Introduction to the PIC18 Microcontroller

Hsiao-Lung Chan
Dept Electrical Engineering
Chang Gung University, Taiwan
chanhl@mail.cgu.edu.tw
Outline

- Introduction to the PIC18 microcontroller
- PIC18 assembly language programming
- Interrupts, resets, and configuration
- Parallel ports
- Timers and CCP modules
- Serial interface
- Analog-to-digital converter
Grading policy

- Midterm examinations + final test 60%
- Attendance + efforts 20%
- Reports 20% (No plagiarism)
PIC microcontroller vs. Intel MCS-51 (8051)

- "Peripheral Interface Controller" made by Microchip Technology
- 8-bit ALU
- Intel's original in the 1980s.
- Several companies offer MCS-51 as IP cores in FPGAs or ASICs.
68HC MCU and AVR

- **68HC microcontroller**
  - 8-bit microcontroller family introduced by Motorola in 1985. Now produced by Freescale Semiconductor
  - CISC (complex instruction set computer) design

- **AVR**
  - 8-bit RISC MCU was sold to Atmel from Nordic VLSI
Computer hardware organization

- Processor
  - Control Unit
  - Datapath
    - Arithmetic Logic Unit
    - Registers
  - Common Bus (address, data, & control)
  - Memory
    - Program Storage
    - Data Storage
  - Output Units
  - Input Units
Microprocessor vs. microcontroller (MCU)

- Microprocessor
  - A processor implemented on a very large scale integration (VLSI) chip
  - Peripheral chips are needed to construct a product
- Microcontroller
  - The processor and peripheral functions implemented on one VLSI chip
Features of PIC18 MCU

- 8-bit CPU
- Memory
  - Electrical erasable programmable read-only memory (EEPROM)
  - Flash memory
  - Static random-access memory (SRAM)
- Timers, including counters, input capture, output compare, real-time interrupt, and watchdog timer
- Pulse-width modulation (PWM)
- Parallel I/O ports
- SPI, I²C, controller area network (CAN) serial interface
- Universal asynchronous receiver transmitter (UART)
- 10-bit A/D converter
Computer software

- Machine instruction
- Assembly language
- High-level language

- Source code
- Object code
# Source code and object code examples

<table>
<thead>
<tr>
<th>address</th>
<th>object code</th>
<th>line no.</th>
<th>Source code</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001E</td>
<td>0E06</td>
<td>00010</td>
<td>movlw 0x06</td>
</tr>
<tr>
<td>000020</td>
<td>6E11</td>
<td>00011</td>
<td>movwf 0x11,A</td>
</tr>
<tr>
<td>000022</td>
<td>0E07</td>
<td>00012</td>
<td>movlw 0x07</td>
</tr>
<tr>
<td>000024</td>
<td>6E12</td>
<td>00013</td>
<td>movwf 0x12,A</td>
</tr>
<tr>
<td>000026</td>
<td>0E08</td>
<td>00014</td>
<td>movlw 0x08</td>
</tr>
<tr>
<td>000028</td>
<td>6E13</td>
<td>00015</td>
<td>movwf 0x13,A</td>
</tr>
<tr>
<td>00002A</td>
<td>0E05</td>
<td>00016</td>
<td>movlw 0x05</td>
</tr>
<tr>
<td>00002C</td>
<td>5E10</td>
<td>00017</td>
<td>subwf 0x10,F,A</td>
</tr>
<tr>
<td>00002E</td>
<td>5E11</td>
<td>00018</td>
<td>subwf x11,F,A</td>
</tr>
</tbody>
</table>
PIC18 memory organization

- Data memory and program memory are separated
  - This makes possible the simultaneous access of data and instruction.

- Data memory
  - General-purpose registers
    - Hold dynamic data
  - Special function registers
    - Registers used by CPU and peripheral modules
PIC18 memory access

Inside the µc chip

Program Memory Space
(a portion of this space is on the µc chip)

21-bit program address

Program counter (PC)

12-bit register address

PIC18 CPU

4096 registers

Data Memory Space
(Special function registers and general purpose RAM)

16-bit instruction bus

8-bit data bus

Address

Contents
After power-on, the PIC18 starts to execute instructions from address 0.

