SER 232

Computer Systems Fundamentals I

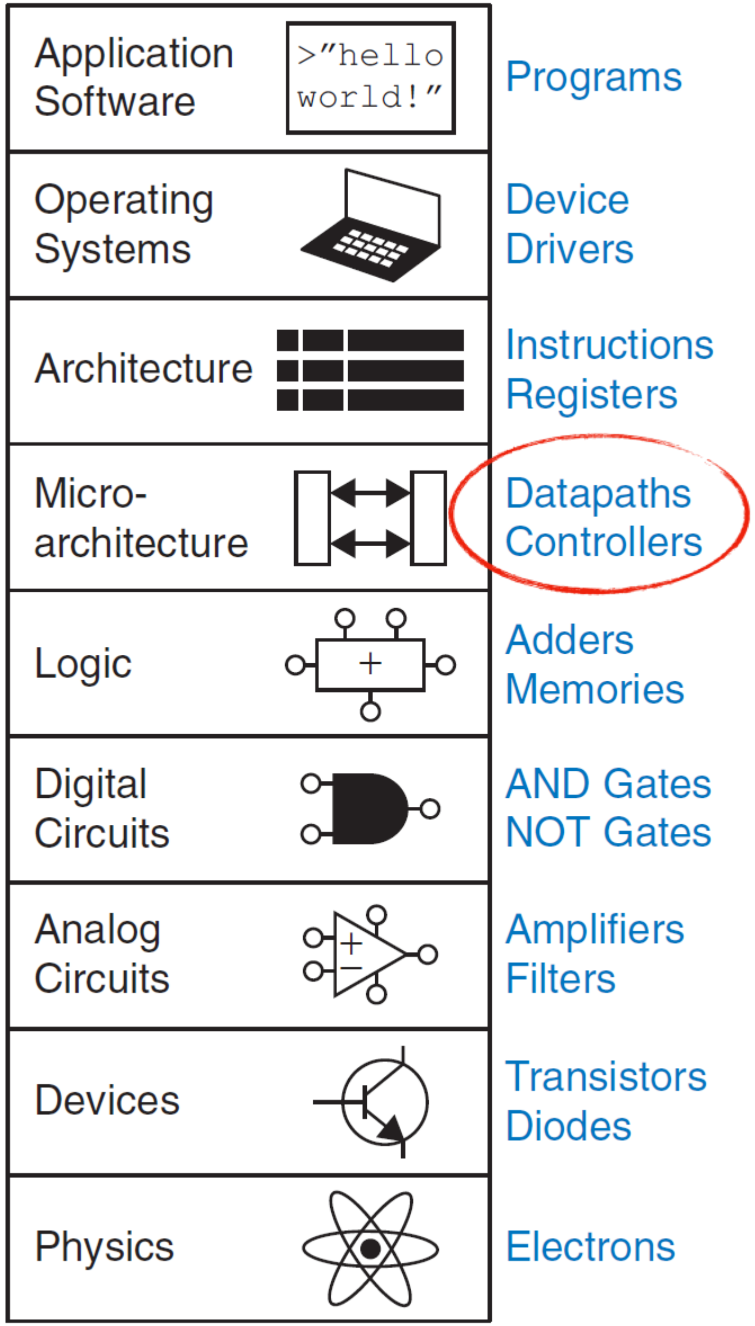
Dr. Heinrichs

Topics

- Design & Abstraction Levels
- Data vs. Control

Higher-Level Design

- Transistor-level design
- Logic-level design
  - Logic gates made up of transistors
- Register-transfer level (RTL) design
  - Datapath components made up of logic gates



Application Software	Programs
Operating Systems	Device Drivers
Architecture	Instructions Registers
Micro-architecture	Datapaths Controllers
Logic	Adders Memories
Digital Circuits	AND Gates NOT Gates
Analog Circuits	Amplifiers Filters
Devices	Transistors Diodes
Physics	Electrons

Data vs. Control

- Data:
  - A value that comes from the datapath (registers) or other devices:
    - Buttons
    - Sensors
    - Memory (RAM)
    - Secondary storage (HDD or SDD)

