WORKSHEET

A8.7 Half adders and full adders

Boolean algebra and logic circuits · A level · OCR H446 1.4.3, AQA 7517 4.6.4.1 · about 20 min

BugBotLab

Name _____ Class _____ Date _____

What this lesson is about

The full adder's expressions, building it from half adders, and chaining full adders into a ripple carry adder.

- The half adder, again
- Why "half"?
- The full adder
- Two half adders and an OR gate
- Adding whole numbers: the ripple carry adder

Questions

5 marks in all

1. Which expressions give a half adder's outputs? [1 mark]

- ☐ **A** $S = A \veebar B$, $C = A \wedge B$
- ☐ **B** $S = A \vee B$, $C = A \wedge B$
- ☐ **C** $S = A \wedge B$, $C = A \veebar B$
- ☐ **D** $S = A \veebar B$, $C = A \vee B$

2. Why is a half adder not enough to add multi-bit numbers? [1 mark]

- ☐ **A** It has no input for the carry from the previous column
- ☐ **B** It cannot add 1 and 1
- ☐ **C** It has no sum output
- ☐ **D** It only works on the leftmost bit

3. A full adder has A = 1, B = 0 and Cin = 1. Give Cout and S as two digits, Cout first. [1 mark]

4. How is a full adder built from half adders? [1 mark]

- ☐ **A** Two half adders, with their two carry outputs joined by an OR gate
- ☐ **B** Two half adders, with their carry outputs joined by an AND gate
- ☐ **C** Three half adders in a row
- ☐ **D** One half adder and a NOT gate

5. What does this program print? [1 mark]

```
def full_adder(a, b, cin):
    s1, c1 = a ^ b, a & b
    return s1 ^ cin, c1 | (s1 & cin)

carry = 0
bits = ""
for a, b in [(1, 1), (1, 0), (0, 1), (1, 0)]:
    s, carry = full_adder(a, b, carry)
    bits = str(s) + bits
print(bits, carry)
```

The task: a 4-bit ripple carry adder

Write three functions. Every bit is the integer 0 or 1. - `half_adder(a, b)` returns a tuple `(sum, carry)`. - `full_adder(a, b, carry_in)` returns a tuple `(sum, carry_out)`, and must be built from two calls to `half_adder` and an OR. - `add4(x, y)` takes two strings of four characters, each `"0"` or `"1"`, with the most significant bit first. It adds them with a ripple of `full_adder` calls, starting from the rightmost bit with a carry in of 0, and returns a tuple `(total, carry)`: `total` a 4-character string of the sum bits, most significant first, and `carry` the final carry out (0 or 1). Your program may not use `int()`, `bin()`, `format()` or `sum()`. For each pair `("0110", "0111")`, `("1011", "0110")`, `("1111", "0001")` and `("0101", "1010")`, in that order, print one line in exactly this form: `0110 + 0111 = 1101 carry 0`

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

def half_adder(a, b):
    return 0, 0

def full_adder(a, b, carry_in):
    return 0, 0

def add4(x, y):
    return "0000", 0
```

Plan your program here, then type it in and press Run.

Do it on the robot

www.bugbotlab.com/learn/a8-7-half-and-full-adders/

The simulator checks it and tells you when it passes. Nothing to install, no account.

Challenges

1. Change `add4` to add numbers of any length, and add two 8-bit numbers.
2. Count the gates in your 4-bit adder, counting XOR, AND and OR as one gate each.
3. Two's complement subtraction A minus B is A plus (NOT B) plus 1. Use your adder with a carry in of 1 to subtract 0011 from 0110.