

# EE3220 System-on-Chip Design

## Tutorial 10

### An Introduction to RISC-V

# Objective

In this tutorial, we will learn the basic information, examples and future development of RISC-V, to explain the difference and trend of this architecture. The objectives are shown as following:

- Review Instruction Set Architecture (ISA)
- Learn basic instructions of RISC-V
- Learn one RISC-V example in CityU Architecture Lab for Arithmetic and Security (CALAS, founded by Prof. Ray)
- Learn the development and trend of RISC-V
- Consider what you can use by RISC-V

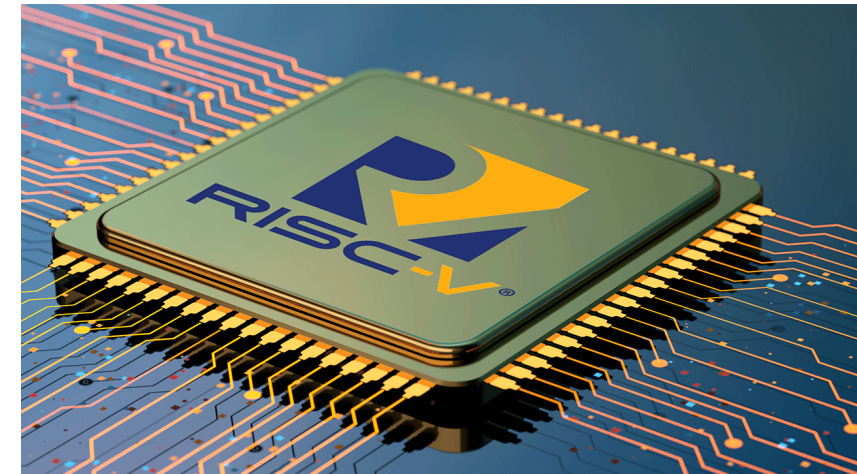
# Review

- In computer science, an instruction set architecture (ISA), also called computer architecture, is an abstract model of a computer.
- A device that executes instructions described by that ISA, such as a central processing unit (CPU), or ARM, or other device is called an implementation.

|                                                                                                                        |   |   |   |        |                      |   |   |   |      |     |   |   |               |    |   |   |           |     |   |   |        |    |     |     |                                |          |                            |                               |                                          |  |           |  |  |                    |  |  |  |  |  |  |  |
|------------------------------------------------------------------------------------------------------------------------|---|---|---|--------|----------------------|---|---|---|------|-----|---|---|---------------|----|---|---|-----------|-----|---|---|--------|----|-----|-----|--------------------------------|----------|----------------------------|-------------------------------|------------------------------------------|--|-----------|--|--|--------------------|--|--|--|--|--|--|--|
| 3 3 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 9 8 7 6 5 4 3 2 1 0<br>1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 |   |   |   |        |                      |   |   |   |      |     |   |   |               |    |   |   |           |     |   |   |        |    |     |     |                                |          |                            |                               |                                          |  |           |  |  |                    |  |  |  |  |  |  |  |
| Cond                                                                                                                   | 0 | 0 | 1 | Opcode |                      |   |   | S | Rn   |     |   |   | Rd            |    |   |   | Operand 2 |     |   |   |        |    |     |     | Data Processing / PSR Transfer |          |                            |                               |                                          |  |           |  |  |                    |  |  |  |  |  |  |  |
| Cond                                                                                                                   | 0 | 0 | 0 | 0      | 0                    | 0 | A | S | Rd   |     |   |   | Rn            |    |   |   | Rs        | 1   | 0 | 0 | 1      | Rm |     |     |                                | Multiply |                            |                               |                                          |  |           |  |  |                    |  |  |  |  |  |  |  |
| Cond                                                                                                                   | 0 | 0 | 0 | 0      | 1                    | U | A | S | RdHi |     |   |   | RdLo          |    |   |   | Rn        |     |   |   | 1      | 0  | 0   | 1   | Rm                             |          |                            |                               | Multiply Long                            |  |           |  |  |                    |  |  |  |  |  |  |  |
| Cond                                                                                                                   | 0 | 0 | 0 | 1      | 0                    | B | 0 | 0 | Rn   |     |   |   | Rd            |    |   |   | 0         | 0   | 0 | 0 | 1      | 0  | 0   | 1   | Rm                             |          |                            |                               | Single Data Swap                         |  |           |  |  |                    |  |  |  |  |  |  |  |
| Cond                                                                                                                   | 0 | 0 | 0 | 1      | 0                    | 0 | 1 | 0 | 1    | 1   | 1 | 1 | 1             | 1  | 1 | 1 | 1         | 1   | 1 | 1 | 0      | 0  | 0   | 1   | Rn                             |          |                            |                               | Branch and Exchange                      |  |           |  |  |                    |  |  |  |  |  |  |  |
| Cond                                                                                                                   | 0 | 0 | 0 | P      | U                    | 0 | W | L | Rn   |     |   |   | Rd            |    |   |   | 0         | 0   | 0 | 0 | 1      | S  | H   | 1   | Rm                             |          |                            |                               | Halfword Data Transfer: register offset  |  |           |  |  |                    |  |  |  |  |  |  |  |
| Cond                                                                                                                   | 0 | 0 | 0 | P      | U                    | 1 | W | L | Rn   |     |   |   | Rd            |    |   |   | Offset    |     |   |   | 1      | S  | H   | 1   | Offset                         |          |                            |                               | Halfword Data Transfer: immediate offset |  |           |  |  |                    |  |  |  |  |  |  |  |
| Cond                                                                                                                   | 0 | 1 | 1 | P      | U                    | B | W | L | Rn   |     |   |   | Rd            |    |   |   | Offset    |     |   |   |        |    |     |     | Single Data Transfer           |          |                            |                               |                                          |  |           |  |  |                    |  |  |  |  |  |  |  |
| Cond                                                                                                                   | 0 | 1 | 1 |        |                      |   |   |   |      |     |   |   |               |    |   |   |           |     |   |   |        |    | 1   |     |                                |          |                            |                               |                                          |  | Undefined |  |  |                    |  |  |  |  |  |  |  |
| Cond                                                                                                                   | 1 | 0 | 0 | P      | U                    | S | W | L | Rn   |     |   |   | Register List |    |   |   |           |     |   |   |        |    |     |     |                                |          |                            |                               | Block Data Transfer                      |  |           |  |  |                    |  |  |  |  |  |  |  |
| Cond                                                                                                                   | 1 | 0 | 1 | L      | Offset               |   |   |   |      |     |   |   |               |    |   |   |           |     |   |   |        |    |     |     |                                |          |                            |                               |                                          |  |           |  |  | Branch             |  |  |  |  |  |  |  |
| Cond                                                                                                                   | 1 | 1 | 0 | P      | U                    | N | W | L | Rn   |     |   |   | CRd           |    |   |   | CP#       |     |   |   | Offset |    |     |     |                                |          |                            |                               | Coprocessor Data Transfer                |  |           |  |  |                    |  |  |  |  |  |  |  |
| Cond                                                                                                                   | 1 | 1 | 1 | 0      | CP Opc               |   |   |   | CRn  |     |   |   | CRd           |    |   |   | CP#       |     |   |   | CP     | 0  | CRm |     |                                |          | Coprocessor Data Operation |                               |                                          |  |           |  |  |                    |  |  |  |  |  |  |  |
| Cond                                                                                                                   | 1 | 1 | 1 | 0      | CP Opc               |   |   |   | L    | CRn |   |   |               | Rd |   |   |           | CP# |   |   |        | CP | 1   | CRm |                                |          |                            | Coprocessor Register Transfer |                                          |  |           |  |  |                    |  |  |  |  |  |  |  |
| Cond                                                                                                                   | 1 | 1 | 1 | 1      | Ignored by processor |   |   |   |      |     |   |   |               |    |   |   |           |     |   |   |        |    |     |     |                                |          |                            |                               |                                          |  |           |  |  | Software Interrupt |  |  |  |  |  |  |  |
| 3 3 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 9 8 7 6 5 4 3 2 1 0<br>1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 |   |   |   |        |                      |   |   |   |      |     |   |   |               |    |   |   |           |     |   |   |        |    |     |     |                                |          |                            |                               |                                          |  |           |  |  |                    |  |  |  |  |  |  |  |

