EE 3170 Microcontroller Applications

Lecture 7 : Instruction Subset & Machine Language:
Conditions & Branches in Motorola 68HC11
- Miller §2.2 & §2.3 & §2.4

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

Lecture Overview

- IF-THEN vs. Branching
- Branch Condition
  - 68HC11 Condition Codes
  - Altering Condition Codes
- Branch Instructions
  - Operations
  - Addressing
  - Types of Branch Instructions

Conditional/Decision in Microprocessors

- Conditional Statements are ESSENTIAL in microprocessors
- Conditional Statements equate to decision boxes in a flowchart
- Conditional Statements equate to if-then in high-level languages
- Conditional Statements equate to CONDITIONAL BRANCHES in assembly language

If-Then vs. Branching

- High-Level Language
  - If (some logical condition is True)
  - Then (do something)
  - Else (do something else)
- Assembly language equivalent
  - Set a “branch” condition
    - If (the branch condition is True)
    - Then (reload PC with address of a “target” instruction)
    - Else (don’t alter PC - fetch the “fall-through” instr.)
Branch Conditions

- A branch condition is very simple
- A logical condition of the result of an instruction, e.g.
  - result = 0
  - result is negative
  - result produces a carry-out
  - result produced an overflow
- These conditions must be remembered after an instruction is completed
  - so the branch instruction can examine it

 Condition Codes

- Branch conditions kept in a “condition code” register
- Each “Condition Code” (CC) is one bit of the register
  - Each CC bit represents one True/False condition
  - CC bits are set at the end of each instruction
- A branch instruction
  - examines one or more CCs
  - decides what to do with PC based on CC contents

68HC11 Condition Code Register

<table>
<thead>
<tr>
<th>C</th>
<th>Carry-out bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Previous instruction had a carry-out</td>
</tr>
<tr>
<td>1</td>
<td>Previous instruction didn’t have a carry-out</td>
</tr>
</tbody>
</table>

- Meaning:
  - C=1: Previous instruction had a carry-out
  - C=0: Previous instruction didn’t have a carry-out

- Usage:
  - C=1 ⇒ Overflow for unsigned arithmetic only
  - Ignore C bit for signed arithmetic

- Relevant Branch opcodes:
  - BCC: Branch if Carry Clear (C=0)
  - BCS: Branch if Carry Set (C=1)
### Condition Code Meanings

**V = Overflow bit (2’s complement)**
- **Meaning:**
  - V=1: Previous instruction had an overflow
  - V=0: Previous instruction didn’t have an overflow
- **Usage:**
  - V=1 ⇒ Overflow for **signed** arithmetic only
  - Ignore V bit for **unsigned** arithmetic
- **Relevant Branch opcodes:**
  - BVC: Branch if Overflow Clear (V=0)
  - BVS: Branch if Overflow Set (V=1)

**Z = Zero bit**
- **Meaning:**
  - Z=1: Previous instruction result = 0
  - Z=0: Previous instruction result ≠ 0
  - **watch these; they are counter-intuitive**
- **Usage:**
  - Z=1 ⇒ the zero condition is true
  - all bits of the result are zeros
- **Relevant Branch opcodes:**
  - BNE: Branch if result not equal to zero (Z=0)
  - BEQ: Branch if result is equal to zero (Z=1)

**N = Negative bit**
- **Meaning:**
  - N=1: Previous instruction result < 0
  - N=0: Previous instruction result ≥ 0
- **Usage:**
  - N=1 ⇒ Left-hand bit = 1 (sign-bit)
  - Evaluates 2’s complement negative
- **Relevant Branch opcodes:**
  - BPL: Branch if result sign is **plus** (N=0) actually branches if result is **not-negative**
  - BMI: Branch if result sign is **minus** (N=1)

**S, X, & I bits**
- Are all related to hardware “interrupts”
- Allow I/O devices to interrupt the current program
- Are not used for branch instructions
- Will see a lot more later

**H = half-carry bit**
- Status of carry out of bit 3 of the result
- Function is similar to C bit,
- Useful for carry between BCD digits
- will probably not need it in this course
Which Instructions Affect CCs?

- Most of them affect one or more condition code bits.
- See Motorola web documentation.
- Common Notation--
  - CC isn’t altered by this instruction
  - 0 CC is cleared to zero
  - 1 CC is set to one
  - Δ CC is set or cleared depending on instruction

A Few Idiosyncracies

- Not all arithmetic and logic instructions affect all CCs
- Load & store instructions alter some CCs, e.g. ld aa:
  - N & Z = Δ
  - V = 0
  - C = -
- Double-precision instructions (using D, X, & Y regs)
  - may use different criteria than single-precision instr
- Bottom line: Always check the table to be sure.
Branch Instruction Operation

- Notation for branch actions:
  - IF branch condition is true, then
    - Branch is “taken”
    - “target” instruction is executed
    - target instruction address is specified in the branch instr.
  - IF branch condition is false, then
    - Branch is “not taken”
    - “fall-through” instruction is executed
    - fall-through instruction is the next sequential instruction after the branch
- Notice: None of branch instructions affect any of CC bits

Branch Instruction Operation

- Branch instructions use “relative” addressing
  - Branch instruction contains 8-bit signed offset
  - Target address = PC + ss: offset
  - Note: PC has already been incremented before “add”
    - PC has address of Opcode of the fall-through instruction
    - When calculating your offsets, adjust accordingly.

