

Simple-As-Possible Computers

Day 1

IEEE NIE Student Branch - Circuits & Systems Society

You have *only used* a computer all this time, so why not try and *build one* for a change?

You can do it in *3 days flat!*

Here is how we do it

1. Basic logic design, arithmetic and computer architecture
 2. SAP-1 architecture (1 block at a time)
 3. Put it all together
-

The SAP-1

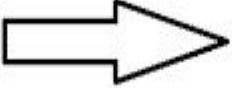

8-bits everything
Addition and subtraction
von Neumann



Recap of Number Systems

Binary Number System

```
0100011101110  
1010111001001  
1101010010000  
0010000100111
```

0  OFF
1  ON

Hexadecimal Number System

Hexadecimal	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

2's Complement

0 1 1 0 1 1 1 0 ← Original binary value

1 0 0 1 0 0 0 1 ← 1's complement

1 0 0 1 0 0 0 1
+ 1
1 0 0 1 0 0 1 0

← 2's complement

Binary representation of 5 is: 0 1 0 1

1's Complement of 5 is: 1 0 1 0

2's Complement of 5 is: (1's Complement + 1) i.e.

1 0 1 0 (1's Complement)

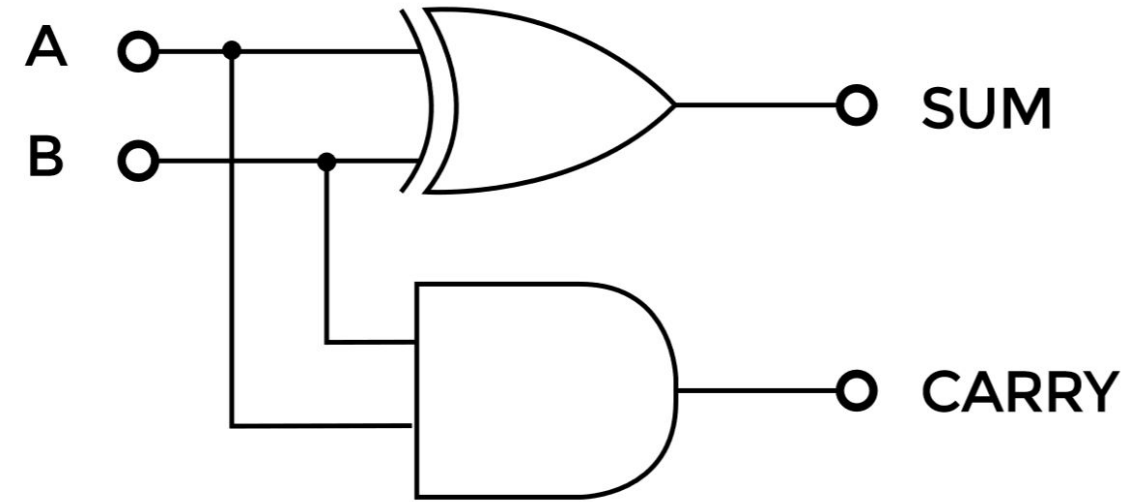
+ 1

1 0 1 1 (2's Complement i.e. -5)

