Aristotle

A Logically Determined (Clockless) RISC-V RV32I

Matthew M. Kim*, Karl M. Fant**, Paul Beckett*

* RMIT University, Melbourne, Australia

** Theseus Research

Agenda

1. Introduce **Aristotle & Circuit Oscillations**
2. Explain **Logically Determined Design** & **Null Convention Logic** (NCL)
3. Show the **Waveform Analysis** of Aristotle CPU Simulation
4. Suggest **Instruction Grouping** & new **Bit Allocation** for a Clockless RISC-V

Aristotle

A logically determined RISC-V

- does only what is logically necessary.

- Implemented entirely in terms of logical relationships with **Null Convention Logic(NCL)**, a threshold logic with hysteresis behaviour, that constructs fully determined, self-regulating logic networks through which spontaneously flow successive computations.

Low Power - Low Noise

- No clock, no state machine, no flip-flops, no glitching, no wasted switching

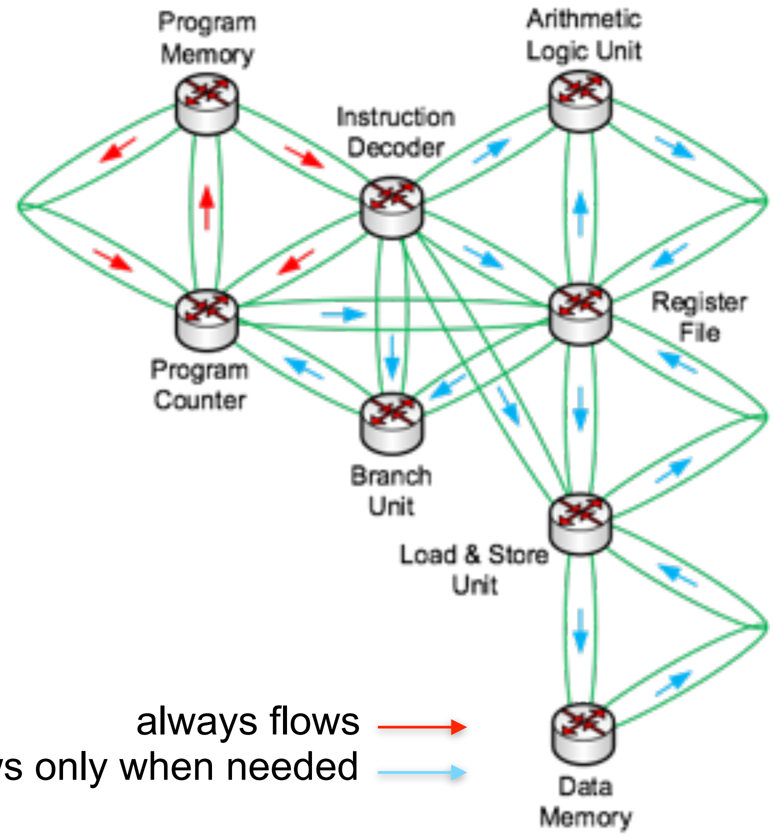
Robust - Reliable - Portable - Evolvable

- no critical timing relations - completely logically determined behavior
- insensitive to variations of voltage, temperature, fabrication

Efficient

- only necessary activity only for as long as needed

A Structure of Linked Oscillations of RISC-V RV32I



always flows →

flows only when needed →

- CPU Function Modules (Cylinder Router Symbol)
- Linked oscillations (Green Eclipses)

How does it work?

Threshold Completeness Operators
with Hysteresis State Holding Behaviour

Defining two logical Values Data and Null ("not data")

NULL explicitly means "not data"

Threshold Completeness Operators with Hysteresis State Holding Behavior

Define operators that demand input completeness to transition output.

