Group V 

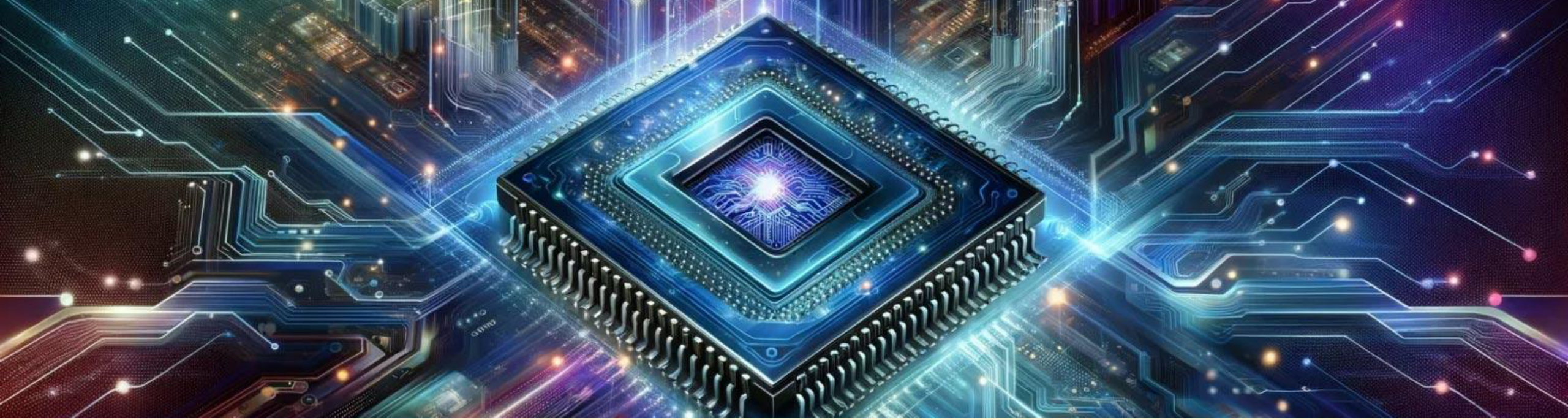
Ethan Liu

Emma Letscher

Wenzi Qian

Agnay Srivastava

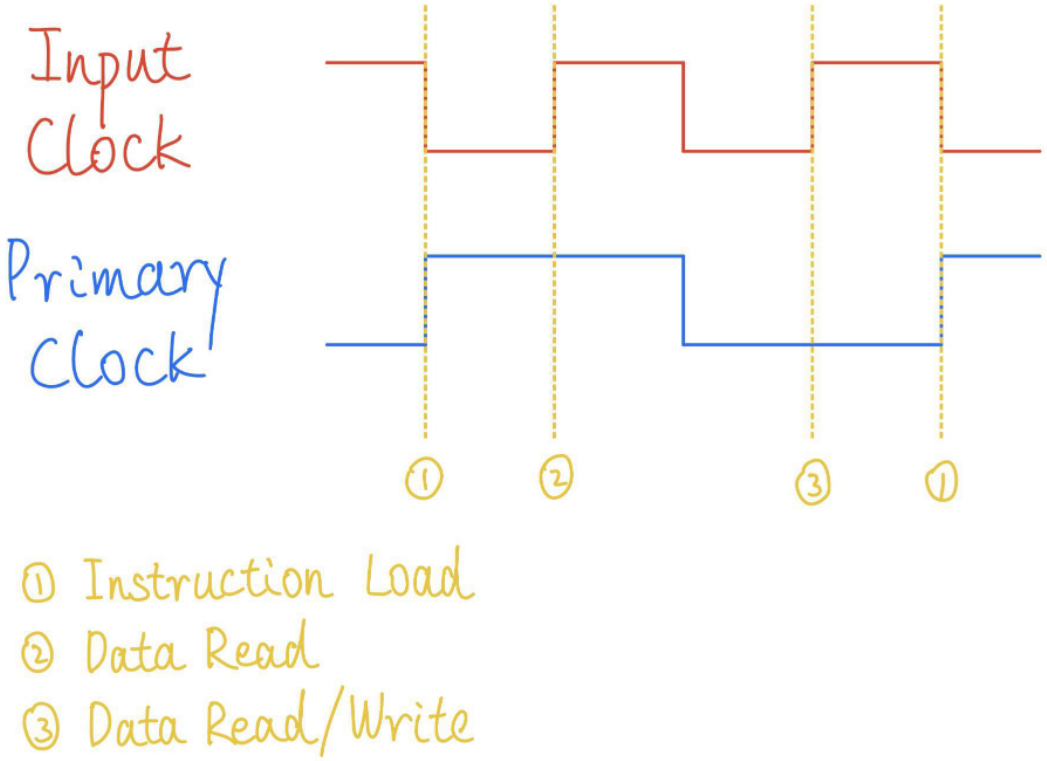
-Cite: Agnay’s dad’s presentation guy



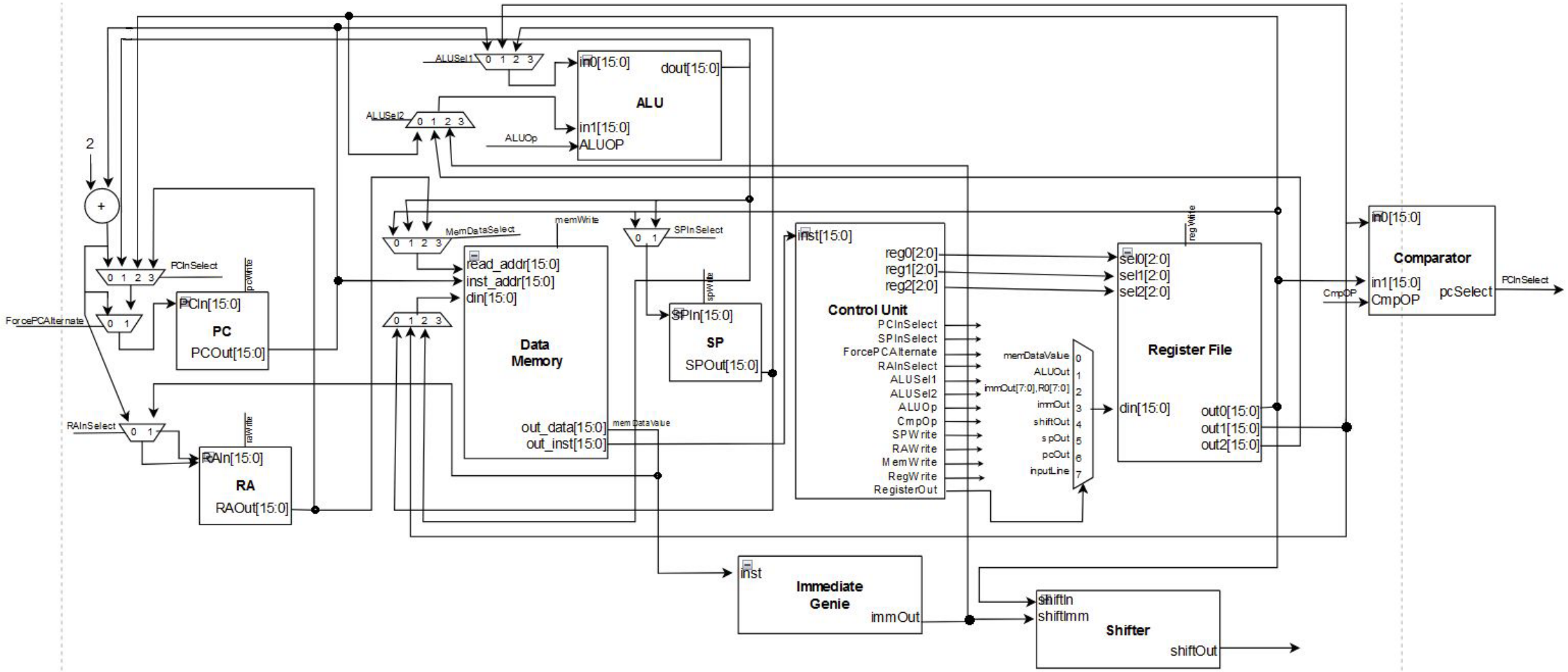
Architecture

Architecture

- Load/store with 8 general-purpose (file) registers and dedicated SP/PC/RA registers
  - You cannot perform arithmetic on SP/PC; their values are only accessible via dedicated commands
- 16-bit input and output line
- ALU + Comparator unit + Bit Shifter
- Psuedo-Multicycle Architecture