Recap of Combinational Logic

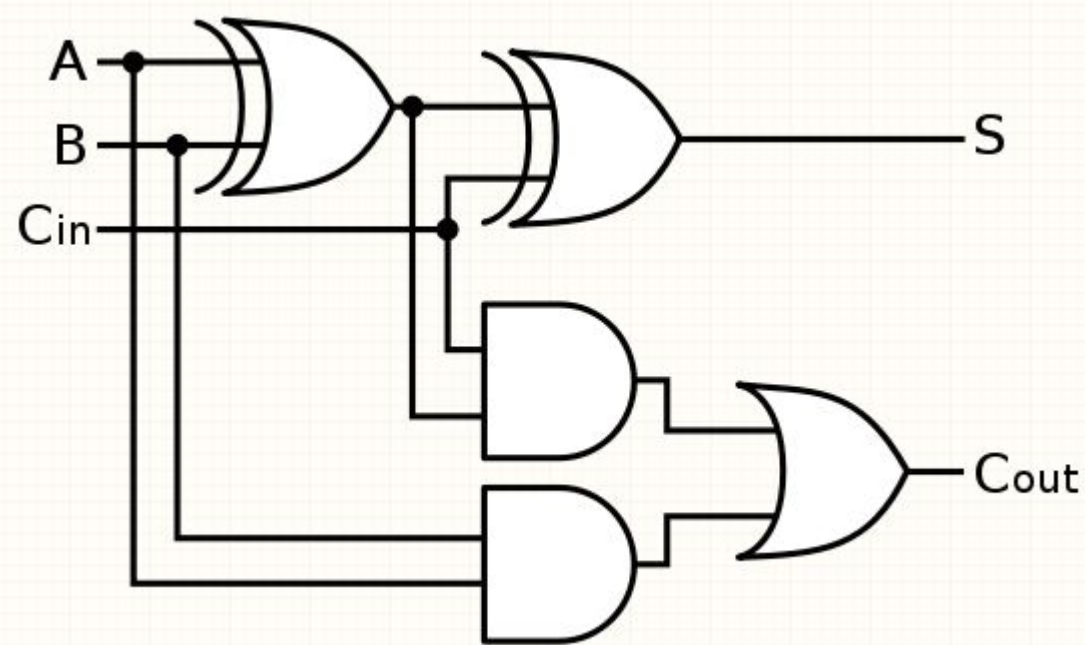
—

Half Adder

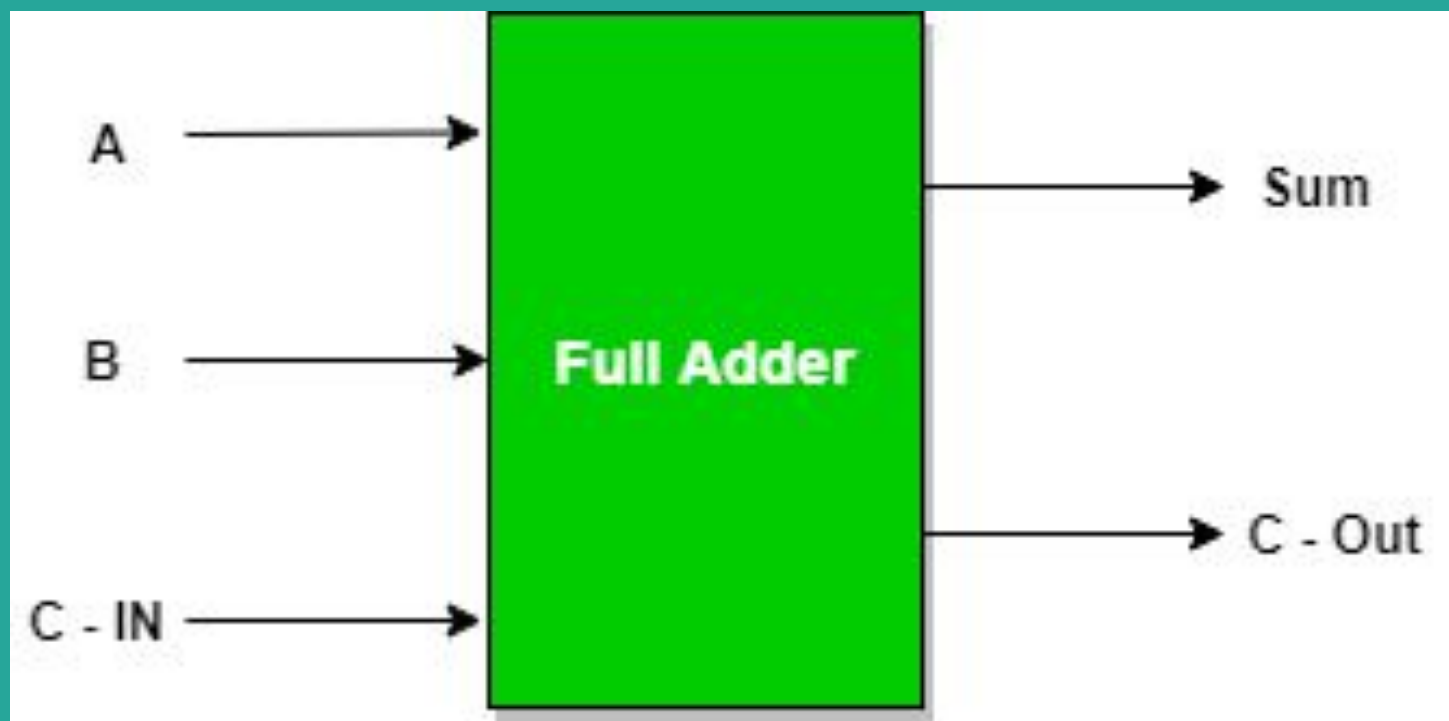


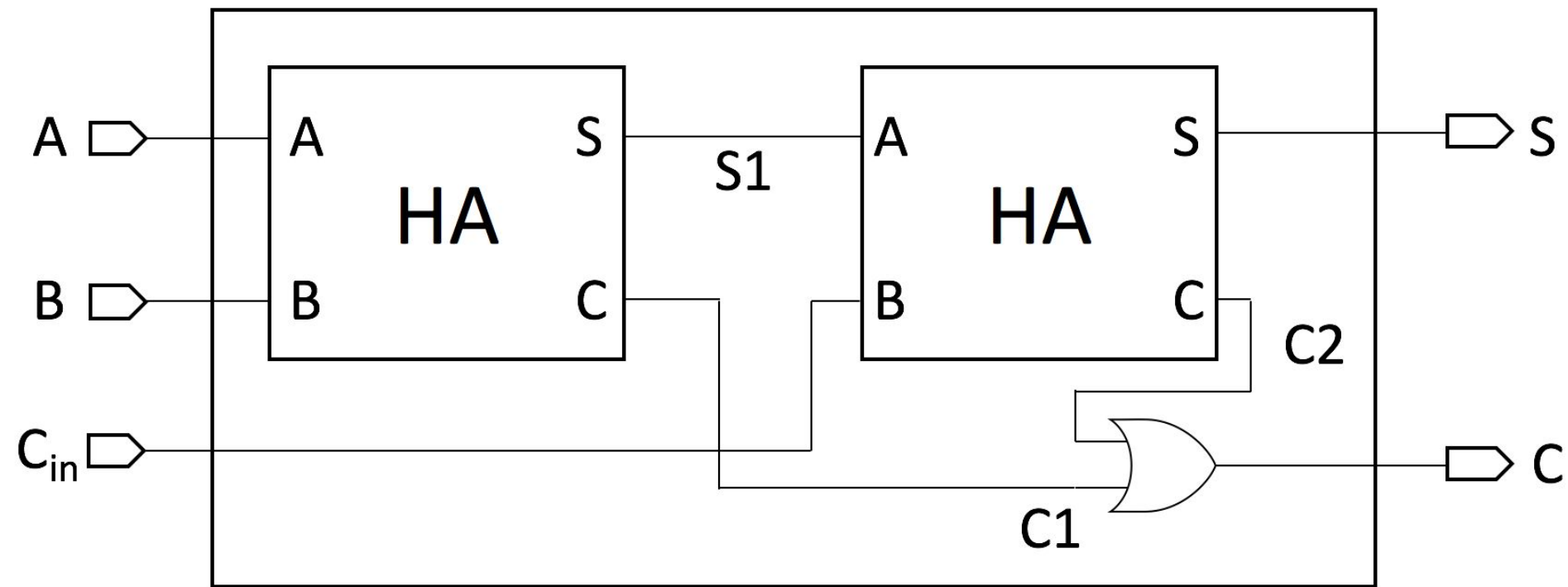
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Full Adder

