Branching and Loops

Chapter 4
Microcontrollers
Objectives

- Use branch and jump instructions in short programs
- Use branch instructions to implement IF-THEN-ELSE programming structure
- Implement finite loops using the WHILE and UNTIL programming structures
Jumping

- Cause a change in program flow.

- JMP instruction directly loads the program counter with a 16 bit address.

- This address is commonly referred to as the destination address.
Jumping II

- Defined for EXT, INDX, and INDY addressing modes.
- No flags affected
- Example
  - JMP $0100
  - JMP STORE
Branching and Relative Address Mode

- Branch instructions do not supply the absolute address.

- The destination address must be calculated relative to the current program counter.

- Relative address follows the opcode in memory.
Calculating Destination Address

- Relative address is a signed 8-bit number that indicated the number of bytes to branch forward or backward in memory.

\[ DA = PC + \text{rr} \]

\[
DA = \text{Destination Address} \\
PC = \text{Program Counter} \\
\text{rr} = \text{relative address}
\]
Example 1

<table>
<thead>
<tr>
<th>Address</th>
<th>Machine Code</th>
<th>Source Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0109</td>
<td>20 6C</td>
<td>BRA PAST</td>
</tr>
</tbody>
</table>

PC = $0109 + 2 → $010B

$rr = $6C, sign extended → $006C

DA = PC + $rr → $0177
### Example 2

<table>
<thead>
<tr>
<th>Address</th>
<th>Machine Code</th>
<th>Source Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>01E2</td>
<td>20 D3</td>
<td>BRA LOOP</td>
</tr>
</tbody>
</table>

- \( PC = 01E2 + 2 \) → \( 01E4 \)
- \( rr = D3, \text{ sign extended} \) → \( FFD3 \)
- \( DA = PC + rr \) → \( 01B7 \)
Branch Instructions

- There are 21 branch instructions.
  - Instructions are paired
  - CCR flags used to test for branch condition
Unconditional Branch

- BRA – Branch Always
- BRN – Branch Never
Carry Flag

- BCC – Branch if carry cleared
- BCS – Branch if carry set
- BHS – Branch if higher or same
- BLO – Branch if lower
Zero Flag

- BEQ – Branch if equal to zero
- BNE – Branch if not equal to zero
Negative Flag

- BMI – Branch if minus
- BPL – Branch if positive
Overflow Flag

- BVC – Branch if overflow cleared
- BVS – Branch if overflow set
Carry and Zero

- BHI – Branch if higher
- BLS – branch if lower or same
Negative XOR Overflow

- BGE – Branch if greater than or equal
- BLT – Branch if less than
Zero or (Negative XOR Overflow)

- BGT – Branch if greater than
- BLE – Branch if less than or equal
Preparation for a Valid Branch Test

- Compare instructions are only executed in preparation for a conditional branch test.

- Performs a subtraction operation and updates the appropriate flags, discards the results of the subtraction.
## Compare Instructions

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMPA</td>
<td>Compare Accumulator to contents of memory</td>
</tr>
<tr>
<td>CMPB</td>
<td>Compare Accumulator to contents of memory</td>
</tr>
<tr>
<td>CPD</td>
<td>Compare 16-bit register to contents of memory</td>
</tr>
<tr>
<td>CPY</td>
<td>Compare 16-bit register to contents of memory</td>
</tr>
<tr>
<td>CPX</td>
<td>Compare A to B</td>
</tr>
<tr>
<td>CBA</td>
<td>Compare A to B</td>
</tr>
</tbody>
</table>
## Compare to Zero Instructions

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSTA</td>
<td>Test contents of Accumulator for zero or minus</td>
</tr>
<tr>
<td>TSTB</td>
<td>Test contents of Accumulator for zero or minus</td>
</tr>
<tr>
<td>TST</td>
<td>Test contents of memory for zero or minus</td>
</tr>
</tbody>
</table>

Copyright 2005 Richard Lokken
Using IF-THEN-ELSE

- IF condition is true the branch passes,

- THEN the program will branch,

- ELSE the branch test fails and the program falls through to the next instruction.
Example

LDDA $10
CMPA #$30 If <$30 BLO END then end
BLO END then end
CMPA #$39 else if >$39
BHI END then end
SUBA #$30 else subtract $30
STAA $11 Store result
END LDAA $12

Copyright 2005 Richard Lokken
While Loop

- Executes instructions only while condition is true.

LDX #$20  While Count =$20
back  CPX #$00  Dec Count
BEQ out
DEX
BRA Back

out

Copyright 2005 Richard Lokken
Until Loop

execute a set of instructions until some condition is true.

ldx #$20
back dex
bne back
Do
dec count
until count = 0

Copyright 2005 Richard Lokken
Timing Loops

- Two instructions that can help cause specific timing conditions
  - NOP – No operation, fetch opcode and do nothing, take two machine cycles.
  - BRN – Branch Never, perform branch test and fall through, takes three machine cycles.