Summary of Instructions

This table shows which instructions can optionally set the condition code and which can be conditionally executed.

	Can optionally set the condition code by adding the suffix 's'	Can be conditionally executed
ADD / SUB	Yes	Yes
AND / ORR / EOR	Yes ... N, Z, C bits	Yes
BL (BRANCH & LINK)	No	Yes
BXX (BRANCH)	No	Yes
CMP	Always set the condition code	Yes
LDR / STR	No	Yes
MOV	Yes ... N, Z, C bits	Yes
STMDB / LDMIA	No	Yes
UMUL / SMULL	Yes ... N, Z, bits	Yes

Notes Related To The ARMSim Assembler And Simulator

These represent items encountered when coding for the ARMSim assembler and simulator. They may represent assembler and simulator implementation decisions as opposed to ARM architectural decisions.

1. Both instruction labels and data declaration names are case sensitive.

2. The way to specify a hex value is: 0xFFFFFFF

3. The load register instruction uses an = to identify addresses or immediate data to put in a register.

```
ldr   r0, =val1            ;r0 = address of val1
ldr   r1, =1000            ;r1 = 1000 decimal
ldr   r2, =0xFFFFFFFF ;r2 = FFFFFFFF hex
```

All other operations use a # to identify immediate data.

```
ldr    r4, [r0], #4          ;post increment
mov   r5, #1000              ;mov
add    r6, r6, #1            ;add constants
cmp   r7, #0                        ;compare
cmp   r8, #0xFFFFFFFF  ;compare
```

4. Note that all loads and stores of memory variables are indirect; they use a base register and second register or constant. Some assemblers will use the Program Counter, r15, as the base register and the offset of the variable into the program as the constant. However, as best I can tell, the ARMSim assembler and simulator do not support that and thus do not support directly loading a variable into a register. Instead you need to load the address of variable into a register then use that register as a pointer.

Desired Code

```
      ldr    r0, v1  ;r0=v1



      .data
v1:   .word  579     ;data item v1
```

Actual ARMSim Code Needed

```
      ldr    r0, =v1     ;r0=address of v1
      ldr    r0, [r0]    ;r0=v1


      .data
v1:   .word  579         ;data item v1
```

5. If an instruction can both optionally set the condition code and be conditionally executed then the mnemonic is: ***opcode condition_code_setting s***

   ... for example: addeqs

6. For conditional execution of load and store when you specify the data size, the mnemonic is:

   opcode condition_code_setting data_size

   ... for example: strneb r2,[r3,r4] if ne condition true, store bottom byte of r2 to address r3+r4

7. The simulator does not have a dup function for building tables. This is what I recommend

```
Table: .byte  00,  00,  00,  00,  00,  00,  00,  00,  00,  00
       .byte  00,  00,  00,  00,  00,  00,  00,  00,  00,  00
       .byte  00,  00,  00,  00,  00,  00,  00,  00,  00,  00
```

Create the table with lines that have 10 values per line. This has two advantages
1 - You can use your editor's copy and paste functions to duplicate lines.
2 - With 10 values per line it will be much easier to debug the table.
It is easy to determine the translated value for any input value.