Up to 128KB (at present time) of program memory is inside the MCU chip.
PIC18 pipeline

<table>
<thead>
<tr>
<th>TCY0</th>
<th>TCY1</th>
<th>TCY2</th>
<th>TCY3</th>
<th>TCY4</th>
<th>TCY5</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVLW 55h</td>
<td>fetch 1</td>
<td>execute 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVWF PORTB</td>
<td></td>
<td>fetch 2</td>
<td>execute 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BRA sub_1</td>
<td>fetch 3</td>
<td>execute 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSF PORTA,BIT3</td>
<td>fetch 4</td>
<td>flush</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instruction @address sub_1</td>
<td>fetch sub_1</td>
<td>execute sub_1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: All instructions are single cycle, except for any program branches.

No data dependency hazard in PIC18 MCU because of 2-stage pipeline.
## Table 1.2 PIC18 CPU registers

<table>
<thead>
<tr>
<th>address</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFE</td>
<td>TOSU</td>
<td>Top of stack (upper)</td>
</tr>
<tr>
<td>0xFFE</td>
<td>TOSH</td>
<td>Top of stack (high)</td>
</tr>
<tr>
<td>0xFFD</td>
<td>TOSL</td>
<td>Top of stack (low)</td>
</tr>
<tr>
<td>0xFFC</td>
<td>STKPTR</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>0xFFB</td>
<td>PCLATU</td>
<td>Upper program counter latch</td>
</tr>
<tr>
<td>0xFFA</td>
<td>PCLATH</td>
<td>High program counter latch</td>
</tr>
<tr>
<td>0xFF9</td>
<td>PCL</td>
<td>Program counter low byte</td>
</tr>
<tr>
<td>0xFF8</td>
<td>TSPTRU</td>
<td>Table pointer upper byte</td>
</tr>
<tr>
<td>0xFF7</td>
<td>TSPTRH</td>
<td>Table pointer high byte</td>
</tr>
<tr>
<td>0xFF6</td>
<td>TSPTRL</td>
<td>Table pointer low byte</td>
</tr>
<tr>
<td>0xFF5</td>
<td>TABLAT</td>
<td>Table latch</td>
</tr>
<tr>
<td>0xFF4</td>
<td>PRC0H</td>
<td>High product register</td>
</tr>
<tr>
<td>0xFF3</td>
<td>PRC0L</td>
<td>Low product register</td>
</tr>
<tr>
<td>0xFF2</td>
<td>INTCON</td>
<td>Interrupt control register</td>
</tr>
<tr>
<td>0xFF1</td>
<td>INTCON1</td>
<td>Interrupt control register 2</td>
</tr>
<tr>
<td>0xFF0</td>
<td>INTCON3</td>
<td>Interrupt control register 3</td>
</tr>
<tr>
<td>0xFE9</td>
<td>INDF0 (4)</td>
<td>Indirect file register pointer 0</td>
</tr>
<tr>
<td>0xBE8</td>
<td>POSTINC0 (4)</td>
<td>Post increment pointer 0 (to GFRs)</td>
</tr>
<tr>
<td>0xBE7</td>
<td>POSTDEC0 (3)</td>
<td>Post decrement pointer 0 (to GFRs)</td>
</tr>
<tr>
<td>0xBE6</td>
<td>PEEINC0 (3)</td>
<td>Preincrement pointer 0 (to GPEs)</td>
</tr>
<tr>
<td>0xBE5</td>
<td>PLUSW0 (2)</td>
<td>Add WREG to PSR 0</td>
</tr>
<tr>
<td>0xBE4</td>
<td>TSR0H</td>
<td>File select register 0 high byte</td>
</tr>
<tr>
<td>0xBE3</td>
<td>TSR0L</td>
<td>File select register 0 low byte</td>
</tr>
<tr>
<td>0xBE2</td>
<td>WREG</td>
<td>Working register</td>
</tr>
<tr>
<td>0xBE1</td>
<td>INDF1 (4)</td>
<td>Indirect file register pointer 1</td>
</tr>
<tr>
<td>0xBE0</td>
<td>POSTINC1 (4)</td>
<td>Post increment pointer 1 (to GFRs)</td>
</tr>
<tr>
<td>0xBE5</td>
<td>POSTDEC1 (3)</td>
<td>Post decrement pointer 1 (to GFRs)</td>
</tr>
<tr>
<td>0xBE4</td>
<td>PEEINC1 (3)</td>
<td>Preincrement pointer 1 (to GPEs)</td>
</tr>
<tr>
<td>0xBE3</td>
<td>PLUSW1 (3)</td>
<td>Add WREG to PSR 1</td>
</tr>
<tr>
<td>0xBE2</td>
<td>TSR1H</td>
<td>File select register 1 high byte</td>
</tr>
<tr>
<td>0xBE1</td>
<td>TSR1L</td>
<td>File select register 1 low byte</td>
</tr>
<tr>
<td>0xBE0</td>
<td>BSF</td>
<td>Bank select register</td>
</tr>
<tr>
<td>0xFD8</td>
<td>INDF2 (4)</td>
<td>Indirect file register pointer 2</td>
</tr>
<tr>
<td>0xFD7</td>
<td>POSTINC2 (4)</td>
<td>Post increment pointer 2 (to GFRs)</td>
</tr>
<tr>
<td>0xFD6</td>
<td>POSTDEC2 (3)</td>
<td>Post decrement pointer 2 (to GFRs)</td>
</tr>
<tr>
<td>0xFD5</td>
<td>PEEINC2 (3)</td>
<td>Preincrement pointer 2 (to GPEs)</td>
</tr>
<tr>
<td>0xFD4</td>
<td>PLUSW2 (3)</td>
<td>Add WREG to PSR 2</td>
</tr>
<tr>
<td>0xFD3</td>
<td>TSR2H</td>
<td>File select register 2 high byte</td>
</tr>
<tr>
<td>0xFD2</td>
<td>TSR2L</td>
<td>File select register 2 low byte</td>
</tr>
<tr>
<td>0xFD1</td>
<td>STATUS</td>
<td>Status register</td>
</tr>
</tbody>
</table>

