Building a computer in Conway's game of life

The purpose of this document is to describe the functioning of a computer built in Conway’s game of life. An in-detail explanation of all the mechanisms and all the components of the computer would be too long to describe here. Only the fundamental principles and main components are explained here. I think this is enough to permit anyone to recreate a similar computer or reuse it's ideas and components for any other project.

The idea here is to illustrate the Turing completeness of game of life with a more impressive example, closer to our computers and easier to program than a basic Turing machine.

The project can be downloaded here:
https://drive.google.com/file/d/0B-4kRwE7luROR1VHclhJUUF3Qnc/view

1. Bases

The information is carried by glider beams. Their spatial period is 30, their time period is 60. This allows gilder beam crossings. The 4 basic components of the computer are:

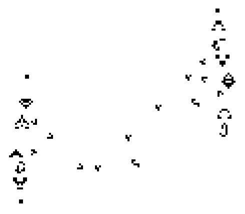
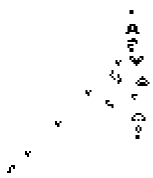
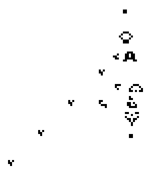
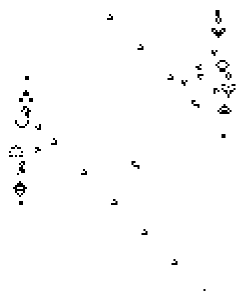
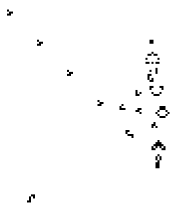
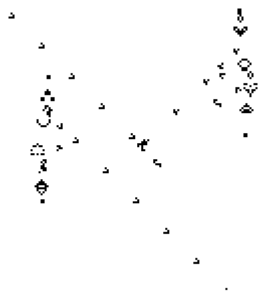
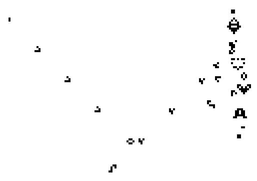
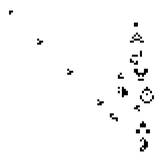
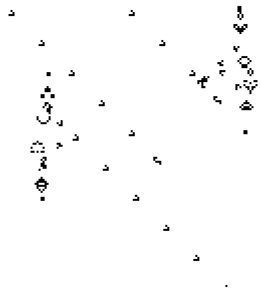
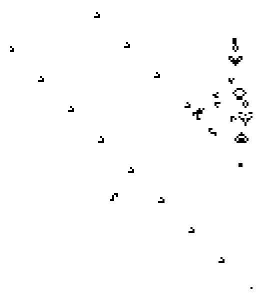
- a period 60 glider gun
  http://conwaylife.appspot.com/pattern/twogun
- a 90° glider reflector
  http://conwaylife.appspot.com/pattern/buckaroo
- a glider duplicator
  http://conwaylife.appspot.com/pattern/gliderduplicator1
- a glider eater
  http://conwaylife.appspot.com/pattern/eater1

The assembly of these components is made using Golly Python scripts.

2. Logic gates

The fact that two orthogonal glider beams can annihilate each other or form an eater if they intersect with the good phase shift is used to make basic logic gates.
This allows us to modify a glider beam with another.

