Microprocessors and microcontrollers

PIC and ARM
Automotive Applications

- Automotive industry a big driver for high performance embedded microcontrollers
## Worldwide 8-bit Microcontroller Market Share - Units

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Motorola</td>
<td>Motorola</td>
<td>Motorola</td>
<td>Motorola</td>
<td>Motorola</td>
<td>Motorola</td>
<td>Motorola</td>
</tr>
<tr>
<td>2</td>
<td>Mitsubishi</td>
<td>Mitsubishi</td>
<td>Mitsubishi</td>
<td>Mitsubishi</td>
<td>Mitsubishi</td>
<td>Mitsubishi</td>
<td>Mitsubishi</td>
</tr>
<tr>
<td>3</td>
<td>NEC</td>
<td>NEC</td>
<td>NEC</td>
<td>NEC</td>
<td>NEC</td>
<td>NEC</td>
<td>NEC</td>
</tr>
<tr>
<td>4</td>
<td>Intel</td>
<td>Intel</td>
<td>Intel</td>
<td>Intel</td>
<td>Intel</td>
<td>Intel</td>
<td>Intel</td>
</tr>
<tr>
<td>5</td>
<td>Hitachi</td>
<td>Hitachi</td>
<td>Hitachi</td>
<td>Hitachi</td>
<td>Hitachi</td>
<td>Hitachi</td>
<td>Hitachi</td>
</tr>
<tr>
<td>6</td>
<td>Philips</td>
<td>Philips</td>
<td>Philips</td>
<td>Philips</td>
<td>Philips</td>
<td>Philips</td>
<td>Philips</td>
</tr>
<tr>
<td>7</td>
<td>Matsushita</td>
<td>Matsushita</td>
<td>Matsushita</td>
<td>Matsushita</td>
<td>Matsushita</td>
<td>Matsushita</td>
<td>Matsushita</td>
</tr>
<tr>
<td>8</td>
<td>National</td>
<td>SGS-Thomson</td>
<td>SGS-Thomson</td>
<td>SGS-Thomson</td>
<td>SGS-Thomson</td>
<td>SGS-Thomson</td>
<td>SGS-Thomson</td>
</tr>
<tr>
<td>9</td>
<td>Siemens</td>
<td>Siemens</td>
<td>Siemens</td>
<td>National</td>
<td>National</td>
<td>National</td>
<td>National</td>
</tr>
<tr>
<td>10</td>
<td>TI</td>
<td>TI</td>
<td>TI</td>
<td>TI</td>
<td>TI</td>
<td>TI</td>
<td>TI</td>
</tr>
<tr>
<td>11</td>
<td>Sharp</td>
<td>National</td>
<td>Zilog</td>
<td>National</td>
<td>Toshiba</td>
<td>Toshiba</td>
<td>Toshiba</td>
</tr>
<tr>
<td>12</td>
<td>Oki</td>
<td>Toshiba</td>
<td>Toshiba</td>
<td>Zilog</td>
<td>National</td>
<td>National</td>
<td>National</td>
</tr>
<tr>
<td>13</td>
<td>Toshiba</td>
<td>Sony</td>
<td>Siemens</td>
<td>Siemens</td>
<td>Sanyo</td>
<td>Sanyo</td>
<td>Sanyo</td>
</tr>
<tr>
<td>14</td>
<td>SGS-Thomson</td>
<td>Sharp</td>
<td>Sanyo</td>
<td>Sharp</td>
<td>Sanyo</td>
<td>Sanyo</td>
<td>Sanyo</td>
</tr>
<tr>
<td>15</td>
<td>Zilog</td>
<td>OKI</td>
<td>Matra MHS</td>
<td>Matra MHS</td>
<td>Sony</td>
<td>Oki</td>
<td>Oki</td>
</tr>
<tr>
<td>16</td>
<td>Matra MHS</td>
<td>Zilog</td>
<td>Sony</td>
<td>Matra MHS</td>
<td>Sanyo</td>
<td>Sanyo</td>
<td>Sanyo</td>
</tr>
<tr>
<td>17</td>
<td>Sony</td>
<td>Microchip</td>
<td>AMD</td>
<td>Fujitsu</td>
<td>Fujitsu</td>
<td>Fujitsu</td>
<td>Fujitsu</td>
</tr>
<tr>
<td>18</td>
<td>Fujitsu</td>
<td>Matra MHS</td>
<td>Sony</td>
<td>Fujitsu</td>
<td>Sanyo</td>
<td>Oki</td>
<td>Oki</td>
</tr>
<tr>
<td>19</td>
<td>AMD</td>
<td>Fujitsu</td>
<td>Oki</td>
<td>Fujitsu</td>
<td>Sony</td>
<td>Sony</td>
<td>Sony</td>
</tr>
<tr>
<td>20</td>
<td>Microchip</td>
<td>Sanyo</td>
<td>AMD</td>
<td>Temic</td>
<td>Sharp</td>
<td>Sharp</td>
<td>Sharp</td>
</tr>
</tbody>
</table>

Based on unit shipment volume 1990-2000, Source: Dataquest, July 2001
PIC16F84 (old but venerable)
Misc PIC16F84 Functions

• OSC Selection

• RESETs - Power-on Reset (POR), Power-up Timer (PWRT), Oscillator Start-up Timer (OST)

• Interrupts

• Watchdog Timer (WDT)

• SLEEP

• Code Protection

• ID Locations
PIC16F84 PINS
PIC16F84 PINS

Pin 1 RA2 Second pin on portA. Has no additional function
Pin 2 RA3 Third pin on portA. Has no additional function.
Pin 3 RA4 Fourth pin on portA. TOCK1 timer is also on this pin
Pin 4 MCLR Reset input and Vpp programming voltage of a microcontroller
Pin 6 RB0 Zero pin on portB. Interrupt input is an additional function.
Pin 7 RB1 First pin on portB.
Pin 8 RB2 Second pin on portB.
Pin 9 RB3 Third pin on portB.
Pin 10 RB4 Fourth pin on portB.
Pin 11 RB5 Fifth pin on portB.
Pin 12 RB6 Sixth pin on portB. 'Clock' line in program mode.
Pin 13 RB7 Seventh pin on portB. 'Given' line in program mode.
Pin 15 OSC2 Pin assigned for connecting with an oscillator
Pin 16 OSC1 Pin assigned for connecting with an oscillator
Pin 17 RA2 Second pin on portA.
Pin 18 RA1 First pin on portA.
Hig Performance RISC CPU