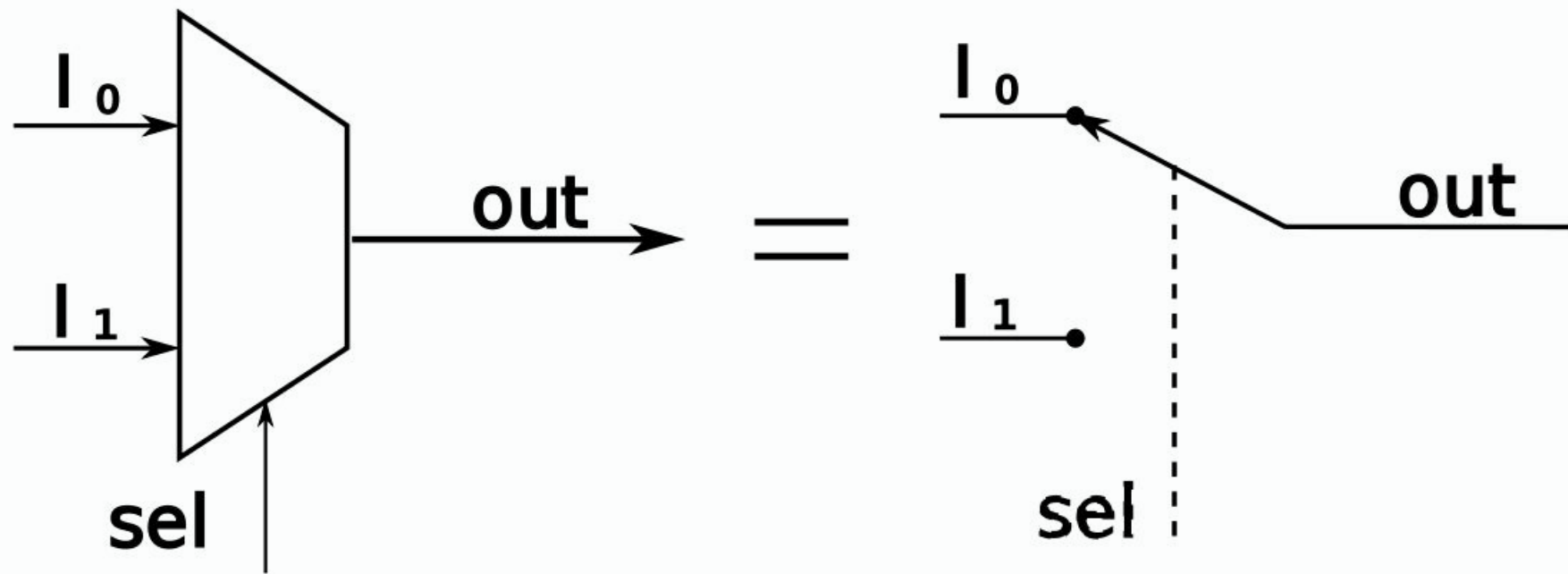


A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1





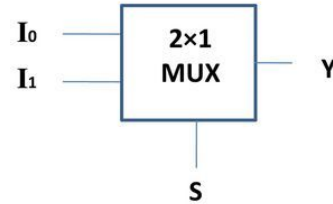
Multiplexers & Demultiplexers



2-to-1 Multiplexer

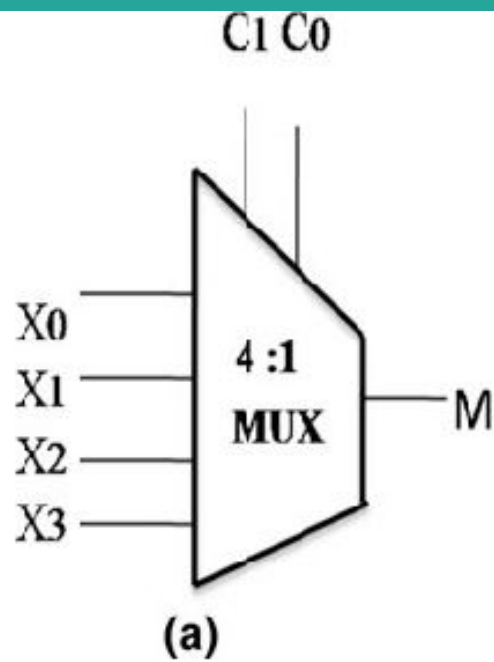
S	I ₀	I ₁	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Truth table for 2x1 MUX



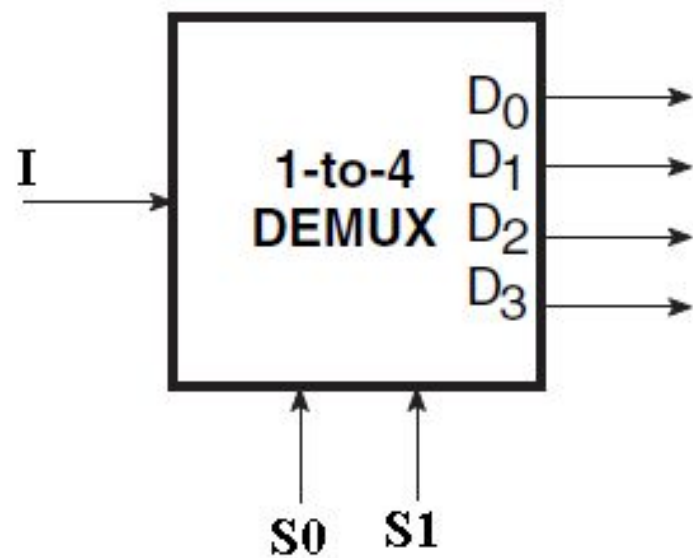
S	Y
0	I ₀
1	I ₁

Condensed Truth Table



C_1	C_0	M
0	0	X_0
0	1	X_1
1	0	X_2
1	1	X_3

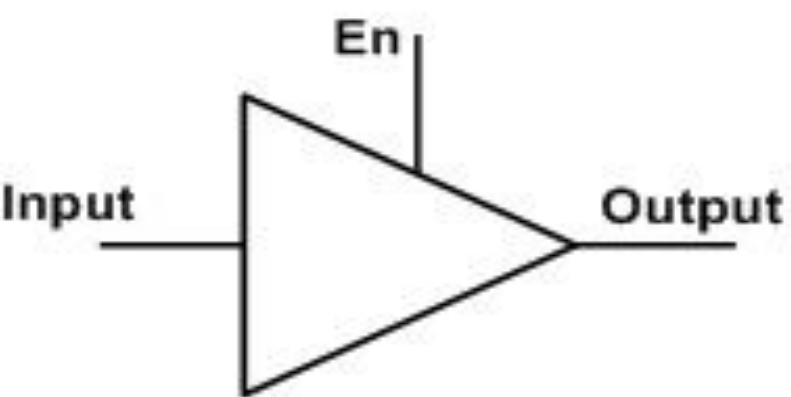
(b)



I	Select		O/P			
	S0	S1	D0	D1	D2	D3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Tri-State Logic

