Chapter 2

Microcontroller Architecture—PIC18F Family

Updated 2/14/2016
Fetch-Execution Cycle

- **Basic Operations of MCU**
  - Fetch, Decode, Execute

- **Two models**
  - **Sequential** fetch-execute cycle
    - Complete the cycle before starting a new one
    - **Time to complete the task:** \( T = t_1 + t_2 + t_3 + \ldots + t_n \)
  - **Pipelining**
    - Break the fetch-execute cycle into a number of separate stages, so that when one stage is being carried out for a particular instruction, the CPU can carry out another stage for a second instruction, and so on.
    - Originated from the basic concept used in assembly lines
      - Each instruction still takes the same number of cycles to complete
      - The gain comes from the fact that the CPU can operate on instructions in the different stages in parallel.
      - The total time to complete the task is the same as above.
      - **Clock period for completing each task:** \( T_p = \max(t_1, t_2, t_3, \ldots, t_n) \)
Pipelining

- Fetch-execute cycle vs. Fetch-execute cycle using pipelining
- Throughput of the operation is defined as $1/T$ (operations or instructions/second)
- Generally, the faster the clock the higher the throughput will be – however…. (next slide)

Assuming all stages are finished in a single (or n) clock cycle
Clock Rate Limitation in Pipelining

- Increasing the clock speed does not guarantee significant performance gains.

- This is because the speed of the processor is effectively determined by the rate at which it can fetch instructions and data from memory.

  **Example:** if the processor spends 90% of its time waiting on memory, the performance gained by doubling the processor speed (without improving the memory access time) is only 5%.
Cache

- One way of improving memory access time
  - use of a cache memory system
- The processor operates at its maximum speed if the data to be processed is in its registers.
  - Register storage capacity is very limited!
- One, very effective way of overcoming the slow access time of main memory, is to design a faster intermediate memory system, that lies between the CPU and main memory.
- Such memory is called cache memory (or simply cache)
  - Cache memory is high speed memory (e.g. SRAM) which can be accessed much more quickly than normal memory (usually dynamic RAM (DRAM)).
  - It has a smaller capacity than main memory and it holds recently accessed data from main memory
- Computers may use separate memories to store instructions and data
  - Harvard Architecture

http://www.csi.ucd.ie/staff/jcarthy/home/alp/alp7.html
Memory Model – Von Neumann Architecture

- Fetches instructions and data from a single memory space
  - Also known as Princeton architecture
- Limits operating bandwidth
- CISC designs are also more likely to feature this model
- Uses **unified cache** memory: instructions and data may be stored in the same cache memory
- Can be either reading an instruction or reading/writing data from/to the memory
  - Both cannot occur at the same time since the instructions and data use the same bus system.

Memory Model – (Pure or Strict) Harvard Architecture

- The original Harvard architecture computer, the Harvard Mark I, employed entirely separate memory systems to store instructions and data.
- Uses two separate memory spaces for program instructions and data - separate pathways with separate address spaces
  - Allows for different bus widths
  - Improved operating throughput
- RISC designs are also more likely to feature this model
- Note that having separate address spaces can create issues for high-level programming no supporting different address spaces (not good for CISC!)
- The CPU can both read an instruction and perform a data memory access at the same time, even without a cache
  - Faster (than Von Neumann) for a given circuit complexity because instruction fetches and data access do not contend for a single memory pathway.
- Example: PIC Microcontrollers (Separate code and data spaces)
Memory Model –
(Modified or Non-Strict) Harvard architecture

- A Modified Harvard architecture machine is very much like a Harvard architecture machine.
- Modification can be different:
  1. The program and data memory occupy different address spaces, but there are operations to read and/or write program memory as data.
  2. It relaxes the strict separation between memories while still letting the CPU concurrently access two (or more) memory busses.
     - It offers separate pathways with the unified address spaces of the memory.
     - As far as the programmer is concerned the machine performs like a von Neumann machine.
- Remember: many modern computers that are documented as Harvard Architecture are, in fact, Modified Harvard Architecture.
- Applications:
  - Atmel AVR 8-bit RISC microcontroller
  - PlayStation Portable's WLAN chip, and many more; anything with enhanced DSP application; x86 (Intel) processors, ARM cores (ARM9) embedded as applications processors in cell phones, and PowerPC.
Some Examples:

- Microcontrollers
  - **LPC210x** - ARM7 Microcontroller LPC210x – RISC-based microcontroller; Harvard
  - **ATmega128** - AVR Microcontroller (developed by Atmel), Harvard, RISC
  - **PIC Microcontroller** – Harvard, RISC
  - **68HC11/MC68HC24**; descended from Motorola 68xxx microprocessor, which is a 8-bit CISC microcontroller - Von Neumann architecture
  - **Z8 Microcontrollers** – Harvard
  - **Intel 8051** - 8-bit Harvard architecture, single chip microcontroller; CISC instruction

- Microprocessors
  - **Intel x86** – CICS; Von Neumann (Intel, AMD, etc.)
  - **68xxx Motorola** - 16/32-bit CISC - competitor to Intelx86
  - **ARM** - 32 bit (used by Atmel), RISC
  - **SPARC V9 ISA** – used in Sun UltraSparc – RISC processor; developed by Sun Microsystems

http://www.experiencefestival.com/microcontroller_-_intel
Main 8-bit Controllers

- **Microchip**
  - RISC architecture (reduced instruction set computer)
  - Has sold over 2 billion as of 2002
  - Cost effective and rich in peripherals

- **Motorola**
  - CISC architecture
  - Has hundreds of instructions
  - Examples: 68HC05, 68HC08, 68HC11

- **Intel**
  - CISC architecture
  - Has hundreds of instructions
  - Examples: 8051, 8052
  - Many difference manufacturers: Philips, Dallas/MAXIM Semiconductor, etc.

- **Atmel**
  - RISC architecture (reduced instruction set computer)
  - Cost effective and rich in peripherals
  - AVR
PIC Microcontroller with the Harvard Architecture

- Three types of memory
  - Data Memory
  - Program Memory
  - Stack Memory

Numbers refer to Data bus (not address)
Program Memory

- Program memory is **16-bits** wide accessed through a **separate** program data bus and address bus inside the PIC18.
- Program memory stores the program and also static data in the system.
  - On-chip
  - External
- On-chip program memory is either PROM or EEPROM.
  - The PROM version is called OTP (one-time programmable) (PIC18C)
  - The EEPROM version is called Flash memory (PIC18F).
- Maximum size for program memory is **2M**
  - Program memory addresses are **21-bit** address starting at location 0x000000

**Example:** PIC18F4520 has 32K program memory – draw it!
Data Memory (1)

- Used for transitory data when the program is being executed
  - Example: \( A=1, B=2, C=3 \) \( X=A+B+C \rightarrow A+B=W; W+C=W \)

- Data memory is either SRAM or EEPROM.
  - Some chips only have SRAM
  - Others may have SRAM and EEPROM
    - EEPROM stores permanently

- Various PIC18 versions contain between 256 and 3968 bytes of data memory
  - For example: SRAM data memory begins at 12-bit address 0x000 and ends at 12-bit address 0xFFF (4K)
Data Memory (2)

- Data memory is often divided into two sections
  - General Function Registers (GFR) or register file location
    - 000-0xF7F locations
  - Special Function Registers (SFR) – specific to PIC
    - 0xF80-0xFFF (upper 128 bytes)

- Depending on the PIC chip, the sizes for GFR and SFR are different
Program Stack Memory
Saving the return address

- The PIC18 contains a program stack that stores up to 31 return addresses from functions.
  - 31-deep
  - The program stack is 21 bits in width as is the program address (remember address memory is 2M)
- Stack memory uses SRAM
- Operation of a stack
  - When a function is called, the return address (location of the next step in a program) is pushed onto the stack.
    - For example: Stack number 1 will have value= 0x1F0000
  - When the return occurs within the function, the return address is retrieved from the stack and placed into the program counter.
Summary: Microcontroller Types

Our focus for the rest of the course:

Microchip Technology

- **PIC** Microcontroller
  - Programmable Interface Controller
  - 8-bit MCU (depending on RAM size, IO pins, stack size, enhanced architecture)
    - Base-line (including Dust, 6-pin, no interrupts)
    - Mid-range
    - High-end (PIC18F, uses C18 compiler)
  - 16-bit MCU
    - Enhanced with dsp features (support for VoIP)
    - Smaller, faster, low-power; uses C30 Compiler
    - PIC24, dsp30/33
  - 32-bit MCU
    - Instruction cache, low-power, faster RAM
    - C32 compiler

See this link:
PIC18F452/4520/45K20

Memory - Example

- **Program Memory**: 32 K ($2^{15}$)
  - Address range: 000000 to 007FFFH
  - 16-bit registers