# Review

- An ISA defines the supported instructions, data types, registers, the hardware support for managing main memory, fundamental features (such as the memory consistency, addressing modes, virtual memory), and the input/output model of a family of implementations of the ISA.
- An ISA can be extended by adding instructions or other capabilities, or adding support for larger addresses and data values. Such as thumb instruction set.

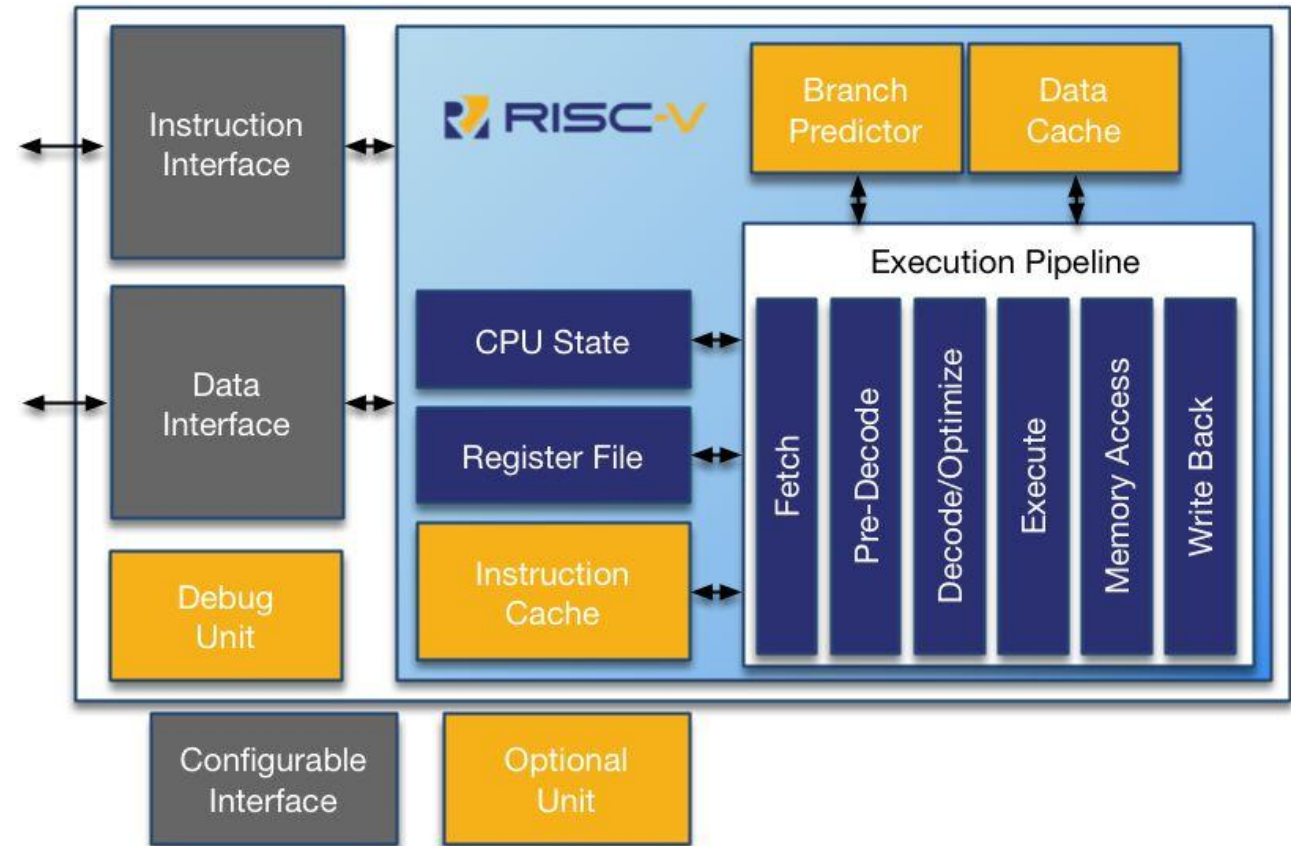


# I. Instruction

- The original (and subsequent) Arm implementation was hardwired without microcode, like the much simpler 8-bit 6502 processor used in prior Acorn microcomputers.
- A **reduced instruction set computer**, or **RISC**, is a computer with a small, highly optimized set of instructions, rather than the more specialized set often found in other types of architecture, such as in a complex instruction set computer (CISC)
- For example, the 32-bit Arm architecture includes RISC features.

# I. Instruction

- RISC-V (pronounced "risk-five") is an open-source implementation of a reduced instruction set computing (RISC) based instruction set architecture (ISA)
- The project was originated in 2010 by researchers in the Computer Science Division at UC Berkeley.



# I. Instruction

- Most ISAs are commercially protected by patents, preventing practical efforts to reproduce the computer systems.
- In contrast, RISC-V is open, permitting any person or group to construct compatible computers and use associated software.
- A completely open ISA that is freely available to academy and industry
- A real ISA suitable for direct native hardware implementation, not just simulation or binary translation
- Support for the revised 2008 IEEE-754 floating-point standard