Symbol



Truth Table

En	Input	Output
0	X	Hi-Z
1	0	0
1	1	1

D Flip-Flop

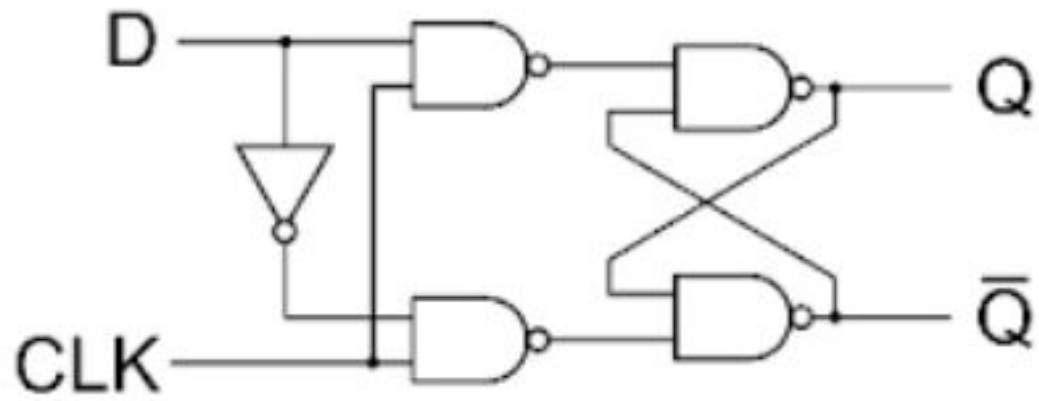
—

Inputs		Outputs		Comments
D	CLK	Q	\overline{Q}	
1	↑	1	0	SET
0	↑	0	1	RESET

(a) Positive-edge triggered

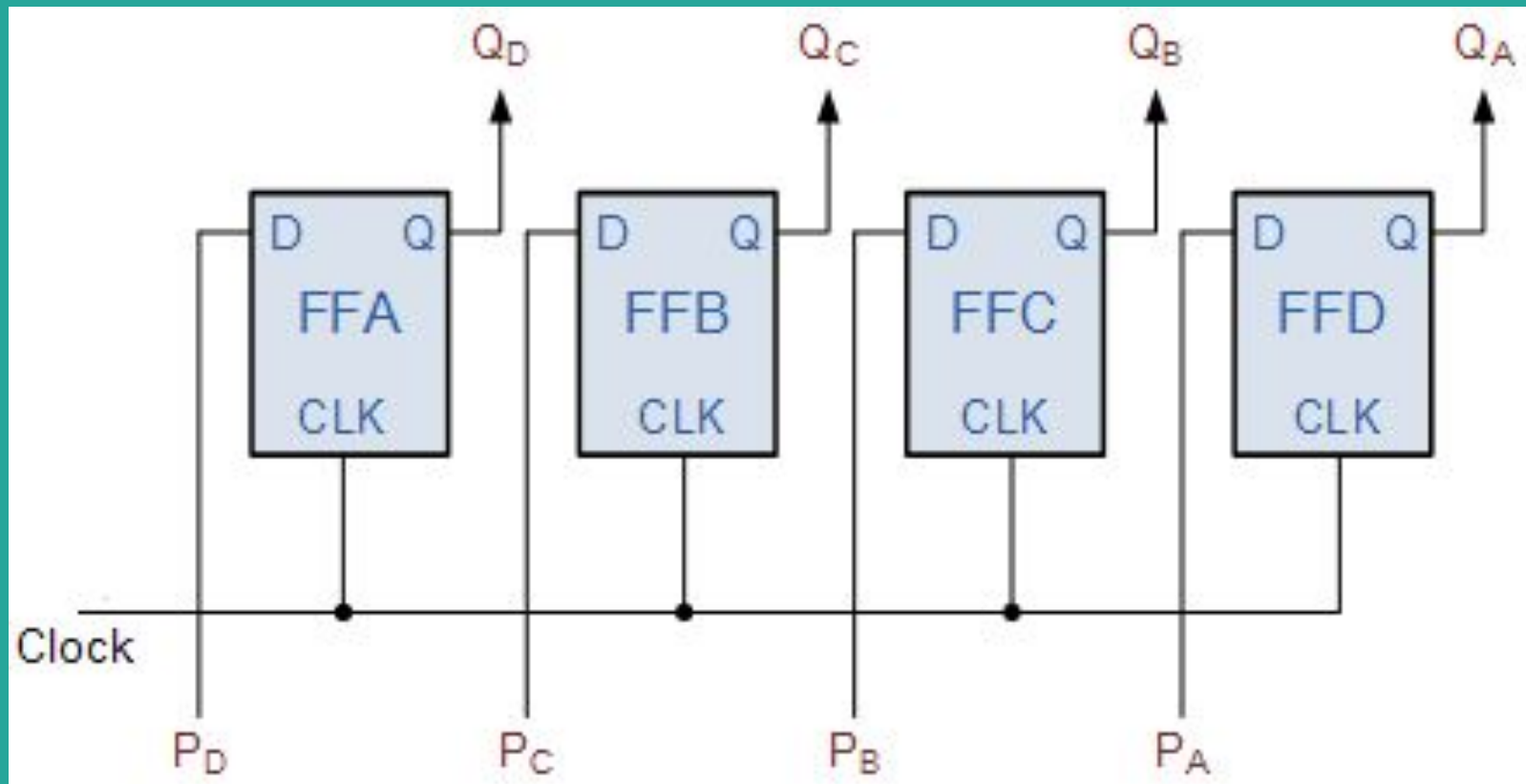
Inputs		Outputs		Comments
D	CLK	Q	\overline{Q}	
1	↓	1	0	SET
0	↓	0	1	RESET

(b) Negative-edge triggered

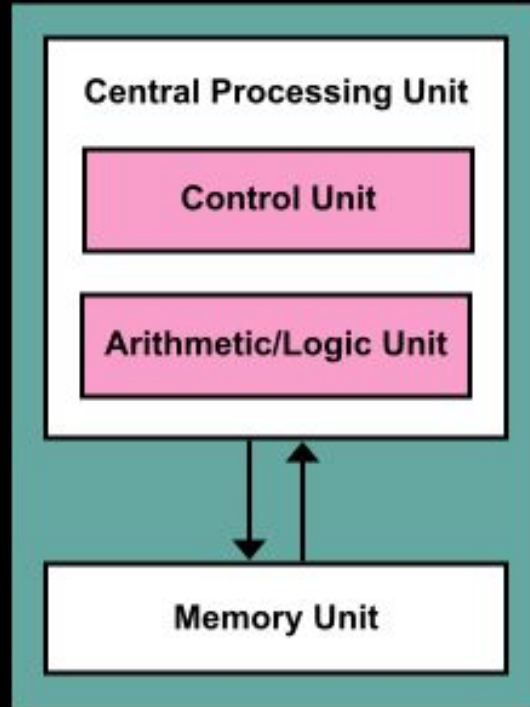


D-Registers

—



von Neumann Architecture



What is a BUS?

—



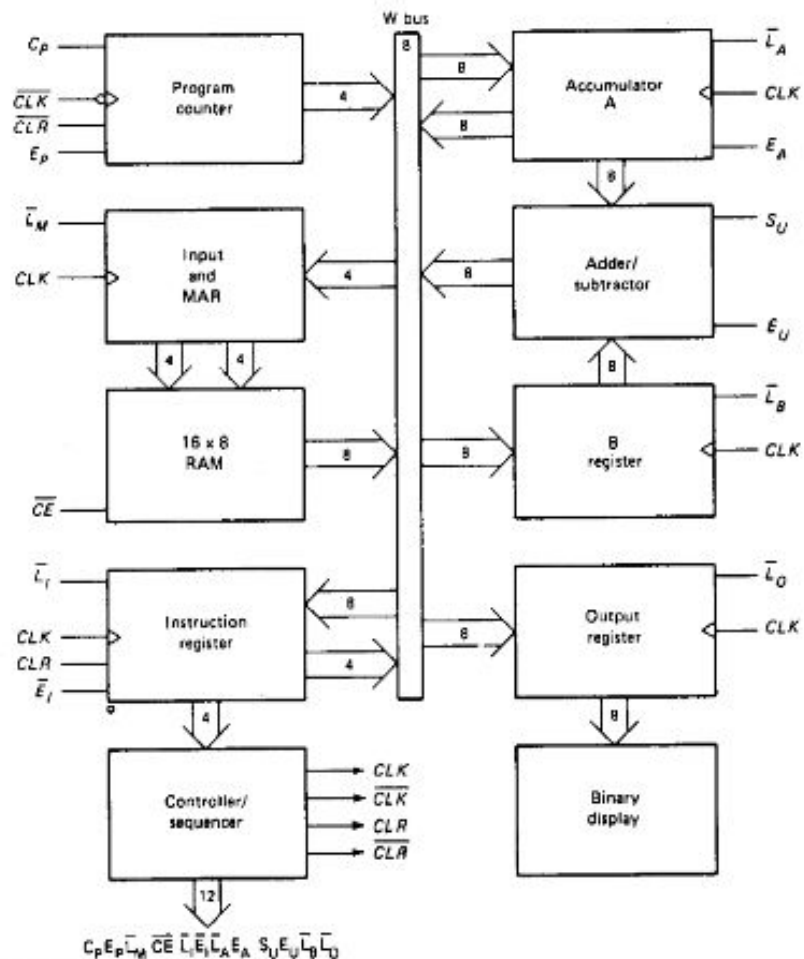
What is a BUS?

Communication system that transmit binary data.



