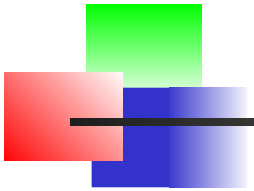


“RISC-V programming & design”

“Rise of RISC-V: The computer chip design you need to know about”

RISC-V: ISA (Instruction Set Architecture)

RISC-V is an open standard instruction set architecture (ISA) based on established reduced instruction set computer (RISC) principles.

Unlike most other ISA designs, RISC-V is provided under royalty-free open-source licenses.

RISC-V ISA is a load–store architecture.

Its floating-point instructions use IEEE 754 floating-point.

Notable features of the RISC-V ISA include:

- instruction bit field locations chosen to simplify the use of multiplexers in a CPU.

A design that is architecturally neutral, and a fixed location for the sign bit of immediate values to speed up sign extension.

RISC-V: ISA (Instruction Set Architecture)

The designers' primary assertion is that the instruction set is the key interface in a computer as it is situated at the interface between the hardware and the software.

If a good instruction set were open and available for use by all, then it can dramatically reduce the cost of software by enabling far more reuse.

The base instruction set has a fixed length of 32-bit naturally aligned instructions, and the ISA supports variable length extensions where each instruction can be any number of 16-bit parcels in length.

Subsets support small embedded systems, personal computers, supercomputers with vector processors.

The instruction set specification defines 32-bit (examples: E902/E906) and 64-bit (examples: C906/C910) address space variants.

RISC-V : Software-Hardware

high level programming

assembly level programming

```
add     a2,a2,a3
mulw    a5,a4,a5
addi    s0,sp,32
jr      ra
```

USER – binaries

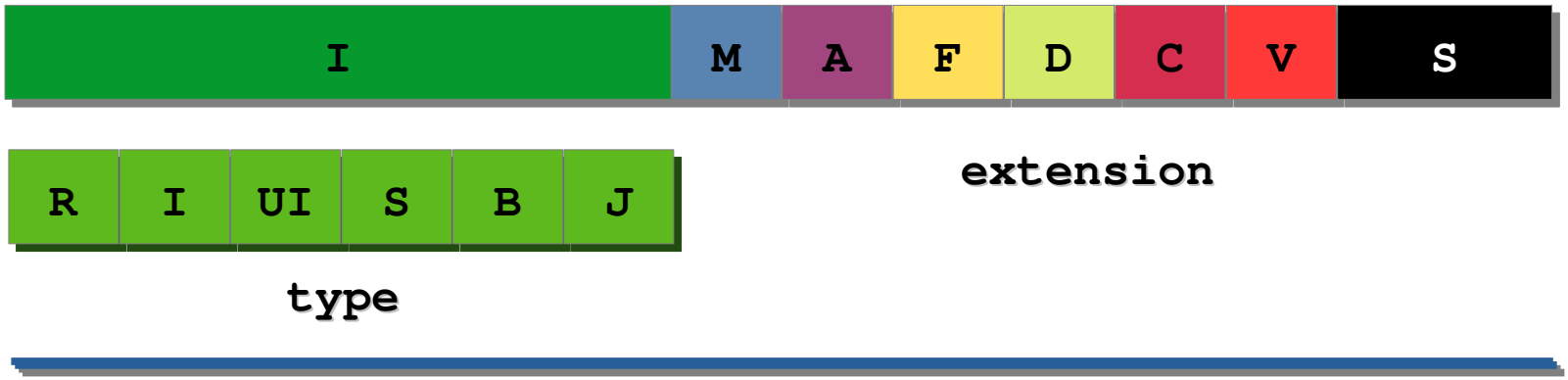
OS

extension

type

design

RISC-V : Software-Hardware



RTL – Verilog : busses,registers,ALUs,memories,decoders,..

Logic – Verilog: and,or,xor,nand,..

FPGA

masks/ASIC

design

RISC-V : Software-Hardware

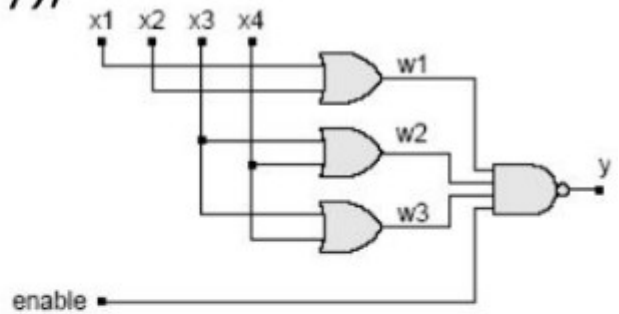
```
module ALU(Out,A,B,S,cin);
    input [3:0] A, B;
    input [2:0] S;
    input cin;
    output [3:0] Out;
    reg [3:0] Out;

    always @(S or A or B or cin)
        case (S)
        0 : Out = 4'b0000;
        1 : Out = B - A - cin;
        2 : Out = A - B - cin;
        3 : Out = A + B + cin;
        4 : Out = A ^ B;
        5 : Out = A | B;
        6 : Out = A & B;
        7 : Out = 4'b1111;
        endcase
endmodule
```

RTL – design

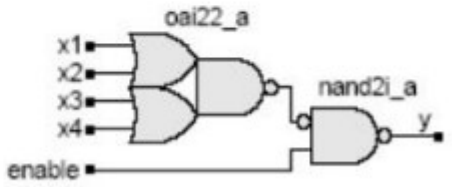
RISC-V : Software-Hardware

Logic design – synthesis

```
module or_nand_1 (enable, x1, x2, x3, x4, y);
  input enable, x1, x2, x3, x4;
  output y;
  wire w1, w2, w3;
  or (w1, x1, x2);
  or (w2, x3, x4);
  or (w3, x3, x4); // redundant
  nand (y, w1, w2, w3, enable);
endmodule
```



Pre-synthesis



Post-synthesis

Programming with Lichee RV Dock

Lichee RV Dock is a **RISC-V Linux development kit**. **C906**

- an **HDMI port** with support for up to 4K@30fps output, a 40-pin header with **GPIO** - speakers, microphones, and more.

- **RGB and MIPI screen** interfaces

- an onboard **2.4G WIFI+BT module**, a 2.4G Patch antenna, an IPEX connector, and a **USB Type-A** host.