Architecture: Datapath



Architecture: Register File

Register	Alias	Usage	Saver
x0	a0	Function Argument, Temporary, Return Value	Caller
x1	a1	Function Argument, Temporary, Return Value	Caller
x2	a2	Function Argument, Temporary, Return Value	Caller
x3	a3	Function Argument, Temporary, Return Value	Caller
x4	a4	Function Argument, Temporary, Return Value	Caller
x5	s0	Saved Value	Callee
x6	s1	Saved Value	Callee
x7	s2	Saved Value	Callee

Instruction Format

Instruction Format - Introduction

Type	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0RR	Opcode			Immediate							Input 2			Input 1		
1R	Opcode			F1	Immediate									Input 1		
1W	Opcode			F1	Immediate									Output		
1WR	Opcode			F1	Immediate						Input 1			Output		
2WR	Opcode			F1	F2	Immediate					Input 1			Output		
2W	Opcode			F1	F2	Immediate								Output		
3	Opcode			F1	F2	F3	Immediate									
4WRR	Opcode			F1	F2	F3	F4	Input 2			Input 1			Output		
7RR	Opcode			F1	F2	F3	F4	F7			Input 2			Input 1		
7WR	Opcode			F1	F2	F3	F4	F7			Input 1			Output		
10R	Opcode			F1	F2	F3	F4	F7			F10			Input 1		
10W	Opcode			F1	F2	F3	F4	F7			F10			Output		

at the front denotes the number of **Func** bits used.

R denotes a **source** register is read from

W denotes a **destination** register is written to

Instruction Format - Machine Code

ADDI x3 x5 -11

Bits	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
Usage	Opcode			F1	Immediate						Input 1			Output		
Value	0	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1
Meaning	-				<SE> Immediate: -11						Input 1: x5			Output: x3		

Instruction Set - RTL breakdown

We have a total of 32 instructions that are grouped into 22 categories based on their single cycle RTL.

Below, we can see the RTL for branching and ALU operations:

Type	RTL
1	`inst = M[PC]` `newPC = PC + 2`
GEQ	`PC = newPC`
NEQ	`a = R[inst[2:0]]`
LT	`b = R[inst[5:3]]`
EQ	`branch = a (cond) b`
	`if (branch) then:`
	`PC = PC + (SE(inst[12:6])<<1)`

Type	RTL
11	`inst = M[PC]` `newPC = PC + 2`
AND	`PC = newPC`
ADD	`a = R[inst[5:3]]`
XOR	`b = R[inst[8:6]]`
OR	`funcCode = R[inst[13:9]]`
NOR	`result = a (op) b`
SUB	`R[inst[2:0]] = result`
NAND	
XNOR	

Instruction Set - Pseudoinstructions

We have three handy pseudo-instructions

LI : Load Immediate that converts to LU and LL (or just LL) based on the size of the immediate

STOP : Stops PC from incrementing, performs no other action

PROCEED : Increments PC, performs no other action

Instruction Set - Optimization

- Procedure calling is optimized to convention
  - Implication: Callers must reserve location 0 on stack

RISC-V	SW RA, 0(SP)
	JAL RA, 30(PC)
		LW RA, 0(SP)
		ADDI SP, SP, 10
		JALR x0, 0(RA)

Ours	JUMP 15	RETURN 5

- Immediate loading is done in two parts, the lower half of which is sign extended.
- Useful for loading small immediates in 1 cycle.