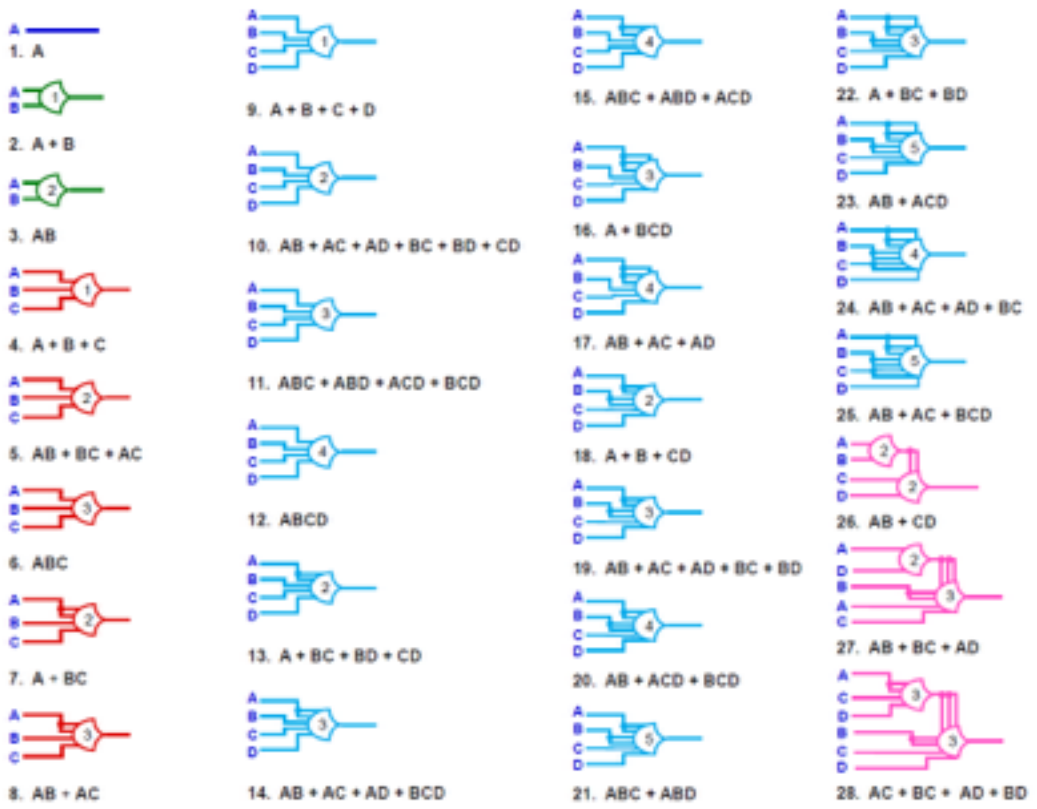
2 of 2

	D	N
D	D	–
N	–	N

2 of 3

	DD	DN	ND	NN
D	D	D	D	–
N	–	–	–	N

28 NCL Threshold Operators - Cover all possible 4 or less Input threshold Functions

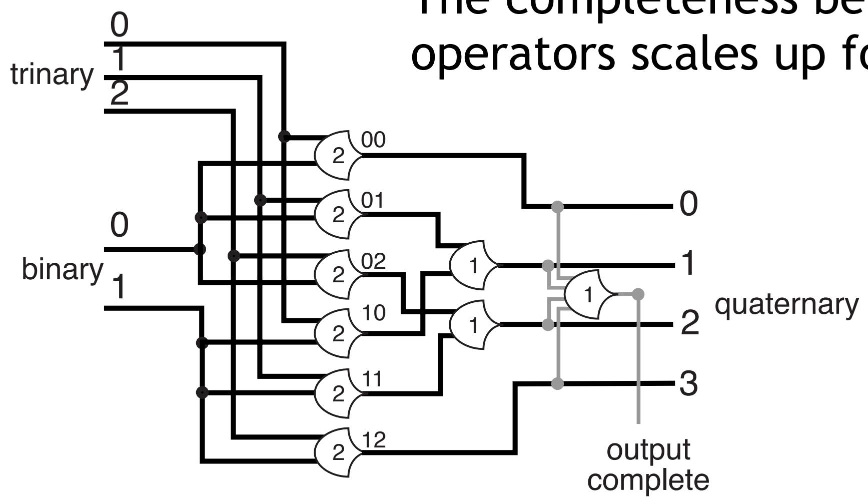


This is Null Convention Logic (NCL)
the Logically Determined design language

Multi-rail Representation

Since there is only one data value variables and their values are represented with multiple rails that mutually exclusively assert data.

Logically Determined Combinational Flow



The completeness behavior of individual operators scales up for the circuit as a whole

The output of a combinational circuit will completely transition its output only when its input has completely transitioned

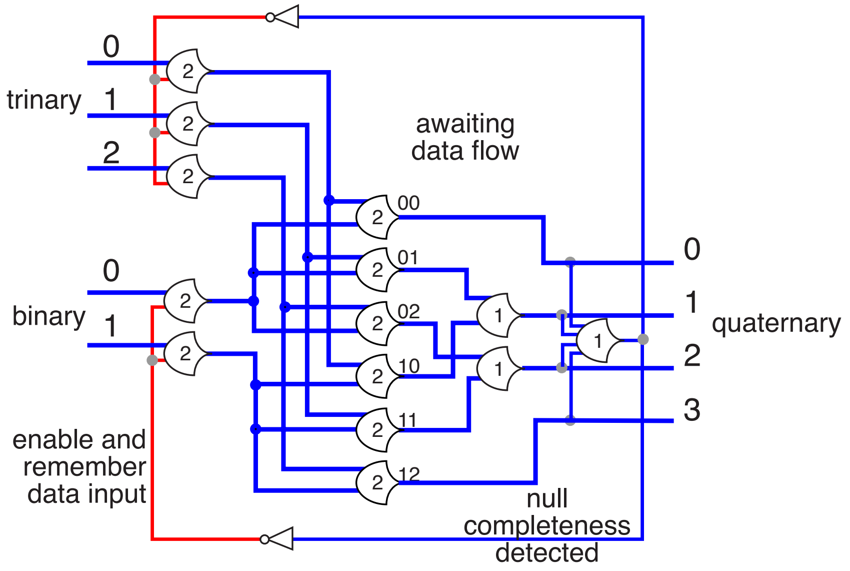
The circuit can logically determine when its output is completed