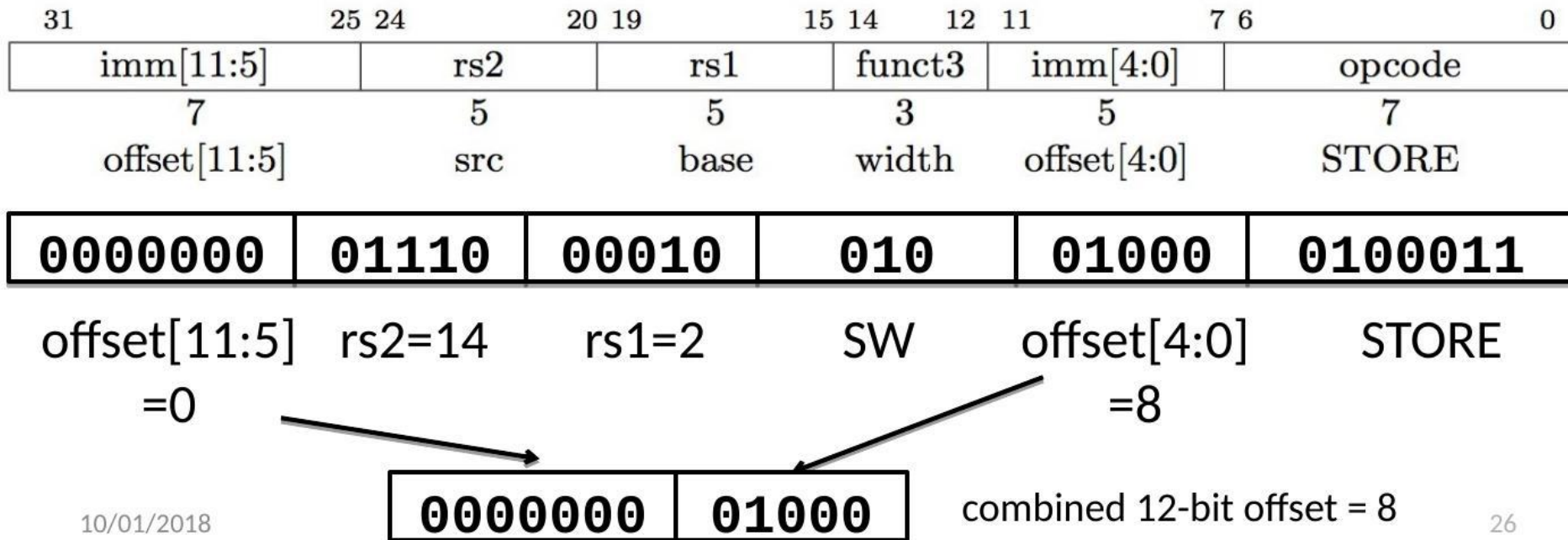
## 2. RISC-V Architecture

- RISC-V has 32 integer registers named x0 to x31 and can have 32 floating-point registers
  - Register number 0 is a constant 0
  - Register number 1 is the return address (link register)
- The memory is addressed by 8-bit bytes
- The instructions must be aligned to 32-bit addresses
- Like many RISC designs, it is a "load-store" machine
  - The only instructions that access main memory are loads and stores
  - All arithmetic and logic operations occur between registers

## 2. RISC-V Architecture

### S-Format Example

- RISC-V Assembly Instruction:  
**sw x14, 8(x2)**



## 2. RISC-V Architecture

### RISC-V integer registers

| Name | Mnemonic | Meaning                              |
|------|----------|--------------------------------------|
| x0   | zero     | always zero                          |
| x1   | ra       | return address                       |
| x2   | sp       | stack pointer                        |
| x3   | gp       | global pointer                       |
| x4   | tp       | thread pointer                       |
| x5   | t0       | temporary (alternate return address) |
| x6-7 | t1-2     | temporary registers                  |
| x8   | s0 / fp  | saved register / frame pointer       |
| x9   | s1       | saved register                       |

## 2. RISC-V Architecture

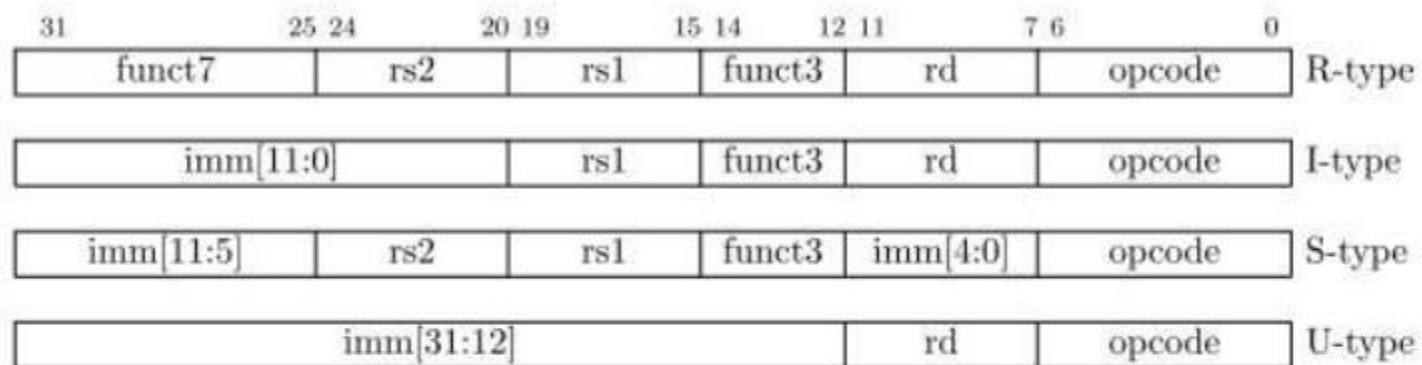
| Name   | Mnemonic | Meaning                             |
|--------|----------|-------------------------------------|
| x10-11 | a0-1     | function arguments and return value |
| x12-17 | a2-7     | function arguments                  |
| x18-27 | s2-11    | saved register                      |
| x28-31 | t3-6     | temporary registers                 |

### Note:

- ra means return address. Common used to store return address before calling a subroutine.
- t0 to t6 are temporary registers. Subroutines make no promise to save these.
- s0 to s11 are saved registers. Subroutines promise to keep these unchanged.
- a0 to a7 mean function arguments. Before jumping to a subroutine, you pass arguments into these registers.

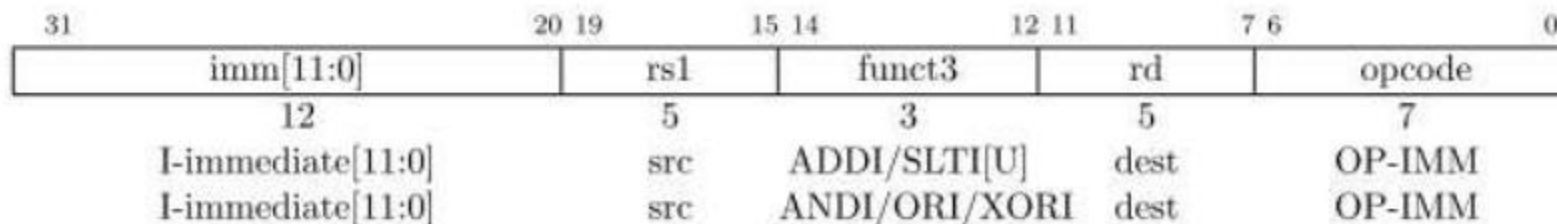
## 2. RISC-V Architecture

- The instruction set includes other features to increase a computer's speed, while reducing its cost and power usage
  - These include placing most-significant bits at a fixed location to speed sign extension, and a bit-arrangement designed to reduce the number of multiplexers in a CPU



Four base instruction formats  
(R/I/S/U)

Integer computational instructions encoded as register-immediate operations using the I-type format