Truth table

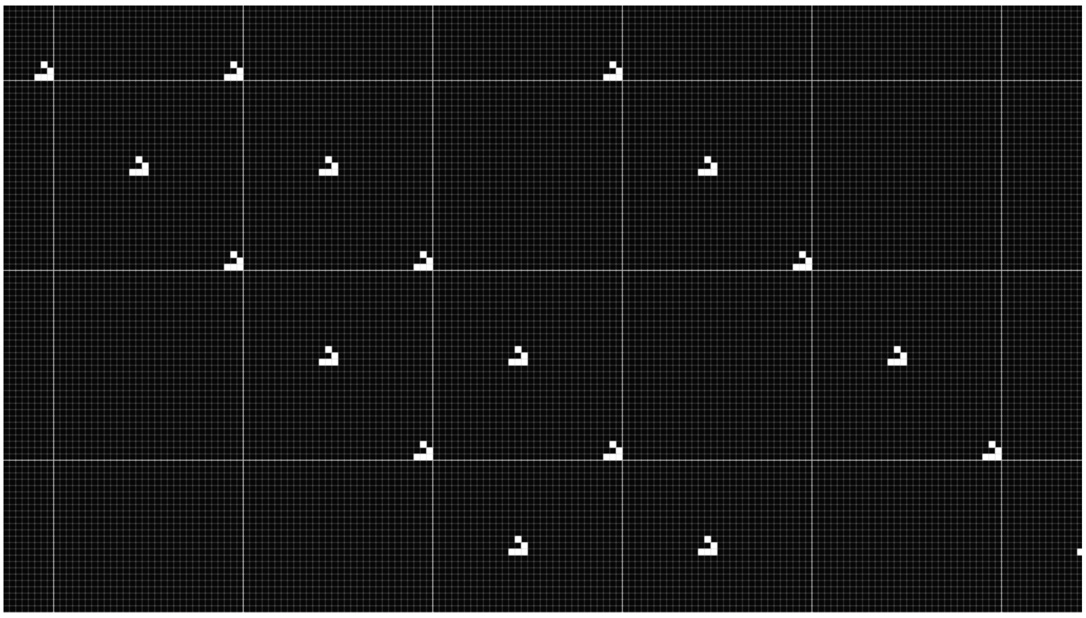
a	b	a OR b	a AND b	NOT a
0	0			
0	1			
1	0			
1	1			

3. Logic gates assembly, adaptators

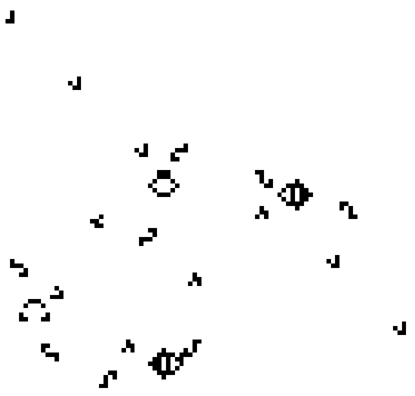
The assembly of logic gates is possible only if the glider beams are in phase and if the gap between the beams is right. While making the components out of multiple logic gates, I used the following conventions:

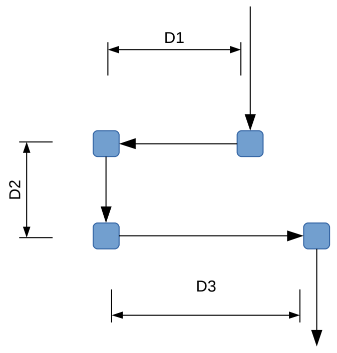
- used a 30*30 grid
- each glider beam has to match the corners of the grid
- main glider beams come from the high left corner.

According to these rules, these glider beams are well in phase:



A series of 3 reflectors called an adaptator is used to shift the phase or the position of a beam. The folder tools contains two Python scripts to make such adaptators. The scripts decamaker.py is used to translate horizontally a well phased beam by a multiple of 1 pixel. The script delamaker.py makes an adaptator of 3 dimensions d1,d2,d3 :

