KINDAI UNIVERSITY

MPSoC'19

Challenges for processor instruction extension in MPSoC Era

1)Yoshinori Takeuchi and 2)Ryo Taketani

1) Dept. of Elect. & Elec. Eng., Kindai University

2) Grad. School of IST, Osaka University

E-mail: takeuchi@ele.kindai.ac.jp

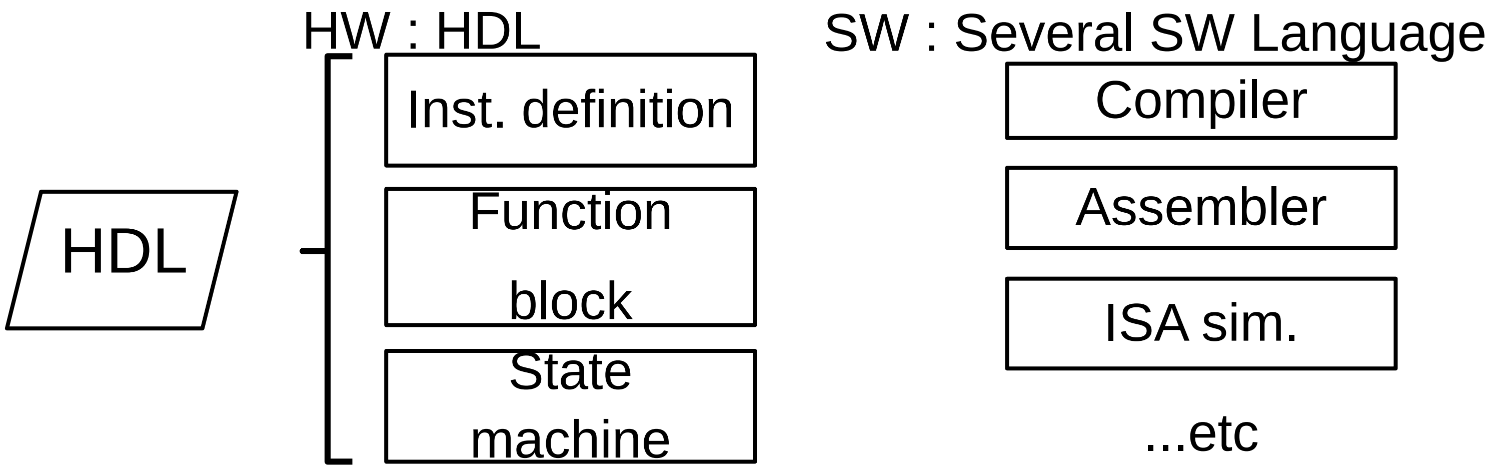
Osaka University

Introduction

- IoT devices are of large variety
  - Almost devices include at least one processor
  - Application specific processings are required for IoT devices
  - Configurable processors are expected to be a solution