# 2. RISC-V Architecture

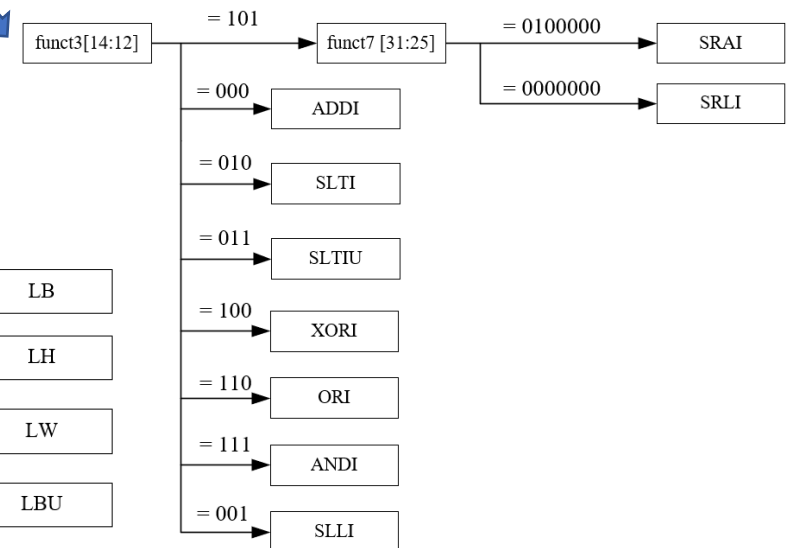
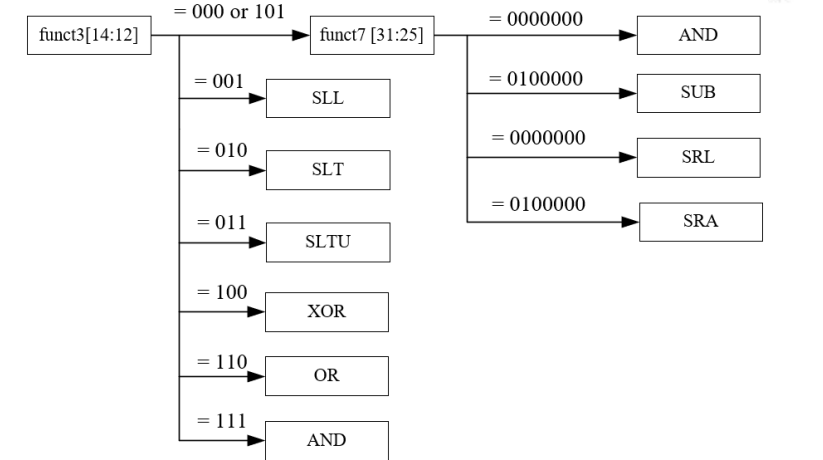
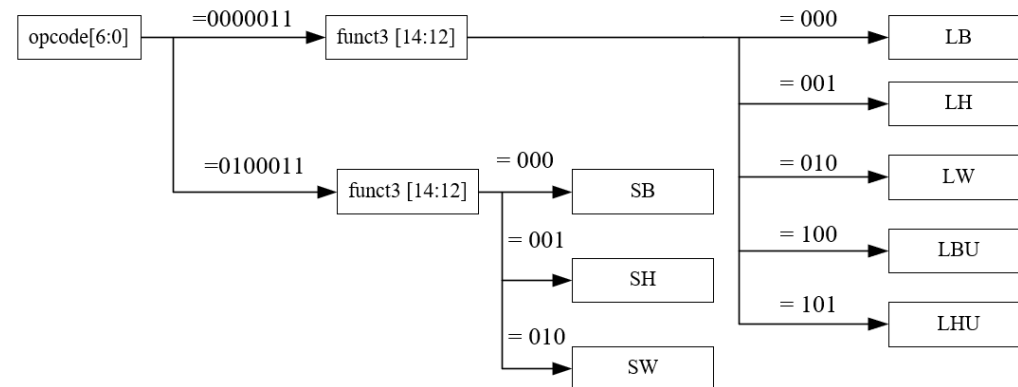
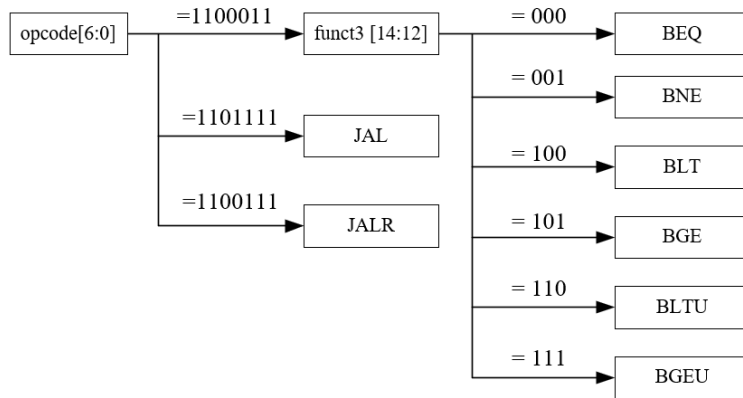
|            |     |     |        |          |        |
|------------|-----|-----|--------|----------|--------|
| funct7     | rs2 | rs1 | funct3 | rd       | opcode |
| imm[11:0]  |     | rs1 | funct3 | rd       | opcode |
| imm[11:5]  | rs2 | rs1 | funct3 | imm[4:0] | opcode |
| imm[31:12] |     |     |        | rd       | opcode |

R-type

I-type

S-type

U-type



# 3. RISC-V Instructions

- Compared to the various instructions of assembly on ARM or x86 platforms, RISC-V in contrast was made specifically to be easy to teach while pragmatic enough to actually allow the implementation of high performance microprocessors.
- <https://www.cs.cornell.edu/courses/cs3410/2019sp/riscv/interpreter/#> to see the changes of the register and the memory.
- For examples, you can check <https://medium.com/swlh/risc-v-assembly-for-beginners-387c6cd02c49>
- For detailed instruction set, please refer to: <https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf>

# 3. RISC-V Instructions

The most important instructions are probably:

- **ADDI** *rd, zero, immediate*, to load a value (immediate) into register *rd*.
- **LW** *rd, offset(rs1)* to load content at memory location  $\text{offset} + \text{rs1}$  into register *rd*.
- **SW** *rd, offset(rs1)* store content of *rd* register at memory location  $\text{offset} + \text{rs1}$ .
- **ADD** *rd, rs1, rs2* to add content of two registers and store value in destination register *rd*.
- The branch instructions **BEQ**, **BNE**, **BLT**.

