Faculty of Science, Engineering and Technology

Computer Systems

Week 4

Overview

In this laboratory session we start to look at memory, encoders and stacks.

Name: Le Quang Hai

Student ID: 104175779

Theory (Memory, Architectures, Interrupts and Stacks)

1.1. What is ROM and what is its primary purpose ?

→ ROM stands for read-only memory. It is usually pre-programmed and can hardly be modified. Purpose: to store permanent data even when power is off.

1.2. What is RAM and how is it different from ROM ?

→ RAM stands for random-access memory, used for temporary storage

→ Differences: In RAM, data is lost when there is no power; data can be read, written or updated and can be accessed quickly and conveniently since data is indexed with an address

1.3. What is the difference between static RAM and dynamic RAM ?

→ Static RAM requires constant and stable electric flow to retain data. It provide (about 1000 times) faster access

→ DRAM only requires frequent refresh of lower power to retain the data. It has smaller area for storing 1 byte that allows more data to be stored within the same size of RAM

1.4. What type of memory is typically used in USB thumb drives ? Why shouldn't we rely on this for critical data storage ?

→ USB uses EEPROM because it has the property that data is not lost if power is off. Furthermore, it is portable and allows data to be read and rewritten.

→ We cannot rely on EEPROM since it has limited number of using times and data can be lost or corrupted after a specific amount of time if not used

2. Consider a computer with 1GB RAM (1024 MB). Given memory addressing is for each byte, how many bits are needed to address all bytes in the system's RAM ?

→ 1GB = 2^10MB = 2^20KB = 2^30B. N bits can address 2^N bytes

→ no. of bytes addressed = lg(n) = lg(2^30) = 30

3. Give a brief description of the Von Neumann and Harvard computing architectures. What are the fundamental differences between the two and for what is each designed to achieve ?

- Von Neumann architecture: Control bits and data bits share a common memory space, using the same bus system. Allows processes to be interrupted while higher priority tasks are executed, and then restored and resumed using stack.

- Harvard architecture: Instructions and data are kept separate, using different bus data. This allows the CPU to access both data and instructions simultaneously to run faster, more secure, more expensive, less extendable, immune to buffer overflow attacks

4. What is cache memory and what is its primary role ?

→ Cache is a memory type used by the CPU to store most frequently used data. Primary role: it enhances performance by reducing time getting data from the memory.

5.1 Explain the concept of an interrupt, and list four common types.

→ An interrupt is a signal that requests attention from the CPU which is handled using the CPU stack. Common types:

- Timer (clock)
- IO events (Keyboard/Mouse)
- Errors (divide by zero), Exception (generated by a try/catch instruction)
- Network (packet received by the NIC)
- Hardware (power button, CDROM ready)

5.2 Polling is an alternative to interrupts ? Briefly explain polling and why it is not commonly used.

→ Polling includes the process of checking state/input of each hardware device in sequence to opt and conduct appropriate processes. It is easy to implement however there are notable disadvantages:

- Time wasting checking hardware
- No taking advantage of the stack.
- One device freezing can cause the entire computer to be unresponsive.

6. Explain the general concept of a stack - how do they work, and what is their primary purpose.

→ Stack is a region memory whose way of organization does not require address to access the data. It provides temporary space to store during execution and can be useful in interrupt handling and function call in programming

6.1. How are stacks useful for handling interrupts ?

→ When an interrupt with higher priority is detected, the current CPU state, such as IP and register value, is pushed to the stack, making way for the task to be executed, after finishing the previous process resumes. If the signal has lower priority than the current task, the coming interrupt will then be pushed into the stack

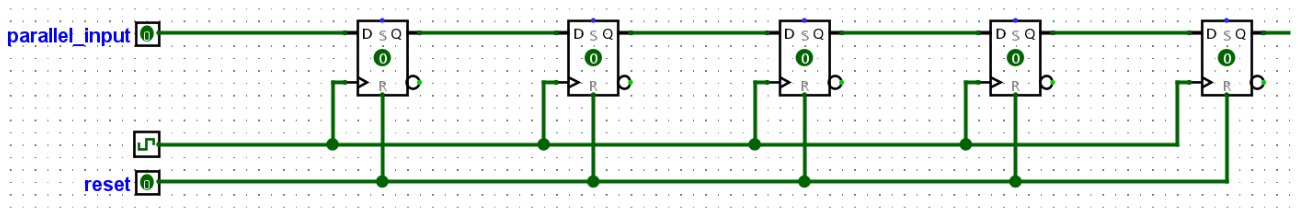
6.2. How are stacks useful in programming ?

→ In programming, stack is useful in recursion, where a function calls itself, and the current function needs to be stored in order to execute the invoked function within itself. Stack structure works perfectly since the former task is stored at the 'bottom' waiting for others' results. Furthermore, when a function is called, all needed variables and parameters can be stored within a stack in order.