Problem for Instruction extensions

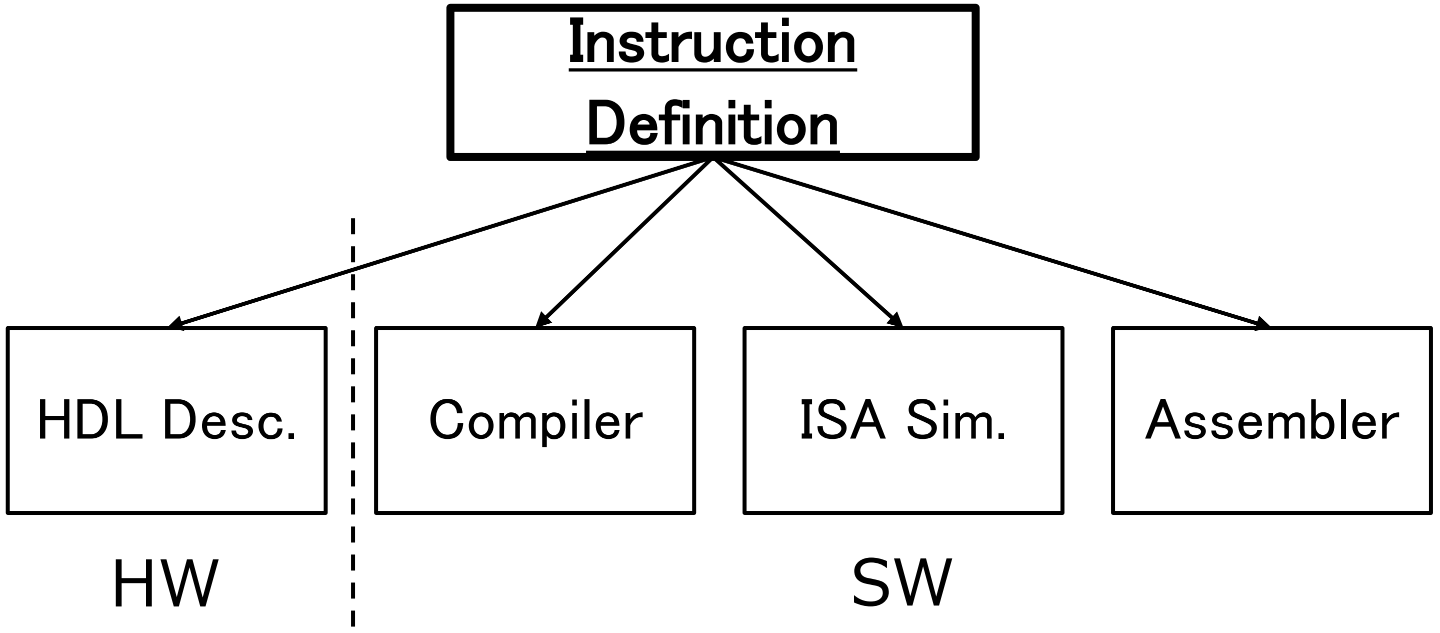
- Hardware and software require separate descriptions for inst. extensions



