

Memory

A collection of storage cells to allow users to read and write information.

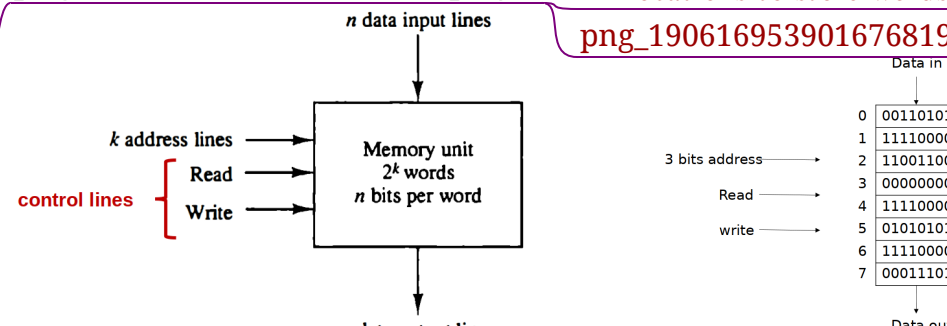
Stores information in binary form of a group of bits called **words**.

A word can represent a number, a character, an instruction...

Memory capacities is computed using the amount of **bytes** they can hold.

png_538135458310324597.png

2^k Locations to store words in
png_1906169539016768198.png



RAM

Random Access Memory

Main memory of any computer system

Allows access to any word in the memory with the **same** time delay, unlike magnetic disks (hard disks) for example.

Volatile: as long as power is switched off, RAM loses the data inside.

To access a specific word, we must have its **address** (location) in the memory, hence we need **address lines**.

address line = where are you in the ram

Read-only memory

Memory unit that performs the read operation only;

Rom

it does not have a write capability.

This implies that the binary information stored in a ROM is made permanent during the hardware production of the unit and cannot be altered by writing different words into it.

Found in BIOS chips (basic input/output system)

used to start computers, in calculators, and embedded systems.

Memory Read

To read a word from memory and transfer it to a register DR (Data register)

$DR \leftarrow M[AR]$

Memory Write

$M[AR] \leftarrow DR$

Arithmetic MacroOps

Complements

1's complement of 1010 is = 0101

2's complement of 1010 is adding 1 to its complement, so 0101+1 = 0110

How to Read a two's complement number?

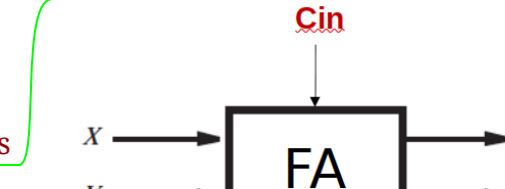
(1/2)ⁿ * (Decimal of the remaining n bits)

read 1110 (-2ⁿ⁻³ + 6)

png_7531981228065551108.png
R3 ← R1 + R2
R2 ← R1 - R2
R2 ← R2
R2 ← R2 + 1
R3 ← R1 + R2 + 1
R1 ← R1 + 1
R1 ← R1 - 1

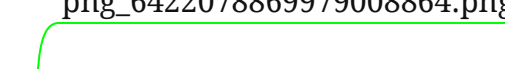
Contents of R1 plus R2 transferred to R3
Contents of R1 minus R2 transferred to R2
Complement the contents of R2 (1's complement)
2's complement the contents of R2 (negate)
R1 plus the 2's complement of R2 (addition)
Increment the contents of R1 by one
Decrement the contents of R1 by one

