Computer Systems
Lecture 6

Register Transfer Machine

Dr José Cano, Dr Lito Michala
School of Computing Science
University of Glasgow
Spring 2020

Outline

- Calculators and memory
- Register File
  - Behaviour of register File
  - Loading: the demultiplexer
  - Readout: the multiplexer
  - Register file circuit
- Arithmetic: the adder
- Register Transfer Machine

Assignment statements and circuits

- A program uses assignment statements to calculate results

  x := 2 + 2

  *y := 3 * (x+1)*

- **Our goal**: a digital circuit that can do this!

Instructions

- The expression in the right hand side of an assignment statement can be arbitrarily large and complex
- For a digital circuit, we need
  - Simple statements (e.g. just one arithmetic operation)
  - Statements with a fixed form
  - A small number of types of statement
- These are called instructions
- Our circuit will use two instructions, of the form
  - *R2 := 6*
  - *R3 := R1 + R0*

Outline

- Calculators and memory
- Register File
  - Behaviour of register File
  - Loading: the demultiplexer
  - Readout: the multiplexer
  - Register file circuit
- Arithmetic: the adder
- Register Transfer Machine

Register file

- A register file is an array of registers: R0, R1, R2, R3, etc
- Each variable is held in a register
- We refer to a variable by its address
  - An address is a binary number 0, 1, 2, 3, …
- What the register file circuit does
  - Contains the array of registers
  - Each register holds a word (a binary number)
  - Enables to specify an address and read out that register
  - Enables to specify another address and load a data value into that register

Behaviour of register file

- The behaviour is
  - At clock tick: if *ld = 1* then *Reg[d] := x*
  - During clock cycle: *a = Reg[sa]* and *b = Reg[sb]*

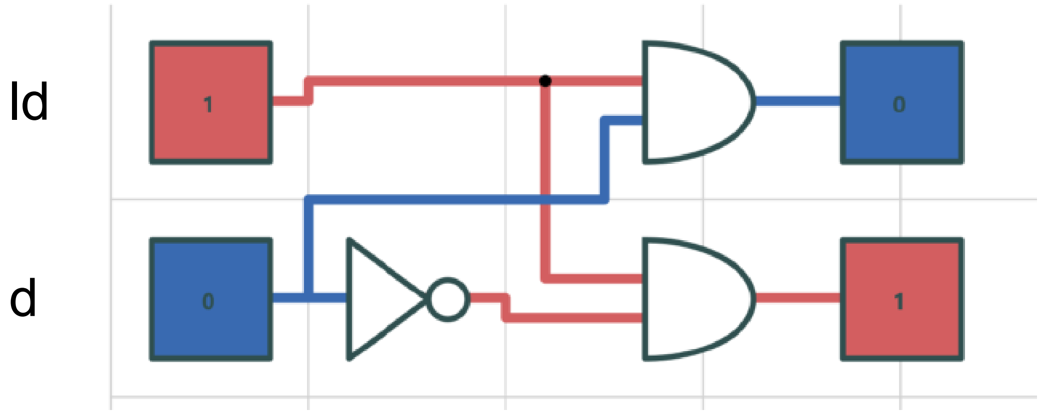
- The operation is determined by the control signals ld, d, sa, sb

- The data input is x, and there are two data outputs a, b

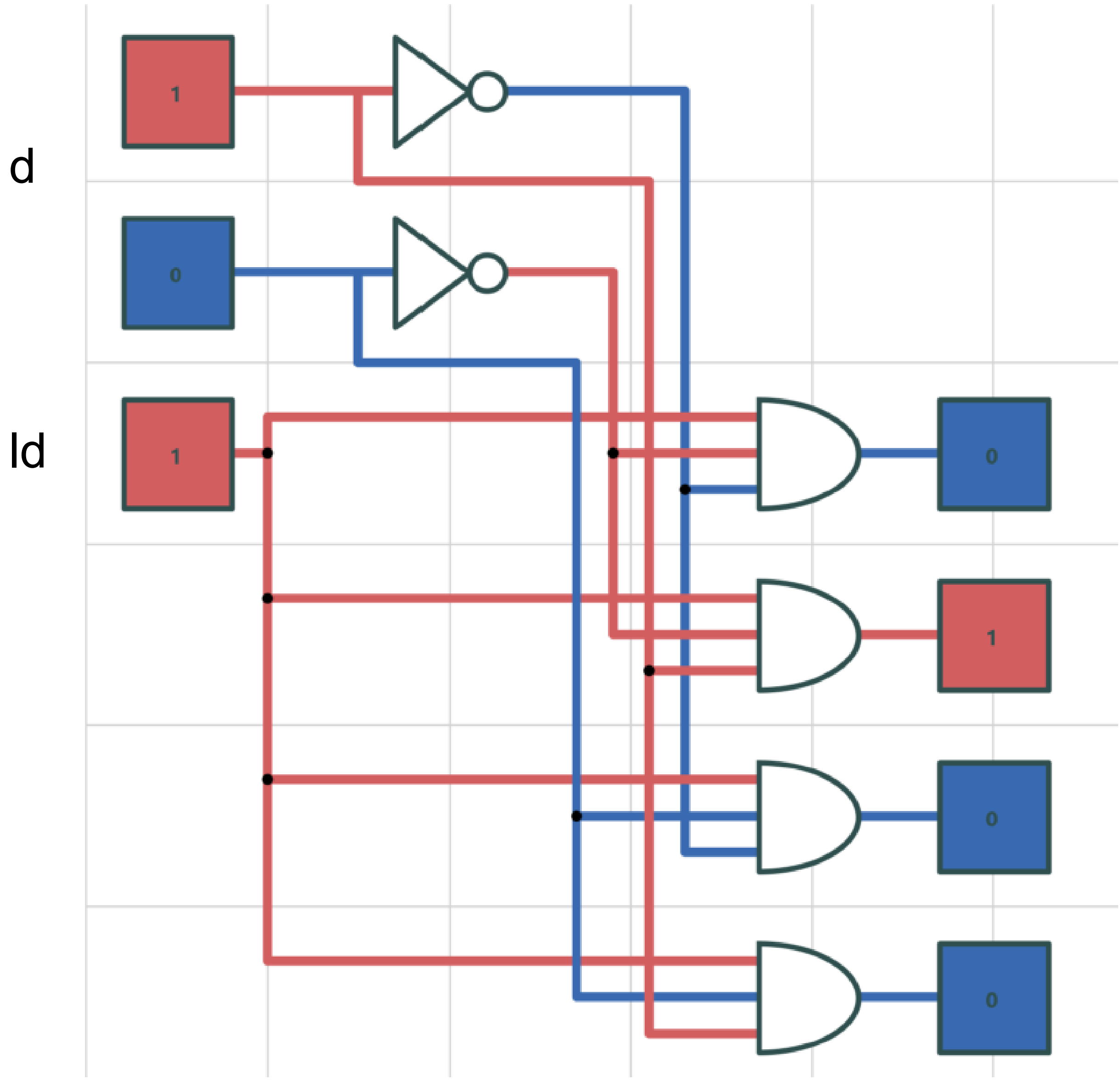
Loading into a register file: demultiplexer