• Only 35 single word instructions to learn
• All instructions single-cycle except for program branches which are two-cycle
• Operating speed: DC - 20 MHz clock input  DC - 200 ns instruction cycle
• 1024 words of program memory
• 68 bytes of Data RAM
• 64 bytes of Data EEPROM
• 14-bit wide instruction words
• 8-bit wide data bytes
• 15 Special Function Hardware registers
• Eight-level deep hardware stack
• Direct, indirect and relative addressing modes
• Four interrupt sources:
  - External RB0/INT pin
  - TMR0 timer overflow
  - PORTB<7:4> interrupt-on-change
  - Data EEPROM write complete
PIC16F84 Instruction set

• For byte-oriented instructions, 'f' represents a file register designator and 'd' represents a destination designator. The file register designator specifies which file register is to be used by the instruction.
• The destination designator specifies where the result of the operation is to be placed. If 'd' is zero, the result is placed in the W register. If 'd' is one, the result is placed in the file register specified in the instruction.
• For bit-oriented instructions, 'b' represents a bit field designator which selects the number of the bit affected by the operation, while 'f' represents the address of the file in which the bit is located.
• For literal and control operations, 'k' represents an eight or eleven bit constant or literal value.
# Opcodes

### TABLE 7-1: OPCODE FIELD DESCRIPTIONS

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>Register file address (0x00 to 0x7F)</td>
</tr>
<tr>
<td>W</td>
<td>Working register (accumulator)</td>
</tr>
<tr>
<td>b</td>
<td>Bit address within an 8-bit file register</td>
</tr>
<tr>
<td>k</td>
<td>Literal field, constant data or label</td>
</tr>
</tbody>
</table>
| x     | Don’t care location (= 0 or 1)  
The assembler will generate code with x = 0.  
It is the recommended form of use for compatibility with all Microchip software tools. |
| d     | Destination select; d = 0: store result in W,  
d = 1: store result in file register f.  
Default is d = 1 |

PC | Program Counter  
TO | Time-out bit  
PD | Power-down bit

The instruction set is highly orthogonal and is grouped into three basic categories:

- **Byte-oriented** operations
- **Bit-oriented** operations
- **Literal and control** operations

### FIGURE 7-1: GENERAL FORMAT FOR INSTRUCTIONS

**Byte-oriented file register operations**  

<table>
<thead>
<tr>
<th>13</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPCODE</td>
<td>d</td>
<td>f (FILE #)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- d = 0 for destination W  
- d = 1 for destination f  
- f = 7-bit file register address

**Bit-oriented file register operations**  

<table>
<thead>
<tr>
<th>13</th>
<th>10</th>
<th>9</th>
<th>7</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPCODE</td>
<td>b (BIT #)</td>
<td>f (FILE #)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- b = 3-bit bit address  
- f = 7-bit file register address

**Literal and control operations**

**General**  

<table>
<thead>
<tr>
<th>13</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPCODE</td>
<td>k (literal)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- k = 8-bit immediate value

**CALL and GOTO instructions only**  

<table>
<thead>
<tr>
<th>13</th>
<th>11</th>
<th>10</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPCODE</td>
<td>k (literal)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- k = 11-bit immediate value

A description of each instruction is available in the PICmicro™ Mid-Range Reference Manual (DS33023).
# Instruction Set

## Table 7-2: PIC16CXXX Instruction Set

<table>
<thead>
<tr>
<th>Mnemonic, Operands</th>
<th>Description</th>
<th>Cycles</th>
<th>14-Bit Opcode</th>
<th>Status Affected</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDWF f, d</td>
<td>Add W and f</td>
<td>1</td>
<td>00 0111 dffe fffe</td>
<td>C, DC, Z</td>
<td>1, 2</td>
</tr>
<tr>
<td>ANDWF f, d</td>
<td>AND W with f</td>
<td>1</td>
<td>00 0101 dffe fffe</td>
<td>Z</td>
<td>1, 2</td>
</tr>
<tr>
<td>CLRF f</td>
<td>Clear f</td>
<td>1</td>
<td>00 0001 1ffe fffe</td>
<td>Z</td>
<td>2</td>
</tr>
<tr>
<td>CLRWF</td>
<td>Clear W</td>
<td>1</td>
<td>00 0001 0ffe fffe</td>
<td>Z</td>
<td>2</td>
</tr>
<tr>
<td>COMF f, d</td>
<td>Complement f</td>
<td>1</td>
<td>00 1001 dffe fffe</td>
<td>Z</td>
<td>1, 2</td>
</tr>
<tr>
<td>DECF f, d</td>
<td>Decrement f</td>
<td>1</td>
<td>00 0011 dffe fffe</td>
<td>Z</td>
<td>1, 2</td>
</tr>
<tr>
<td>DECFSZ f, d</td>
<td>Decrement f, Skip if 0</td>
<td>1 (2)</td>
<td>00 1011 dffe fffe</td>
<td></td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>INCF f, d</td>
<td>Increment f</td>
<td>1</td>
<td>00 1010 dffe fffe</td>
<td>Z</td>
<td>1, 2</td>
</tr>
<tr>
<td>INCFSZ f, d</td>
<td>Increment f, Skip if 0</td>
<td>1 (2)</td>
<td>00 1111 dffe fffe</td>
<td>Z</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>IORWF f, d</td>
<td>Inclusive OR W with f</td>
<td>1</td>
<td>00 0100 dffe fffe</td>
<td>Z</td>
<td>1, 2</td>
</tr>
<tr>
<td>MOVWF f, d</td>
<td>Move f</td>
<td>1</td>
<td>00 1000 dffe fffe</td>
<td>Z</td>
<td>1, 2</td>
</tr>
<tr>
<td>MOVWF f</td>
<td>Move W to f</td>
<td>1</td>
<td>00 0000 1ffe fffe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td>No Operation</td>
<td>1</td>
<td>00 0000 0ffe fffe</td>
<td>C</td>
<td>1, 2</td>
</tr>
<tr>
<td>RLF f, d</td>
<td>Rotate Left f through Carry</td>
<td>1</td>
<td>00 1101 dffe fffe</td>
<td>C</td>
<td>1, 2</td>
</tr>
<tr>
<td>RRF f, d</td>
<td>Rotate Right f through Carry</td>
<td>1</td>
<td>00 1100 dffe fffe</td>
<td>C</td>
<td>1, 2</td>
</tr>
<tr>
<td>SUBWF f, d</td>
<td>Subtract W from f</td>
<td>1</td>
<td>00 0010 dffe fffe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWAPF f, d</td>
<td>Swap nibbles in f</td>
<td>1</td>
<td>00 1110 dffe fffe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XORWF f, d</td>
<td>Exclusive OR W with f</td>
<td>1</td>
<td>00 0110 dffe fffe</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Instruction set