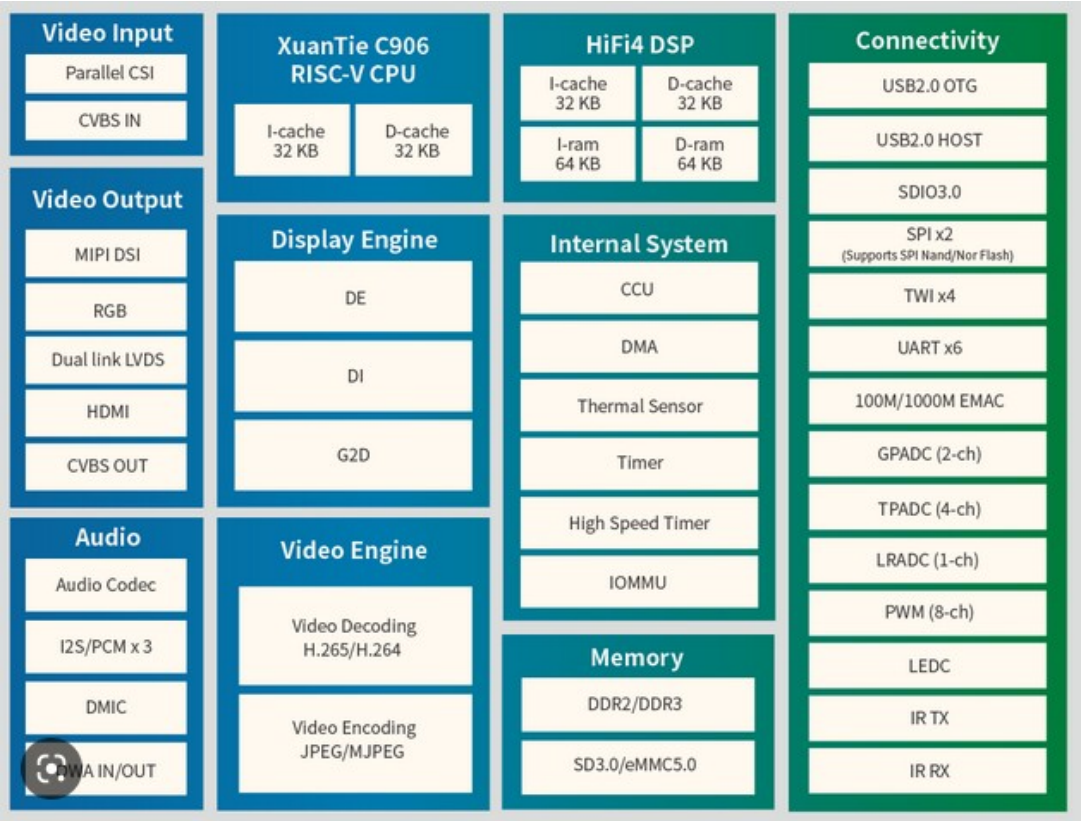
Allwinner – D1 SoC with C906

XuanTie C906 with the RISC-V Vector Extension (**V0.7**)

XuanTie C906 is a **64-bit processor** based on a 64-bit RISC-V architecture.

This processor is designed with a **five to eight stage integer pipeline**. It is also equipped with **128-bit vector operation units**.

Data formats, including **`int8`, `int16`, `int32`, `int64`, `bf16`, `fp16`, `fp32`**, and **`fp64`**, are supported.

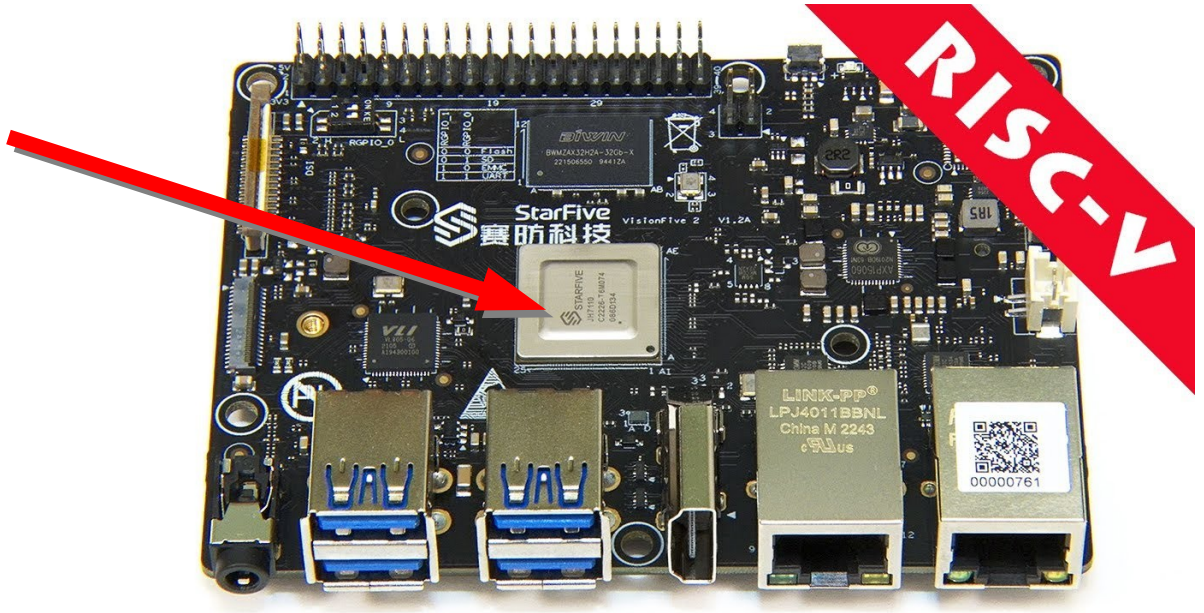


RV64IAMFDCV

Programming with StarFive VisionFive 2

VisionFive 2 is high-performance RISC-V single board computer (SBC) with an integrated GPU.

JH7110 (SiFive)



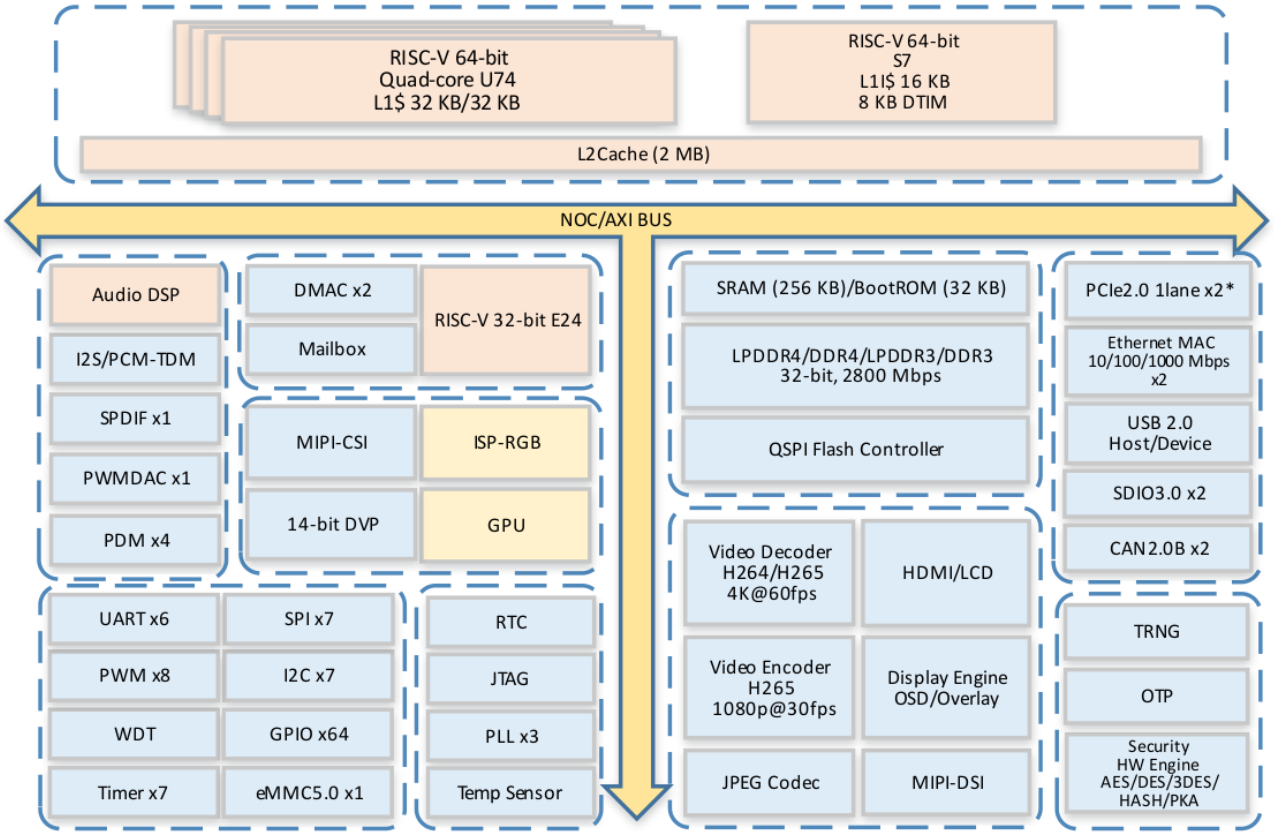
StarFive VisionFive 2 SBC

A strong partnership with **SiFive**, **StarFive** started as an exclusive distributor of **SiFive** RISC-V core IP products in the greater China region

JH7110 (SiFive)

(2023) SiFive and StarFive

End of cooperation !