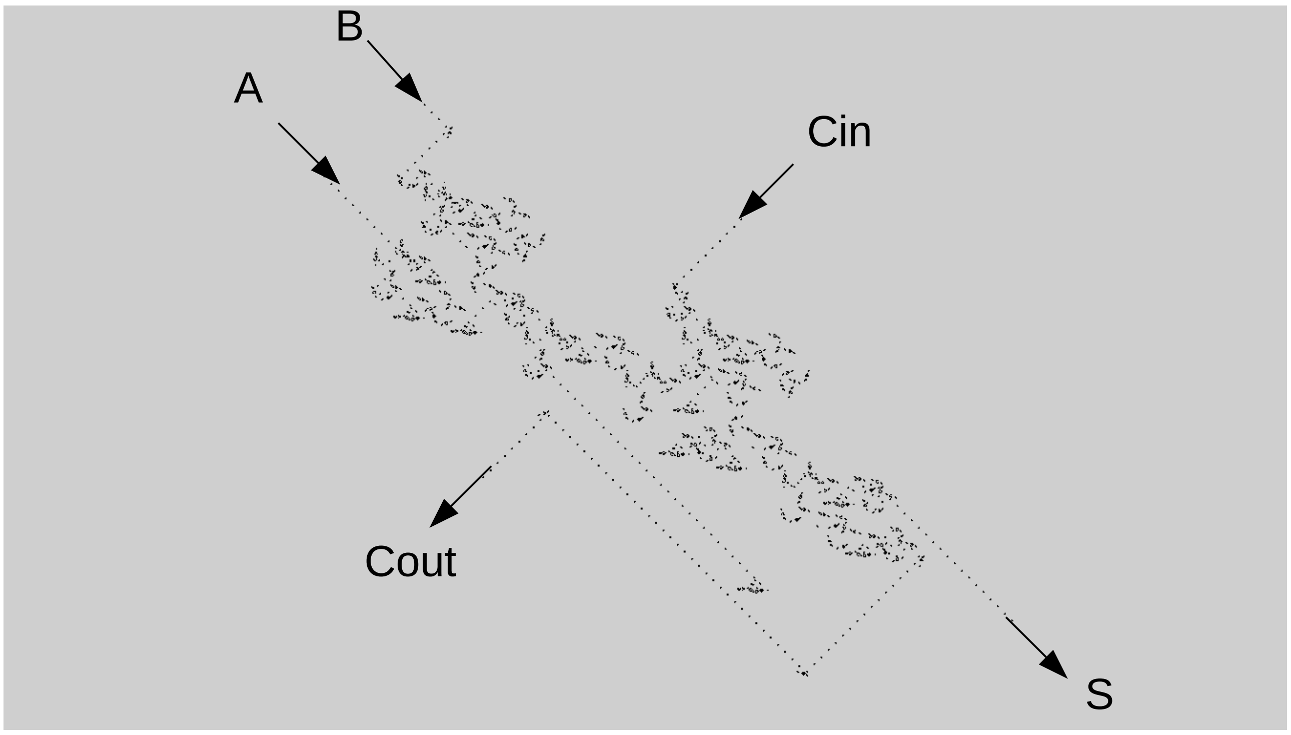




Adaptators are already included at the inputs and outputs of the logic gates from the folder « logic gates ». All their inputs and outputs are well in phase and it is just required to use the decamaker script to assemble them. I made the main components using these logic gates.