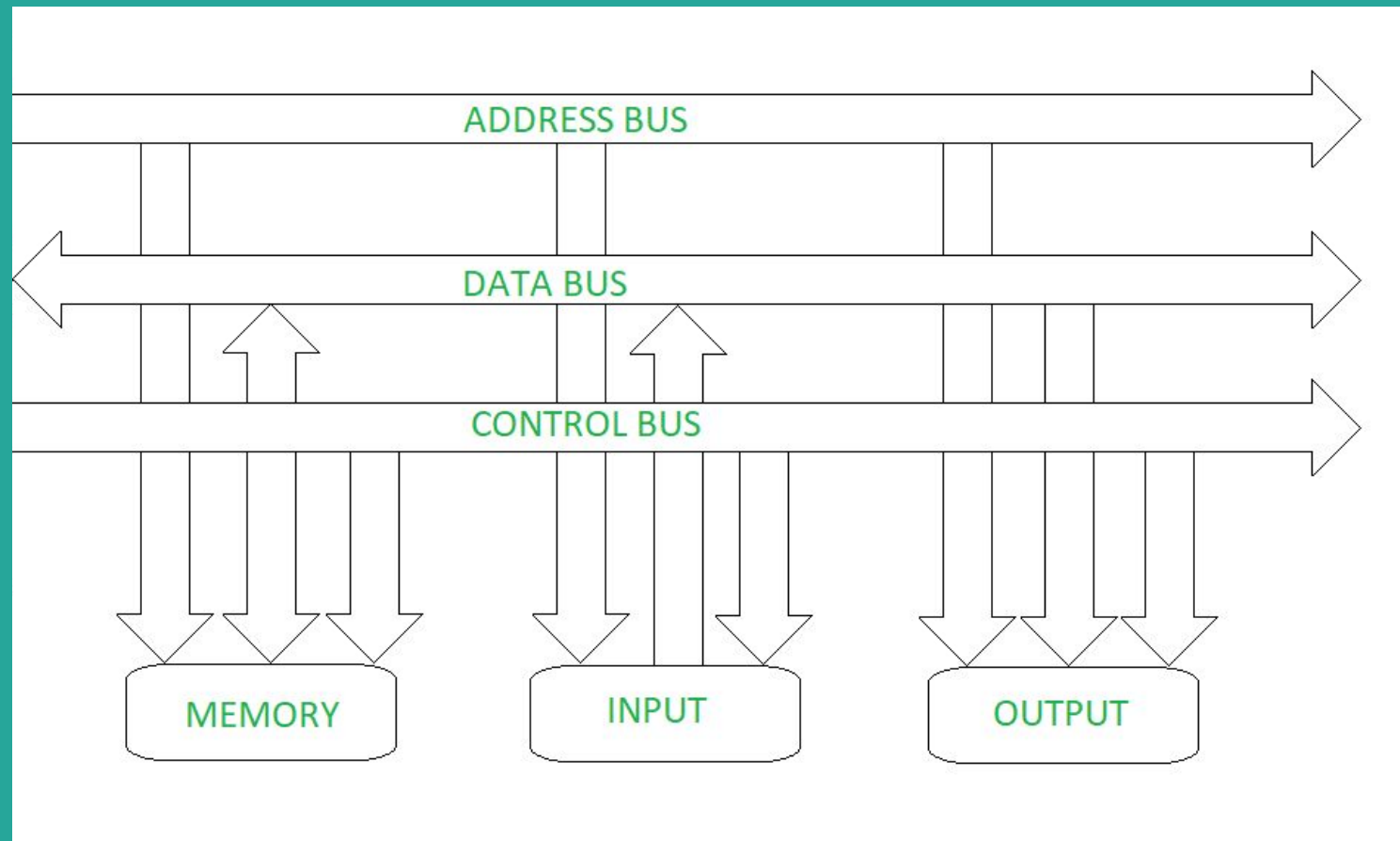
What is a BUS?

- Communication system that transmit binary data.
- ● Providing a path for data and address to traverse in the circuit.



Types of Buses

1. **Control bus** : carries control signal for controlling all the components. It is uni-directional from CPU to all other components.
2. **Address bus** : carries memory address. Computer performs all its task through the memory address. It is uni-directional from CPU to all other components.
3. **Data bus** : carries data from one component to another. It is uni-directional for input and output devices and bi-directional for memory and CPU.



See you tomorrow!

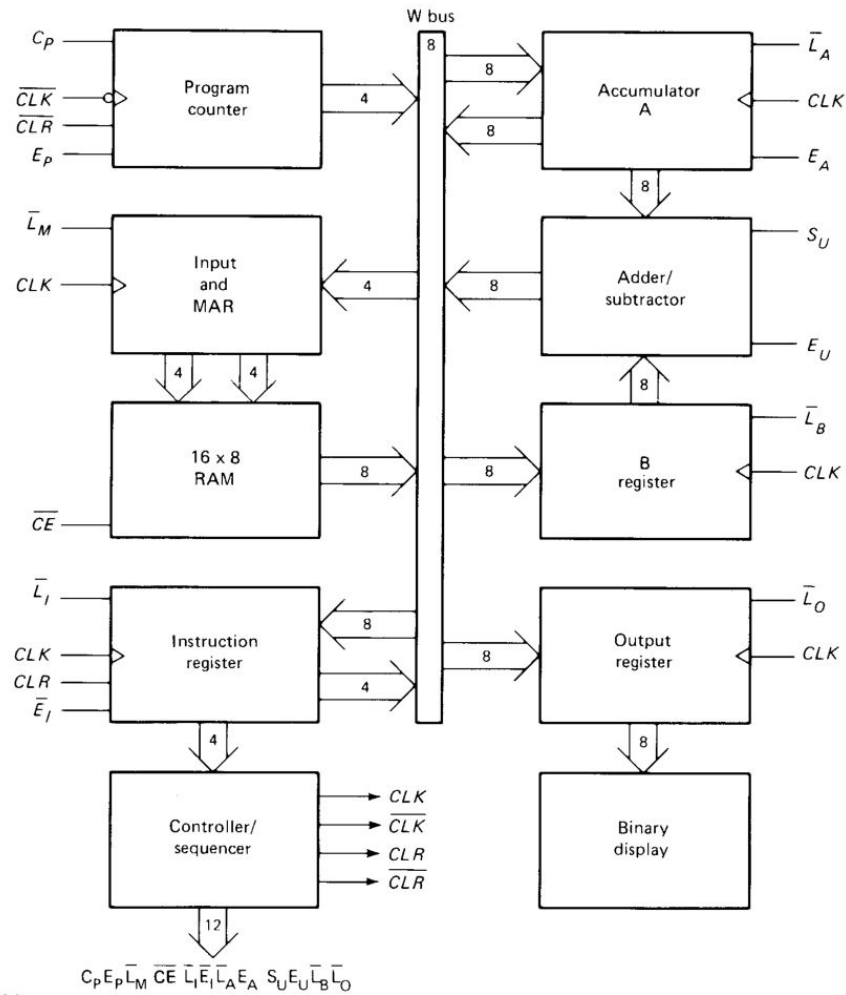
Simple-As-Possible Computers

Day 2

IEEE NIE Student Branch - Circuits & Systems Society

Recap

SAP 1 Block Diagram



The Program Counter

- 4-bit counter (0000 to 1111)
- Output is the address of the instruction to be executed
- Output is loaded into MAR

—

How to subtract?

- Take 2's complement of the subtrahend.
- Add Minuend.
- • The addition result is the answer!

Features of Adder/Subtractor Block

- Performs addition and subtraction on 8 bit binary data.
- One input is taken from the accumulator.
- The other comes from the B register.
- Since it's a combinational circuit the output is immediately calculated.
- You can control the data going in and out through control signals .

Let's go and build one!