- Often, every register in the register file retains its previous state
- Sometimes we will modify one (just one!) register in the register file
- We need a way to take a destination register number (R0 or R1) and tell just that register to perform a load
- This is done with a demultiplexer
- R0 and R1 each need an individual load control
- The register selected by *d* gets the register file *load* input
- The register not selected by *d* gets a load control of 0

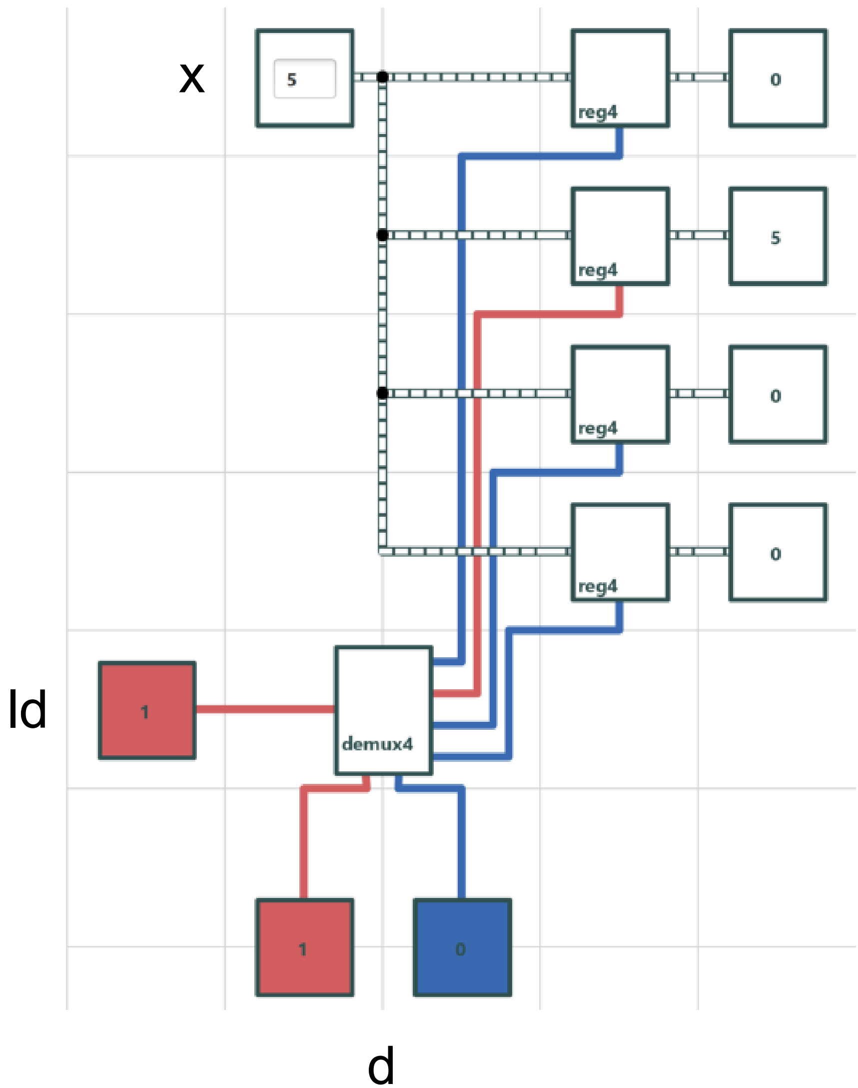
demux1: generate 2 control signals from a bit address



demux2: generate 4 control signals from 2-bit address



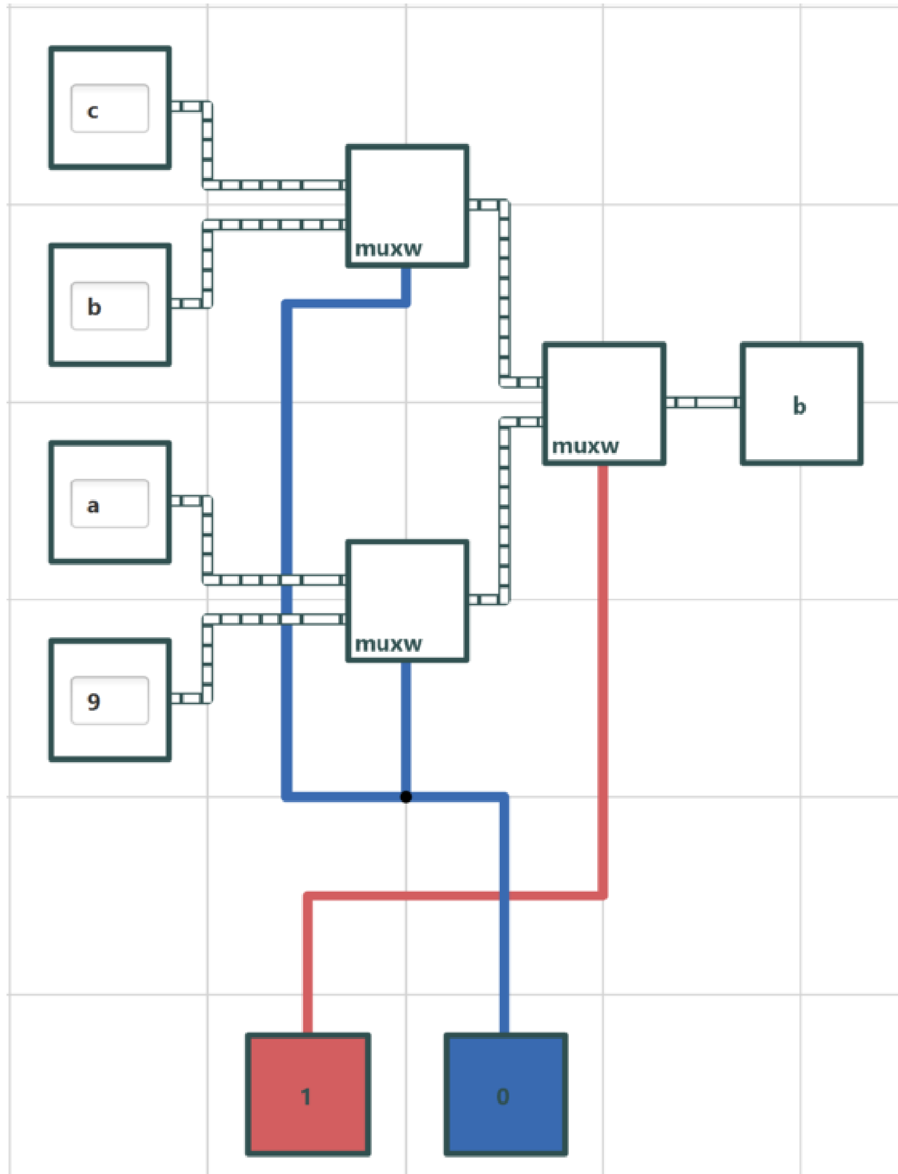
Register file with demultiplexers: R2 := x



