPR bank 0
MOVWF FSR ; Indirect address register ← W

Bank0_LP
CLRF INDF0 ; address in GPR ← 0
INCF FSR ; FSR++ (next GPR address)
BTFSS FSR, 7 ; skip if (FSR<7> == 1) ⇒ FSR = 80h
GOTO Bank0_LP ; continue

; ** IF DEVICE HAS BANK1 **

MOVLW 0xA0 ; W ← 1st address in GPR bank 1
MOVWF FSR ; Indirect address register ← W

Bank1_LP
CLRF INDF0 ; address in GPR ← 0
INCF FSR ; FSR++ (next GPR address)
BTFSS STATUS, C ; skip if (STATUS<0> == 1) ⇒ FSR = 00h
GOTO Bank1_LP ; continue
```
Sample Program Fragments

Branch to address in new page

Prog:

movlw HIGH Prog10 ; W ← Prog10<15:8>
;
; operator HIGH reads bits <15:8> of pointer
movwf PCLATH ; PCLATH ← W

goto Prog10 ; PC<10:0> ← Prog10<7:0>
;
; PC<12:11> ← PCLATH<4:3>

Prog10:

; ; Prog10 labels some address in program memory
;
Sample Program Fragments
Computed goto

movlw HIGH Prog20    ; W ← Prog10<15:8>
movwf PCLATH         ; PCLATH ← W
movlw LOW Prog20      ; W ← Prog10<7:0>
movwf PCL             ; PCL ← Prog10<7:0>
                      ; PCH ← PCLATH<4:0>

Prog20:
;
; Prog10 labels some address in program memory
;
Sample Programs Fragments

if-else branch

*btfss f,b ; skip one instruction if
; bit b in register f = 1

goto Action2
Action1:
; instructions for Action1

goto Action3
Action2:
; instructions for Action2

Action3:
; instructions for Action3
Sample Programs Fragments

Static loop

\begin{verbatim}
movlw times ; W ← times
movwf COUNTER ; COUNTER ← W (times)

Loop:

; ; loop instructions
;

decfsz COUNTER, f ; COUNTER--
; COUNTER = 0 ⇒ skip next instruction

goto Loop ; next iteration

End:

;
;
\end{verbatim}
; Function call returns data at Table.INDEX
movlw HIGH Table ; W ← Table<15:8>
movwf PCLATH ; PCLATH ← W
movf INDEX, W ; W ← INDEX
call Table ; Call to subroutine table

; Table:
addwf PCL, f ; PCL ← PCL + W = PCL + INDEX
; computed goto
retlw 'A'; return with W ← 'A'
retlw 'B'
retlw 'C'
retlw 'D'
retlw 'E'

Data table in instruction memory
PIC MCU and the Outside World

**Oscillator**
Generates device clock
Four device clock periods per instruction cycle

**Ports**
Data I/O pins
- Electrical connections to external circuits
- Configured to function as
  - Digital I/O
  - Analog inputs to A/D converter

**Peripheral Modules**
Share data pins with general ports

<table>
<thead>
<tr>
<th>Module</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>Counts clock cycles → interrupt on preset count</td>
</tr>
<tr>
<td>A/D</td>
<td>Samples analog level → converts to digital representation</td>
</tr>
<tr>
<td>Comparator</td>
<td>Samples 2 analog levels → outputs bit (A1 &gt; A2)</td>
</tr>
<tr>
<td>USART</td>
<td>Bit-parallel ↔ bit-serial converter for communications</td>
</tr>
<tr>
<td>CCP</td>
<td>Capture/Compare/PWM (Pulse Width Modulation)</td>
</tr>
</tbody>
</table>

**Controls**
- Device dependent
- Power + ground
- Interrupt (INT)
- External clock
Configurable Oscillator Modes

**Quartz crystal time base**

Crystal connected between PIC pins **OSC1** and **OSC2**
Vibrates in electric field → piezoelectric resonance in voltage

Modes
- LP Low Frequency / Low Power Crystal — 32 kHz to 200 kHz
- XT Crystal/Resonator — 100 kHz to 4 MHz
- HS High Speed Crystal/Resonator — 8 MHz to 20 MHz

**Resistor/ Capacitor time base**

Capacitor discharges through resistor in time = $2\pi RC$
Oscillator frequency $f = 1 / (2\pi RC)$

Modes
- EXTRC — External RC connected between PIC pin **OSC1** and ground
- INTRC — Internal 4 MHz RC
- CLKOUT
  EXTRC or INTRC with instruction clock ($= f/4$) output on **OSC2**
Interrupts

**Interrupt**

Instruction at current PC executes
Instruction at current PC+1 fetched
Stack ← PC+2
PC ← interrupt pointer
On return from interrupt PC ← stack

**Interrupt sources**

External interrupt pin
Pin RB0 on some PIC devices (separate pin on other devices)

Peripheral modules
  General internal interrupt
  A/D
  Timer
  Comparator
  USART
  CCP
## Interrupt Control Register

### Interrupt Control Register (INTCON)

<table>
<thead>
<tr>
<th></th>
<th>GIE</th>
<th>PEIE</th>
<th>TOIE</th>
<th>INTE</th>
<th>RBIE</th>
<th>T0IF</th>
<th>INTF</th>
<th>RBIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **GIE** - Global Interrupt Enable
  - 1 = Enables all un-masked interrupts
  - 0 = Disables all interrupts

- **PEIE** - Peripheral Interrupt Enable
  - 1 = Enables all un-masked peripheral interrupts
  - 0 = Disables all peripheral interrupts

- **TOIE** - Overflow Interrupt Enable
  - 1 = Enables TMR0 overflow interrupt
  - 0 = Disables TMR0 overflow interrupt

- **INTE** - External Interrupt Enable
  - 1 = Enables INT external interrupt