Data vs. Control

- Control:
  - Indicates what do with data
  - Source:
    - Custom processor:
      - Determined by controller (current state and inputs)
    - General purpose microprocessor:
      - Control signals dictated by program instructions

Data vs. Control

- Analogy: TV
    - Control: Remote gives control (volume, channel, brightness, …)
    - Data: Content of channel (video, audio)

Datapath Components

- Data storage
  - Registers
- Data manipulation
  - Boolean and arithmetic operations

SER 232

Computer Systems Fundamentals I

Dr. Heinrichs

Topics

- Datapath Components: Data Storage

Register

- Storage device for multiple bit values
  - Bus input for storing a new value
  - Bus output for reading currently stored value
- Usually stores data if the clock input sees a *rising edge*

Parallel-Load Register

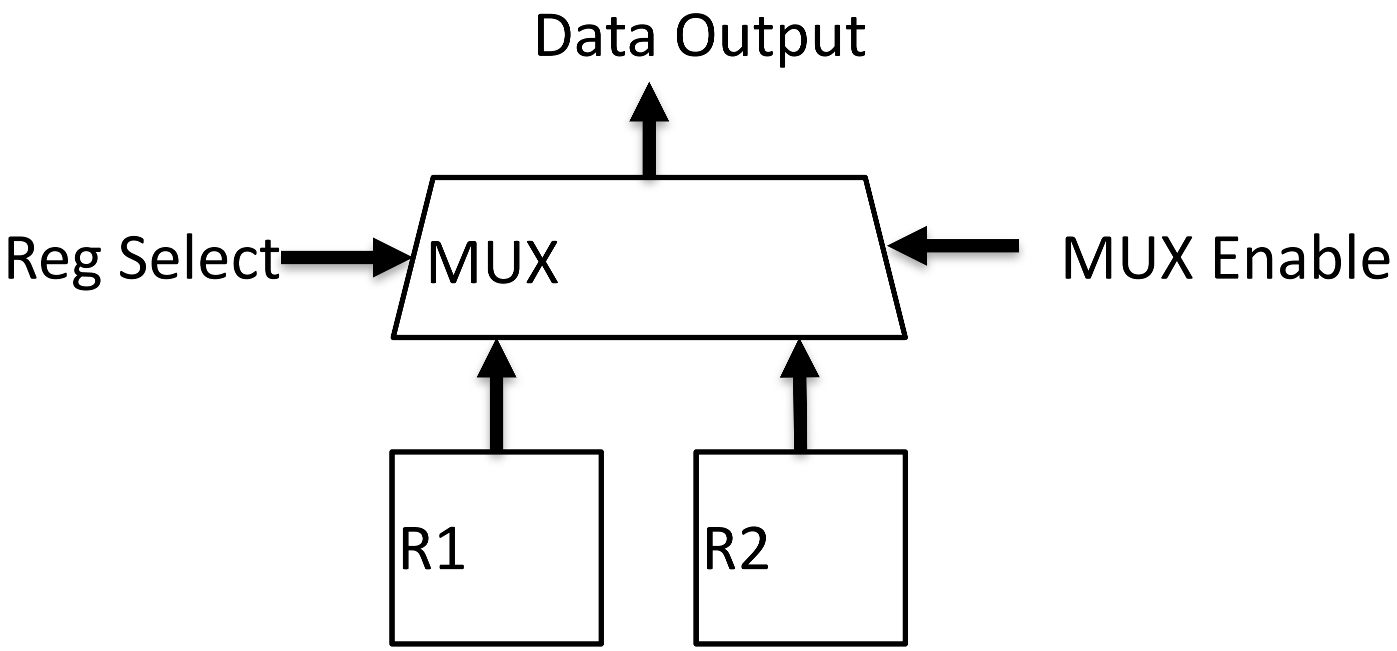
- The *load* control line is equivalent to Logisim register's *en* (enable) pin
  - Method for preventing clock ticks from updating the register value
  - Gives control over when the register is supposed to store values
  - *en = 0* means the register will retain stored value, independent of rising edges from clock and changed input values

