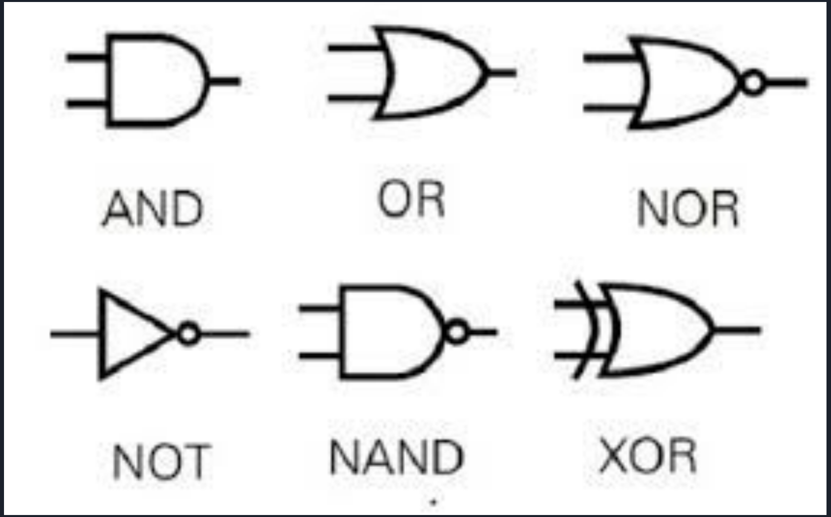
JOIN THE REMIND

Remind code: **489c72f**

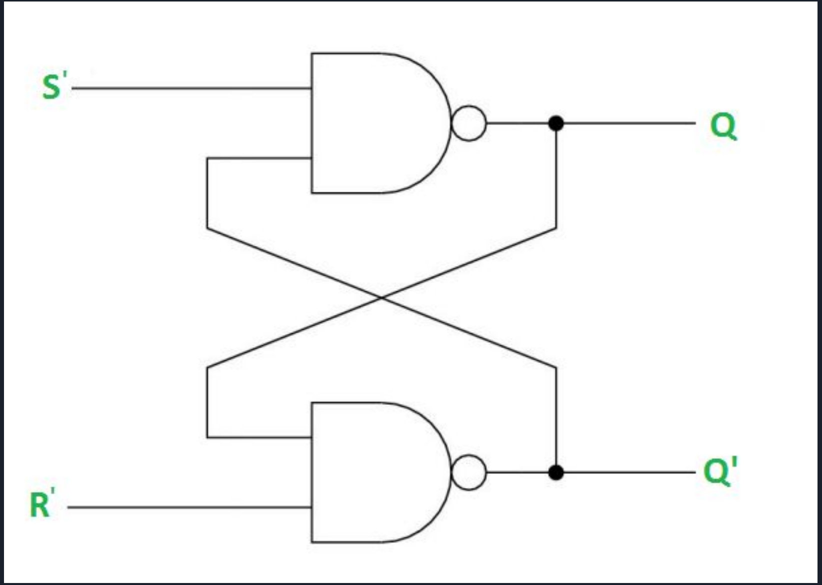
Hardware presentation

EXTRA LEARNING

Logic Gates



Memory



A latch is a basic way to store a single value.

A latch send its output as its input, allowing the latch to store a bit indefinitely until an outside value is added.

One type of latch is a SR flip flop or a SET, RESET latch . The way it works is that set is 1, and reset is 0, and can be changed by preset(PR) or clear(CLR).

There are a lot more types of latches, such as jk,d,and t types.

Sequential logic

A latch is, by its nature a sequential memory unit. All that means is that it feeds it's output into its input. This can be considered as reading the previous input and will sometimes be seen as an n-1 input. The routing of this input requires time. Sending the signal is not instant even if it is extremely quick. This creates a challenge when trying to sync up inputs, something you will have to consider.

Flip flops and registers also use sequential logic but circumvent the lining up problem with their added complexity. These memory units are sequential but not all memory is sequential.

Delay

Delay is something you will always have to think about. When you make a circuit there will always be delay and you will always have to think about it.

Writing a value to a circuit takes a little bit of time. It is not instant! The value itself needs to be stable before it can be accepted. This is called setup time.

The signal also needs to be stable enough to be processed. This is called hold time.

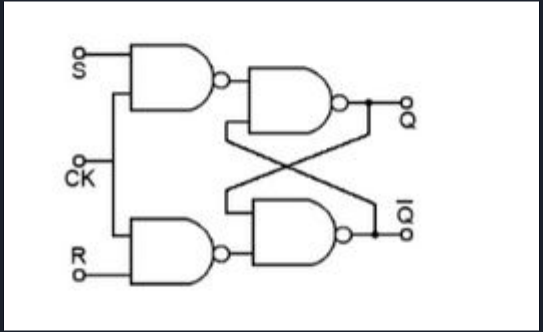
Reading a value requires waiting for the signal to be made so that it can be read effectively and the delay of the internal portions of whatever you are reading.

These are only a couple types of delay. There are more complex interpretations and types of delay!