# 3. RISC-V Instructions

```
int plus (int x, int y);  
void sumstore (int x, int y, int *dest)  
{  
    int t = plus (x,y);  
    *dest = t;  
}
```

Sumstore:

```
addi    sp, sp, -8  
sw      ra, 0(sp)  
sw      a2, 4(sp)  
jal     x0, plus  
lw      a2, 4(sp)  
sw      a0, 0(a2)  
lw      ra, 0(sp)  
addi    sp, sp, 8  
jalr    x0, 0(ra)
```

# 3. RISC-V Instructions

\*dest = t;

## **C code**

Store value t to memory location pointed to by “dest”

## **Assembly**

Move 4-byte value to memory

Word in RISC-V

lw a2, 4(sp)

sw a0, 0(a2)

Intel defined “word” as 16 bits

Other may define “word” differently, 32-bit is common

First have to get the memory location back from stack

## **Operands:**

t: Register a0

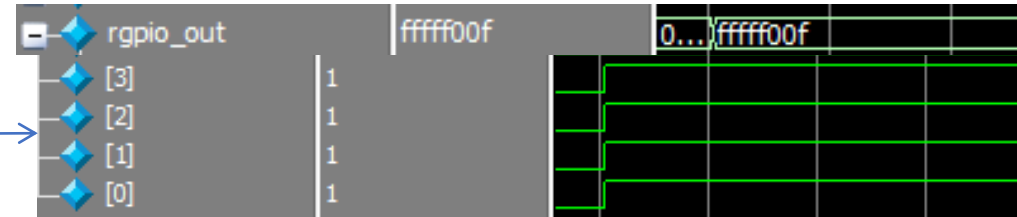
dest: Register a2 (on stack)

\*dest: Memory Mem[a2]

# 3. RISC-V Instructions

```
.org 0x0
.global _start
_start:
lui x1,0x20000
ori x1,x1,0x008
lui x2,0xfffff
ori x2,x2,0xf
sw x2,0(x1)
lui x1,0x20000
ori x1,x1,0x00c
lui x2,0x00000
ori x2,x2,0x000
sw x2,0(x1)
lui x1,0x20000
ori x1,x1,0x004
lui x2,0xfffff
ori x2,x2,0xf
sw x2,0(x1)
```

Configure registers



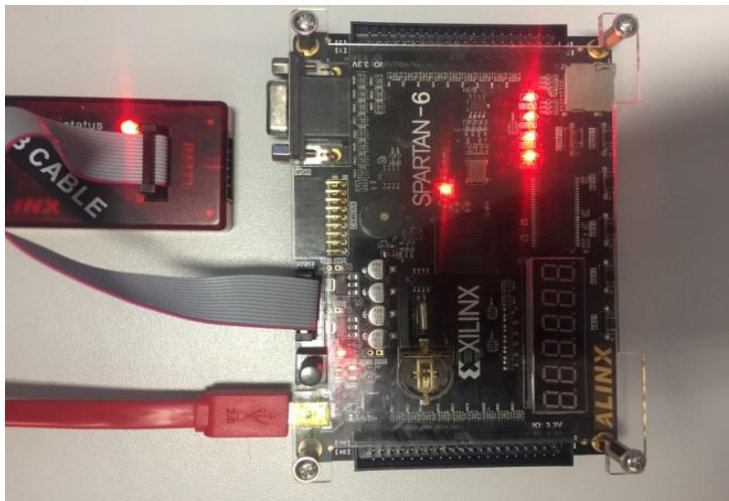
Pins constraints

```
#####LED Pin define#####
NET gpio_o<0>      LOC = P4 | IOSTANDARD = "LVCMOS33"; ##
NET gpio_o<1>      LOC = N5 | IOSTANDARD = "LVCMOS33"; ##
NET gpio_o<2>      LOC = P5 | IOSTANDARD = "LVCMOS33"; ##
NET gpio_o<3>      LOC = M6 | IOSTANDARD = "LVCMOS33"; ##
```

Map pins to board

|                       |    |       |
|-----------------------|----|-------|
| IO_L48N_RDWR_B_VREF_2 | N5 | LED1  |
| IO_L49P_D3_2          | P5 | LED2  |
| IO_L49N_D4_2          | L8 | L8    |
| IO_L62P_D5_2          | L7 | VGAD4 |
| IO_L62N_D6_2          | P4 | LED0  |
| IO_L63P_2             | T4 | T4    |
| IO_L63N_2             | M6 | LED3  |
| IO_L64P_D8_2          |    |       |

Program bit stream



C++ source #1 X

A B + - v f

C++

```
1
2 int get_value(int x)
3
4 return 42;
5
```

RISC-V rv64gc gcc 11.2.0 (C++, Editor #1, Compiler #1) X

RISC-V rv64gc gcc 11.2.0 version -O3

A B + - v f

```
1 get_value(int):
2     li    a0,42
3     ret
```

x86-64 gcc 11.2 (C++, Editor #1, Compiler #2) X

x86-64 gcc 11.2 nversion -O3

A B + - v f

```
1 get_value(int):
2     mov    eax, 42
3     ret
```

ARM64 gcc 11.2 (C++, Editor #1, Compiler #4) X

ARM64 gcc 11.2 rsion -O3

A B + - v f

```
1 get_value(int):
2     mov    w0, 42
3     ret
```

Output (0/4) RISC-V rv64gc gcc 11.2.0 i - 613ms (27758) - 191

lines filtered

Output (0/4) x86-64 gcc 11.2 i - 569ms (27118) - 167 lines filtered

lines filtered

Output (0/4) ARM64 gcc 11.2 i - 438ms (31948) - 201 lines

filtered

x86-64 clang 14.0.0 (C++, Editor #1, Compiler #3) X

x86-64 clang 14.0.0 --target=risc

A B + - v f

```
1 get_value(int):
2     li    a0, 42
3     ret
```

x86-64 clang 14.0.0 (C++, Editor #1, Compiler #5) X

x86-64 clang 14.0.0 -std=c++20

A B + - v f

```
1 get_value(int):
2     mov    eax, 42
3     ret
```

x86-64 clang 14.0.0 (C++, Editor #1, Compiler #6) X

x86-64 clang 14.0.0 -std=c++

A B + - v f

```
1 get_value(int):
2     mov    w0, #42
3     ret
```

COMPILER EXPLORER

Add... More

Sponsors

Backtrace

intel

Solid Compiler

Share Policies Other

C++ source #1

C++

```
1 int get_value(int x, int y)
2 {
3     return 44+x*5+y;
4 }
5
```

RISC-V rv64gc gcc 11.2.0 (C++, Editor #1, Compiler #1)

RISC-V rv64gc gcc 11.2.0

version -O3

```
1 get_value(int, int):
2     slliw    a5,a0,2
3     addw    a5,a5,a0
4     addiw   a5,a5,44
5     addw    a0,a5,a1
6     ret
```

Output (0/0)

RISC-V rv64gc gcc 11.2.0

- 276ms (31398) - 218 lines filtered

x86-64 gcc 11.2 (C++, Editor #1, Compiler #2)

x86-64 gcc 11.2

rch=alderlake

```
1 value(int, int):
2     lea     eax, [rdi+44+rdi*4]
3     add     eax, esi
4     ret
```

Output (0/0)

x86-64 gcc 11.2

- 583ms (28606) - 173 lines filtered

ARM64 gcc 11.2 (C++, Editor #1, Compiler #4)

ARM64 gcc 11.2

ersion -O3

```
1 get_value(int, int):
2     add     w0, w0, w0, lsl
3     add     w0, w0, 44
4     add     w0, w0, w1
5     ret
```

