EE 3170 Microcontroller Applications

Lecture 6: Instruction Subset & Machine Language:
Addressing Modes in Motorola 68HC11
- Miller §2.1 & §2.2 & §2.3

Based on slides for ECE3170 by Profs. Davis, Kieckhafer, Tan, and Cischke

Lecture Overview

- More Instructions
- Example: first assembly language program
- Addressing Modes
  - Introduction, definitions, and notation
- The 68HC11 Addressing Modes
  - definition
  - address calculation
  - advantages and disadvantages
  - common uses

A Few More Instructions

- Instruction Set Table:
  - Explains operation of instruction
  - Presents additional details
  - There will be more columns later

<table>
<thead>
<tr>
<th>Source Form</th>
<th>Operation</th>
<th>Boolean Expression</th>
<th>Machine Code</th>
<th>Operands</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDA</td>
<td>Add Memory to A</td>
<td>A+M \rightarrow A</td>
<td>BB hh ll</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>LDAA</td>
<td>Load Accumulator A</td>
<td>M \rightarrow A</td>
<td>B6 hh ll</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>STAA</td>
<td>Store Accumulator A</td>
<td>A \rightarrow M</td>
<td>B7 hh ll</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>STOP</td>
<td>Stop program</td>
<td></td>
<td>3F</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2-15: Part of the instruction set table

Instruction Format

- From Reference Manual
- Each ONE of the TEN encodings is a different format (or addressing mode) for the same LDA instruction
- Only concern ourselves with one at a time

Example - First Program

- "C" Version: SUM = N1 + N2
  - SUM, N1, & N2 are variables
  - C compiler assigns a mem address to each variable

- Assembly language version:
  - Programmer assigns addresses to variables, e.g.
    - N1 ⇒ C20B
    - N2 ⇒ C20C
    - SUM ⇒ C20D
  - At some point programmer must initialize N1 & N2

Example - First Program

- "SUM = N1 + N2" in Assembler Language (AL)
  - ldaa C20B * Load N1 into A
  - adda C20C * Add N2 to A
  - staa C20D * Store A in SUM
  - stop * Halt the program

- Assembler program translates AL code into ML code
  - B6C20B, B8C20C, B7C20D, 3F

- Loader program loads ML "object code" into mem

Loaded Object Program and Data

- Idaa C20B
  - LDAA
- adda C20C
  - ADDA
- staa C20D
  - STAA
- stop

How Would We Extend This Program to Add Three Numbers?

- What would stay the same?
- What would change?
Considerations for Third Number

- We need another `adda` which is a 3-byte instruction.
- That means stop will be in C20C.
- N1, N2, N3 and SUM will follow.

Program to Add Three Numbers

- Program
  - `ldaa C20E`
  - `adda C20F`
  - `adda C210`
  - `stop`

- Memory
  - C200
  - C201
  - C202
  - C203
  - C204
  - C205
  - C206
  - C207
  - C208
  - C209
  - C20A
  - C20B
  - C20C
  - C20D
  - C20E
  - C20F
  - C210
  - C211

Programming Reference Guide

Available at: http://www.freescale.com/files/microcontrollers/doc/ref_manual/M68HC11ERG.pdf
How Can We Address Memory? -Addressing Modes

- So far, all data addresses have been embedded in their entirety within the instruction.

- This approach can have disadvantages:
  - It takes a lot of space in the instruction,
    - 68HC11: 64KB address space ⇒ 2 bytes / address
    - Pentium et al: 4GB address space ⇒ 4 bytes / address
  - It takes extra clock ticks to fetch the whole address
  - May need to calculate or modify address at run-time
    - an array entry
    - a linked list pointer

- We need a shorter, faster way to address memory.

Definitions

- EA = Effective Address = the complete 16-bit address of a data item in memory

- Addressing Mode = a method the processor uses to calculate an effective address, including:
  - The information used in the calculation
  - The algorithm used in the calculation

Notation

- $hh$ = high-order byte of an address
- $ll$ = low-order byte of an address
- $dd$ = one byte of data
- $M$ = a one-byte memory word
- ":" = concatenation:
  - $hh:ll$ = full 16-bit address
  - $00:value$ = unsigned byte zero-extended to 16-bits
  - $ss:value$ = signed byte sign-extended to 16 bits
  - $M:(M-1)$ = 16-bit concat. of two adjacent mem. bytes

68HC11 Addressing Modes

- 6 Basic Addressing Modes:
  - Some minor variations to be introduced later

- Mode Names:
  - Extended Addressing
  - Direct Addressing
  - Inherent Addressing
  - Immediate Addressing
  - Indexed Addressing
  - Relative Addressing
Addressing modes in a nutshell