ISA - instructions/mnemonics

Save or Shift:

`sw,sh,sb :`

`sll,srl,..`

Load :

`lw,lh,lb,..`

Jump :

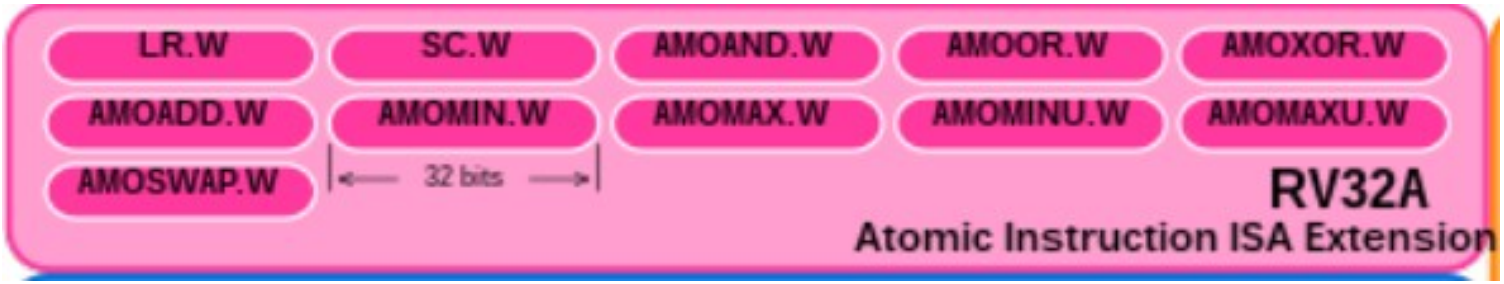
`jal,jalr,..`

Branch :

`beq,blt, bltu, ..`

47-basic instructions

ISA - instructions/mnemonics



Load and Write

`lr.w,..`

Multiply and Divide :

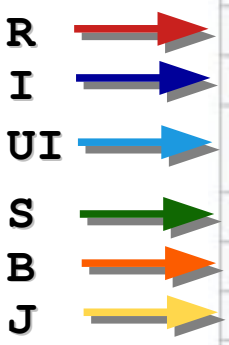
`mul,div,rem, ..`

ISA - instructions/mnemonics

Compressed (16-bit) :

`c.lw, c.sw, c.add, ..`

RISC-V ISA - formats



32-bit RISC-V Instruction Formats

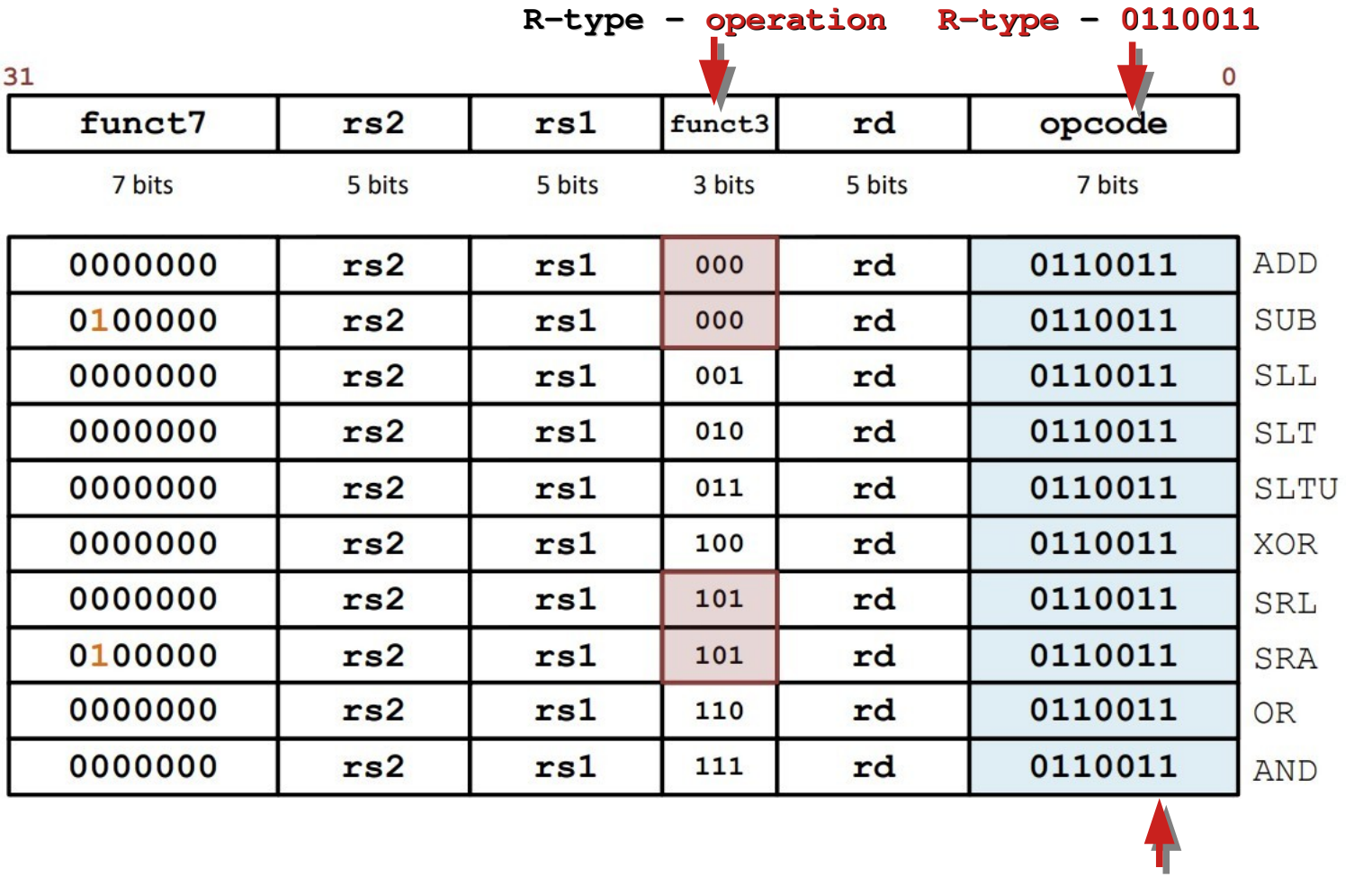
	Instruction Formats	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Register/register	funct7							rs2					rs1					funct3			rd						opcode						
I	Immediate	imm[11:0]												rs1					funct3			rd						opcode						
UI	Upper Immediate	imm[31:12]																				rd						opcode						
S	Store	imm[11:5]							rs2					rs1					funct3			imm[4:0]						opcode						
B	Branch	[12]	imm[10:5]						rs2					rs1					funct3			imm[4:1]				[11]		opcode						
J	Jump	[20]	imm[10:1]										[11]	imm[19:12]								rd						opcode						

