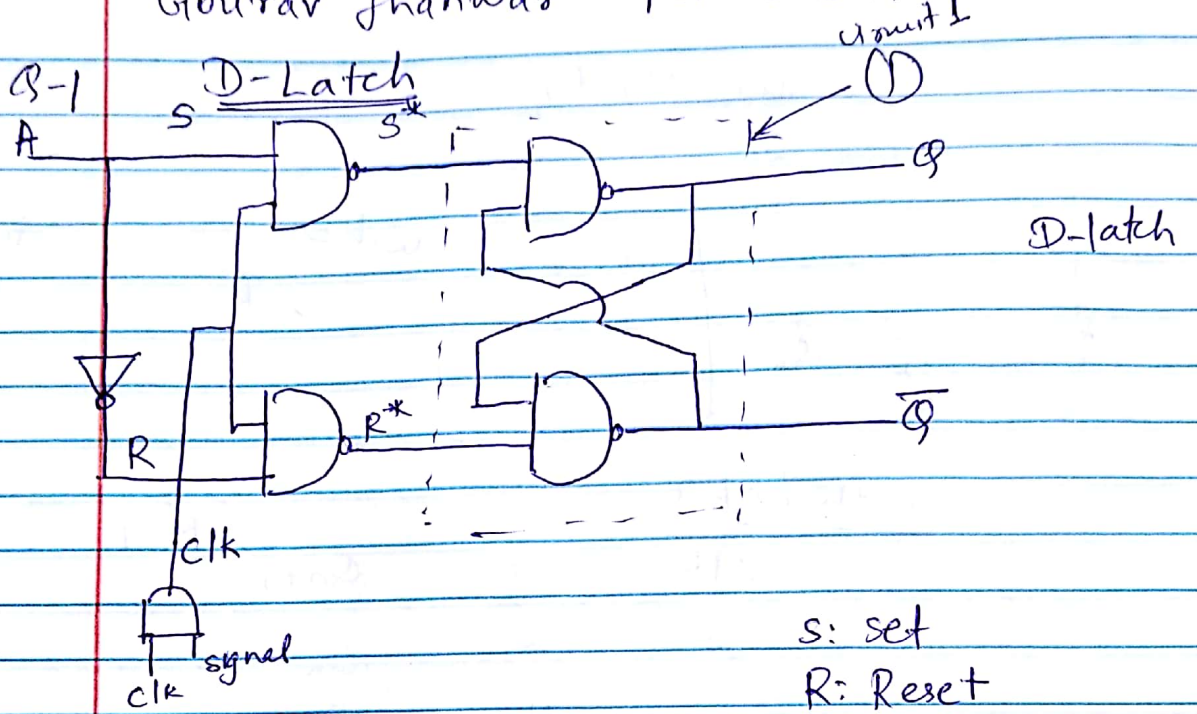
HW-4

IE590 Information Engineering

Gourav Jhanwar PUID: 0030142739

Q-1



S: set

R: Reset

From above diagram,

$$S^* = (S \cdot clk)' = S' + clk'$$

$$\& \; R^* = (R \cdot clk) = R' + clk'$$

Let us evaluate circuit 1 without clock

S^*	R^*	Q	$\bar{Q}$
0	0	Not used	
0	1	1	0
1	0	0	1
1	1	Memory of previous state	

Table 1

Now, let us evaluate S^* & R^* when clock turns 0 & 1 without not gate in the beginning.

when clock = 1

$S^* = S'$

$R^* = R'$

when clock = 0

$S^* = 1$

$R^* = 1$

since in NAND gate, when any input is 0, output is 1 always

let us check Q & $\bar{Q}$ w.r.t S & R with clock

	Clock	S	R	Q	$\bar{Q}$	from table ①
	0	*	*	Memory of previous		
$S^*=1$ $R^*=1$	⇐ 1	0	0	Memory of previous		
		0	1	0	1	
$S^*=1$ $R^*=0$	⇐ 1					
	1	1	0	1	0	
	1	1	1	Not used		

Characteristics table

clock	S	R	Q(1)
0	*	*	Q(0)
1	0	0	Q(0)
1	0	1	0
1	1	0	1
1	1	1	Not used

Now, when NOT gate is put in between S & R & single input is given;

Truth table

clk	A	Q(1)
0	*	Q(0)
1	0	0
1	1	1

(Q-2) Write an assembly language program to multiply two numbers:

soln:-

	Operatn			
D	+	0	0	C
E	if	B	I	F
F	a +	C	A	C
G	-	B	I	B
H	go to	E		
I	END			

∴ Final output is stored in C.