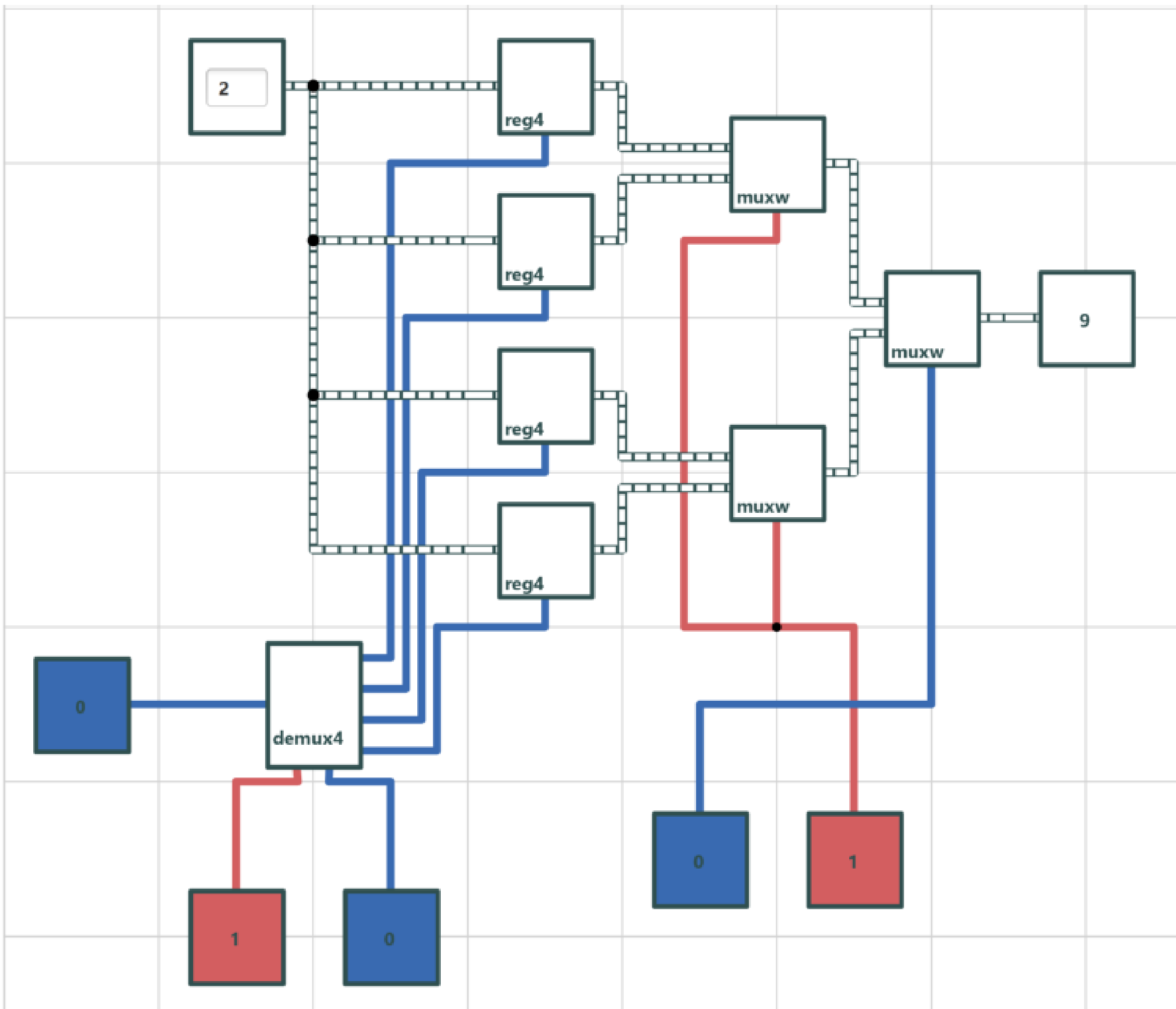
Multiplexers

- A bit-level multiplexer mux1 uses a control bit to choose between two data bits
- The word-level multiplexer wmux1 uses a control bit to choose between two data words
- A multiplexer for 4-bit words consists of 4 copies of the basic multiplexer

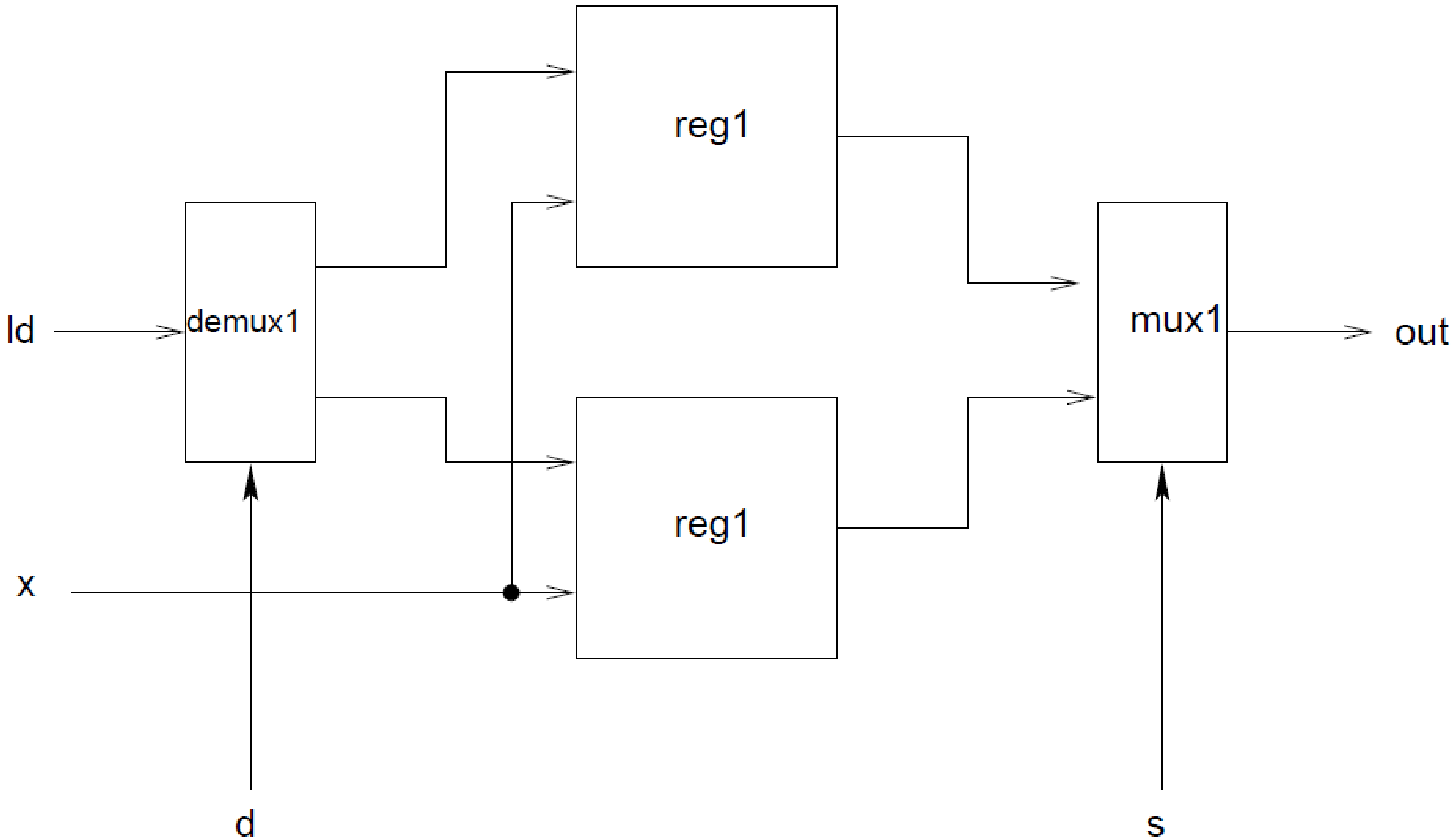
mux2w: choose among 4 values with 2-bit address



