Microcontroller (μC) vs. Microprocessor (μP)

• μC intended as a single chip solution, μP requires external support chips (memory, interface)
• μC has on-chip non-volatile memory for program storage, μP does not.
• μC has more interface functions on-chip (serial interfaces, analog-to-digital conversion, timers, etc.) than μP
• μC does not have virtual memory support (i.e., could not run Linux), while μP does.
• General purpose μPs are typically higher performance (clock speed, data width, instruction set, cache) than μCs
• Division between μPs and μCs becoming increasingly blurred
### PIC24 Family µC

<table>
<thead>
<tr>
<th>Features</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction width</td>
<td>24 bits</td>
</tr>
<tr>
<td>On-chip program memory (non-volatile, electrically erasable)</td>
<td>PIC24HJ32GP202 has 32K bytes/11264 instructions, architecture supports 24Mbytes/4M instructions</td>
</tr>
<tr>
<td>On-chip Random Access Memory (RAM)</td>
<td>PIC24HJ32GP202 has 2048 bytes, architecture supports up 65536 bytes</td>
</tr>
<tr>
<td>Clock speed</td>
<td>DC to 80 MHz</td>
</tr>
<tr>
<td>Architecture</td>
<td>General purpose registers, 71 instructions not including addressing mode variants</td>
</tr>
<tr>
<td>On-chip modules</td>
<td>Async serial IO, I2C, SPI, A/D, three 16-bit timers, one 8-bit timer, comparator</td>
</tr>
</tbody>
</table>
PIC24 Core (Simplified Block Diagram)

The **instruction register** contains the machine code of the instruction currently being executed.

ALU (Arithmetic Logic Unit) is 16 bits wide, can accept as operands working registers or data memory.
Memory Organization

Memory on the PIC24 µC family is split into two types: Program Memory and Data Memory.

PIC24 instructions are stored in program memory, which is non-volatile (contents are retained when power is lost).

A PIC24 instruction is 24 bits wide (3 bytes). PIC24HJ32GP202 program memory supports 11264 instructions; the PIC24 architecture can support up to 4M instructions.

PIC24 data is stored in data memory, also known as the file registers, and is a maximum size of 65536 x 8. Data memory is volatile (contents are lost when power is lost).
Program Memory

<table>
<thead>
<tr>
<th>MSW Address</th>
<th>most significant word</th>
<th>least significant word</th>
<th>PC Address (LSW Address)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000001</td>
<td>00000000</td>
<td></td>
<td>0x000000</td>
</tr>
<tr>
<td>0x000003</td>
<td>00000000</td>
<td></td>
<td>0x000002</td>
</tr>
<tr>
<td>0x000005</td>
<td>00000000</td>
<td></td>
<td>0x000004</td>
</tr>
<tr>
<td>0x000007</td>
<td>00000000</td>
<td></td>
<td>0x000006</td>
</tr>
</tbody>
</table>

Program Memory

‘Phantom’ Byte
(read as ‘0’)

Instruction Width

Figure redrawn by author from Fig 3-2 found in the PIC24HJ32GP202/204 datasheet (DS70289A), Microchip Technology Inc.

PC is 23 bits wide, but instructions start on even word boundaries (the PC least significant bit is always 0), so the PC can address 4 Mi instructions.

Locations 0x000000- 0x0001FF reserved, User program starts at location 0x000200.
Data Memory Organization

Data memory for PIC24HJ32GP202

2048 byte SFR space

2048 byte SRAM space

0x0001
0x07FF
0x8001
0x0FFF

0x7FFF
0x8001

0xFFF

0x0000
0x07FE
0x0800
0x0FFF
0x1000

0x1FFF
0x2001

0x7FFF
0x8000

MSB = Most Significant Byte
LSB = Least Significant Byte

Unimplemented on PIC24HJ32GP202

Optional mapped into program memory

16 bits

Figure redrawn by author from Fig 3-3 found in the PIC24HJ32GP202/204 datasheet (DS70289A), Microchip Technology Inc.
Special Function Registers (SFRs)

**Special Function Registers** (SFR) are addressed like normal data memory locations but have specified functionality tied to hardware subsystems in the processor. We typically refer to SFRs by name (W0, T3CON, STATUS, etc) instead of by address.