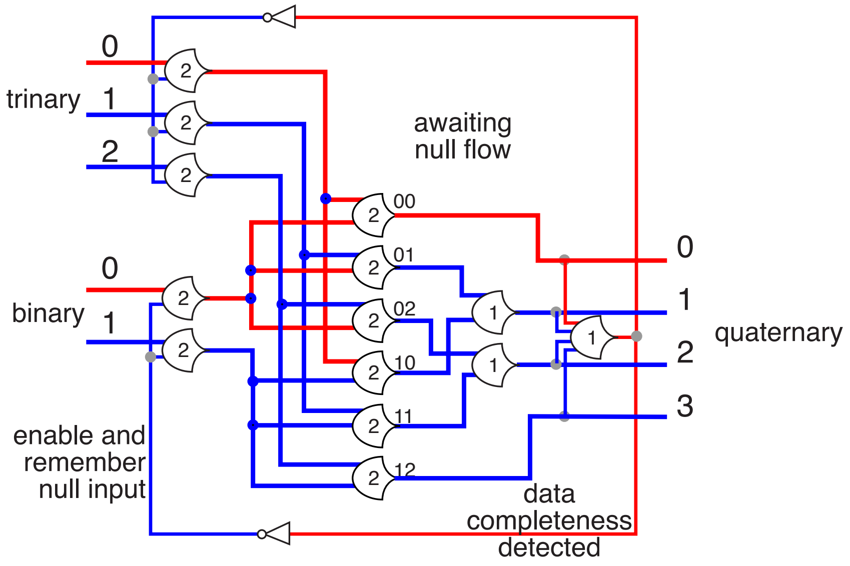
The flow of the circuit is completely logically determined

No races, No glitches, No spurious transitions

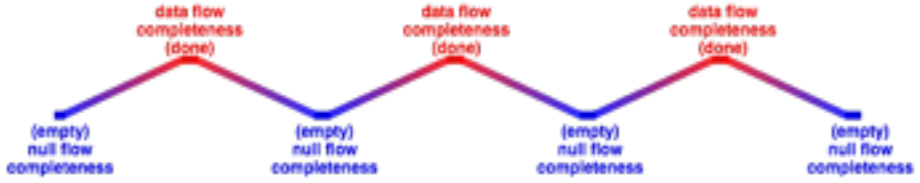
Self Regulating Circuit

The circuit can use its output completeness to regulate its input.





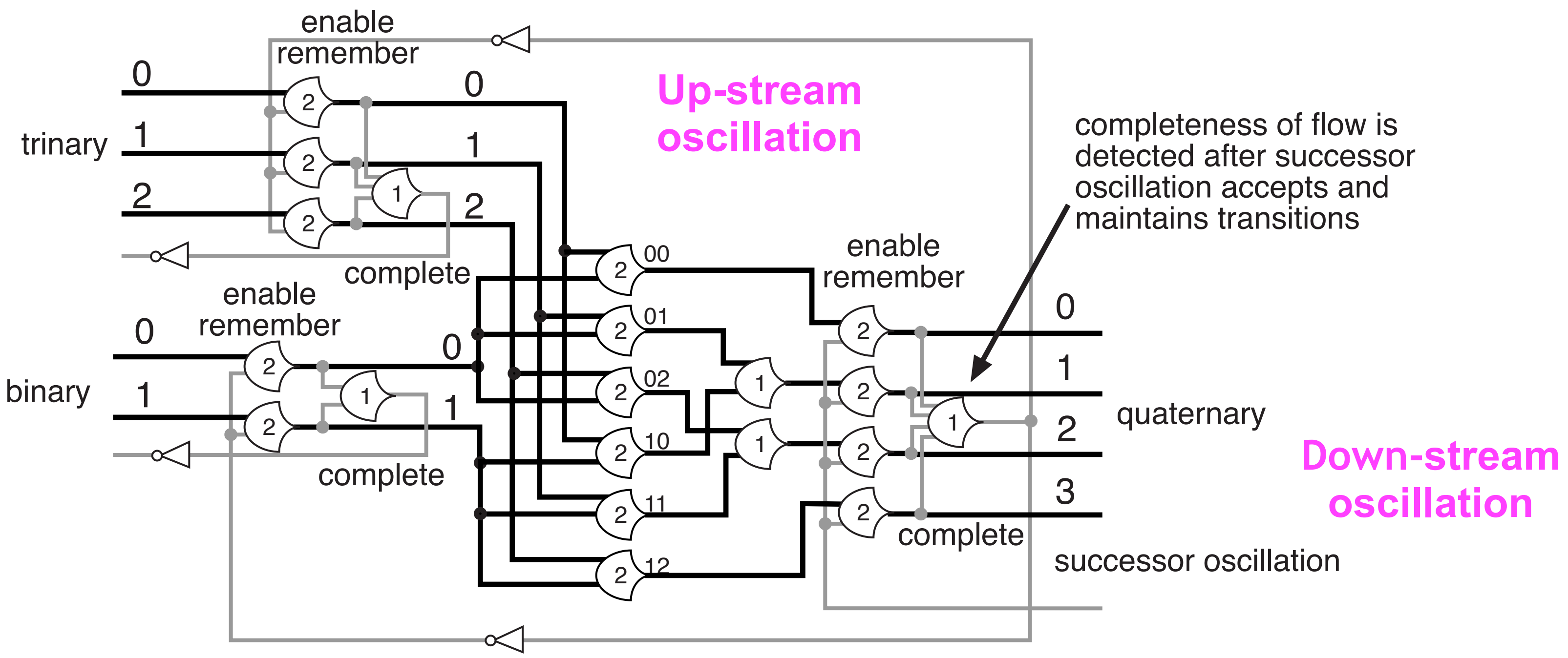
The feedback with the **single inversion** turns the circuit into an oscillation that **monotonically oscillates** between data completeness and null completeness (emptiness), **regulating its own input** and **providing liveness.**