  - 0 = Disables INT external interrupt

- **RBIE** - RB Port Change Interrupt Enable
  - 1 = Enables RB port change interrupt
  - 0 = Disables RB port change interrupt

- **T0IF** - Overflow Interrupt Flag
  - 1 = TMR0 register has overflowed
  - 0 = TMR0 register did not overflow

- **INTF** - External Interrupt Flag
  - 1 = INT external interrupt occurred
  - 0 = INT external interrupt did not occur

- **RBIF** - RB Port Change Interrupt Flag
  - 1 = At least one of RB7:RB4 pins changed state
  - 0 = None of RB7:RB4 pins have changed state
### Peripheral Interrupt Enable (PIE)

**Number of PIE registers device dependent**

<table>
<thead>
<tr>
<th>PIE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMR1IE</td>
<td>TMR1 Overflow</td>
</tr>
<tr>
<td>TMR2IE</td>
<td>TMR2 to PR2 Match</td>
</tr>
<tr>
<td>CCP1IE</td>
<td>CCP1</td>
</tr>
<tr>
<td>CCP2IE</td>
<td>CCP2</td>
</tr>
<tr>
<td>SSPIE</td>
<td>Synchronous Serial Port</td>
</tr>
<tr>
<td>RCIE</td>
<td>USART Receive</td>
</tr>
<tr>
<td>TXIE</td>
<td>USART Transmit</td>
</tr>
<tr>
<td>ADIE</td>
<td>A/D Converter</td>
</tr>
<tr>
<td>ADCIE</td>
<td>Slope A/D Converter Comparator Trip</td>
</tr>
<tr>
<td>OVFIE</td>
<td>Slope A/D TMR Overflow</td>
</tr>
<tr>
<td>PSPIE</td>
<td>Parallel Slave Port Read/Write</td>
</tr>
<tr>
<td>EEIE</td>
<td>EE Write Complete</td>
</tr>
<tr>
<td>LCDIE</td>
<td>LCD</td>
</tr>
<tr>
<td>CMIE</td>
<td>Comparator</td>
</tr>
</tbody>
</table>

1 = enable device interrupt  
0 = disable device interrupt
### Peripheral Interrupt Register (PIR) — 1

**Number of PIR registers device dependent**

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
</table>
| TMR1IE   | 1 = TMR1 register overflowed  
0 = TMR1 register did not overflow |
| TMR2IE   | Same as TMR1IE |
| CCP1IE   | CCP1 Interrupt Flag bit  
Capture Mode  
1 = TMR1 register capture occurred  
0 = No TMR1 register capture occurred  
Compare Mode  
1 = A TMR1 register compare match occurred  
0 = No TMR1 register compare match occurred  
PWM Mode  
Unused in this mode |
| CCP2IE   | Same as CCP1IE |
| SSPIE    | 1 = Transmission/reception complete  
0 = Waiting to transmit/receive |
| RCIE     | 1 = USART receive buffer RCREG full  
0 = USART receive buffer is empty |
Peripheral Interrupt Register (PIR) — 2

Number of PIR registers device dependent

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TXIE</td>
<td>USART transmit buffer TXREG empty</td>
<td>1 = USART transmit buffer TXREG empty 0 = USART transmit buffer is full</td>
</tr>
<tr>
<td>ADIE</td>
<td>A/D conversion complete</td>
<td>1 = A/D conversion complete 0 = A/D conversion not complete</td>
</tr>
<tr>
<td>ADCIE</td>
<td>A/D conversion complete</td>
<td>1 = A/D conversion complete 0 = A/D conversion not complete</td>
</tr>
<tr>
<td>OVFIE</td>
<td>Slope A/D TMR overflow</td>
<td>1 = Slope A/D TMR overflow 0 = Slope A/D TMR did not overflow</td>
</tr>
<tr>
<td>PSPIE</td>
<td>Read or write operation occurred</td>
<td>1 = Read or write operation occurred 0 = Read or write did not occur</td>
</tr>
<tr>
<td>EEIE</td>
<td>Data EEPROM write operation complete</td>
<td>1 = Data EEPROM write operation complete 0 = Data EEPROM write operation not complete</td>
</tr>
<tr>
<td>LCDIE</td>
<td>LCD interrupt occurred</td>
<td>1 = LCD interrupt occurred 0 = LCD interrupt did not occur</td>
</tr>
<tr>
<td>CMIE</td>
<td>Comparator input changed</td>
<td>1 = Comparator input changed 0 = Comparator input not changed</td>
</tr>
</tbody>
</table>
Interrupt Latency

**On interrupt**

1. Current instruction execution completes
2. Current instruction fetch completes
3. \( PC \leftarrow \) interrupt pointer

Latency ~ 3 to 4 instruction cycles
Interrupt Initialization + Enabling

\[
\text{PIE1\_MASK1 EQU B'01101010'} \quad ; \text{Interrupt Enable}
\]
\[
; \text{Register mask (device dependent)}
\]

; 
; 

CLRF STATUS \quad ; \text{Bank0}

CLRF INTCON \quad ; \text{Disable interrupts during configuration}

CLRF PIR1 \quad ; \text{Clear flags}

BSF STATUS, RP0 \quad ; \text{Bank1}

MOVLW PIE1\_MASK1 \quad ; \text{set PIE1 via W}

MOVWF PIE1

BCF STATUS, RP0 \quad ; \text{Bank0}

BSF INTCON, GIE \quad ; \text{Enable Interrupts}
Macros for Register Save / Restore

PUSH_MACRO MACRO
  MOVWF W_TEMP
  SWAPF STATUS,W
  MOVWF STATUS_TEMP
ENDM

; Save register contents
; Temporary register ← W
; W ← swap STATUS nibbles
; Temporary register ← STATUS
; End this Macro

POP_MACRO MACRO
  SWAPF STATUS_TEMP,W
  MOVWF STATUS
  SWAPF W_TEMP,F
  SWAPF W_TEMP,W
ENDM

; Restore register contents
; W ← swap STATUS
; STATUS ← W
; W_Temp ← swap W_Temp
