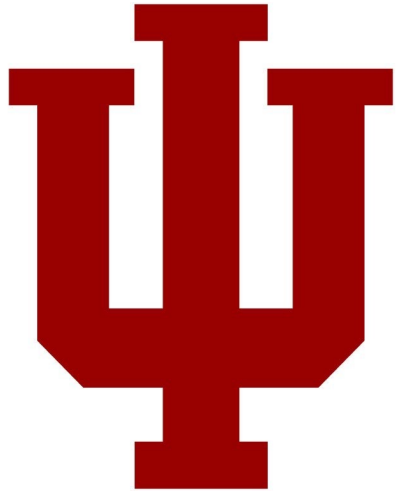
19: Parallelism II

ENGR 315: Hardware/Software CoDesign

Andrew Lukefahr

Indiana University



Some material taken from:
https://github.com/trekhleb/homemade-machine-learning/tree/master/homemade/neural_network
http://cs231n.github.io/neural-networks-1/

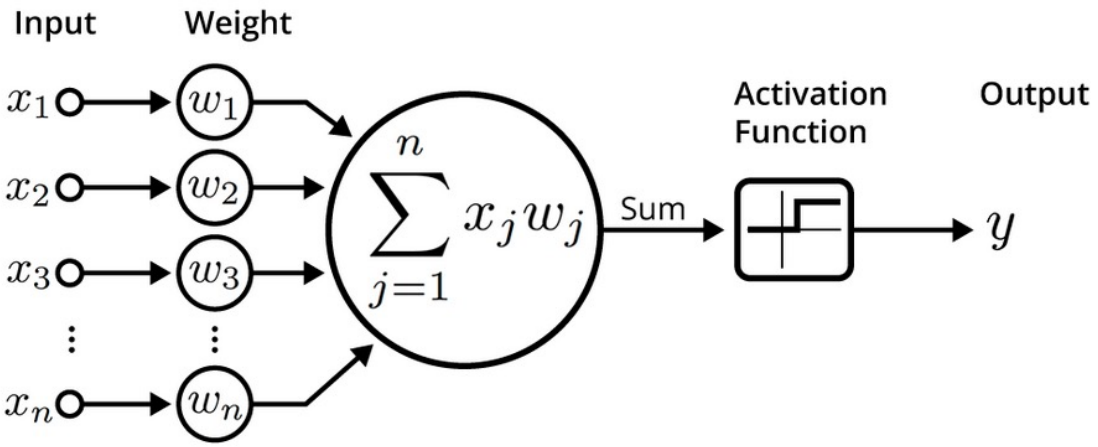
Announcements

- P8: Pipelining
- P9: Parallelism

P8+ are part of the “Design Challenge”

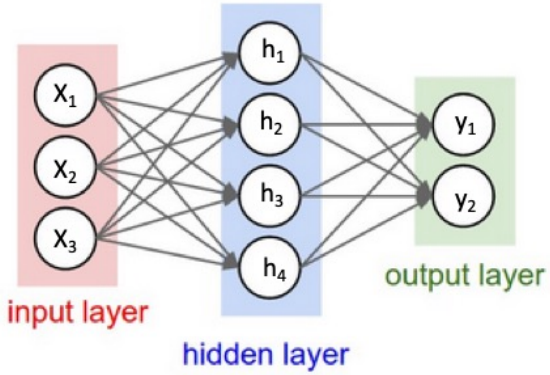
- Goal: Accelerate reference neural network
- Harder, more open-ended projects

Python Neuron




```
class Neuron(object):
  # ...
  def forward(self, inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
    return firing_rate
```

