Register Transfer Language

Memory

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

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

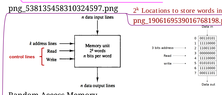
A word can represent a number, a character, an instruction...

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

png_5381354583103242597.png

2^ 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

Rom

Read-only memory

Memory unit that performs the read operation only;

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 <- M[AR]

Memory Write

M[AR] <- DR

Arithmetic MocroOPs

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 ?

(¿ 2)^ + (Decimal of the remaining n bits)

read 1110 (-2)^3 + 6

png_7531981228065551108.png

$R1 \leftarrow R1 + R2$	Contents of R1 plus R2 transferred to R1
$R3 \leftarrow R1 - R2$	Contents of R1 minus R2 transferred to R3
$R2 \leftarrow \overline{R2}$	Complement the contents of R2 (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement the contents of R2 (negate)
$R3 \leftarrow R1 + \overline{R2} + 1$	R1 plus the 2's complement of R2 (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of R1 by one
$R1 \leftarrow R1 - 1$	Decrement the contents of R1 by one

Adders

1-bit

png_7046101930043306762.png

Cin

full adders

Full adder

png_6422078869979008864.png

half adders

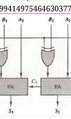
Half adder

Ripple Carry

png_2811781492447094697.png

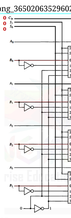
Adder/Subtractor

png_3994149754646303770.png



4-bit Arithmitic Circiut

png_3650206352960254830.png



Adder

Incrementer

Subtractor

Decrementer

png_3961094151632048285.png

Select		Input	Output	
S_1	S_0	C_n	Y	$D = A + Y + C_n$	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

Logic Microops

V logic **OR** we still can use +

^ logic **AND** we still can use *

⊕ logic **XOR**

png_4049344150778555685.png

$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR

png_4862704750146142674.png

$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive-NOR
$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow A \wedge \overline{B}$	NAND
$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

Selection and Masking operations

Selective Set

Sets the bits of register A to 1, when you see a 1 in B (in the same location). therwise, keep them as is.

Logic **OR** B: 1 1 0 0

A: 1 1 1 0

Selective Complement

Complement in A when you see 1 in B

A: 1 0 1 0

Logic **XOR** B: 1 1 0 0

A: 0 1 1 0

Selective Clear

Set 0 in A when you see 1 in B

A: 1 0 1 0

Logic A ^ **B'** B: 1 1 0 0

A: 0 0 1 0

Masking

Only keep in A when you see 1 in B

A: 1 0 1 0

Logic **AND** 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

✉000 1010

Setting

0000 1010

OR 1001 0000

✉ 1001 1010

Shifting

png_3627610825448050589.png

Right Shift

Left Shift

Logical Shift

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

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**

so the sign stays the same

divide by 2

Arithmetic shift **left**: shift using **0**

multiply by 2

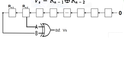
There's a problem in this shift when the two MSB aren't the same

the problem is that the sign is reversed

When they are the same no problem arises

To solve the problem we detect if the two bits aren't the same

png_5935519168404583961.png



if the **XOR** = 1 there's a problem

Implementation

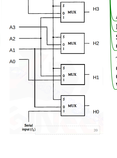
Sequential

using flip flops

Combinational

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

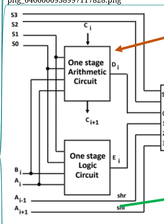
combining all circuits in one

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



png_5891691575578020 35.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_n	Operation	Function
0	0	0	0	0	$F = A$	Transfer A
0	0	0	0	1	$F = A + 1$	Increment A
0	0	0	1	0	$F = A + B$	Addition
0	0	0	1	1	$F = A + B + 1$	Add with carry
0	0	1	0	0	$F = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$F = A + \overline{B} + 1$	Subtraction
0	0	1	1	0	$F = A - 1$	Decrement A
0	0	1	1	1	$F = A$	Transfer A
0	1	0	0	×	$F = A \wedge B$	AND
0	1	0	1	×	$F = A \vee B$	OR
0	1	1	0	×	$F = A \oplus B$	XOR
0	1	1	1	×	$F = \overline{A}$	Complement A
1	0	×	×	×	$F =$ shr A	Shift right A into F
1	1	×	×	×	$F =$ shl A	Shift left A into F