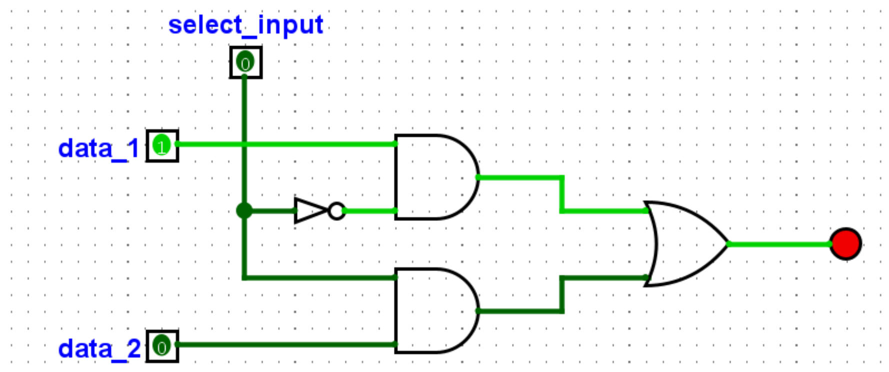
Provide all the answers to the above questions in your submission document.

Practical - Stacks of Stacks !

9. Start by building a simple shift register that moves bits from one flip flop to the next each clock pulse.



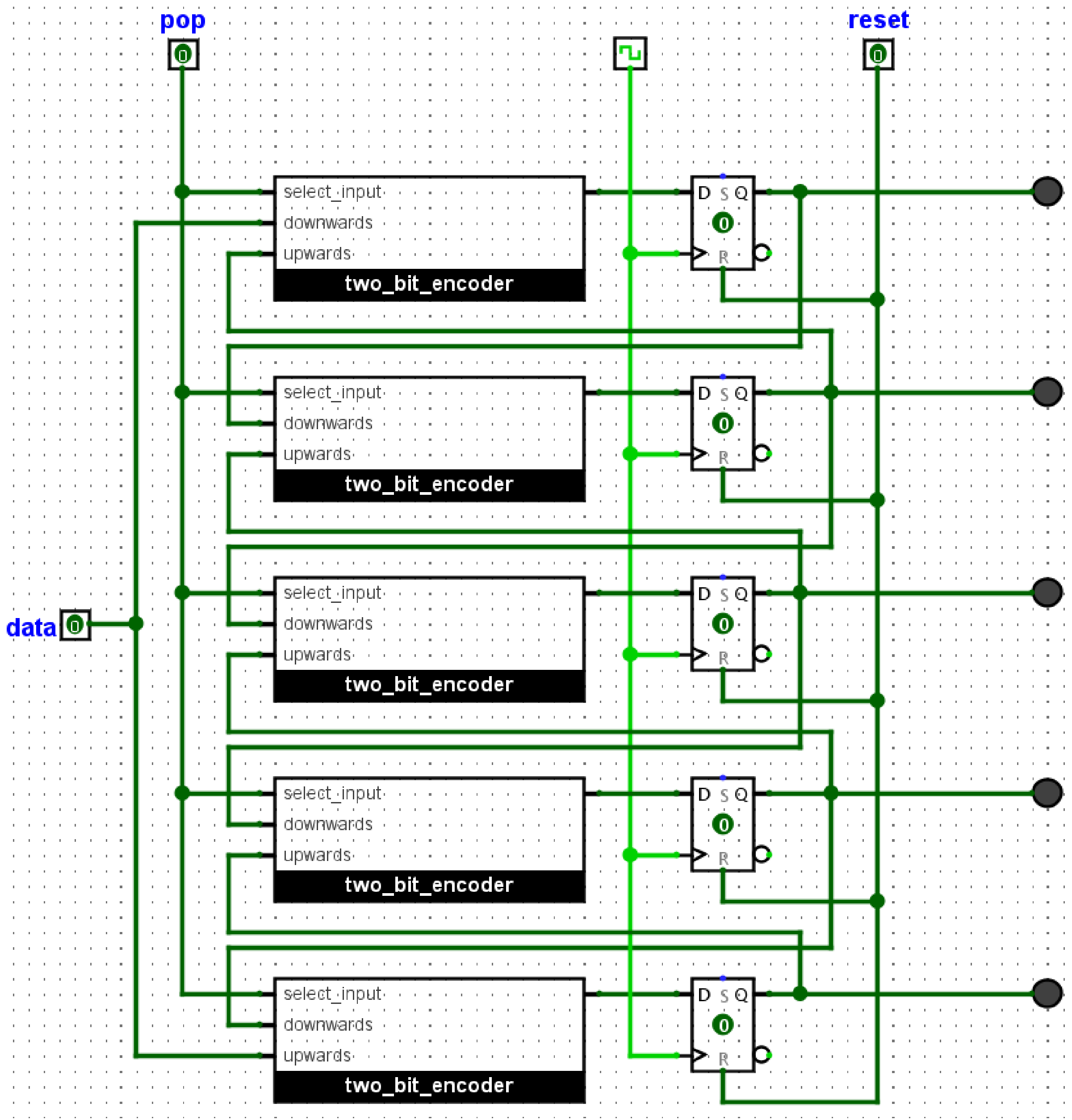
10. For your shift register to work as a stack, it needs to be bi-directional. This means the input to any Flip Flop could come from two places - the left or the right. In lectures we discussed a simple "encoder" circuit that selects which of two data inputs is allowed through, based on a third selection bit. Design the logic for this 2-bit encoder, and demonstrate it to your lab demonstrator.