### BIT-ORIENTED FILE REGISTER OPERATIONS

<table>
<thead>
<tr>
<th>Instruction</th>
<th>f, b</th>
<th>Description</th>
<th>Mode</th>
<th>Opcode</th>
<th>RDDDD</th>
<th>EEEEEE</th>
<th>CCCCCC</th>
<th>CCCCCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCF</td>
<td></td>
<td>Bit Clear f</td>
<td>1</td>
<td>01</td>
<td>00bb</td>
<td>bfff</td>
<td>ffff</td>
<td>1,2</td>
</tr>
<tr>
<td>BSF</td>
<td></td>
<td>Bit Set f</td>
<td>1</td>
<td>01</td>
<td>01bb</td>
<td>bfff</td>
<td>ffff</td>
<td>1,2</td>
</tr>
<tr>
<td>BTFSC</td>
<td></td>
<td>Bit Test f, Skip if Clear</td>
<td>1 (2)</td>
<td>01</td>
<td>10bb</td>
<td>bfff</td>
<td>ffff</td>
<td>3</td>
</tr>
<tr>
<td>BTFSS</td>
<td></td>
<td>Bit Test f, Skip if Set</td>
<td>1 (2)</td>
<td>01</td>
<td>11bb</td>
<td>bfff</td>
<td>ffff</td>
<td>3</td>
</tr>
</tbody>
</table>

### LITERAL AND CONTROL OPERATIONS

<table>
<thead>
<tr>
<th>Instruction</th>
<th>k</th>
<th>Description</th>
<th>Mode</th>
<th>Opcode</th>
<th>RDDDD</th>
<th>EEEEEE</th>
<th>CCCCCC</th>
<th>CCCCCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDLW</td>
<td>k</td>
<td>Add literal and W</td>
<td>1</td>
<td>11</td>
<td>111x</td>
<td>kkkk</td>
<td>kkkk</td>
<td>C,DC,Z</td>
</tr>
<tr>
<td>ANDLW</td>
<td>k</td>
<td>AND literal with W</td>
<td>1</td>
<td>11</td>
<td>1001</td>
<td>kkkk</td>
<td>kkkk</td>
<td>Z</td>
</tr>
<tr>
<td>CALL</td>
<td>k</td>
<td>Call subroutine</td>
<td>2</td>
<td>10</td>
<td>0kkk</td>
<td>kkkk</td>
<td>kkkk</td>
<td></td>
</tr>
<tr>
<td>CLRWDT</td>
<td></td>
<td>Clear Watchdog Timer</td>
<td>1</td>
<td>00</td>
<td>0000</td>
<td>0100</td>
<td>0100</td>
<td>Z</td>
</tr>
<tr>
<td>GOTO</td>
<td>k</td>
<td>Go to address</td>
<td>2</td>
<td>10</td>
<td>1kkk</td>
<td>kkkk</td>
<td>kkkk</td>
<td></td>
</tr>
<tr>
<td>IORLW</td>
<td>k</td>
<td>Inclusive OR literal with W</td>
<td>1</td>
<td>11</td>
<td>1000</td>
<td>kkkk</td>
<td>kkkk</td>
<td></td>
</tr>
<tr>
<td>MOVLW</td>
<td>k</td>
<td>Move literal to W</td>
<td>1</td>
<td>11</td>
<td>00xx</td>
<td>kkkk</td>
<td>kkkk</td>
<td></td>
</tr>
<tr>
<td>RETFIE</td>
<td></td>
<td>Return from interrupt</td>
<td>2</td>
<td>00</td>
<td>0000</td>
<td>0000</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>RETLW</td>
<td>k</td>
<td>Return with literal in W</td>
<td>2</td>
<td>11</td>
<td>01xx</td>
<td>kkkk</td>
<td>kkkk</td>
<td></td>
</tr>
<tr>
<td>RETURN</td>
<td></td>
<td>Return from Subroutine</td>
<td>2</td>
<td>00</td>
<td>0000</td>
<td>0000</td>
<td>1000</td>
<td>TO,PD</td>
</tr>
<tr>
<td>SLEEP</td>
<td></td>
<td>Go into standby mode</td>
<td>1</td>
<td>00</td>
<td>0000</td>
<td>0110</td>
<td>0011</td>
<td></td>
</tr>
<tr>
<td>SUBLW</td>
<td>k</td>
<td>Subtract W from literal</td>
<td>1</td>
<td>11</td>
<td>110x</td>
<td>kkkk</td>
<td>kkkk</td>
<td>TO,PD</td>
</tr>
<tr>
<td>XORLW</td>
<td>k</td>
<td>Exclusive OR literal with W</td>
<td>1</td>
<td>11</td>
<td>1010</td>
<td>kkkk</td>
<td>kkkk</td>
<td>C,DC,Z</td>
</tr>
</tbody>
</table>

