Outline

1. Digital Systems
   - half adders
   - adder circuits

2. Looping Constructs
   - the while and for loops
   - Euclid’s algorithm and the state table

MCS 260 Lecture 11
Introduction to Computer Science
Jan Verschelde, 26 June 2023
1 Digital Systems
   - half adders
   - adder circuits

2 Looping Constructs
   - the while and for loops
   - Euclid’s algorithm and the state table
Half Adders
adding two bits

A half adder takes on input two bits $b_1$ and $b_2$, and returns the sum $S$ and the carry over $C$.

An adder takes on input $b_1$, $b_2$, and carry over $C$.

As logical functions, they are

$C = b_1 \ AND \ b_2$

$S = ( \ NOT \ b_1 \ AND \ b_2 ) \ OR \ ( b_1 \ AND \ NOT \ b_2 )$

\[\begin{array}{cccccccc}
  b_1 & b_2 & C & \text{NOT } b_1 \ AND \ b_2 & b_1 \ AND \ NOT \ b_2 & S \\
  0 & 0 & 0 & 0 & 0 & 0 \\
  1 & 0 & 0 & 0 & 1 & 1 \\
  0 & 1 & 0 & 1 & 0 & 1 \\
  1 & 1 & 1 & 0 & 0 & 0 \\
\end{array}\]

Does $S$ look familiar? $S = \text{xor} = \text{exclusive or}$
Realization of a Half Adder
with logic gates: 2 NOTs, 3 ANDs, and 1 OR

\[
C = b_1 \text{ AND } b_2 \\
S = ( \text{ NOT } b_1 \text{ AND } b_2 ) \text{ OR } ( b_1 \text{ AND } \text{ NOT } b_2 )
\]

exercise: realize with NAND & NOR
Digital Systems
- half adders
- adder circuits

Looping Constructs
- the while and for loops
- Euclid’s algorithm and the state table
Adding Four Bits

\[
\begin{array}{cccc}
\times & \times & \times & \times \\
3 & 2 & 1 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
y & y & y & y \\
3 & 2 & 1 & 0 \\
\end{array}
\]

\[
\begin{array}{ccc}
a & b & c \\
\text{ADDC} & \text{ADDC} & \text{ADDC} \\
C & C & C \\
\end{array}
\]

\[
\begin{array}{cccc}
z & z & z & z \\
3 & 2 & 1 & 0 \\
\end{array}
\]

\[
\begin{array}{cc}
a & b \\
\text{ADD} \\
C & S \\
\end{array}
\]

Intro to Computer Science (MCS 260) | adder circuits and loops | L-11 26 June 2023 6 / 16
To add sequences of bits, we need a more complex circuit, an adder which has three inputs:

1. the two bits to add at the current position;
2. the carry over from the previous position.

To speedup the addition of bit sequences, circuits are enabled with carry look ahead.

Other operations that run fast on bit sequences are

- **rotate**: $1101 \rightarrow 1011$
- **shift**: $1101 \rightarrow 1010$
Bitwise Operators in Python

Multiplying numbers with a power of two is just shifting bits to the left, padding with zeroes to the right.

Using the bitwise operator `<<` we efficiently create large numbers, e.g. $2^{100}$:

```python
>>> 1 << 100
1267650600228229401496703205376
```

Shifting $k$ bits to the right, dividing by $2^k$ is done via `>>`.

A summary of bitwise operators:

<table>
<thead>
<tr>
<th>operator</th>
<th>example</th>
<th>explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>∼</td>
<td>∼ $a$</td>
<td>not $a = -a - 1$</td>
</tr>
<tr>
<td>&amp;</td>
<td>$a &amp; b$</td>
<td>$a$ and $b$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$a</td>
</tr>
<tr>
<td>^</td>
<td>$a ^ b$</td>
<td>$a$ xor $b$</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>$a &lt;&lt; b$</td>
<td>$a \times 2^b$</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>$a &gt;&gt; b$</td>
<td>$a/2^b$</td>
</tr>
</tbody>
</table>
1 Digital Systems
   - half adders
   - adder circuits

2 Looping Constructs
   - the while and for loops
   - Euclid’s algorithm and the state table
List Comprehensions — lists and loops

A list is the counterpart data structure to the control structure loop. Combining lists and for loops to show evolving balance:

```python
>>> (B, r) = (1000, 3.14)
>>> L = [ B*(1+r/100)**n for n in range(5) ]
>>> s = [ f"{x:.2f}" for x in L ]
>>> s
['1000.00', '1031.40', '1063.79', '1097.19', '1131.64']
```

General syntax of a list comprehension:

```
[ <function of i> for <i> in range(<a>, <b>) ]
```
the while loop
continue until condition becomes false

The flowchart of the while loop:
syntax of the while statement

A while statement has the format
< initialize variables >
while < continue ? >
   < body of loop >
All statements in the body must be indentated!

The calculations to compute $B(1 + r/100)^i$, for $i$ from 1 to $n$:

```python
i = 1
while i <= n:
    B = B*(1 + r/100)
    i = i + 1
```

Forgetting $i = i + 1$ results in an infinite loop!

Press the keys `ctrl` and `c` simultaneously to exit.

Also the Unix command `kill` terminates processes.
the for loop: enumerating items

Because enumerating items occurs so frequently, the for statement does automatically
1. the initialization of the counter, and
2. the update of the counter in the body of the loop.

Syntax of the for statement:
for < counter > in < sequence > :
    < body of loop >

where < sequence > is typically a range of numbers:

>>> [x for x in range(3, 11)]
[3, 4, 5, 6, 7, 8, 9, 10]

Note the ending of the range!

What type is returned by range()?
**1 Digital Systems**
- half adders
- adder circuits

**2 Looping Constructs**
- the while and for loops
- Euclid's algorithm and the state table
The Greatest Common Divisor — the state table

**Input:** two positive numbers $a > b > 0$.

**Output:** the greatest common divisor of $a$ and $b$.

An example: $a = 60$, $b = 51$

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$r = a % b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>51</td>
<td>9</td>
</tr>
<tr>
<td>51</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

- The *state table* shows the value of all variables for every stage in the loop.
- We compute a sequence of remainders $r = a \% b$.
- The sequence stops when $r$ equals zero.
Exercises

1. Using logical NOT, AND, OR gates construct a circuit which realizes an adder.

2. Modify the Python program gcd.py so that it prints out the values of all variables for each step.

3. The factorial of a positive numbers $n$ is $n! = n(n-1)(n-2) \cdots 1$. Give Python code which takes $n$ on input and has $n!$ as output. Make two versions: one with `while` and the other with `for`.

4. Use `>>` in a loop to determine how many bits there are in any given number.

5. Draw a flowchart for the summation to approximate $\pi$, summing the area of $n$ trapezoids over $n$ subintervals of $[0, 1]$, for the function $\sqrt{1 - x^2}$. 