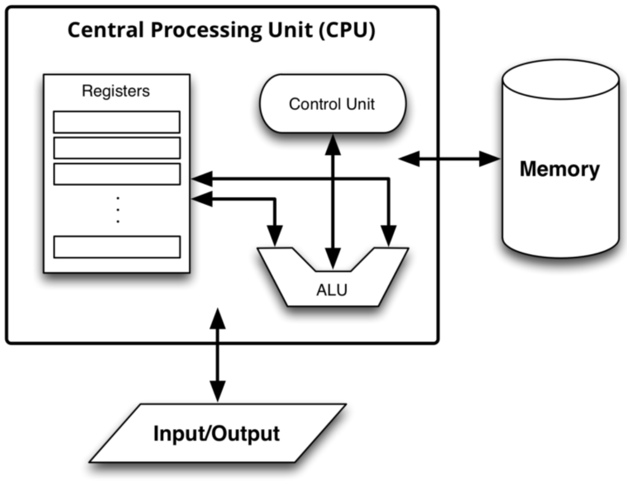
FIT1047 PASS WEEK 5

CPU, Sequential Circuit, Combinational Circuit
Memory and I/O
MARIE (Subroutine and Indirect Addressing)

Von Neumann architecture



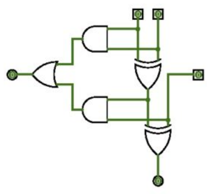
- Registers store temp results & moves instructions and data around.
- ALU performs actual calculations (+, x, logical operations)
- CU coordinates components, e.g. switch "read" to "write", instruct ALU.

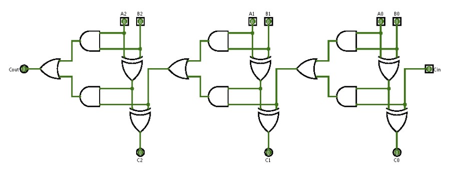
Fetch-decode-execute
The Fetch-Execute Cycle: What's Your Computer Actually Doing?

What's a distinctive feature of Von Neumann architecture?
Program instructions and data are stored in the same memory.

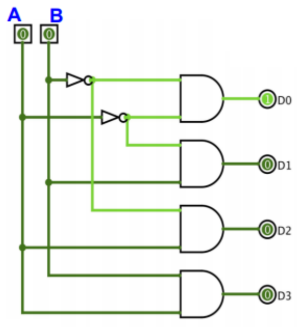
Combinational circuit

- Output depends on inputs - circuit computes a simple function of inputs.

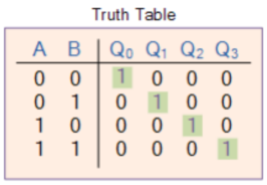
Half adders	• Adds 2 one-bit inputs, A and B. • It has 2 outputs, Result and Carry-out.
Full adders 	• Adds 3 one-bit inputs, often written as A, B, and C_{in} • A and B are the operands, and C_{in} is a bit carried in from the previous less-significant stage. • 2 outputs, Result and Carry-out.
Ripple-carry adders	• Several full adders combined to perform longer bits addition • Example: add two 3-bit numbers by constructing a chain of 3 full adders.



Decoders

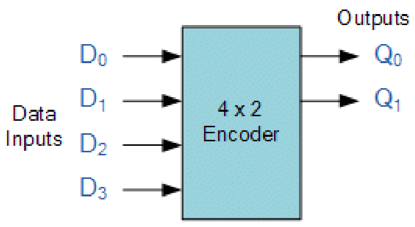


- Activate 1 output (*unary* representation) based on a binary number.
- **n** inputs produces **2^n** outputs.

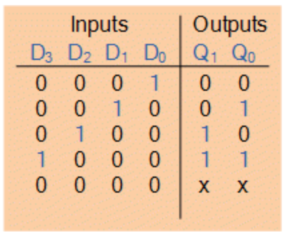

Truth Table

A	B	Q_0	Q_1	Q_2	Q_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Encoders

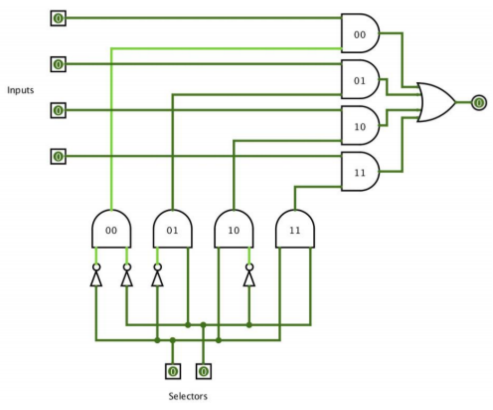


- **One-hot** to binary converter.
- only one bit is “hot” or TRUE (1) at any time.