; W ← swap W_Temp s
; no affect on STATUS
; End this Macro
Typical Interrupt Service Routine (ISR) — 1

```assembly
org ISR_ADDR ; store at ISR address
PUSH_MACRO ; save context registers W, STATUS
CLRF STATUS ; Bank0

; switch implementation in PIC assembly language
BTFSC PIR1, TMR1IF ; skip next if (PIR1<TMR1IF> == 1)
GOTO T1_INT ; go to Timer1 ISR
BTFSC PIR1, ADIF ; skip next if (PIR1<ADIF> == 1)
GOTO AD_INT ; go to A/D ISR
BTFSC PIR1, LCDIF ; skip next if (PIR1<LCDIF> == 1)
GOTO LCD_INT ; go to LCD ISR
BTFSC INTCON, RBIF ; skip next if (PIR1<RBIF> == 1)
GOTO PORTB_INT ; go to PortB ISR
GOTO INT_ERROR_LP1 ; default ISR
```
Typical Interrupt Service Routine (ISR) — 2

T1_INT

: ; Timer1 overflow routine

BCF PIR1, TMR1IF ; Clear Timer1 overflow interrupt flag

GOTO END_ISR ; Leave ISR

AD_INT ; Routine when A/D completes

: ; Routine when A/D completes

BCF PIR1, ADIF ; Clear A/D interrupt flag

GOTO END_ISR ; Leave ISR

LCD_INT ; LCD Frame routine

: ; LCD Frame routine

BCF PIR1, LCDIF ; Clear LCD interrupt flag

GOTO END_ISR ; Leave ISR

PORTB_INT ; PortB change routine

: ; PortB change routine

END_ISR ; Leave ISR

POP_MACRO ; Restore registers

RETFIE ; Return and enable interrupts
Timers

**Watchdog timer (WDT)**

Normal program resets timer before timeout (18 ms)

Timeout

Non-sleep mode — MCU resets

Sleep mode — MCU wakes up → executes instruction following sleep

Configured in `OPTION_REG` SFR

**Timer0**

Generic programmable 8-bit timer/counter

Shares prescaler (divide by $2^k$, $k = 0,...,8$) with WDT

Configured in `OPTION_REG` SFR

**Timer1**

Generic programmable 16-bit timer/counter

Read / write two 8-bit registers

**Timer2**

Generic programmable 8-bit timer/counter

Time base for PWM mode
Watchdog Timer (WDT)

**Time base**

Internal RC oscillator

Timer0 clock source

From TMR0 Clock Source
## OPTION_REG SFR (File Address 081h)

<table>
<thead>
<tr>
<th>RBPU</th>
<th>INTEDG</th>
<th>T0CS</th>
<th>T0SE</th>
<th>PSA</th>
<th>PS2</th>
<th>PS1</th>
<th>PS0</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**RBPU**  
**Weak Pull-up Enable**  
1 = Weak pull-ups are disabled  
0 = Weak pull-ups are enabled by port latch values  
_**Underline**_ — active = 0 / inactive = 1

**INTEDG**  
**Interrupt Edge Select**  
1 = Interrupt on rising edge of INT pin  
0 = Interrupt on falling edge of INT pin

**T0CS**  
**TMR0 Clock Source Select**  
1 = Transition on TOCKI pin  
0 = Internal instruction cycle clock (CLKOUT)

**T0SE**  
**TMR0 Source Edge Select**  
1 = Increment on high-to-low transition on TOCKI pin  
0 = Increment on low-to-high transition on TOCKI pin

**PSA**  
**Prescaler Assignment**  
1 = Prescaler is assigned to the WDT (watchdog)  
0 = Prescaler is assigned to the Timer0 module

### Prescaler Rate Select

<table>
<thead>
<tr>
<th>PS2 : PS0</th>
<th>TMR0</th>
<th>WDT</th>
<th>PS2 : PS0</th>
<th>TMR0</th>
<th>WDT</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>1:2</td>
<td>1:1</td>
<td>100</td>
<td>1:32</td>
<td>1:16</td>
</tr>
<tr>
<td>001</td>
<td>1:4</td>
<td>1:2</td>
<td>101</td>
<td>1:64</td>
<td>1:32</td>
</tr>
<tr>
<td>010</td>
<td>1:8</td>
<td>1:4</td>
<td>110</td>
<td>1:128</td>
<td>1:64</td>
</tr>
<tr>
<td>011</td>
<td>1:16</td>
<td>1:8</td>
<td>111</td>
<td>1:256</td>
<td>1:128</td>
</tr>
</tbody>
</table>
8-bit timer/counter

Readable / writable at TMR0 SFR (File Address 081h)

8-bit software programmable prescaler

- Divide input pulse train (slows time scale)
- Scale by 1:1, 1:2, 1:4, ... , 1:128

Selectable clock source

- External / internal

Interrupt on overflow FFh → 00h

Edge select (phase synchronization with external clock)
Timer0 Operation

**Timer mode**

\[ T0CS = 0 \]

\[ \text{TMR0}++ \quad \text{on every instruction cycle (without prescaler)} \]

Write to \text{TMR0} register ⇒ no increment for two instruction cycles

**Counter mode**

\[ T0CS = 1 \]

\[ \text{TMR0}++ \quad \text{on every rising or falling edge of T0CKI (external clock)} \]

Edge determined by \text{T0SE} bit

**Prescaler**

Set by PSA control bits

**TMR0 Interrupt**

Generated on overflow \( \text{FFh} \rightarrow \text{00h} \)

Sets bit \text{T0IF (INTCON<2>)}

Timer0 interrupt service routine

Clear \text{T0IF}

Re-enable interrupt
Initialize Timer0 with Internal Clock Source

CLRF TMR0 ; Clear Timer0 register
CLRF INTCON ; Disable interrupts and clear T0IF
BSF STATUS, RP0 ; Bank1
MOVLW 0xC3 ; Disable PortB pull-ups
    ; C3 = 11000011
MOVWF OPTION_REG ; Interrupt on rising edge of RB0
    ; Timer0 increment from internal clock
    ; Prescale = 1:16.
