Overflow

- When we use fixed size Register for arithmetic operands there is the hazard at each micro-ops of overflow.
- If Register R stores an n-bit 2’s compliment number, then:
  
  \[-2^{n-1} \leq R \leq 2^{n-1}-1\]

- Note the asymmetry, so even negation can cause overflow, e.g.

  \[
  R \leftarrow -2^{n-1} \text{ then } R \leftarrow -R \Rightarrow \text{OVERFLOW}
  \]
We can detect overflow for all the previous operations simply by recording the status bits $C = \text{Carry}$ and $V = \text{Overflow}$ (see section 3.10 in Mano and Kime).

$$K_1: C \leftarrow C_n, \quad V \leftarrow C_n \oplus C_{n-1}$$
This then is the basis for the adder-subtractor on the next page which uses control input $X$ to select addition and $X$ for subtraction.

\[
\begin{align*}
X.K_1: R1 & \leftarrow R1 + R2 \\
X.\overline{K}_1: R1 & \leftarrow R1 + \overline{R2} + 1 \\
K_1: C & \leftarrow C_n, V \leftarrow C_n \oplus C_{n-1}
\end{align*}
\]
CS2022 Add & Subtract Micro operation
The aim here is to provide an effective set of bit-wise functions. A typical basic set follows:

<table>
<thead>
<tr>
<th>Symbolic micro-op</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_0 \leftarrow \overline{R_1}$</td>
<td>Logical bitwise NOT (1’s compliment)</td>
</tr>
<tr>
<td>$R_0 \leftarrow R_1 \land R_2$</td>
<td>Logical bitwise AND (clears bits)</td>
</tr>
<tr>
<td>$R_0 \leftarrow R_1 \lor R_2$</td>
<td>Logical bitwise OR (sets bits)</td>
</tr>
<tr>
<td>$R_0 \leftarrow R_1 \oplus R_2$</td>
<td>Logical bitwise XOR (complements bits)</td>
</tr>
</tbody>
</table>
George Bode Prof. Of mathematics in UCC introduced the notation ‘∧’ and ‘∨’ in 1854, and they are used in Register Transfer (RT) notation if it is necessary to distinguish addition from logical OR. For example:

\[ K_1 + K_2 : R1 \leftarrow R2 + R3, \ R4 \leftarrow R5 \lor R6 \]

The OR micro-ops will always use ∨.
Shift Micro-operations

- These provide lateral bitwise shift which are essential for many basic arithmetic algorithms e.g. multiplication, division, square root...

- The minimal set is:

  \[
  R \leftarrow \text{sr } R \equiv R_{i} \leftarrow R_{i+1} \quad i=0, n-2, \quad R_{n-1} \leftarrow 0
  \]

  \[
  R \leftarrow \text{sl } R \equiv R_{i} \leftarrow R_{i-1} \quad i=1, n-1, \quad R_{0} \leftarrow 0
  \]
These are logical shifts and from them you can develop variants which handle the end bits differently, e.g. arithmetic shift, rotates…

Please see table 9-5 in Mano and Kime.
Transfer Micro-operations

- Providing choice of path
- Two approaches are used:
  - Multiplexer-based transfer for speed
  - Bus-based transfers for flexibility and economy
Transfer with Multiplexer

The if-then else control structure, when applied to identity micro-ops, results in the destination register requiring selective access to two different source registers.

If \((K_1=1)\) then \(R0 \leftarrow R1\)
else if \((K_2=1)\) then \(R0 \leftarrow R2\)

\[K_1: R0 \leftarrow R1, \overline{K}_1 K_2: R0 \leftarrow R2\]
To route two sources to one destination we can use a 2:1 MUX with control input $S$, data input $D0 \land D1$, and $R0$ must have a load control $LOAD_{R0}$.

From the RT description we deduce the following functions for these control inputs.

$$LOAD_{R0} = K_1 + \overline{K}_1K_2 = K_1 + K_2$$

$$S = K_1 \Rightarrow D1 = R1 \land D0 = R2$$
Multiplexer

From this we derive the circuit schematic, assuming 4-bit Registers.
Note changes in K1 only affect the MUX – R0 path, so transients are short, allowing fast operation.