⇒ Description amount is not small,
and requires professional skills

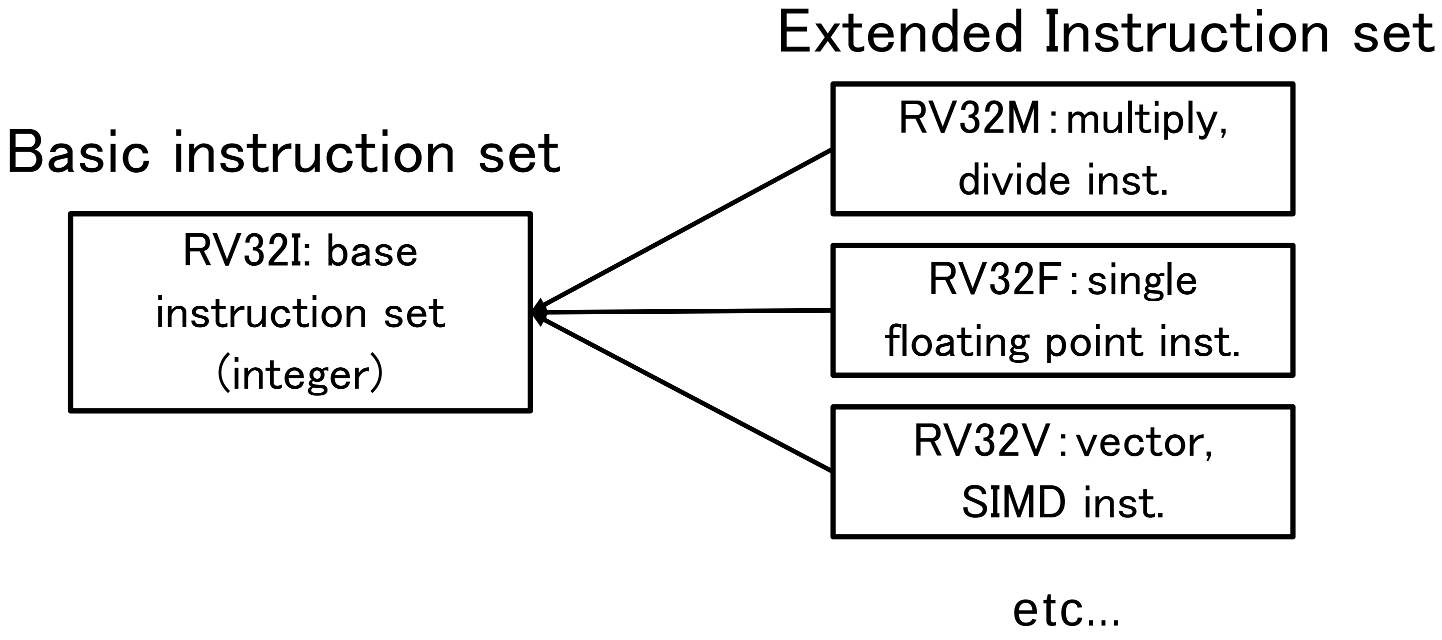
Our challenge

- Simple and error free instruction extensions
  - **From unique instruction definition, <u>HW&SW environments are generated</u>**



RISC-V Instruction set [1]

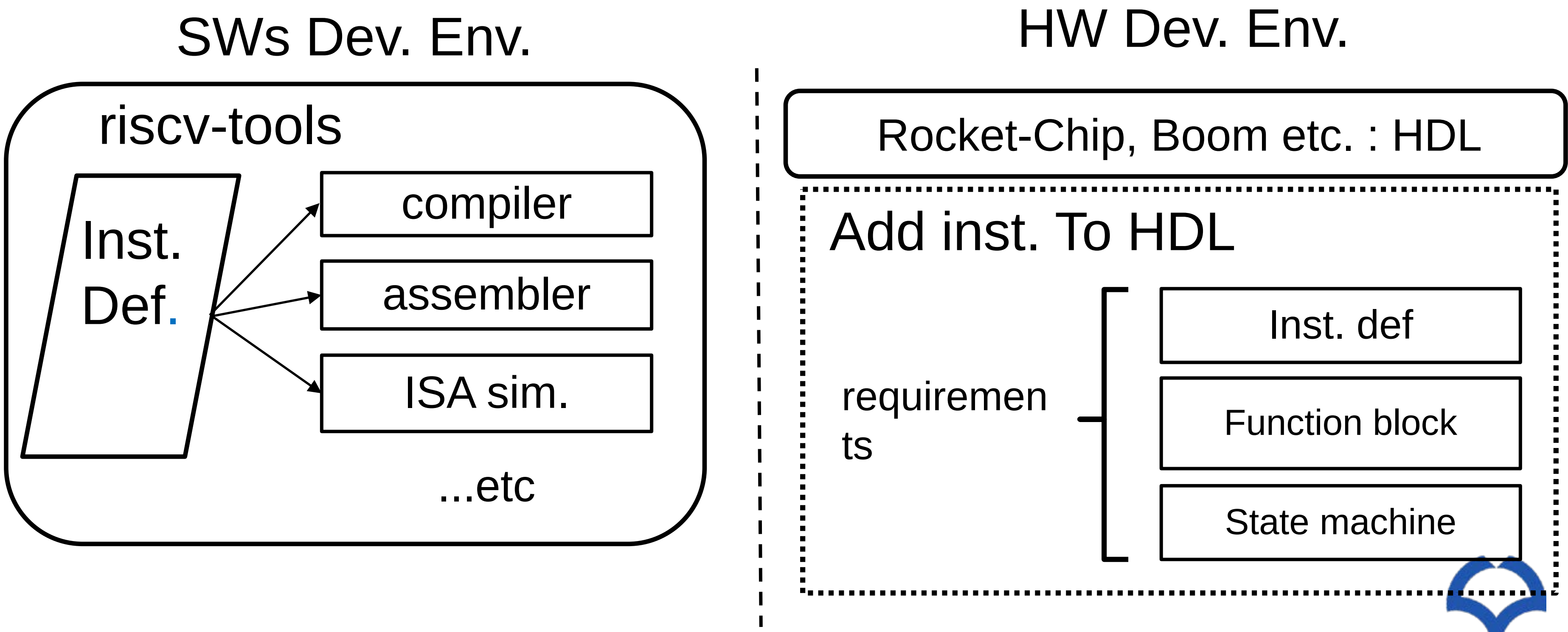
- Open Instruction set Architecture
- ISA organization（32bit addressing mode）