There are many SFRs in the PIC24 µC – they are used as control registers and data registers for processor subsystems (like the serial interface, or the analog-to-digital converter). We will cover their use and names as we need to.

SFRs live in the address range 0x0000 to 0x07FE in data memory. See the datasheet for a complete list of SFRs.

Other locations in data memory that are not SFRs can be used for storage of temporary data; they are not used by the processor subsystems. These are sometimes referred to as GPRs (general purpose registers). MPLAB refers to these locations as file registers.
8-bit, 16-bit, 32-bit Data

We will deal with data that is 8 bits, 16 bits (2 bytes), and 32 bits (4 bytes) in size. Initially we will use only 8 bit and 16 bit examples.

<table>
<thead>
<tr>
<th>Size</th>
<th>Unsigned Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bits</td>
<td>0 to $2^8-1$ (0 to 255, 0 to 0xFF)</td>
</tr>
<tr>
<td>16-bit</td>
<td>0 to $2^{16}-1$ (0 to 65536, 0 to 0xFFFF)</td>
</tr>
<tr>
<td>32-bit</td>
<td>0 to $2^{32}-1$ (0 to 4,294,967,295, 0 to 0xFFFFFFFF)</td>
</tr>
</tbody>
</table>

The lower 8 bits of a 16-bit value or of a 32-bit value is known as the Least Significant Byte (LSB).

The upper 8 bits of a 32-bit value is known as the Most Significant Byte (MSB).
Storing Multi-byte Values in Memory

16-bit and 32-bit values are stored in memory from least significant byte to most significant byte, in increasing memory locations (little endian order).

Assume the 16-bit value 0x8B1A stored at location 0x1000
Assume the 32-bit value 0xF19025AC stored at location 0x1002

| Location | Contents |  | Location | Contents |
|----------|----------|--------------------------|----------|
| 0x1000   | 0x1A     | LSB                     | 0x1000   | 0x8B1A|
| 0x1001   | 0x8B     | LSB                     | 0x1002   | 0x25AC|
| 0x1002   | 0xAC     | LSB                     | 0x1004   | 0xF190|
| 0x1003   | 0x25     |                          | 0x1006   | ?????|
| 0x1004   | 0x90     |                          | 0x1008   | ?????|
| 0x1005   | 0xF1     |                          | 0x1006   | ?????|

Memory shown as 8 bits wide

Memory shown as 16 bits wide

The LSB of a 16-bit or 32-bit value must begin at an even address (be word aligned).
Data Transfer Instruction

Copies data from Source (src) location to Destination (dst) Location

\[(\text{src}) \rightarrow \text{dst} \quad \text{‘}()\text{’ read as ‘contents of’}\]

This operation uses \textit{two operands}.

The method by which an operand ADDRESS is specified is called the \textit{addressing mode}.

There are many different addressing modes for the PIC24.

We will use a very limited number of addressing modes in our initial examples.
# Data Transfer Instruction Summary

<table>
<thead>
<tr>
<th>Dest Source</th>
<th>Memory</th>
<th>Register direct</th>
<th>Register indirect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Literal</td>
<td>X</td>
<td>MOV{.B} #lit8/16, Wnd lit → Wnd</td>
<td>X</td>
</tr>
<tr>
<td>Memory</td>
<td>X</td>
<td>MOV f_{ALL}, Wnd MOV{.B} f, {WREG} (f_{ALL}) → Wnd/WREG</td>
<td>X</td>
</tr>
<tr>
<td>Register direct</td>
<td>MOV Wns, f_{ALL} MOV{.B} WREG, f (Wns/WREG) → f_{ALL}</td>
<td>MOV{.B} Wso, Wdo (Wso) → Wdo</td>
<td>MOV{.B} Wso, [Wdo] (Wso) → (Wdo)</td>
</tr>
<tr>
<td>Register indirect</td>
<td>X</td>
<td>MOV{.B} [Wso], Wdo ((Wso)) → Wdo</td>
<td>MOV{.B} [Wso], [Wdo] ((Wso)) → (Wdo)</td>
</tr>
</tbody>
</table>

**Key:**
- MOV{.B} #lit8/16, Wnd lit → Wnd
- PIC24 assembly Data transfer

Yellow shows varying forms of the same instruction

f: near memory (0…8095)  
f_{ALL}: all of memory (0…65534)
MOV{.B} Wso, Wdo Instruction

“Copy contents of Wso register to Wdo register”. General form:

\[ \text{mov}\{.b\} \ Wso, Wdo \quad (\text{Wso}) \rightarrow Wdo \]

Wso is one of the 16 working registers W0 through W15 (‘s’ indicates Wso is an operand source register for the operation).