BCF STATUS, RP0 ; Bank0

T0_OVFL_WAIT
BTFSS INTCON, T0IF ; poll overflow bit
GOTO T0_OVFL_WAIT ; on timer overflow
Timer1

16-bit timer/counter

TMR1 pair TMR1H:TMRL

Readable and writable 8-bit registers

Counter 0000h to FFFFh with rollover to 0000h

Generate Timer1 interrupt on rollover (if enabled)

Modes

Synchronous timer

TMR1++ on every instruction cycle (Fosc / 4)

Asynchronous counter

TMR1++ on rising edge of input pin

Synchronous counter

TMR1++ on rising edge of sampled input pin

Synchronized to internal clock Tosc

Sample input pin on rising edge of Tosc

Input must be high / low for at least 2Tosc
## T1CON SFR

<table>
<thead>
<tr>
<th></th>
<th>T1CKPS1</th>
<th>T1CKPS0</th>
<th>T1OScen</th>
<th>T1SYNC</th>
<th>TMR1CS</th>
<th>TMR1ON</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T1CKPS1</th>
<th>Timer1 Input Clock Prescale Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1:8 Prescale value</td>
</tr>
<tr>
<td>10</td>
<td>1:4 Prescale value</td>
</tr>
<tr>
<td>01</td>
<td>1:2 Prescale value</td>
</tr>
<tr>
<td>00</td>
<td>1:1 Prescale value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T1OScen</th>
<th>Timer1 Oscillator Enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Oscillator mode enabled</td>
</tr>
<tr>
<td>0</td>
<td>Oscillator mode disabled</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T1SYNC</th>
<th>Timer1 External Clock Input Synchronization Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Asynchronous Counter Mode</td>
</tr>
<tr>
<td>0</td>
<td>Synchronous Counter Mode</td>
</tr>
</tbody>
</table>

When TMR1CS = 1:

- TMR1CS = 1 (Counter Mode)
  - 1 = Asynchronous Counter Mode
  - 0 = Synchronous Counter Mode

When TMR1CS = 0:

- TMR1CS = 0 (Timer Mode)
  - Ignored

<table>
<thead>
<tr>
<th>TMR1CS</th>
<th>Timer1 Clock Source Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Counter Mode (count external clock)</td>
</tr>
<tr>
<td>0</td>
<td>Timer Mode (count internal clock F_OSC / 4)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TMR1ON</th>
<th>Timer1 On</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Enables Timer1</td>
</tr>
<tr>
<td>0</td>
<td>Stops Timer1</td>
</tr>
</tbody>
</table>
Timer1 Operation

Set TMR1IF flag bit on Overflow

CLK in / OSC out

T1OSC

T1OSI

Oscillator Mode ⇒ T1CKI ← T1OSI

T1OSC

FOSC/4 Internal Clock Timer Mode

T1OSCN

Enable Oscillator(1)

T1SYNC

Prescaler

1, 2, 4, 8

T1CKPS1:T1CKPS0

Timer Mode

Counter Mode

Synchronized clock input

Async Input

Synchronization

det

SLEEP input

1

0

TMR1ON

on/off

CCP Special Trigger

TMR1

CLR

TMR1H

TMR1L
Reading Timer1

; All interrupts disabled

    MOVF TMR1H, W  ; W ← high byte
    MOVWF TMPH    ; TMPH ← W
    MOVF TMR1L, W  ; W ← low byte
    MOVWF TMPL    ; TMPL ← W

; TMR1L can roll-over between reads of high and low bytes

    MOVF TMR1H, W  ; W ← high byte again
    SUBWF TMPH, W   ; Verify high byte
    BTFSC STATUS,Z  ; bad read (Z = 0 ⇒ not equal) ⇒ re-do
    GOTO CONTINUE

; New reading ⇒ good value.

    MOVF TMR1H, W  ; W ← high byte
    MOVWF TMPH    ; TMPH ← W
    MOVF TMR1L, W  ; W ← low byte
    MOVWF TMPL    ; TMPL ← W

; Re-enable interrupts (if required)

CONTINUE

    ; Continue
Writing Timer1

; All interrupts are disabled

    CLRF TMR1L ; Clear Low byte

    ; Prevents rollover to TMR1H

    MOVLW HI_BYTE ; W ← HI_BYTE
    MOVWF TMR1H, F ; TMR1H ← W

    MOVLW LO_BYTE ; W ← LO_BYTE
    MOVWF TMR1L, F ; TMR1L ← W

; Re-enable interrupts (if required)

CONTINUE

; Continue
Readable / writable 8-bit timer

Prescaler

Period register \( \text{PR2} \)

Readable / writable

\[ \text{TMR2} = \text{PR2} \implies \text{reset} \; (\text{TMR2} \leftarrow 0) \]

Postscaler

Diagram:

- \( F_{\text{osc}}/4 \)
- Prescaler: 1:1, 1:4, 1:16
- \( T2\text{CKPS1}:T2\text{CKPS0} \)
- 2
- Comparator
- \( \text{PR2 reg} \)
- Postscaler: 1:1 to 1:16
- \( 4 \)
- \( \text{EQ} \)
- \( \text{TMR2 output} \)
- \( \text{SETs flag bit TMR2IF} \)
- Reset
## T2CON SFR

<table>
<thead>
<tr>
<th></th>
<th>TOUTPS3</th>
<th>TOUTPS2</th>
<th>TOUTPS1</th>
<th>TOUTPS0</th>
<th>TMR2ON</th>
<th>T2CKPS1</th>
<th>T2CKPS0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TOUTPS3 : 0**

**Timer2 Output Postscale Select**

- 0000 = 1:1 Postscale
- 0001 = 1:2 Postscale
- 0010 = 1:3 Postscale
- 0011 = 1:4 Postscale
- 1111 = 1:16 Postscale

**TMR2ON**

**Timer2 On**

- 1 = Timer2 is on
- 0 = Timer2 is off

**T2CKPS1 : 0**

**Timer2 Clock Prescale Select**

- 00 = Prescaler is 1
- 01 = Prescaler is 4
- 1x = Prescaler is 16
Ports

I/O pins

Electrical connections to external circuits

Configurable in SFR ADCON1 as

Digital I/O
Analog inputs to A/D converter

Mid-Range PIC configurations

Minimal — Port A (6 pins) + Port B (8 pins)
Maximal — Port A (6 pins), Port B (8 pins), ..., Port G (8 pins)

Special Function Registers

Data

PORTA, ..., PORTG
PORTi<x> = data bit on pin x of port i

Direction

TRISA, ..., TRISG
TRISi<x> = 1 ⇒ pin x of port i is Input
TRISi<x> = 0 ⇒ pin x of port i is Output
Port Access

Output
Set TRIS\textsubscript{i<x>} = 0
Write data bit to PORT\textsubscript{i<x>}

Input
Set TRIS\textsubscript{i<x>} = 1
Read data bit from PORT\textsubscript{i<x>}

Order of operations
Read
Reads levels on physical I/O pins (not data register file)
Write
Implemented as read \rightarrow modify
Causes update of all input data registers from physical I/O pins
Program must read all required inputs before any write
PORTA

5 general purpose I/O pins

RA5 and RA3:RA0

Standard electrical behavior

TTL input levels and CMOS output drivers

Special input

RA4

Schmitt trigger input

Threshold decision converts input to binary \((RA4 > \text{threshold})\)

Open drain output

Permits specialize electrical functions on output

Wired-OR, analog weighting, ...