Instruction Set - Optimization

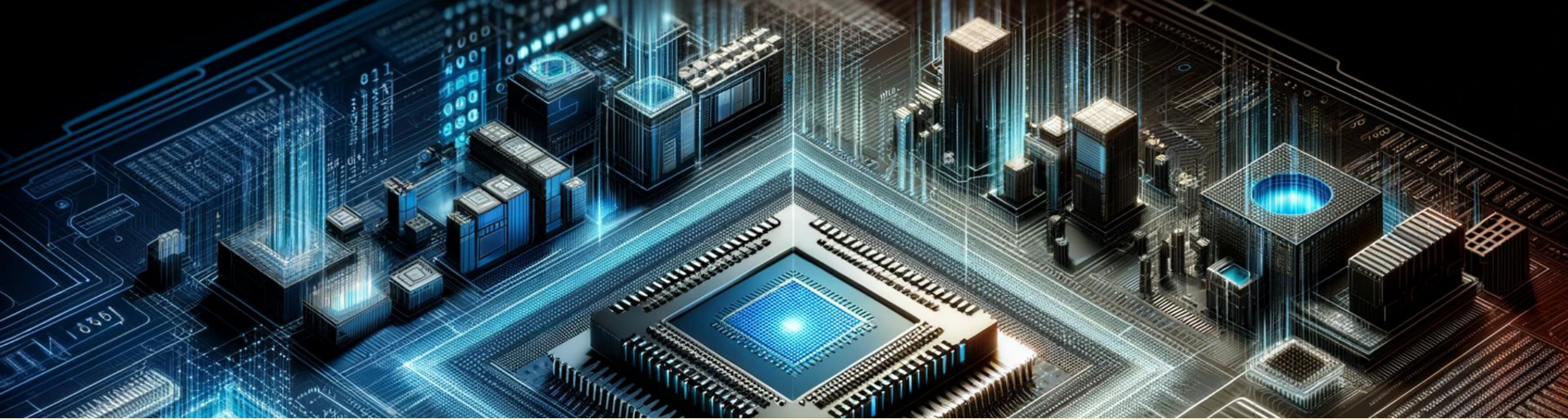
The calling convention optimization, as seen in relprime:

Initial Program	M-Finder	RelPrime
read a0 **jump 3** write a0 stop	**chgspi -3** strsp s0 1 strsp s1 2 addi s0 a0 0 li s1 2 li a3 1 addi a0 s0 0 addi a1 s1 0 **jump 8** eq a0 a3 3 addi s1 s1 1 chgpci -5 addi a0 s1 0 rtvsp s0 1 rtvsp s1 2 **return 3**	li a2 0 neq a2 a0 3 addi a0 a1 0 return 0 eq a1 a2 6 geq a1 a0 3 sub a0 a0 a1 chgpci -3 sub a1 a1 a0 chgpci -5 **return 0**

Design Challenges

- **Large instruction set**
  - Difficult to balance usage between opcode bits, immediate bits, and vacancies for future expansion
  - Difficult to delegate instruction types

- **Datapath design**
  - An output for each place where input is relevant
  - Lots of control signals
  - Lots of muxes