Multiplexers: readout R1



Simple register file: 2 registers, 1 readout

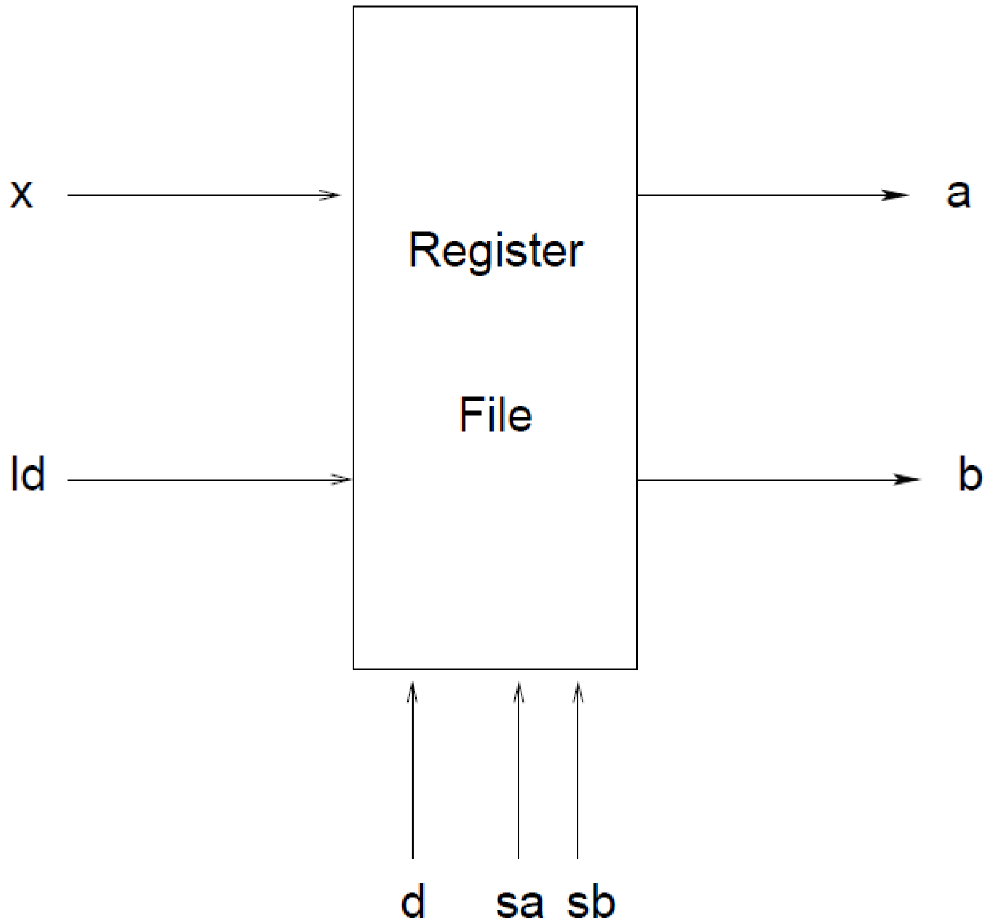


The output is the value of *reg[s]* - At a clock tick, if ld then *reg[d] := x*

Register File with 2 readouts

- Our aim is to do calculations like p + q or x - y
- The variables will be in registers
- So we need to read out two registers, not just one
- Therefore we extend the register file so we give it two source addresses sa, sb and it will read out both
  - *a = reg[sa]*
  - *b = reg[sb]*