Offset Field Example

| AC00 | | AC00 |
| AC01 | | AC01 |
| AC02 | | AC02 |
| AC03 | | AC03 |
| AC04 | | AC04 |
| AC05 | | AC05 |
| AC06 | | AC06 |
| AC07 | | AC07 |
| AC08 | | AC08 |
| AC09 | | AC09 |

Simple Branch Instructions

- Each of the following tests exactly 1 CC bit
  - BCC: Branch on Carry Clear : C=0
  - BCS: Branch on Carry Set : C=1
  - BVC: Branch on Overflow Clear : C=0
  - BVS: Branch on Overflow Set : C=1
  - BNE: Branch on Not Equal : Z = 0
  - BEQ: Branch on Equal : Z=1
  - BPL: Branch on Positive/Plus : N=0
  - BMI: Branch on Negative/Minus : N=1
More Complex Branch Instructions

- Each test more than one CC bit
  - BGE: Branch Greater than or Equal to zero: N \or V = 0
  - BGT: Branch Greater Than zero: Z \or (N \or V) = 0
  - BHI: Branch if Higher: C \or Z = 0
  - BLE: Branch if Less than or Equal to zero: Z \or (N \or V) = 0
  - BLO: Branch if Lower: C = 1
  - BLS: Branch if Lower or Same: C \or Z = 1
  - BLT: Branch if Less than Zero: N \or V = 1
  (refer to Appendix A: p506)

- Mnemonic written to be logically preceded by either a “compare” or “subtract” instruction

Comparison Instructions

- Subtraction:
  - does a subtraction
  - alters the CCs
  - stores the result

- Comparison
  - does a subtraction
  - alters the CCs

- Comparison instructions alter CCs without changing any data values

All Branches

<table>
<thead>
<tr>
<th>Test</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>Z = 0</td>
</tr>
<tr>
<td>Neg</td>
<td>N</td>
</tr>
<tr>
<td>Carry</td>
<td>C</td>
</tr>
<tr>
<td>Overflow</td>
<td>V</td>
</tr>
<tr>
<td>Always</td>
<td>Z = 1</td>
</tr>
</tbody>
</table>

Subtraction & Comparison Examples

- SBA: A – B \rightarrow A
- SUBA: A – M \rightarrow A
- SUBB: B – M \rightarrow B
- CBA: A – B
- CMPA: A – M
- CMPB: B – M
Unconditional Branch

- **BRA = Branch Always**
  - tests no CCs
  - branch is always “taken”
  - always goes to “target” instruction

- A great way to get into an infinite loop
  - needs to be “guarded” by a conditional branch

- Uses the same old relative addressing mode

Long Range Branching

- **Problem**: branch instructions use short-range offsets
  - range = \((PC - 128) \ldots (PC + 127)\)
  - can’t handle long branches

- **Solution**: one instruction with full target address
  - JMP = jump Instruction
  - Use extended or indexed addressing

- Long conditional branch =
  - Branch to a JMP instruction,
  - JMP to the final target

Long Range Conditional Branch

- Long conditional branch
  - conditional Branch lets JMP be conditional

- Two options:
  - Conditional br. to JMP
  - JMP to the final target

- or
  - Invert Condition
  - Branch around JMP
  - JMP to final target

What you would like:

- \((JZ: \text{jump if zero})\)
  - JZ $1800
  - ADDA $9000

- \((BNZ: \text{branch if non-zero})\)
  - BNZ SKIP
  - JMP $1800
  - SKIP:
  - ADDA $9000

Branch Summary

- Instructions alter the Condition Codes (CCs)
  - Each instruction has the same effect on the CC bits regardless of the addressing mode

- Branch Instructions
  - Test one or more CCs
  - Decide whether to reload PC

- 4 CCs of primary interest in 68HC11
  - C
  - V
  - Z
  - N (not used in ECE 3170)
Branch Summary

- Branch Instruction Types:
  - Simple: test only 1 CC
  - Complex: test >1 CC
    - must be preceded by subtract or compare
  - Unconditional: always branch to “target” instruction

- Target Addressing:
  - Branch instructions use Relative Addressing
    - Range = (PC-128) … (PC+127)
  - JMP instr. allows long-range unconditional branch