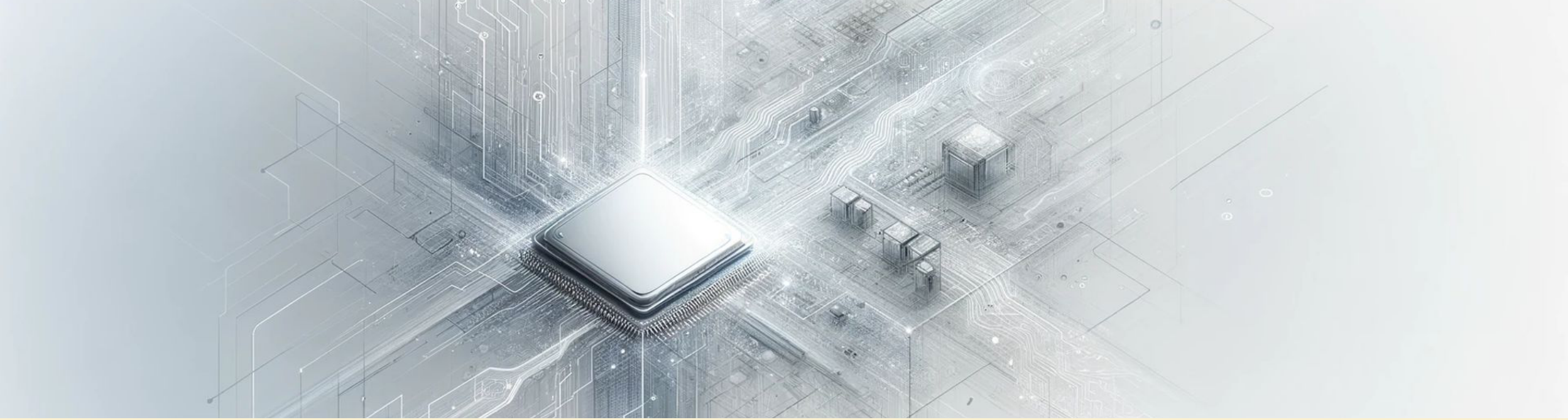
Sample Programs

Example Program: Iteration

Assembly	Machine Code	Comments
li a0 7	0001 1000 0011 1000	a0 = x, initialized to 7
li a1 10	0001 1000 0011 1000	a1 = 10
li a2 0	0001 1000 0000 0010	a2 = i, initialized to 0
geq a2 a1 4	1110 0001 0000 1010	if i >= 10, we are done, jump forward 4 instructions (to rest of program not shown)
addi a0 a0 2	0010 0000 1000 0000	x += 2
addi a2 a2 1	0010 0000 0101 0010	i++
chgpci -3	0111 1111 1111 1101	go to start of loop

```
int x = 7;
for (int i = 0; i<10; i++)
{
x+=2;
}
```

Note : CHGPCI behaves as a “goto” instruction.