- **Extended Addressing**
  - $EA = hh:ll$

- **Direct Addressing**
  - $EA = 00:ll$

- **Inherent Addressing**
  - n/a: no memory access – uses registers

- **Immediate Addressing**
  - n/a: no memory access – data contained in machine code

- **Indexed Addressing**
  - $EA = index + 00 : offset$

- **Relative Addressing**
  - $EA = PC + ss : offset$

### 1. Extended Addressing

- **Definition:**
  - The complete address is contained in the instruction
  - Exactly what we have been looking at

- **Instruction:** opcode $hh ll$

- **EA Calculation:** $EA = hh:ll$

- **Range:** 0000 … FFFF
  - Entire Address Space

### 1. Extended Addressing

- **Advantages:**
  - can access the entire address space

- **Disadvantages:**
  - Requires two bytes in the instruction for address
  - Requires two clock ticks to fetch the address
  - Address of the data must be known at assembly time

- **Uses:**
  - Scalar Variables

---

**Extended Addressing Example**

**ADD**

- **Operation:** $ACCX = (ACCX) + (M)$
- **Description:** Adds the contents of $M$ into the contents of ACCX and places the result in ACCX. This instruction affects the $H$, $N$, and $V$ condition code bits, depending on the result of the subtraction.

- | $S$ | $X$ | $Y$ | $M$ |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>X3</td>
<td>M3</td>
<td>M3 + R3</td>
</tr>
<tr>
<td>N</td>
<td>RF</td>
<td>PS</td>
<td>PS + R3</td>
</tr>
<tr>
<td>V</td>
<td>M7</td>
<td>MT</td>
<td>MT + RF</td>
</tr>
</tbody>
</table>

- **Condition Codes and Boolean Formulae:**
  - $S$: Sign of result
  - $X$: Overflow
  - $Y$: Carry in
  - $M$: Memory bit

Extended Addressing Example

Addressing Modes, Machine Code, and Cycle-by-Cycle Execution:

2. Direct Addressing

- **Definition:**
  - Low-order byte of the address is in the instruction
  - High-order byte is set to zero

- **Instruction:** Opcode \( l \)

- **EA Calculation:** \( EA = 00 : l \)

- **Range:** 0000 ... 00FF
  - Lower 256 bytes of memory

- **Advantages:** Relative to extended
  - Saves one byte in the instruction
  - Saves one clock tick in the Fetch Phase

- **Disadvantages:**
  - Range limited to lower 256 bytes of memory

- **Uses:**
  - Fast access to frequently used scalar variables
3. Inherent Addressing

- **Definition:**
  - All relevant data is already in registers, not memory
  - 68HC11 registers are all special purpose
  - Register I.D. is “inherent” in the Opcode

- **Instruction: Opcode**
  - e.g. ABA adds accumulator B to accumulator A
  - i.e. $A + B \rightarrow A$

- **EA Calculation:** there is none

- **Range:** Not Applicable
3. Inherent Addressing

- **Advantages:**
  - relative to extended
  - saves two bytes in the instruction
  - saves two clock ticks in the Fetch Phase

- **Disadvantages:**
  - there are very few registers

- **Uses:**
  - Fast ops on data previously loaded into registers
  - Instructions which do not need data (e.g. “STOP”).

---

### Inherent Addressing Example

Source Forms: `INCA, INCB, INC (spr)`

<table>
<thead>
<tr>
<th>Cycle</th>
<th>ADDR (INR)</th>
<th>ADDR (INR)</th>
<th>ADDR (INR)</th>
<th>ADDR (INR)</th>
<th>ADDR (INR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OP 4C 1</td>
<td>OP 5C 1</td>
<td>OP 7C 1</td>
<td>OP 6C 1</td>
<td>OP 16 1</td>
</tr>
<tr>
<td>2</td>
<td>OP+1 - 1</td>
<td>OP+1 - 1</td>
<td>OP+1 - 1</td>
<td>OP+1 - 1</td>
<td>OP+1 - 1</td>
</tr>
<tr>
<td>3</td>
<td>OP+2 H</td>
<td>OP+2 H</td>
<td>OP+2 H</td>
<td>OP+2 H</td>
<td>OP+2 H</td>
</tr>
<tr>
<td>4</td>
<td>MOV (INR)</td>
<td>MOV (INR)</td>
<td>MOV (INR)</td>
<td>MOV (INR)</td>
<td>MOV (INR)</td>
</tr>
<tr>
<td>5</td>
<td>FFFF - 1</td>
<td>FFFF - 1</td>
<td>FFFF - 1</td>
<td>FFFF - 1</td>
<td>FFFF - 1</td>
</tr>
<tr>
<td>6</td>
<td>MOV result 0</td>
<td>MOV result 0</td>
<td>MOV result 0</td>
<td>MOV result 0</td>
<td>MOV result 0</td>
</tr>
<tr>
<td>7</td>
<td>result 0</td>
<td>result 0</td>
<td>result 0</td>
<td>result 0</td>
<td>result 0</td>
</tr>
</tbody>
</table>