data flow completeness (done)
data flow completeness (done)
data flow completeness (done)
(empty) null flow completeness
(empty) null flow completeness
(empty) null flow completeness
(empty) null flow completeness

Linking Oscillations

Oscillations are linked by placing the input enable of a downstream oscillation before the completeness detection of upstream oscillation.



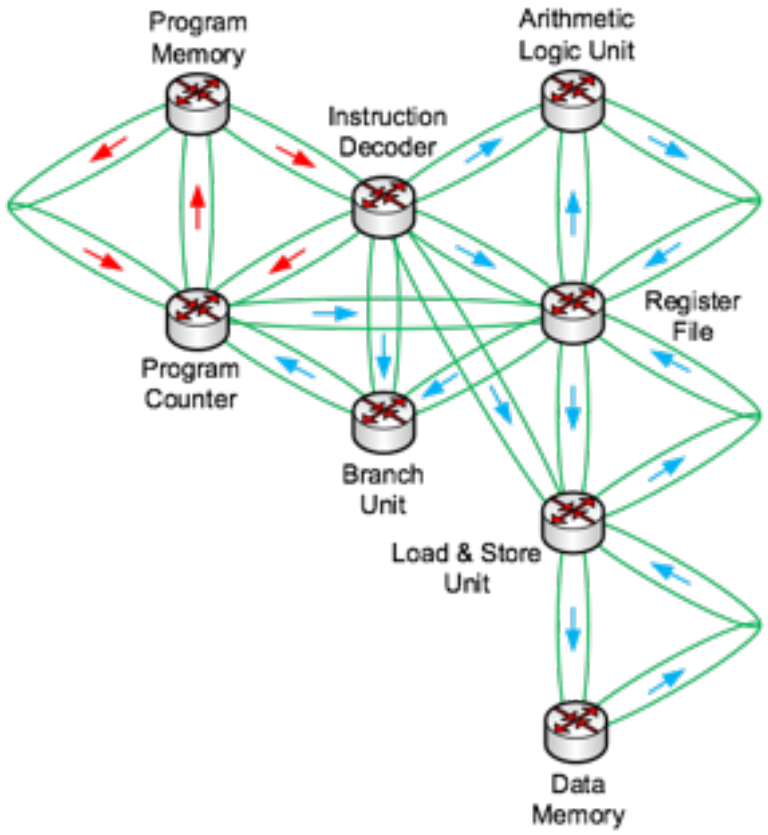
Wavefronts of transition are stably handed from oscillation to oscillation as they flow through a network of linked oscillations transforming as they flow