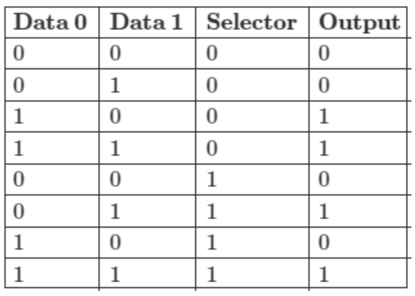


Inputs				Outputs	
D_3	D_2	D_1	D_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
0	0	0	0	x	x

Multiplexers



- Select **one** of several inputs based on the selector.
- **n** selection inputs, determines which one of the **2^n** data inputs to pick.

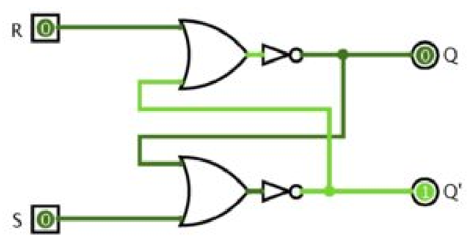


Data 0	Data 1	Selector	Output
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

Sequential circuits

- Output depends on *sequence* of inputs
- ‘remember past’ by **passing output back into input** - feedback loop

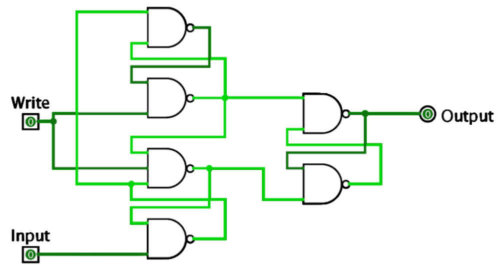
Set/Reset Latch



- Store single bit of data
- Set/Reset stage: Q always follows S.
- Q and Q’ should always be opposite, hence invalid (forbidden) state when they are the same.

Stage	S	R	Q	Q’
Remains previous value	0	0	1	0
			0	1
Reset	0	1	0	1
Set	1	0	1	0
Invalid	1	1	0	0

D flip-flop



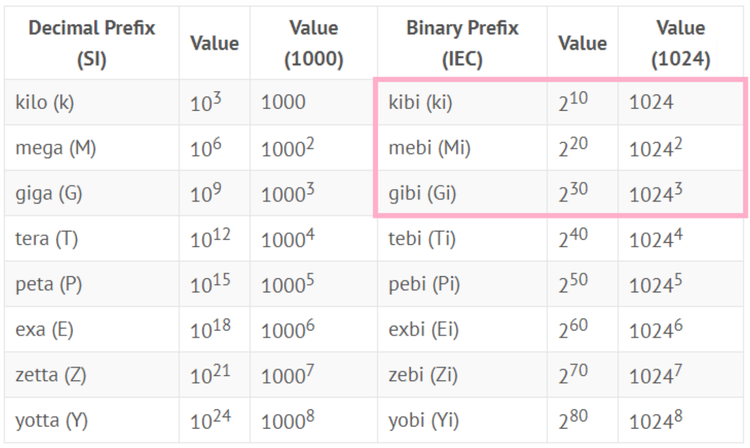
- 2 inputs: stored bit & signal (read/write)
- 1 output: bit currently stored

Write signal	Output
0	Unaffected by input
1	Same as input

Briefly explain the concept of a flipflop circuit. Name a computer component where a flipflop is used. (3%)
A flipflop is a sequential circuit that can store one bit of information, and the stored information can be read and changed at a later point in time. It can be used to implement registers.

