Non-restoring Method

- The non-restoring division defers restoration to reduce operations
  - Addition is performed instead of subtraction in the next step
  - This requires a final restoration, when last operation yielded a negative remainder

Non-Restoring Method

- Initialization
  - Clear *A*. Requires 1 extra bit for *A* to be used as a sign bit.
  - *Q* gets dividend.
  - *M* gets divisor.
- Loop for each bit of the dividend *Q*
  - If *A* (previous result) is negative ($A_n = 1$)
    - Shift *AQ* to the left.
    - Add ($A \leftarrow A + M$)
    - Q_0 gets the complement value of the MSB of *A*
  - Else
    - Shift *AQ* to the left.
    - Subtract ($A \leftarrow A - M$)
    - Q_0 gets the complement value of the MSB of *A*
- Perform restoration if last result in *A* is negative, *i.e.*, if $A_n = 1$ then $A \leftarrow A + M$
- Quotient in *Q* while remainder in *A*; adjust sign as needed

Non-Restoring Division

111 (Q) / 11 (M)

Initialize

```
-M: 1101
A: 0000    Q: 111
M: 0011
```

1st pass

```
A: 0001    Q: 11_
A: 1110    Q: 110
```

2nd pass

```
A: 1101    Q: 10_
A: 0000    Q: 101
```

3rd pass

```
A: 0001    Q: 01_
A: 1110
A: 0001    Q: 010
```

- Initialization
  - Clear *A*. Requires 1 extra bit for *A* to be used as a sign bit.
  - *Q* gets dividend.
  - *M* gets divisor.
- Loop for each bit of the dividend *Q*
  - If *A* (previous result) is negative ($A_n = 1$)
    - Shift *AQ* to the left.
    - Add ($A \leftarrow A + M$)
    - Q_0 gets the complement value of the MSB of *A*
  - Else
    - Shift *AQ* to the left.
    - Subtract ($A \leftarrow A - M$)
    - Q_0 gets the complement value of the MSB of *A*
- Perform restoration if last result in *A* is negative, *i.e.*, if $A_n = 1$ then $A \leftarrow A + M$
- Quotient in *Q* while remainder in *A*; adjust sign as needed

Try

Try: 01101 (Q) / 00101 (M) (using non-restoring division)
Show the value of A and Q after the end of each pass

After this pass	A	Q
1st	111011	11010
2nd	111100	10100
3rd	11110	01000
4th	000001	10001
5th	000011	00010