```
 1   ; arith.asm
 2   ;
 3   ; Testing basic ALU operations
 4   ;
 5   ; This file tests the following instructions:
 6   ;     NEG Rm, Rn
 7   ;     ADD Rm, Rn
 8   ;     SUB Rm, Rn
 9   ;     DT Rn
10   ;
11   ; Revision History:
12   ;   11 May 2025    Zack Huang     Initial revision.
13
14   ProgramStart:
15       ; Test negation
16       MOV #1, R1       ; R1 <- 1
17       NEG R1, R1       ; Negate R1
18       MOV #$0, R0;
19       MOV R1, @R0;     ; Output R1 to 0x00 (expect -1)
20
21       NEG R1, R1       ; Negate R1
22       MOV #$4, R0
23       MOV R1, @R0      ; Output R1 to 0x04 (expect 1)
24
25       ; Test addition
26       MOV #1, R1       ; R1 <- 1
27       ADD #8, R1       ; Add 8 to R1
28       MOV #$8, R0;
29       MOV R1, @R0;     ; Output R1 to 0x08 (expect 9)
30
31       ; Test subtraction
32       MOV #1, R1       ; R1 <- 1
33       MOV #8, R2       ; R2 <- 8
34       SUB R2, R1       ; Subtract R2 from R1
35       MOV #$C, R0
36       MOV R1, @R0      ; Output R1 to 0x0C (expect -7)
37
38       ; Test DT
39       MOV #2, R2       ; R2 <- 2
40       DT R2            ; Decrement R2
41       MOV #$10, R1
42       MOV R2, @R1      ; Output R2 (1) to 0x10
43
44       STC SR, R0       ; Store SR to R0
45       AND #1, R0       ; mask out T bit
46       MOV #$14, R1
47       MOV R0, @R1      ; Output R0 to 0x14 (expect 0)
48
49       DT R2            ; Decrement R2
50       MOV #$18, R1
51       MOV R2, @R1      ; Output R2 to 0x18 (expect 0)
52
53       STC SR, R0       ; Store SR to R0
54       AND #1, R0       ; mask out T bit
55       MOV #$1C, R1
56       MOV R0, @R1      ; Output R0 to 0x1C (expect 1)
57
58       ; Quit test program
59       MOV #-4, R0;
60       MOV.B R0, @R0;
61
```

```
1  00000000 FFFFFFFF
2  00000004 00000001
3  00000008 00000009
4  0000000C FFFFFFF9
5  00000010 00000001
6  00000014 00000000
7  00000018 00000000
8  0000001C 00000001
9
```

```
1    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
2    ;
3    ; branch.asm
4    ;
5    ; This file tests the following instructions:
6    ;
7    ;   BF    <label>
8    ;   BF/S  <label>
9    ;   BT    <label>
10   ;   BT/S  <label>
11   ;   BRA   <label>
12   ;   BRAF  Rm
13   ;   BSR   <label>
14   ;   BSRF  Rm
15   ;   JMP   @Rm
16   ;   JSR   @Rm
17   ;   RTS
18   ;
19   ; Branches are tested by writing to memory based on address we jump to. Note
20   ; that if jumps do not work at all, we will encounter the "exit" signal and
21   ; the expected memory will be incomplete. The execution of the instructions in
22   ; the delay slot is tested by seeing if the contents of a register are altered
23   ; or not when we arrive at the target address.
24   ;
25   ; Revision History:
26   ;   26 May 2025  Chris M.  Initial revision.
27   ;
28   ; TODO:
29   ;   - Use long-word constants instead of manually calculating addresses.
30   ;   - Format nicely.
31   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
32
33   Start:
34
35       ; Test BF. Write at 0x00
36
37       CLRT              ; Clear the T flag.
38       BT    TRGET_T0    ; This branch should not be taken.
39       BF    TRGET_F0    ; This branch is taken. The PC should be changed to TRGET_F0
40       NOP
41       NOP
42
43   TRGET_T0:
44
45       MOV   TrueVar,  R1  ; Write TrueVar @ 0x00 to detect wrong branch.
46       MOV   #$00,     R0
47       MOV   R1,       @R0
48
49   TRGET_T1:
50
51       MOV   TrueVar,  R1  ; Write TrueVar @ 0x08 to detect wrong branch.
52       MOV   #$08,     R0
53       MOV   R1,       @R0
54
55   End_T0:
56
57       ; The test bench interprets a read of 0xFFFFFFFC (-4)
58       ; as system exit.
59       MOV     #-4,  R0
60       MOV.B   R0,  @R0
61
62
63   TRGET_F0:
64
65       ; Test BF/S. Write at 0x04 and 0x08
66
```

```
67      MOV   FalseVar,  R1    ; Write FalseVar at 0x00
68      MOV   #$00,      R0
69      MOV   R1,        @R0
70
71      MOV   #6,    R2
72      MOV   #7,    R3
73
74      CLRT            ; Clear the T flag.
75
76      BF/S  TRGET_F1  ; Branch to TRGET_F1 and exeucte the delay slot.
77      ADD   R2, R3    ; 6 + 7 = 13 should be in R3 when we get to TRGET_F1.
78
79      BT/S  TRGET_T1  ; This branch should not be taken.
80      NOP
81
82      NOP
83
84   TRGET_F1:
85
86      MOV   #$04,  R0   ; If the delay slot was executed, then 0x0D is in R3.
87      MOV   R3,    @R0  ; Write 0x0D to 0x04
88
89      MOV   FalseVar,  R1  ; Write FalseVar at 0x08
90      MOV   #$08,      R0
91      MOV   R1,        @R0
92
93      ; -- Test BT. Write at 0x0C ------------------------------------------
94
95      SETT            ; Set the T flag
96      BF    TRGET_F2  ; This branch should not be taken.
97      BT    TRGET_T2  ; This branch is taken.
98      NOP
99      NOP
100
101  TRGET_F2:
102
103     MOV   FalseVar,  R1  ; Write FalseVar at 0x0C to detect wrong branch.
104     MOV   #$0C,      R0
105     MOV   R1,        @R0
106
107  TRGET_T2:
108
109     MOV   TrueVar,  R1    ; Write TrueVar at 0x0C
110     MOV   #$0C,     R0
111     MOV   R1,       @R0
112
113     CLRT                  ; Clear T flag
114
115     ; -- Test BT/S. Write at 0x10 ---------------------------------------
116
117     MOV   #$09,     R3    ; Set up register for addition
118     MOV   #$0C,     R4
119
120     SETT                  ; Set T flag.
121
122     BF/S  TRGET_F3        ; Should not be taken
123     NOP
124
125     BT/S  TRGET_T3        ; Should be taken
126     ADD   R3,       R4    ; 0x15 should be in R4 if the branch slot is executed.
127     NOP
128
129  TRGET_F3:
130
131     MOV   FalseVar,  R1  ; Write FalseVar at 0x10 to detect wrong branch.
132     MOV   #$10,      R0
```

```
133        MOV    R1,        @R0
134
135        ; The test bench interprets a read of 0xFFFFFFFC (-4)
136        ; as system exit.
137        MOV     #-4, R0;   ; PC = 0x1A
138        MOV.B   R0, @R0;   ; PC = 0x1C
139
140   TRGET_T3:
141
142        MOV    #$10,      R0  ; If the delay slot is executed, then 0x15 is in R4
143        MOV    R4,        @R0  ; Write 0x15 @ 0x10
144
145        MOV    TrueVar,   R1  ; Write TrueVar at 0x14 if `BT/S` jumps to the correct address.
146        MOV    #$14,      R0
147        MOV    R1,        @R0
148
149
150
151        ; -- Test BRA. Write to 0x18 ----------------------------------------
152
153        MOV    #$01,   R2
154        MOV    #$02,   R3
155
156        BRA TRGET_BRA;
157        ADD    R2,     R3  ; Branch slot for BRA
158
159
160        ; If BRA doesn't work, the test exits.
161        ;
162        ; The test bench interprets a read of 0xFFFFFFFC (-4)
163        ; as system exit.
164        MOV     #-4, R0;   ; PC = 0x1A
165        MOV.B   R0, @R0;   ; PC = 0x1C
166
167
168   TRGET_BRA:
169
170        MOV    #$18,      R0  ; Write result of branch slot (0x03) at 0x18
171        MOV    R3,        @R0
172
173        MOV    TrueVar,  R1  ; Write TrueVar at 0x1C
174        MOV    #$1C,      R0
175        MOV    R1,        @R0
176
177
178        ; -- Test BRAF -----------------------------------------------------
179
180
181        MOV    #7,   R2
182        MOV    #9,   R3
183
184        MOV    #10,   R4  ; The target of BRAF is 4 instructions away from the BRAF.
185                         ; So we add 8 + 2 because our PC points to current
186                         ; instruction instead of the next as expected by the spec.
187
188
189        BRAF  R4        ; Branch to TRGET_BRAF
190        ADD   R2,   R3
191
192        NOP
193
194        ; If the branch is not taken, the system exits
195        ; The test bench interprets a read of 0xFFFFFFFC (-4)
196        ; as system exit.
197        MOV     #-4, R0
198        MOV.B   R0, @R0
```

```
199
200
201  TRGET_BRAF:
202
203      MOV  #$20,     R0  ; Write the result of the branch slot (0x10) at 0x20
204      MOV  R3,       @R0
205
206      MOV  #$24,     R0 ; Write TrueVar at 0x24 to signal that BRAF made it to
207      MOV  TrueVar,  R1 ; the target.
208      MOV  R1,       @R0
209
210      ; -- Test BSR -------------------------------------------------------
211
212      MOV #10,  R3  ; Values to test execution of branch slot.
213      MOV #12,  R4
214
215
216      BSR  TRGET_BSR
217      ADD R3,   R4
218
219      NOP                 ; RTS returns here.
220
221      MOV  #$30,  R0    ; Write the result of the RTS branch slot (0x11) at 0x30
222      MOV  R1,    @R0
223
224
225      ; -- Test BSRF ------------------------------------------------------
226
227      MOV #5, R3    ; Values to test execution of branch slot.
228      MOV #3, R4
229
230      MOV  #38, R1    ; TRGET_BSRF is 18 instructions (36 bytes) away from BSRF.
231                     ; We add an extra 2 bytes because our PC points to the
232                     ; current instruction instead of the next instruction as
233                     ; expected by the spec.