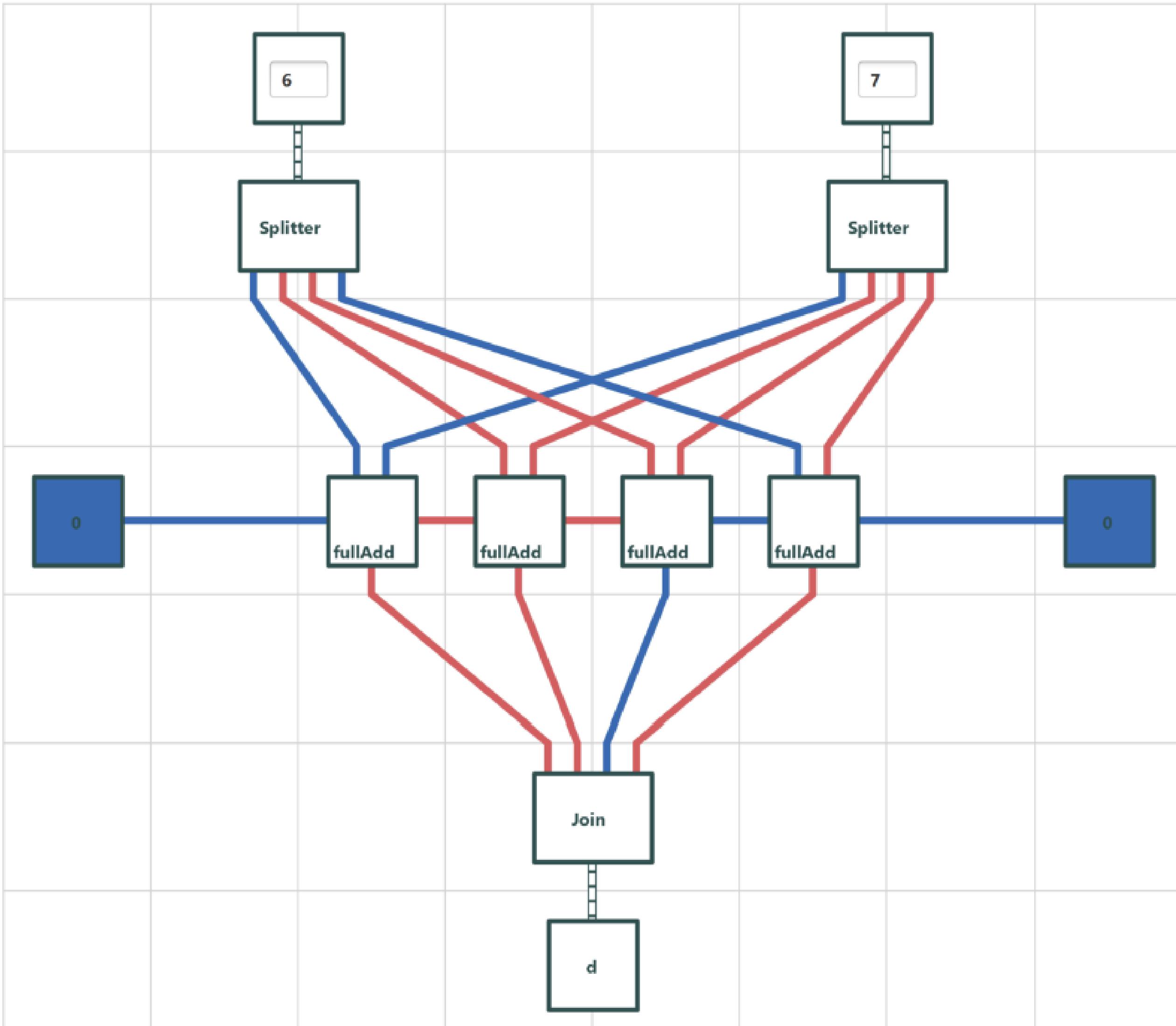
Register File with 2 readouts



Outline

- Calculators and memory
- Register File
  - Behaviour of register File
  - Loading: the demultiplexer
  - Readout: the multiplexer
  - Register file circuit
- Arithmetic: the adder
- Register Transfer Machine

add4: 6 + 7 = d



6
7
Splitter
Splitter
0
fullAdd
fullAdd
fullAdd
fullAdd
0
Join
d

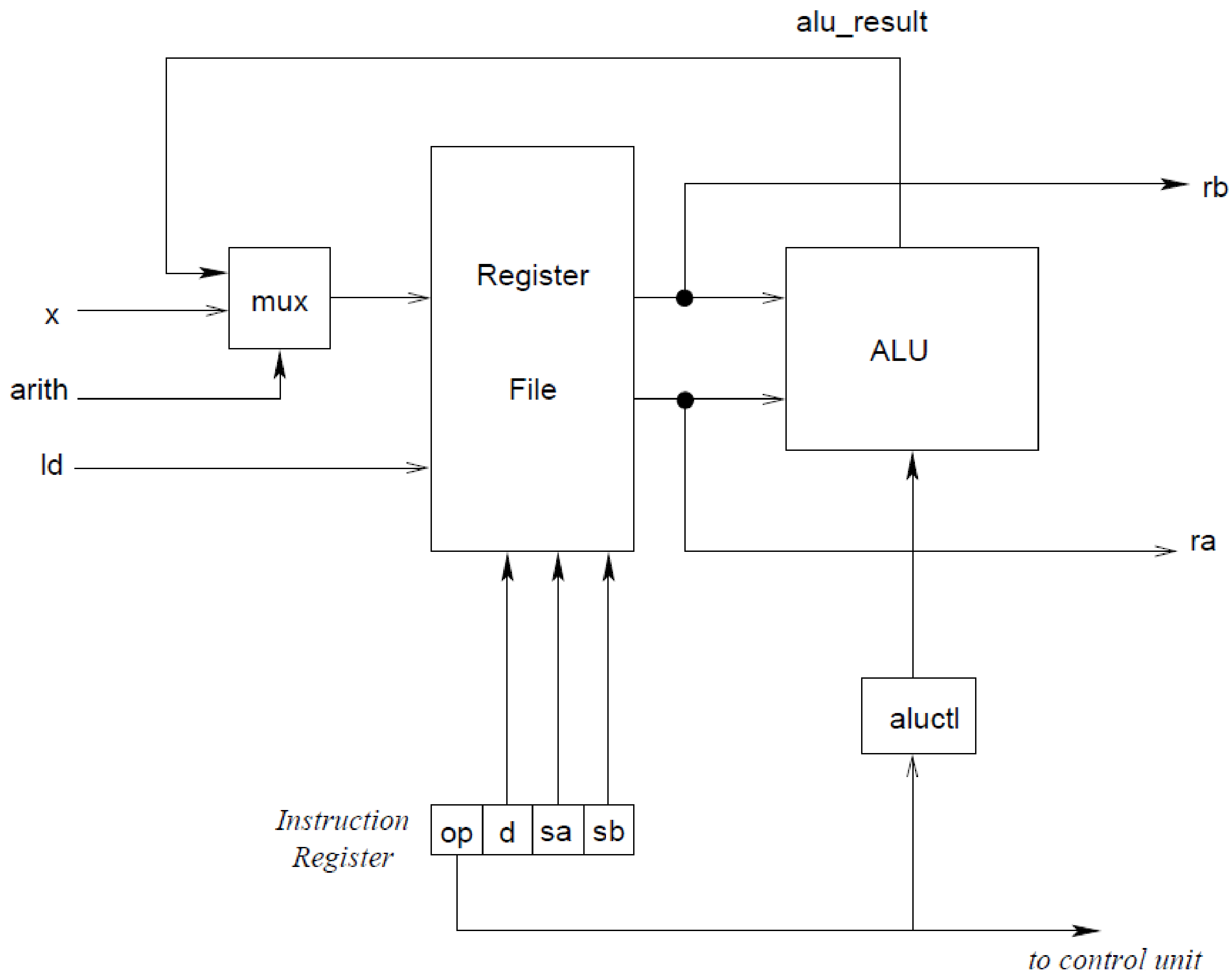
Outline

- Calculators and memory
- Register File
  - Behaviour of register File
  - Loading: the demultiplexer
  - Readout: the multiplexer
  - Register file circuit
- Arithmetic: the adder
- Register Transfer Machine