Register Files

- A collection of registers
- Separate control inputs for reading and writing to:
  - Select a register to read/write from/to
  - Enable access to write and/or read
- Uses multiplexer to route data from selected register to output
  - Register file can have only 1 output but several registers to store data in and load from

Register File Example

- Example for the reading part of a register file



Logical Shift Right

- Suppose we want to shift the value 0110_2 to the right once
  - Programming example:

```
y = 0b0110 >> 1
```

0	**1**	**1**	**0**

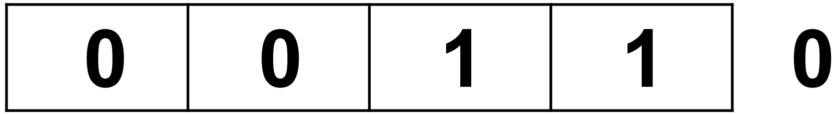
Logical Shift Right

- Each bit is moved right one position

0	1	1	0

Logical Shift Right

- Least significant bit is removed
- Most significant bit become a zero



Logical Shift Right

- The original value: $0110_2 = 6_{10}$
- The new value: $0011_2 = 3_{10}$
  - Shifting right is equivalent to dividing by 2

0	**0**	**1**	**1**

Logical Shift Left

- Same process as shifting right
  - Each shift to the left is equivalent to multiplying by 2
  - Programming example:

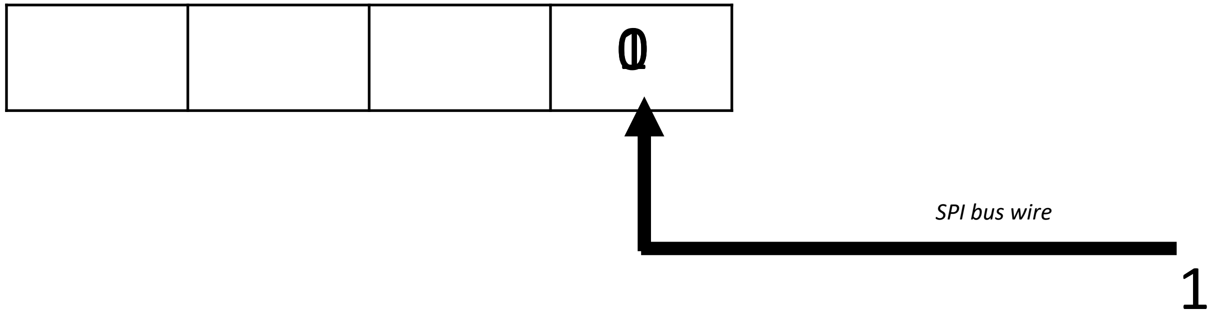
    ```
    y = 0b0110 << 1
    ```

  - Shifting result:

    ```
    y = 0b1100
    ```

Shift Registers

- A register with a control inputs that causes the stored value to be shifted
  - E.g. shift register is used for a communication bus called SPI (serial peripheral interface), which transmits bits using 1 wire (1 bit at a time)



Shifters

- Shifting can also be done by components that do not include a register
- More sophisticated shifters, such as a barrel shifter, can shift by a specified number of bits
  - x = x << 4