234
235      BSRF  R1
236      ADD   R3,  R4    ; Branch slot for BSRF
237
238      NOP              ; RTS returns here
239
240      MOV  #$3C,  R0   ; Write the result of the RTS branch slot, 0x22 at 0x38
241      MOV  R1,    @R0
242
243      ; -- Test JMP  ------------------------------------------------------
244
245      ; MOV.W   TRGET_JMP, R0
246
247      ; MOVA  TRGET_JMP, R0  ; Load address of target into R0
248
249      ; MOV #$36, R0         ; 17 * 2 + 2 = 36
250
251      MOV.L  TRGET_JMP_DATA, R0 ;Load target address of JMP into R0
252
253      MOV #2,  R3       ; Values to test execution of branch slot.
254      MOV #5,  R4
255
256      JMP  @R0
257      ADD  R3,  R4       ; Branch slot of JMP
258
259
260      MOV    #-4,  R0   ; If the JMP is not taken, the system exits.
261      MOV.B  R0,  @R0
262
263
264  TRGET_BSR:
```

4

```
265       MOV   #$28,   R0    ; Write result of the BSR branch slot (0x16) at 0x28
266       MOV   R4,     @R0
267
268
269       MOV   #$2C,     R0  ; Write TrueVar at 0x2C to signal that BSR jumped to the
270       MOV   TrueVar, R1   ; correct target.
271       MOV   R1,      @R0
272
273       RTS                 ; Return to the instruction after the branch slot of BSR
274       MOV   #$11,    R1   ; Branch slot of RTS
275
276   TRGET_BSRF:
277
278
279       MOV   #$34,    R0    ; Write the result of the BSRF branch slot, 0x08 at 0x34
280       MOV   R4,      @R0
281
282       MOV   #$38,    R0    ; Write TrueVar at 0x3C to signal that BSRF jumped
283       MOV   TrueVar, R1    ; to the correct target.
284       MOV   R1,      @R0
285
286       RTS                  ; Return to the instruction after the branch slot of BSRF
287
288       MOV  #$22, R1        ; Branch slot of RTS
289
290   TRGET_JMP:
291
292       MOV  #$40,     R0   ; Write the result of the JMP branch slot, 0x07 at 0x40
293       MOV  R4,       @R0
294
295       MOV  #$44,     R0   ; Write TrueVar at 0x40 to signal that JMP jumped to
296       MOV  TrueVar,  R1   ; the correct target.
297       MOV  R1,       @R0
298
299       ; -- Test JSR -------------------------------------------------------
300
301       MOV.L   TRGET_JSR_DATA, R0 ; Load target address of JSR into R0
302
303
304       MOV   #10,    R3   ; Values to check the execution of the branch slot.
305       MOV   #20,    R4
306
307       JSR   @R0
308       ADD   R3,     R4   ; Branch slot of JSR
309
310       NOP                   ; RTS returns here.
311
312       MOV   #$50,  R0    ; Write the result of RTS's branch slot, 0x33 at 0x50
313       MOV   R1,    @R0
314
315   End:
316
317       ; The test bench interprets a read of 0xFFFFFFFC (-4)
318       ; as system exit.
319       MOV      #-4, R0
320       MOV.B  R0, @R0
321
322
323
324   TRGET_JSR:
325
326       MOV      #$48,    R0 ; Write the result of the JSR branch slot, 0x1E, at 0x48.
327       MOV      R4,      @R0
328
329       MOV.L   TrueVar,  R1 ; Write TrueVar at 0x4C to signal that JSR jumped to
330       MOV      #$4C,    R0 ; the correct target.
```

```
331     MOV     R1,     @R0
332
333     RTS                     ; Return
334     MOV     #$33,    R1  ; Branch slot of RTS
335
336
337
338     ; Expected memory:
339     ; 00000000 BBBBBBBB  ; FalseVar is written to 0x00 if `BF` jumps to the correct target.
340     ; 00000004 0000000D  ; 0x0D (13) is written to 0x04 if the delay slot of a `BF/S` is executed.
341     ; 00000008 BBBBBBBB  ; FalseVar is written to 0x08 if `BF/S` jumps to the correct target.
342     ; 0000000C AAAAAAAA  ; TrueVar is writen to 0x0C if `BT` jumps to the correct target.
343     ; 00000010 00000015  ; 0x15 is written to 0x10 if the delay slot of a `BT/S` is exeucted.
344     ; 00000014 AAAAAAAA  ; TrueVar is written to 0x14 if `BT/S` jumps to the correct target.
345     ; 00000018 00000003  ; 0x03 is written to 0x18 if the branch slot of the `BRA` is executed.
346     ; 0000001C AAAAAAAA  ; TrueVar is written to 0x1C of `BRA` jumps to the correct target.
347     ; 00000020 00000010  ; 0x10 is written to 0x20 if the branch slot of the `BRAF` is executed.
348     ; 00000024 AAAAAAAA  ; TrueVar is written to 0x24 if `BRAF` jumps to the correct target.
349     ; 00000028 00000016  ; 0x16 is written to 0x28 if the branch slot of `BSR` is executed.
350     ; 0000002C AAAAAAAA  ; TrueVar is written to 0x2C if `BSR` jumps to the correct instruction.
351     ; 00000030 00000011  ; 0x11 is written to 0x30 if the branch slot of `RTS` is executed.
352     ; 00000034 00000008  ; 0x08 is written to 0x34 if the branch slot of `BSRF` is executed.
353     ; 00000038 AAAAAAAA  ; TrueVar is written to 0x38 if `BSRF` jumps to the correct instruction.
354     ; 0000003C 00000022  ; 0x22 is written to 0x3C if the branch slot of `RTS` is executed.
355     ; 00000040 00000007  ; 0x07 is written to 0x40 if the branch slot of `JMP` is executed.
356     ; 00000044 AAAAAAAA  ; TrueVar is written to 0x44 if `JMP` jumps to the correct target.
357     ; 00000048 0000001E  ; 0x1E is written to 0x48 if the branch slot of `JSR` is executed.
358     ; 0000004C AAAAAAAA  ; TrueVar is written to 0x4C if `JSR` jumps to the correct target.
359     ; 00000050 00000033  ; 0x33 is written to 0x50 if the branch slot of `RTS` is executed.
360
361     align 4
362     LTORG
363
364 TrueVar:        dc.l $AAAAAAAA
365 FalseVar:       dc.l $BBBBBBBB
366 TRGET_JMP_DATA: dc.l TRGET_JMP
367 TRGET_JSR_DATA: dc.l TRGET_JSR
368
```

```
 1   00000000  BBBBBBBB
 2   00000004  0000000D
 3   00000008  BBBBBBBB
 4   0000000C  AAAAAAAA
 5   00000010  00000015
 6   00000014  AAAAAAAA
 7   00000018  00000003
 8   0000001C  AAAAAAAA
 9   00000020  00000010
10   00000024  AAAAAAAA
11   0000002C  AAAAAAAA
12   00000030  00000011
13   00000034  00000008
14   00000038  AAAAAAAA
15   0000003C  00000022
16   00000040  00000007
17   00000044  AAAAAAAA
18   00000048  0000001E
19   0000004C  AAAAAAAA
20   00000050  00000033
21
```

```
 1   ; bshift.asm
 2   ;
 3   ; Testing barrel shift instructions
 4   ;
 5   ; This file tests the following instructions:
 6   ;     SHLL2 Rn
 7   ;     SHLR2 Rn
 8   ;     SHLL8 Rn
 9   ;     SHLR8 Rn
10   ;     SHLL16 Rn
11   ;     SHLR16 Rn
12   ;
13   ; Revision History:
14   ;   25 May 2025    Zack Huang     Initial revision.
15
16   ProgramStart:
17       ; Test SHLL2
18       MOV #$F, R0;
19       SHLL2 R0;
20       MOV #$0, R1;
21       MOV R0, @R1;    Expect: R0 = 0x000000F0
22
23       ; Test SHLR2
24       MOV #$F, R0;
25       SHLR2 R0;
26       MOV #$4, R1;
27       MOV R0, @R1;    Expect: R0 = 0x000000F0
28
29       ; Test SHLL8
30       MOV #$F, R0;
31       SHLL8 R0;
32       MOV #$8, R1;
33       MOV R0, @R1;    Expect: R0 = 0x0000F000
34
35       ; Test SHLR8
36       SHLR8 R0;
37       MOV #$C, R1;
38       MOV R0, @R1;    Expect: R0 = 0x0000000F
39
40       ; Test SHLL16
41       MOV #$F, R0;
42       SHLL16 R0;
43       MOV #$10, R1;
44       MOV R0, @R1;    Expect: R0 = 0x000F0000
45
46       ; Test SHLR8
47       SHLR16 R0;
48       MOV #$14, R1;
49       MOV R0, @R1;    Expect: R0 = 0x0000000F
50
51       ; Quit test program
52       MOV #-4, R0;
53       MOV.B R0, @R0;
54
```

```
1  00000000 0000003C
2  00000004 00000003
3  00000008 00000F00
4  0000000C 0000000F
5  00000010 000F0000
6  00000014 0000000F
7
```

```
1    ; cmp.asm
2    ;
3    ; Testing all comparison operations
4    ;
5    ; This file tests the following instructions:
6    ;    CMP/EQ  #imm,RO
7    ;    CMP/EQ  Rm, Rn
8    ;    CMP/HS  Rm, Rn
9    ;    CMP/GE  Rm, Rn
10   ;    CMP/HI  Rm, Rn
11   ;    CMP/GT  Rm, Rn
12   ;    CMP/PL  Rn
13   ;    CMP/PZ  Rn
14   ;    CMP/STR Rm, Rn
15   ;
16   ; Revision History:
17   ;   25 May 2025     Zack Huang     Initial revision.