- **Data Memory**: 4 K
  - Address range: 000 to FFFH
  - 8-bit registers

- **Data EEPROM**
  - Not part of the data memory space
  - Addressed through special function registers

See this link:
PIC18F – MCU and Memory

- ALU
  - WREG
  - Status
  - Instruction Decoder
- Registers
  - Bank Select Register
  - File Select Registers
  - Program Counter
- Control Unit
  - Read
  - Write

Program Memory
- 2 MB
- \(2^{21}\)

Data Memory
- 4 KB
- \(2^{12}\)

16-bit
- 21-Bit Address Bus
- 16-Bit Instruction/Data Bus

8-bit
- 12-Bit Address Bus
- 8-Bit Data Bus
Design a microcontroller with the following specifications:

- Program Memory: 32 K (15 bits)
- Data Memory: 4 K (12 bits)

In your design show where the counter registers are located.

In your design show where the working registers are located (which part of the microprocessor unit).

Assuming each memory has a R/W and OE, show how they are connected to MPU.

- Show where the read/write lines are connected to – specify the direction of each.

Do it on your own!
Microprocessor Unit (1 of 3)
Includes Arithmetic Logic Unit (ALU)

- Includes Arithmetic Logic Unit (ALU), Registers, and Control Unit
  - Arithmetic Logic Unit (ALU)
    - Performs logical and arithmetic functions
    - WREG – working register (acts as an accumulator) – used to perform arithmetic or logical functions
    - Status register that stores flags – indicates the status of the operation done by ALU
    - Instruction decoder (ID) – when the instruction is fetched it goes into the ID to be interpreted – tell the processor what to do
Microprocessor Unit (1 of 3)
Includes Arithmetic Logic Unit (ALU)
General ALU Architecture

All arithmetic and logical instructions are carried out by the ALU.

What is the function?

Microprocessor Unit (1 of 3)
Includes Arithmetic Logic Unit (ALU)
General ALU Architecture

An eight bit instruction informs the ALU which operation it is to carry out.

One number to be manipulated comes from the accumulator, the other from memory or another register.

Flags in the status register are set to indicate the result, such as negative etc.
Microprocessor Unit (2 of 3)

- Registers – hold memory address
  - Bank Select Register (BSR)
    - 4-bit register used in direct addressing the data memory
  - File Select Registers (FSRs)
    - 16-bit registers used as memory pointers in indirect addressing data memory
  - Program Counter (PC)
    - 21-bit register that holds the program memory address while executing programs
Microprocessor Unit (3 of 3)

- Control unit
  - Provides timing and control signals to various Read and Write operations
Examples

- Show how four 8-bit RAM blocks each having 1KB capacity can be connected to the CPU. Assume the address bus is limited to 10 bits and each RAM chip has the following pins: OE and R/W

Refer to your notes: Using DEMUX
### List of Selected Microcontroller Families from Microchip