SER 232

Computer Systems Fundamentals I

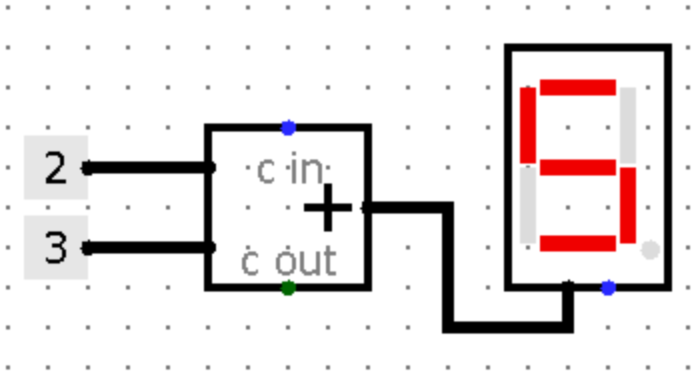
Dr. Heinrichs

Topics

- Datapath Components: Data Manipulation

Adders

- Component adds two inputs (and a carry)
    - Outputs result and a carry bit



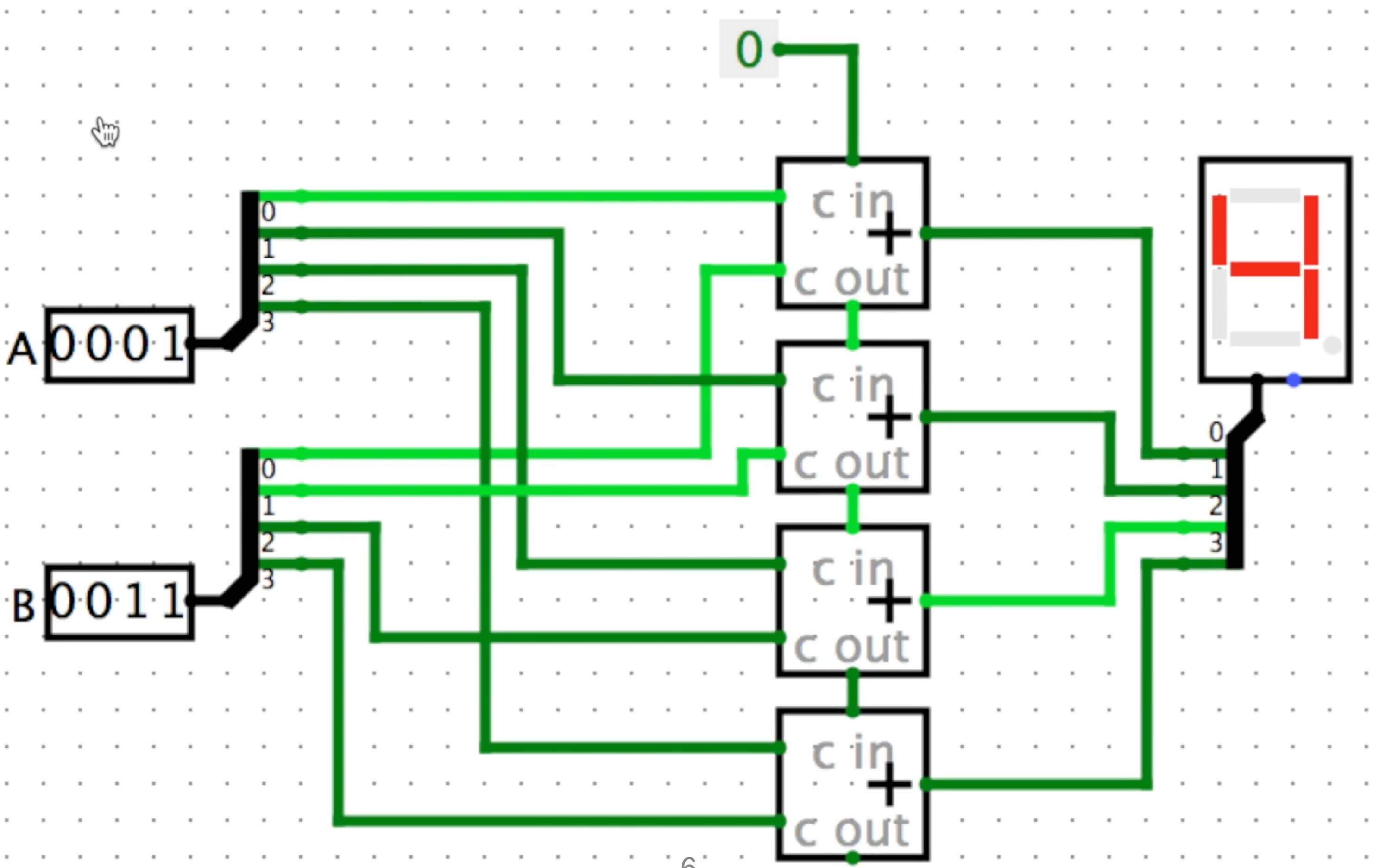
Adder: Ripple-Carry Style

- Half adder:
  - Two 1-bit number inputs
  - Result and carry output
- Full adder:
  - Two 1-bit number inputs and a carry input
  - Result and carry output