18
19   ProgramStart:
20       ; Test CMP/EQ (when unequal)
21       MOV #50, R1;
22       MOV #40, R2;
23
24       CMP/EQ R1, R2;  Should set T = 0
25       STC SR, R0;
26       AND #1, R0;
27       MOV #$00, R3;
28       MOV R0, @R3;    Expect R0 = 00000000
29
30       ; Test CMP/EQ (when equal)
31       MOV #41, R1;
32       MOV #41, R2;
33
34       CMP/EQ R1, R2;  Should set T = 1
35       STC SR, R0;
36       AND #1, R0;
37       MOV #$04, R3;
38       MOV R0, @R3;    Expect R0 = 00000001
39
40       ; Test CMP/HS
41       MOV #$10, R1;
42       MOV #$30, R2;
43       CMP/HS R1, R2;  Should set T = 1
44       STC SR, R0;
45       AND #1, R0;
46       MOV #$08, R3;
47       MOV R0, @R3;    Expect R0 = 00000001
48
49       CMP/HS R2, R1;  Should set T = 0
50       STC SR, R0;
51       AND #1, R0;
52       MOV #$0C, R3;
53       MOV R0, @R3;    Expect R0 = 00000000
54
55       ; Test CMP/GE
56       MOV #-4, R1;
57       MOV #10, R2;
58       CMP/GE R1, R2;  Should set T = 1
59       STC SR, R0;
60       AND #1, R0;
61       MOV #$10, R3;
62       MOV R0, @R3;    Expect R0 = 00000001
63
64       CMP/GE R2, R1;  Should set T = 0
65       STC SR, R0;
66       AND #1, R0;
```

```
67        MOV #$14, R3;
68        MOV R0, @R3;     Expect R0 = 00000000
69
70        ; Test CMP/HI
71        MOV #-1, R1;
72        MOV #-1, R2;
73        CMP/HI R1, R2;  Should set T = 0
74        STC SR, R0;
75        AND #1, R0;
76        MOV #$18, R3;
77        MOV R0, @R3;     Expect R0 = 00000000
78
79        MOV #10, R1;
80        MOV #-1, R2;
81        CMP/HI R1, R2;  Should set T = 1
82        STC SR, R0;
83        AND #1, R0;
84        MOV #$1C, R3;
85        MOV R0, @R3;     Expect R0 = 00000001
86
87        CMP/HI R2, R1;  Should set T = 0
88        STC SR, R0;
89        AND #1, R0;
90        MOV #$20, R3;
91        MOV R0, @R3;     Expect R0 = 00000000
92
93        ; Test CMP/GT
94        MOV #-1, R1;
95        MOV #-1, R2;
96        CMP/GT R1, R2;  Should set T = 0
97        STC SR, R0;
98        AND #1, R0;
99        MOV #$24, R3;
100       MOV R0, @R3;     Expect R0 = 00000000
101
102       MOV #10, R1;
103       MOV #-1, R2;
104       CMP/GT R1, R2;  Should set T = 0
105       STC SR, R0;
106       AND #1, R0;
107       MOV #$28, R3;
108       MOV R0, @R3;     Expect R0 = 00000000
109
110       CMP/GT R2, R1;  Should set T = 1
111       STC SR, R0;
112       AND #1, R0;
113       MOV #$2C, R3;
114       MOV R0, @R3;     Expect R0 = 00000001
115
116       ; Test CMP/EQ #imm, Rn
117       MOV #10, R0;
118       CMP/EQ #8, R0;
119       STC SR, R0;
120       AND #1, R0;
121       MOV #$30, R3;
122       MOV R0, @R3;     Expect R0 = 00000000
123
124       MOV #10, R0;
125       CMP/EQ #10, R0;
126       STC SR, R0;
127       AND #1, R0;
128       MOV #$34, R3;
129       MOV R0, @R3;     Expect R0 = 00000001
130
131       ; Test CMP/PL Rn
132       MOV #0, R0;
```

```
133        CMP/PL R0;
134        STC SR, R0;
135        AND #1, R0;
136        MOV #$38, R3;
137        MOV R0, @R3;     Expect R0 = 00000000
138
139        MOV #10, R0;
140        CMP/PL R0;
141        STC SR, R0;
142        AND #1, R0;
143        MOV #$3C, R3;
144        MOV R0, @R3;     Expect R0 = 00000001
145
146        MOV #-10, R0;
147        CMP/PL R0;
148        STC SR, R0;
149        AND #1, R0;
150        MOV #$40, R3;
151        MOV R0, @R3;     Expect R0 = 00000000
152
153        ; Test CMP/PL Rn
154        MOV #0, R0;
155        CMP/PZ R0;
156        STC SR, R0;
157        AND #1, R0;
158        MOV #$44, R3;
159        MOV R0, @R3;     Expect R0 = 00000001
160
161        MOV #10, R0;
162        CMP/PZ R0;
163        STC SR, R0;
164        AND #1, R0;
165        MOV #$48, R3;
166        MOV R0, @R3;     Expect R0 = 00000001
167
168        MOV #-10, R0;
169        CMP/PZ R0;
170        STC SR, R0;
171        AND #1, R0;
172        MOV #$4C, R3;
173        MOV R0, @R3;     Expect R0 = 00000000
174
175        ; Test CMP/STR
176        MOV #$0, R1;
177        MOV #-1, R2;
178        CMP/STR R0, R2;
179        STC SR, R0;
180        AND #1, R0;
181        MOV #$50, R3;
182        MOV R0, @R3;     Expect R0 = 00000000
183
184        MOV #$0, R1;
185        MOV #$A, R2;
186        CMP/STR R0, R2;
187        STC SR, R0;
188        AND #1, R0;
189        MOV #$54, R3;
190        MOV R0, @R3;     Expect R0 = 00000001
191
192        ; Quit test program
193        MOV #-4, R0;
194        MOV.B R0, @R0;
195
```

```
 1   00000000 00000000
 2   00000004 00000001
 3   00000008 00000001
 4   0000000C 00000000
 5   00000010 00000001
 6   00000014 00000000
 7   00000018 00000000
 8   0000001C 00000001
 9   00000020 00000000
10   00000024 00000000
11   00000028 00000000
12   0000002C 00000001
13   00000030 00000000
14   00000034 00000001
15   00000038 00000000
16   0000003C 00000001
17   00000040 00000000
18   00000044 00000001
19   00000048 00000001
20   0000004C 00000000
21   00000050 00000000
22   00000054 00000001
23
```

```
1    ; control.asm
2    ;
3    ; Testing all control register operations
4    ;
5    ; This file tests the following instructions:
6    ;   STC    SR, Rn
7    ;   STC    GBR, Rn
8    ;   STC    VBR, Rn
9    ;   STC.L  SR, @-Rn
10   ;   STC.L  GBR, @-Rn
11   ;   STC.L  VBR, @-Rn
12   ;   STS    MACH, Rn
13   ;   STS    MACL, Rn
14   ;   STS    PR, Rn
15   ;   STS.L  MACH, @-Rn
16   ;   STS.L  MACL, @-Rn
17   ;   STS.L  PR, @-Rn
18   ;
19   ; Revision History:
20   ;   25 May 2025    Zack Huang     Initial revision.
21
22   ProgramStart:
23       ; Test LDS
24       MOV #$F, R0;
25       LDS R0, PR;      SR  <- 0x0000000F
26
27       ADD #1, R0;
28       LDS R0, MACH;    MACH <- 0x00000100
29
30       ADD #1, R0;
31       LDS R0, MACL;    MACL <- 0x00000101
32
33       STS PR, R0;
34       MOV #$0, R1;
35       MOV R0, @R1;    Expect R0 = 0x0000000F
36
37       STS MACH, R0;
38       MOV #$4, R1;
39       MOV R0, @R1;    Expect R0 = 0x00000010
40
41       STS MACL, R0;
42       MOV #$8, R1;
43       MOV R0, @R1;    Expect R0 = 0x00000011
44
45       ; Test LDS.L
46       MOV #$70, R3;
47       MOV #$1A, R2;
48       MOV R2, @R3;    Set memory at 0x70 to be 0x1A
49
50       MOV #$74, R3;
51       MOV #$2B, R2;
52       MOV R2, @R3;    Set memory at 0x74 to be 0x2B
53
54       MOV #$78, R3;
55       MOV #$3C, R2;
56       MOV R2, @R3;    Set memory at 0x78 to be 0x3C
57
58       MOV #$70, R2;
59       LDS.L @R2+, PR;     Read SR from memory, increment R2
60       STS PR, R0;
61       MOV #$C, R1;
62       MOV R0, @R1;       Expect R0 = 0x000000AA
63
64       LDS.L @R2+, MACH;    Read MACH from memory, increment R2
65       STS MACH, R0;
66       MOV #$10, R1;
```

```
67      MOV R0, @R1;        Expect R0 = 0x000000BB
68
69      LDS.L @R2+, MACL;     read MACL from memory, increment R2
70      STS MACL, R0;
71      MOV #$14, R1;
72      MOV R0, @R1;        Expect R0 = 0x000000CC
73
74      ; Test STS.L
75      MOV #$4C, R0;
76      LDS R0, PR;      SR  <- 0x0000004C
77
78      ADD #1, R0;
79      LDS R0, MACH;     MACH <- 0x0000004D
80
81      ADD #1, R0;
82      LDS R0, MACL;     MACL <- 0x0000004E
83
84      MOV #$24, R1;
85      STS.L PR, @-R1;     Expect 0x0000004C at 0x20
86      STS.L MACH, @-R1;    Expect 0x0000004D at 0x1C
87      STS.L MACL, @-R1;    Expect 0x0000004E at 0x18
88
89      ; Test CLRMAC
90      CLRMAC;
91
92      MOV #$30, R1;
93      STS.L MACH, @-R1;    Expect 0x00000000 at 0x2C
94      STS.L MACL, @-R1;    Expect 0x00000000 at 0x28
95
96      ; Quit test program
97      MOV #-4, R0;
98      MOV.B R0, @R0;
99
100
```

```
 1   00000000 0000000F
 2   00000004 00000010
 3   00000008 00000011
 4   0000000C 0000001A
 5   00000010 0000002B
 6   00000014 0000003C
 7   00000020 0000004C
 8   0000001C 0000004D
 9   00000018 0000004E
10   0000002C 00000000
11   00000028 00000000
12
```

```
1   ; ext.asm
2   ;
3   ; Testing all sign/zero extend instructions
4   ;
5   ; This file tests the following instructions:
6   ;   EXTS.B  Rm, Rn
7   ;   EXTS.W  Rm, Rn
8   ;   EXTU.B  Rm, Rn
9   ;   EXTU.W  Rm, Rn
10  ;
11  ; Revision History:
12  ;   25 May 2025    Zack Huang    Initial revision.