<table>
<thead>
<tr>
<th>Part No.</th>
<th>Program</th>
<th>EE PROM</th>
<th>RAM</th>
<th>Total Pins</th>
<th>I/O Pins</th>
<th>Analog Comp.</th>
<th>Digital</th>
<th>CCP/</th>
<th>Max Speed</th>
<th>Instruction</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>10F200</td>
<td>256x12</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>1-8 bit, 1-WDT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10F220</td>
<td>256x12</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>1-8 bit, 1-WDT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12F510</td>
<td>1536x12</td>
<td>38</td>
<td>8</td>
<td>6</td>
<td>1-8 bit, 1-WDT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16F506</td>
<td>1536x12</td>
<td>67</td>
<td>14</td>
<td>12</td>
<td>1-8 bit, 1-WDT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16C55A</td>
<td>768x12</td>
<td>24</td>
<td>28</td>
<td>20</td>
<td>1-8 bit, 1-WDT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16CR58B</td>
<td>3072x12</td>
<td>73</td>
<td>18</td>
<td>12</td>
<td>1-8 bit, 1-WDT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12F683</td>
<td>2048x14</td>
<td>256</td>
<td>128</td>
<td>8</td>
<td>1-16 bit, 2-8 bit, 1-WDT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16F687</td>
<td>2048x14</td>
<td>256</td>
<td>128</td>
<td>20</td>
<td>1-16 bit, 1-8 bit, 1-WDT</td>
<td>EU/</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18F1230</td>
<td>2048x16</td>
<td>256</td>
<td>18-28</td>
<td>16</td>
<td>2-16 bit</td>
<td>EU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18F4520</td>
<td>16384x16</td>
<td>256</td>
<td>1536</td>
<td>40-44</td>
<td>1-16 bit, 1-WDT</td>
<td>EU/ MiC/</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18F6527</td>
<td>24576x16</td>
<td>1024</td>
<td>3936</td>
<td>64</td>
<td>2-8 bit, 1-16 bit</td>
<td>EU/ MiC/</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18F8622</td>
<td>32768x16</td>
<td>1024</td>
<td>3936</td>
<td>80</td>
<td>2-8 bit, 1-16 bit</td>
<td>EU/ MiC/</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18F96J60</td>
<td>32768x16</td>
<td>2048</td>
<td>100</td>
<td>72</td>
<td>2-8 bit, 1-16 bit</td>
<td>EU/ MiC/</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24FJ128GA-010</td>
<td>65536x16</td>
<td>8192</td>
<td>100-128</td>
<td>86</td>
<td>16x10-bit</td>
<td>2-16 bit</td>
<td>EU/ MiC/</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Abbreviations:**
1) ADC: Analog-Digital Converter, 2) AUSART: Addressable USART, 3) CCP: Capture/Compare/PWM, 4) ECCP: Enhanced CCP,
5) EU: Enhanced USART, 6) Enh Flash: Enhanced Flash, 7) I²C: Inter-integrated Circuit Bus, 8) MiC/SPI: Master I²C/SPI, 9) OTP: One-Time Programmable,
10) SPI: Serial Peripheral Interface, 11) USART: Universal Synchronous/Asynchronous Receiver/Transmitter, 12) WDT: Watchdog Timer
### List of Selected Microcontroller Families from Microchip