png_7046101930043306762.png



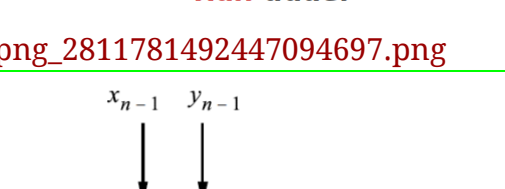
Full adder

png_6422078869979008864.png



Half adder

png_2811781492447094697.png



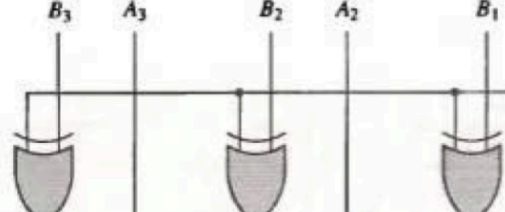
Ripple Carry

MSB position

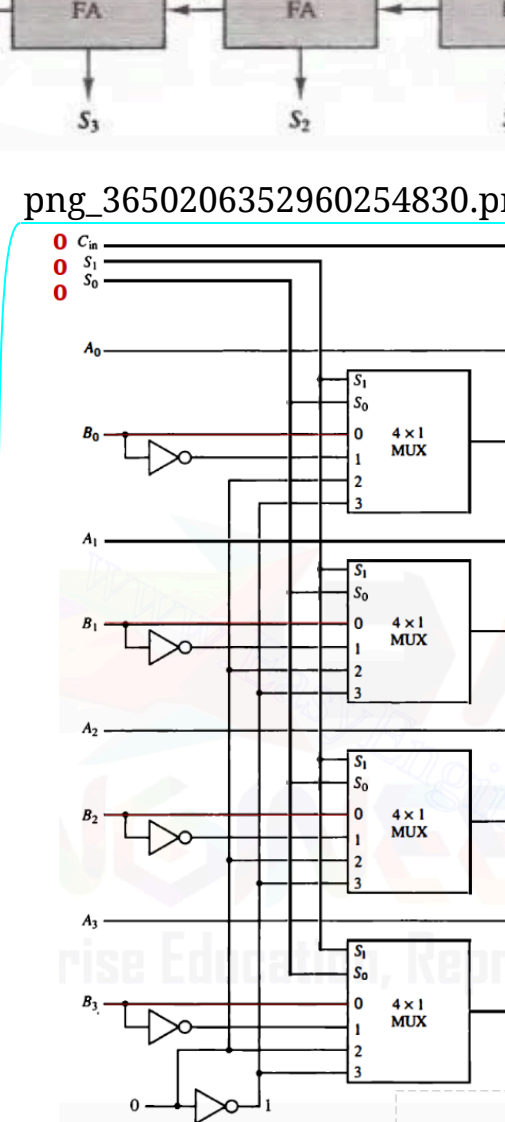
LSB position

Adder/Subtractor

png_3994149754646303770.png



png_3650206352960254830.png



Adder
Incrementer
Subtractor
Decrementer

png_3961094151632048285.png

Select	S ₃	S ₂	S ₁	S ₀	C _{in}	Input Y	Output D = A + Y + C _{in}	Microoperation
0	0	0	0	0	0	B	D = A + B	Add
0	0	0	0	1	0	B	D = A + B + 1	Add with carry
0	0	0	1	0	0	B	D = A + B + 1	Subtract with borrow
0	0	0	1	1	0	B	D = A + B + 1	Subtract
1	0	0	0	0	0	B	D = A	Transfer A
1	0	0	0	1	0	B	D = A + 1	Increment A
1	0	0	1	0	0	B	D = A - 1	Decrement A
1	0	0	1	1	0	B	D = A	Transfer A
1	1	1	1	1	1	B	D = A	Transfer A

Logic Microops

png_4049344150778555685.png

F₀ = 0
F₁ = xy
F₂ = xy'
F₃ = x
F₄ = x'y
F₅ = y
F₆ = x ⊕ y
F₇ = x + y

F ← 0
F ← A ∧ B
F ← A ∧ B̄
F ← A
F ← Ā ∧ B
F ← B
F ← A ⊕ B
F ← A ∨ B

Clear
AND
Transfer A

Transfer B
Exclusive-OR
OR

png_486270475014614267.png

F₈ = (x + y)'
F₉ = (x ⊕ y)'
F₁₀ = y'
F₁₁ = x + y'
F₁₂ = x'
F₁₃ = x' + y
F₁₄ = (xy)'
F₁₅ = 1