Adder: Ripple-Carry Style

- N bit adder made up of N-1 full adders (FA) and a single half adder (HA)
  - Each adder handles a specific bit position
  - Carry out of each adder goes to the next higher bit position adder
    - HA used in LSB where it doesn't need a carry input

Adder: Ripple-Carry Style

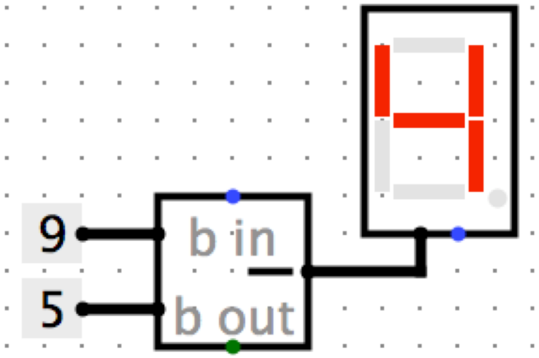


Subtraction

- Logisim subtraction similar to adder, but has borrow (b) in and out

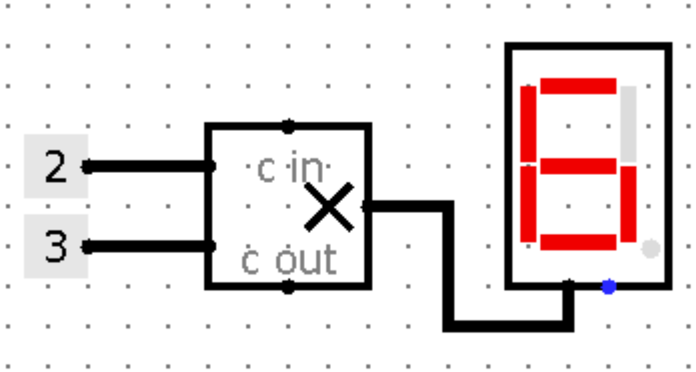
```
   0010
-  0001
=  0001
```

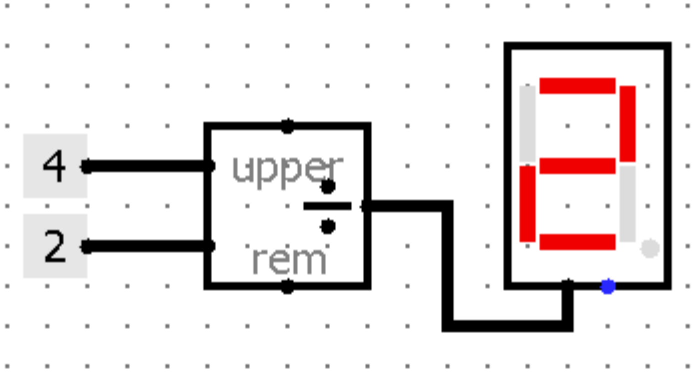
- Position 0: We are performing 0b0 minus 0b1, which does not work
  - We have to borrow a 0b1 from position 1 to perform 0b10 minus 0b1, which results into 0b1
  - Borrowing 0b1 from position 1 will turn it from 0b1 to 0b0



Multiplier & Divider

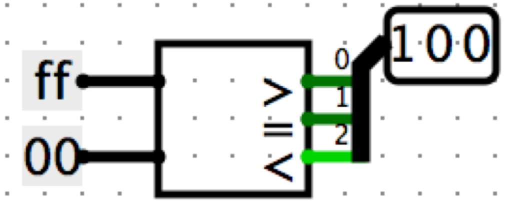
- Component multiplies / divides two inputs
  - Outputs result