13
14  ProgramStart:
15      ; Test EXTU.B
16      MOV #-1, R0;
17      EXTU.B R0, R1;
18      MOV #$0, R2;
19      MOV R1, @R2;    Expect 000000FF
20
21      ; Test EXTU.W
22      MOV #-1, R0;
23      EXTU.W R0, R1;
24      MOV #$4, R2;
25      MOV R1, @R2;    Expect 0000FFFF
26
27      ; Test EXTS.B
28      MOV #-1, R0;
29      AND #$FF, R0;
30      EXTS.B R0, R1;
31      MOV #$8, R2;
32      MOV R1, @R2;    Expect FFFFFFFF
33
34      ; Test EXTS.W
35      MOV #-1, R0;
36      AND #$FF, R0;
37      EXTS.W R0, R1;
38      MOV #$C, R2;
39      MOV R1, @R2;    Expect 000000FF
40
41      MOV #-1, R0;
42      AND #$FF, R0;
43
44      SETT
45      ROTCL R0;
46      SETT
47      ROTCL R0;
48      SETT
49      ROTCL R0;
50      SETT
51      ROTCL R0;
52      SETT
53      ROTCL R0;
54      SETT
55      ROTCL R0;
56      SETT
57      ROTCL R0;
58      SETT
59      ROTCL R0;        Set R0 to 0000FFFF
60
61      EXTS.W R0, R1;
62      MOV #$10, R2;
63      MOV R1, @R2;    Expect FFFFFFFF
64
65      ; Quit test program
66      MOV #-4, R0;
```

1

```
67      MOV.B R0, @R0;
68
69
```

```
1  00000000 000000FF
2  00000004 0000FFFF
3  00000008 FFFFFFFF
4  0000000C 000000FF
5  00000010 FFFFFFFF
6
```

```
1    ; logic.asm
2    ;
3    ; Testing ALU logic operations
4    ;
5    ; This file tests the following instructions:
6    ;   NOT  Rm, Rn
7    ;   AND  Rm, Rn
8    ;   AND  #imm, R0
9    ;   OR   Rm, Rn
10   ;   OR   #imm, R0
11   ;   XOR  Rm, Rn
12   ;   XOR  #imm, R0
13   ;   TST  Rm, Rn
14   ;   TST  #imm, R0
15   ;
16   ; Revision History:
17   ;   11 May 2025    Zack Huang    Initial revision.
18
19   ProgramStart:
20       ; Test AND
21       MOV #$3, R0
22       MOV #$5, R1
23       AND R1, R0      ; set R0 <- 3 & 5
24       MOV #0, R1
25       MOV R0, @R1     ; expect R0 <- 1
26
27       ; Test AND immediate
28       MOV #$E, R0
29       AND #$5, R0     ; set R0 <- 0xE & 5
30       MOV #4, R1
31       MOV R0, @R1     ; expect R0 <- 4
32
33       ; Test OR
34       MOV #$3, R0
35       MOV #$5, R1
36       OR R1, R0       ; set R0 <- 3 | 5
37       MOV #8, R1
38       MOV R0, @R1     ; expect R0 <- 7
39
40       ; Test OR immediate
41       MOV #$E, R0
42       OR #$5, R0      ; set R0 <- 0xE | 5
43       MOV #12, R1
44       MOV R0, @R1     ; expect R0 <- 0xF
45
46       ; Test XOR
47       MOV #$3, R0
48       MOV #$5, R1
49       XOR R1, R0      ; set R0 <- 3 ^ 5
50       MOV #16, R1
51       MOV R0, @R1     ; expect R0 <- 6
52
53       ; Test XOR immediate
54       MOV #$E, R0
55       XOR #$5, R0
56       MOV #20, R1     ; set R0 <- 0xE ^ 5
57       MOV R0, @R1     ; expect R0 <- 0xB
58
59       ; Test TST
60       MOV #$2, R0
61       MOV #$1, R1
62       TST R1, R0      ; test 2 & 1
63       STC SR, R0      ; store SR to R0
64       AND #1, R0      ; mask to get T flag
65       MOV #24, R1
66       MOV R0, @R1     ; expect R0 <- 1
```

1

```
67
68      MOV #$2, R0
69      MOV #$2, R1
70      TST R1, R0      ; test 2 & 2
71      STC SR, R0      ; store SR to R0
72      AND #1, R0      ; mask to get T flag
73      MOV #28, R1
74      MOV R0, @R1     ; expect R0 <- 0
75
76      ; Test TST immediate
77      MOV #$2, R0
78      TST #1, R0      ; test 2 & 1
79      STC SR, R0      ; store SR to R0
80      AND #1, R0      ; mask to get T flag
81      MOV #32, R1
82      MOV R0, @R1     ; expect R0 <- 1
83
84      MOV #$2, R0
85      TST #2, R0      ; test 2 & 2
86      STC SR, R0      ; store SR to R0
87      AND #1, R0      ; mask to get T flag
88      MOV #36, R1
89      MOV R0, @R1     ; expect R0 <- 0
90
91      ; Test NOT
92      MOV #-1, R0
93      XOR #$FF, R0    ;   R0 <- 0xFFFFFF00
94      NOT R0, R0
95      MOV #40, R1
96      MOV R0, @R1     ;   Expect R0 = 0x000000FF
97
98      ; Quit test program
99      MOV #-4, R0
100     MOV.B R0, @R0
101
```

```
 1   00000000 00000001
 2   00000004 00000004
 3   00000008 00000007
 4   0000000C 0000000F
 5   00000010 00000006
 6   00000014 0000000B
 7   00000018 00000001
 8   0000001C 00000000
 9   00000020 00000001
10   00000024 00000000
11   00000028 000000FF
12
```

```
1   ; mov_at_disp_gbr_r0.asm
2   ;
3   ; This file tests the following instructions:
4   ;     MOV.B @(disp, GBR), R0
5   ;     MOV.W @(disp, GBR), R0
6   ;     MOV.L @(disp, GBR), R0
7   ;
8   ; This test writes 0x11223344 at address 0x10 and uses the instructions under
9   ; test to read the contents into R0. The byte, word, and longword are written
10  ; to 0x00, 0x04, and 0x08 respectively.
11  ;
12  ; Revision History:
13  ;   25 May 2025  Chris M.  Initial revision.
14  ;
15  Start:
16
17      MOV   ZeroVar, R0 ; Zero out 0x00, 0x04, and 0x08
18
19      MOV   #$00, R1
20      MOV   R0,   @R1
21
22      MOV   #$04, R1
23      MOV   R0,   @R1
24
25      MOV   #$08, R1
26      MOV   R0,   @R1
27
28      MOV   Var,  R0  ; Read 0x11223344 into R0. Note that this is converted to
29                      ; a PC-relative move by the assembler.
30
31      MOV   #$10, R1  ; Write at 0x10
32      MOV   R0,   @R1
33
34
35      MOV   #$4,  R0  ; Load 0x04 into GBR
36      LDC   R0,   GBR
37
38      MOV.B @($0C,GBR), R0 ; 0x0C + 0x04 = 0x10, our target address.
39                          ; We expect to read 0x11
40
41      MOV   #$00, R1      ; Write read memory contents at 0x00
42      MOV   R0,   @R1
43
44
45      MOV.W @($0C,GBR), R0 ; We expect to read 0x1122
46
47      MOV   #$04, R1      ; Write read memory contents at 0x04
48      MOV   R0,   @R1
49
50      MOV.L @($0C,GBR), R0 ; We expect to read 0x11223344
51
52      MOV   #$08, R1      ; Write read memory contents at 0x08
53      MOV   R0,   @R1
54
55      ; Expected memory
56      ; 00000000 00000011
57      ; 00000004 00001122
58      ; 00000008 11223344
59
60  End:
61      ; The test bench interprets a read of 0xFFFFFFFC (-4)
62      ;as system exit.
63      MOV #-4, R0;
64      MOV.B R0, @R0;
65
66
```

```
67
68     align 4 ; long-word align program memory.
69     LTORG
70
71 ; Long-word constants (in program memory).
72 ;
73 ZeroVar:  dc.l $00000000
74 Var:      dc.l $11223344
75
```

```
1  00000000 00000011
2  00000004 00001122
3  00000008 11223344
4
```

```
1   ; mov_b_at_r0_rm_rn.asm
2   ;
3   ; This file tests the following instructions:
4   ;   MOV.B @(R0, Rm), Rn
5   ;   MOV.W @(R0, Rm), Rn
6   ;   MOV.L @(R0, Rm), Rn
7   ;
8   ; It assumes that moving into memory works.
9   ;
10  ; Revision History:
11  ;   19 May 2025  Chris M.  Initial revision.
12
13  Start:
14      ; Write 0x1234567 at 0x14. Read byte, word, and longword from 0x14 using
15      ; instructions under test and write the result at 0x00, 0x04, and 0x08
16      ; respectively.