### 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’.

### Note 2
If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be assigned to the Timer0 Module.

### Note 3
If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The execution is executed as a NOP.
<table>
<thead>
<tr>
<th><strong>ADDLW</strong></th>
<th><strong>Add Literal and W</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>[label] ADDLW k</td>
</tr>
<tr>
<td>Operands:</td>
<td>$0 \leq k \leq 255$</td>
</tr>
<tr>
<td>Operation:</td>
<td>$(W) + k \rightarrow (W)$</td>
</tr>
<tr>
<td>Status Affected:</td>
<td>C, DC, Z</td>
</tr>
<tr>
<td>Description:</td>
<td>The contents of the W register are added to the eight-bit literal 'k' and the result is placed in the W register.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>BCF</strong></th>
<th><strong>Bit Clear f</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>[label] BCF f,b</td>
</tr>
<tr>
<td>Operands:</td>
<td>$0 \leq f \leq 127$</td>
</tr>
<tr>
<td>Operation:</td>
<td>$0 \rightarrow (f\langle b\rangle)$</td>
</tr>
<tr>
<td>Status Affected:</td>
<td>None</td>
</tr>
<tr>
<td>Description:</td>
<td>Bit 'b' in register 'f' is cleared.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>ADDFW</strong></th>
<th><strong>Add W and f</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>[label] ADDWF f,d</td>
</tr>
<tr>
<td>Operands:</td>
<td>$0 \leq f \leq 127$</td>
</tr>
<tr>
<td></td>
<td>$d \in [0,1]$</td>
</tr>
<tr>
<td>Operation:</td>
<td>$(W) + (f) \rightarrow (\text{destination})$</td>
</tr>
<tr>
<td>Status Affected:</td>
<td>C, DC, Z</td>
</tr>
<tr>
<td>Description:</td>
<td>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'.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>BSF</strong></th>
<th><strong>Bit Set f</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>[label] BSF f,b</td>
</tr>
<tr>
<td>Operands:</td>
<td>$0 \leq f \leq 127$</td>
</tr>
<tr>
<td>Operation:</td>
<td>$0 \leq b \leq 7$</td>
</tr>
<tr>
<td>Operation:</td>
<td>$1 \rightarrow (f\langle b\rangle)$</td>
</tr>
<tr>
<td>Status Affected:</td>
<td>None</td>
</tr>
<tr>
<td>Description:</td>
<td>Bit 'b' in register 'f' is set.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>ANDLW</strong></th>
<th><strong>AND Literal with W</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>[label] ANDLW k</td>
</tr>
<tr>
<td>Operands:</td>
<td>$0 \leq k \leq 255$</td>
</tr>
<tr>
<td>Operation:</td>
<td>$(W) \cdot \text{AND} (k) \rightarrow (W)$</td>
</tr>
<tr>
<td>Status Affected:</td>
<td>Z</td>
</tr>
<tr>
<td>Description:</td>
<td>The contents of W register are AND'ed with the eight-bit literal 'k'. The result is placed in the W</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>BTFSS</strong></th>
<th><strong>Bit Test f, Skip if Set</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>[label] BTFSS f,b</td>
</tr>
<tr>
<td>Operands:</td>
<td>$0 \leq f \leq 127$</td>
</tr>
<tr>
<td></td>
<td>$0 \leq b &lt; 7$</td>
</tr>
<tr>
<td>Operation:</td>
<td>skip if $(f\langle b\rangle) = 1$</td>
</tr>
<tr>
<td>Status Affected:</td>
<td>None</td>
</tr>
<tr>
<td>Description:</td>
<td>If bit 'b' in register 'f' is '0', the next instruction is executed.</td>
</tr>
</tbody>
</table>
Simple Assembly Language

Let us look at some examples of PIC16F84 microcontroller assembler instructions. What do these instructions do?

At the start, the working register (W) contains the value 0x1C and the PORTB register contains the value 0. What values are in W and PORTB after each instruction?

```
MOVLW 0x24
ADDLW 0x02
ADDLW 0x17
MOVWF PORTB
CLRW
ADDLW 0x05
MOVWF PORTB
CLRF PORTB
```
Simple Assembly Language

; WRITTEN BY SIW
; DATE 01/01/2004
; FILE SAVED AS TEST.ASM
; DEVICE 16F84
; OSCILLATOR XT (4MHZ)
; WATCHDOG DISABLED
; FUNCTION OUTPUTS THE VALUE 0XF1 TO 8 LEDS CONNECTED TO PORTB

; ---------------------------------- EQUATES ----------------------------------
PORTB EQU 0X06 ;ASSIGN THE PORTB REGISTER TO THE LABEL 'PORTB'

; -------------------------------- MAIN PROGRAM --------------------------------
START ORG 0X00 ;'ORG' SPECIFIES THE MEMORY LOCATION OF THE PROGRAM
    MOVLW 0X00 ;MOVE THE VALUE 00, I.E., ALL 0'S TO W
    TRIS PORTB ;CONFIGURE PORTB WITH THE VALUE IN W (THE
    ;WORKING REGISTER) 1=INPUT AND 0=OUTPUT.
    CLRF PORTB ;CLEAR THE PORTB REGISTER
    MOVLW 0XF1 ;MOVE THE HEX VALUE F1 TO THE WORKING REGISTER
    MOVWF PORTB ;OUTPUT THE VALUE TO PORTB
LOOP GOTO LOOP
END

3213: Digital Systems & Microprocessors: L#20
(Source: Dr Sandra I. Woolley. Uni Birmingham, UK)
How to Assemble PIC Assembly Language

- WINDOWS: MPLAB from MicroChip (free download from http://www.microchip.com)

- LINUX: use GPASM part of GPUTILS package

Neither provide a C compiler
Example: Write to PORTB

;--Program "Port.asm". A simple program to write a byte to PORTB.
;--Author: Montague Paravrov-Nikots.
;--Date: 7 July 1997.
;--Version: 1.01
    LIST    p=PIC16C84, r=hex, f=INHX8M
    include "P16C84.inc"
    org 0x0           ; Reset vector.
;--Set up the pins of PORTB to act as outputs.
;--Select Bank 1 to access TRISB register.
    bsf STATUS,RP0    ; Select Bank 1.
    clrf TRISB       ; Make all PORTB pins outputs.
;--Switch back to Bank 0 to access PORTB register.
    bcf STATUS,RP0    ; Back to Bank 0.
;--Load a byte into PORTB.
    movlw B'10110101' ; Move a byte into the working register.
    movwf PORTB      ; Move contents of W into PORTB register.
    wait             ; Loop forever until power is removed.
    goto wait
    end              ; End of program "Port.asm".
P16C84.inc: Header file

; This header file defines configurations, registers, and other useful bits of
; information for the PIC16C84 microcontroller. These names are taken to match \
; the data sheets as closely as possible.

; Note that the processor must be selected before this file is included. The processor may be selected the following ways:

; 1. Command line switch:
;       C:\ MPASM MYFILE.ASM /PIC16C84
; 2. LIST directive in the source file
;       LIST   P=PIC16C84
; 3. Processor Type entry in the MPASM full-screen interface
;--- Register Files-----------------------------------------------

INDF EQU H'0000'
TMR0 EQU H'0001'
PCL EQU H'0002'
STATUS EQU H'0003'
FSR EQU H'0004'
PORTA EQU H'0005'
PORTB EQU H'0006'
EEDATA EQU H'0008'
EEADR EQU H'0009'
PCLATH EQU H'000A'
INTCON EQU H'000B'

OPTION_REG EQU H'0081'
TRISA EQU H'0085'
TRISB EQU H'0086'
EECON1 EQU H'0088'
EECON2 EQU H'0089'
GPASM (the assembly language compiler)

```assembly
bsf STATUS, RP0
clrf TRISB
bcf STATUS, RP0
movlw B'10110101'
movwf PORTB
wait    goto wait
end
```

Checksum: B'10110101'

TRISB

No need to encode 'wait'

3213: Digital Systems & Microprocessors: L#21
The C Programming Language

• C is a high level language... Looks more like VERILOG than assembly language. Dennis Ritchie (Circa 1973)

• LINUX and WINDOWS are written in C

• Can use C to program PICs and to written complex software involving Graphical User Interfaces (GUIs)

• C++ = Object oriented C.
The C Versions

• C has an ANSI standard form “ANSI Standard C”.