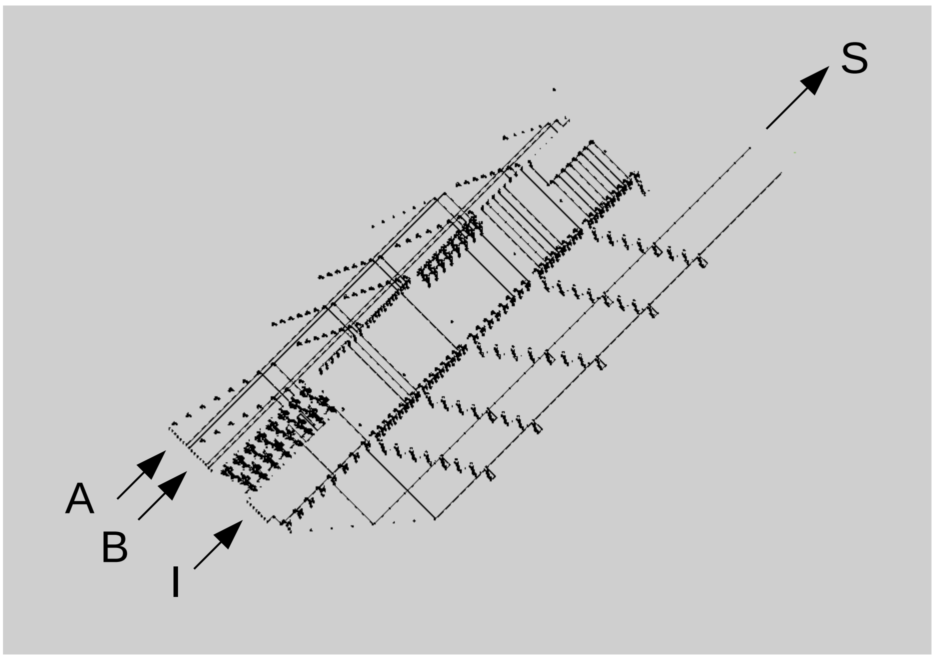
4. Adder

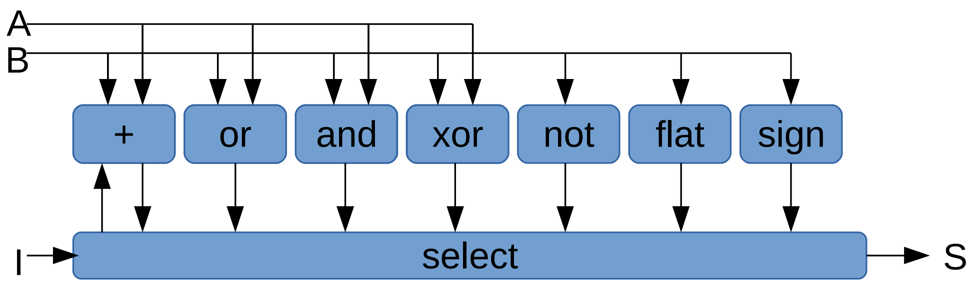
The implementation of the logic gates allows the making of any asynchronous logic circuit. Below is a full adder:



A,B : inputs
Cin : carry input
Cout : carry output
S : output
A ripple-carry-adder can be made by juxtaposing n full adders.

5. Arithmetic and logic unit (ALU)





A, B : inputs
I : instruction
S : output

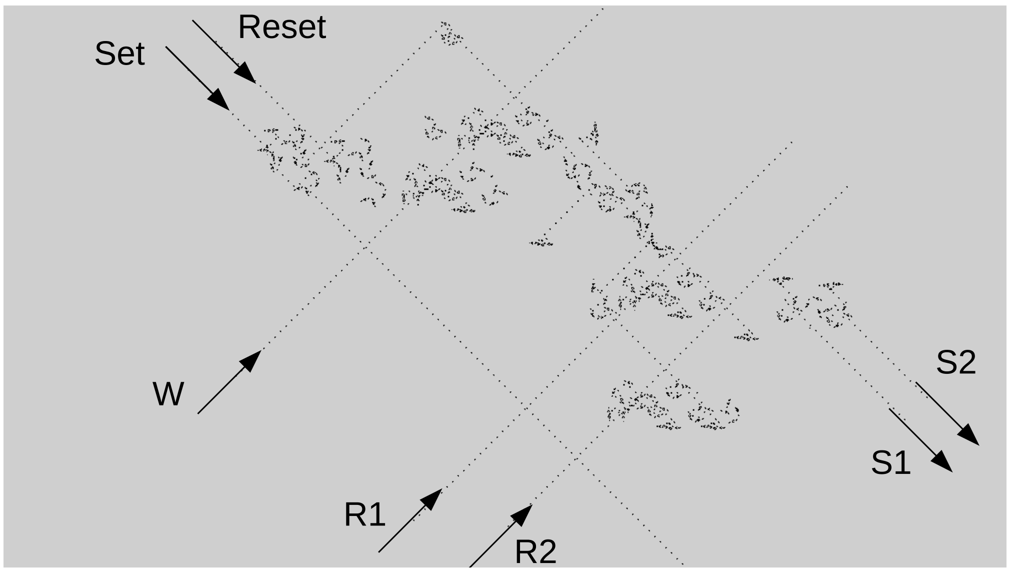
The ALU is composed of some logic functions and a unit which is composed of AND gates that select the result. The inputs are 8 bit each, the instruction is made of 8 independent bits. The instruction flat returns 11111111 if B≠00000000, the instruction sign selects the most significant bit of B. The most significant bit is at the high left corner. The arrow from the select module to the adder represents the increment instruction.