**Note 1. This is not a physical register**
BSR<3:0>

- 0000: Bank 0
- 0001: Bank 1
- 0010: Bank 2
- 0011: Bank 3
- 0100: Bank 4 to Bank 13
- 0110: Bank 14
- 0111: Bank 15
- 1110: Access RAM low
- 1111: Access RAM high

Access Bank

Note. 1. **BSR** is the 4-bit bank select register.

---

**16-bit instruction**

<table>
<thead>
<tr>
<th>15</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode</td>
<td>d</td>
<td>a</td>
<td>f</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Access bank

- **PIC18 banks**
  - 4096 registers are divided into 16 banks.
  - Only one bank is active at a time.

- **Bank switching**
  - When operating on a register in a different bank, bank switching is needed.
  - Bank switching incurs overhead and may cause program errors.

- **Access bank**
  - is created to minimize the problems of bank switching.
  - the lowest 96 bytes are general-purpose registers; the highest 160 bytes, special function registers.
  - When operands (f) are in the access bank, no bank switching is needed.
Data movement instructions

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>16-bit instruction word</th>
<th>Status affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>lfsr f, k</td>
<td>load FSR</td>
<td>1110 1110 00ff k\textsubscript{11}kkk</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1111 0000 k\textsubscript{7}kkk kkkk</td>
<td></td>
</tr>
<tr>
<td>movf f, d, a</td>
<td>Move f</td>
<td>0101 00da ffff ffff</td>
<td>Z, N</td>
</tr>
<tr>
<td>movff fs, fd</td>
<td>Move fs (source) to f</td>
<td>1100 ffff ffff ffff</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1111 ffff ffff ffff</td>
<td></td>
</tr>
<tr>
<td>movwf, f,a</td>
<td>Move WREG to f</td>
<td>0110 111a ffff ffff</td>
<td>None</td>
</tr>
<tr>
<td>swapf f, d, a</td>
<td>Swapp nibbles in f</td>
<td>0011 10da ffff ffff</td>
<td>None</td>
</tr>
<tr>
<td>movlb k</td>
<td>Move literal to BSR&lt;3:0&gt;</td>
<td>0000 0001 kkkk kkkk</td>
<td>None</td>
</tr>
<tr>
<td>movlw k</td>
<td>Move literal to WREG</td>
<td>0000 1110 kkkk kkkk</td>
<td>None</td>
</tr>
</tbody>
</table>