• Particular versions of C for different applications: Microsoft C, GNU C for Intel, uCLINUX and SDCC for embedded devices.

• Can use C to program PICs as well as to write complex software involving Graphical User Interfaces (GUIs)... just depends on the compiler.
main module and printf()

```c
#include <stdio.h>

int main() {
    printf("Hello World!!\n");
    return 1;
}
```

preprocessor directive

main module

printf.. the most useful debugging tool
C modules, type, control loops, return values

```c
int b2b( int btst[8] ) {  
    int theByte = 0;
    int index = 1;
    int i;

    for(i=0;i < 8; i++) {
        theByte = theByte + index*btst[i];
        index = 2*index;
    }
    return theByte;
}

int main() {  
    int bts[8], i;
    int TheByte;

    for(i=7;i>=0;i--) {  
        bts[i] = i%2;
        printf("%d",bts[i]);
    }
    printf("\n");
    TheByte = b2b(bts);
    printf("TheByte = %d\n",TheByte);
    return 1;
}
```

TheByte is returned. But mostly you return the status of the call.
More data types

- Strings are pointers to chars
- Floating point and double precision numbers
- Short 16 bit
- Int, long 32 bit
- Char = 8 bit (the byte!)

```c
int main()
{
    char c = 'A';  // A character
    char * name = "ALFRED";  // A string

    float f = 32.33;
    double g = 32.33;
    double d = 1.2e-13;

    short sshort = 32767;
    int sint = 32767;
    long slong = 32767;

    unsigned short usshort = 32767;
    unsigned int usint = 32767;
    unsigned long uslong = 32767;

    char sc = 255;
    char tc = 0x0;  // Hex
    char yc;

    unsigned char uc = 255;
    unsigned char vc = 0x0;  // Hex
    unsigned char wc;

    return 1;
}
```
#include <math.h>
#define PI 3.1416

int main() {

    int i;
    double X = 0.000;
    double Y, Z;
    double dX = .01;

    for(i=0;i<100;i++) {

        Y = sin(PI*X);
        Z = cos(PI*X);
        X = X + dX;

        printf("X = %g\t, Y = %g\t, Z = %g\n",X,Y,Z);
    }

    return 1;
}

Math header
Preprocessor directive #define

cosine, sine, log, exp etc
```c
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char **argv)
{
    char filename[10], *name;
    FILE *infile;
    char next_char;
    int flag;

    if ( argc < 2 )
    {
        strcpy(filename, "in.dat");
    }

    infile = fopen(filename, "r");
    flag = fscanf( infile, "%c", &next_char);

    while( EOF != flag )
    {
        fprintf(outfile, "%c", next_char);
        flag = fscanf(infile, "%c", &next_char);
    }
    fclose(infile);
    return 1;
}
```
Pointers

Pointers point to **memory** occupied by a variable.

Use the * operator

q and j passed by reference

k passed only by value

A pointer

Assigning a value to a pointer

The & gives the address of the variable following it

**Question:**

What happens to p, i and k?

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void squareme( int * q, int * j, int k ) {
    *q = (*q)*(*q);
    *j = (*j)*(*j);
    k = k*k;
}

int main(void)
{
    int *p = malloc(sizeof(int));
    int i, k;

    *p = 10;
    i = 10;
    k = 10;
    printf("*p = %d\n",*p);
    printf("i = %d\n",i);
    printf("k = %d\n",k);

    squareme( p, &i, k);
    printf("*p = %d\n",*p);
    printf("i = %d\n",i);
    printf("k = %d\n",k);

    return 1;
}
```
What output does this code produce?

```c
void swap(int a, int *b)
{
    int temp;
    temp = a;
    a = *b;
    *b = temp;
}

int main()
{
    int x = 2, y = 3;
    swap(x, &y);
    printf("x = %d, y = %d\n", x, y);
    return 1;
}
```
More fun with pointers: KEEP IT SIMPLE STUPID PROTOCOL