Initializing PORTA

CLRF STATUS ; Bank0
CLRF PORTA ; Initialize PORTA
BSF STATUS, RP0 ; Select Bank1

MOVLW 0xCF ; Initialize data directions
; CFh = 11001111
; = x x Out Out In In In In

MOVWF TRISA ; PORTA<3:0> = inputs
; PORTA<5:4> = outputs
; TRISA<7:6> always read 0
PORTB

8 general purpose I/O pins

RB7 : RB0

Standard electrical behavior

TTL input levels and CMOS output drivers

Interrupt on change

Input pins RB7 : RB4

Inputs compared with previous read of PORTB

OR(compare bits) = 1 → RB Port Change Interrupt

Can wake device from SLEEP

Example — wake-up on key press

Clear interrupt

Read or write PORTB

Clear flag bit RBIF
Ports C to E

Ports C to E

8 binary I/O pins

Ri7:Ri0, i = C, D, E

Schmitt trigger on each input pin

Ports F and G

Ri7:Ri0, i = F, G

8 binary inputs

Schmitt trigger on each input

8 LCD driver outputs

Direct connection to 7-segment display
Analog-to-Digital (A/D) Converter Module

**Converts analog input signals**
- Sample and hold
- One of 8 analog inputs (channels)
- Conversion to 8-bit binary number

**Analog reference voltage**
- Software selectable
  - Device supply voltage
  - Voltage level on $V_{REF}$ pin
- Can operate in sleep mode

**Three registers**
- A/D Result Register (**ADRES**) 
- A/D Control Register0 (**ADCON0**) 
- A/D Control Register1 (**ADCON1**)
## ADCON0 SFR

<table>
<thead>
<tr>
<th>ADCS1</th>
<th>ADCS0</th>
<th>CHS2</th>
<th>CHS1</th>
<th>CHS0</th>
<th>GO/DONE</th>
<th>Resv</th>
<th>ADON</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

### A/ D Conversion Clock Select

<table>
<thead>
<tr>
<th>ADCS1 : ADCS0</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>f_{OSC}/2</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>f_{OSC}/8</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>f_{OSC}/32</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>f_{RC}</td>
<td>(internal A/D RC osc)</td>
</tr>
</tbody>
</table>

### Analog Channel Select

<table>
<thead>
<tr>
<th>CHS2 : CHS0</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000</td>
<td>channel 0 (AN0)</td>
</tr>
<tr>
<td>001</td>
<td>001</td>
<td>channel 1 (AN1)</td>
</tr>
<tr>
<td>010</td>
<td>010</td>
<td>channel 2 (AN2)</td>
</tr>
<tr>
<td>011</td>
<td>011</td>
<td>channel 3 (AN3)</td>
</tr>
<tr>
<td>110</td>
<td>110</td>
<td>channel 4 (AN4)</td>
</tr>
<tr>
<td>111</td>
<td>111</td>
<td>channel 7 (AN7)</td>
</tr>
</tbody>
</table>

### A/ D Conversion Status

<table>
<thead>
<tr>
<th>GO/DONE</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>in progress</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>not in progress</td>
</tr>
</tbody>
</table>

### Reserved

<table>
<thead>
<tr>
<th>Reserved</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

### A/ D On

<table>
<thead>
<tr>
<th>ADON</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>activated</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>deactivated</td>
</tr>
</tbody>
</table>
### ADCON1 SFR

<table>
<thead>
<tr>
<th>PCFG2 : PCFG0</th>
<th>AN7</th>
<th>AN6</th>
<th>AN5</th>
<th>AN4</th>
<th>AN3</th>
<th>AN2</th>
<th>AN1</th>
<th>AN0</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>001</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>VREF</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>010</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>011</td>
<td>D</td>
<td>D</td>
<td>A</td>
<td>A</td>
<td>VREF</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>100</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>A</td>
<td>D</td>
<td>A</td>
</tr>
<tr>
<td>101</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>VREF</td>
<td>D</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>11x</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

- **A**: Port pin configured for analog input
- **D**: Port pin configured for digital I/O
- **AN3 = VREF**: Conversion compares to reference voltage $V_{REF} = \text{voltage on AN3}$
- **AN3 = D**: Conversion compares to reference voltage $V_{REF} = \text{device supply voltage}$
### Operation of A/D Converter

#### Configure A/D module
- Analog pins + voltage reference + digital I/O in **ADCON1**
- Select A/D input channel (**ADCON0**)
- Select A/D conversion clock (**ADCON0**)
- Activate A/D module (**ADCON0**)

#### Configure A/D interrupt (optional)
- Clear **ADIF**
- Set **ADIE + GIE**

#### Start conversion
- Set **GO/DONE** bit (**ADCON0**)

#### Wait for A/D conversion to complete
- Poll **GO/DONE** until cleared or wait for A/D interrupt

#### Read result
- A/D Result register (**ADRES**)

#### Repeat
A/D Conversion

BSF STATUS, RP0 ; Bank1
CLRF ADCON1 ; Configure inputs as analog
BSF PIE1, ADIE ; Enable A/D interrupts
BCF STATUS, RP0 ; Bank0
MOVLW 0xC1 ; C1h = 11000001