<table>
<thead>
<tr>
<th>Part No.</th>
<th>Program OTP/Flash</th>
<th>EE PROM</th>
<th>RAM</th>
<th>Total Pins</th>
<th>I/O Pins</th>
<th>Analog ADC</th>
<th>Comp.</th>
<th>Digital Timers/ WDT</th>
<th>CCP/ ECCP</th>
<th>Max Speed MHz</th>
<th>Instruction Size</th>
<th>Total Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>10F200</td>
<td>256x12 Flash</td>
<td></td>
<td>16</td>
<td>8</td>
<td>4</td>
<td></td>
<td></td>
<td>1-8 bit, 1-WDT</td>
<td>12-bit</td>
<td>4</td>
<td>12-bit</td>
<td>33</td>
</tr>
<tr>
<td>10F220</td>
<td>256x12 Flash</td>
<td></td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2x8-bit</td>
<td></td>
<td>1-8 bit, 1-WDT</td>
<td>12-bit</td>
<td>8</td>
<td>12-bit</td>
<td>33</td>
</tr>
<tr>
<td>12F510</td>
<td>1536x12 Flash</td>
<td></td>
<td>20</td>
<td>16</td>
<td>12</td>
<td></td>
<td></td>
<td>1-8 bit, 1-WDT</td>
<td>12-bit</td>
<td>20</td>
<td>12-bit</td>
<td>33</td>
</tr>
<tr>
<td>16F506</td>
<td>1536x12 Flash</td>
<td></td>
<td>20</td>
<td>16</td>
<td>12</td>
<td></td>
<td></td>
<td>1-8 bit, 1-WDT</td>
<td>12-bit</td>
<td>20</td>
<td>12-bit</td>
<td>33</td>
</tr>
<tr>
<td>16C55A</td>
<td>768x12 OTP</td>
<td></td>
<td>12</td>
<td>16</td>
<td>12</td>
<td></td>
<td></td>
<td>1-8 bit, 1-WDT</td>
<td>12-bit</td>
<td>12</td>
<td>12-bit</td>
<td>33</td>
</tr>
<tr>
<td>16CR58B</td>
<td>3072x12 ROM</td>
<td></td>
<td>12</td>
<td>16</td>
<td>12</td>
<td></td>
<td></td>
<td>1-8 bit, 1-WDT</td>
<td>12-bit</td>
<td>12</td>
<td>12-bit</td>
<td>33</td>
</tr>
<tr>
<td>12F683</td>
<td>2048x14 Flash</td>
<td></td>
<td>20</td>
<td>16</td>
<td>12</td>
<td></td>
<td></td>
<td>1-8 bit, 1-WDT</td>
<td>12-bit</td>
<td>20</td>
<td>12-bit</td>
<td>33</td>
</tr>
<tr>
<td>16F687</td>
<td>2048x14 Flash</td>
<td></td>
<td>20</td>
<td>16</td>
<td>12</td>
<td></td>
<td></td>
<td>1-8 bit, 1-WDT</td>
<td>12-bit</td>
<td>20</td>
<td>12-bit</td>
<td>33</td>
</tr>
<tr>
<td>18F1230</td>
<td>2048x16 Enh Flash</td>
<td></td>
<td>256</td>
<td>18-28</td>
<td>16</td>
<td>4x10-bit</td>
<td>16</td>
<td>1-8 bit, 1-WDT</td>
<td>EU</td>
<td>40</td>
<td>16-bit</td>
<td>77</td>
</tr>
<tr>
<td>18F4520</td>
<td>16384x16 Enh Flash</td>
<td></td>
<td>256</td>
<td>1536</td>
<td>40-44</td>
<td>32</td>
<td></td>
<td>13x10-bit</td>
<td>EU/ MIF C /SPI</td>
<td>40</td>
<td>16-bit</td>
<td>77</td>
</tr>
<tr>
<td>18F6527</td>
<td>24576x16 Enh Flash</td>
<td></td>
<td>3936</td>
<td>64</td>
<td>64</td>
<td>12x10-bit</td>
<td></td>
<td>2-16 bit</td>
<td>EU/ MIF C /SPI</td>
<td>40</td>
<td>16-bit</td>
<td>77</td>
</tr>
<tr>
<td>18F8622</td>
<td>32768x16 Enh Flash</td>
<td></td>
<td>3936</td>
<td>80</td>
<td>70</td>
<td>16x10-bit</td>
<td></td>
<td>2-8 bit, 1-WDT</td>
<td>2EU/2 - MIF C /SPI</td>
<td>40</td>
<td>16-bit</td>
<td>77</td>
</tr>
<tr>
<td>18F96J60</td>
<td>32768x16 Flash</td>
<td></td>
<td>2048</td>
<td>100</td>
<td>72</td>
<td>16x10-bit</td>
<td></td>
<td>2-8 bit, 3-16 bit</td>
<td>2EU/2 - MIF C /SPI</td>
<td>42</td>
<td>16-bit</td>
<td>77</td>
</tr>
<tr>
<td>24FJ128GA-010</td>
<td>65536x16 Flash</td>
<td></td>
<td>8192</td>
<td>100-128</td>
<td>86</td>
<td>16x10-bit</td>
<td></td>
<td>5-16 bit, 1-WDT</td>
<td>2-UART 2-16C /SPI</td>
<td>32</td>
<td>16-bit</td>
<td>77</td>
</tr>
</tbody>
</table>


**Flash (4K)**

EEPROM – can be accessed individually

**36 I/O ports**

F→FLASH - ROM

C→PROM (OTP)- ROM
PIC18F452/4520 Memory

- Program memory with addresses (Flash)-
- Data memory with addresses
- Also called Data Register or File Register

Remember: all instructions in PIC18 family are one word in length – read by the processor in one cycle;
Instructions

8-bit Instruction on typical 8-bit MCU
Example: Freescale ‘Load Accumulator A’:
• 2 Program Memory Locations
• 2 Instruction Cycles to Execute

\[ \text{inst} \ k \]

\[
\begin{array}{cccccc}
1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
\hline
k & k & k & k & k & k & k & k
\end{array}
\]

- Limits Bandwidth
- Increases Memory Size Requirements

16-bit Instruction on PIC18 8-bit MCU
Example: ‘Move Literal to Working Register’
• 1 Program Memory Location
• 1 Instruction Cycle to Execute

\[ \text{movlw} \ k \]

\[
\begin{array}{ccccccccc}
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & k & k & k & k & k & k & k & k
\end{array}
\]

- Separate busses allow different widths
- 2k x 16 is roughly equivalent to 4k x 8
Direct and Indirect Addressing

- **Direct addressing**
  - MOVWF REG10 ; Directly writing R→ REG10

- **Indirect addressing**
  - We don’t directly access the register by its address
  - We use pointers to access registers
  - For example, FSR0 contains the pointer value
    - We move W→FSR0 (special register)
    - Then the value stored in W will go into the register identified by FSR0
Data Memory Organization

- Data Memory up to 4k bytes
  - Data register map - with 12-bit address bus 000-FFF

$$FFF = 2^{12} = 16 \times 256 = 4096 = 4\text{KB SRAM}$$
Data Memory Organization