Output (0/0)

ARM64 gcc 11.2

- 626ms (35978) - 227 lines filtered

x86-64 clang 14.0.0 (C++, Editor #1, Compiler #3)

x86-64 clang 14.0.0

get=riscv64

```
1 get_value(int, int):
2     slliw    a2, a0, 2
3     addw    a0, a0, a2
4     addw    a0, a0, a1
5     addiw   a0, a0, 44
6     ret
```

Output (0/0)

x86-64 clang 14.0.0

- 276ms (31398) - 218 lines filtered

x86-64 clang 14.0.0 (C++, Editor #1, Compiler #5)

x86-64 clang 14.0.0

-march=alde

```
1 get_value(int, int):
2     lea     eax, [rdi + 4*rdi]
3     add     eax, esi
4     add     eax, 44
5     ret
```

Output (0/0)

x86-64 clang 14.0.0

- 583ms (28606) - 173 lines filtered

x86-64 clang 14.0.0 (C++, Editor #1, Compiler #6)

x86-64 clang 14.0.0

et=arm64

```
1 get_value(int, int):
2     add     w8, w0, w0, lsl
3     add     w8, w8, w1
4     add     w0, w8, #44
5     ret
```

Output (0/0)

x86-64 clang 14.0.0

- 626ms (35978) - 227 lines filtered

# 3. RISC-V Instructions

<https://www.godbolt.org/>

Compile the code into RISC-V RV32gc clang

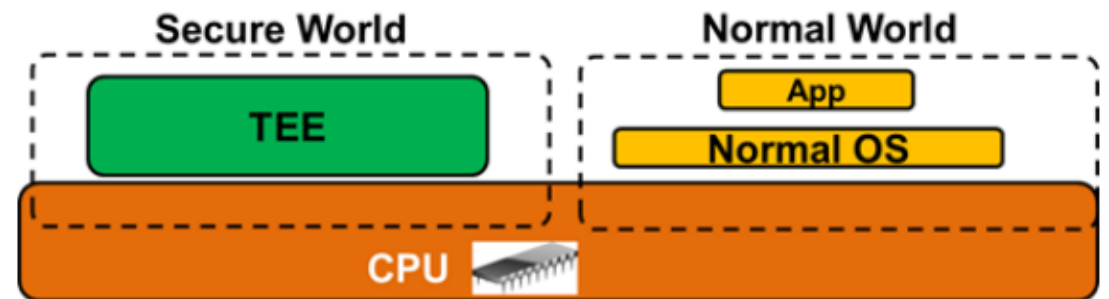
```
int square(int num)
{
    return num * num;
}
```

## 4. RISC-V Examples

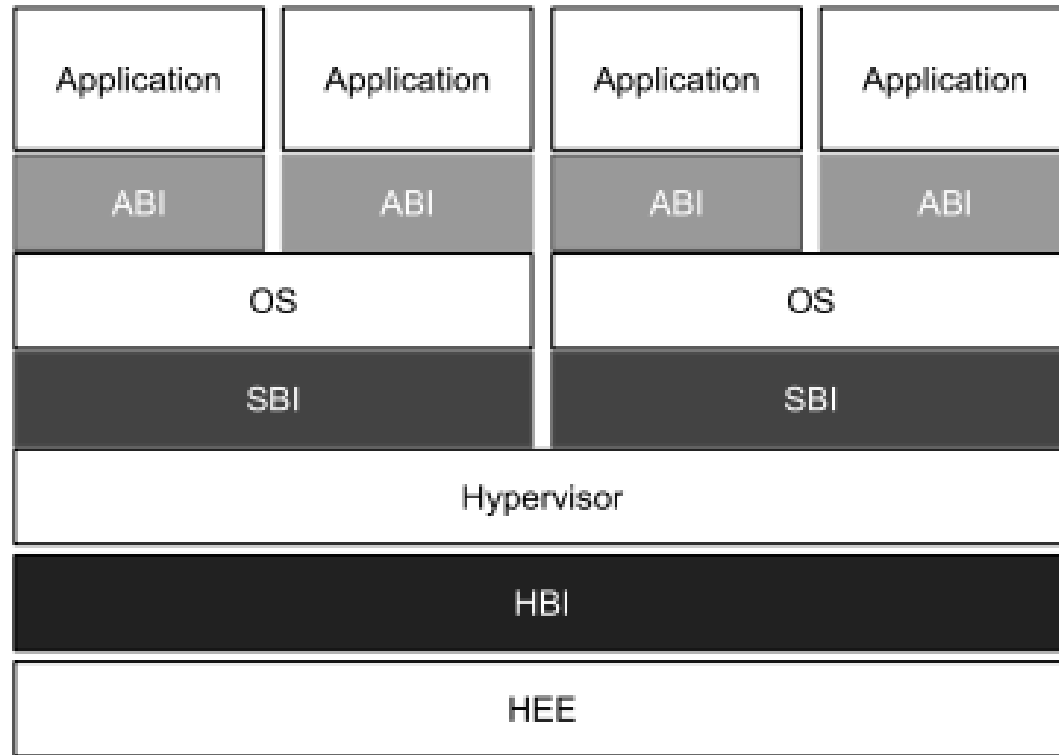
### Trusted Execution Environment

Current OS and Hardware have many vulnerabilities, and Critical Applications are involved. Critical Applications are desired to be run independent from the OS.

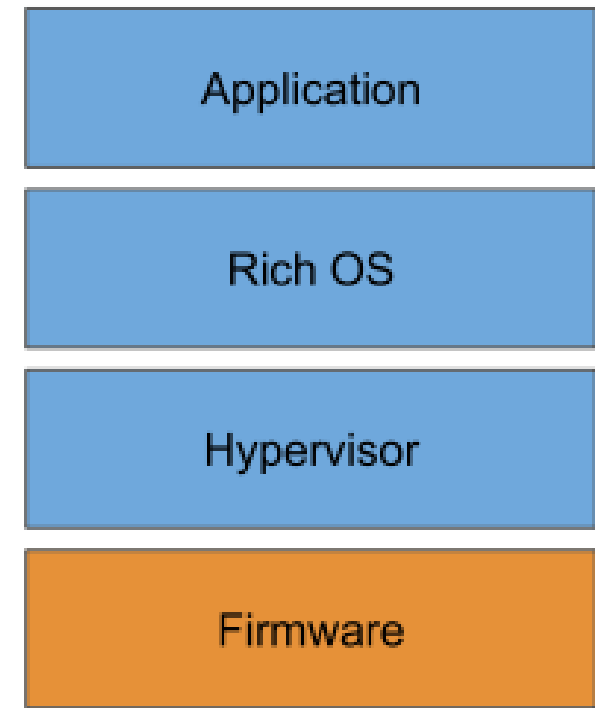
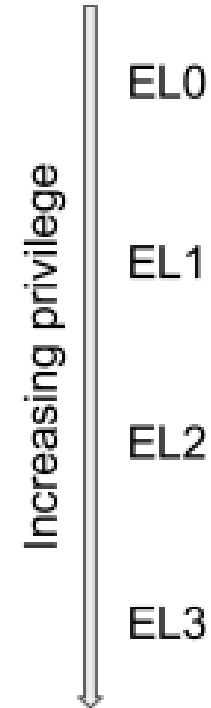
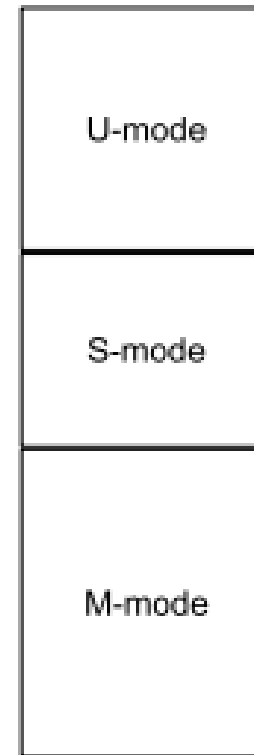
Trusted Execution Environment (TEE) is a secure area of a computer's hardware that ensures the integrity and confidentiality of sensitive data and code.