A System of Linked Oscillations

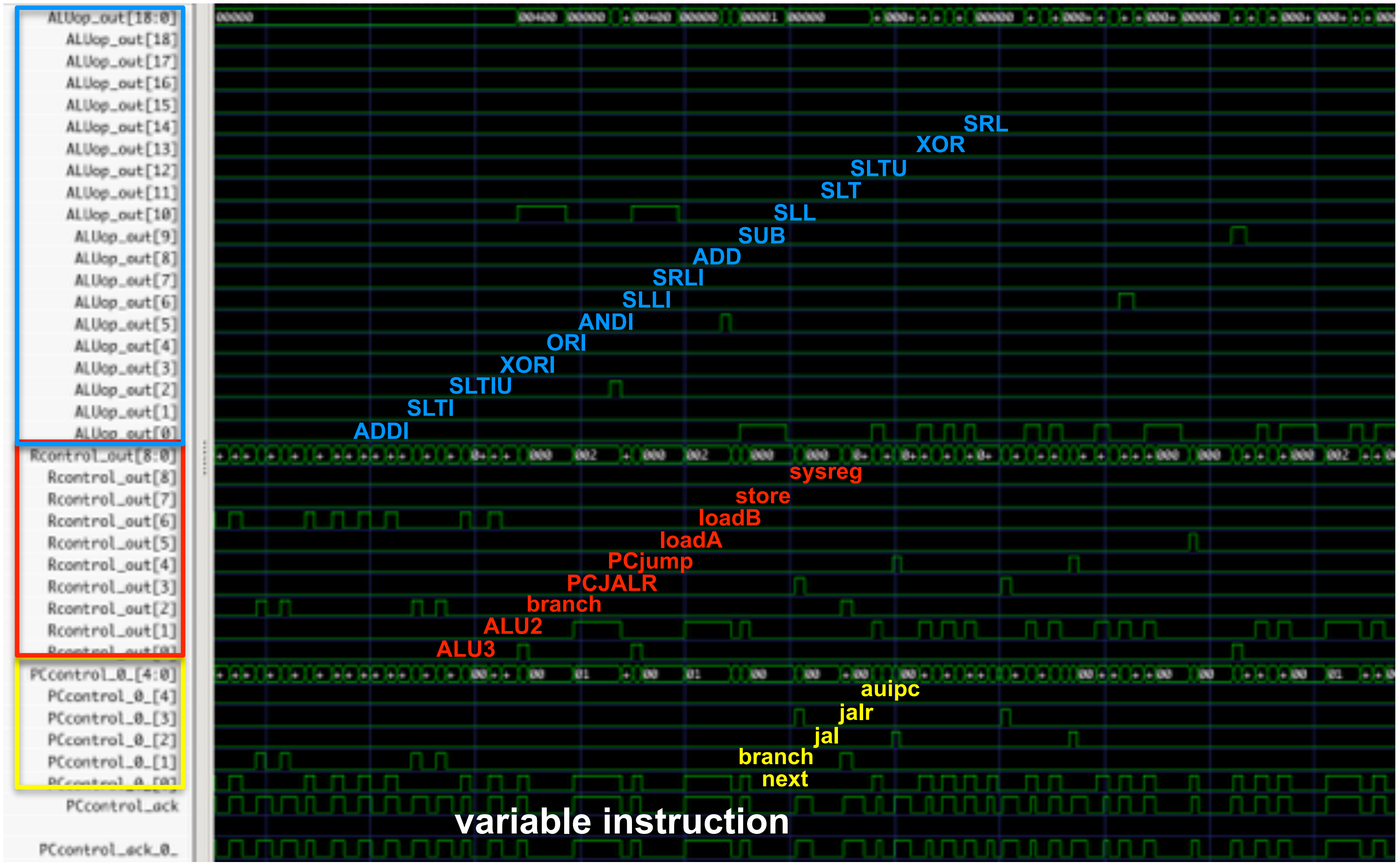
An entire system -
combinational logic,
coordination logic,
memory
- is expressed solely
in terms of logical connectivity
as a Network of Linked Oscillations.

No clock,
No flip-flops,
No registers,
No state machine.

A System of
linked oscillations.



Multi-rail Control and Variable length instructions

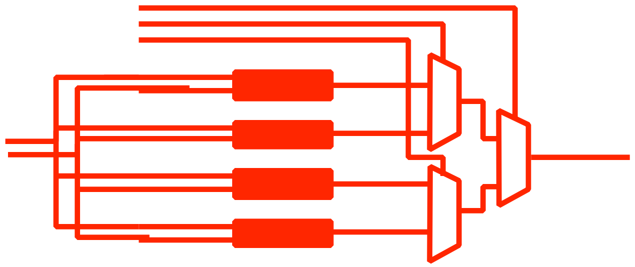


ALU Control Rails
Register File Control Rails
Program Counter Control Rails

Power Economy by Steering Instead of Selecting

Select (MUX)

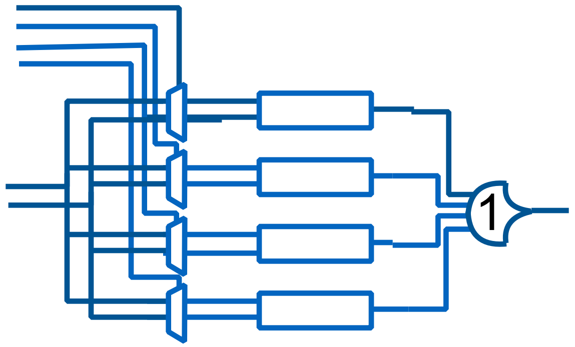
do everything, Mux discards undeeded



Always presenting data
Present new data, wait appropriate time,
select desired flow, discard rest