(Q-3) Write an assembly language program to divide two numbers.

→ soln:-

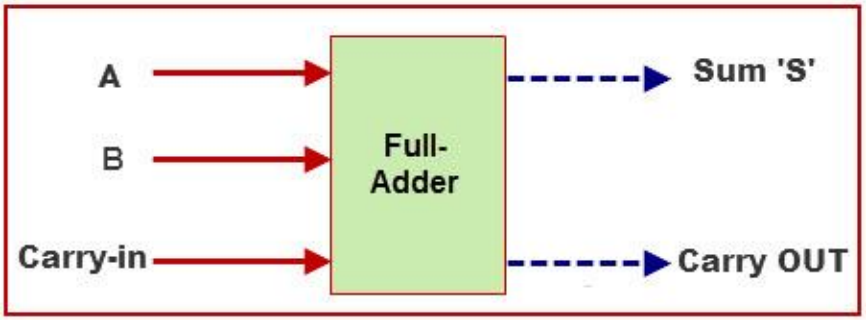
	Operation			
F	+	0	0	C
G	-	A	B	A
H	+	C	I	C
I	comp	A	I	D
J	if	D	G	K
K	comp	A	-I	E
L	If	E	N	M
M	-	C	I	
N	END			

∴ Final quotient is stored in C.

Q4) Write a SUM circuit and show that circuit perform full addition

Full Adder Circuit

This adder is difficult to implement than a half-adder. The difference between a half-adder and a full-adder is that the full-adder has three inputs and two outputs, whereas half adder has only two inputs and two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. When a full-adder logic is designed, you string eight of them together to create a byte-wide adder and cascade the carry bit from one adder to the next.



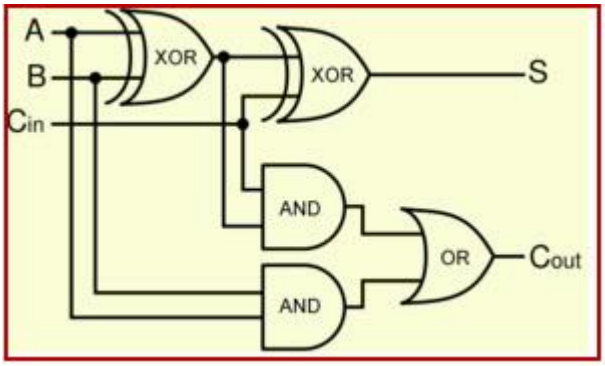
Full Adder

The output carry is designated as C-OUT and the normal output is designated as S.

Full Adder Truth Table:

INPUTS			OUTPUT	
A	B	C-IN	C-OUT	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

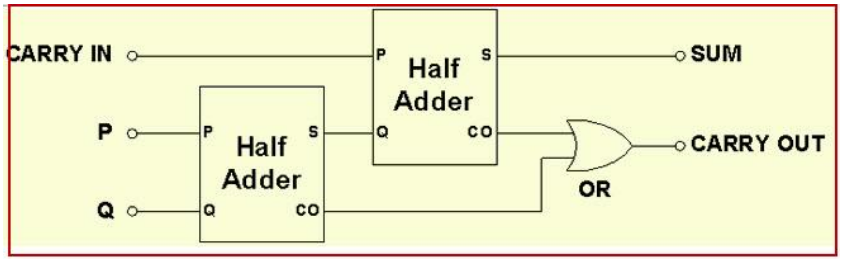
With the truth-table, the full adder logic can be implemented. You can see that the output S is an XOR between the input A and the half-adder, SUM output with B and C-IN inputs. We take C-OUT will only be true if any of the two inputs out of the three are HIGH.
So, we can implement a full adder circuit with the help of two half adder circuits. At first, half adder will be used to add A and B to produce a partial Sum and a second half adder logic can be used to add C-IN to the Sum produced by the first half adder to get the final S output.