Why focus on Dot Product?




```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

Matrix Multiplication (Dot Product)

inputs

$$\begin{bmatrix} 0.1 & 0.2 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} =$$

weights

$$= \begin{bmatrix} (0.1 \times 1 + 0.2 \times 4) & (0.1 \times 2 + 0.2 \times 5) & (0.1 \times 3 + 0.2 \times 6) \end{bmatrix}$$

(Answer)

$$= \begin{bmatrix} 0.9 & 1.2 & 1.5 \end{bmatrix}$$

Multiply-Accumulate Dot Computations

$$\begin{bmatrix} 0.1 & 0.2 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 0.9 & 1.2 & 1.5 \end{bmatrix}$$

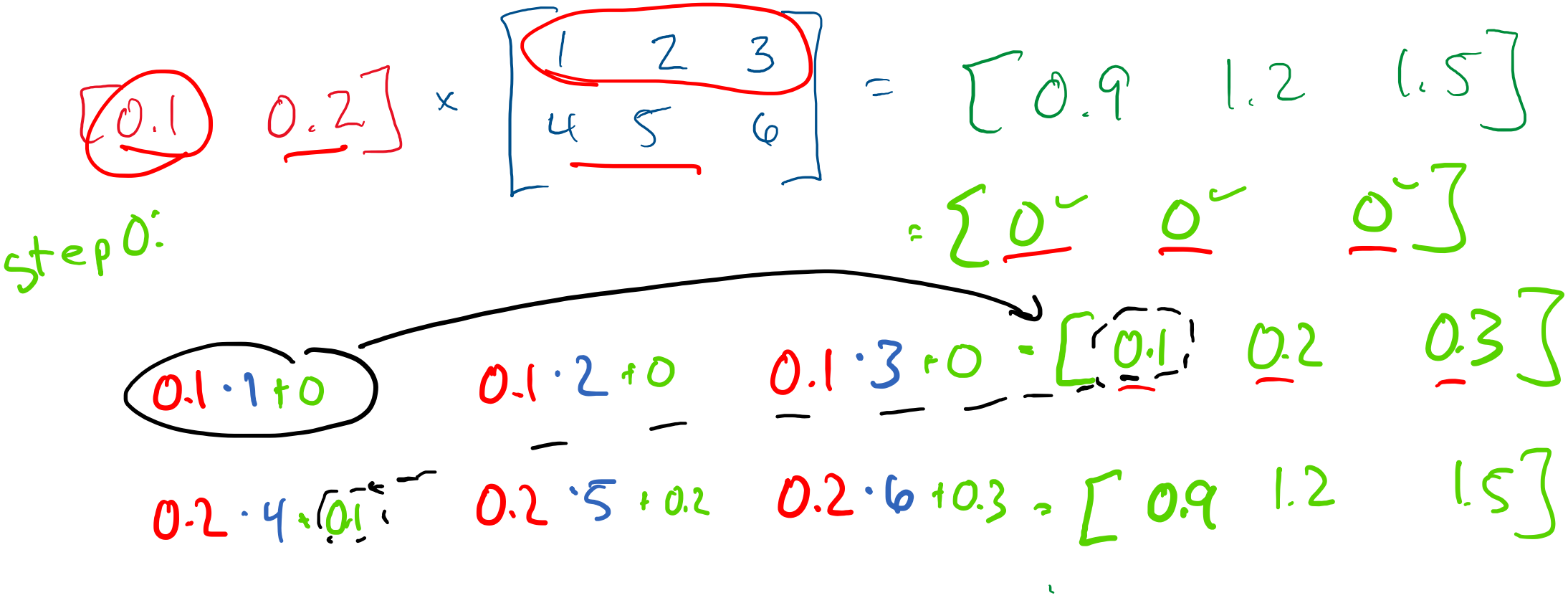
round 1

$$\begin{bmatrix} 0.1 \cdot 1 & 0.1 \cdot 2 & 0.1 \cdot 3 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix}$$

round 2

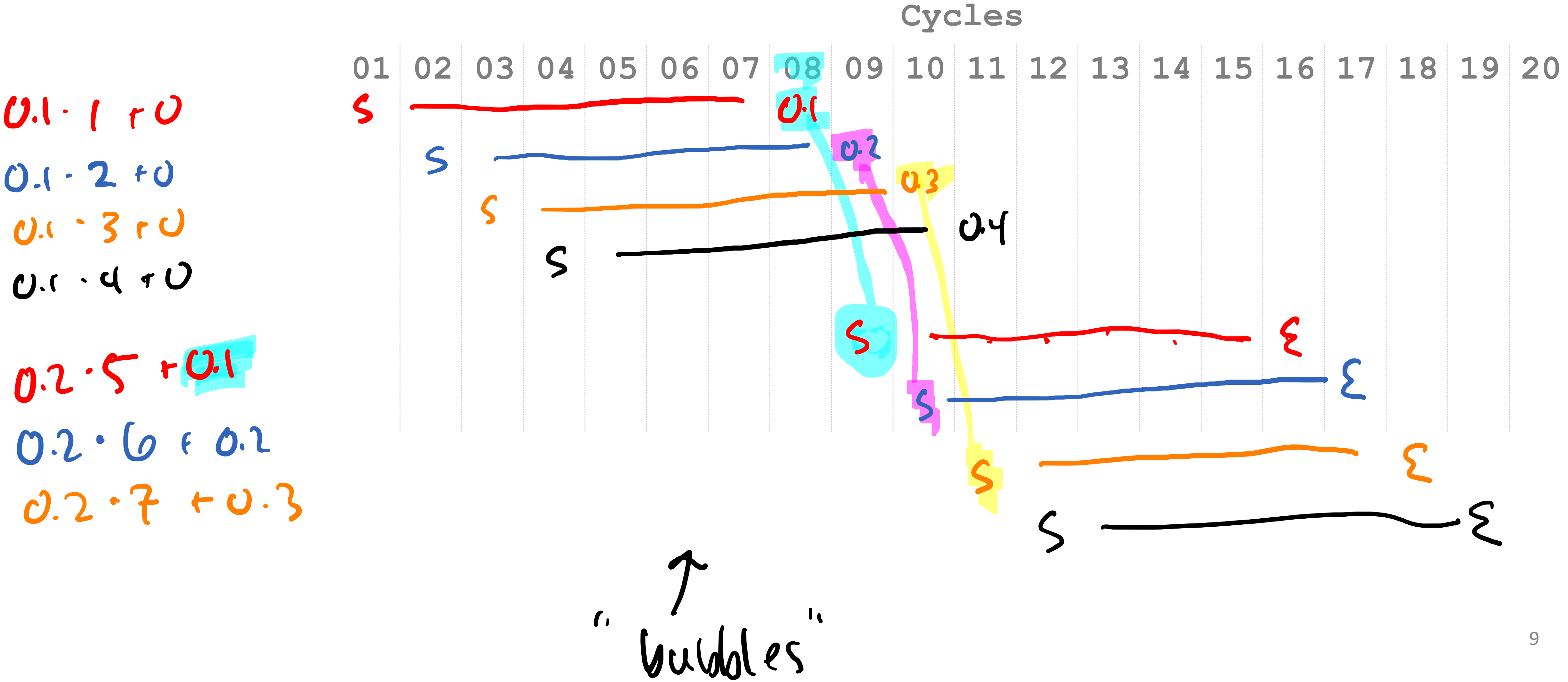
$$0.2 \cdot 4 \quad 0.2 \cdot 5 \quad 0.2 \cdot 6 + \begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix} = \begin{bmatrix} 0.9 & 1.2 & 1.5 \end{bmatrix}$$

Multiply-Accumulate Dot Computations



Dependencies

```
Input                 Weights              Output
[[0.1 0.2 0.3]]   .   [1.  2.  3.  4.]   = [3.8000002 4.4     5.     5.6000004]
                      [5.  6.  7.  8.]
                      [ 9. 10. 11. 12.]
```



Latency on Pipelined FMAC

- Solution: Stall at the end of a row.
- “Drain” the pipeline.

- Restart new row
- “Refill” the pipeline

P8: Dot Organization

Verilog Parameters

- Parameters are Verilog constructs that allow a module to be **reused with a different specification**.