- **opcode (7 bit):** partially specifies which of the 6 types of *instruction formats*
- **funct7 + funct3 (10 bit):** combined with **opcode**, these two fields describe what operation to perform
- **rs1 (5 bit):** specifies register containing first operand
- **rs2 (5 bit):** specifies second register operand
- **rd (5 bit):**: Destination register specifies register which will receive result of computation

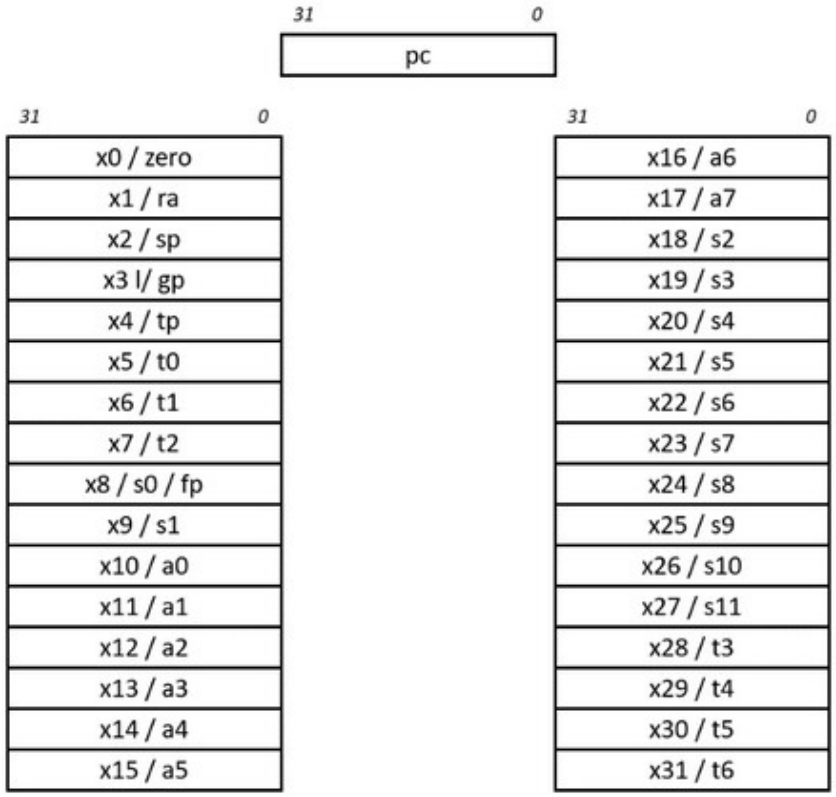
ISA *computational instructions* use a three-operand format, in which the first operand is the destination register, the second operand is a source register, and the third operand is either a second source register or an immediate value. This is an example three-operand instruction:

```
add x1,x2,x3
```

RISC-V ISA - opcodes/functions



RISC-V ISA - registers (RV32I)



ra: Function return address.
sp: Stack pointer.
gp: Global data pointer.
tp: Thread-local data pointer.
t0–t6: Temporary storage.
fp: Frame pointer for function-local stack data (this usage is optional).
s0–s11: Saved registers (if the frame pointer is not in use, **x8** becomes **s0**).
a0–a7: Arguments passed to functions. Any additional arguments are passed on the stack. Function return values are passed in **a0** and **a1**.

RISC-V : assembly programming

```
// HelloWorld.c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
  printf("HelloWorld\n");
  exit(0);
}
```

2 system calls

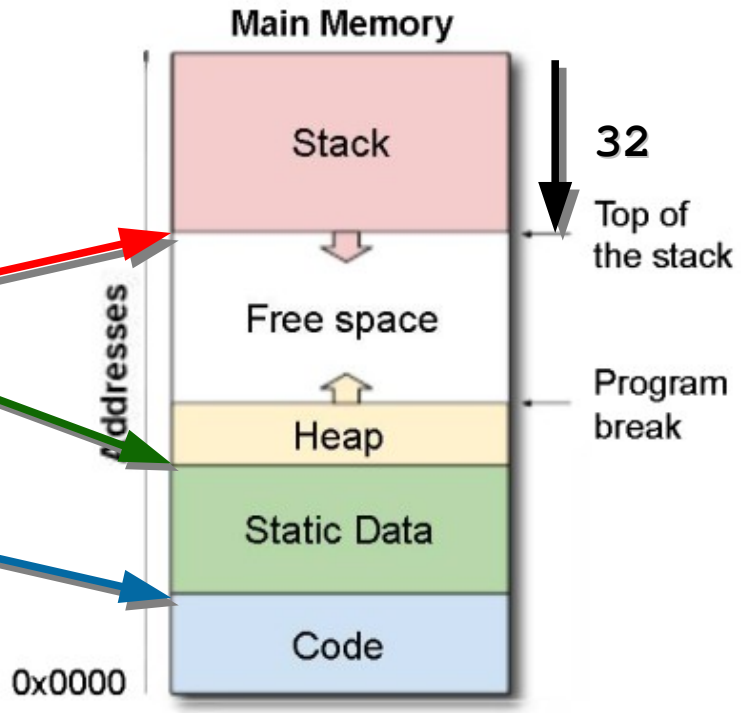
```
gcc HelloWorld.c -o HelloWorld
```

```
gcc HelloWorld.c -c -S
```

```
gcc HelloWorld.c -c -S -fverbose-asm
```