```c
static int kiss_esc(unsigned char *s, unsigned char *d, int len) {
    unsigned char *ptr = d;
    unsigned char c;
    /*
     * Send an initial END character to flush out any
     * data that may have accumulated in the receiver
     * due to line noise.
     */
    *ptr++ = END;
    while (len-- > 0) {
        switch (c = *s++) {
            case END:
                *ptr++ = ESC;
                *ptr++ = ESC_END;
                break;
            case ESC:
                *ptr++ = ESC;
                *ptr++ = ESC_ESC;
                break;
            default:
                *ptr++ = c;
                break;
        }
    }
    *ptr++ = END;
    return ptr - d;
}
```
What is SDCC?

SDCC is a retargettable, optimizing ANSI - C compiler that targets the Intel 8051, Maxim 80DS390, Zilog Z80 and the Motorola 68HC08 based MCUs.

Work is in progress on supporting the Microchip PIC16 and PIC18 series.

SDCC is Free Open Source Software, distributed under GNU General Public License (GPL)
The Compiling process with SDCC

Tool flow when using sdcc libraries
Doing it in C...

/*
   project1.c
   GGB

   sdcc -mpic14 -p16f84 project1c.c
*/

/* Define processor and include header file. */
#define __16f84
#include"pic/pic16f84.h"

// Set the __CONFIG word:
typedef unsigned int word;
word at 0x2007 __CONFIG = 0x3f72;

void main(void) {

    unsigned char PB = 0xD5;

    /* PORTB.1 is an output pin */
    TRISB = 0;
    PORTB = PB;
    while(1) { /* Loop forever */ }
}
The Compiling process with SDCC

[ggb112@localhost C]$  sdcc -mpic14 -p16f84 project1c.c
message: using default linker script
"/usr/local/share/gputils/lkr/"
[ggb112@localhost C]$  ls -l
total 36
-rw-rw-r--  1  ggb112  ggb112  3326  May 24  16:32  project1c.asm
-rw-rw-r--  1  ggb112  ggb112  389  May 24  16:19  project1c.c
-rw-rw-r--  1  ggb112  ggb112  5120  May 24  16:32  project1c.cod
-rw-rw-r--  1  ggb112  ggb112  1264  May 24  16:32  project1c.hex
-rw-rw-r--  1  ggb112  ggb112  9661  May 24  16:32  project1c.lst
-rw-rw-r--  1  ggb112  ggb112  1772  May 24  16:32  project1c.o
[ggb112@localhost C]$

sdcc -S -mpic14 -p16f84 project1c.c... just compiles only (produces the asm file)
The Compiling process with SDCC

project1c.asm contains the output of compiler and contains the assembly language version of the C code.

project1c.o contains the output of gpasm.

project1c.lst contains debugging information form the linker.

project1c.hex contains the final PIC program in a format suitable to download into your programmer.

project1c.cod contains a version of the final program with additional information that can be used with gpsim.

Now you need to use your programmer to transfer project1c.hex into your PIC
PIC16F84 CPU

GPR and SFR here
CPU and ALU

CPU's most important role is to decode machine language instructions (opcodes)

- Must be connected to all parts of the microcontroller
- Data (8 bit) and address (14 bit) busses
- PIC16F84 has 8 bit ALU and 8 bit WR.
- With two operands: one in WR and the other either GPR or SFR.
- With one operand: either in WR or GPR or SFR.
- ALU affects values of Carry (C), Digit Carry (DC), and Zero (Z) bits in STATUS register.
Memory Organisation

- PIC16F84 has two separate memory blocks, one for data and the other for program.
- EEPROM memory and GPR registers in RAM memory make up the data block.
- FLASH memory makes up the program block.
  - The size of program memory is 1024 locations with 14 bits
  - Locations zero and four are reserved for reset and interrupt vector.
- Data memory consists of EEPROM and RAM memories.
  - EEPROM has 64X8 bit locations of non-volatile memory.
  - EEPROM memory indirectly accessed thru EEADR and EEDATA registers
  - RAM data memory resides on memory map from 0x0C to 0x4F (68 locations)
- Besides linear division in SFR and GPR, memory is also divided into two banks: BANK0 and BANK1.
PIC16F84 Data Memory Organisation

- Data memory = GP registers and SFR registers in RAM and EEPROM data memory
- Has 8 bit data bus
- Both GPR and SFR are banked
- SFR are first 12 registers of each bank.
- GPR banked to give 116 bytes of static RAM data storage
- Banking requires control bits (status register) for bank selection
- Bank0 is selected by clearing RP0 (Status<5>) or Bank1 by setting that bit.
- The entire memory can be accessed either by using the absolute memory address or
- Indirectly through the File Select Register (FSR)
PIC16F84 Program Memory Organisation

- Two separate memory blocks one for data and one for programs
- Has 13 bit PC to address 14 bit memory space
- The first 1k X 14 bit flash program memory (000h – 03FFh) are implemented
- An access to program space above the max causes wraparound.
- The RESET vector is at 0000h and the interrupt vector is at 0004h.
FIGURE 2-2: REGISTER FILE MAP - PIC16F84A

File Address | Indirect addr.\(^{(1)}\) | Indirect addr.\(^{(1)}\) | File Address
---|---|---|---
00h | | | 80h
01h | TMR0 | OPTION_REG | 81h
02h | PCL | PCL | 82h
03h | STATUS | STATUS | 83h
04h | FSR | FSR | 84h
05h | PORTA | TRISA | 85h
06h | PORTB | TRISB | 86h
07h | — | — | 87h
08h | EEDATA | EECON1 | 88h
09h | EEADR | EECON2\(^{(1)}\) | 89h
0Ah | PCLATH | PCLATH | 8Ah
0Bh | INTCON | INTCON | 8Bh
0Ch | | | 8Ch

---

68 General Purpose Registers (SRAM) | Mapped (accesses) in Bank 0

4Fh | CFh
50h | D0h

7Fh | Bank 0
Bank 1 | FFh

\(^{(1)}\) Unimplemented data memory location, read as '0'.

Note 1: Not a physical register.
Memory

Stack level 1
Stack level 2
Stack level 8

PC<12:0>

Address bus

Reset address
Interrupt vector address

Program memory 1024x14

Address

0000h
0004h

Accessing these locations has the same result regardless of the bank from which we are making an access.

Unimplemented memory locations, by reading them we always get "0".

68 bajta zajedničke RAM memorije

GPR Registers

EEPROM for data 64XE
## TABLE 2-1: SPECIAL FUNCTION REGISTER FILE SUMMARY

<table>
<thead>
<tr>
<th>Addr</th>
<th>Name</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Value on Power-on RESET</th>
<th>Details on page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bank 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00h</td>
<td>INDF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>---</td>
<td>11</td>
</tr>
<tr>
<td>01h</td>
<td>TMR0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0000 0000</td>
<td>20</td>
</tr>
<tr>
<td>02h</td>
<td>PCL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0000 0000</td>
<td>11</td>
</tr>
<tr>
<td>03h</td>
<td>STATUS(2)</td>
<td>IRP</td>
<td>RP1</td>
<td>RP0</td>
<td>TO</td>
<td>PD</td>
<td>Z</td>
<td>DC</td>
<td>C</td>
<td>0001 1xxx</td>
<td>8</td>
</tr>
<tr>
<td>04h</td>
<td>FSR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0xxx 0xxx</td>
<td>11</td>
</tr>
<tr>
<td>05h</td>
<td>PORTA(4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x 0xxx</td>
<td>16</td>
</tr>
<tr>
<td>06h</td>
<td>PORTB(5)</td>
<td>RB7</td>
<td>RB6</td>
<td>RB5</td>
<td>RB4</td>
<td>RB3</td>
<td>RB2</td>
<td>RB1</td>
<td>RB0/INT</td>
<td>xxx xxx</td>