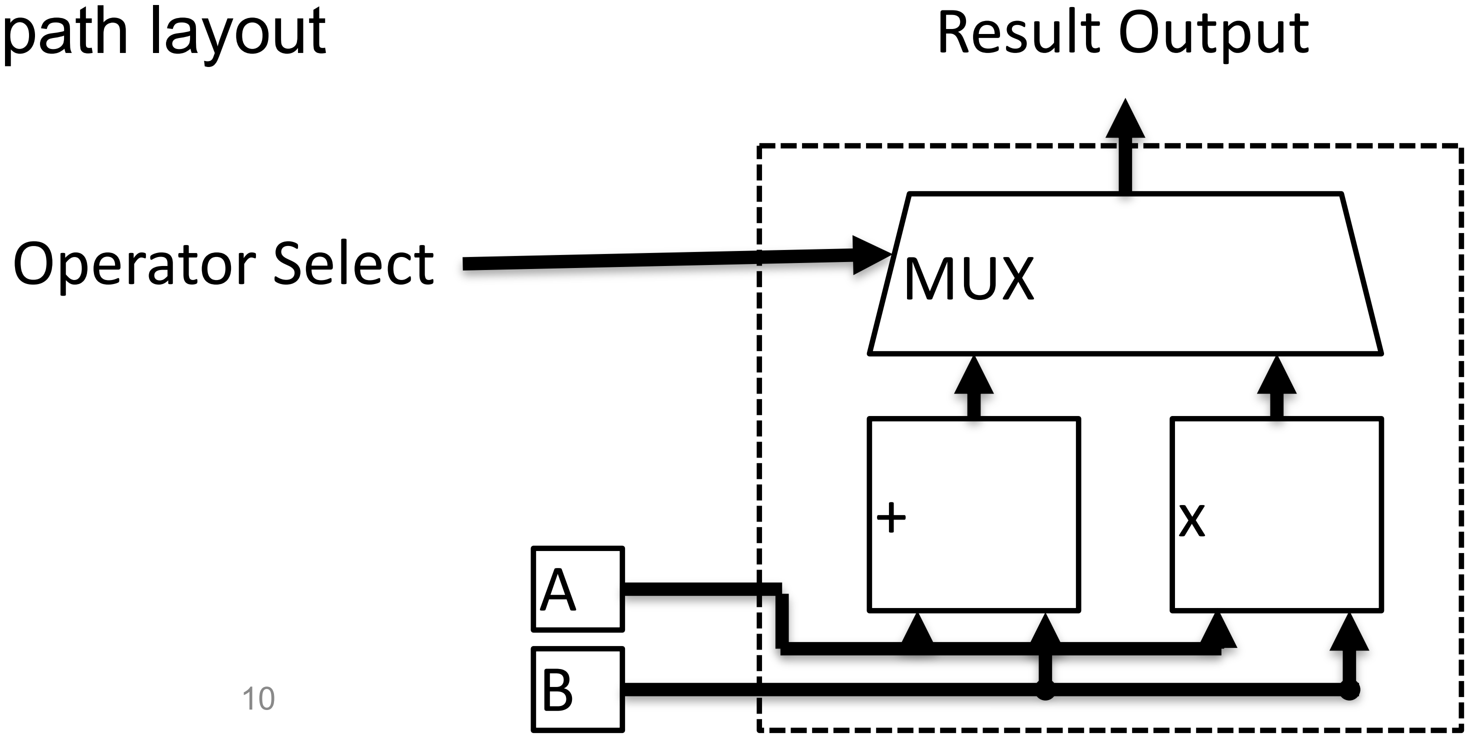
Comparators

- Equality
  - Are two inputs the same?
- Inputs *A* and *B,* and 3 Boolean outputs:
  - A > B
  - A = B
  - A < B



ALU

- Arithmetic Logic Unit
  - Merges multiple arithmetic operator components into a single component
    - Simplifies datapath layout


Result Output
Operator Select
MUX
+
x
A
B