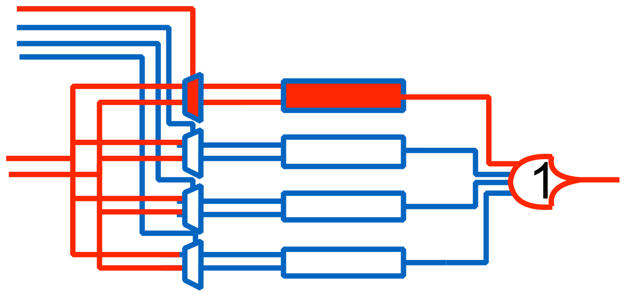
Steer (DEMUX)

do only what is necessary



Normally NULL (not data, empty)

multi-rail
steering

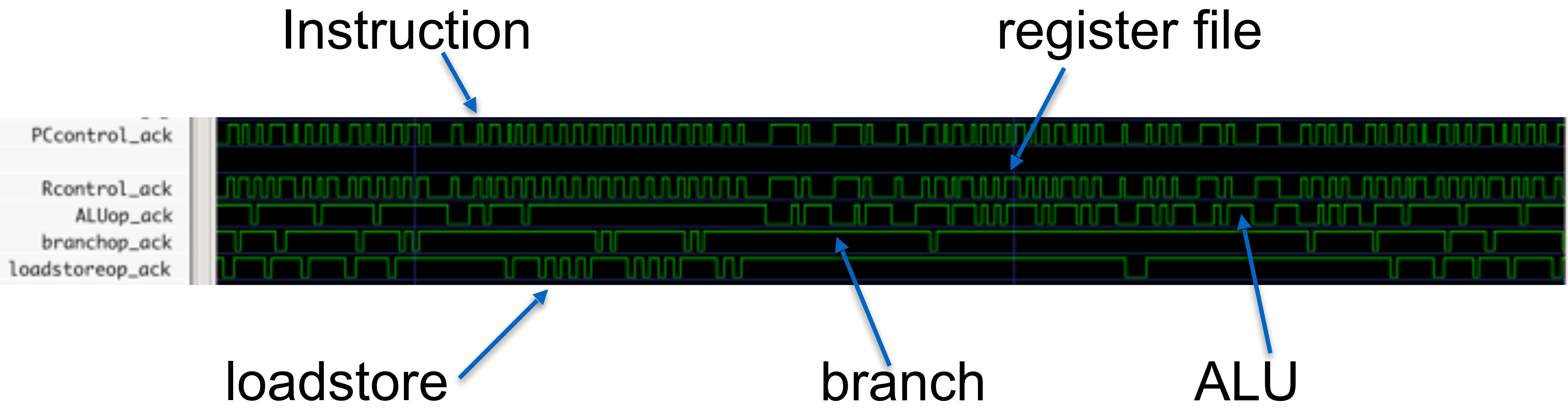


Present next DATA, steer to function
through a background null emptiness

Only steered circuit is active,
this is the main reason of low power.