<td>18</td>
</tr>
<tr>
<td>07h</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08h</td>
<td>EEDATA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0xxx 0xxx</td>
<td>13,14</td>
</tr>
<tr>
<td>09h</td>
<td>EEADR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0xxx 0xxx</td>
<td>13,14</td>
</tr>
<tr>
<td>0Ah</td>
<td>PCLATH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>--- 0000</td>
<td>11</td>
</tr>
<tr>
<td>0Bh</td>
<td>INTCON</td>
<td>GIE</td>
<td>EEIE</td>
<td>T0IE</td>
<td>INTE</td>
<td>RBIE</td>
<td>T0IF</td>
<td>INTF</td>
<td>RBIF</td>
<td>0000 000x</td>
<td>10</td>
</tr>
<tr>
<td>Bank 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0Ch</td>
<td>INDF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>---</td>
<td>11</td>
</tr>
<tr>
<td>0Dh</td>
<td>OPTION_REG</td>
<td>RBPU</td>
<td>INTEDG</td>
<td>T0CS</td>
<td>T0SE</td>
<td>PSA</td>
<td>PS2</td>
<td>PS1</td>
<td>PS0</td>
<td>1111 1111</td>
<td>9</td>
</tr>
<tr>
<td>0Eh</td>
<td>PCL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0000 0000</td>
<td>11</td>
</tr>
<tr>
<td>0Fh</td>
<td>STATUS(2)</td>
<td>IRP</td>
<td>RP1</td>
<td>RP0</td>
<td>TO</td>
<td>PD</td>
<td>Z</td>
<td>DC</td>
<td>C</td>
<td>0001 1xxx</td>
<td>8</td>
</tr>
<tr>
<td>10h</td>
<td>FSR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0xxx 0xxx</td>
<td></td>
</tr>
<tr>
<td>11h</td>
<td>TRISA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>--- 1111</td>
<td>16</td>
</tr>
<tr>
<td>12h</td>
<td>TRISB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1111 1111</td>
<td>18</td>
</tr>
<tr>
<td>13h</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14h</td>
<td>EECON1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>--- 0 x000</td>
<td>13</td>
</tr>
<tr>
<td>15h</td>
<td>EECON2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>16h</td>
<td>PCLATH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>--- 0 000</td>
<td>11</td>
</tr>
<tr>
<td>17h</td>
<td>INTCON</td>
<td>GIE</td>
<td>EEIE</td>
<td>T0IE</td>
<td>INTE</td>
<td>RBIE</td>
<td>T0IF</td>
<td>INTF</td>
<td>RBIF</td>
<td>0000 000x</td>
<td>10</td>
</tr>
</tbody>
</table>

**Legend:**
- **x** = unknown, **\_** = unchanged.
- **\_** = unimplemented, read as '0'.
- **c** = value depends on condition

**Note 1:** The upper byte of the program counter is not directly accessible. PCLATH is a slave register for PC<12:8>. The contents of PCLATH can be transferred to the upper byte of the program counter, but the contents of PC<12:8> are never transferred to PCLATH.

**Note 2:** The TO and PD status bits in the STATUS register are not affected by a **MCLR** Reset.

**Note 3:** Other (non-power-up) Resets include: external Reset through **MCLR** and the **Watchdog Timer** Reset.

**Note 4:** On any device Reset, these pins are configured as inputs.

**Note 5:** This is the value that will be in the port output latch.
Status Register

- STATUS register contains arithmetic status ALU (C (carrier), DC (digit carry), Z (zero test)), RESET status (TO (time-out), PD (power down)) and bits for selecting of memory bank (BANK0 or BANK1) (IRP, RP0 (register bank select bits)). *N.B. RP1 not used.*

- Because selection of memory bank is controlled through this register, it has to be present in each bank.
**Status register**

**STATUS Register**

<table>
<thead>
<tr>
<th>IRP</th>
<th>RP1</th>
<th>RP0</th>
<th>TO</th>
<th>PD</th>
<th>Z</th>
<th>DC</th>
<th>C</th>
</tr>
</thead>
</table>

**Legend:**
- **R** = Readable bit
- **W** = Writable bit
- **U** = Unimplemented bit, read as '0'
- **-n** = Value at POR reset

- Bit 0: Carry C bit is affected by ADDWF, ADDLW, SUBLW, SUBWF instructions. *Set with a bit carry*
- Bit 1: Digit carry (DC) is affected by ADDWF, ADDLW, SUBLW, SUBWF instructions. *Set with a bit carry from fourth bit.*
- Bit 2: Indication of a zero result. *Set when result produces a zero*
- Bit 3: Power down bit. *1 = power up, 0 = sleep mode etc (see datasheet)*
- Bit 4: Timeout. *Set while running. Reset by watchdog timer in case of fault.*
- Bit 6:5  RP1:RP0 Register back select bits. Direct addressing
- Bit 7  IRP Register bank select for indirect addressing
Program Counter and Stack

- Program counter (PC) is a 13 bit register that contains the address of the instruction being executed.
- PIC16F84 has a 13 bit, 8 level stack.
- Main use of stack is to store the value of the program counter when a branch has occurred.
- When moving from a program to a subprogram, the current PC value is **pushed** onto the stack
- The stack value is **popped** when a return is executed.
Interrupts

The PIC16F84A has 4 sources of interrupt:

- External interrupt RB0/INT pin
- TMR0 overflow interrupt
- PORTB change interrupts (pins RB7:RB4)
- Data EEPROM write complete interrupt

The interrupt control register (INTCON) records individual interrupt requests in flag bits. It also contains the individual and global interrupt enable bits. Interrupts recorded on the stack like subroutine branches.
REGISTER 2-3: INTCON REGISTER (ADDRESS 0Bh, 8Bh)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value at POR</th>
<th>Legend</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit 0</td>
<td>RBIF: RB Port Change Interrupt Flag bit</td>
<td>0</td>
<td>R = Readable bit, W = Writable bit, U = Unimplemented bit, read as ‘0’, ‘1’ = Bit is set, ‘0’ = Bit is cleared, x = Bit is unknown</td>
</tr>
<tr>
<td>bit 1</td>
<td>INTF: RB0/INT External Interrupt Flag bit</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>bit 2</td>
<td>TOIF: TMR0 Overflow Interrupt Flag bit</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>bit 3</td>
<td>RBIE: RB Port Change Interrupt Enable bit</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>bit 4</td>
<td>INTE: RB0/INT External Interrupt Enable bit</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>bit 5</td>
<td>TOIE: TMR0 Overflow Interrupt Enable bit</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>bit 6</td>
<td>EEIE: EE Write Complete Interrupt Enable bit</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>bit 7</td>
<td>GIE: Global Interrupt Enable bit</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

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

- Physical connection to outside world
- A port is a register inside the microcontroller
- Some pins have two functions, some pins have twofold roles like PA4/TOCKI
- Selection done in one of the configurational registers
- An illustration of this is the fifth bit T0CS in OPTION register
Ports: PORTA and PORTB

- All pins can be input or output
- Need to write a combination of 0's (output) and 1's (input) to TRISA/TRISB registers
- Every port has its proper TRIS register. Thus, port A has TRISA at address 85h, and port B has TRISB at address 86h.
Free Timer TRM0

- Counts 0:255 repeatedly
- prescaler divides the timer clock as shown
- Can generate interrupts to program execution
Free Timer TRM0

• Programmer must reset TOIF so that a new interrupt can be detected.

• Interrupts can also be generated by an external clock on RA4/TOCK1