```
module adder #(parameter BITS = 2) (
     input [BITS-1:0] a,
     input [BITS-1:0] b,
     output [BITS-1:0 c) );
     assign c = a + b;
endmodule
```

Verilog Parameters

```
module adder #(parameter BITS = 2) (
       input [BITS-1:0] a,
      input [BITS-1:0] b,
      output [BITS-1:0 c) );
      assign c = a + b;
endmodule
```

```
// 2-bit adder
adder add2 (a2, b2, c2);
//another 2-bit adder
adder #(BITS=2) add2b (a2b,b2b,c2b);
//a 3-bit adder
adder #(BITS=3) add3 (a3, b3, c3);

Adder #(BITS=32) add32 (a32, b32,
c32);
```

Verilog Parameters in Dot_20_10.sv

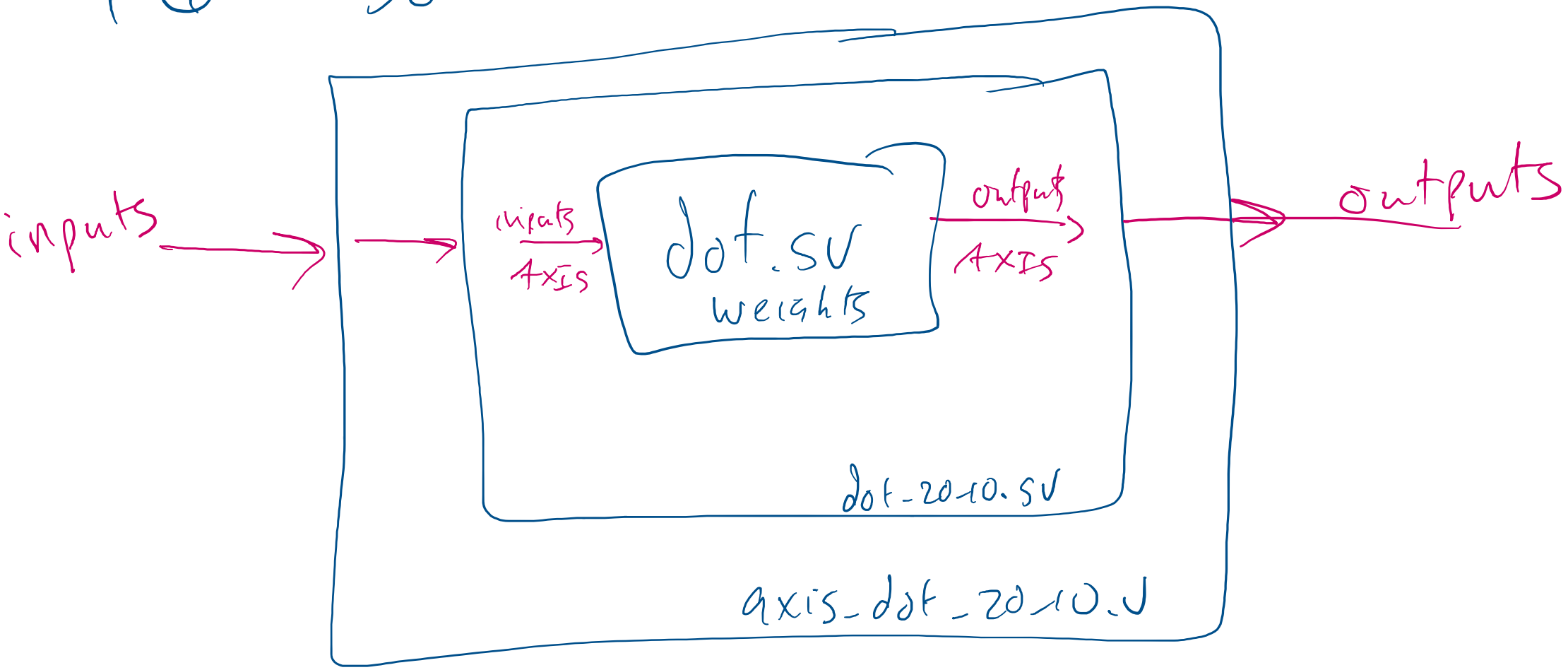
```
// This is autogenerated. see python/dot_20_10.py for details.
localparam ROWS = 20;
localparam COLS = 10;

localparam [31:0] weights [0:ROWS-1] [0:COLS-1] = '{
  `{...}... };

 dot #(
        .ROWS(ROWS),
        .COLS(COLS)
    ) dot0(
        // AXI4-Stream Interface
        .clk(clk),
        .rst(rst),