17
18
19      MOV ZeroVar, R0
20
21      MOV   #$00, R1    ; Zero out 0x00, 0x04, and 0x08
22      MOV   R0,   @R1
23
24      MOV   #$04, R1
25      MOV   R0,   @R1
26
27      MOV   #$08, R1
28      MOV   R0,   @R1
29
30      MOV   Var,  R0  ; Write 0x12345678 at 0x14
31      MOV   #$14, R1
32      MOV   R0,   @R1
33
34
35      ; Test MOV.B @(R0, Rm), Rn
36
37      MOV     #$0C,    R0  ; 0x0C + 0x8 = 0x14, our target address.
38      MOV     #$08,    R1
39      MOV.B   @(R0,R1), R2
40
41      MOV   #$00, R3       ; We expect signExtend(0x12) = 0x00000012 at 0x00
42      MOV   R2,   @R3
43
44      ; Test MOV.W @(R0, Rm), Rn
45
46      MOV     #$0A,    R0  ; 0x0A + 0x0A = 0x14, our target address.
47      MOV     #$0A,    R1
48      MOV.W   @(R0,R1), R2
49
50      MOV   #$04, R3       ; We expect signExtend(0x1234) = 0x00001234 at 0x00
51      MOV   R2,   @R3
52
53      ; Test MOV.L @(R0, Rm), Rn
54
55      MOV     #$10,    R0  ; 0x10 + 0x04 = 0x14, our target address.
56      MOV     #$04,    R1
57      MOV.L   @(R0,R1), R2
58
59      MOV   #$08, R3       ; We expect 0x12345678 at 0x00
60      MOV   R2,   @R3
61
62      ; Expected memory layout:
63
64      ;00000000 12000000
65      ;00000004 12340000
66      ;00000008 12345678
```

1

```
67
68  End:
69      ; The test bench interprets a read of 0xFFFFFFFC (-4)
70      ;as system exit.
71      MOV #-4, R0;
72      MOV.B R0, @R0;
73
74        align 4
75        LTORG
76
77  ZeroVar:  dc.l $00000000
78  Var:      dc.l $12345678
79
```

```
1  00000000 00000012
2  00000004 00001234
3  00000008 12345678
4
```

```
1   ; mov_bwl_at_disp_rm_r0_or_rn.asm
2   ;
3   ; This file tests the following instructions:
4   ;    MOV.B @(disp, Rm), R0
5   ;    MOV.W @(disp, Rm), R0
6   ;    MOV.L @(disp, Rm), Rn
7   ;
8   ; It assumes that moving into memory works. We write 0xDEADBEEF at address
9   ; 0x10 and then use the move instruction under test to move a byte, word, and
10  ; longword of this into a register. The byte is written to 0x00, word to
11  ; 0x04, and longword to 0x08.
12  ;
13  ; Revision History:
14  ;   19 May 2025  Chris M.  Initial revision.
15
16  Start:
17
18      MOV   Var,  R0  ; Write 0xDEADBEEF to 0x10
19      MOV   #$10, R1
20      MOV   R0,   @R1
21
22
23      ; Test MOV.B @(disp, Rm), R0
24
25      MOV    #$0C,     R1
26      MOV.B  @(4,R1),  R0  ;0x0C + 0x4 = 0x10, our target address.
27
28      MOV   #$00,   R1    ; The low byte of 0xDEADBEEF is 0xDE. Sign-extended
29                         ; into a register this is 0xFFFFFFDE, which we expect
30                         ; at 0x00.
31
32      MOV   R0,   @R1
33
34
35
36      ; Test MOV.W @(disp, Rm), R0
37
38      MOV    #$0A,     R1
39      MOV.W  @(6,R1),  R0 ; 0x0A + 0x06 = 0x10, our target address.
40
41      MOV   #$04, R1     ; The low word of 0xDEADBEEF is 0xDEAD. Sign-extended
42                         ; into a register this is is 0xFFFFDEAD, which we expect
43                         ; at 0x04
44
45      MOV   R0,   @R1
46
47
48
49      ; Test MOV.L @(disp, Rm), Rn
50
51      MOV    #$08,     R1
52      MOV.L  @(8,R1),  R2 ; 0x08 + 0x08 = 0x10, our target address.
53
54      MOV   #$08,   R1  ; We expect 0xDEADBEEF at 0x08.
55      MOV   R2,     @R1
56
57
58      ; Expected memory layout:
59
60      ; 00000000 FFFFFFDE
61      ; 00000004 FFFFDEAD
62      ; 00000008 DEADBEEF
63
64  End:
65    ; The test bench interprets a read of 0xFFFFFFFC (-4)
66    ;as system exit.
```

1

```
67    MOV #-4, R0;
68    MOV.B R0, @R0;
69
70
71      align 4
72      LTORG
73
74 ZeroVar:  dc.l $00000000
75 Var:      dc.l $DEADBEEF
76
```

```
1  00000000 FFFFFFDE
2  00000004 FFFFDEAD
3  00000008 DEADBEEF
4
```

```
1   ; mov_bwl_at_rm_plus_rn.asm
2   ;
3   ; This file tests the following instructions:
4   ;    MOV.B @Rm+, Rn
5   ;    MOV.W @Rm+, Rn
6   ;    MOV.L @Rm+, Rn
7   ;
8   ; It assumes that moving into memory works.
9   ;
10  ; Revision History:
11  ;   16 May 2025  Chris M.  Initial revision.
12
13  Start:
14      ; We set up the memory as follows:
15      ;
16      ; 78: 12 34 56 78
17
18      MOV   #$12, R1  ;0x12 at 0x78
19      MOV   #$78, R0
20      MOV.B R1,   @R0
21
22      MOV   #$34, R1  ;0x34 at 0x79
23      MOV   #$79, R0
24      MOV.B R1,   @R0
25
26      MOV   #$56, R1  ;0x56 at 0x7A
27      MOV   #$7A, R0
28      MOV.B R1,   @R0
29
30      MOV   #$78, R1  ;0x78 at 0x7B
31      MOV   #$7B, R0
32      MOV.B R1,   @R0
33
34      ;Test instructions.
35
36      MOV   #$78, R0  ; Start reading at 0x78
37      MOV.B @R0+, R1  ; Read 0x12 into R1.
38                     ; Address in R0 should be 0x79 after this.
39
40      MOV   #$00, R2  ; Move 0x00000012 (R1) @ 0x00000000.
41      MOV.L R1,   @R2  ;
42
43      MOV   #$04, R2  ; Move 0x00000079 (R0) @ 0x00000004.
44      MOV.L R0,   @R2
45
46
47      MOV  #$78, R0   ; Start reading at 0x78
48      MOV.W @R0+, R1  ; Read 0x1234 into R1.
49                     ; Address in R0 should be 0x7A after this
50
51      MOV   #$08, R2  ; Move 0x00001234 @ 0x00000008.
52      MOV.L R1,   @R2  ;
53
54      MOV   #$0C, R2  ; Move 0x0000007A (R0) @ 0x0000000C.
55      MOV.L R0,   @R2
56
57      MOV  #$78, R0   ; Start reading at 0x78
58      MOV.L @R0+, R1  ; Read 0x12345678 into R1.
59                     ; Address in R0 should be 0x7C after this
60
61      MOV   #$10, R2  ; Move 0x12345678 @ 0x00000010.
62      MOV.L R1,   @R2  ;
63                     ;
64
65      MOV   #$14, R2  ; Move 0x0000007C (R0) @ 0x00000014.
66      MOV.L R0,   @R2
```

```
67
68
69      ; Expected memory
70      ; 00000000: 00000012
71      ; 00000004: 00000079
72      ; 00000008: 00001234
73      ; 0000000C: 0000007A
74      ; 00000010: 12345678
75      ; 00000014: 0000007C
76
77  End:
78      ; The test bench interprets a read of 0xFFFFFFFC (-4)
79      ;as system exit.
80      MOV #-4, R0;
81      MOV.B R0, @R0;
82
```

```
1  00000000 00000012
2  00000004 00000079
3  00000008 00001234
4  0000000C 0000007A
5  00000010 12345678
6  00000014 0000007C
7
```

```
 1   ; mov_bwl_at_rm_rn.asm
 2   ;
 3   ; This file tests the following instructions:
 4   ;    MOV.B @Rm, Rn
 5   ;    MOV.W @Rm, Rn
 6   ;    MOV.L @Rm, Rn
 7   ;
 8   ; It assumes that moving into memory works. The memory is organized as follows:
 9   ;
10   ;   78: 11 22 33 44
11   ;
12   ; Revision History:
13   ;   13 May 2025  Chris M.  Initial revision.
14
15   Start:
16     MOV    #$11,  R1   ; 0x11 at address 0x78
17     MOV    #$78,  R0
18     MOV.B  R1,   @R0
19
20     MOV    #$22,  R1   ; 0x22 at address 0x79
21     MOV    #$79,  R0
22     MOV.B  R1,   @R0
23
24     MOV    #$33,  R1   ; 0x33 at address 0x7A
25     MOV    #$7A,  R0
26     MOV.B  R1,   @R0
27
28     MOV    #$44,  R1   ; 0x44 at address 0x7B
29     MOV    #$7B,  R0
30     MOV.B  R1,   @R0
31
32
33     MOV   #$78, R0
34     MOV.B @R0, R1     ; Move byte from 0x78 to R1
35
36     MOV.W @R0, R2     ; Move word from 0x78 to R2
37
38     MOV.L @R0, R3     ; Move longword from 0x78 to R2
39
40
41     MOV   #$00, R0    ; 0x00000011 expected at 0x00
42     MOV.L R1, @R0
43
44     MOV   #$04, R0    ; 0x00001122 expected at 0x04
45     MOV.L R2, @R0
46
47     MOV   #$08, R0    ; 0x11223344 expected at 0x08
48     MOV.L R3, @R0
49
50   End:
51     ; The test bench interprets a read of 0xFFFFFFFC (-4)
52     ;as system exit.
53     MOV #-4, R0;
54     MOV.B R0, @R0;
55
```

```
1  00000000 00000011
2  00000004 00001122
3  00000008 11223344
4
```

```
1   ; mov_bwl_at_disp_gbr.asm
2   ;
3   ; This file tests the following instructions:
4   ;   MOV.B R0, @(disp, GBR)
5   ;   MOV.W R0, @(disp, GBR)
6   ;   MOV.L R0, @(disp, GBR)
7   ;
8   ; Revision History:
9   ;   24 May 2025  Chris M.  Initial revision.
10  ;
11  Start:
12      ; Use the instructions under test to move to 0x20, 0x24, and 0x28.