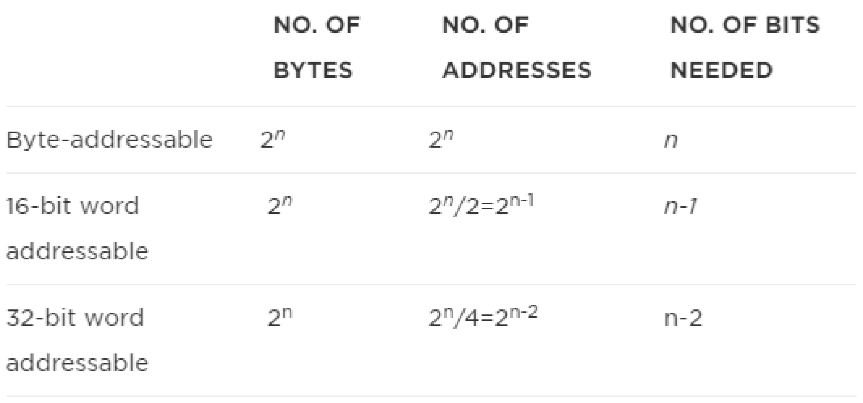
Briefly explain the difference between sequential and combinational circuits. (3%)
The output of a sequential circuit depends on previous inputs (e.g. a flipflop). The output of a combinational circuit only depends on its current inputs, i.e., it simply computes a Boolean function of the inputs.

Memory



Decimal Prefix (SI)	Value	Value (1000)	Binary Prefix (IEC)	Value	Value (1024)
kilo (k)	10^3	1000	kibi (ki)	2^{10}	1024
mega (M)	10^6	1000^2	mebi (Mi)	2^{20}	1024^2
giga (G)	10^9	1000^3	gibi (Gi)	2^{30}	1024^3
tera (T)	10^{12}	1000^4	tebi (Ti)	2^{40}	1024^4
peta (P)	10^{15}	1000^5	pebi (Pi)	2^{50}	1024^5
exa (E)	10^{18}	1000^6	exbi (Ei)	2^{60}	1024^6
zetta (Z)	10^{21}	1000^7	zebi (Zi)	2^{70}	1024^7
yotta (Y)	10^{24}	1000^8	yobi (Yi)	2^{80}	1024^8

- Byte-addressable - 1 memory location stores 1 byte.
- Word-addressable - 1 memory location stores 1 word (word depends on CPU)
- $\lceil \log_2 n \rceil$ bits needed to address **n** different addresses.



	NO. OF BYTES	NO. OF ADDRESSES	NO. OF BITS NEEDED
Byte-addressable	2^n	2^n	n
16-bit word addressable	2^n	$2^n/2=2^{n-1}$	$n-1$
32-bit word addressable	2^n	$2^n/4=2^{n-2}$	n-2

How many bits are in 32 Gibit? (Give answer in power of 2)
1 gibit = 2^{30} bits
32 x 2^{30} = 2^5 x 2^{30} = **2^{35} bits**

How many bits are needed to to address 2 gibibytes in a word-addressable architecture, where 1 word is 4 bytes?
2 gibibytes = 2 x 2^{30} bytes = 2^{31} bytes
1 address holds 4 bytes (2^2 bytes), therefore we need 2^{31} / 2^2 = 2^{29} different addresses, ranging from 0 ... 2^{29} - 1.
Total bits needed = **$\lceil \log_2 2^{29} \rceil$ = 29 bits.**

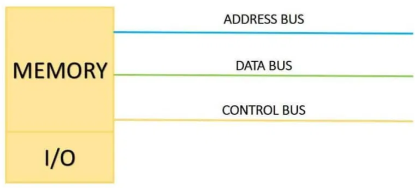
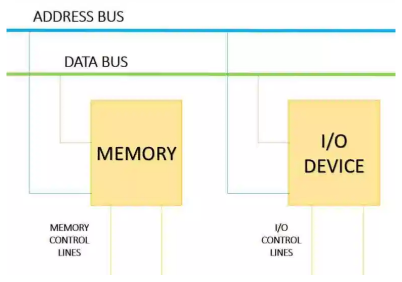
RAM

- RAM modules made up of multiple chips
- Each chip has fixed size (no. of locations x no. of bits per location)

Compute the total number of bits in the following chips based on their size.

2K x 8	2K x 8 = 2 x 2^{10} locations **x** 8 bits per location = 2 x 2^{10} **x** 2^3 bits in total = 2^{1+10+3} bits in total = **2^{14} bits per chip**
2M x 16	2M x 16 = 2 x 2^{20} locations **x** 16 bits per location = 2 x 2^{20} **x** 2^4 bits in total = 2^{1+20+4} bits in total = **2^{25} bits per chip**

I/O Access Methods - How to communicate with I/O?