        .weights(weights),
```

P6 Dot Data flow



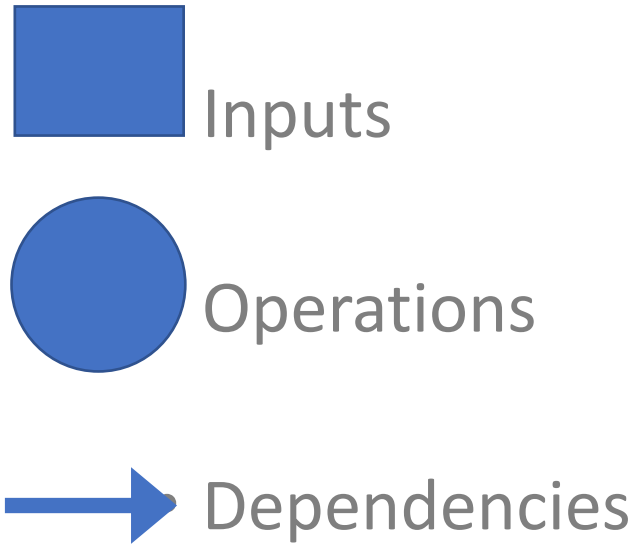
Data flow graph

- A data-flow graph is **a collection of arcs and nodes in which the nodes are either places where variables are assigned or used**, and the arcs show the relationship between the places where a variable is assigned and where the assigned value is subsequently used.

[ref]

Data Flow Graph

- Illustrates the dependencies between inputs and operations to produce an output



Data Flow Graph

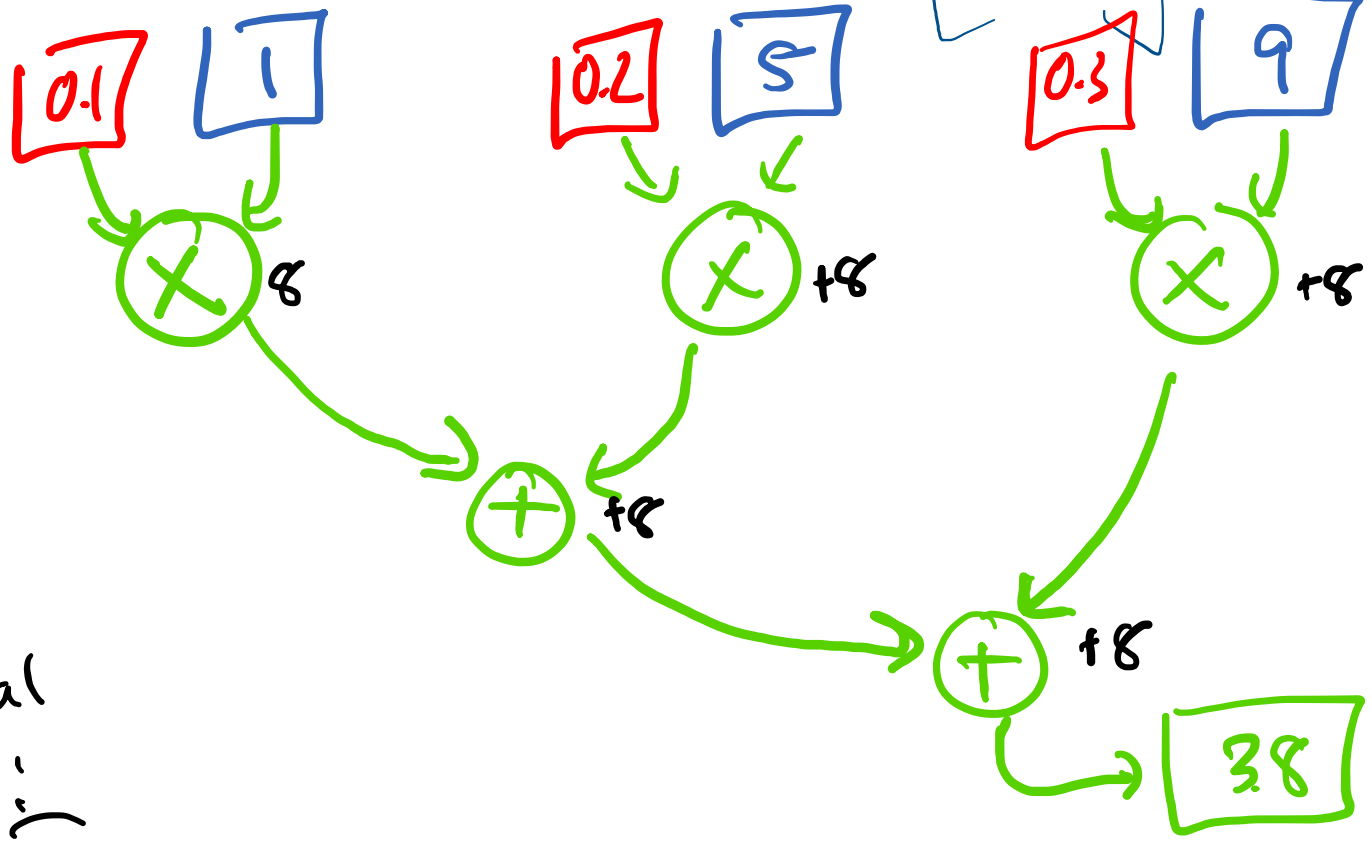
MAC for NOW

+ NO

+ NO

pipelining

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} 1 \\ 5 \\ 9 \end{bmatrix} = \begin{bmatrix} 3.8 \end{bmatrix}$$