• Choosing one of TRM0 and RA4/TOCK1 is chosen by the TOCS in the OPTION register.

• If the external clock is chosen then you can define the edge on which the interrupt is generated.
Clock and Instruction Cycle

- Clock from the oscillator enters microcontroller via OSC1 pin where internal circuit of a microcontroller divides the clock into four even clocks Q1, Q2, Q3, and Q4 which do not overlap.
- These four clocks make up one instruction cycle (also called machine cycle) during which one instruction is executed.
Instruction cycle consists of cycles Q1, Q2, Q3 and Q4.

Cycle of calling begins with Q1 clock, by writing into instruction register (IR). Decoding and executing begins with Q2, Q3 and Q4 clocks.

To call and execute an instruction, one instruction cycle is needed, and one more is needed for decoding and execution.

With pipelining each instruction is effectively executed in one cycle.
Pipelining normal case without branch instruction

• TCY0 reads in instruction MOVLW 55h.
• TCY1 executes instruction MOVLW 55h and reads in MOVWF PORTB.
• TCY2 executes MOVWF PORTB and reads in CALL SUB_1.
• TCY3 executes a call of a subprogram CALL SUB_1, and reads in instruction BSF PORTA, BIT3.
  • As this instruction is not the one we need, or is not the first instruction of a subprogram SUB_1 whose execution is next in order, instruction must be flushed and read in again.
  • This is a example of an instruction needing more than one cycle.
• TCY4 instruction cycle is totally used up for reading in the first instruction from a subprogram at address SUB_1.
• TCY5 executes the first instruction from a subprogram SUB_1 and reads in the next one.
Pipelining with branch

All instructions are single cycle, except for any program branches. These take two cycles since the fetch instructions is "flushed" from the pipeline while the new instruction is being fetched and then executed.

If instruction causes a change in PC, and PC doesn't point to the following but to some other address (which can be the case with jumps or with calling subprograms), two cycles are needed for executing an instruction.
Misc PIC16F84 Functions

• OSC Selection

• RESETs - Power-on Reset (POR), Power-up Timer (PWRT), Oscillator Start-up Timer (OST)

• Interrupts

• Watchdog Timer (WDT)

• SLEEP

• Code Protection

• ID Locations
In System Programming

• In order to program the program memory, the microcontroller must be set to special working regime by bringing up MCLR pin to 13.5V, and supply voltage Vdd has to be stabilized between 4.5V to 5.5V.

• Program memory can be programmed serially using two 'data/clock' pins which must previously be separated from device lines, so that errors do not arise during programming.

• In circuit serial programming (ICSP) is used to program PICs.

There is free software to do this (MPLAB, ICPROG EXE...
FIGURE 7: TYPICAL CIRCUIT INCORPORATING THE ICSP PROTOCOL INTO AN APPLICATION

Note 1: Component values are recommended starting points for design. Final values must be validated in the actual application. Some applications may not require the use of the MCLR series resistor.

2: Because of multiple considerations, no other application components are shown connected to these lines. Actual designs may require passive components. See text for recommendations.

3: PGM is only used in devices capable of single-supply ICSP operation. OTP and some Flash devices use a 5-wire connection.
Linux usage on Intel (4 level Von Neuman)

Figure 1-1. A split view of the kernel
ARM Architecture

Advanced RISC Machines (1990)
(ACORN and Apple Computer)
ARM history

• 1983 developed by Acorn computers
  – To replace 6502 in BBC computers
  – 4-man VLSI design team
  – Its simplicity comes from the inexperienced team
  – Match the needs for generalized SoC for reasonable power, performance and die size

• 1990 ARM (Advanced RISC Machine), owned by Acorn, Apple and LSI
ARM Ltd

Design and license ARM core design but not fabricate
Why ARM?

- One of the most licensed and thus widespread processor cores in the world
  - Used in PDA, cell phones, multimedia players, handheld game console, digital TV and cameras
  - ARM7: GBA, iPod
  - ARM9: NDS, PSP, Sony Ericsson, BenQ
  - ARM11: Apple iPhone, Nokia N93, N800
  - 75% of 32-bit embedded processors

- Used especially in portable devices due to its low power consumption and reasonable performance
ARM powered products
Naming ARM

- ARMxyzTDMIEJFS
  - x: series
  - y: MMU
  - z: cache
  - T: Thumb
  - D: debugger
  - M: Multiplier
  - I: Interrupt
  - E: Enhanced
  - J: Jazelle
  - F: Floating-point
Popular ARM architecture

• ARM7TDMI
  – 3 pipeline stages
  – One of the most used ARM-version (for low-end systems)

• ARM9TDMI
  – Compatible with ARM7
  – 5 pipeline stages
  – Separate instruction and data cache

• ARM11
ARM architecture

- Load/store architecture
- A large array of uniform registers
- Fixed-length 32-bit instructions
- 3-address instructions
# Processor modes

<table>
<thead>
<tr>
<th>Processor mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>usr</td>
</tr>
<tr>
<td>FIQ</td>
<td>fiq</td>
</tr>
<tr>
<td>IRQ</td>
<td>irq</td>
</tr>
<tr>
<td>Supervisor</td>
<td>svc</td>
</tr>
<tr>
<td>Abort</td>
<td>abt</td>
</tr>
<tr>
<td>Undefined</td>
<td>und</td>
</tr>
<tr>
<td>System</td>
<td>sys</td>
</tr>
</tbody>
</table>
ARM architecture

- 37 registers
  - 1 Program counter
  - 1 current program status registers
  - 5 saved program status registers
  - 30 general purpose registers
Registers

• Only 16 registers are visible to a specific mode.

  A mode could access
  – A particular set of r0-r12
  – r13 (sp, stack pointer)
  – r14 (lr, link register)
  – r15 (pc, program counter)
  – Current program status register (cpsr)
General-purpose registers

- 6 data types (signed/unsigned)
- All ARM operations are 32-bit. Shorter data types are only supported by data transfer operations.
Program counter

• Store the address of the instruction to be executed
• All instructions are 32-bit wide and word-aligned
• Thus, the last two bits of pc are undefined.
Program status register (CPSR)

- N: Negative
- Z: Zero
- C: Carry/Borrow
- V: Overflow
- Q: Sign
- I: FIQ disable
- F: IRQ disable
- T: Mode bits
- M0-M3: Mode bits

3213: Digital Systems & Microprocessors: L#22_23
Summary

- Load/store architecture
- Most instructions are RISCy, operate in single cycle.
  - Some multi-register operations take longer.
- All instructions can be executed conditionally.
Welcome to Futurlec. The ELECTRONIC COMPONENTS Superstore. To find the component you’re looking for, either search by part number or visit the relative department. Need Help.

CONTROLLERS

ARM2103 Controller - Technical Data

- Microcontroller: LPC2103
- Main Crystal: 19.66MHz
- Speed: Up to 59.9842MHz
- Processor Language: ARM

Memory

- Program Flash Memory (Internal): 32kBytes
- RAM Memory - Scratchpad (Internal): 8 kBytes
- EEPROM Memory (Internal): None

Input/Output

- I/O Points Available: 32
- I/O Points Connection: IDC Connector