Note. Both LFSR f, k and MVFF fs, fd are 32 bit instructions
**movff**

- a 32-bit instruction
- Copy a file register in one bank to a file register in another bank **without referring to the BSR register**
- **Example**
  - `movff 0x100,0x300 ; copy data register 0x100 to data register 0x300`

```
movff   fs, fd   Move fs (source) to f
           1100 ffff  ffff  ffff
           1111 ffff  ffff  ffff
```
Instruction format: Byte-to-byte operations

- movff fs, fd

\[
\begin{array}{ccc}
15 & 12 & 11 & 0 \\
\hline
\text{opcode} & f \text{ (source file register)} \\
15 & 12 & 11 & 0 \\
\hline
1111 & f \text{ (destination file register)} \\
\end{array}
\]

f = 12-bit file register address
movwf

- a 16-bit instruction
- Copy a working register (WREG) to a file register in the force access bank or the selected bank
- Example
  - movwf 0x30,A ; copy WREG to a register in access bank

```
movwf, f,a       Move WREG to f  0110 111a ffff ffff
```

*WREG: A special register that is involved in the execution of many instructions*
Instruction format: byte-oriented operations

- **movwf f,a**

```
<table>
<thead>
<tr>
<th>15</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode</td>
<td>d</td>
<td>a</td>
<td></td>
<td></td>
<td>f</td>
</tr>
</tbody>
</table>
```

d = 0 for result destination to be WREG register.
d = 1 for result destination to be file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address
movf

- a 16-bit instruction
- Copy a file register to the WREG register
- Examples
  - movf 0x20,W,A
    - Copy register 0x20 in access bank to WREG
  - movf 0x20,W,BANKED
    - Copy register 0x20 in the specified by the BSR register to WREG

```
movf f, d, a  Move f  0101 00da ffff ffff    Z, N
```
Status register

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>N</td>
<td>OV</td>
<td>Z</td>
<td>DC</td>
<td>C</td>
</tr>
</tbody>
</table>

N: Negative bit
1 = arithmetic result is negative
0 = arithmetic result is positive

OV: Overflow bit
1 = Overflow occurred for signed arithmetic
0 = No overflow occurred

Z: Zero flag
1 = The result of an arithmetic or logic operation is zero.
0 = The result of an arithmetic or logic operation is not zero.

DC: Digit carry/borrow bit
For ADDWF, ADDLW, SUBLW, SUBWF instructions.
1 = A carry-out from the 4th low-order bit of the result occurred.
0 = No carry-out from the 4th low-order bit of the result occurred.
For borrow, the polarity is reversed. For rotate (RRF, RLF) instructions, this bit is loaded with either the bit 4 or bit 3 of the source register.

C: Carry/borrow bit
For ADDWF, ADDLW, SUBLW, SUBWF instructions.
1 = A carry-out from the most significant bit of the result occurred.
0 = No carry-out from the most significant bit of the result has occurred.
For borrow, the polarity is reversed. For rotate (RRF, RLF) instructions, this bit is loaded with either the high or low order bit of the source register.
**movlw/movlb**

- 16-bit instructions
- Copy a value to the WREG register/ set bank active
- Examples
  - `movlw 0x10 ; WREG ← 0x10`
  - `movb 3 ; load 3 into BSR (set bank 3 active)`

<table>
<thead>
<tr>
<th>movlb</th>
<th>k</th>
<th>Move literal to BSR&lt;3:0&gt;</th>
<th>0000 0001 kkkk kkkk</th>
</tr>
</thead>
<tbody>
<tr>
<td>movlw</td>
<td>k</td>
<td>Move literal to WREG</td>
<td>0000 1110 kkkk kkkk</td>
</tr>
</tbody>
</table>
Instruction format: literal operations