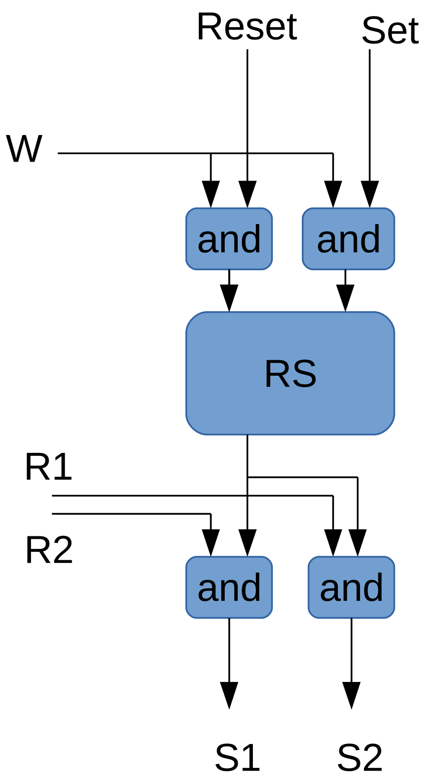
Instruction codes:

+	00000001
or	00000010
and	00000100
xor	00001000
not	00010000
flat	00100000
sign	01000000
increment	10000000

6. Memory

Memory unit:

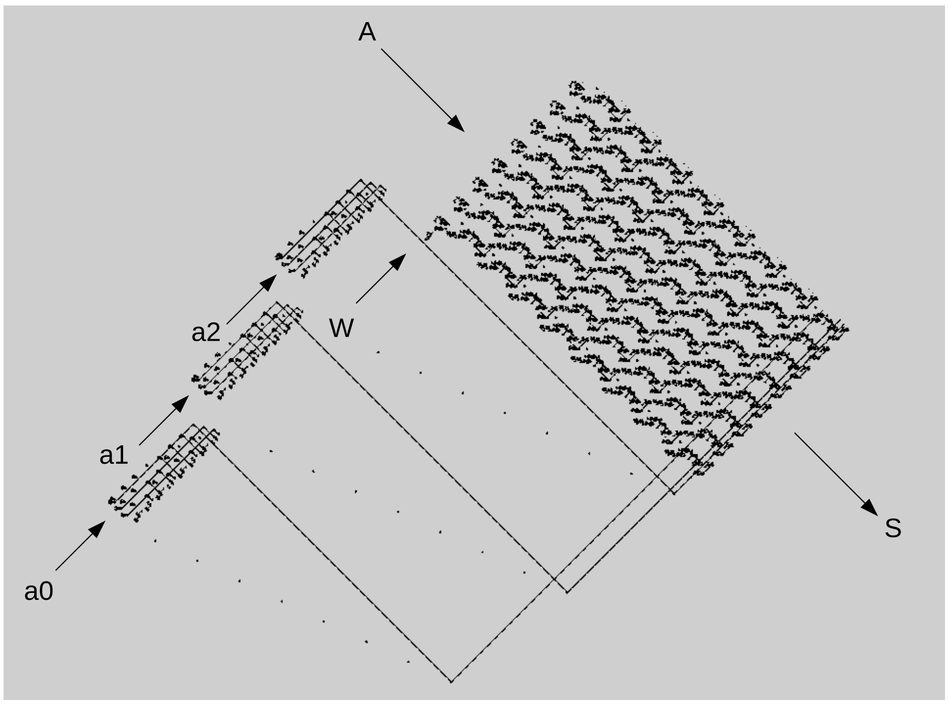




W : write 1 if set=1 0 if reset=1
R1 : read the state of the memory to the S1 output
R1 : read the state of the memory to the S2 output

A memory unit is composed of an RS latch and logic gates AND to manage reading and writing tasks. The memory is an assembly of 64 memory units in 8 lines and 8 columns thus forming an 8x8 bytes memory.

Memory:

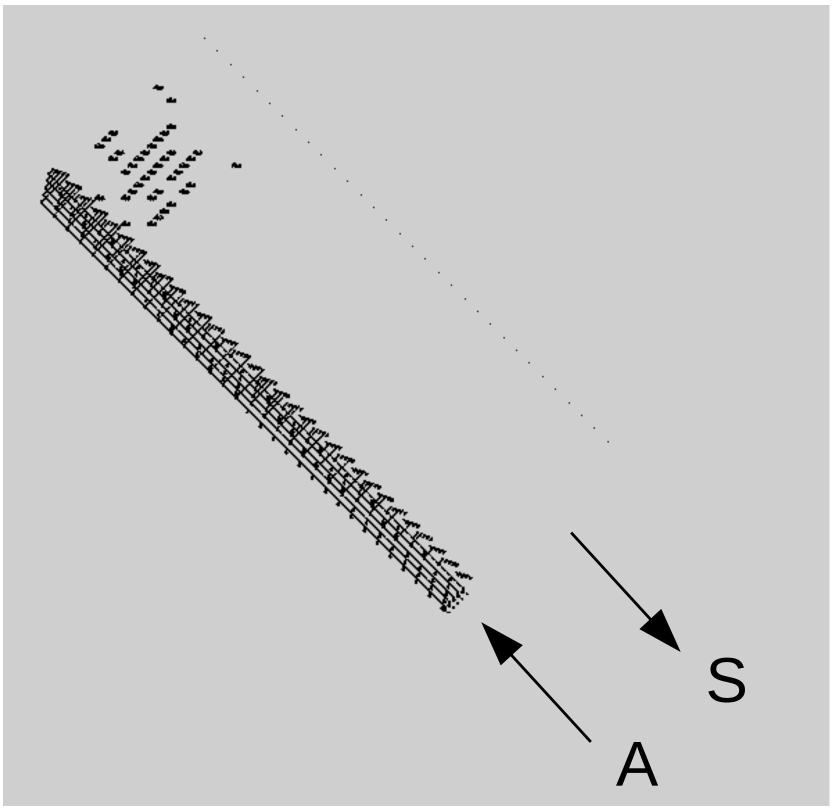


A : data to write, 1 byte
a0 : address for writing, 3 bits
a1 : address for reading, 3 bits
a2 : address for reading, 3 bits
S : output, 2 bytes
W : writes A to address a0 if W=1

The 3 modules at the left of the memory are decoders from 3 bit to 8 for decoding addresses. This memory allows us to read data from two addresses a1 and a2 at the same time. The output is made of 16 bit alternated between S1 and S2.

7. Program

The program is of a static memory form. The big rectangular module is a decoder from 5 bit to 32 which is used to select a line of the program given the input address.
Each line is made of 21 bits distributed as follow from the high right corner:
- 8 bits for data
- 3 bits for reading address
- 3 bits for reading address
- 3 bits for writing address
The script assembly.py contains more information on computer programming.