Register Transfer Machine (RTM)

- The register file produces two outputs and the adder requires two inputs
- The adder produces one data word output, and the register file takes one data word input
- We can connect them in a feedback loop!
- The effect will be (at a clock tick)
  - *if ld then reg[d] := reg[sa] + reg[sb]*
- Add an external data input, and use a multiplexor to decide whether the external input or the ALU result should be sent to the register file

Register Transfer Machine: Circuit



Behaviour of RTM

- Data input x

- Control inputs ld, arith

- Address inputs d, sa, sb (specify registers)

- At clock tick

  if ld

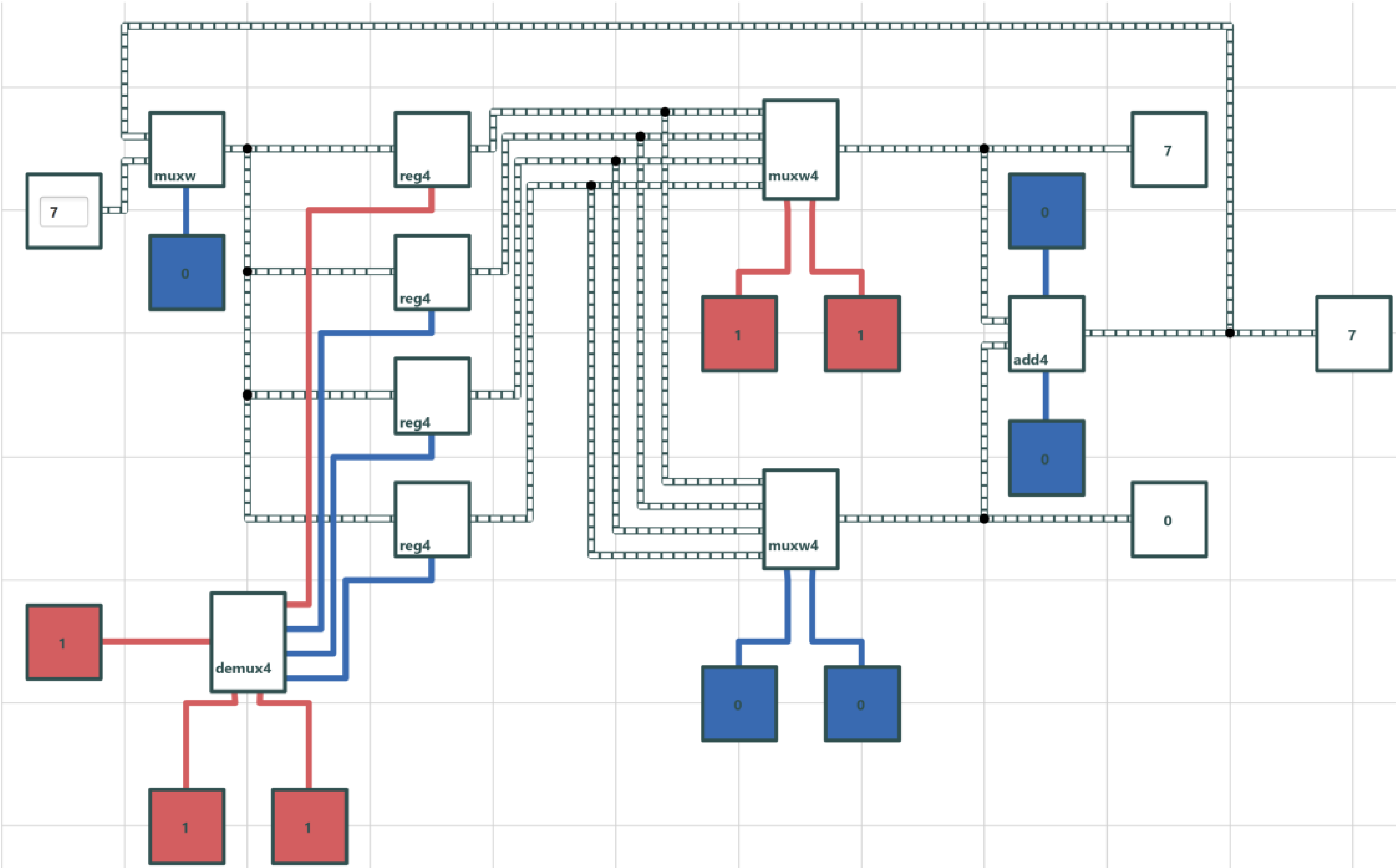
  then if arith

  then reg[d] := reg[sa] + reg[sb]

  else reg[d] := x

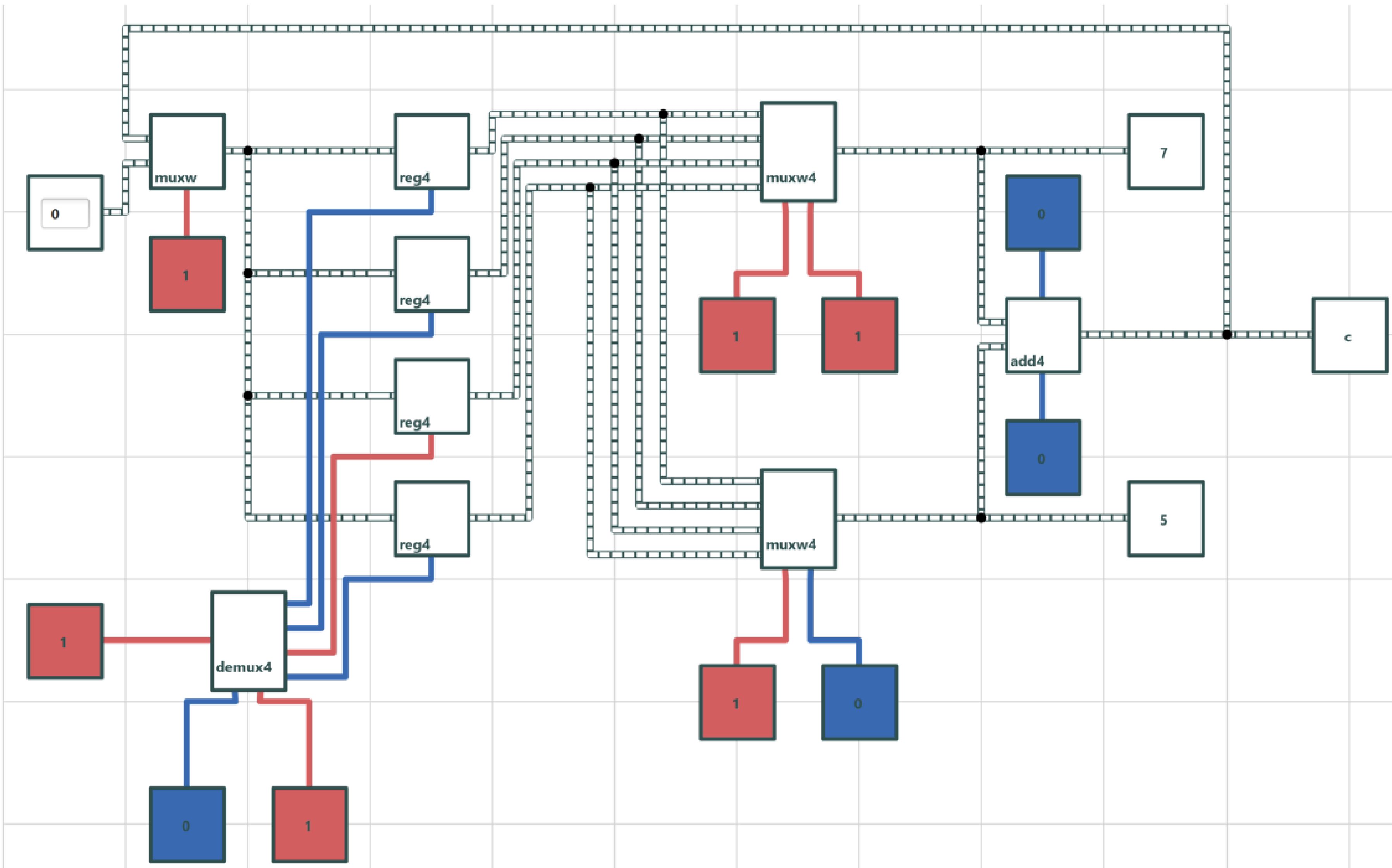
- All other registers remain unchanged (the variables you aren’t assigning to don't change)