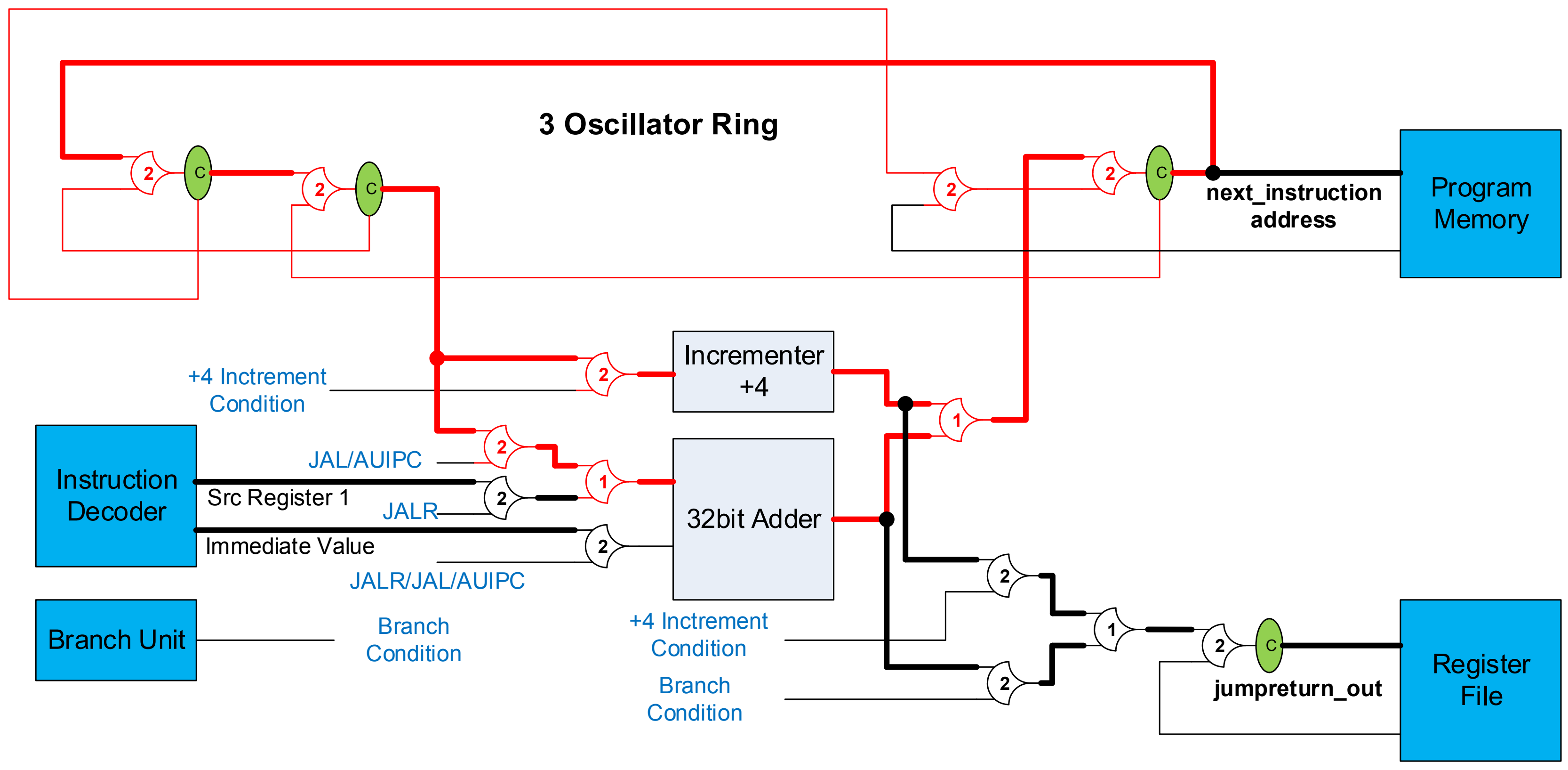
Processor Unit Activity

Unit activity occurs only when needed



This is illustrating the Steering.

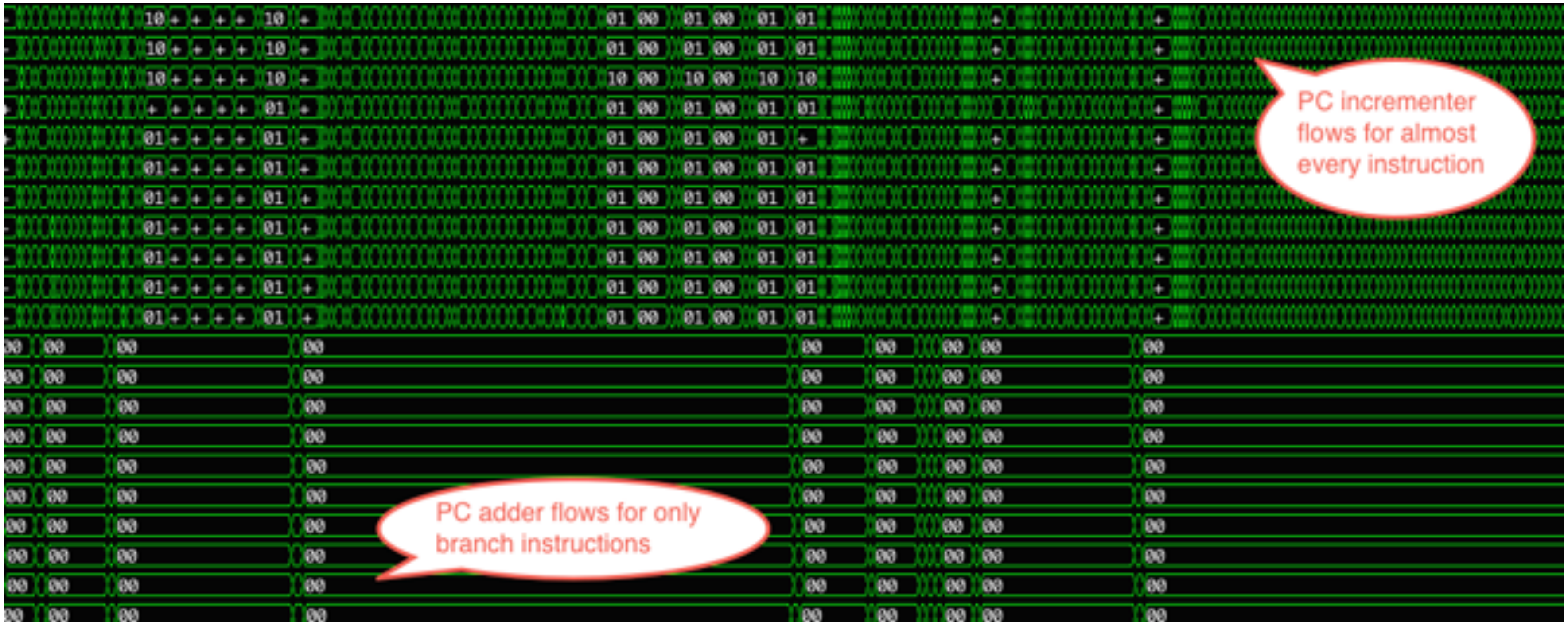
Program Counter Circuit Diagram with 3 Oscillator Ring



PC Circuit Diagram with 3-Oscillator Ring (Red Colour)

PC Incrementer and Adder activity

The incrementer is running almost every instructions.