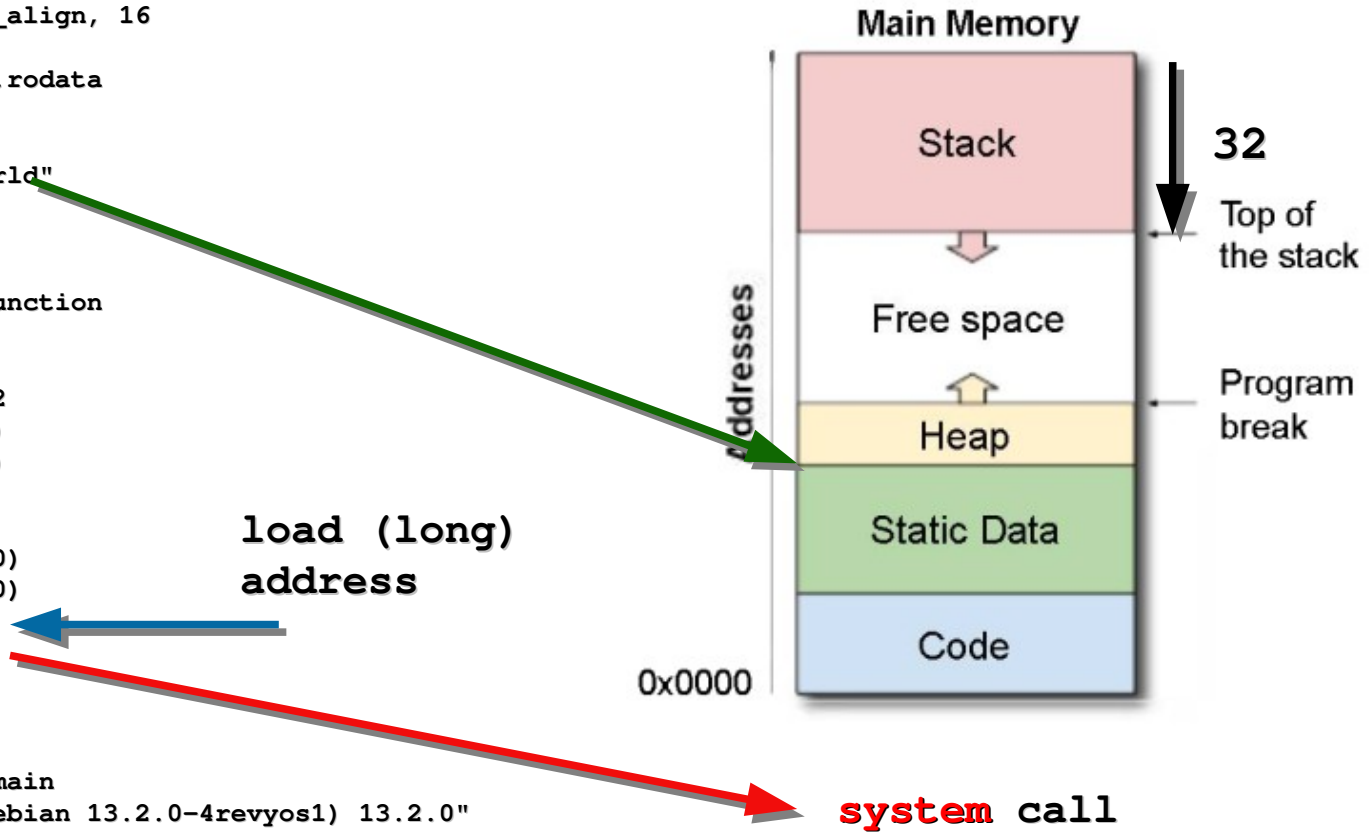
RISC-V : assembly programming

```
        .file   "HelloWorldExit.c"
        .option pic
        .attribute arch, "rv64i2p1_m2p0_a2p1_f2p2_d2p2_c2p0_zicsr2p0_zifencei2p0"
        .attribute unaligned_access, 0
        .attribute stack_align, 16
        .text
        .section        .rodata
        .align  3
.LC0:
        .string "HelloWorld"
        .text
        .align  1
        .globl  main
        .type   main, @function
main:
.LFB6:
        addi    sp,sp,-32
        sd      ra,24(sp)
        sd      s0,16(sp)
        addi    s0,sp,32
        mv      a5,a0
        sd      a1,-32(s0)
        sw      a5,-20(s0)
        lla     a0,.LC0
        call    puts@plt
        li      a0,0
        call    exit@plt
.LFE6:
        .size   main, .-main
        .ident  "GCC: (Debian 13.2.0-4revyos1) 13.2.0"
        .section        .note.GNU-stack,"",@progbits
```



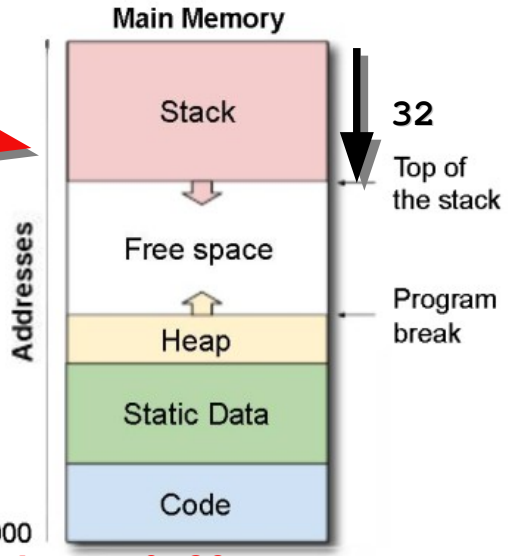
RISC-V : assembly programming

```
        .file   "HelloWorldExit.c"
        .option pic
        .attribute arch, "rv64i2p1_m2p0_a2p1_f2p2_d2p2_c2p0_zicsr2p0_zifencei2p0"
        .attribute unaligned_access, 0
        .attribute stack_align, 16
        .text
        .section        .rodata
        .align  3
.LC0:
        .string "HelloWorld"
        .text
        .align  1
        .globl  main
        .type   main, @function
main:
.LFB6:
        addi    sp,sp,-32
        sd      ra,24(sp)
        sd      s0,16(sp)
        addi    s0,sp,32
        mv      a5,a0
        sd      a1,-32(s0)
        sw      a5,-20(s0)
        lla     a0,.LC0
        call    puts@plt
        li      a0,0
        call    exit@plt
.LFE6:
        .size   main, .-main
        .ident  "GCC: (Debian 13.2.0-4revyos1) 13.2.0"
        .section        .note.GNU-stack,"",@progbits
```



RISC-V : assembly programming

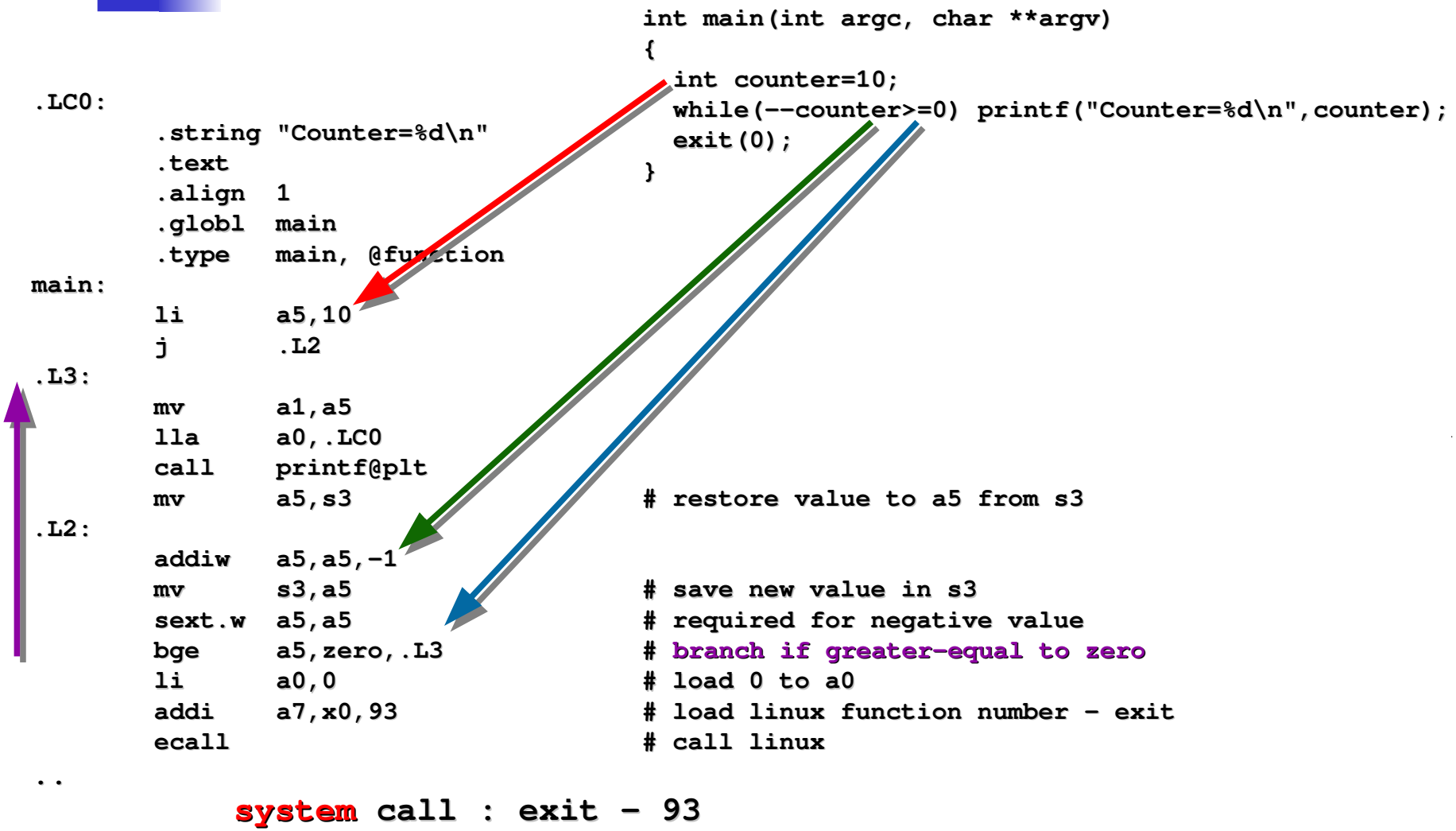
```
int main(int argc, char **argv)
{
  printf("argv[1]=%s\n",argv[1]);
  exit(0);
}
```




```
main:
        addi    sp,sp,-32
        sd      ra,24(sp)
        sd      s0,16(sp)
        addi    s0,sp,32
        mv      a5,a0           # move a0 (argc) to a5
        sd      a1,-32(s0)      # a1 (argv address) is stored at s0-32
        sw      a5,-20(s0)      # a5 (a0) is stored at s0-20
        ld      a5,-32(s0)      # a5 receives a1 - argv address
        addi    a5,a5,8         # +8, we are going to read argv[1] not argv[0]
        ld      a5,0(a5)        # load address of argv[1] to a5
        mv      a1,a5           # mv a5 to a1 (for printf)
        lla     a0,.LC0         # load address of string with conversion
        call    printf@plt      # call printf
..
```

