In general, for the ARM processors you can add an *s* to an instruction and it will alter the flags. Using the same instruction without an *s* will not alter the flags. For the compiler/assembler that we have this is often not an option. Using `add r1, r2` will generate an error which says that the "flag preserving form of the instruction is not available". Change such instructions to `adds r1, r2`

Loading immediate data into a register:
```
ldr R3, =0x000003FF;  Loads the 32 bit hex number into R3
ldr R2, =41;          Loads 41 base 10 into R2
```

Store a register into a memory:
```
str R2, [R1];         Stores R2 into the memory whose address is in R1
```

Copy one register into another:
```
movs R2, R3;           Copies R3 into R2
movs R2, #0xFF;        Copies 255 into R2. Immediate limited to 0 to 255
```

Add:
```
adds R2, R3;           Add R2 + R3 and puts result in R2 (flags set)
adds R10, R11;         R10 = R10 + R11 but flags not set
```

Note: Flags are set only if registers are R0 to R7

```
adcs R2, R3;           R2 = R2 + R3 + carry
```

Note: Can be used only with registers are R0 to R7

Subtract:
```
subs R1, R2;           R1 = R1 – R2
subs R1, R2, R3;       R1 = R2 – R3
```

Note: Can be used only with registers are R0 to R7

```
sbcs R1, R2;           R1 = R1 – R2 and accounts for carry flag (borrow)
```

Note: Can be used only with registers are R0 to R7

Multiply:
```
mpys R1, R2;           R1 = R1*R2
```

Note: Can be used only with registers are R0 to R7

Compare:
```
cmp R2, R10;           Subtracts R10 from R2, discards result and sets flags
cmp R1, #0x20;         Subtracts 0x20 from R1, discards result and set flags
```

Note: for immediate operand value must be 0 to 255.

Logical AND, OR, and Exclusive OR
```
ands R1, R2;           R1 = R1 AND R2
orrs R1, R2;           R1 = R1 OR R2
eors R1, R2;           R1 = R1 Ex Or R2
```

Note: Can be used only with registers are R0 to R7
Negation:

\[ \text{neg } R2, R2; \quad R2 = -R2 \]
\[ \text{neg } R2, R3; \quad R2 = -R3 \]

Left and Right Shift

\[ \text{lsls } R2, R3, #5; \quad \text{shift left } R3 \text{ 5 times and store in } R2 \]
\[ \text{lsls } R2, R3; \quad \text{shift left } R2, R3 \text{ value times. If } R3 \text{ is 31, } R2 \text{ is cleared, if } R3 > 31 \text{ } R2 \text{ and carry are cleared.} \]
\[ \text{lsrs } R2, R3, #5; \quad \text{shift right } R3 \text{ 5 times and store in } R2 \]
\[ \text{lsrs } R2, R3; \quad \text{shift right } R2, R3 \text{ value times. If } R3 \text{ is 31, } R2 \text{ is cleared, if } R3 > 31 \text{ } R2 \text{ and carry are cleared.} \]
\[ \text{asrs } R2, R3, #5; \quad \text{arithmetic shift right } R3 \text{ 5 times and store in } R2 \]
\[ \text{asrs } R2, R3; \quad \text{arithmetic shift right } R2, R3 \text{ value times. If } R3 \text{ is 31, } R2 \text{ is cleared, if } R3 > 31 \text{ } R2 \text{ and carry are cleared.} \]

Branch instructions

\[ \text{B Target; } \quad \text{unconditional branch to target within } \pm 2K \text{ (10 bits)} \]
\[ \text{B\{Cond\} Target; } \quad \text{conditional branch to target within } -252 \text{ to } +258 \]
\[ \text{BL\{Cond\} Target; } \quad \text{Branch and link. Stores return address in \text{R}14 \text{ and conditionally branches to target. To return from subroutine move } \text{R}14 \text{ (link reg) into } \text{R}15 \text{ (PC)}} \]

<table>
<thead>
<tr>
<th>Condition</th>
<th>Flags</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>Z set</td>
<td>equal</td>
</tr>
<tr>
<td>NE</td>
<td>Z clear</td>
<td>not equal</td>
</tr>
<tr>
<td>CS/HS</td>
<td>C set</td>
<td>unsigned higher or same</td>
</tr>
<tr>
<td>CC/LO</td>
<td>C clear</td>
<td>unsigned lower</td>
</tr>
<tr>
<td>MI</td>
<td>N set</td>
<td>negative</td>
</tr>
<tr>
<td>PL</td>
<td>N clear</td>
<td>positive or zero</td>
</tr>
<tr>
<td>VS</td>
<td>V set</td>
<td>overflow</td>
</tr>
<tr>
<td>VC</td>
<td>V clear</td>
<td>no overflow</td>
</tr>
<tr>
<td>HI</td>
<td>C set and Z clear</td>
<td>unsigned higher</td>
</tr>
<tr>
<td>LS</td>
<td>C clear or Z set</td>
<td>unsigned lower or same</td>
</tr>
<tr>
<td>GE</td>
<td>N equals V</td>
<td>signed greater or equal</td>
</tr>
<tr>
<td>LT</td>
<td>N not equal to V</td>
<td>signed less than</td>
</tr>
<tr>
<td>GT</td>
<td>Z clear AND (N equals V)</td>
<td>signed greater than</td>
</tr>
<tr>
<td>LE</td>
<td>Z set OR (N not equal to V)</td>
<td>signed less than or equal</td>
</tr>
<tr>
<td>AL</td>
<td>(ignored)</td>
<td>always (usually omitted)</td>
</tr>
</tbody>
</table>

Push and Pop

\[ \text{push \{R1, R2\}; } \quad \text{Pushes } R1 \text{ and } R2 \text{ on the stack} \]
\[ \text{push \{R1-R5\}; } \quad \text{Pushes } R1, R2, R3, R4, \text{ and } R5 \text{ on the stack} \]
\[ \text{push \{R1, R2, LR\}; } \quad \text{Pushes } R1, R2, \text{ and the link register on the stack} \]
\[ \text{pop \{R1, R2\}; } \quad \text{Pops } R1 \text{ and } R2 \text{ on the stack} \]
\[ \text{pop \{R1-R5\}; } \quad \text{Pops } R1, R2, R3, R4, \text{ and } R5 \text{ from the stack} \]
pop (R1, R2, LR); Pops R1, R2, and the link register from the stack
Note: Can be used only with registers are R0 to R7 and the link register.