A : address of the line to be read: 5 bits
S : output: 21 bits

8. Architecture

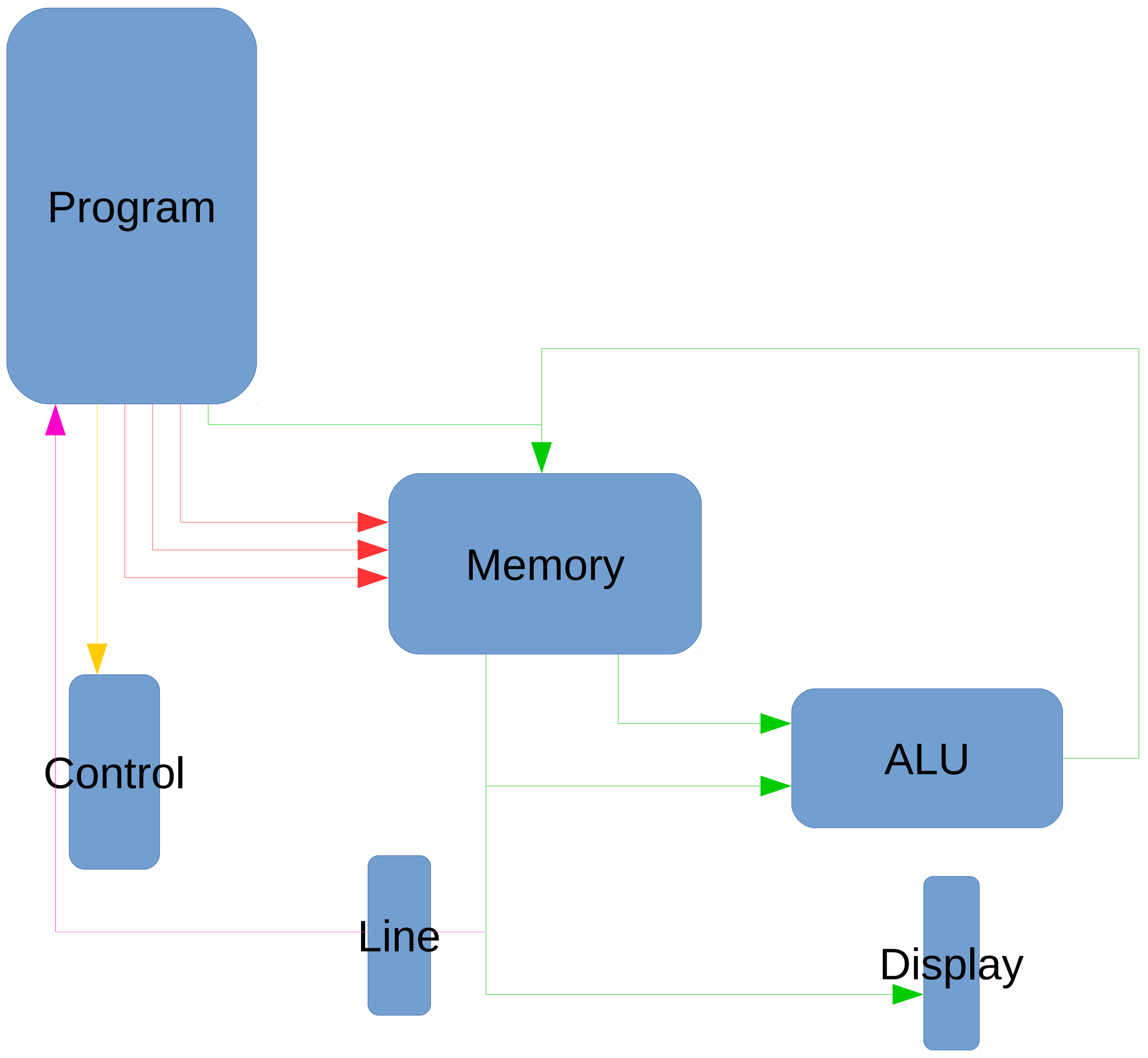


Diagram of the computer architecture. Only the main information channels are represented. The module “line” is a 5 bit memory for writing the address of the line which is running (PC). The module “control” is principally a decoder which takes as input an instruction code of 4 bits, decodes it into 16 independent bits and communicates with all the other modules to execute the instruction.