system call

RISC-V : counting – no stack



```
                                    int main(int argc, char **argv)
                                    {
                                      int counter=10;
.LC0:                                 while(--counter>=0) printf("Counter=%d\n",counter);
        .string "Counter=%d\n"        exit(0);
        .text                       }
        .align  1
        .globl  main
        .type   main, @function
main:
        li      a5,10
        j       .L2
.L3:
        mv      a1,a5
        lla     a0,.LC0
        call    printf@plt
        mv      a5,s3               # restore value to a5 from s3
.L2:
        addiw   a5,a5,-1
        mv      s3,a5               # save new value in s3
        sext.w  a5,a5               # required for negative value
        bge     a5,zero,.L3         # branch if greater-equal to zero
        li      a0,0                # load 0 to a0
        addi    a7,x0,93            # load linux function number - exit
        ecall                       # call linux
..
```

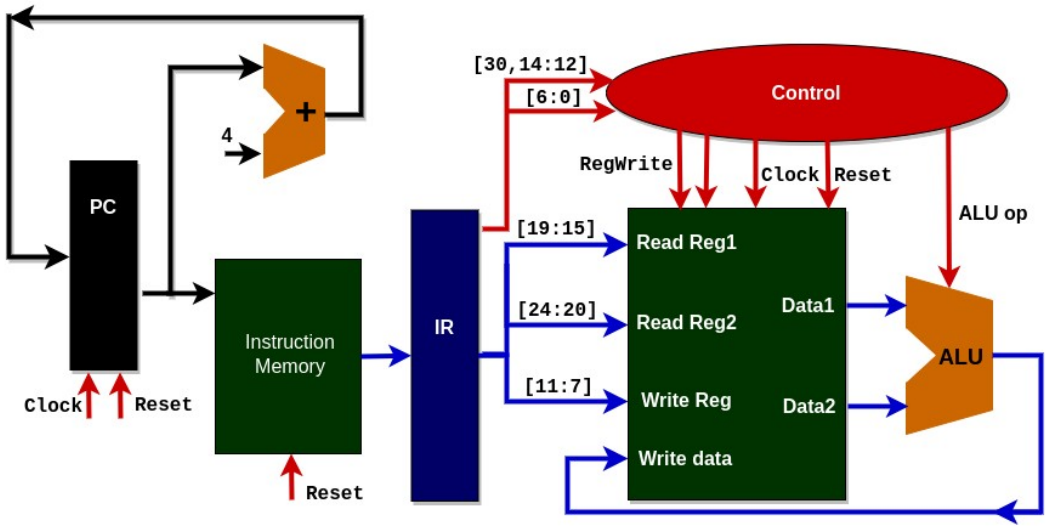
system call : exit – 93

RISC-V : design (Verilog)