PC incrementer flows for almost every instruction
PC adder flows for only branch instructions

PC Adder(32 + 32) is active only when needed

RISV-V ISA Instruction Suggestions for a Logically Determined CPU

Instruction Grouping

- RISC-V RV32I Instruction Grouping

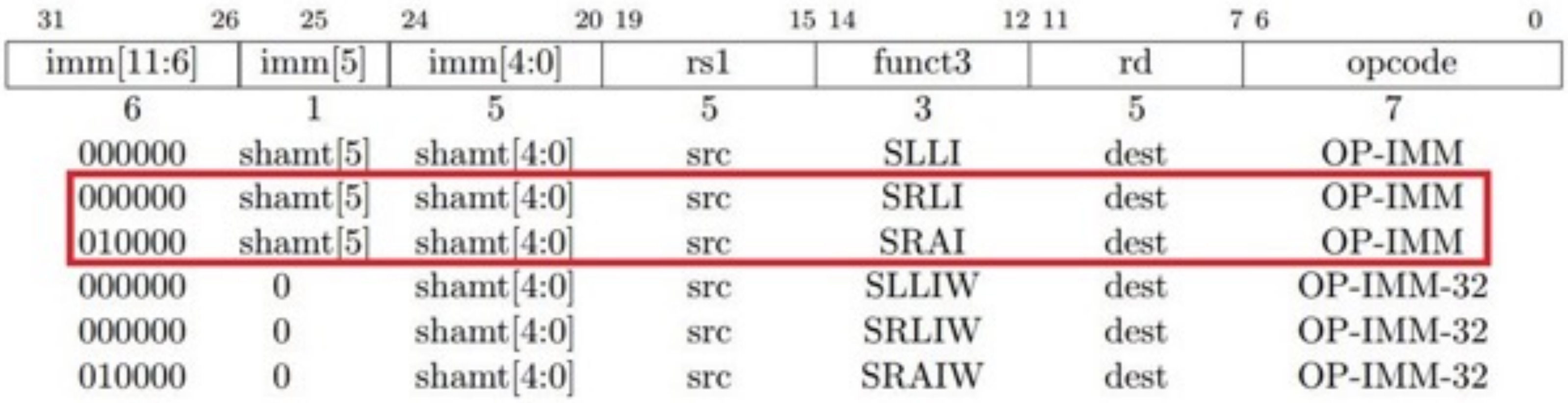
Groups	Instructions
ALU Immediate Group	ADDI, SLTI, SLTIU, XORI, ORI, ANDI, SLLI, SRLI
ALU Group	ADD, SUB, SLL, SLT, SLTU, XOR, SRL, SRA, OR, AND
Sysop Group A	FENCE, FENCE.I, SCALL, SBREAK
Sysop Group B	RDCYCLE, RDCYCLEH, RDTIME, RDTIMEH, RDINSTRET, RDINSTRETH
Branch Group	BEQ, BNE, BLT, BGE, BLTU, BGEU
Load Immediate Group	LUI
Load Group	LB, LH, LW, LBU, LHU
Store Group	SB. SH, SW
Jump Group A	JALR
Jump Group B	JAL, AUIPC

Instruction Decoder design is entirely related to the ISA therefore to implement the high energy efficient and high performance CPU, the instruction codes can be grouped more efficiently to generate the control signals with simpler decoding logic.
All the control signals in the CPU are generated from this table.

Ideal Instruction codes for a Logically Determined Design

- NCL operators have maximum 4-input variables such as TH44 therefore 4bit variables are ideal for NCL decoder to increase the performance.
  ”4 variables for grouping(max 16 groups) and 4 variables for Instructions(max 16 instructions for each group) are good for NCL”

- For example: ALU Immediate Group
  - SRLI, SRAI instructions, their opcode and function3[14:12] are the same but only different imm[11:6] part
  - To distinguish these two commands, we have to decode 16bit variables. opcode[6:0], function3[14:12], imm[11:6]
  - It is not ideal for NCL circuit and causes more area and delay time.
  - This bit allocation will be changed to 4bit variables for an NCL.



31 26	25	24 20	19 15	14 12	11 7	6 0
imm[11:6]	imm[5]	imm[4:0]	rs1	funct3	rd	opcode
6	1	5	5	3	5	7
000000	shamt[5]	shamt[4:0]	src	SLLI	dest	OP-IMM
000000	shamt[5]	shamt[4:0]	src	SRLI	dest	OP-IMM
010000	shamt[5]	shamt[4:0]	src	SRAI	dest	OP-IMM
000000	0	shamt[4:0]	src	SLLIW	dest	OP-IMM-32
000000	0	shamt[4:0]	src	SRLIW	dest	OP-IMM-32
010000	0	shamt[4:0]	src	SRAIW	dest	OP-IMM-32

Where We Are at This Very Early Stage

- Executing compiled **quicksort** at approximately 400 mips (without serious optimisation)

Where We Are Going?

- Optimisation
- Supervisory Instructions
- Place and Route
- Floating Point
- Branch prediction

Conclusion

- We presented the Logically Determined CPU architecture and emulation analysis results based on RISC-V Instruction Set
- We explained the Logically Determined Design (NCL) theory
- We also suggested the Instruction Grouping and Ideal Instruction Codes for an Clockless CPU
- The details also discussed at the Poster Session with Demos