Hand Assembling a Program

To hand-assemble a program, do the following:

1. Start with the org statement, which shows where the first byte of the program will go into memory. (E.g., org $2000 will put the first instruction at address $2000.)

2. Look at the first instruction. Determine the addressing mode used. (E.g., ldab #10 uses IMM mode.)

3. Look up the instruction in the CPU 12 Reference Guide, find the appropriate Addressing Mode, and the object code for that addressing mode. (E.g., ldab IMM has object code C6 ii.)

4. Put in the object code for the instruction, and put in the appropriate operand. Be careful to convert decimal operands to hex operands if necessary. (E.g., ldab #10 becomes C6 0A.)

5. Add the number of bytes of this instruction to the address of the instruction to determine the address of the next instruction. (E.g., $2000 + 2 = $2002 will be the starting address of the next instruction.)

**LDAB: page 12, CLRA: page 8, DBNE: page 10, table 4 on page 23**
**SWI: page 18**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Addressing Mode</th>
<th>Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>org $2000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c6 0a</td>
<td>ldab #10</td>
<td>1</td>
</tr>
<tr>
<td>87</td>
<td>clra</td>
<td>1</td>
</tr>
<tr>
<td>04 31 fc</td>
<td>dbne b,loop</td>
<td>3</td>
</tr>
<tr>
<td>3f</td>
<td>swi</td>
<td>9</td>
</tr>
</tbody>
</table>

You can determine how many cycles an instruction takes by looking up the CPU cycles for that instruction in the CPU 12 Reference Guide.

– For example, LDAB using the IMM addressing mode shows one CPU cycle (of type P).
– LDAB using the EXT addressing mode shows three CPU cycles (of type rOP).
– Section 6.6 of the CPU12 Reference Manual explains what the HC12 is doing during each of the different types of CPU cycles.

The program executes the ldab #10 instruction once (which takes one cycle). It then goes through loop 10 times (which has two instructions, one with one cycle and one with three cycles), and finishes with the swi instruction (which takes 9 cycles).

Total number of cycles:

\[1 + 10 \times (1 + 3) + 9 = 50\]
Handling Assembler Directives

2000       org $2000
2000 23 17 f2 a3 56 table1: dc.b $23,$17,$f2,$a3,$56
2005 05    table2:   dc.b 5
2006 43 af  var:     dc.w $43af

Disassembly of an HC12 Program

• It is sometimes useful to be able to convert HC12 op codes into mnemonics.
• For example, consider the hex code:

ADDR   DATA (13 Byte 1000-100C)
----   ---------------------------------------
1000   C6 05 CE 20 00 E6 01 18 06 04 35 EE 3F

• To determine the instructions, use Table 6 of the CPU12 Reference Guide

  – If the first byte of the instruction is anything other than $18, use Sheet 1 of 2 (Page 28). From this table, determine the number of bytes of the instruction and the addressing mode. For example, $C6 is a two-byte instruction, the mnemonic is LDAB, and it uses the IMM addressing mode. Thus, the two bytes C6 05 is the op code for the instruction LDAB #$05.

If the first byte is $18, use Sheet 2 of 2 (Page 29), and do the same thing. For example, 18 06 is a two byte instruction, the mnemonic is ABA, and it uses the INH addressing mode, so there is no operand. Thus, the two bytes 18 06 is the op code for the instruction ABA.

  – Indexed addressing mode is fairly compicated to disassemble. You need to use Table 1 to determine the operand. For example, the op code $E6 indicates LDAB indexed, and may use two to four bytes (one to three bytes in addition to the op code). The postbyte 01 indicates that the operand is 1,X , which is 5-bit constant offset, which takes only one additional byte. All 5-bit constant offset, pre and post increment and decrement, and register offset instructions use one additional byte. All 9-bit constant offset instructions use two additional bytes, with the second byte holding 8 bits of the 9 bit offset. (The 9th bit is a direction bit, which is held in the first postbyte.) All 16-bit constant offset instructions use three postbytes, with the 2nd and 3rd holding the 16-bit unsigned offset.

  – Transfer (TFR) and exchange (EXG) instructions all have the op code $B7. Use Table 3 (Page 22) to determine whether it is TFR or an EXG, and to determine which registers are being used. If the most significant bit of the postbyte is 0, the instruction is a transfer instruction.

  – Loop instructions (Decrement and Branch, Increment and Branch, and Test and Branch) all have the op code $04. To determine which instruction the op code $04 implies, and whether the branch is positive (forward) or negative (backward), use Table 4 (Page 23). For example, in the sequence 04 35 EE, the 04 indicates a loop instruction. The 35 indicates it is a DBNE X instruction (decrement register X and branch if result is not equal to zero), and the direction is backward (negative). The EE indicates a branch of -12 bytes.

Use up all the bytes for one instruction, then go on to the next instruction.

C6 05   => LDAA #$05 ;two-byte LDAA, IMM addressing mode
CE 20 00 => LDX #$2000 ;three-byte LDX, IMM addressing mode
E6 01   => LDAB 1,X ;two to four-byte LDAB, IDX addressing
;mode. Operand 01 => 1,X, a 5b constant
;offset which uses only one postbyte

18 06  => ABA ;two-byte ABA, INH addressing mode
04 35 EE => DBNE X,(-18) ;three-byte loop instruction
;Postbyte 35 indicates DBNE X, negative
3F     => SWI ;one-byte SWI, INH addressing mode