## 4. RISC-V Examples



(a) RISC-V.

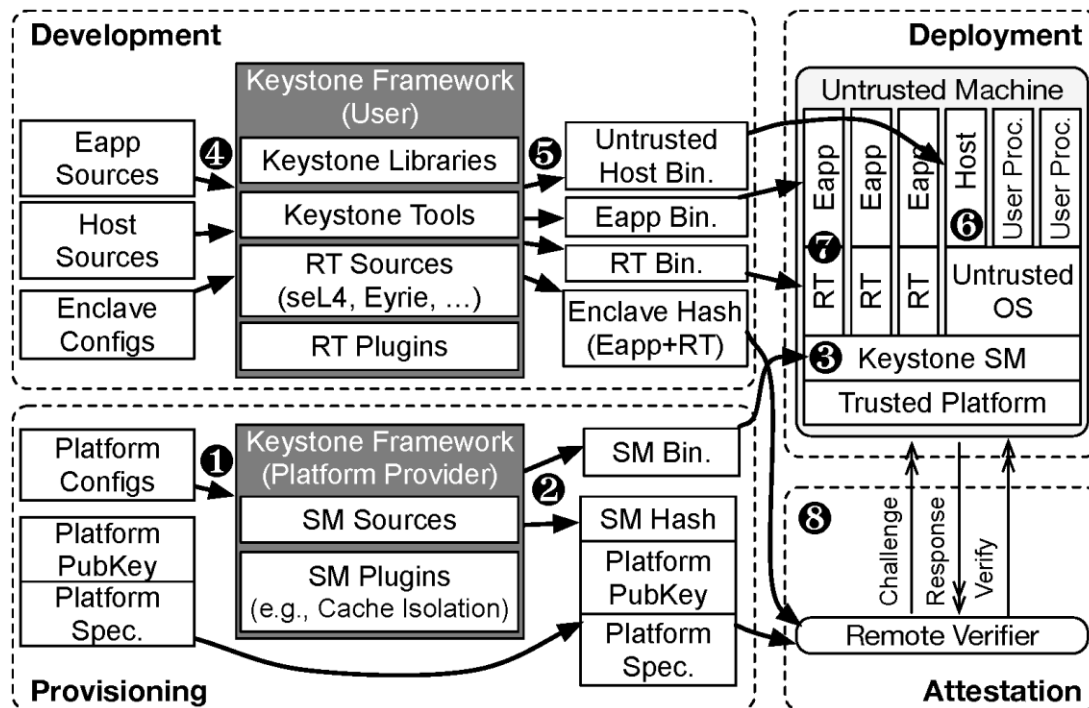


(b) Armv8-A.

Privilege Levels for (a) RISC-V and (b) ARMv8-A

# 4. RISC-V Examples

## Keystone Workflow

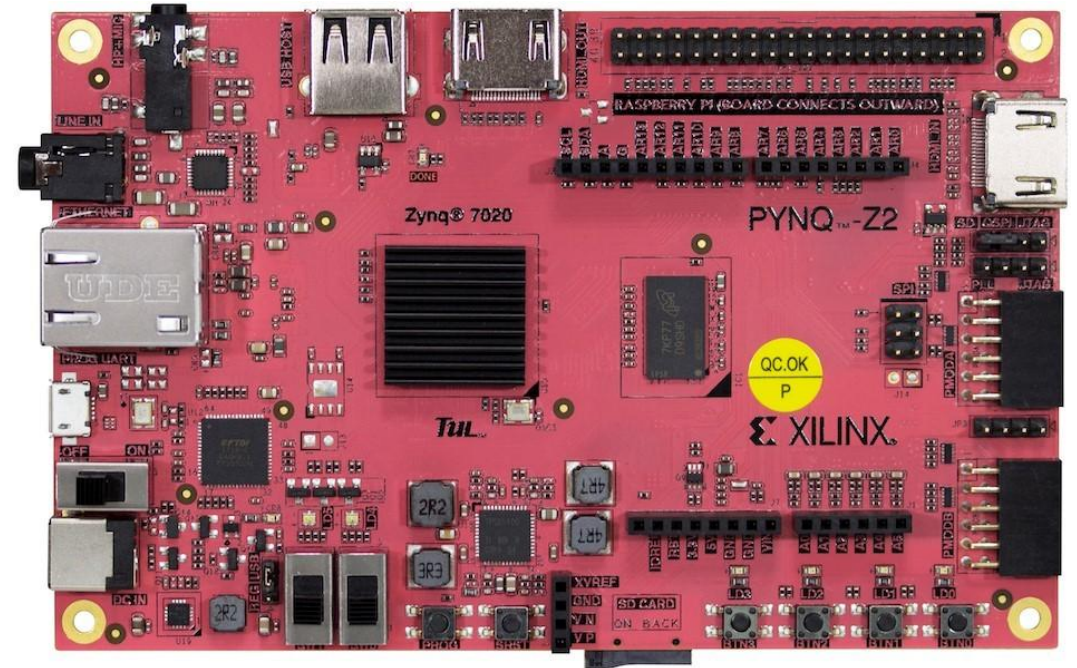


- 1 Platform provider configures the SM.
- 2 Keystone compiles and generates the SM boot image.
- 3 Platform provider deploys the SM.
- 4 Developer writes an eapp, configures the enclave.
- 5 Keystone builds the binaries, computes measurements.
- 6 Untrusted host binary is deployed to the machine.
- 7 Host deploys the RT, the eapp, and initiates the enclave creation.
- 8 Remote verifier can attest based on known platform specifications, keys, and SM/enclave measurements.