13      ; Zero out locations that don't move longwords to avoid unitialized values.
14
15      MOV    ZeroVar, R0
16
17      MOV    #$20, R1     ; Zero out 0x20
18      MOV    R0,   @R1
19
20      MOV    #$24, R1     ; Zero out 0x24
21      MOV    R0,   @R1
22
23      MOV    #$28, R1     ; Zero out 0x28
24      MOV    R0,   @R1
25
26
27      MOV    #$10, R0        ; Load 0x10 into the GBR
28      LDC    R0,   GBR
29
30      MOV    Var, R0       ; Load 0xDEADBEEF into R0
31
32      MOV.B  R0, @($10,GBR) ; Move 0xEF into 0x20.
33
34      MOV.W  R0, @($14,GBR) ; Move 0xBEEF into 0x24.
35
36      MOV.L  R0, @($18,GBR) ; Move 0xDEADBEEF into 0x28.
37
38      ; Expected memory layout
39      ;
40      ; 00000020 FE000000
41      ; 00000024 BEEF0000
42      ; 00000028 DEADBEEF
43
44  End:
45    ; The test bench interprets a read of 0xFFFFFFFC (-4)
46    ;as system exit.
47    MOV #-4, R0;
48    MOV.B R0, @R0;
49
50      align 4
51      LTORG
52
53  ZeroVar:  dc.l $00000000
54  Var:      dc.l $DEADBEEF
55
56
57
```

```
1  00000020 EF000000
2  00000024 BEEF0000
3  00000028 DEADBEEF
4
```

```
1   ; mov_bwl_r0_or_rm_at_disp_rn.asm
2   ;
3   ; This file tests the following instructions:
4   ;     MOV.B R0, @(disp, Rn)
5   ;     MOV.W R0, @(disp, Rn)
6   ;     MOV.L Rm, @(disp, Rn)
7   ;
8   ; It assumes that moving into memory and move with PC relative addressing
9   ; works. Move with PC relative addressing is used to load long-word constants
10  ; into registers.
11  ;
12  ; Revision History:
13  ;   16 May 2025  Chris M.  Initial revision.
14
15  Start:
16
17      ; Zero out words at 0x08, 0x0C, and 0x14
18      MOV ZeroVar, R0
19
20      MOV #$08, R1
21      MOV R0, @R1
22
23      MOV #$0C, R1
24      MOV R0, @R1
25
26      MOV #$14, R1
27      MOV R0, @R1
28
29      MOV Var, R0 ; Move 0x12345678 into R0
30
31      MOV #04, R1
32      MOV.B R0, @(4,R1)    ; Target address = 0x04 + zeroExtend(0x04) = 0x08
33                          ; We expected 0x78 at 0x08
34
35      MOV #04, R1
36      MOV.W R0, @(8,R1)    ; Target address = 0x04 + zeroExtend(0x08) = 0x0C
37                          ; We expect 0x5678 at 0x0C
38
39      MOV Var, R1 ; Move 0x12345678 into R1
40
41      MOV #$04, R2
42      MOV.L R1, @(16,R2)   ; Target address = 0x04 + zeroExtend(0x16) = 0x14
43                          ; We expect 0x12345678 at 0x14
44
45      ; 00000008 78000000
46      ; 0000000A 78560000
47      ; 00000014 12345678
48
49  Done:
50      ; The test bench interprets a read of 0xFFFFFFFC (-4)
51      ; as system exit.
52      MOV   #-4, R0;
53      MOV.B R0, @R0;
54
55      align 4
56      LTORG
57
58  Var: dc.w $1234
59      dc.w $5678
60
61  ZeroVar: dc.w $0000
62          dc.w $0000
63
64
```

```
1  00000008 78000000
2  0000000C 56780000
3  00000014 12345678
4
```

```
1   ; mov_bwl_rm_at_minus_rn.asm
2   ;
3   ; This file tests the following instructions:
4   ;    MOV.B Rm, @-Rn
5   ;    MOV.W Rm, @-Rn
6   ;    MOV.L Rm, @-Rn
7   ;
8   ; It assumes that moving into memory works.
9   ;
10  ; Revision History:
11  ;   14 May 2025  Chris M.  Initial revision.
12
13  Start:
14    MOV    #$11,  R1   ; 0x11 at address 0x78
15    MOV    #$78,  R0
16    MOV.B  R1,    @R0
17
18    MOV    #$22,  R1   ; 0x22 at address 0x79
19    MOV    #$79,  R0
20    MOV.B  R1,    @R0
21
22    MOV    #$33,  R1   ; 0x33 at address 0x7A
23    MOV    #$7A,  R0
24    MOV.B  R1,    @R0
25
26    MOV    #$44,  R1   ; 0x44 at address 0x7B
27    MOV    #$7B,  R0
28    MOV.B  R1,    @R0
29
30    MOV   #$78, R0     ; Move 11223344 into R1
31    MOV.L @R0,  R1
32
33
34    MOV   #$01,  R0    ; Write byte from R1 (0x44) into address 0x00. Note that
35                      ; R0 is pre-decremented by 1.
36    MOV.B  R1, @-R0
37
38
39    MOV  #$01, R0      ; Write 0x00 at 0x01 since the proceeding word write must
40                      ; be word aligned.
41    MOV  #$00, R2
42    MOV.B R2, @R0
43
44
45    MOV   #$04,  R0    ; Write word from R1 (0x3344) into address 0x02. Note
46                      ; that R0 is pre-decremented by 2.
47    MOV.W  R1, @-R0
48
49
50    MOV   #$08,  R0    ; Write longword from R1 (0x11223344) into address 0x04. Note
51                      ; that R0 is pre-decremented by 4.
52    MOV.L  R1, @-R0
53
54    ; Expected memory layout
55    ;
56    ; 00000000  44003344
57    ; 00000004  11223344
58
59  End:
60    ; The test bench interprets a read of 0xFFFFFFFC (-4)
61    ;as system exit.
62    MOV #-4, R0;
63    MOV.B R0, @R0;
64
```

```
1  00000000  44003344
2  00000004  11223344
3
```

```
1   ; mov_Reg.asm
2   ;
3   ; Tests simple register MOVs
4   ;
5   ; This file tests the following instructions:
6   ;    MOV  Rm, Rn
7   ;
8   ; Revision History:
9   ;   11 May 2025    Zack Huang     Initial revision.
10
11  ProgramStart:
12      ; Move constant through each register
13      MOV #15, R0;
14      MOV R0, R1;
15      MOV R1, R2;
16      MOV R2, R3;
17      MOV R3, R4;
18      MOV R4, R5;
19      MOV R5, R6;
20      MOV R6, R7;
21      MOV R7, R8;
22      MOV R8, R9;
23      MOV R9, R10;
24      MOV R10, R11;
25      MOV R11, R12;
26      MOV R12, R13;
27      MOV R13, R14;
28      MOV R14, R15;
29
30      ; Write contents of R15 to memory
31      MOV #0, R0;
32      MOV R15, @R0    ; expect R0 <- 15
33
34      ; Quit test program
35      MOV #-4, R0;
36      MOV.B R0, @R0;
37
```

```
1  00000000 0000000F
2
```

```
1   ; mov_rm_at_r0_rn.asm
2   ;
3   ; This file tests the following instructions:
4   ;    MOV.B Rm, @(R0, Rn)
5   ;    MOV.W Rm, @(R0, Rn)
6   ;    MOV.L Rm, @(R0, Rn)
7   ;
8   ; Revision History:
9   ;   19 May 2025  Chris M.  Initial revision.
10
11  Start:
12      ; We will move a byte, word, and longword of 0x11223344 into 0x08, 0x0C,
13      ; and 0x10 respectively. This range will be zeroed out to prevent "X"s.
14
15      MOV ZeroVar, R2 ; Zero out 0x00, 0x04, and 0x08
16
17      MOV   #$08, R0
18      MOV   R2,   @R0
19
20      MOV   #$0C, R0
21      MOV   R2,   @R0
22
23      MOV   #$10, R0
24      MOV   R2,   @R0
25
26
27      MOV   Var, R2
28
29      ; Test MOV.B Rm, @(R0, Rn)
30
31      MOV #$02, R0    ; Our target address is 0x08 (8) and 0x02 + 0x06 = 0x08
32      MOV #$06, R1
33
34      MOV.B R2, @(R0,R1) ; We expect 0x44 at 0x08
35
36      ; Test MOV.W Rm, @(R0, Rn)
37
38      MOV #$08, R0    ; Our target address is 0x0C (12) and 0x08 + 0x04  = 0x0C
39      MOV #$04, R1
40
41      MOV.W R2, @(R0,R1) ; We expect 0x3344 at 0x0C
42
43      ; Test MOV.L Rm, @(R0, Rn)
44
45      MOV #$0C, R0    ; Our target address is 0x10 (16) and 0x0C + 0x04  = 0x10
46      MOV #$04, R1
47
48      MOV.L R2, @(R0,R1) ; We expect 0x11223344 at 0x10
49
50      ; Expected memory layout
51      ; 00000008  44000000
52      ; 0000000C  33440000
53      ; 00000010  11223344
54
55  End:
56      ; The test bench interprets a read of 0xFFFFFFFC (-4)
57      ;as system exit.
58      MOV #-4, R0;
59      MOV.B R0, @R0;
60
61      align 4
62      LTORG
63
64  ZeroVar:  dc.l $00000000
65  Var:      dc.l $11223344
66
```

```
1  00000008  44000000
2  0000000C  33440000
3  00000010  11223344
4
```

```
1   ; mov_wl_at_disp_pc_rn.asm
2   ;
3   ; This file tests the following instructions:
4   ;    MOV.W @(disp, PC), Rn
5   ;    MOV.L @(disp, PC), Rn
6   ;
7   ; It assumes that moving into memory works. The MOV.W and MOV.L instructions
8   ; must be hand assembled.
9   ;
10  ; Revision History:
11  ;   12 May 2025  Chris M.  Initial revision.