### Inherent Addressing Example

**Definition:**
- The data is contained in the instruction
- Limited to 1 byte data

**Instruction:** `Opcode dd`
- where `dd` = the actual data value (not an address)

**EA Calculation:** there is none

**Range:** Not Applicable

---

4. Immediate Addressing

- **Definition:**
  - the data is contained in the instruction
  - Limited to 1 byte data

- **Instruction:** `Opcode dd`
  - where `dd` = the actual data value (not an address)

- **EA Calculation:** there is none

- **Range:** Not Applicable
4. Immediate Addressing

- **Advantages**: relative to extended
  - saves the extra operation of fetching the data after an address is fetched or calculated
  - saves the memory required to store an address

- **Disadvantages**: data value must be known at assembly time
  - data value can not be changed by the program

- **Uses**:
  - constants needed by the program
  - e.g. length of a loop

Immediate Addressing Example

Immediate Addressing Example

5. Indexed Addressing

- **Definition**: The effective address is the sum of
  - an “index” field contained in an index register
  - an “offset” field contained in the instruction

- **Instruction**: OPCODE offset

- **EA Calculation**: \( EA = index + 00 : offset \)
  - 68HC11 has two 16-bit index registers (X and Y)
  - 68HC11 uses an 8-bit offset

- **Range**: (Val. in index reg.) + (00 ... FF)
  - i.e. (X+0) ... (X+255) or (Y+0) ... (Y+255)
5. Indexed Addressing

- **Advantages**: relative to extended
  - address need not be known at assembly time
  - supports loop and array operations
  - The effect of altering index reg. is to alter the EA of the instruction
  - The instruction code including the offset byte remains unchanged.

- **Disadvantages**: addition needed to calculate the address

- **Uses**:
  - Arrays
  - Incrementing addresses inside a loop

### Indexed Addressing Example

<table>
<thead>
<tr>
<th>Cycles</th>
<th>ADEA (INX)</th>
<th>ADEA (INY)</th>
<th>ADEA (INX) (X)</th>
<th>ADEA (INY) (Y)</th>
<th>ADEA (INX) (X) (Y)</th>
<th>ADEA (INY) (X) (Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CP</td>
<td>CB</td>
<td>1</td>
<td>0001</td>
<td>CP+1</td>
<td>CB+1</td>
</tr>
<tr>
<td>2</td>
<td>CP+1</td>
<td>CB</td>
<td>1</td>
<td>0001</td>
<td>CP+2</td>
<td>CB+2</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>CB</td>
<td>1</td>
<td>0001</td>
<td>CP+1</td>
<td>CB+1</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>CB</td>
<td>1</td>
<td>0001</td>
<td>CP+2</td>
<td>CB+2</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>CB</td>
<td>1</td>
<td>0001</td>
<td>CP+1</td>
<td>CB+1</td>
</tr>
</tbody>
</table>

### Indexed Addressing Example

A Few Indexed Addressing Instructions

- **Indexed Addressing Support Instructions**
  - **LDX** = Load X register  ⇒  M:(M+1)  →  X
  - **LDY** = Load Y register  ⇒  M:(M+1)  →  Y
  - **STX** = Store X register  ⇒  X  →  M:(M+1)
  - **STY** = Store Y register  ⇒  Y  →  M:(M+1)
  - **INX** = Increment X register  ⇒  X+1  →  X
  - **INY** = Increment Y register  ⇒  Y+1  →  Y
  - **DEX** = Decrement X register  ⇒  X-1  →  X
  - **DEY** = Decrement Y register  ⇒  Y-1  →  Y