## 4. RISC-V Examples

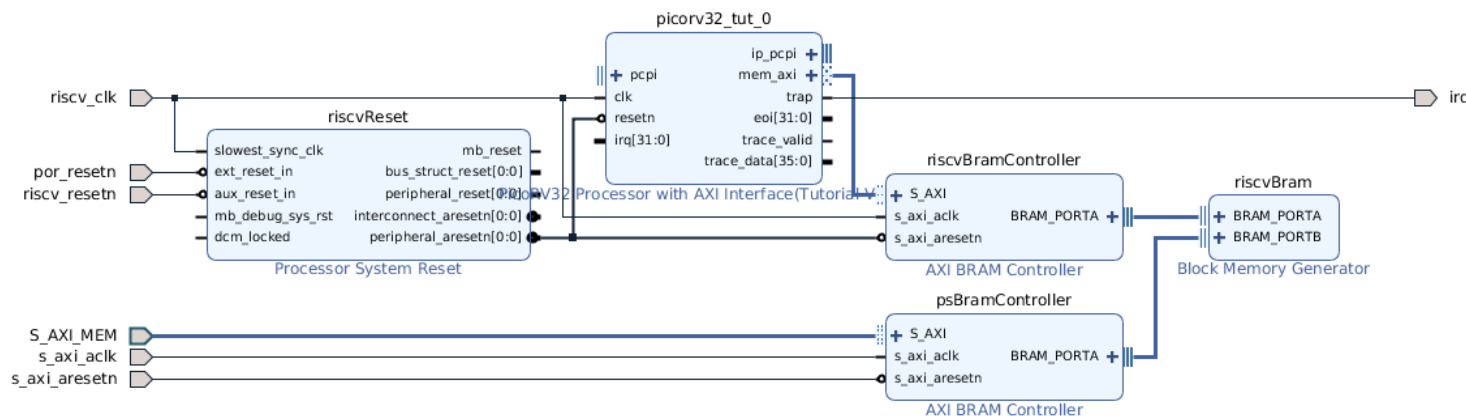
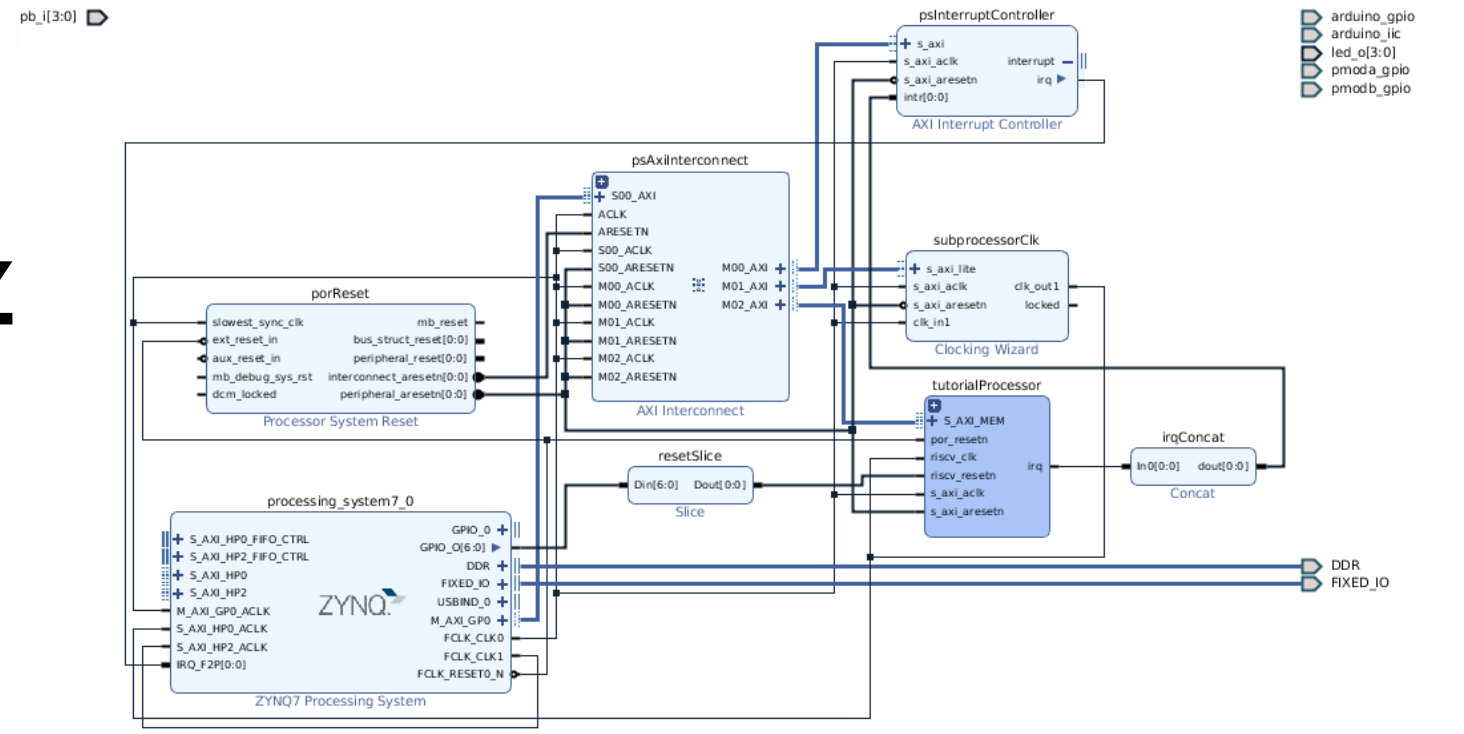
### RISC-V and TEE on PYNQ-Z2

- Board setup and downloading dependencies
- Building a Bitstream
- Compiling RISC-V GCC Toolchain
- Packaging a Bitstream into an Overlay
- Install Multizone-Linux and FPGA



# 4. RISC-V Examples

## RISC-V on PYNQ-Z



# 5. RISC-V vs ARM

|                 | <b>RISC-V</b>                                                                                       | <b>ARM</b>                                                                      |
|-----------------|-----------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------|
| Origin          | University of California, Berkeley (2010)                                                           | Acorn Computers (1980s)                                                         |
| Architecture    | RISC, load-store architecture                                                                       | RISC, load-store architecture                                                   |
| Instruction set | RV32I (32-bit), RV64I (64-bit), and RV128I (128-bit)                                                | ARM (32-bit), Thumb (16-bit) and AArch64 (64-bit)                               |
| Processor cores | Rocket core, SiFive products including performance, automotive, intelligence and essential families | Cortex-A, Cortex-R, Cortex-M and Neoverse series                                |
| Licenses        | Royalty-free open-source licenses                                                                   | Closed ISA, a proprietary architecture with licensing fees                      |
| Key features    | Open-source architecture with modularity and extensibility                                          | Dominant choice for embedded systems and mobile devices with a mature ecosystem |

# Reference

- Extensive application prospect of RISC-V cores and rapid growth of RISC-V community:
  - <https://www.youtube.com/watch?v=plU800gGdws>
- RISC-V assembly programming: ISA, registers, load/store instructions and compiling C codes into assembly codes:
  - [https://www.youtube.com/watch?v=XVU\\_RNiz09A](https://www.youtube.com/watch?v=XVU_RNiz09A)
- Comparing ARM vs RISC-V vs x86\_64 with GCC vs Clang using Compiler Explorer:
  - [https://www.youtube.com/watch?v=NE\\_v4RI dPMA](https://www.youtube.com/watch?v=NE_v4RI dPMA)