—

Memory

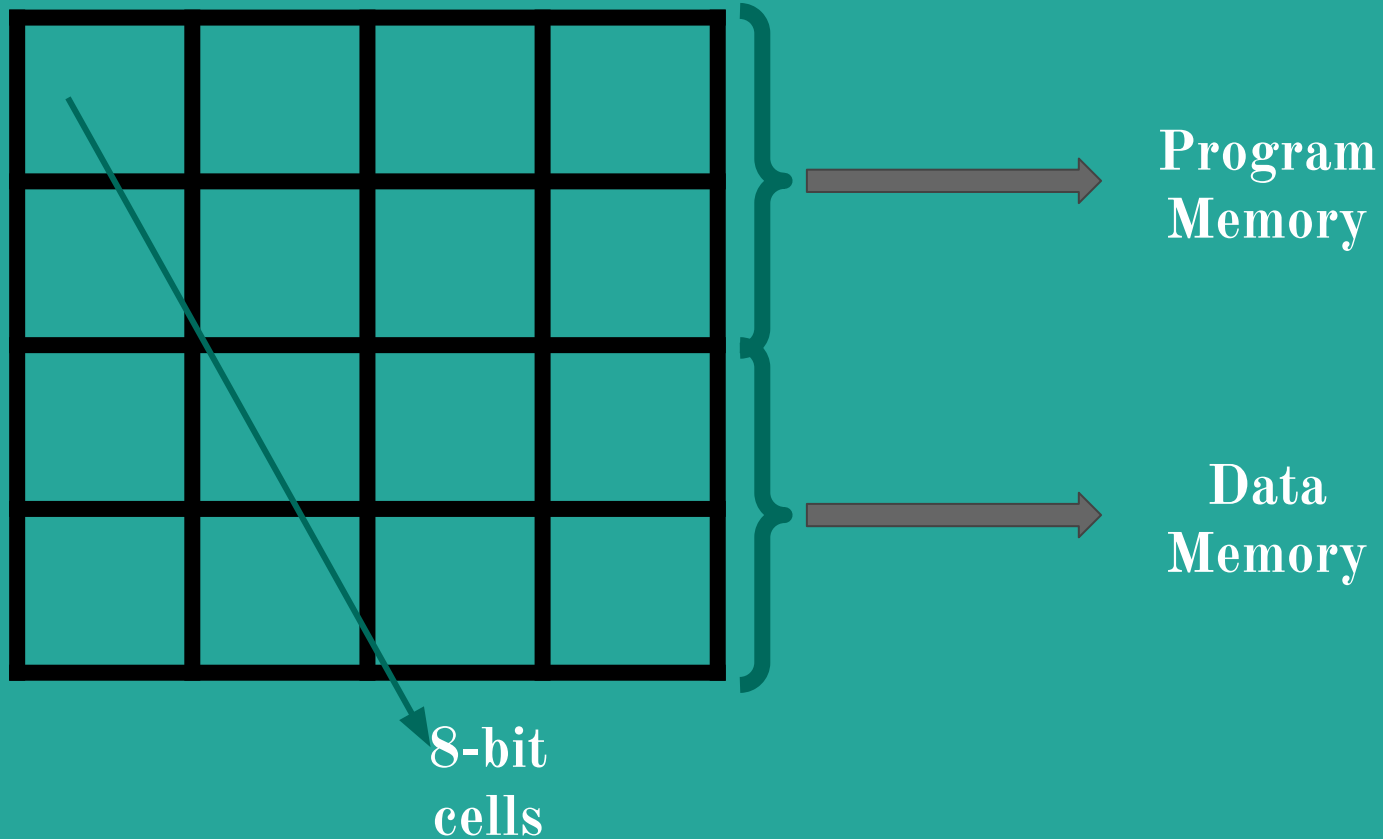
Volatile

RAM

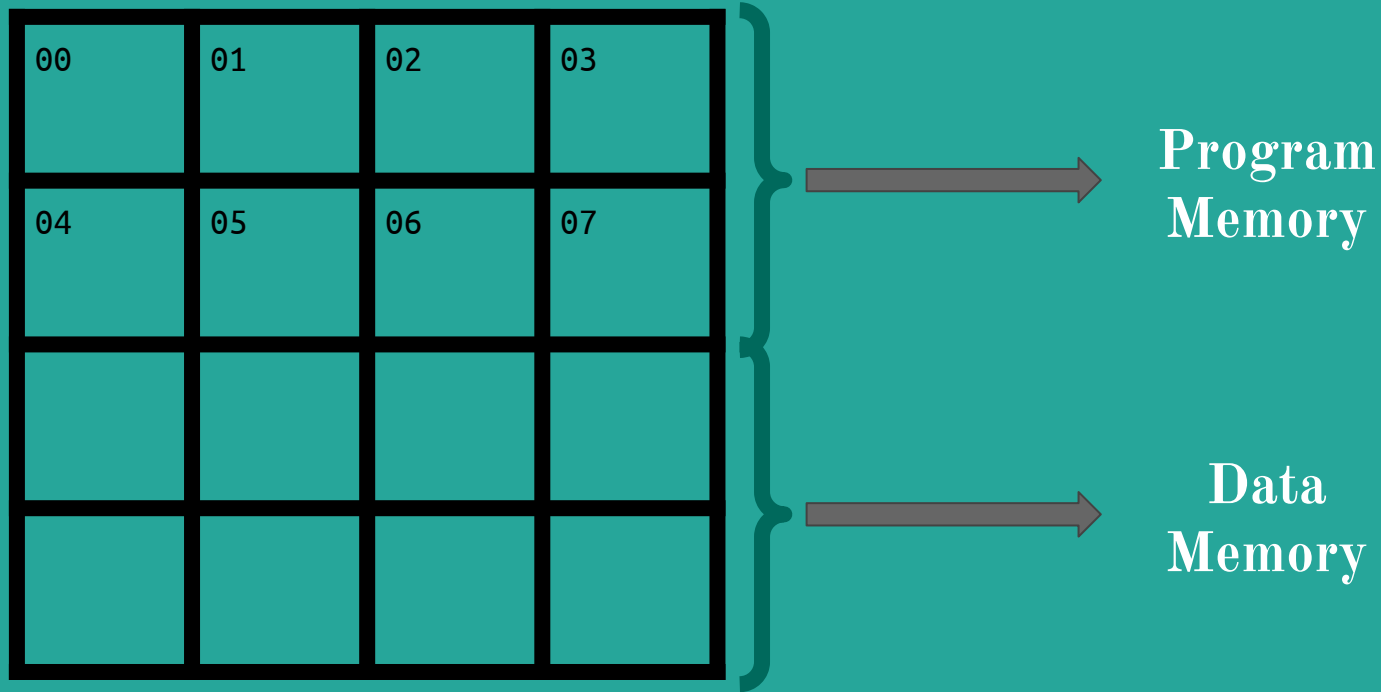
Non-Volatile

ROM,
✓EEPROM

Memory Partitioning Scheme

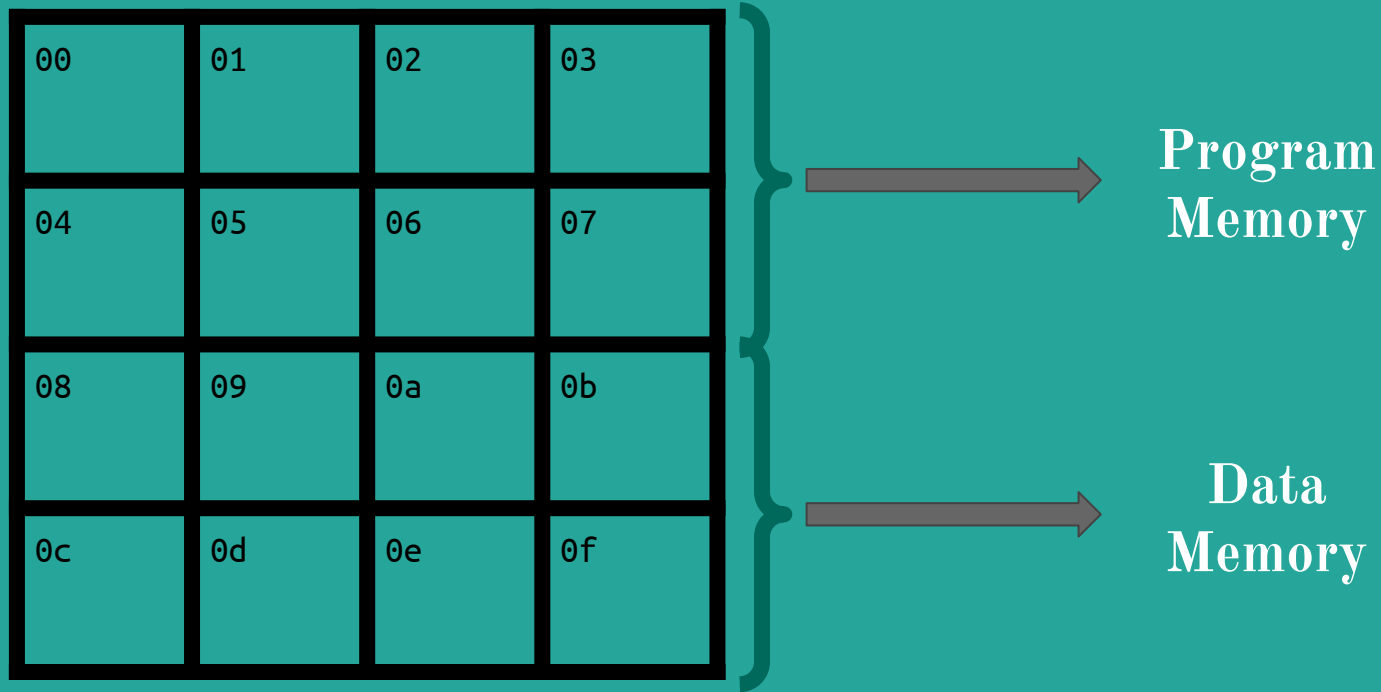


Memory Addressing



All addresses are in hexadecimal

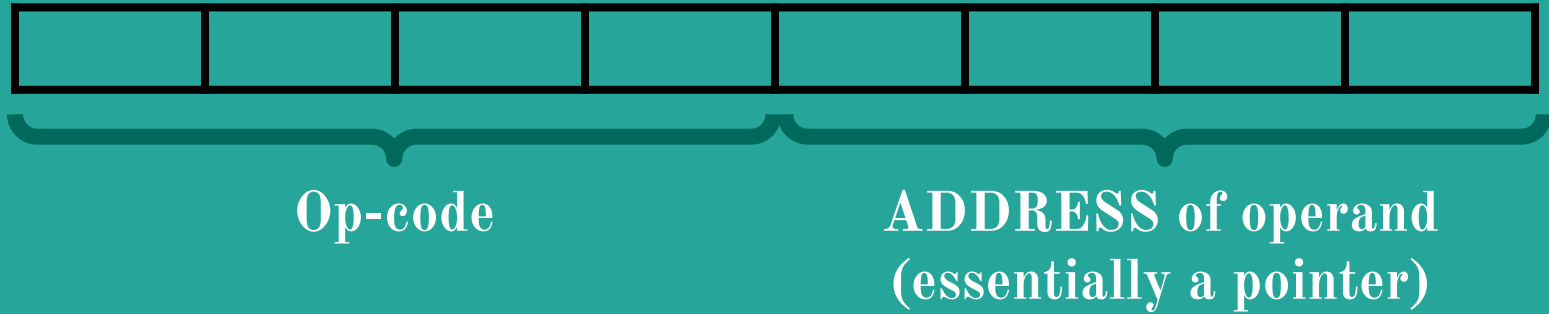
Memory Addressing



All addresses are in hexadecimal

The Instruction Set

8-bits



The Instruction Set

Op-Code	Machine Code