- `movlb k`
- `movlw k`

\[
\begin{array}{cccc}
15 & 8 & 7 & 0 \\
\hline
\text{opcode} & k
\end{array}
\]

\( k = 8\text{-bit immediate value} \)
**swapf**

- a 16-bit instruction
- Swap nibbles
- Example
  - swapf 0x30,F,A
    - swap the upper and lower 4 bits of the register 0x30

```
swapf f, d, a    Swapp nibbles in f    0011 10da fff  fff
```
Ifstr

- a 32-bit instruction
- Load 12-bit value in the FSR (file select register)
- 3 FSR (FSR0, FSR1, FSR2), each FSR has low byte and high byte

**Example**
- Ifsr FSR1,0xB00 ; place the value 0xB00 in FSR1

```
<table>
<thead>
<tr>
<th>Ifsr f, k</th>
<th>load FSR</th>
<th>1110 1110 00ff k11kkk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1111 0000 k7kkk kkkk</td>
</tr>
</tbody>
</table>
```
ADD instruction: addwf

- Add WREG and f
  - addwf f,d,a

- Examples
  - addwf 0x20,W,A
    - add data register in access bank and WREG and place sum in WREG
  - addwf 0x20,F,A
ADD instruction: addwfc

- Add WREG, carry bit and f
  - addwfc f,d,a

- Examples
  - addwfc sum,F,A
    - add WREG, the register with the name sum, and carry and leave the result in sum
  - addwfc sum,W,A
ADD instruction: addlw

- Add literal and WREG
  - `addlw k`
- Example
  - `addlw 0x5 ; increment WREG by 5`
Example: write an instruction sequence to increment the contents of three registers 0x30-0x32 by 3

```
movlw 0x3    ; WREG ← 0x3
addwf 0x30, F, A
addwf 0x31, F, A
addwf 0x32, F, A
```
Example: write an instruction sequence to add the contents of three registers 0x40-0x42 and store the sum at 0x50

```
    movf 0x40, W, A ; WREG ← [0x40]
    addwf 0x41, W, A
    addwf 0x42, W, A
    movwf 0x50, A ; [0x50] ← WREG
```
Examples

- **addwf 0x20,F,A**
  - add the register at 0x20 in access bank with WREG and store the sum in 0x20.

- **addwf 0x20,F,BANKED**
  - add the register 0x20 in the bank specified by the BSR register and store the sum in 0x20.
SUB instruction: subwf

- Subtract WREG from f
  - subwf f,d,a

- Examples
  - subwf 0x50, W, A ; WREG ← [0x50] – [WREG]
  - subwf 0x30, F, A ; 0x30 ← [0x30] – [WREG]
ADD instructions: subfwb, subwfb

- **Subfwb**  \( f, d, a \); subtract \( f \) from WREG with borrow
- **Subwfb**  \( f, d, a \); subtract WREG from \( f \) with borrow

**Examples**

- `subfwb 0x30,W,A`; WREG ← [WREG] − [0x20] − borrow flag
- `subwfb 0x10,F,A`; 0x10 ← [0x10] − [WREG] − borrow flag
SUB instruction: sublw

- Subtract WREG from literal
  - sublw \( k \)
- Example
  - sublw 0x10 ; WREG ← 0x10 - [WREG]
Example: write an instruction sequence to subtract 9 from the registers 0x50-0x53

```assembly
movlw 0x09 ; WREG ← 0x9
subwf 0x50, F, A
subwf 0x51, F, A
subwf 0x52, F, A
subwf 0x53, F, A
```
PIC18 addressing modes

- Register direct
- Immediate mode
- Inherent mode
- Indirect mode
- Bit-direct addressing mode
Register direct

- Use an 8-bit value to specify a data register as operand.
  - movwf 0x45,A
  - movwf 0x1A,BANKED
  - movff 0x120,0x220
Immediate mode

- A value in the instruction to be used as an operand.
  - `addlw 0x20` ; add the hex value 0x20 to WREG
  - `movlw 0x15` ; load 0x15 into WREG
  - `movlb 3` ; place decimal value 3 to the lower 4 bits of
    ; the BSR register
Inherent mode

- The operand is the opcode
  - `andlw 0x3C` ; the operand WREG is implied
  - `movlw 0x20` ; the operand WREG is implied
Indirect mode