Elements which are not represented in this diagram:

- the control channels from the control module to other modules
- the clock
- the control channels from the clock to the other modules

Color code:

Green: 8bit data bus
Red: 3bits address
Yellow: 4bits instruction
Pink: 5bits address

9. Detail of a clock cycle

The clock is constituted of 4 loops, formed by glider reflectors in which glider beams rotate. In each loop, a glider duplicator is placed on the glider beam path. This makes it possible to extract the signal from the clock. The 4 loops give rhythm to: the execution of an instruction, the writing in the memory, the incrementation of PC, the writing of PC in the memory.

PC is stored in memory at address 000, however, it is needed to read it all along the execution of an instruction. It is therefore copied into the Line module on each cycle. The following table summarizes a clock cycle.

Generation	Clock loop	Signal edge	Action
0	0	Rising	Reading of an instruction, exectution of it by the ALU without writing on the memory
590 000	1	Rising	Writing of the ALU’s output on the memory
650 000	1	Falling	End of the writing on the memory
860 000	0	Falling	End of the reading of the instruction
1 100 000	2	Rising	Reading PC from address 000 to the ALU for incrementation Copy of PC from 000 to the Line module
1 440 000	3	Rising	Writing of the ALU’s output (PC+1) to the address 000
1 480 000	3	Falling	End of the writing of PC+1 to the address 000
1 600 000	2	Falling	End of reading from the address 000 and end of copying from 000 to the Line module At this time, PC+1 is written at 000 and on the module Line

The column “clock loop” gives he position of the clock loop which generates the signal. The index rises from the high left corner. Generation gives the time at which the signal leaves the clock loop. Each cycle is made of two parts: the execution of the instruction (loops 0 and 1) ant the incrementation of PC (loops 2 and 3). Each cycle has a period of 2,003,880 generations.

10. Programing

The Golly Python script assembly.py helps to program the computer. 8 variables can be used: a,b,c,d,e,f,g,h. h is at the address 000 and is used for storing PC.

Instruction	Effect
write a n	Write the number n in to the variable a
goto n	Go to line n
move a b	b=a
jumpif a	Jump the next line if a!=0
print a	Display a
add a b c	c=a+b
or a b c	c=(a or b)
and a b c	c=(a and b)
xor a b c	c=(a xor b)
not a b	b=not(a)
flat a b	B=0 if a=0 ; b=11111111 else
sign a b	Write the most significant bit of a to b (sign of a if a is written in 2’s complement)
increment a	a=a+1

Example : Euclid's algorithm

Display a modulo b		
line	instruction	pseudo-code
0	write a 8	a = 8
1	write b 3	b = 3
2	write e 1	d = -b
3	not b d	
4	add d e d	
5	add a d a	a = a+d
6	sign a c	if a>=0
7	jumpif c	
8	goto 5	go to line 5
9	add a b a	a = a+b
10	print a	print a
11	goto 11	end

Display GCD(a,b)		
line	instruction	pseudo-code
0	write a 8	a = 8
1	write b 6	b = 6
2	write e 1	c = a modulo b
3	not b d	
4	add d e d	
5	add a d a	
6	sign a f	
7	jumpif f	
8	goto 5	
9	add a b c	
10	jumpif c	if c = 0
11	goto 15	go to line 15
12	move b a	a = b
13	move c b	b = c
14	goto 3	go to line 3
15	print b	print b
16	goto 16	fin