- Data Memory up to 4k bytes
  - Data register map - with 12-bit address bus 000-FFF
- Divided into 256-byte banks
- There are total of F banks
- Half of bank 0 and half of bank 15 form a virtual (or access) bank that is accessible no matter which bank is selected – this selection is done via 8-bits
  - Access Bank
    - SFR: Special Function Register (e.g., accessing the IO ports)
    - GPR: Used as a general purpose register

FFF=2^{12}=16\times256=4096=4KB SRAM

PIC16F8F2520/4520 Register File (data memory) Map

GPR=General Purpose Reg.
SFR=Special Function Reg.
PIC18F452/4520 – Data Memory with Access Banks

- Three ways to access data registers from the MPU:
  - Direct using Bank Select Registers (BSR)
    - Bank address (4-bit) + Instruction (8-bit)
  - Indirect using File Select Registers (FSR)
    - FSR contains the address of the data register
    - MPU uses FSR to access data registers
  - Access Bank
    - Directly accessible via 8-bits of register

Don’t confuse FSR and SFR!

So how do we know, say, address 0xF4 is referring to a SFR or GPR in BANK 0?
Basic Programming Model

- d-bit refers to destination. E.g., d=1, the result will go into data memory.
- a-bit determines if we are accessing the access bank or BANK.

<table>
<thead>
<tr>
<th>Op-code</th>
<th>8-bit data memory address</th>
</tr>
</thead>
<tbody>
<tr>
<td>a-bit</td>
<td>a = 0 access bank</td>
</tr>
<tr>
<td></td>
<td>a = 1 use BSR</td>
</tr>
<tr>
<td>d-bit</td>
<td>d = 0 WREG</td>
</tr>
<tr>
<td></td>
<td>d = 1 data memory address</td>
</tr>
</tbody>
</table>
Basic Programming Model

- Note that the RAM (file register or data memory) can be accessed via the following:
  - BSR + 8-bit
  - FSR (three File Select Registers - FSR)
- When ACCESS BANK is selected:
  - BANK0,F + 4-bits
Basic Programming Model

Examples:

<table>
<thead>
<tr>
<th>Label</th>
<th>Op-code</th>
<th>Operand</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start:</td>
<td>MOVLW</td>
<td>0x00</td>
<td>;load WREG with 0x00</td>
</tr>
<tr>
<td></td>
<td>GOTO</td>
<td>Start</td>
<td>;repeat</td>
</tr>
</tbody>
</table>

MOVLW 0x06 ;place a 0x06 into W
ADDLW 0x02 ;add a 0x02 to W
MOVWF 0x00, 0 ;copy W to access bank register 0x00

; OR another version using the ACCESS keyword

MOVLW 0x06 ;place a 0x06 into W
ADDLW 0x02 ;add a 0x02 to W
MOVWF 0x00, ACCESS ;copy W to access bank register 0x00
Basic Programming Model

Examples:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVLW 0x06</td>
<td>place a 0x06 into W</td>
</tr>
<tr>
<td>ADDLW 0x02</td>
<td>add a 0x02 to W</td>
</tr>
<tr>
<td>MOVLB 2</td>
<td>load BSR with bank 2</td>
</tr>
<tr>
<td>MOVWF 0x00, 1</td>
<td>copy W to data register 0x00</td>
</tr>
<tr>
<td></td>
<td>of bank 2 or address 0x200</td>
</tr>
</tbody>
</table>

; OR using the BANKED keyword

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVLW 0x06</td>
<td>place a 0x06 into W</td>
</tr>
<tr>
<td>ADDLW 0x02</td>
<td>add a 0x02 to W</td>
</tr>
<tr>
<td>MOVLB 2</td>
<td>load BSR with 2</td>
</tr>
<tr>
<td>MOVLF 0x00, BANKED</td>
<td>copy W to data register 0x00</td>
</tr>
<tr>
<td></td>
<td>of bank 2 or address 0x200</td>
</tr>
</tbody>
</table>

; OR without any bank indication

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVLW 0x06</td>
<td>place a 0x06 into W</td>
</tr>
<tr>
<td>ADDLW 0x02</td>
<td>add a 0x02 to W</td>
</tr>
<tr>
<td>MOVLB 2</td>
<td>load BSR with bank 2</td>
</tr>
<tr>
<td>MOVWF 0x00</td>
<td>copy W to data register 0x00</td>
</tr>
<tr>
<td></td>
<td>of bank 2 or address 0x200</td>
</tr>
</tbody>
</table>
PIC18F452 I/O Ports