MOVWF ADCON0 ; Internal RC, A/D active, Channel 0
BCF PIR1, ADIF ; Clear A/D interrupt flag
BSF INTCON, PEIE ; Enable peripheral interrupts
BSF INTCON, GIE ; Enable all interrupts

; Wait required sampling time for selected input

BSF ADCON0, GO ; ADCON0<2> ← 1 ⇒ Start conversion
; On completion ADIF bit ← 1 and GO/DONE ← 0
Comparator

Two analog comparators

Inputs shared with I/O pins
Access via CMCON SRF (device-dependent file address)

Operation

Analog input at \( V_{IN}^+ < V_{IN}^- \Rightarrow output = \text{binary 0} \)
Analog input at \( V_{IN}^+ > V_{IN}^- \Rightarrow output = \text{binary 1} \)

CMCON SFR

<table>
<thead>
<tr>
<th>C2OUT</th>
<th>C1OUT</th>
<th>CIS</th>
<th>CM2</th>
<th>CM1</th>
<th>CM0</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C2OUT</th>
<th>C1OUT</th>
<th>Comparator Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>( 1 = V_{IN}^+ &gt; V_{IN}^- )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( 0 = V_{IN}^+ &gt; V_{IN}^- )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CIS</th>
<th>CM2 : CM0</th>
<th>Comparator Input Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>See table on following slides</td>
</tr>
</tbody>
</table>

\[ V_{IN}^+ \quad + \quad \text{Output} \]
\[ V_{IN}^- \quad - \]

response time

See table on following slides
Comparator Modes — 1

CM2:CM0 = 000
Comparators Reset (POR Default Value)

- RA0/AN0: A → VIN- → C1 → Off (Read as '0')
- RA3/AN3: A → VIN+ → C1 → Off (Read as '0')
- RA1/AN1: A → VIN- → C2 → Off (Read as '0')
- RA2/AN2: A → VIN+ → C2 → Off (Read as '0')

CM2:CM0 = 111
Comparators Off

- RA0/AN0: D → VIN- → C1 → Off (Read as '0')
- RA3/AN3: D → VIN+ → C1 → Off (Read as '0')
- RA1/AN1: D → VIN- → C2 → Off (Read as '0')
- RA2/AN2: D → VIN+ → C2 → Off (Read as '0')

CM2:CM0 = 100
Two Independent Comparators

- RA0/AN0: A → VIN- → C1 → C1OUT
- RA3/AN3: A → VIN+ → C1 → C1OUT
- RA1/AN1: A → VIN- → C2 → C2OUT
- RA2/AN2: A → VIN+ → C2 → C2OUT

CM2:CM0 = 010
Four Inputs Multiplexed to Two Comparators

- RA0/AN0: A → C1 with CIS = 0
- RA3/AN3: A → C1 with CIS = 1
- RA1/AN1: A → C2 with CIS = 0
- RA2/AN2: A → C2 with CIS = 1

From VREF Module
Comparitor Modes — 2

CM2:CM0 = 011
Two Common Reference Comparators

CM2:CM0 = 110
Two Common Reference Comparators with Outputs

CM2:CM0 = 101
One Independent Comparator

CM2:CM0 = 001
Three Inputs Multiplexed to Two Comparators
Initialize Comparator

FLAG_REG EQU 0x20 ; FLAG_REG points to address 20h
CLRF FLAG_REG ; flag register ← 0
CLRF PORTA ; PORTA ← 0
ANDLW 0xC0 ; Mask comparator bits W<5:0> ← 0
IORWF FLAG_REG,F ; FLAG_REG ← FLAG_REG OR W
MOVLW 0x03 ; Init comparator mode
MOVWF CMCON ; CM<2:0> = 011 (2 common reference)
BSF STATUS,RP0 ; Bank1
MOVLW 0x07 ; Initialize data direction
MOVWF TRISA ; Set RA<2:0> as inputs
 ; RA<4:3> as outputs
 ; TRISA<7:5> read 0
BCF STATUS,RP0 ; Bank0
CALL DELAY 10 ; 10ms delay
MOVF CMCON,F ; Read CMCON (enter read mode)
BCF PIR1,CMIF ; Clear pending interrupts
BSF STATUS,RP0 ; Bank1
BSF PIE1,CMIE ; Enable comparator interrupts
BSF STATUS,RP0 ; Bank0
BSF INTCON,PEIE ; Enable peripheral interrupts
BSF INTCON,GIE ; Global interrupt enable
Universal Synchronous / Asynchronous Receiver / Transmitter

Serial Communications Interface (SCI)
   PC serial port / modem

Transmit
   Parallel → serial
   Data byte as 8 serial bits

Receive
   Serial → parallel
   Assemble 8 bits as data byte

Modes

Asynchronous
   Full duplex — simultaneous transmit + receive

Synchronous
   Half duplex — transmit or receive
   Master — synchronize data to internal clock
   Slave — synchronize data to external clock
**USART Transmit Operation**

**Data**

Byte → **TXREG** → framing **TSR** → bit **FIFO** → **TX** pin

Framing — add start bit / parity bit

---

**Diagram:**
- TXIF
- TXIE
- Interrupt on empty TXREG
- TXIF
- TXIE
- Interrupt
- TXEN
- Baud Rate CLK
- Baud Rate Generator
- MSb
- (8)
- TSR register
- Data Bus
- 8
- TXREG register
- LSB
- 0
- Pin Buffer and Control
- TX/CK pin
- TRMT
- SPEN
- TSR full / empty
- Port Enable
- Transmit Speed
- Parity bit
- TX9D
- TX9
**USART Receive Operation**

**Data**

RX pin → bit FIFO → RSR → RCREG → byte

**Framing**

Identify data between stop bits
Capture / Compare / PWM (CCP) Module

SFRs
- CCP control register (CCPON)
- CCPR High byte / Low byte (CCPRH / CCPRL)

I/O pin
- CPP pin (CPPx) — device-dependent pin configured in TRIS SFR

Capture Mode
- Captures 16-bit value of register TMR1 on CCPx event
  - Every falling edge / rising edge / 4th rising edge / 16th rising edge
  - Triggers interrupt

Compare mode
- Compare 16-bit (CCPR == TMR1)
  - Configurable response on match
    - CCPx ← 0 / 1
  - Interrupt with no change on CPPx

Pulse Width Modulation (PWM) mode
- Generates duty cycle waveform on CCPx