extension

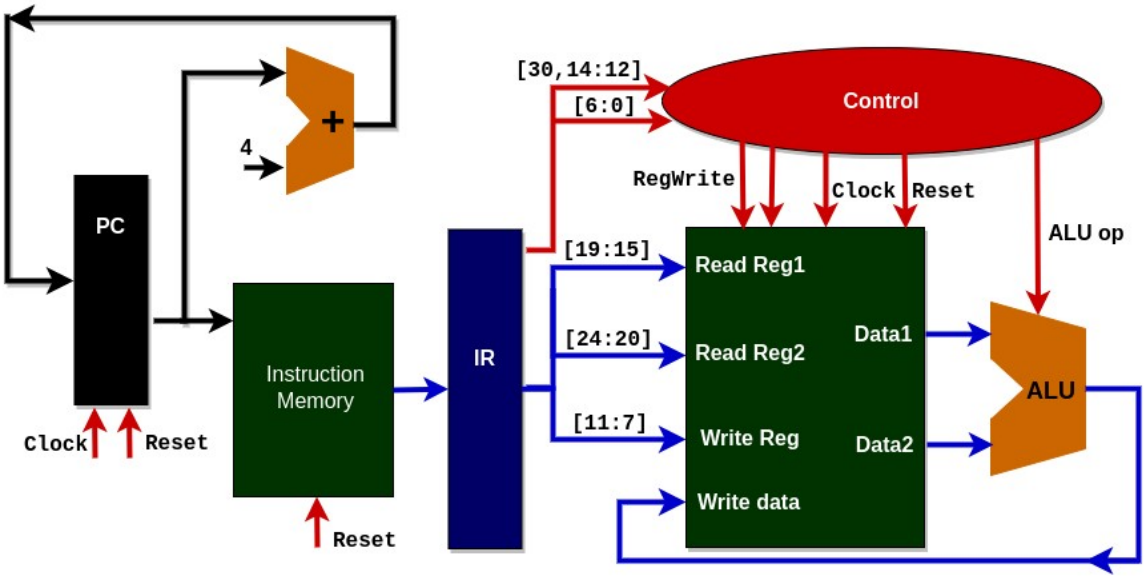
type

design



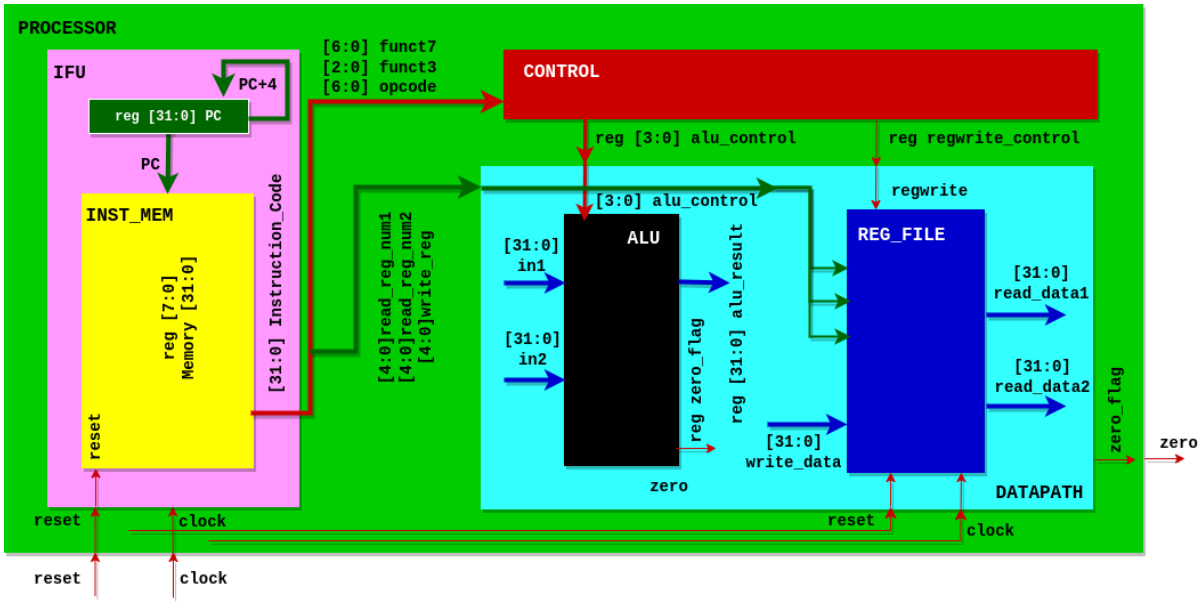

```
add     a2,a2,a3
```

RISC-V : Software-Hardware




```
`include "CONTROL.v"
`include "DATAPATH.v"
`include "IFU.v"

module PROCESSOR(
    input clock,
    input reset,
    output zero
);

wire [31:0] instruction_code;
wire [3:0] alu_control;
wire regwrite;

IFU IFU_module(clock, reset, instruction_code);

CONTROL control_module(instruction_code[31:25],instruction_code[14:12],
                       instruction_code[6:0],alu_control, regwrite);

DATAPATH datapath_module(instruction_code[19:15], instruction_code[24:20],
                  instruction_code[11:7], alu_control, regwrite, clock, reset, zero);

endmodule
```

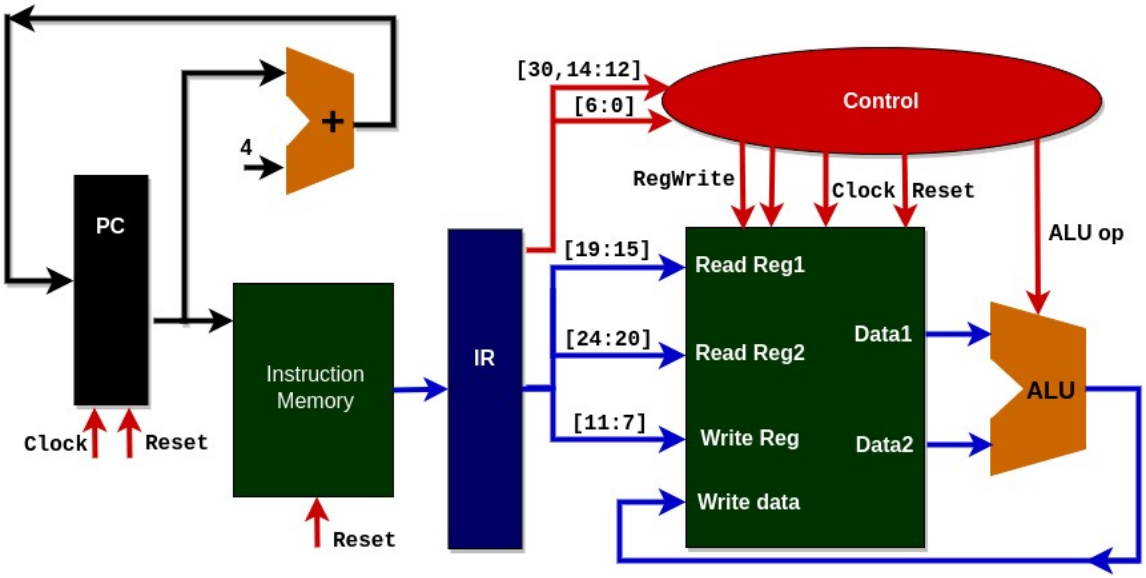
RISC-V : Software-Hardware




```
`include "CONTROL.v"
`include "DATAPATH.v"
`include "IFU.v"

module PROCESSOR(
    input clock,
    input reset,
    output zero
);

wire [31:0] instruction_code;
wire [3:0] alu_control;
wire regwrite;

IFU IFU_module(clock, reset, instruction_code);

CONTROL control_module(instruction_code[31:25],instruction_code[14:12],
                       instruction_code[6:0],alu_control, regwrite);

DATAPATH datapath_module(instruction_code[19:15], instruction_code[24:20],
                  instruction_code[11:7], alu_control, regwrite, clock, reset, zero);

endmodule
```