LDA	0000 (0x0)

The Instruction Set

Op-Code	Machine Code
LDA	0000 (0x0)
ADD	0001 (0x1)

The Instruction Set

Op-Code	Machine Code
LDA	0000 (0x0)
ADD	0001 (0x1)
SUB	0010 (0x2)

The Instruction Set

Op-Code	Machine Code
LDA	0000 (0x0)
ADD	0001 (0x1)
SUB	0010 (0x2)
OUT	1110 (0xe)

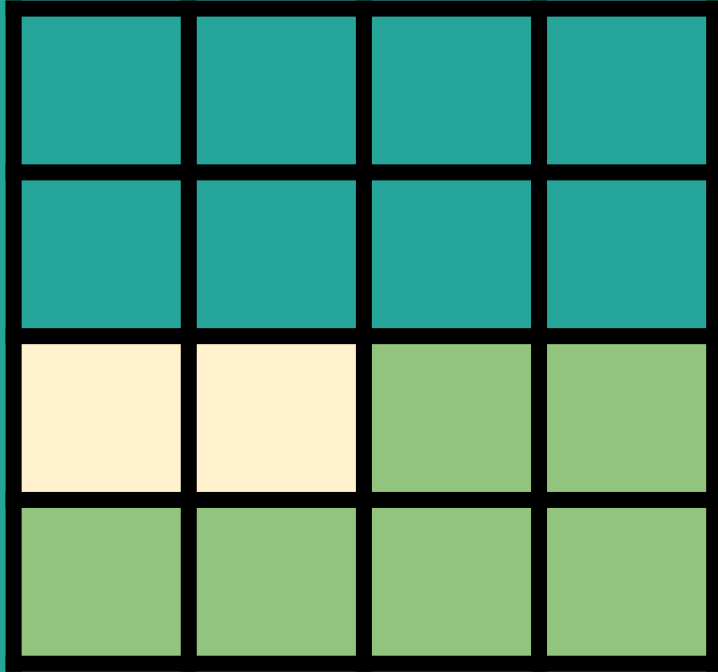
The Instruction Set

Op-Code	Machine Code
LDA	0000 (0x0)
ADD	0001 (0x1)
SUB	0010 (0x2)
OUT	1110 (0xe)
HALT	1111 (0xf)