### CCPxCON SFR

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>DCxB1 : DCxB0</td>
</tr>
<tr>
<td>6</td>
<td>PWM Duty Cycle bit1 and bit0</td>
</tr>
<tr>
<td>5</td>
<td>DCx1:DCx0 of 10-bit PWM duty cycle</td>
</tr>
<tr>
<td>4</td>
<td>DCx9:DCx2 in CCPRxL</td>
</tr>
<tr>
<td>3</td>
<td>CCPxM3 : CCPxM0</td>
</tr>
<tr>
<td>2</td>
<td>CCPx Mode Select bits</td>
</tr>
<tr>
<td>1</td>
<td>0000 = Capture/Compare/PWM off (resets CCPx module)</td>
</tr>
<tr>
<td>0</td>
<td>0100 = Capture mode, every falling edge</td>
</tr>
<tr>
<td></td>
<td>0101 = Capture mode, every rising edge</td>
</tr>
<tr>
<td></td>
<td>0110 = Capture mode, every 4th rising edge</td>
</tr>
<tr>
<td></td>
<td>0111 = Capture mode, every 16th rising edge</td>
</tr>
<tr>
<td></td>
<td>1000 = Compare mode, CCP low to high</td>
</tr>
<tr>
<td></td>
<td>1001 = Compare mode, CCP high to low</td>
</tr>
<tr>
<td></td>
<td>1010 = Compare mode, software interrupt on match</td>
</tr>
<tr>
<td></td>
<td>1011 = Compare mode, Trigger special event</td>
</tr>
<tr>
<td></td>
<td>11xx = PWM mode</td>
</tr>
</tbody>
</table>
Capture Mode

Event

Input pin CCPx divided by prescaler sampled on rising / falling edge

Interrupt flag, \( x = 1, 2 \)

Set flag bit CCPxIF

CCPx Pin

Prescaler
\( \div 1, 4, 16 \)

and edge detect

Q’s

Configuration
CCP control register, \( x = 1, 2 \)

CCPxCON<3:0>

Capture Enable

CCPRxH

CCPRxL

16-bit value in Timer1

TMR1H

TMR1L

Capture CCPxR = CCPRxH:CCPRxL, \( x = 1, 2 \)
Compare Mode

16-bit compare

(CCPRx == TMR1), x = 1, 2

Reset Timer1

Special Event

Set flag bit CCPxIF

Trigger

Interrupt

16-bit preset value in CCPRx

Output Logic

match

Comparator

CCPRxH

CCPRxL

TMR1H

TMR1L

16-bit value of Timer1

CCPx Pin

TRIS

Output Enable

CCPxCON<3:0>

Mode Select

Q

SR
Pulse Width Modulation (PWM) mode

Generates duty cycle waveform

Period = (PR2 + 1) × 4 × prescale × T_{osc}

Duty cycle = DC × prescale × T_{osc}

0 ≤ PR2 ≤ 255
0 ≤ DC ≤ 1023
**Duty Cycle**

\[
\Delta T_{PWM} = 4 \times P \times T_{OSC}
\]

\[
T_{PWM} = 4 \times (PR2 + 1) \times P \times T_{OSC}
\]

\[
T_{ON} = DC \times P \times T_{OSC}
\]

\[
\Delta T_{ON} = P \times T_{OSC}
\]

\[
\frac{T_{ON}}{T_{PWM}} = \frac{DC \times P \times T_{OSC}}{4 \times (PR2 + 1) \times P \times T_{OSC}} = \frac{DC}{4 \times (PR2 + 1)}
\]

\[
\frac{T_{ON}}{T_{PWM}} \leq 1 \Rightarrow DC \leq 4 \times (PR2 + 1)
\]

**Controllability**

\[
\frac{\Delta T_{ON}}{T_{PWM}} = \frac{P \times T_{OSC}}{4 \times (PR2 + 1) \times P \times T_{OSC}} = \frac{1}{4 \times (PR2 + 1)}
\]

**Resolution**

\[
DC \leq 4 \times (PR2 + 1) \Rightarrow 2^r = \frac{T_{ON}}{T_{OSC}} \leq 4 \times (PR2 + 1) \times P \Rightarrow r = \frac{\log\left(4 \times (PR2 + 1) \times P\right)}{\log(2)}
\]
Duty Cycle Example

Frequency and duty cycle from given parameters

\[ f_{\text{OSC}} = 20 \text{ MHz} \Rightarrow T_{\text{OSC}} = (20 \text{ MHz})^{-1} = 50 \text{ ns} \]

\[ P = 1 \]

\[ PR2 = 63 \Rightarrow T_{\text{PWM}} = 4 \times 64 \times 50 \text{ ns} = 12.8 \mu\text{s} \]

\[ f_{\text{PWM}} = (12.8 \mu\text{s})^{-1} = 78.125 \text{ kHz} \]

\[ DC = 32 \Rightarrow T_{\text{ON}} = 32 \times 50 \text{ ns} = 1.6 \text{ ms} \]

\[ \frac{T_{\text{ON}}}{T_{\text{PWM}}} = \frac{32}{4 \times 64} = 0.125 = 12.5\% \]

\[ \Delta T_{\text{ON}} = \frac{1}{4 \times 64} = \frac{1}{256} = 0.00390625 \]

\[ 2^r = \frac{T_{\text{ON}}}{T_{\text{OSC}}} \leq 4 \times 64 \Rightarrow r = \frac{\log(256)}{\log(2)} = 8 \]
PWM Example

Choosing parameters

**Internal oscillator**

\[ f_{OSC} = 4 \text{ MHz} \Rightarrow T_{OSC} = 0.25 \mu s \]

**PWM frequency**

\[ T_{PWM} = 1 \text{ ms} = 4 \times (PR2 + 1) \times P \times 0.25 \mu s = (PR2 + 1) \times P \times 1 \mu s \]

Require \((PR2 + 1) \times P = 1000\)

**Preset and PR2**

\[ P = 4 \Rightarrow PR2 + 1 = 250 \Rightarrow PR2 = 249 = 0xF9 \]
\[ \Delta T_{ON} = P \times T_{OSC} = 1 \mu s \]

**Duty cycle = 10%**

\[ T_{ON} = 0.10 \times 1 \text{ ms} = 100 \mu s \]
\[ DC = 0.10 \times 4 \times (PR2 + 1) = 100 = 0x064 \Rightarrow DCH = 0x19 = 25 \quad DCL = 0 \]
**PWM Example**

**Code**

List p = 16F873

```assembly
#include "P16F873.INC"

Init_pwm:
    movlw 0x01 ; Stop Timer2
    movwf T2CON ; Prescaler ← 4
    clrf CCP1CON ; Reset module CCP1
    clrf TMR2 ; Timer2 ← 0
    movlw .25 ; 10% duty cycle
    movwf CCPR1L ; DC1B9:DC1B2
    bsf STATUS, RP0 ; Bank 1
    movlw .249 ; Timer2
    movwf PR2
    bcf PIE1, TMR2IE ; Disable Timer2 interrupt
    bcf PIE1, CCP1IE ; Disable CCP1 interrupt
    bcf TRISC, 2 ; Pin CCP1 = output
    bcf STATUS, RP0 ; Bank 0
    clrf PIR1 ; Clear interrupt flags
    movlw 0x0C ; CCP1 in PWM mode
    movwf CCP1CON
    bsf T2CON, TMR2ON ; Start Timer2
    return

TON_pwm:
    movwf CCPR1L ; Call to change