Testing

Goal

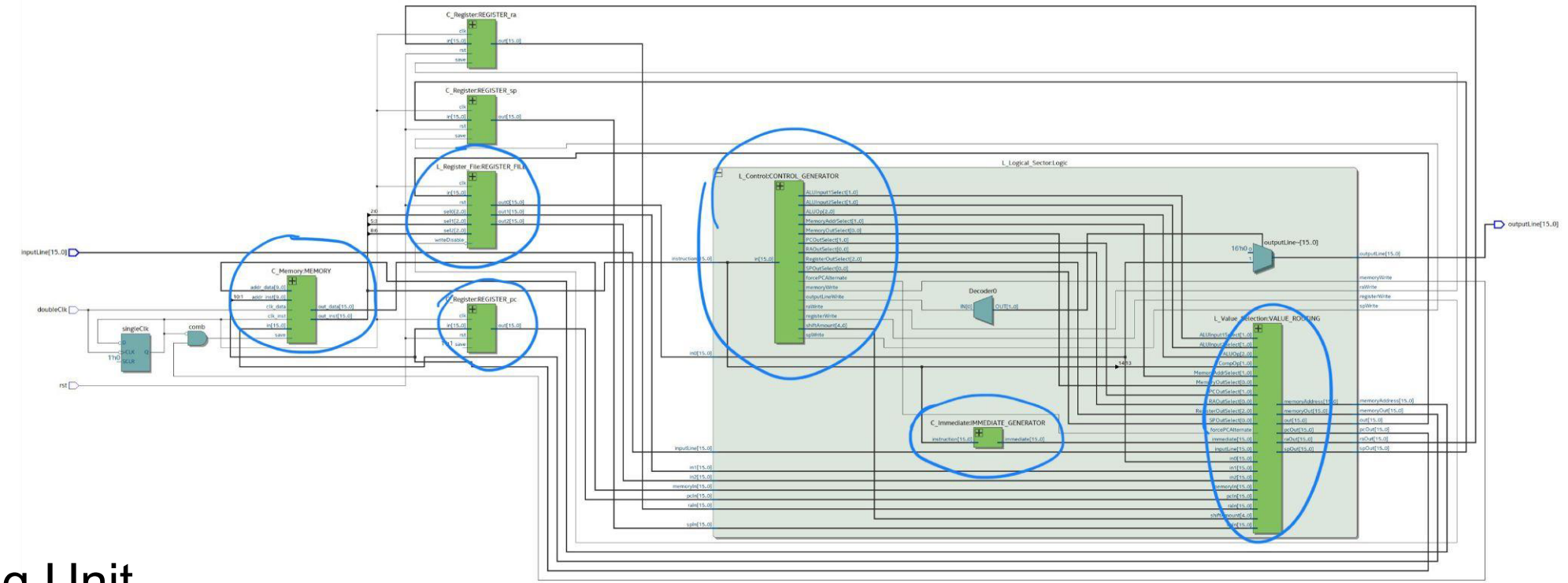
- Automatic test with a single run
- Easy-to-read test results
  - No need to examine waveform
- Balance between time and accuracy

```
VSIM 349> run -all
# [L_Control Flag Test] Testing 'Type 1: Branch' PASSED
#
# [L_Control Flag Test] ERROR, Testing '   Type 2: STRSP', [ MemoryAddrSelect expected: 2 actual: 1], Time:              1000
# [L_Control Flag Test] ERROR, Testing '   Type 2: STRSP', [ MemoryAddrSelect expected: 2 actual: 1], Time:              1200
# [L_Control Flag Test] Testing 'Type 2: STRSP' FAILED
#
# [L_Control Flag Test] ERROR, Testing '   Type 3: RTVSP', [ MemoryAddrSelect expected: 2 actual: 0], Time:              1400
# [L_Control Flag Test] ERROR, Testing '   Type 3: RTVSP', [ MemoryAddrSelect expected: 2 actual: 0], Time:              1600
# [L_Control Flag Test] Testing 'Type 3: RTVSP' FAILED
#
# [L_Control Flag Test] ERROR, Testing '    Type 4: ADDI', [RegisterOutSelect expected: 1 actual: 0], Time:              1800
# [L_Control Flag Test] ERROR, Testing '    Type 4: ADDI', [RegisterOutSelect expected: 1 actual: 0], Time:              2000
# [L_Control Flag Test] Testing 'Type 4: ADDI' FAILED
#
# [L_Control Flag Test] ERROR, Testing '      Type 5: LL', [RegisterOutSelect expected: 3 actual: 0], Time:              2100
# [L_Control Flag Test] ERROR, Testing '      Type 5: LL', [RegisterOutSelect expected: 3 actual: 0], Time:              2200
# [L_Control Flag Test] Testing 'Type 5: LL' FAILED
#
# [L_Control Flag Test] ERROR, Testing '      Type 6: LU', [RegisterOutSelect expected: 2 actual: 0], Time:              2300
# [L_Control Flag Test] ERROR, Testing '      Type 6: LU', [RegisterOutSelect expected: 2 actual: 0], Time:              2400
# [L_Control Flag Test] Testing 'Type 6: LU' FAILED
#
# [L_Control Flag Test] Testing 'Type 7: SHIFT' PASSED
#
# [L_Control Flag Test] Testing 'Type 8A: CHGSPI' PASSED
#
# [L_Control Flag Test] ERROR, Testing ' Type 8B: CHGPCI', [  ALUInput1Select expected: 0 actual: 1], Time:              3400
# [L_Control Flag Test] ERROR, Testing ' Type 8B: CHGPCI', [      PCOutSelect expected: 1 actual: 0], Time:              3400
# [L_Control Flag Test] ERROR, Testing ' Type 8B: CHGPCI', [  ALUInput1Select expected: 0 actual: 1], Time:              3600
# [L_Control Flag Test] ERROR, Testing ' Type 8B: CHGPCI', [      PCOutSelect expected: 1 actual: 0], Time:              3600
# [L_Control Flag Test] Testing 'Type 8B: CHGPCI' FAILED
#
# [L_Control Flag Test] Testing 'Type 9: JUMP' PASSED
#
# [L_Control Flag Test] ERROR, Testing ' Type 10: RETURN', [      memoryWrite expected: 1 actual: 0], Time:              4200
# [L_Control Flag Test] ERROR, Testing ' Type 10: RETURN', [      memoryWrite expected: 1 actual: 0], Time:              4400
# [L_Control Flag Test] Testing 'Type 10: RETURN' FAILED
#
```