MULT: 8 cycles

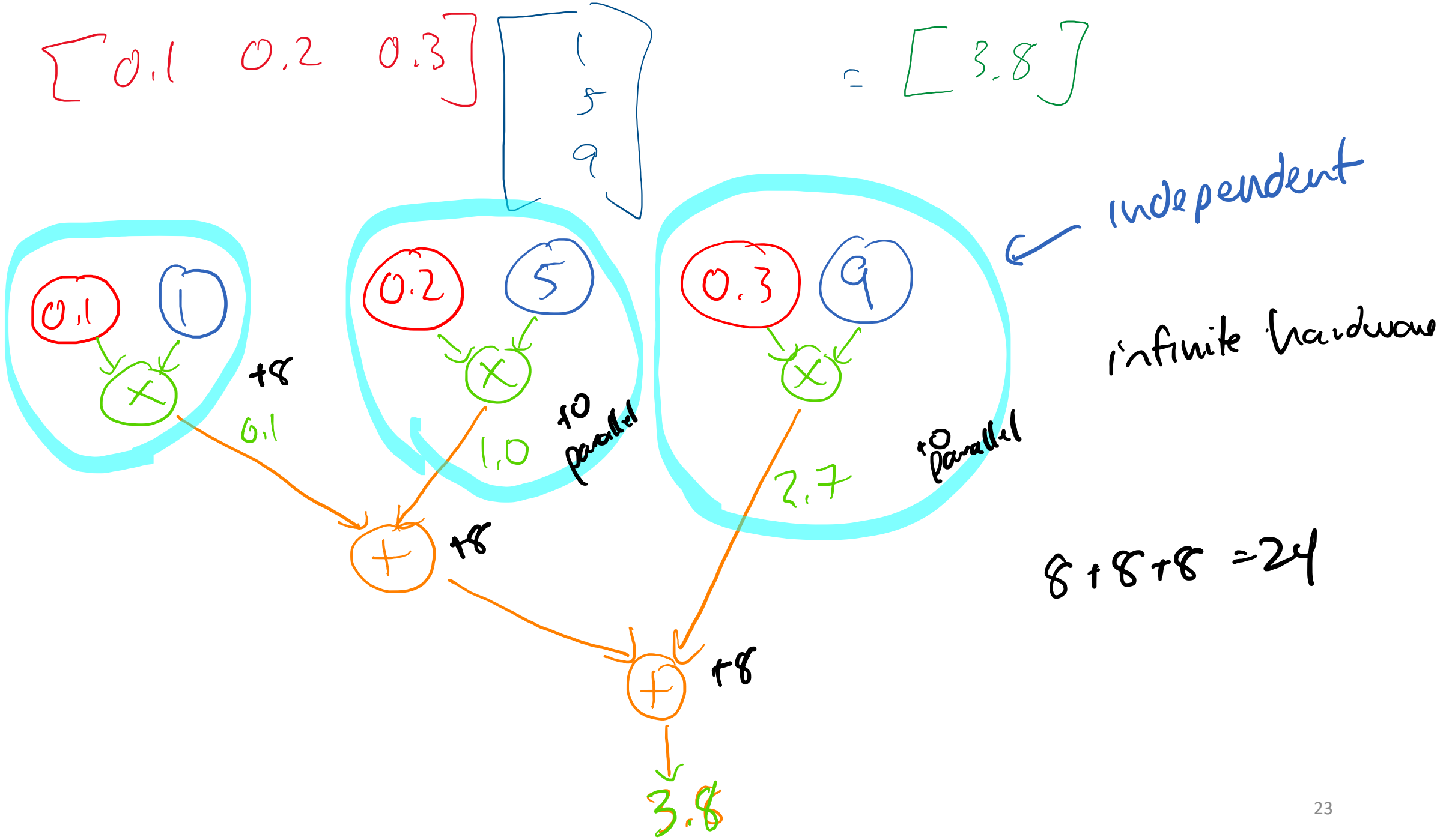
ADD: 8 cycles

8 cycles x 5 ops = 40 cycles

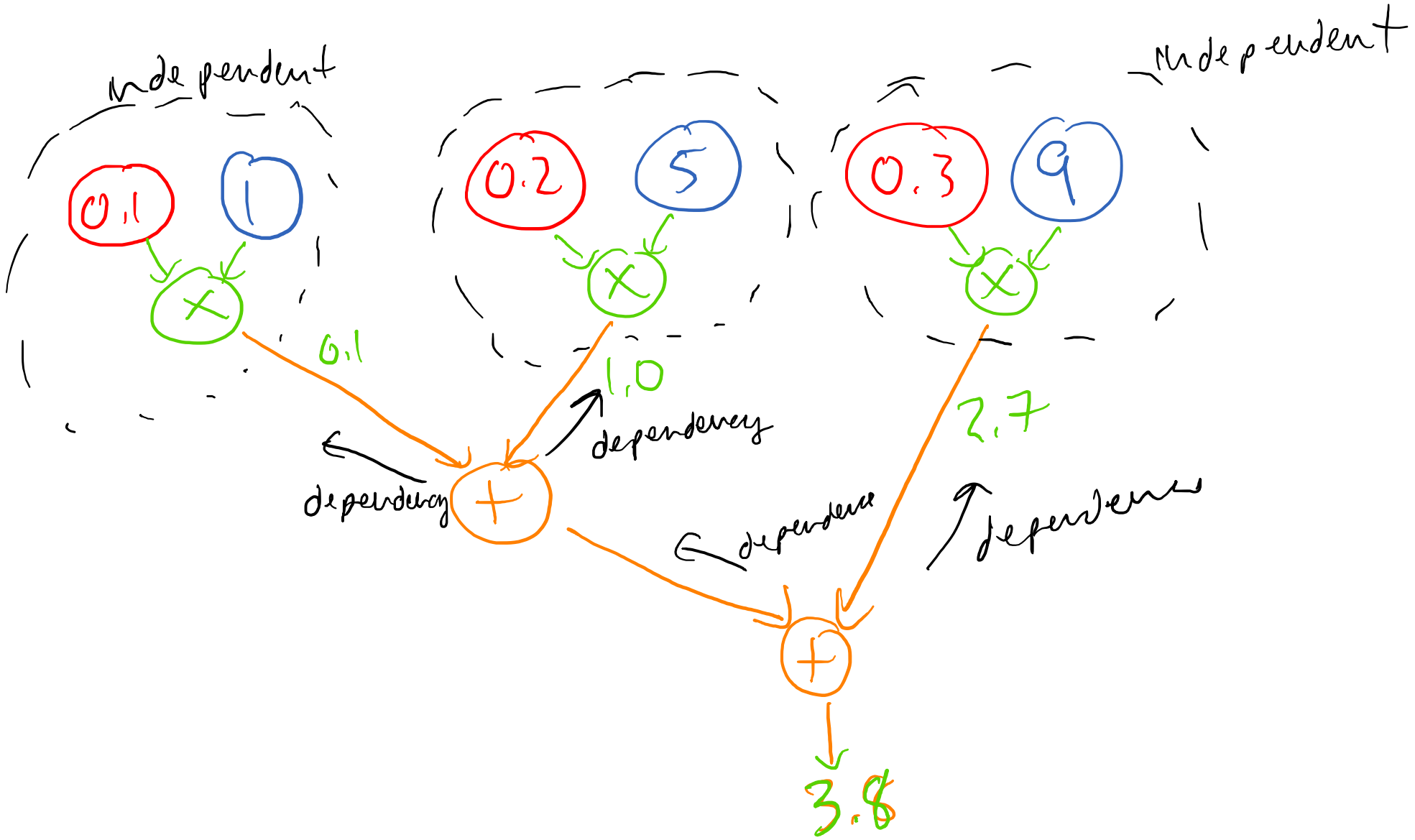
160 total

Finding Parallelism

- Find some computation that doesn’t depend on other computation’s results.
- No arrows between blocks.
- Shared Inputs are OK.

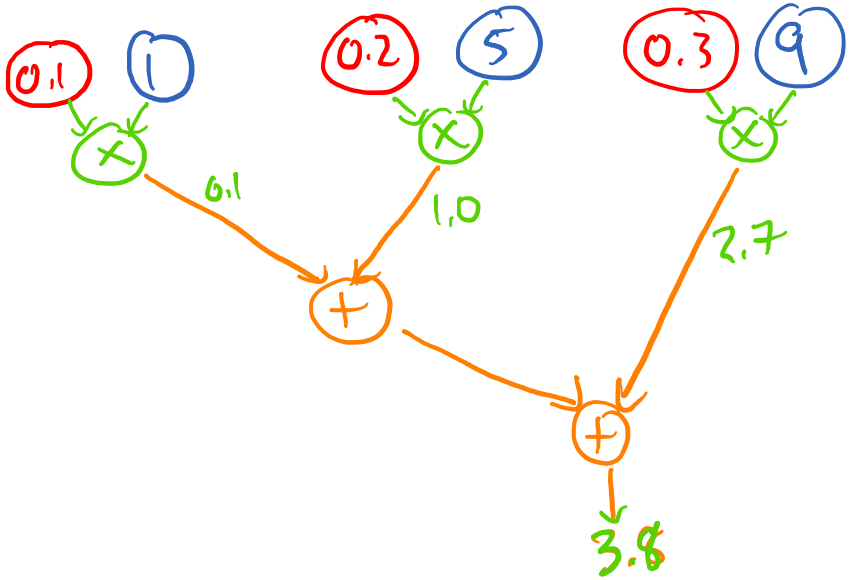

[0.1 0.2 0.3]
1
5
9
= [3.8]
independent
infinite hardware
0.1
1
0.2
5
0.3
9
+8
0.1
1.0
+0 parallel
2.7
+0 parallel
+8
+8
3.8
8+8+8 = 24

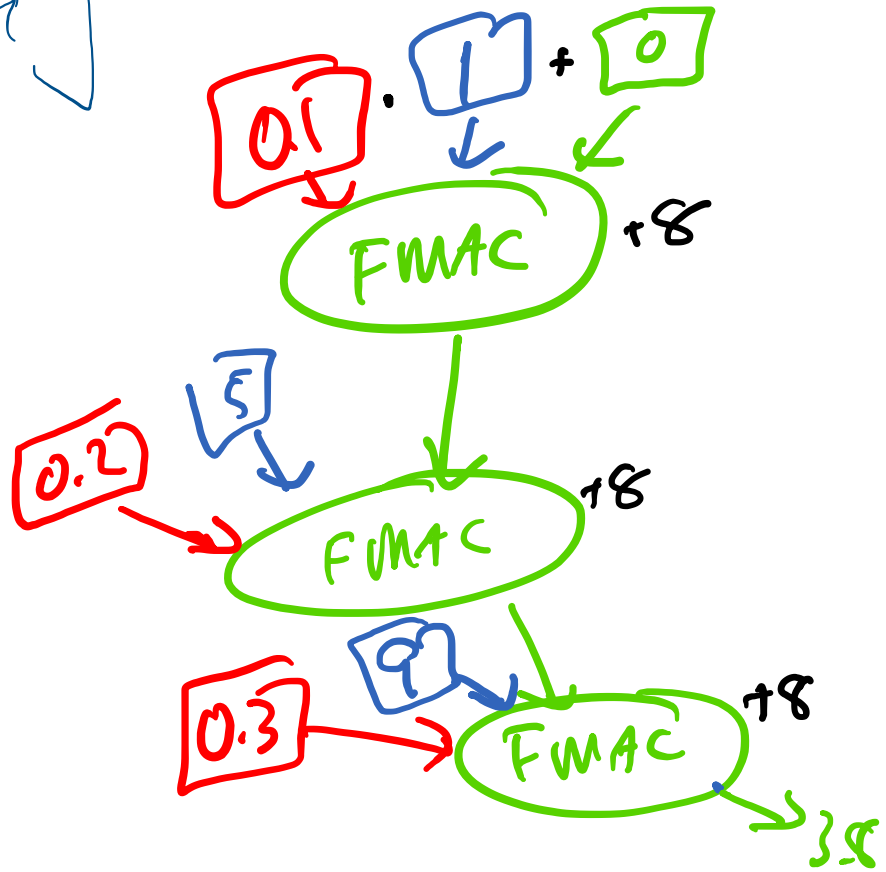
How many cycles *should* this take?



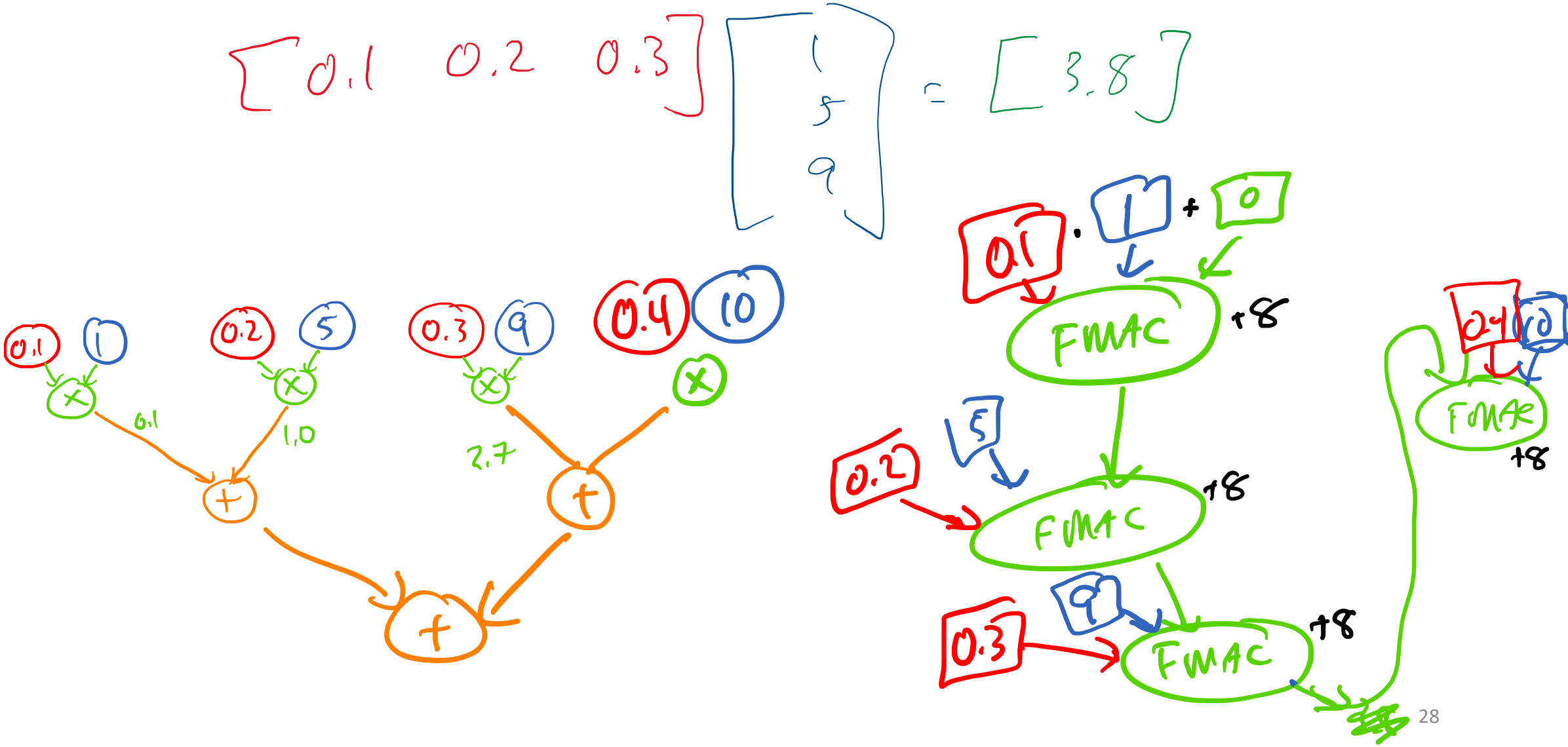