12  ;
13  ; TODO:
14  ;  - Use assembly directives to load 32-bit constants instead of MOV #imm, Rn
15  ;
16
17  ProgramStart:
18    ;MOV #imm, Rn is limited from -128 to 127. immediates are sign-extended.
19
20    MOV   #127, R1  ; PC is 0x00                (E1 7F)
21    MOV   #$7A, R0  ; PC is 0x02                (E0 7A)
22    MOV.B R1, @R0   ; PC is 0x04                (20 10)
23
24    MOV   #126, R1  ; PC is 0x06                (E1 7E)
25    MOV   #$7B, R0  ; PC is 0x08                (E0 7B)
26    MOV.B R1, @R0   ; PC is 0x0A                (20 10)
27
28    MOV   #125, R1  ; PC is 0x0C                (E1 7D)
29    MOV   #$7C, R0  ; PC is 0x0E                (E0 7C)
30    MOV.B R1, @R0   ; PC is 0x10                (20 10)
31
32    MOV   #124, R1  ; PC is 0x12                (E1 7C)
33    MOV   #$7D, R0  ; PC is 0x14                (E0 7D)
34    MOV.B R1, @R0   ; PC is 0x16                (20 10)
35
36    MOV.W Var, R2
37
38    MOV Var, R3;
39
40    ; Move R2 and R3 into memory to check for correctness.
41
42    MOV   #0, R0  ; 0x00007F7E expected at 0x00000000
43    MOV.L R2, @R0 ;
44
45    MOV.L #4, R0  ; 0x7F7E7D7C expected at 0x00000004
46    MOV.L R3, @R0 ;
47
48  Done:
49    ; The test bench interprets a read of 0xFFFFFFFC (-4)
50    ;as system exit.
51    MOV   #-4, R0;
52    MOV.B R0, @R0;
53
54      align 4
55      LTORG
56  ; Var: dc.l $12345678
57  Var: dc.w $1234
58      dc.w $5678
59
```

1

```
1  00000000 00001234
2  00000004 12345678
3
```

```
1   ; mova_at_disp_pc_r0.asm;
2   ;
3   ; This file tests the following instruction:
4   ;     MOVA @(disp, PC), R0
5   ;
6   ; Revision History:
7   ;   25 May 2025  Chris M.  Initial revision.
8   Start:
9
10      ; Usage: MOV <var name>, R0
11      ; NOTE: Can't do MOVA @(disp, PC), R0 directly
12
13      ; 00000000: c702  ..  MOVA   @(disp, PC), R0
14      ; 00000002: e100  ..  MOV    #00,        R1
15      ; 00000004: 2102  !.  MOV    R0,         @R1
16      ; 00000006: e0fc  ..  MOV    #-4,         R0
17      ; 00000008: 2000   .  MOV.B  R0,         @R0
18      ; 0000000a: ffff  ..  .align 4
19      ; 0000000c: 0000  ..  ZeroVar
20      ; 0000000e: 0000  ..  ZeroVar
21      ; 00000010: 1122  ."  Var
22      ; 00000012: 3344  3D  Var
23
24
25      MOVA Var, R0    ; PC = 0x00. Since Var starts at ROM address 0x10, we expect
26                      ; 0x10 in R0
27
28      MOV   #00,  R1  ; PC = 0x02
29      MOV   R0,   @R1 ; PC = 0x04
30
31      ; Expected memory:
32      ; 00000000 00000010
33
34  End:
35      ; The test bench interprets a read of 0xFFFFFFFC (-4)
36      ;as system exit.
37      MOV #-4, R0;      ; PC = 0x06
38      MOV.B R0, @R0;    ; PC = 0x08
39
40
41      ; long-word align program memory.
42      align 4         ; PC = 0x0A
43      LTORG
44
45  ; Long-word constants (in program memory).
46  ;
47  ZeroVar:  dc.l $00000000  ; PC = 0x0C (for 0x0000), PC = 0x0E (for 0x0000)
48  Var:      dc.l $11223344  ; PC = 0x10 (for 0x1122), PC = 0x12 (for 0x3344)
49
```

```
1  00000000 00000010
2
```

```
1   ; movt_rn.asm
2   ;
3   ; This file tests the following instruction:
4   ;     MOVT Rn
5   ;
6   ; Revision History:
7   ;   25 May 2025  Chris M.  Initial revision.
8   ;
9   Start:
10
11  ; Example:
12  ;
13  ;     XOR     R2,R2 ; R2 = 0
14  ;     CMP/PZ  R2    ; T = 1
15  ;     MOVT    R0    ; R0 = 1
16  ;     CLRT          ; T = 0
17  ;     MOVT    R1    ; R1 = 0
18
19      MOV     #0,   R2
20      XOR     R2,   R2  ; Anything XORed with itself is zero
21      CMP/PZ  R2        ; Compare to zero sets T=1 because the result is zero.
22
23      MOVT    R0        ; Write T to 0x00. We expect 0x00000001 at 0x00
24      MOV     #$00, R1
25      MOV     R0,   @R1
26
27      CLRT              ; Clear T flag.
28
29      MOVT    R0        ; Write T to 0x04. We expect 0x00000000 at 0x04
30      MOV     #$04, R1
31      MOV     R0,   @R1
32
33      ; Expected memory:
34      ; 00000000 00000001
35      ; 00000004 00000000
36
37  End:
38    ; The test bench interprets a read of 0xFFFFFFFC (-4)
39    ; as system exit.
40    MOV #-4, R0;
41    MOV.B R0, @R0;
42
43
44      align 4
45      LTORG
46
47  ZeroVar:  dc.l $00000000
48  Var:      dc.l $DEADBEEF
49
```

1

```
1  00000000 00000001
2  00000004 00000000
3
```

```
 1   ; reg_indirect.asm
 2   ;
 3   ; Testing indirect register addressing
 4   ;
 5   ; This file tests the following instructions:
 6   ;     MOV.B Rm, @Rn
 7   ;     MOV.W Rm, @Rn
 8   ;     MOV.L Rm, @Rn
 9   ;
10   ; Revision History:
11   ;   11 May 2025     Zack Huang      Initial revision.
12   ;   14 May 2025     Zack Huang      Ensure endianness is consistent
13
14   ProgramStart:
15       ; Write 0A0B0C0D to 0x00000000 using byte mode addressing
16       ; Note, for the purposes of testing, all longwords in memory are
17       ; interpreted as being big-endian, which means that the CPU will read this
18       ; as 0D0C0B0A.
19       MOV #0, R0;
20       MOV #$D, R1;
21       MOV.B R1, @R0;
22
23       MOV #1, R0;
24       MOV #$C, R1;
25       MOV.B R1, @R0;
26
27       MOV #2, R0;
28       MOV #$B, R1;
29       MOV.B R1, @R0;
30
31       MOV #3, R0;
32       MOV #$A, R1;
33       MOV.B R1, @R0;
34
35       ; Write 00120034 to 0x00000004 using word mode
36       ; Note, for the purposes of testing, all longwords in memory are
37       ; interpreted as being big-endian, which means that the CPU will read this
38       ; as 00340012.
39       MOV #$F, R1;
40       ADD #$3, R1;
41       MOV #6, R0;
42       MOV.W R1, @R0;
43
44       ADD #$F, R1;
45       ADD #$F, R1;
46       ADD #$4, R1;
47       MOV #4, R0;
48       MOV.W R1, @R0;
49
50       ; Write FFFFFFFF to 0x00000008 using longword mode
51       MOV #8, R0;
52       MOV #-1, R1;
53       MOV.L R1, @R0;
54
55       ; Quit test program
56       MOV #-4, R0;
57       MOV.B R0, @R0;
58
59
```

```
1  00000000 0D0C0B0A
2  00000004 00340012
3  00000008 FFFFFFFF
4
```

```
1    ; shift.asm
2    ;
3    ; Testing ALU shift operations
4    ;
5    ; This file tests the following instructions:
6    ;    ROTL  Rn
7    ;    ROTR  Rn
8    ;    ROTCL Rn
9    ;    ROTCR Rn
10   ;    SHAL  Rn
11   ;    SHAR  Rn
12   ;    SHLL  Rn
13   ;    SHLR  Rn
14   ;
15   ; Revision History:
16   ;    12 May 2025    Zack Huang      Initial revision.