Flip-Flops

A flip-flop is a very very simple memory circuit. Comprised of 4 NAND gates or 4 NOR gates, it hold either a high 1 or a low 0. Only when there is outside power sent to the system will it switch its value.

Flip flops take on the name of the type of latches they use, so if they use an SR latch, then it's an SR flip flop.



Registers

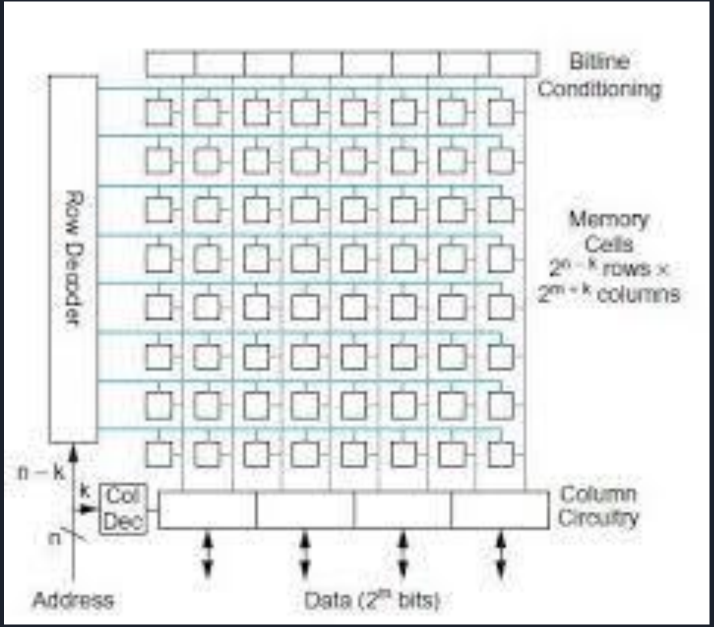
A register is a made of a bunch of latches or flip flops stuck together. The main purpose of a register is to be the fastest form of memory on your computer.

There are a couple of trade offs that a register makes in order to do that. The power of a register is very high for an individual memory unit. Registers come in sizes such as 16, 32, and 64 bits.

Some registers that are stored like this are the program counter, the instruction pointer and stack pointer.

This means that registers are used for fast, temporary storage.

Register files



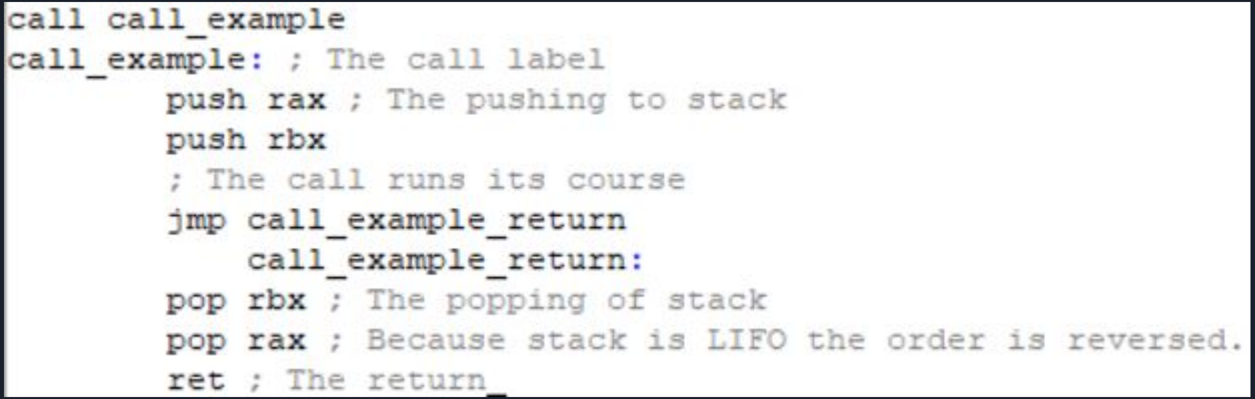
Along with registers there are also register files. Register files are a collection of registers that are right next to each other.

These collections of memory units are known as memory arrays.

Generally you will see general purpose registers as register files. General purpose registers are registers that are used in regular operations and don't have a special purpose.

Assembly (For Context)

Assembly is the lowest level of programming language. The primary way you interact with memory in assembly is through register files.



```
call call_example
call_example: ; The call label
        push rax ; The pushing to stack
        push rbx
        ; The call runs its course
        jmp call_example_return
            call_example_return:
        pop rbx ; The popping of stack
        pop rax ; Because stack is LIFO the order is reversed.
        ret ; The return_
```

Memory arrays

Memory arrays make up the more complicated memory systems and are the other approach to making memory.

<u>Memory arrays take a memory address and then extract the data from that location.</u> They are made of a bitline and a worldline. The size of these lines are dependent on the specific memory array that is being built.

Primary vs Secondary Memory

Primary - RAM (Random Access Memory)/ ROM (Read only memory), VRAM (GPUs)

Secondary - Storage (SSDs, HDDs, SD Card, Floppy Disk, CD-ROM)?