- Five I/O ports
  - PORT A through PORT E
  - Most I/O pins are multiplexed
  - Generally have eight I/O pins with a few exceptions
  - Addresses already assigned to these ports in the design stage
  - Each port is identified by its assigned Special Function Registers (SFR) – look at the previous slide
    - PORTA (address of F80)
    - PORTB (address of F81)
    - these are part of data memory or register file

TRISB must be set to specify signal direction of PORT B.
Processes and Conditions of Data Transfer

- Interrupt is a process of communication between two devices
  -  If provides efficient communication between the two devices
  -  Examples: Sending a file to a printer, pressing a key on the keyboard

- External or Internal to the MPU
Processes and Conditions of Data Transfer

Parallel data transfer
Serial data transfer

MPU Initiating

MPU Initiated

Unconditional

Conditional (asks if device is ready)

Unconditional

Polling

Handshake

Externally Requested

Interrupt Process

RST – upon different conditions

HW – a key is pressed!

SW – overflow occurs
Processes and Conditions of Data Transfer

- **Reset**
  - Special type of external interrupt
  - Examples:
    - Manual Reset
    - Power-on Reset
    - Brown-out Reset (power goes below a specified value)
MCU Support Devices (1 of 2)

- **Timers**
  - A value is loaded in the register and continue changing at every clock cycle – time can be calculated
  - Can count on rising or falling edge
  - There are several timers: 8-bit, 16-bit
  - Controlled by SFR

- **Master Synchronous Serial Port (MSSP)**
  - Serial interface supporting RS232

- **Addressable USART**
  - Universal Sync/Async Rec/Transmitter
  - Another serial data communication
  - Similar to modem interfacing – also supports transfer between two microcontrollers to enhance IO ports

- **A/D converter**
  - 10-bit
  - Accepts analog signals from 13 channels

- **Parallel Slave Port (PSP)**
  - Used for interfacing with other MPU or MCU

- **Capture, Compare and PWM (CCP Module)**

![Diagram of MCU Support Devices](attachment:diagram.png)
PIC18F Special Features

- Sleep mode
  - Power-down mode
- Watchdog timer (WDT)
  - Able to reset the processor if the program is caught in unknown state (e.g., infinite loop)
- Code protection
  - EEPROM can be protected through SFR
- In-circuit serial programming
- In-circuit debugger
PIC18F4X2 Architecture Block Diagram (page 46)
PIC16F87
Architecture
Block Diagram
PIC18F Instructions and Assembly Language

- Has 77 instructions
  - Earlier PIC family of microcontrollers have either 33 or 35 instructions (Table 2-1)
- In PIC18F instruction set, all instructions are 16-bit word length except four instructions that are 32-bit length
Instruction Description and Illustrations

- **Copy (Load) 8-bit number into W register**
  - Mnemonics: **MOVLW** 8-bit
  - Binary format:
    - **0000 1110 XXXX XXXX** (any 8-bit number)

- **Copy Contents of W register in PORTC**
  - Mnemonics: **MOVWF PORTC, a**
    - (‘a’ indicates that PORTC is in the Access Bank)
  - Binary format:
    - **0000 1110 1000 0010** (82H is PORTC address)
## Instruction Set Overview

### Literal and Control Operations

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Literals</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVLW</td>
<td>0x25</td>
<td></td>
</tr>
</tbody>
</table>

**Diagram: OR**

- **MOVLW 0x25**
Problem statement:
- Write instructions to light up alternate LEDs at PORTC.

Hardware:
- PORTC
  - bidirectional (input or output) port; should be setup as output port for display
- Logic 1 will turn on an LED in Figure 2.10.
Illustration (2 of 5)

- Interfacing LEDs to PORTC
- Port C is F82H
- Note that PORT C is set to be an output!
- Hence, TRISC (address 94H) has to be set to 0
Illustration (3 of 5)

- Program (software)
  - Logic 0 to TRISC sets up PORTC as an output port
  - Byte 55H turns on alternate LEDs
    - MOVLW 0x7F
    - MOVWF ADCON1 ;select all digital pins for ports
    - MOVLW 00 ;Load W register with 0
    - MOVWF TRISC, 0 ;Set up PORTC as output
    - MOVLW 0x55 ;Byte 55H to turn on LEDs
    - MOVWF PORTC,0 ;Turn on LEDs
    - SLEEP ;Power down

- [Illustration content]

- Diagram content
IO Port Access

- Bit n in TRISx controls the data direction of Bit n in PORTx
- 1 = Input, 0 = Output
Analog or Digital I/O?