Full Adder Logic Circuit

If any of the half adder logic produces a carry, there will be an output carry. So, COUT will be an OR function of the half-adder Carry outputs. Take a look at the implementation of the full adder circuit shown below.

The implementation of larger logic diagrams is possible with the above full adder logic a simpler symbol is mostly used to represent the operation. Given below is a simpler schematic representation of a one-bit full adder.



Full Adder Design Using Half Adders

With this type of symbol, we can add two bits together, taking a carry from the next lower order of magnitude, and sending a carry to the next higher order of magnitude. In a computer, for a multi-bit operation, each bit must be represented by a full adder and must be added simultaneously. Thus, to add two 8-bit numbers, you will need 8 full adders which can be formed by cascading two of the 4-bit blocks.

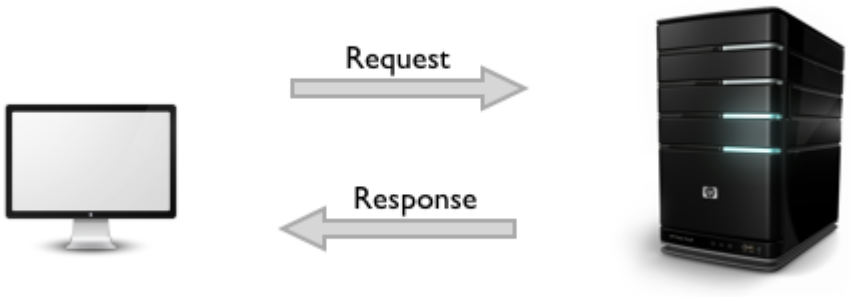
The relationship between the Full-Adder and the Half-Adder is half adder produces results and full adder uses half adder to produce some other result. Similarly, while the Full-Adder is of two Half-Adders, the Full-Adder is the actual block that we use to create the arithmetic circuits.

Q5) Write a summary of http including complete set of verbs and complete set of status code and their meaning

HTTP BASICS

HTTP allows for communication between a variety of hosts and clients, and supports a mixture of network configurations.
To make this possible, it assumes very little about a system, and does not keep state between different message exchanges.

This makes HTTP a **stateless** protocol. The communication usually takes place over TCP/IP, but any reliable transport can be used. The default port for TCP/IP is **80**, but other ports can also be used.



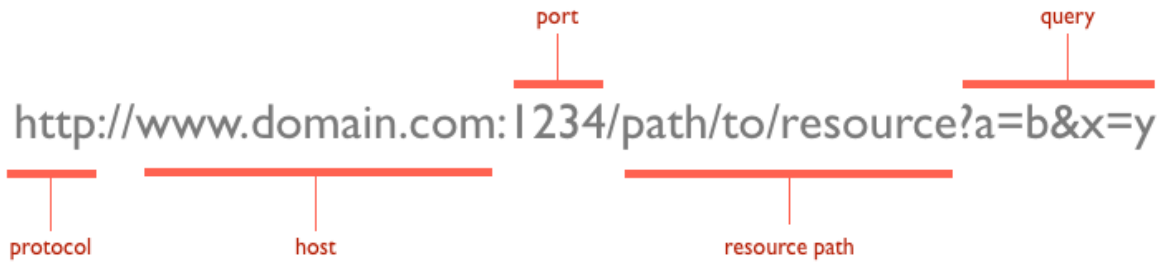
Custom headers can also be created and sent by the client.

Communication between a host and a client occurs, via a **request/response pair**. The client initiates an HTTP request message, which is serviced through a HTTP response message in return. We will look at this fundamental message-pair in the next section.

The current version of the protocol is **HTTP/1.1**, which adds a few extra features to the previous 1.0 version. The most important of these, in my opinion, includes *persistent connections*, *chunked transfer-coding* and *fine-grained caching headers*.

URLs

At the heart of web communications is the request message, which are sent via Uniform Resource Locators (URLs). I'm sure you are already familiar with URLs, but for completeness sake, I'll include it here. URLs have a simple structure that consists of the following components:



The protocol is typically http, but it can also be https for secure communications. The default port is 80, but one can be set explicitly, as illustrated in the above image. The resource path is the *local path* to the resource on the server.