Wdo is one of the 16 working registers W0 through W15 (‘d’ indicates Wdo is an operand destination register for the operation).

\[ \text{mov} \ W3, W5 \quad (\text{W3}) \rightarrow W5 \quad \text{(word operation)} \]
\[ \text{mov.b} \ W3, W5 \quad (\text{W3.LSB}) \rightarrow W5.\text{LSB} \quad \text{(byte operation)} \]

Contents of working register W3 copied to working register W5.

This can either be a word or byte operation. The term ‘copy’ is used here instead of ‘move’ to emphasize that Wso is left unaffected by the operation.

The addressing mode used for both the source and destination operands is called register direct. The mov instruction supports other addressing modes which are not shown.
**MOV** \( W_{so}, W_{do} \)  Instruction Execution

(a) Execute: mov \( W_2, W_1 \) (word mode operation)

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>0x1AF3</td>
</tr>
<tr>
<td>W1</td>
<td>0x8B1A</td>
</tr>
<tr>
<td>W2</td>
<td>0x64DE</td>
</tr>
<tr>
<td>W3</td>
<td>0xFB90</td>
</tr>
</tbody>
</table>

(b) Execute: mov.b \( W_2, W_1 \) (byte mode operation)

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>0x1AF3</td>
</tr>
<tr>
<td>W1</td>
<td>0x8B1A</td>
</tr>
<tr>
<td>W2</td>
<td>0x64DE</td>
</tr>
<tr>
<td>W3</td>
<td>0xFB90</td>
</tr>
</tbody>
</table>
**MOV Wso, Wdo** Instruction Format

(a) **mov{.b} Wso,Wdo**

\[
\begin{array}{cccccccccc}
BB & BB & BB & BB & BB & BB & BB & BB & BB & BB \\
2222 & 1111 & 1111 & 1100 & 0000 & 0000 \\
3210 & 9876 & 5432 & 1098 & 7654 & 3210 \\
\end{array}
\]

\[
0111 \text{ wwww wBhh hddd dggg ss} \\
\]

- **www** = base register \((Wb)\) for indirect offset
- **www** = addressing mode \([Wso/Wdo + Wb]\); otherwise 0
- **B** = 0 for word, 1 for byte
- **hhh** = \(Wdo\) addressing mode (Register direct = 000)
- **ddd** = \(Wdo\) register number (0 to 15)
- **ggg** = \(Wso\) addressing mode (Register direct = 000)
- **ssss** = \(Wso\) register number (0 to 15)

(b) Assembly:

\[
\text{mov W3,W5} \quad \text{Machine Code: 0x780283}
\]

Machine Code = 0111 1000 0[00] 0100 0000 001

- **B** = word mode = 0
- **ssss** = 0011 (register number is 3)
- **ddd** = 0101 (register number is 5)

**ggg**, **hhh**, **www** fields are all 0 because indirect addressing is not used

(c) \text{mov.b W3,W5} \quad 0x784283 \quad \text{Byte mode, only difference is } B = 1
**MOV Wns, f** Instruction

“Copy contents of Wns register to data memory location f.”

General form:

\[
\text{MOV} \quad \text{Wns, } f \quad (\text{Wns}) \rightarrow f
\]

\(f\) is a memory location in data memory, Wns is one of the 16 working registers W0 through W15 (‘s’ indicates Wns is an operand source register for the operation)

\[
\text{MOV} \quad \text{W3, } 0x1000 \quad (\text{W3}) \rightarrow 0x1000
\]

Contents of register W3 copied to data memory location 0x1000. This instruction form only supports WORD operations.

The addressing mode used for both the source operand is register direct.