    return ; Duty cycle
end
```
Data EEPROM

Additional long term data memory
  Internal EEPROM

Indirect addressing
  Not directly mapped in register file space
  Access through SFRs
    EECON1
      Control bits
    EECON2
      Initiates read / write operation
      Virtual register — not physically implemented
    EEDATA
      8-bit data for read / write
    EEADR
      Access address in EEPROM
      8-bit address ⇒ 256 EEPROM locations
### EECON1 SFR

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value Interpretation</th>
</tr>
</thead>
</table>
| 7   | EEIF Write Operation Interrupt Flag| 1 = Write operation completed  
0 = Write operation not complete / not started                                      |
| 6   | WRERR Error Flag                   | 1 = Write operation prematurely terminated  
0 = Write operation completed                                                      |
| 5   | WREN Write Enable                  | 1 = Allows write cycles  
0 = Inhibits write to data EEPROM                                                   |
| 4   | WR Write Control                   | 1 = Initiates write cycle  
0 = Write cycle to data EEPROM is complete (cleared by hardware)                   |
| 3   | RD Read Control                    | 1 = Initiates read  
0 = Does not initiate an EEPROM read (cleared by hardware)                           |
EECON2 SFR

Not physical register

Read $\text{EECON2} \rightarrow 0$

SFR access to EEPROM write hardware

Data EEPROM write sequence

$\text{EEDATA} \leftarrow \text{data}$

$\text{EEADR} \leftarrow \text{address\_for\_write}$

$W \leftarrow 55h$

$\text{EECON2} \leftarrow W$

$W \leftarrow AAh$

$\text{EECON2} \leftarrow W$ ; $\text{EECON2} \leftarrow 55AAh$

$\text{EECON1}_W \leftarrow 1$ ; initiate (write control bit set)
**EEPROM Read / Write / Verify — 1**

**Read**

\[
\begin{align*}
&\text{BCF STATUS, RP0} ; \text{ Bank0} \\
&\text{MOVLW CONFIG_ADDR} ; \text{ Address in Data EEPROM} \\
&\text{MOVWF EEADR} ; \text{ Set read address} \\
&\text{BSF STATUS, RP0} ; \text{ Set Bank1} \\
&\text{BSF EECON1, RD} ; \text{ Initiate EEPROM Read} \\
&\text{BCF STATUS, RP0} ; \text{ Set Bank0} \\
&\text{MOVF EEDATA, W} ; \text{ W ← EEDATA}
\end{align*}
\]

**Write**

\[
\begin{align*}
&\text{BSF STATUS, RP0} ; \text{ Bank1} \\
&\text{BCF INTCON, GIE} ; \text{ Disable INTs} \\
&\text{BSF EECON1, WREN} ; \text{ Enable write} \\
&\text{MOVLW 55h} ; \text{ W ← 55h} \\
&\text{MOVWF EECON2} ; \text{ EECON2 ← W} \\
&\text{MOVLW AAh} ; \text{ W ← AAh} \\
&\text{MOVWF EECON2} ; \text{ EECON2 ← W} \\
&\text{BSF EECON1,WR} ; \text{ Set WR bit (initiates write)} \\
&\text{BSF INTCON, GIE} ; \text{ Enable INTs}
\end{align*}
\]
EEPROM Read / Write / Verify — 2

Verify

BCF STATUS, RP0 ; Bank0
MOVF EEDATA, W ; copy write request data to W
BSF STATUS, RP0 ; Bank1

READ

BSF EECON1, RD ; Initiate read
BCF STATUS, RP0 ; Bank0
SUBWF EEDATA, W ; W ← write request – read
BTFSS STATUS, Z ; Skip next if (Z == 1)
GOTO WRITE_ERR ; Handle write error
## PIC Configuration Bits — 1

**Determines certain device modes**
- Oscillator mode, WDT reset, copy protection

**Sets device state on power-up**
- Configured during EEPROM programming
- Mapped to program memory location 2007h
- Not accessible at run time

| CP1 : CP0 | Code Protection                | 11 = Code protection off  
|          |                               | 10 = device dependent  
|          |                               | 01 = device dependent  
|          |                               | 00 = memory code protected |
| DP       | Data EEPROM Code Protection   | 1 = Code protection off  
|          |                               | 0 = Data EEPROM Memory code protected |
| BODEN    | Brown-out Reset Enable        | 1 = BOR enabled  
|          |                               | 0 = BOR disabled |
| PWRTE    | Power-up Timer Enable         | 1 = **PWRT** disabled  
|          |                               | 0 = **PWRT** enabled |
| MCLRE    | MCLR (master clear) Pin Function | 1 = Pin function = **MCLR**  
|          |                               | 0 = Pin function = digital I/O |
# PIC Configuration Bits — 2

| WDTE | Watchdog Timer Enable | 1 = WDT enabled  
0 = WDT disabled |
|------|-----------------------|----------------|
| **FOSC1 : FOSC0** | Oscillator Selection  
For devices with no internal RC | 11 = RC oscillator  
10 = HS oscillator  
01 = XT oscillator  
00 = LP oscillator |
| **FOSC2 : FOSC0** | Oscillator Selection  
For devices with internal RC | 111 = EXTRC oscillator, with CLKOUT  
110 = EXTRC oscillator  
101 = INTRC oscillator, with CLKOUT  
100 = INTRC oscillator  
011 = Reserved  
010 = HS oscillator  
001 = XT oscillator  
000 = LP oscillator |