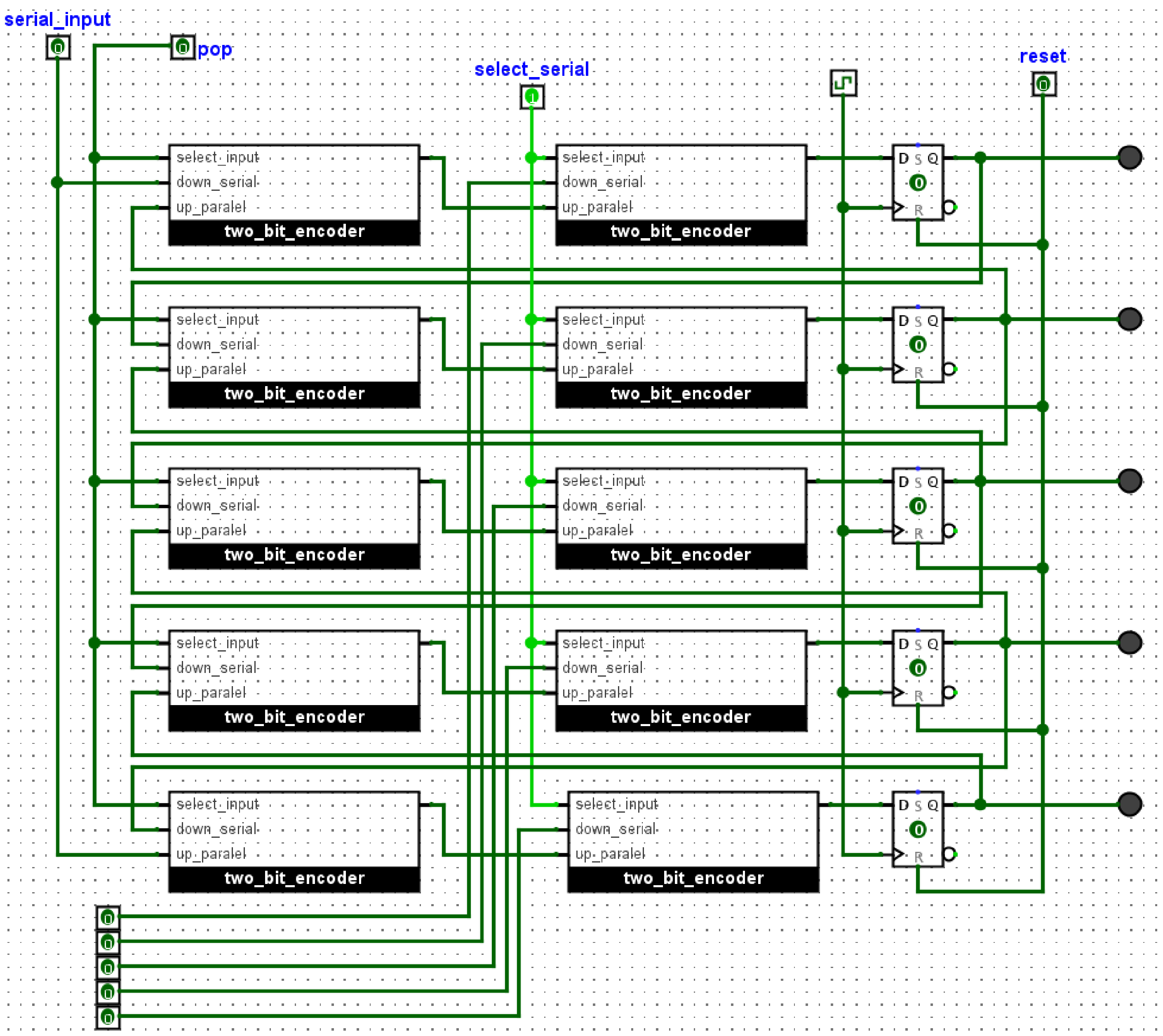
11. Now incorporate your encoder above to allow bi-directional shifting of your stack. Your stack should:

    11.1. push and pop bits onto and off the stack, using clock pulses and a direction toggle switch

    11.2. show the state of each Flip Flop using LEDs.



12. Modify your stack so that it has the option to read out its contents ***in parallel*** to a separate register of D Flip Flops. This should only occur when a “stack dump” toggle switch (i.e., pin) is enabled. When the toggle is disabled, the register of D Flip Flops should retain the last state read in (and should have LEDs connected to each Flip Flop out showing its state).


serial_input
pop
select_serial
reset
select_input
down_serial
up_paralel
two_bit_encoder
D
S
Q
R