How many cycles *should* this take?

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} 1 \\ 5 \\ 9 \end{bmatrix} = \begin{bmatrix} 3.8 \end{bmatrix}$$

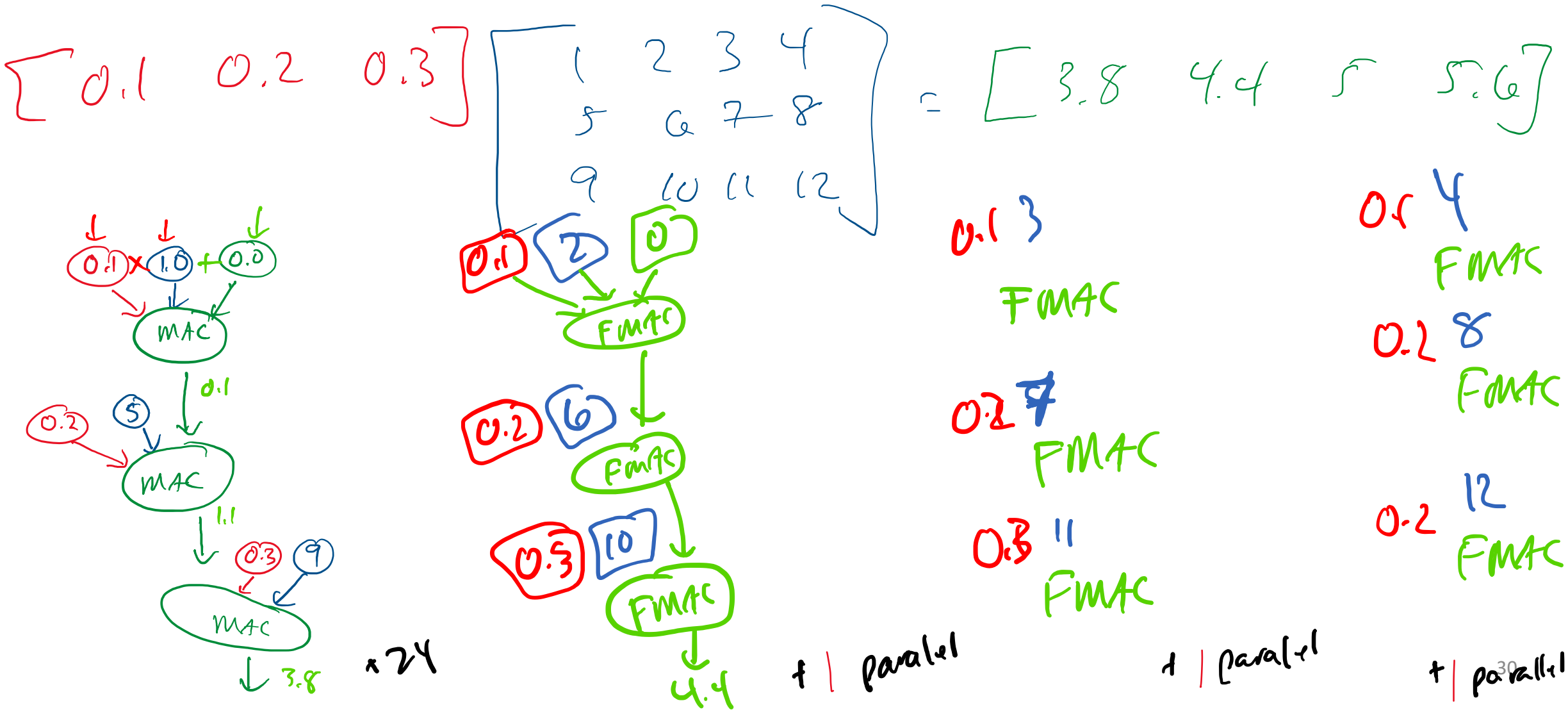


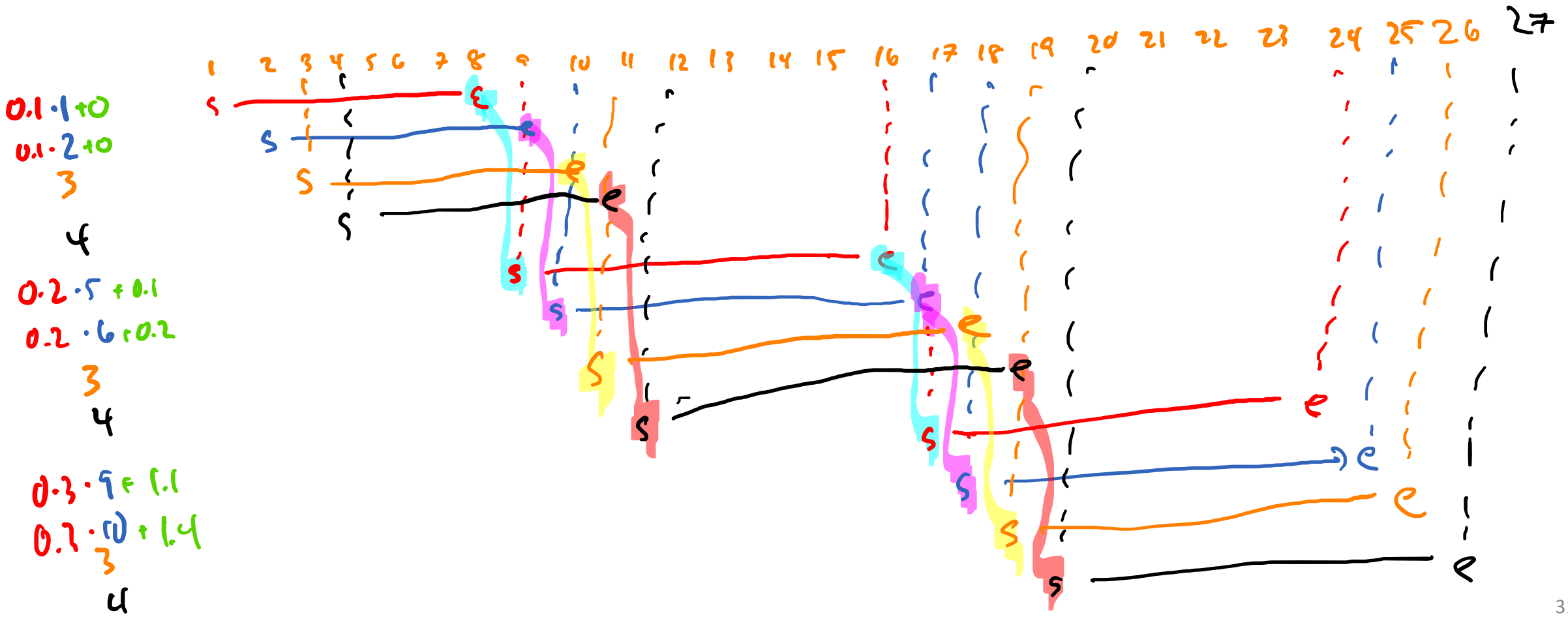


How many cycles *should* this take?



More Data Flow




0.1 · 1 + 0
0.1 · 2 + 0
3
4
0.2 · 5 + 0.1
0.2 · 6 + 0.2
3
4
0.3 · 9 = 1.1
0.7 · 10 + 1.4
3
4
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 3.8 & 4.4 & 5 & 5.6 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}$$

$$0.1 \cdot \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 \end{bmatrix}$$