RTM: R3 := 7



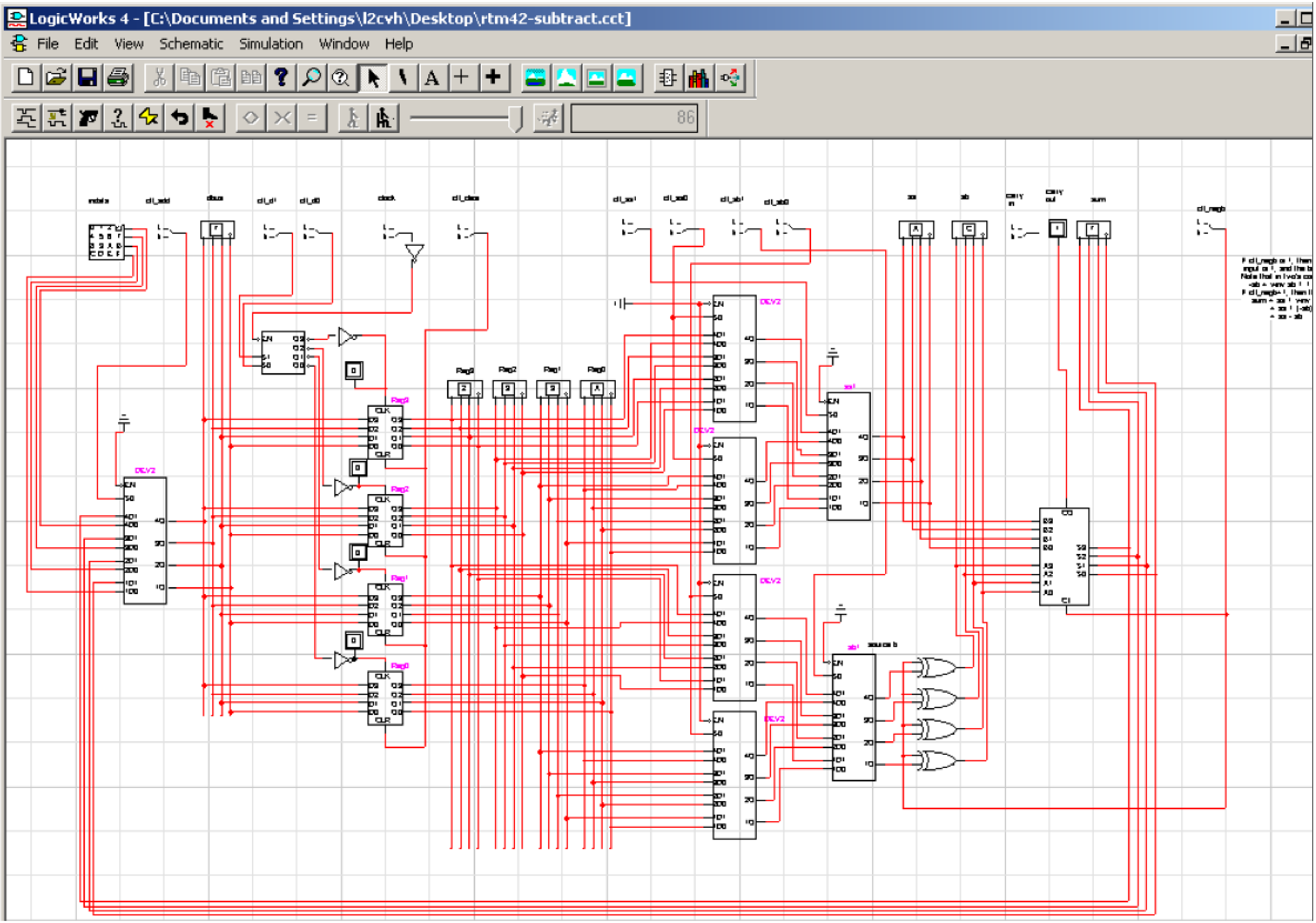
muxw
7
0
reg4
reg4
reg4
reg4
muxw4
1
1
muxw4
0
0
7
0
add4
0
7
0
1
demux4
1
1

RTM: R3 := 7; R2 := 5; R1 := R3 + R2



Register Transfer Machine in Logic Works

- We have simplified the description of the circuit using 4-bit components and 4-bit signals
- But in the real hardware, all of the logic gates and wires are really there
- Real circuits are complicated!