- Some I/O pins multiplexed with analog inputs (analog by default)
- ADCON1 used to determine whether pin is analog in or digital I/O

Set lower 4 bits to ‘1’ to make all multiplexed pins digital
PIC18 Simulator

- Using the Program Memory editor type in the opcode `MOVLW 00` and `MOWWF TRISC,0` as described in page 52 of your textbook.
- Run the program in step-by-step mode and observe the PC.
- Observe how the NEXT INSTRUCTION changes.
- What is the value of final clock cycle?
- How long does it take to complete the program in sec.?
PIC18 Simulator IDE
Questions - PIC18 Simulator IDE

- What is the address for TRISC? SFR → F94
- What is the address for PORTE?
- How many SFR registers we have? FFF-F80
- How many GPR? 000-5FF
- How many bit PC has? 21
Example

<table>
<thead>
<tr>
<th>Address</th>
<th>Hex Value</th>
<th>Binary Value</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000h</td>
<td>0E00h</td>
<td>0001110000000000</td>
<td>MOV LW 0x00</td>
</tr>
<tr>
<td>000002h</td>
<td>6E94h</td>
<td>0110111010010100</td>
<td>MOVWF TRISC,A</td>
</tr>
<tr>
<td>000004h</td>
<td>0EEAh</td>
<td>0000111010101010</td>
<td>MOV LW 0xAA</td>
</tr>
<tr>
<td>000006h</td>
<td>6E82h</td>
<td>0110111010000001</td>
<td>MOVWF PORTC,A</td>
</tr>
<tr>
<td>000008h</td>
<td>0003h</td>
<td>0000000000000000</td>
<td>SLEEP</td>
</tr>
</tbody>
</table>

Leave space
Illustration (4 of 5)

- Execution of the instruction:
  \[ \text{WREG} = AA \]
  \[ \text{MOVWF PORTC} \]

Copy from WREG \(\rightarrow\) PORT C (82H)

Program Memory Editor:

<table>
<thead>
<tr>
<th>Address</th>
<th>Hex Value</th>
<th>Binary Value</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000h</td>
<td>0E00h</td>
<td>000000000000</td>
<td>MOVWF 0x00</td>
</tr>
<tr>
<td>000002h</td>
<td>6E00h</td>
<td>000000000000</td>
<td>MOVWF TRISC,A</td>
</tr>
<tr>
<td>000004h</td>
<td>6E82h</td>
<td>000000000001</td>
<td>MOVWF PORTC,A</td>
</tr>
<tr>
<td>000008h</td>
<td>000000000000</td>
<td>SLEEP</td>
<td></td>
</tr>
<tr>
<td>00000Ah</td>
<td>FFFFh</td>
<td>111111111111</td>
<td>NOP</td>
</tr>
<tr>
<td>000010h</td>
<td>FFFFh</td>
<td>111111111111</td>
<td>NOP</td>
</tr>
<tr>
<td>000012h</td>
<td>FFFFh</td>
<td>111111111111</td>
<td>NOP</td>
</tr>
<tr>
<td>000014h</td>
<td>FFFFh</td>
<td>111111111111</td>
<td>NOP</td>
</tr>
<tr>
<td>000016h</td>
<td>FFFFh</td>
<td>111111111111</td>
<td>NOP</td>
</tr>
<tr>
<td>000018h</td>
<td>FFFFh</td>
<td>111111111111</td>
<td>NOP</td>
</tr>
<tr>
<td>00001Ah</td>
<td>FFFFh</td>
<td>111111111111</td>
<td>NOP</td>
</tr>
<tr>
<td>00001Ch</td>
<td>FFFFh</td>
<td>111111111111</td>
<td>NOP</td>
</tr>
<tr>
<td>00001Eh</td>
<td>FFFFh</td>
<td>111111111111</td>
<td>NOP</td>
</tr>
</tbody>
</table>
Another Example
References

- Read the Wiki on Microchip:
- Flag simulator: http://www.ee.unb.ca/cgi-bin/tervo/alu.pl
- PIC Tutorial (flash-based speaking instructor will be tutoring you...):
  http://www.pictutorials.com/Flash_Tutorials.htm