Programming the SAP-1

1. Start with LDA
2. End with HALT
3. All data goes through the **accumulator**

An example program: Add 2 numbers



- LDA 0b1000
- ADD 0b1001
- OUT 0bXXXX
- HALT 0bXXXX

An example program: Add 2 numbers

08			

→ LDA 0b1000

→ ADD 0b1001

An example program: Add 2 numbers

08	19	e0	f0
05	01		

→ LDA 0b1000
→ ADD 0b1001
→ OUT 0bXXXX
→ HALT 0bXXXX

The Controller/Sequencer

Op-code



Control Word

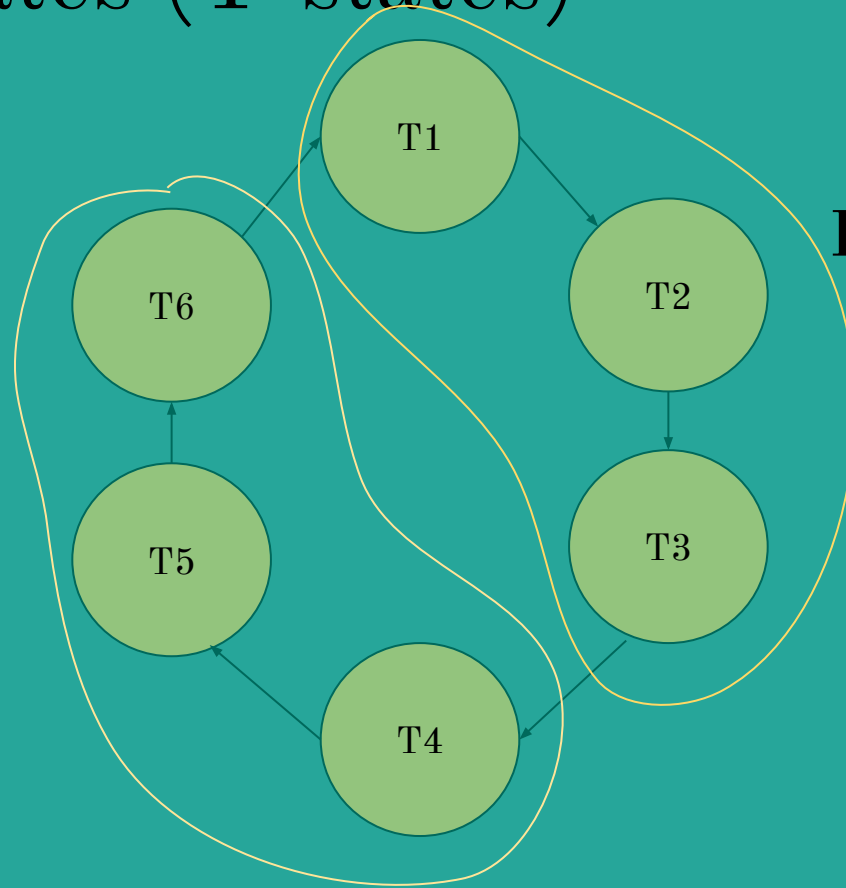
The Controller/Sequencer



\sim halt

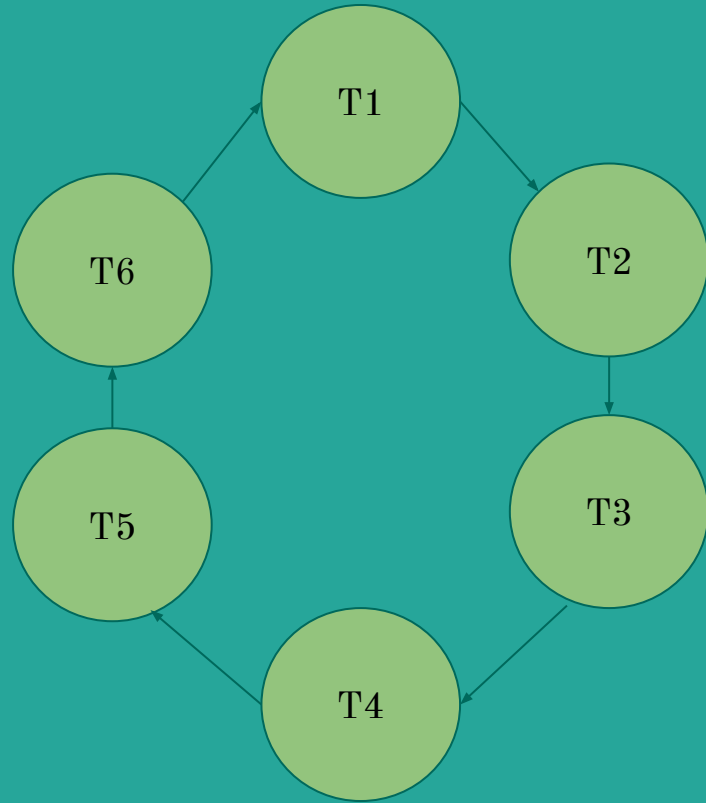
Timing states (T-states)

**Execute
cycle**



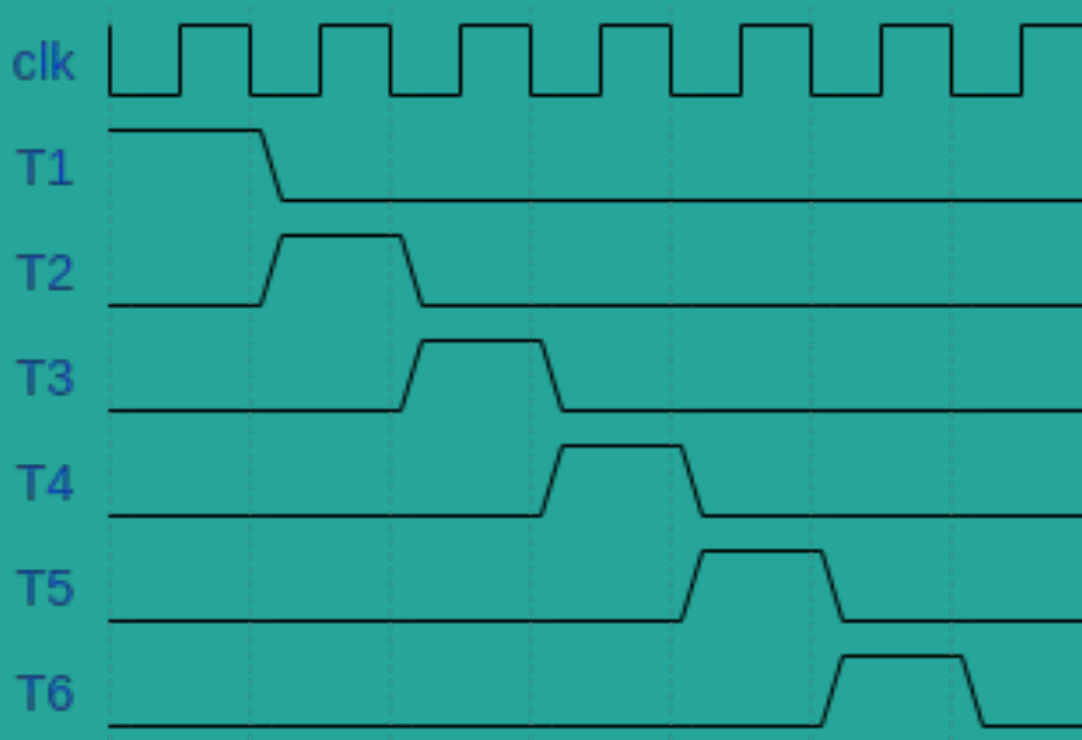
Fetch cycle

T-counter



T-state	Counter output
T1	100000
T2	010000
T3	001000
T4	000100
T5	000010
T6	000001

T-counter



T1 state (Address state)

- pc_out_en
- ~ld_mar

T2 state (Increment state)

- inc

T3 state (Memory state)

1. `~mem_out_en`
2. `~ld_ir`

LDA Routine

- T4 state
 - \sim ir_out_en
 - \sim ld_mar
- T5 state
 - \sim mem_out_en
 - \sim ld_acc
- T6 state
 - All signals are inactive (NOP state)

ADD Routine

- T4 state
 - ~ir_out_en
 - ~ld_mar
- T5 state
 - ~mem_out_en
 - ~ld_b_reg
- T6 state
 - subadd_out_en
 - ~ld_acc

SUB Routine

- T4 state
 - ~ir_out_en
 - ~ld_mar
- T5 state
 - ~mem_out_en
 - ~ld_b_reg
- T6 state
 - sub
 - subadd_out_en
 - ~ld_acc

OUT Routine

- T4 state
 - acc_out_en
 - ~ld_out_reg
- T5 state
 - NOP state
- T6 state
 - NOP state