ECE331 Handout 6- Advanced ASM

This handout describes 68HC12 Instructions and Address Modes that will NOT be covered in class or on exams, but you can use them in programming class assignments if you want to learn them on your own.

**Advanced Address Modes**

**Indexed-Indirect**
- offset from reference stored in accumulator
- offset + reference points to address containing data (not to data itself)
- defined by putting operands in brackets [ ]

**Example:**
LDAA [D,X] ; \{A ← <D+IX>\}
accA loaded with value at address specified by D+IX

**Indexed-Immediate with Increment**
- adjusts reference by offset
- reference can be IX, IY, SP (not PC)
- adjustment can be before or after instruction execution
  - pre-increment/decrement (+)
  - post-increment/decrement (-)
- defined by putting +/- in operand
  - before operand = pre-decrement
  - after operand = post-decrement

**Examples:**
pre-increment
LDAB $3,+Y
1) \{IY ← IY + $3\}
2) accB loads value from address specified by <new IY>

post-decrement
LDAB $2,Y-
1) accB loads value from address specified by <IY>
2) \{IY ← IY - $2\}

Note in pre-increment, data is loaded after IY is adjusted while in post-increment data is loaded before IY is adjusted.
**Advanced Branch Instructions**

**Loop Primitive Instructions**
- simultaneously branch and increment counter
- counter can be in A, B, D, X, Y, SP
- can increment, decrement, or test
- can branch if counter = 0 or ≠ 0

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Function</th>
<th>Equation or Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBEQ cntr, rel</td>
<td>Decrement counter and branch if = 0 (counter = A, B, D, X, Y, or SP)</td>
<td>counter ← (counter) - 1 If (counter) = 0, then branch else continue to next instruction</td>
</tr>
<tr>
<td>DBNE cntr, rel</td>
<td>Decrement counter and branch if ≠ 0 (counter = A, B, D, X, Y, or SP)</td>
<td>counter ← (counter) - 1 If (counter) ≠ 0, then branch else continue to next instruction</td>
</tr>
<tr>
<td>IBEQ cntr, rel</td>
<td>Increment counter and branch if = 0 (counter = A, B, D, X, Y, or SP)</td>
<td>counter ← (counter) + 1 If (counter) = 0, then branch else continue to next instruction</td>
</tr>
<tr>
<td>IBNE cntr, rel</td>
<td>Increment counter and branch if ≠ 0 (counter = A, B, D, X, Y, or SP)</td>
<td>counter ← (counter) + 1 If (counter) ≠ 0, then branch else continue to next instruction</td>
</tr>
<tr>
<td>TBEQ cntr, rel</td>
<td>Test counter and branch if = 0 (counter = A, B, D, X, Y, or SP)</td>
<td>If (counter) = 0, then branch else continue to next instruction</td>
</tr>
<tr>
<td>TBNE cntr, rel</td>
<td>Test counter and branch if ≠ 0 (counter = A, B, D, X, Y, or SP)</td>
<td>If (counter) ≠ 0, then branch else continue to next instruction</td>
</tr>
</tbody>
</table>

Note. 1. cnt is the loop counter and can be accumulators A, B, or D and register X, Y, or SP.
2. rel is the relative branch offset and is usually a label

Table 2.5  Summary of loop primitive instructions

**Example 1**

Write a program to add an array of N 8-bit numbers and store the sum at memory location $800$–$801$. Use the For $i = n1$ to n2 do looping construct.

**Solution:** We will use variable $i$ as the array index. This variable can also be used to keep track of the number of iterations remained to be performed. We will use a two-byte variable sum to hold the sum of array elements. The logic flow of the program is illustrated in Figure 2.9.

[Diagram of logic flow for Example 1]

**Example 2**

Write a program to find the maximum element from an array of N 8-bit elements using the repeat S until C looping construct.

**Solution:** We will use the variable $i$ as the array index and also as the loop count. The variable max_val will be used to hold the array maximum. The logic flow of the program is shown in Figure 2.10.

[Diagram of logic flow for Example 2]