Verbs

URLs reveal the identity of the particular host with which we want to communicate, but the action that should be performed on the host is specified via HTTP verbs. Of course, there are several actions that a client would like the host to perform. HTTP has formalized on a few that capture the essentials that are universally applicable for all kinds of applications.

These request verbs are:

- **GET**: *fetch* an existing resource. The URL contains all the necessary information the server needs to locate and return the resource.
- **POST**: *create* a new resource. POST requests usually carry a payload that specifies the data for the new resource.
- **PUT**: *update* an existing resource. The payload may contain the updated data for the resource.
- **DELETE**: *delete* an existing resource.

The above four verbs are the most popular, and most tools and frameworks explicitly expose these request verbs. PUT and DELETE are sometimes considered specialized versions of the POST verb, and they may be packaged as POST requests with the payload containing the exact action: *create*, *update* or *delete*.

There are some lesser used verbs that HTTP also supports:

- **HEAD**: this is similar to GET, but without the message body. It's used to retrieve the server headers for a particular resource, generally to check if the resource has changed, via timestamps.
- **TRACE**: used to retrieve the hops that a request takes to round trip from the server. Each intermediate proxy or gateway would inject its IP or DNS name into the via header field. This can be used for diagnostic purposes.
- **OPTIONS**: used to retrieve the server capabilities. On the client-side, it can be used to modify the request based on what the server can support.

Status Codes

With URLs and verbs, the client can initiate requests to the server. In return, the server responds with status codes and message payloads. The status code is important and tells the client how to interpret the server response. The HTTP spec defines certain number ranges for specific types of responses:

1xx: Informational Messages

All HTTP/1.1 clients are required to accept the Transfer-Encoding header.

This class of codes was introduced in HTTP/1.1 and is purely provisional. The server can send a message, telling the client to continue sending the remainder of the request, or ignore if it has already sent it. HTTP/1.0 clients are supposed to ignore this header.

2xx: Successful:

This tells the client that the request was successfully processed. The most common code is **200 OK**. For a GET request, the server sends the resource in the message body. There are other less frequently used codes:

- **202** Accepted: the request was accepted but may not include the resource in the response. This is useful for async processing on the server side. The server may choose to send information for monitoring.
- **204** No Content: there is no message body in the response.
- **205** Reset Content: indicates to the client to reset its document view.
- **206** Partial Content: indicates that the response only contains partial content. Additional headers indicate the exact range and content expiration information.

3xx: Redirection:

This requires the client to take additional action. The most common use-case is to jump to a different URL in order to fetch the resource.

- **301** Moved Permanently: the resource is now located at a new URL.
- **303** See Other: the resource is temporarily located at a new URL. The location response header contains the temporary URL.
- **304** Not Modified: the server has determined that the resource has not changed and the client should use its cached copy. This relies on the fact that the client is sending (Entity Tag) information that is a hash of the content. The server compares this with its own computed ETag to check for modifications.

4xx: Client Error:

These codes are used when the server thinks that the client is at fault, either by requesting an invalid resource or making a bad request. The most popular code in this class is **404 Not Found**, which I think everyone will identify with. 404 indicates that the resource is invalid and does not exist on the server.

The other codes in this class include:

- **400** Bad Request: the request was malformed.
- **401** Unauthorized: request requires authentication. The client can repeat the request with the Authorization header. If the client already included the Authorization header, then the credentials were wrong.
- **403** Forbidden: server has denied access to the resource.
- **405** Method Not Allowed: invalid HTTP verb used in the request line, or the server does not support that verb.
- **409** Conflict: the server could not complete the request because the client is trying to modify a resource that is newer than the client's timestamp. Conflicts arise mostly for PUT requests during collaborative edits on a resource.

5xx: Server Error:

This class of codes are used to indicate a server failure while processing the request. The most commonly used error code is **500 Internal Server Error.** The others in this class are:

- **501 Not Implemented**: the server does not yet support the requested functionality.
- **503 Service Unavailable**: this could happen if an internal system on the server has failed or the server is overloaded. Typically, the server won't even respond and the request will timeout.