### Indexed Addressing Example

<table>
<thead>
<tr>
<th>ADD</th>
<th>ADD without Carry</th>
<th>ADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation: ACX = (ACX) + (M)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description: Adds the contents of M to the contents of ACX and places the result in ACX. This instruction affects the H and N condition code bits. It is suitable for use in GCC arithmetic operations (see DA instruction for additional information).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Condition Codes and Boolean Formulas:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>X</td>
<td>H</td>
</tr>
<tr>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>H</td>
<td>X+3 + M3 + M3 · R3 + R3 · X3</td>
<td>Set if there was a carry from bit 3; cleared otherwise.</td>
</tr>
<tr>
<td>N</td>
<td>R7</td>
<td>Set if N or V is set; cleared otherwise.</td>
</tr>
<tr>
<td>Z</td>
<td>R7 + R7 + R7 + R7 + R7 + R7 + R7</td>
<td>Set if result is 0; cleared otherwise.</td>
</tr>
<tr>
<td>V</td>
<td>X7 + M7 + R7 + R7 + R7 + R7</td>
<td>Set if a true complement overflow resulted from the operation; cleared otherwise.</td>
</tr>
<tr>
<td>C</td>
<td>X7 + M7 + M7 + R7 + R7 + X7</td>
<td>Set if there was a carry from the MSB of the result; cleared otherwise.</td>
</tr>
<tr>
<td>Source Form:</td>
<td>ADDA (opr), ADDS (opr)</td>
<td></td>
</tr>
</tbody>
</table>
6. Relative Addressing

- **Definition**: A variation on indexed addressing
  - The PC is used as the index register
  - the “offset” field in the instruction is a signed value

- **Instruction**: Opcode offset

- **EA Calculation**: \( EA = PC + ss : offset \)

- **Range**: (Value in PC) + [-7F … (+7F-1)]
  - i.e. (PC -128) … (PC + 127)

---

**Advantages**: relative to extended
- full address need not be known at assembly time
- if the code is “relocated” to a different starting address (PC reg.) then the data address moves with it
- EA is formed during the execute phase of the instruction

**Disadvantages**:
- the extra addition needed to calculate the address
- the small range relative to current value of PC

**Uses**:
- Branch Instructions

---

**Relative Addressing Example**

**BRA**

- **Operation**: \( PC \leftarrow (PC) + 0 \times Rel \)
- **Description**: Unconditional branch to the address given by the following formula, in which \( Rel \) is the relative offset stored as a two-complement number in the second byte of machine code corresponding to the branch instruction.
  - The source program specifies the destination of any branch instruction by its absolute address, either as a numerical value or as a symbolic or expression, that can be numerically evaluated by the assembler. The assembler obtains the relative address, \( Rel \), from the absolute address and the current value of the location counter.

- **Condition Codes and Boolean**
  - N: +1, H: 0
  - None affected

- **Source Form**: BRA (re)

**Addressing Modes, Machine Code, and Cycle-by-Cycle Execution**

- **Cycle**: 0 1 2 3 4
- **op**: 00 01 10 11
- **imm**: 0 1

---

**Relative Addressing Operations**

- **Address in PC is unsigned number**
  - for EA calculation we can think of it as signed

- **Relative offset in instruction is 2’s complement signed number**

- **Sign extend relative offset and add to PC contents**

- **Interpret result as unsigned number (effective address)**
One More Complication (Skipped)

- Program counter is incremented during fetch.
  - => points to address of next instruction in sequence
  - Since branch instruction is two bytes long, this next instruction is 2 after BR.
- Branch is executed during execute.
  - =>branches from incremented PC
  - =>branching back 2 gives infinite loop.
  - Why?

Addressing Mode Summary

- 68HC11 Has Six Addressing Modes
  - Could be logically grouped into three classes
- 1. “Concatenated-Address” Modes
  - Extended Addressing:
    - Instruction: Opcode \( hh \ ll \)
    - \( EA = hh:ll \)
  - Direct Addressing:
    - Instruction: Opcode \( ll \)
    - \( EA = 00:ll \)

Addressing Mode Summary

- 2. “No-Address” Modes:
  - Inherent Addressing:
    - Instruction: Opcode
    - \( EA = \text{none} \) (data is already in registers)
  - Immediate Addressing:
    - Instruction: Opcode \( dd \)
    - \( EA = \text{none} \) (data value is in the \( dd \) byte of instruction)

Addressing Mode Summary

- 3. “Summed-Address” Modes:
  - Indexed Addressing:
    - Instruction: Opcode \( offset \)
    - \( EA = (X + 00:offset) \) or \( (Y + 00:offset) \)
  - Relative Addressing:
    - Instruction: Opcode \( offset \)
    - \( EA = PC + ss:offset \)
Addressing Mode Summary

- The addressing mode used in an instruction is encoded into the Opcode

- e.g. ldaa has several address mode options
  - each option has a different ML opcode, e.g.:
    - 86 ⇒ Use Immediate mode
    - 96 ⇒ Use Direct Mode
    - B6 ⇒ Use Extended mode