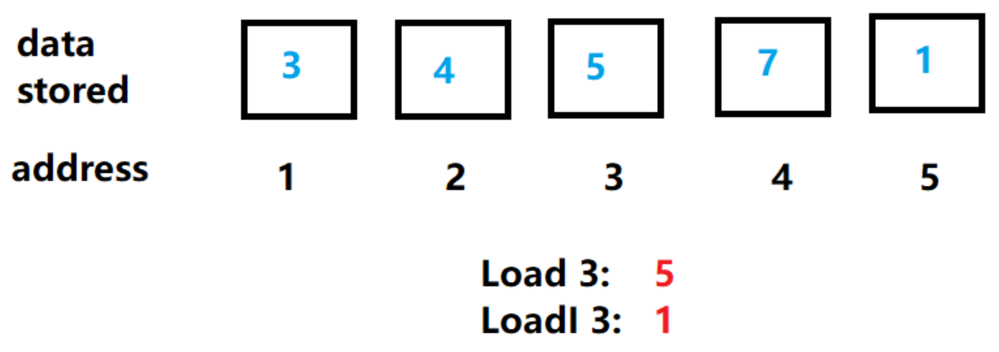
Memory-mapped	**Instruction-based**
I/O registers "mapped" into "address space" of CPU. 	CPU has special instructions to read from/write to particular I/O devices.  ADDRESS BUS DATA BUS MEMORY I/O DEVICE MEMORY CONTROL LINES I/O CONTROL LINES
• New instructions not needed • Simple - device deals with fewer instructions	• Use reduced address width for I/O addresses than memory addresses • Hardware circuits are simpler because of limited I/O ports.
• Cannot use "mapped" addresses for memory anymore • Overall amount of usable memory reduces (unavailable addresses) • Programs may accidentally access I/O	• Compared to memory-mapped I/O, more instructions are required to complete the same task.

I/O Control Methods - When to do I/O?

Programmed I/O (software responsible)	• **Checks for I/O new data periodically** • Simple (no extra hardware) • Full control polling - prioritise certain I/Os • CPU constantly busy (to check for I/O)
Interrupt-based I/O (hardware responsible)	• **Hardware notifies CPU when new I/O data available** • CPU interrupts the program and jumps to a special subroutine to process I/O request, then continues as normal. • Programmers don't need to be aware of I/O.
Direct Memory Access (DMA) I/O	• **CPU delegates memory transfer operations to a dedicated *controller*** • Example: hard disk controller copies file directly from disk to RAM; graphic cards fetch image directly from RAM → CPU free to do other work! • CPU and DMA share the same data bus - only 1 performs memory transfer at a time.

MARIE - Indirect Addressing

LoadI X	Loads value stored at address of address X into AC
JnS X	Stores PC at address X and jumps to X+1
JumpI X	Uses value at X as the address to jump to



MARIE - Subroutine

Simple example: Print value X using subroutine PrintX.

```
1   JnS PrintX // subroutine
2   Halt
3
4   PrintX,     Hex 000 // store return address
5               Load X
6               Output
7
8               // Exit subroutine PrintX
9               JumpI PrintX
10
11  X, Dec 19
```

1. **Stores PC at address PrintX** - `JnS` stores address of next instruction (line 2) in return address at line 4 (to be used later to restore program execution)
2. **Jumps to PrintX+1** - It jumps to instruction immediately below PrintX label (line 5)
3. Performs subroutine (line 5 to 8) like a normal function call code.
4. **Uses value at PrintX as the address to jump to** - loads address stored at PrintX label, and stores it to PC register. This is the return address earlier (line 2).

Exercise: **Write a MARIE program that performs exponential calculations with base and power inputs.**

MARIE file (.mas) has been attached in email. Here's the Python code for reference:

```
# subroutines
def exp(base, power):
   res = 1
   while power > 0:
      res = mult(res,base)
      power = power - 1
   return res
```

```
def mult(a,b):
   multres = 0
   while b > 0:
      multres = multres + a
      b = b - 1
   return multres

# main program
if __name__ == "__main__":
   base = int(input())
   power = int(input())
   res = exp(base, power)
   print(res)
```