- A special function register is used as a pointer to the data memory location (actual data register) based on file select registers (FSRn, n = 0, 1, 2)
  - Indirect file register pointers: INDFn
    - Do nothing to FSRn after indirect access
  - Postdecrement register: POSTDECN
    - Auto-decrement FSRn after an indirect access
  - Postincrement register: POSTINCn
  - Preincrement register: PREINCn
    - Auto-increment FSRn before an indirect access
  - Use the value of WREG as an offset to FSRn: PLUSWN
    - FSRn is not modified after the access
Indirect addressing through FSR register

- **File select registers**
  - FSR0, FSR1, FSR2
  - 12 bits for addressing the entire memory (4096 bytes)
Examples

- **movwf INDF0**
  - copy the contents of WREG to the memory specified by the FSR0

- **movwf POSTDEC0**
  - The same operation as the above but the content of FSR0 is decremented by 1 after the operation.

- **movwf PREINC0**
  - First increments FSR0 by 1 and then copy the contents of WREG to the memory specified by the FSR0

- **clrf PLUSW0**
  - Clear the memory at the address equal to the value of WREG and that of FSR0
Examples (cont.)

- `movf POSTDEC0,W`
  - Copy the memory specified by the FSR0 to the WREG.
  - The content of FSR0 is incremented by 1 after the operation.

- `movff POSTINC0,PRODL`
  - Similar operation as the above but data is copied to the register with the name PRODL

- `movff PLUSW2,PRODL`
  - Copy the memory at the address (= WREG + FSR2) to PRODL

- `addwf PREINC1,F`
  - FSR1 is incremented by 1
  - Add the content of memory by FSR1 with that of WREG.
  - The sum leave the memory specified by FSR1
Bit-direct addressing mode

- **Deal with individual bit**

  ![Bit-oriented file register operations](redraw.png)

  - b = 3-bit position of bit in the file register (f).
  - a = 0 to force Access Bank
  - a = 1 for BSR to select bank
  - f = 8-bit file register address

  BCF PORTB,3, A  ; pull the port B pin RB3 to low (clear)
  BSF PORTA,4,A  ; pull the port A pin RA4 to high (set)
Instruction format of control operations

- Change the program execution sequence

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Immediate Value</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>n&lt;19:8&gt; (literal)</td>
<td>GOTO label</td>
</tr>
<tr>
<td>1111</td>
<td>n&lt;19:8&gt; (literal)</td>
<td>CALL funct_name</td>
</tr>
<tr>
<td>15 8 7 0</td>
<td>n&lt;7:0&gt; (literal)</td>
<td></td>
</tr>
</tbody>
</table>

- n = 20-bit immediate value

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Fast Bit</th>
<th>Immediate Value</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>S</td>
<td>n&lt;19:8&gt; (literal)</td>
<td>GOTO label</td>
</tr>
<tr>
<td>1111</td>
<td>S</td>
<td>n&lt;19:8&gt; (literal)</td>
<td>CALL funct_name</td>
</tr>
<tr>
<td>15 8 7 0</td>
<td>S</td>
<td>n&lt;7:0&gt; (literal)</td>
<td></td>
</tr>
</tbody>
</table>

- S = fast bit

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Immediate Value</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 11 10</td>
<td>n&lt;10:0&gt; (literal)</td>
<td>BRA label</td>
</tr>
<tr>
<td>15 8 7 0</td>
<td>n&lt;7:0&gt; (literal)</td>
<td>BC label</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Immediate Value</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 8 7 0</td>
<td>n&lt;7:0&gt; (literal)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1.12 Control operations (redraw with permission of Microchip)
RISC vs. CISC

- Reduced instruction set computer
  - Simple instruction set
  - Regular and fixed instruction format
  - Simple address modes
  - Pipelined instruction execution
  - Most operations are register to register
  - Take shorter time to design and debug

- Complex instruction set computer
  - Complex instruction set
  - Irregular instruction format
  - Complex address modes
  - May also pipeline instruction execution
  - Most operations can be register to memory
  - Take longer time to design and debug
Reference

- Wikipedia