F ← A ∨ B̄
F ← A ⊕ B̄
F ← B̄
F ← A ∨ B̄
F ← Ā
F ← Ā ∨ B̄
F ← Ā ∧ B̄
F ← all 1's

NOR
Exclusive-NOR
Complement B
Complement A
NAND
Set to all 1's

Selection and Masking operations

Selective Set

Logic OR
A: 1 0 1 0
B: 1 1 0 0
A: 1 1 1 0

Selective Complement

Logic XOR
A: 1 0 1 0
B: 1 1 0 0
A: 0 1 1 0

Selective Clear

Set 0 in A when you see 1 in B
Logic A ∧ B'
A: 1 0 1 0
B: 1 1 0 0
A: 0 0 1 0

Masking

Only keep in A when you see 1 in B
Logic AND
A: 1 0 1 0
B: 1 1 0 0
A: 1 0 0 0

Inserting

mask first then oring
0110 1010
Example
Insert value 1001 in the most significant part.

Masking
A: 0110 1010
B: 0000 1111
0000 1010
0000 1010
Setting
OR 1001 0000
1001 1010

Shifting

png_3627610825448050589.png

Right Shift

Left Shift

the serial input is always a "0" either from left or right.

Logical Shift

Symbol: shl for shift left and shr for shift right

png_8380303952422635157.png
R1 ← shl R1
R2 ← shr R2

Circular Shift

rotate shift
Symbol: cil and cir
Example 00001101 cir 10000110

Arithmetic Shift

Arithmetic shift right: shift using the MSB

multiply by 2
divide by 2

There's a problem in this shift when the two MSB aren't the same

the problem is that the sign is reversed
When they are the same no problem arises

png_5935519168404583961.png
V_n = R_{n-1} ⊕ R_{n-2}

if the XOR = 1 there's a problem

Implementation

Sequential using flip flops

using Muxes

png_1382475158498188053.png

A is the input
H is the output

Select line decides whether to shift up or down.

Two serial inputs (left or right) # controlled by another control unit depending on what category of shift we want)

Try:
A = 0100
Shift left logical
Select must be set to '1' then

the ALU

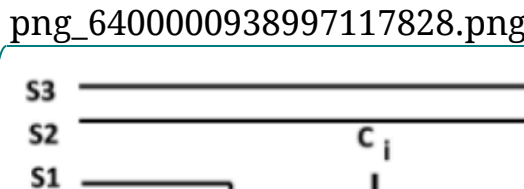
combining all circuits in one Arithmetic Logic Unit

ALU is a vital part inside any processor (CPU) and is responsible of carrying all computations.

ALU inputs should be connected to the registers as well as the ALU

output should be also connected to the registers to write the results back.

png_6400000938997117828.png



1-bit stage for addition/subtraction, increment/decrement

from slide 4

png_3029509375402852427.png

1-bit stage shifter

png_8265803753242467898.png

1-bit stage logic operations

You need to replicate this stage n times for n-bit ALU.

Of course a special care has to be taken for first and last stages for the Cin, cout, and the shift elements.

png_1872818176077968494.png

Operation select	S ₃	S ₂	S ₁	S ₀	C _{in}	Operation	Function
0	0	0	0	0	0	F = A	Transfer A
0	0	0	0	1	0	F = A + 1	Increment A
0	0	0	0	1	0	F = A + B	Addition
0	0	0	1	1	0	F = A + B + 1	Add with carry
0	0	1	0	0	0	F = A + B̄	Subtract with borrow
0	0	1	0	1	0	F = A + B̄ + 1	Subtraction
0	0	1	1	0	0	F = A - 1	Decrement A
0	0	1	1	1	0	F = A	Transfer A
0	1	0	0	0	0	F = A ∧ B	AND
0	1	0	0	1	0	F = A ∨ B	OR
0	1	0	1	0	0	F = A ⊕ B	XOR
0	1	0	1	1	0	F = Ā	Complement A
1	0	x	x	x	x	F = shr A	Shift right A into F
1	1	x	x	x	x	F = shl A	Shift left A into F