LogicWorks 4 - [C:\Documents and Settings\l2cvh\Desktop\rtm42-subtract.cct]
File Edit View Schematic Simulation Window Help
86

Controlling a circuit

- The “core” of the circuit is a register file and an adder
- To make it perform useful computations, we need to control it
- This is done with control signals
- A control signal is a bit (either 0 or 1), just like any other signal
- But it helps conceptually to make a distinction between data signals and control signals

An analogy: Marionnette

- There are lots of systems with
  - A core part that does something
  - An external operator that tells the core what to do
  - Control signals that enable the operator to control the core

- A traditional example: the Marionnette
  - A doll that can move around, like an actor
  - A person (out of sight) manipulating the doll
  - Strings the person can pull, controlling the doll

- This is the origin of the expression “to pull strings” or “to pull someone's strings”

- See one in action:
  - https://www.youtube.com/watch?v=SPBm8I7hoBQ

Marionnette: controlled by pulling strings

- By SoHome Jacaranda Lilau, Tamelifa Puppeters, Pierre S Frana Line - Own work, CC BY-SA 3.0 https://commons.wikimedia.org/w/index.php?curid=8392787

Running a simple program on RTM

- For now, let's just consider a fixed sequence of simple assignment statements
- Each variable must be a register: R0, R1, R2, R3
- Each statement must be written in one of these forms
  - reg = constant
  - reg = reg + reg
  - reg = reg - reg
- Example:

  R1 = 3

  R2 = 1

  R3 = R1 + R2

  R1 = R2 - R2

- We can execute this program on the RTM!

What can you make the circuit do?

- You'll operate the circuit so that it will execute little programs like this

```
R0 := 4          ; R0 = 4
R1 := 6          ; R1 = 6
R2 := R0 + R1    ; R2 = a
R0 := R0 + R2    ; R0 = e
R3 := 1          ; R3 = 1
R1 := R1 + R1    ; R1 = c
```

How to execute R1 := 5

- Tell it to use the indata input (the hex keypad) as the data input to register file: ctl_add = 0
- Tell it to put the data into R1: ctl_d1, ctl_d0 = 01
- Tell it to perform a load
  - Actually, the circuit does a load every clock cycle, but in a real computer we would have to set ctl_ld = 1
- **Note**: the values of *ctl_sa1*, *ctl_sa0*, *ctl_sb1*, *ctl_sb0* don't matter
  - The registers they specify will be read out and you can look at them on the hex displays, but these values won't actually be used
- Perform a clock tick: pulse clock (click it to 1, then click it to 0)

How to execute R2 := R0 + R3

- Tell it to read out R0 on the a output: ctl_sa1, ctl_sa0 = 00
- Tell it to read out R3 on the b output: ctl_sb1, ctl_sb0 = 11
- Tell it to use the adder output as the data input to register file: ctl_add = 1
- Tell it to put the data into R2: ctl_d1, ctl_d0 = 10
- Tell it to perform a load
  - Actually, the circuit does a load every clock cycle, but in a real computer we would have to set ctl_ld = 1
- Note: the value of indata is ignored, so you can set it to any value you like, makes no difference
  - This is called a “don't care” value
- Perform a clock tick: pulse clock (click it to 1, then click it to 0)

Beyond the RTM

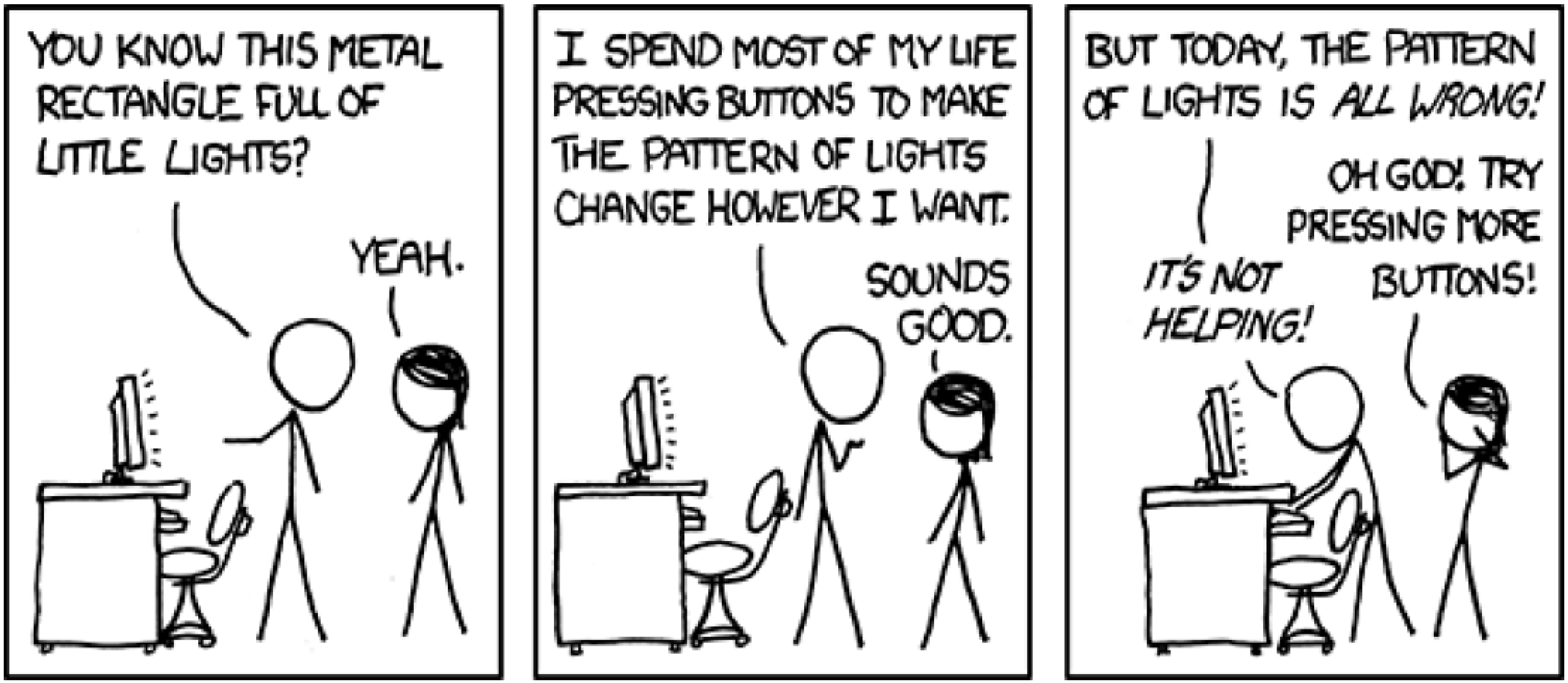
- The RTM can execute a simple sequence of assignments

- We can extend it gradually to turn it into a computer
  - Enable it to subtract as well as add
  - Provide a way to set the control signals automatically, not by hand
  - Give it the ability to make decisions (if-then-else)

- We won't add all these capabilities to the circuit, but will look at the end result

To do

- Be sure to complete last week's quiz
- This week's quiz will be available after lecture
- Study the key circuits: register file and register transfer machine
- You don't need to memorise the circuits
- Just understand the general ideas



https://xkcd.com/722/