The address mode used for the destination operand is called *register direct*. 

\[ \text{V 0.1} \]
MOV Wns, f Instruction Execution

Execute: mov W3, 0x1002

<table>
<thead>
<tr>
<th>Location</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1000</td>
<td>0x8B1A</td>
</tr>
<tr>
<td>0x1002</td>
<td>0x25AC</td>
</tr>
<tr>
<td>0x1004</td>
<td>0xFB90</td>
</tr>
<tr>
<td>0x1006</td>
<td>0x9ED7</td>
</tr>
</tbody>
</table>

W3 = 0x64DE

<table>
<thead>
<tr>
<th>Location</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1000</td>
<td>0x8B1A</td>
</tr>
<tr>
<td>0x1002</td>
<td>0x64DE</td>
</tr>
<tr>
<td>0x1004</td>
<td>0xFB90</td>
</tr>
<tr>
<td>0x1006</td>
<td>0x9ED7</td>
</tr>
</tbody>
</table>

Before

W3 = 0x64DE (unaffected)

After

Copyright Delmar Cengage Learning 2008. All Rights Reserved.
From: Reese/Bruce/Jones, “Microcontrollers: From Assembly to C with the PIC24 Family”.
**MOV Wns, f** Instruction Format

(a) 

```
mov Wns, f
BBBB BBBB BBBB BBBB BBBB BBBB
2222 1111 1111 1100 0000 0000
3210 9876 5432 1098 7654 3210
```

(Wns) → f

```
1000 1fff ffff ffff ssss
```

f ... f = upper 15 bits of 16-bit address (lower bit assumed = 0)

ssss = Wns register number (0 to 15)

(b) Assembly:  

```
mov W3, 0x1002
```

Machine Code:

```
0x888013
```

Machine Code = 1000 10000000000001 0011 = 0x888013

(f ... f = 0001 0000 0000 0010

(upper 15-bits of 0x1002)

ssss = 0011 (register number is 3)
MOV $f$, Wnd Instruction

“Copy contents of data memory location \( f \) to register Wnd”. General form:

\[
\text{MOV } f, \text{Wnd} \quad (f) \rightarrow \text{Wnd}
\]

\( f \) is a memory location in data memory, Wnd is one of the 16 working registers W0 through W15 (‘d’ indicates Wnd is an operand destination register for the operation).

\[
\text{MOV } 0x1000, \text{W3} \quad (0x1000) \rightarrow \text{W3}
\]

Contents of data memory location 0x1000 copied to W3.

() is read as “Contents of”.

This is a 16-bit (WORD) operation.
\textbf{MOV f, Wnd} Instruction Execution

Execute: \texttt{mov \text{0x1002},W3}

\begin{tabular}{|c|c|}
\hline
Location & \text{Contents} \\
\hline
\text{0x1000} & \text{0x8B1A} \\
\text{0x1002} & \text{0x25AC} \\
\text{0x1004} & \text{0xFB90} \\
\text{0x1006} & \text{0x9ED7} \\
\hline
\end{tabular}

\begin{tabular}{|c|c|}
\hline
Location & \text{Contents} \\
\hline
\text{0x1000} & \text{0x8B1A} \\
\text{0x1002} & \text{0x25AC} \\
\text{0x1004} & \text{0xFB90} \\
\text{0x1006} & \text{0x9ED7} \\
\hline
\end{tabular}

\begin{itemize}
\item \text{W3 = 0x64DE} \text{ (Before)}
\item \text{W3 = 0x25AC (modified)} \text{ (After)}
\item \text{Location} \text{0x1000} \text{ (unaffected)}
\item \text{Location} \text{0x1002} \text{ (unaffected)}
\item \text{Location} \text{0x1004} \text{ (unaffected)}
\item \text{Location} \text{0x1006} \text{ (unaffected)}
\end{itemize}
A Note on Instruction Formats

• The instruction formats (machine code) of some instructions will be presented for informational purposes
  – However, studying the machine code formats of the instructions is not a priority; understanding instruction functionality will be emphasized.
  – All instruction formats can be found in the dsPIC30F/dsPIC33F Programmers Reference manual from Microchip
  – The PIC24 family is a subset of the dsPIC30F/dsPIC33FF instruction set – the PIC24 family does not implement the DSP instructions.
A Broader View of the Instruction Set

- Instruction set summary
- Appendix A
- MOV, ADD, and INC summary