17
18   ProgramStart:
19       ; ROTL
20       MOV #-1, R0;    R0 <- 0xFFFFFFFF
21       XOR #1,  R0;    R0 <- 0xFFFFFFFE
22       ROTL R0;
23       ROTL R0;
24       ROTL R0;
25       ROTL R0;        Rotate R0 left one byte
26       MOV #0, R1;
27       MOV R0, @R1;    Expect: R0 = 0xFFFFFFEF
28
29       STC SR, R0;
30       AND #1, R0;
31       MOV #4, R1;
32       MOV R0, @R1;    Expect: T = 1
33
34       ; ROTR
35       MOV #-1, R0;    R0 <- 0xFFFFFFFF
36       XOR #1,  R0;    R0 <- 0xFFFFFFFE
37       ROTR R0;
38       ROTR R0;
39       ROTR R0;
40       ROTR R0;        Rotate R0 right one byte
41       MOV #8, R1;
42       MOV R0, @R1;    Expect: R0 = 0xEFFFFFFF
43
44       STC SR, R0;
45       AND #1, R0;
46       MOV #12, R1;
47       MOV R0, @R1;    Expect: T = 1
48
49       ; ROTCL
50       MOV #-1, R0;    R0 <- 0xFFFFFFFF
51       CLRT;
52       ROTCL R0;
53       ROTCL R0;
54       ROTCL R0;
55       ROTCL R0;
56       MOV #16, R1;
57       MOV R0, @R1;    Expect: R0 = 0xFFFFFFF7
58
59       STC SR, R0;
60       AND #1, R0;
61       MOV #20, R1;
62       MOV R0, @R1;    Expect: T = 1
63
64       ; ROTCR
65       MOV #-1, R0;    R0 <- 0xFFFFFFFF
66       CLRT;
```

1

```
67       ROTCR R0;
68       ROTCR R0;
69       ROTCR R0;
70       ROTCR R0;
71       MOV #24, R1;
72       MOV R0, @R1;     Expect: R0 = 0xEFFFFFFF
73
74       STC SR, R0;
75       AND #1, R0;
76       MOV #28, R1;
77       MOV R0, @R1;     Expect: T = 1
78
79       ; SHLL/SHAL
80       MOV #1, R0;
81       SHAL R0;
82       SHLL R0;
83       SHAL R0;
84       SHLL R0;
85       MOV #32, R1;
86       MOV R0, @R1;     Expect: R0 = 0x00000010
87
88       STC SR, R0;
89       AND #1, R0;
90       MOV #36, R1;
91       MOV R0, @R1;     Expect: T = 0
92
93       ; SHAR
94       MOV #-1, R0;
95       XOR #$80, R0;    R0 <- FFFFFF7F
96       SHAR R0;
97       SHAR R0;
98       SHAR R0;
99       SHAR R0;
100      MOV #40, R1;
101      MOV R0, @R1;     Expect: R0 = 0xFFFFFFF7
102
103      STC SR, R0;
104      AND #1, R0;
105      MOV #44, R1;
106      MOV R0, @R1;     Expect: T = 1
107
108      ; SHLR
109      MOV #-1, R0;
110      AND #$F0, R0;    R0 <- 000000F0
111      SHLR R0;
112      SHLR R0;
113      SHLR R0;
114      SHLR R0;
115      MOV #48, R1;
116      MOV R0, @R1;     Expect: R0 = 0x0000000F
117
118      STC SR, R0;
119      AND #1, R0;
120      MOV #52, R1;
121      MOV R0, @R1;     Expect: T = 0
122
123      ; Quit test program
124      MOV #-4, R0;
125      MOV.B R0, @R0;
126
```

```
 1  00000000 FFFFFFEF
 2  00000004 00000001
 3  00000008 EFFFFFFF
 4  0000000C 00000001
 5  00000010 FFFFFFF7
 6  00000014 00000001
 7  00000018 EFFFFFFF
 8  0000001C 00000001
 9  00000020 00000010
10  00000024 00000000
11  00000028 FFFFFFF7
12  0000002C 00000001
13  00000030 0000000F
14  00000034 00000000
15
```

```
 1   ; sr.asm
 2   ;
 3   ; Testing status register commands
 4   ;
 5   ; This file tests the following instructions:
 6   ;    SETT
 7   ;    CLRT
 8   ;    STC Rm, SR
 9   ;
10   ; Revision History:
11   ;   11 May 2025    Zack Huang     Initial revision.
12
13   ProgramStart:
14       ; Set T bit
15       SETT             ; set T bit
16       STC SR, R0       ; store SR to R0
17       AND #1, R0       ; mask T bit
18       MOV #0, R1
19       MOV R0, @R1      ; expect R0 <- 1
20
21       ; Clear T bit
22       CLRT             ; clear T bit
23       STC SR, R0       ; store SR to R0
24       AND #1, R0       ; mask T bit
25       MOV #4, R1
26       MOV R0, @R1      ; expect R0 <- 0
27
28       ; Load A into SR
29       MOV #$A, R0
30       LDC R0, SR       ; load 0xA into SR
31       STC SR, R2       ; store SR to R2
32       MOV #8, R1
33       MOV R2, @R1      ; expect R2 <- 0xA
34
35       ; Quit test program
36       MOV #-4, R0
37       MOV.B R0, @R0
38
```

```
1  00000000 00000001
2  00000004 00000000
3  00000008 0000000A
4
```

```
1   ; mov_at_disp_gbr_r0.asm
2   ;
3   ; This file tests the following instructions:
4   ;     SWAP.B Rm, Rn
5   ;     SWAP.W Rm, Rn
6   ;
7   ;   SWAP.B swaps the upper and lower halves of the lower two bytes of Rm and moves
8   ;   them into Rn.
9   ;
10  ;   SWAP.W swaps the upper and lower words of Rm and moves them into Rn.
11  ;
12  ; Revision History:
13  ;   25 May 2025  Chris M.  Initial revision.
14  ;
15  Start:
16      MOV     Var,  R0
17
18      SWAP.B  R0,   R1    ; 0x12345678 -> 0x12347856
19
20      MOV     #$00, R2    ; Write result at address 0x00
21      MOV     R1,   @R2
22
23      SWAP.W  R0,   R1    ; 0x12345678 -> 0x56781234
24
25      MOV     #$04, R2    ; Write result at address 0x04
26      MOV     R1,   @R2
27
28      ; Expected memory:
29      ; 00000000 12347856
30      ; 00000004 56781234
31
32  End:
33      ; The test bench interprets a read of 0xFFFFFFFC (-4)
34      ;as system exit.
35      MOV #-4, R0;
36      MOV.B R0, @R0;
37
38
39
40      align 4 ; long-word align program memory.
41      LTORG
42
43  ; Long-word constants (in program memory).
44  ;
45  ZeroVar:  dc.l $00000000
46  Var:      dc.l $12345678
47
```

```
1  00000000 12347856
2  00000004 56781234
3
```

```
 1   ; system.asm
 2   ;
 3   ; Testing system register operations
 4   ;
 5   ; This file tests the following instructions:
 6   ;   LDC     Rm, SR
 7   ;   LDC     Rm, GBR
 8   ;   LDC     Rm, VBR
 9   ;   LDC.L @Rm+, SR
10   ;   LDC.L @Rm+, GBR
11   ;   LDC.L @Rm+, VBR
12   ;   STC     SR, Rn
13   ;   STC     GBR, Rn
14   ;   STC     VBR, Rn
15   ;   STC.L SR, @-Rn
16   ;   STC.L GBR, @-Rn
17   ;   STC.L VBR, @-Rn
18   ;
19   ; Revision History:
20   ;   12 May 2025    Zack Huang     Initial revision.
21   ;   16 May 2025    Zack Huang     add in .L variants
22
23   ProgramStart:
24       ; Test LDC
25       MOV #$F, R0;
26       LDC R0, SR;     SR  <- 0x0000000F
27
28       ADD #1, R0;
29       LDC R0, GBR;    GBR <- 0x00000100
30
31       ADD #1, R0;
32       LDC R0, VBR;    VBR <- 0x00000101
33
34       STC SR, R0;
35       MOV #$0, R1;
36       MOV R0, @R1;    Expect R0 = 0x0000000F
37
38       STC GBR, R0;
39       MOV #$4, R1;
40       MOV R0, @R1;    Expect R0 = 0x00000010
41
42       STC VBR, R0;
43       MOV #$8, R1;
44       MOV R0, @R1;    Expect R0 = 0x00000011
45
46       ; Test LDC.L
47       MOV #$70, R3;
48       MOV #$1A, R2;
49       MOV R2, @R3;    Set memory at 0x70 to be 0x1A
50
51       MOV #$74, R3;
52       MOV #$2B, R2;
53       MOV R2, @R3;    Set memory at 0x74 to be 0x2B
54
55       MOV #$78, R3;
56       MOV #$3C, R2;
57       MOV R2, @R3;    Set memory at 0x78 to be 0x3C
58
59       MOV #$70, R2;
60       LDC.L @R2+, SR;     Read SR from memory, increment R2
61       STC SR, R0;
62       MOV #$C, R1;
63       MOV R0, @R1;       Expect R0 = 0x000000AA
64
65       LDC.L @R2+, GBR;    Read GBR from memory, increment R2
66       STC GBR, R0;
```

```
67      MOV #$10, R1;
68      MOV R0, @R1;          Expect R0 = 0x000000BB
69
70      LDC.L @R2+, VBR;      read VBR from memory, increment R2
71      STC VBR, R0;
72      MOV #$14, R1;
73      MOV R0, @R1;          Expect R0 = 0x000000CC
74
75      ; Test STC.L
76      MOV #$4C, R0;
77      LDC R0, SR;      SR  <- 0x0000004C
78
79      ADD #1, R0;
80      LDC R0, GBR;     GBR <- 0x0000004D
81
82      ADD #1, R0;
83      LDC R0, VBR;     VBR <- 0x0000004E
84
85      MOV #$24, R1;
86      STC.L SR, @-R1;      Expect 0x0000004C at 0x20
87      STC.L GBR, @-R1;     Expect 0x0000004D at 0x1C
88      STC.L VBR, @-R1;     Expect 0x0000004E at 0x18
89
90      ; Quit test program
91      MOV #-4, R0;
92      MOV.B R0, @R0;
93
94
```

```
 1   00000000 0000000F
 2   00000004 00000010
 3   00000008 00000011
 4   0000000C 0000001A
 5   00000010 0000002B
 6   00000014 0000003C
 7   00000020 0000004C
 8   0000001C 0000004D
 9   00000018 0000004E
10
```

```
 1  ; xtrct.asm
 2  ;
 3  ; This file tests the following instruction:
 4  ;    XTRCT Rm, Rn
 5  ;
 6  ; This instruction moves the center 32 bits of registers Rm and Rn into Rn.
 7  ; It takes the low word of Rm and moves it into the high word of Rn, and
 8  ; moves the high word of Rn into the low word of Rn.
 9  ;
10  ; Revision History:
11  ;   25 May 2025  Chris M.  Initial revision.
12  ;
13
14
15
16
17  Start:
18      ; XTRCT R0,R1   Before execution R0 = H'01234567, R1 = H'89ABCDEF
19      ;               After execution  R1 = H'456789AB
20
21      MOV   Var1,   R0
22      MOV   Var2,   R1
23
24      XTRCT R0,     R1
25
26      MOV   #$00,   R2
27      MOV   R1,     @R2
28
29      ; Expected memory:
30      ; 00000000 456789AB
31
32  End:
33    ; The test bench interprets a read of 0xFFFFFFFC (-4)
34    ;as system exit.
35    MOV #-4, R0;
36    MOV.B R0, @R0;
37
38
39
40      align 4 ; long-word align program memory.
41      LTORG
42
43  ; Long-word constants (in program memory).
44  ;
45  Var1:     dc.l $01234567
46  Var2:     dc.l $89ABCDEF
47
48
```

1