Testing - Unit Testing

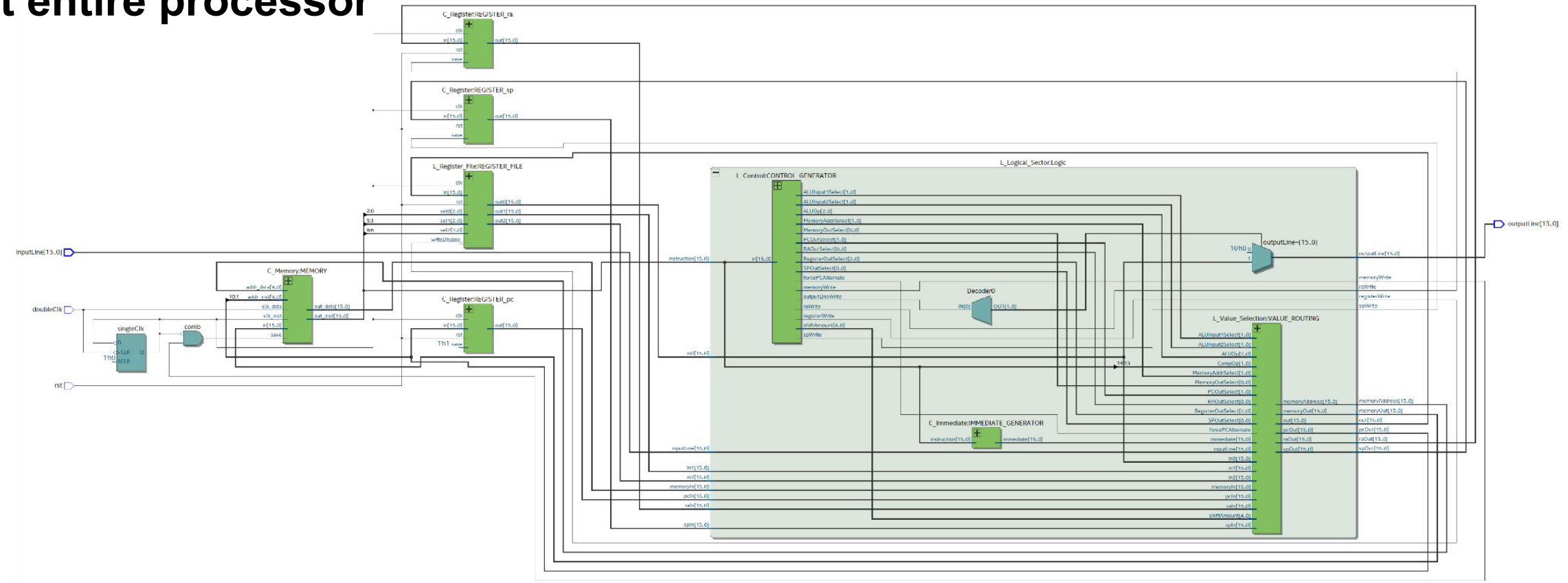
Components unit tested:

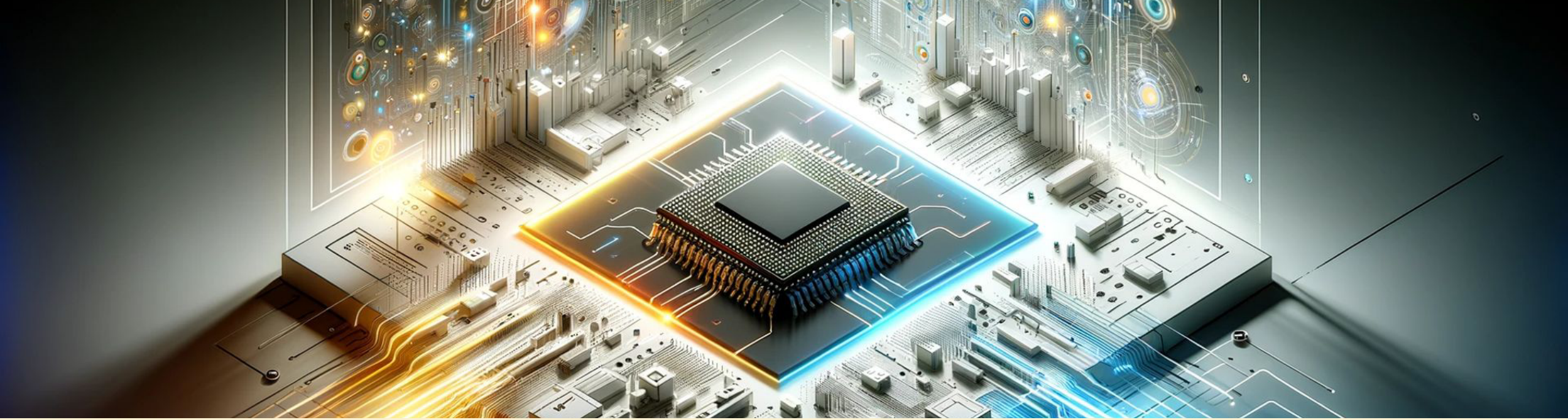
- ALU
- Comparator
- Memory
- Register
- Register File
- Control Unit
- Value Routing Unit