RISC-V : Software-Hardware

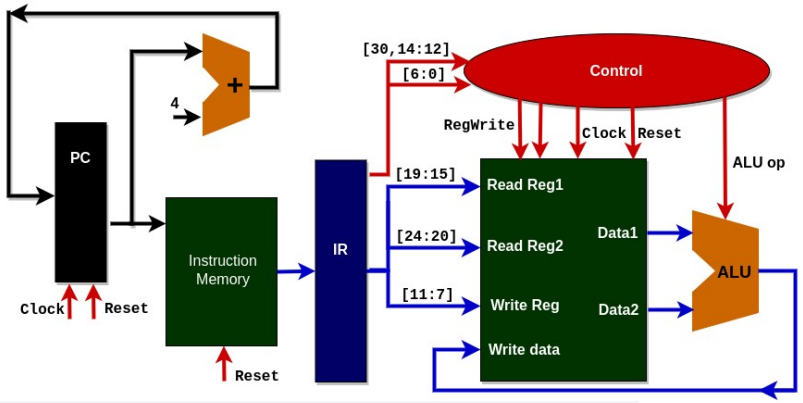
Lab3
with
iverilog
and
gtkwave

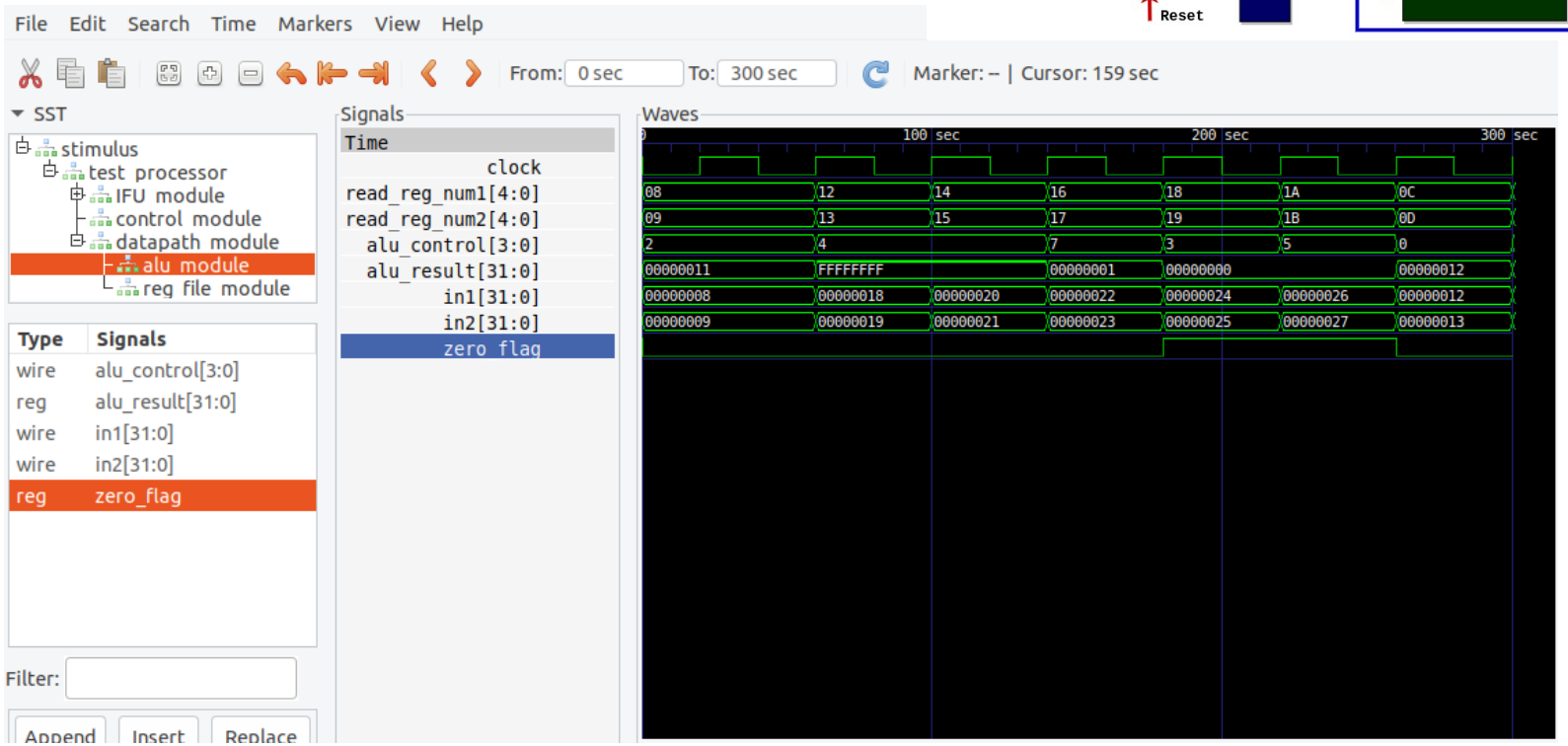



```
bako@lpi4a:~/lab3$iverilog riscv.v riscv_tb.v -o riscv_model

bako@lpi4a:~/lab3$vvp riscv_model

bako@lpi4a:~/lab3$gtkwave riscv_wave.vcd
```

RISC-V : Software-Hardware

GTKWave waveforms





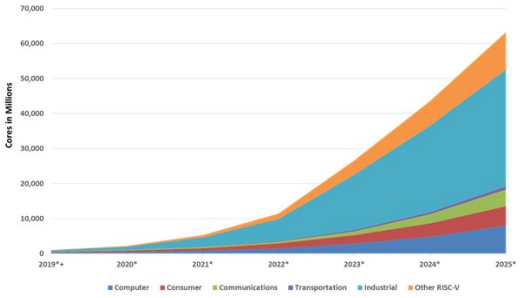
Summary

The rise of RISC-V open architecture exponential growth of cores

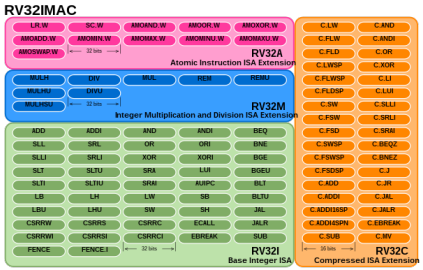
RISC-V open source & business model

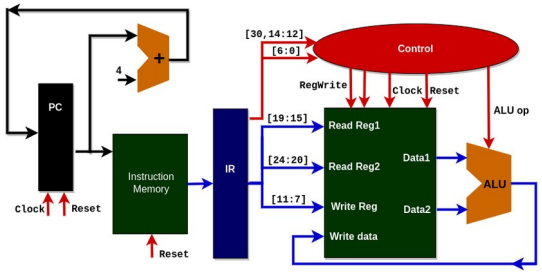
RISC-V optimized & extensible ISA

RISC-V architecture & Verilog synthetisable models



ISA	Chips?	Architecture License?	Commercial Core IP?	Add Own Instructions?	Open-Source Core IP?
x86	Yes, *three* vendors	*No*	*No*	*No*	*No*
ARM	Yes, *many* vendors	Yes, *expensive*	Yes, *one* vendor	*No (Mostly)*	*No*
RISC-V	Yes, *many* vendors	Yes, *free*	Yes, *many* vendors	*Yes*	*Yes, many available*



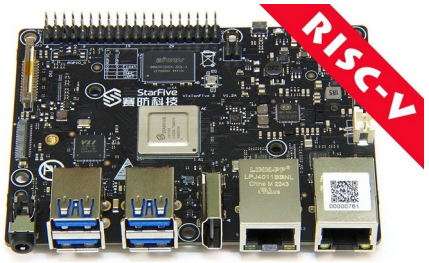


SmartComputerLab

RISC-V labs

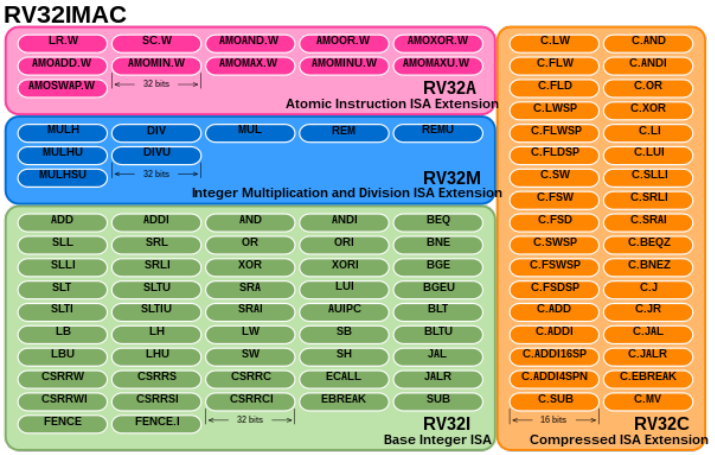
Lab0 (mini-lab)

Starting with the RV64 - Dock (mono-core)



Lab1 & Lab2

RISC-V ISA and assembly programming on StarFive RV64 quad-core



Lab3

RISC-V design: architecture & simple Verilog model

```
add     a2,a2,a3
mulw    a5,a4,a5
addi    s0,sp,32
jr      ra
```

```
case(alu_control)
4'b0000: alu_result = in1&in2;
4'b0001: alu_result = in1|in2;
4'b0010: alu_result = in1+in2;
4'b0100: alu_result = in1-in2;
```