$$0.2 \cdot \begin{bmatrix} 5 & 6 & 7 & 8 \end{bmatrix} + \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 \end{bmatrix} = \begin{bmatrix} 1.1 & 1.4 & 1.7 & 2.0 \end{bmatrix}$$

$$0.3 \cdot \begin{bmatrix} 9 & 10 & 11 & 12 \end{bmatrix} + \begin{bmatrix} 1.1 & 1.4 & 1.7 & 2.0 \end{bmatrix} = \begin{bmatrix} 3.8 & 4.4 & 5 & 5.6 \end{bmatrix}$$

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} = \begin{bmatrix} 3.8 & 4.4 & 5 & 5.6 \end{bmatrix}$$

	One Output	All 4 Outputs	
		Serial	Parallel
Multiply & Add separate:	24	96	24
MAC	24	96	24
Pipelined MAC	24 + 1 extra column		24 + 1 + 1 + 1 = 27
Maximum Parallel:	24		24

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} = \begin{bmatrix} 9 & 10 & 11 & 12 \end{bmatrix}$$

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} = \begin{bmatrix} 9 & 10 & 11 & 12 \end{bmatrix}$$

0.1 0.2 0.3 0.4
× 1 × 5 × 9 × 13
0.1 + 1 2.7 + 5.2
1.1 + 7.9
9

0.1 0.2 0.3 0.4
× 2 × 6 × 10 × 14
0.2 + 1.2 3.0 + 5.6
1.4 + 8.6
10

0.1 0.2 0.3 0.4
× 3 × 7 × 11 × 15
0.3 + 1.4 3.3 + 6.0
1.7 + 9.3
11

0.1 0.2 0.3 0.4
× 4 × 8 × 12 × 16
0.4 + 1.6 3.6 + 6.4
2.0 + 10
12

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} = \begin{bmatrix} 9 & 10 & 11 & 12 \end{bmatrix}$$

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} = \begin{bmatrix} 9 & 10 & 11 & 12 \end{bmatrix}$$

Next Tim: Working out parallelism