Testing - Integration Testing

1. Test Logic Sector in isolation
2. Test entire processor





Performance

Performance - Some stats

- Cycles to finish relprime on input 5040: 40864
- Maximum Clock Speed synthesized under balanced optimization: 75.25MHz
  - Total Logic Units: 903
  - Total Registers: 177
  - Total Memory Bits: 16384
  - **Time to finish relprime: 0.5430 ms**
- Maximum Clock Speed synthesized under “make it go over 9000” (max speed optimization): 80.42MHz (+6.9%)
  - Total Logic Units: 956 (+5.9%)
  - Total Registers: 206 (+16.4%)
  - Total Memory Bits: 16384
  - **Time to finish relprime: 0.5081 ms**

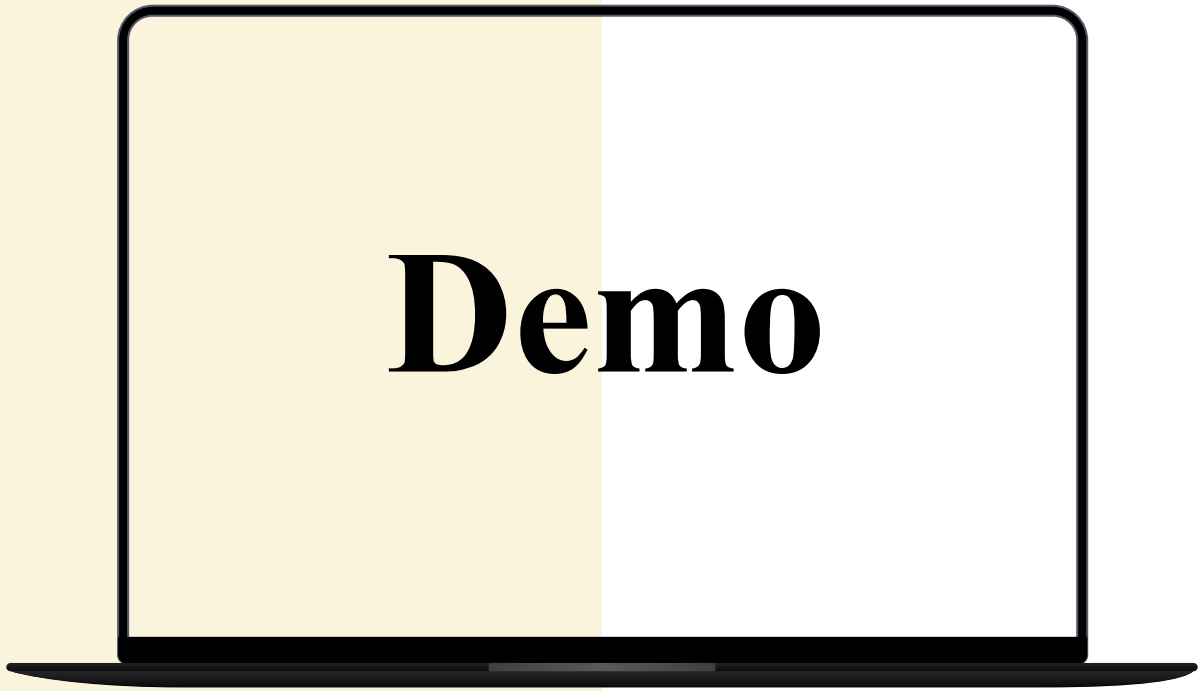
Performance - Our single cycle processor is super fast!

- Dedicated RA, SP, and PC registers, ability to directly modify special registers SP and PC via instructions
  - Maximum number of general purpose registers
  - No need for 4 bit register specifiers
- Hand-optimized instruction set
  - Special instructions to simplify jump and return
- Fixed register read & write specifier locations, minimal read delay

Further Changes

- More enhanced I/O
  - Multiple/wider I/O ports
  - Add signals to notify data ready
- Conversion to multicycle
  - The processor is inherently 2-cycle
- Pivot to pipelined architecture

Demo



Questions