RAM and ROM

RAM is Random access memory
ROM is Read Only Memory

Both are usually on the motherboard

RAM - DRAM (Dynamics RAM) and SRAM (Static RAM)

RAM is Volatile (Will be lost when the computer is turned off)

RAM is volatile because it is built for speed. Capacitors store the byte values in RAM and must be constantly refreshed.

ROM - PROM (Programmable ROM), EPROM (Erasable ROM), EEPROM (Electronically Erasable ROM) and Flash memory.

ROM is for your BIOS/UEFI

RAM/ROM architecture

RAM is situated on the motherboard and connects to the CPU via a bus.

RAM is made of Memory cells, an address decoder, a write driver, and a

Sense amplifier.

ROM is made of an address decoder and memory cells.

https://www.eeeguide.com/rom-memory/

https://www.researchgate.net/publication/356901190_A_review_paper_on_memory_fault_models_and_test_algorithms

Stack Memory

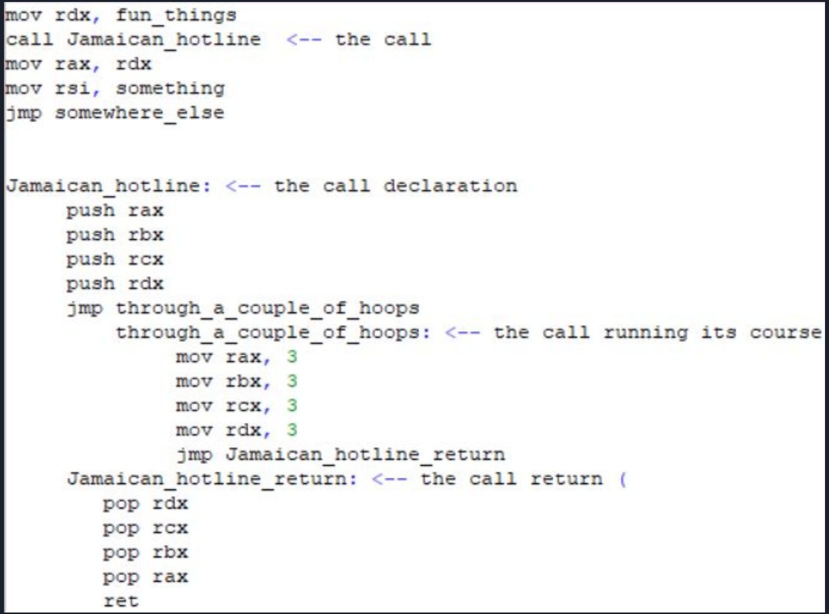
When a process is allowed to run, it takes up a space on the RAM, within that space is an area called the stack.

The stack is LIFO or Last In First Out.

Think of LIFO as a pancake stack. When you put a pancake onto the stack you store that information on the top of the pile via a push. When you take it off of the stack you take the topmost one via a pop.

What that means is that the stack is a small, fast memory storage system that has to be used in such a way that you know when you are going to need the value of each register.

Stack Memory ext.



```
mov rdx, fun_things
call Jamaican_hotline  <-- the call
mov rax, rdx
mov rsi, something
jmp somewhere_else


Jamaican_hotline: <-- the call declaration
     push rax
     push rbx
     push rcx
     push rdx
     jmp through_a_couple_of_hoops
         through_a_couple_of_hoops: <-- the call running its course
              mov rax, 3
              mov rbx, 3
              mov rcx, 3
              mov rdx, 3
              jmp Jamaican_hotline_return
     Jamaican_hotline_return: <-- the call return (
        pop rdx
        pop rcx
        pop rbx
        pop rax
        ret
```

Everything that is getting pushed or pulled is a register.

rdx is used both in the call and outside of it. The value of rdx is copied onto the stack when rdx is pushed onto the stack.

rdx is being modified and changed so the value of rdx is not the same as rdx was before the call executed (on line 2).

Stack Memory ext.

Calls are not the only use case for stack. Besides calls, the stack is used whenever you want to save the value of a something (It can be a register or a variable). Remember that when you store something on the stack, it will be at the very top

That means that you have to set it up to be the only value that is pushed or that the values that you pushed after pushing your initial value will be popped beforehand.

Stack vs Heap

The heap is different from the stack, the stack has a fixed size on the RAM, imagine a little box that you can stack plates in, it's still LIFO, but only to a certain extent.

The heap can dynamically shift from size to size on the RAM depending on what is needed.

When using the heap, data is put on kinda like the stack, where its in a more organized fashion, but the heap is NOT LIFO, meaning any data can be removed and accessed at anytime, making it very UNorganized and when trying to use it, unpredictable. This is called fragmentation.

Flash Memory

NOR vs NAND flash memory.

Pin count is a major factor when deciding whether NOR or NAND is used.

NAND is more efficient than efficient.

Both types of flash can be serial and parallel.

Both of these behave **like** their respective gates. They are not made up of their respective gates.

Cache

XiP is not always possible and reading from storage is not efficient. A common tactic is moving a copy of the information that is needed. That is done with the cache.

The cache is made up of 3 levels. The first level L1, is the closest to the datapath and is the fastest. L2 and L3 are bigger and slower.