Extended Instruction set
Basic instruction set
RV32I: base instruction set (integer)
RV32M：multiply, divide inst.
RV32F：single floating point inst.
RV32V：vector, SIMD inst.
etc…

[1] https://content.riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

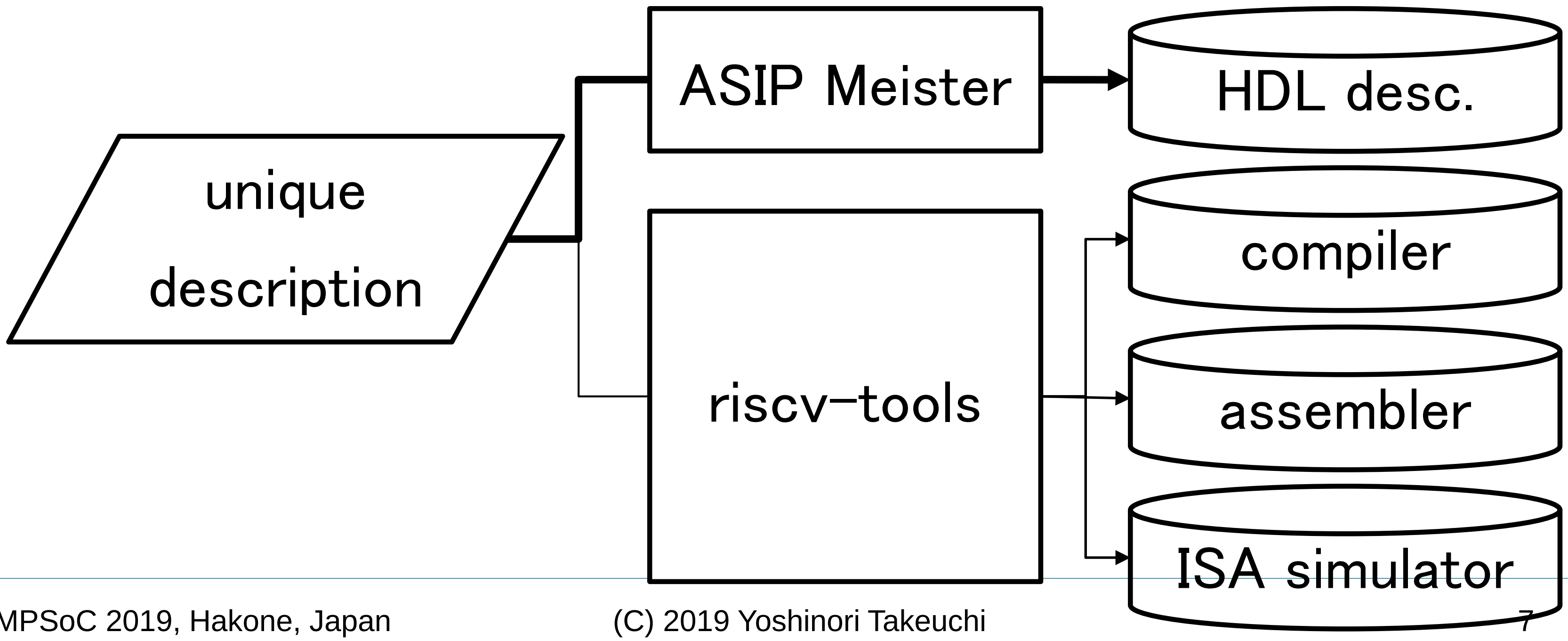
RISC-V Development Environment

- Fields for custom instructions
- Instruction Extension for SW & HW Dev. Env.



Basic idea and overview

- From unique instruction definition, generate HW&SW environments for RISC-V processor with extended Instructions



Instruction extension on ASIP Meister 1

- ASIP Meister: Application domain-specific processor development environment

1. Opcode and operand definition

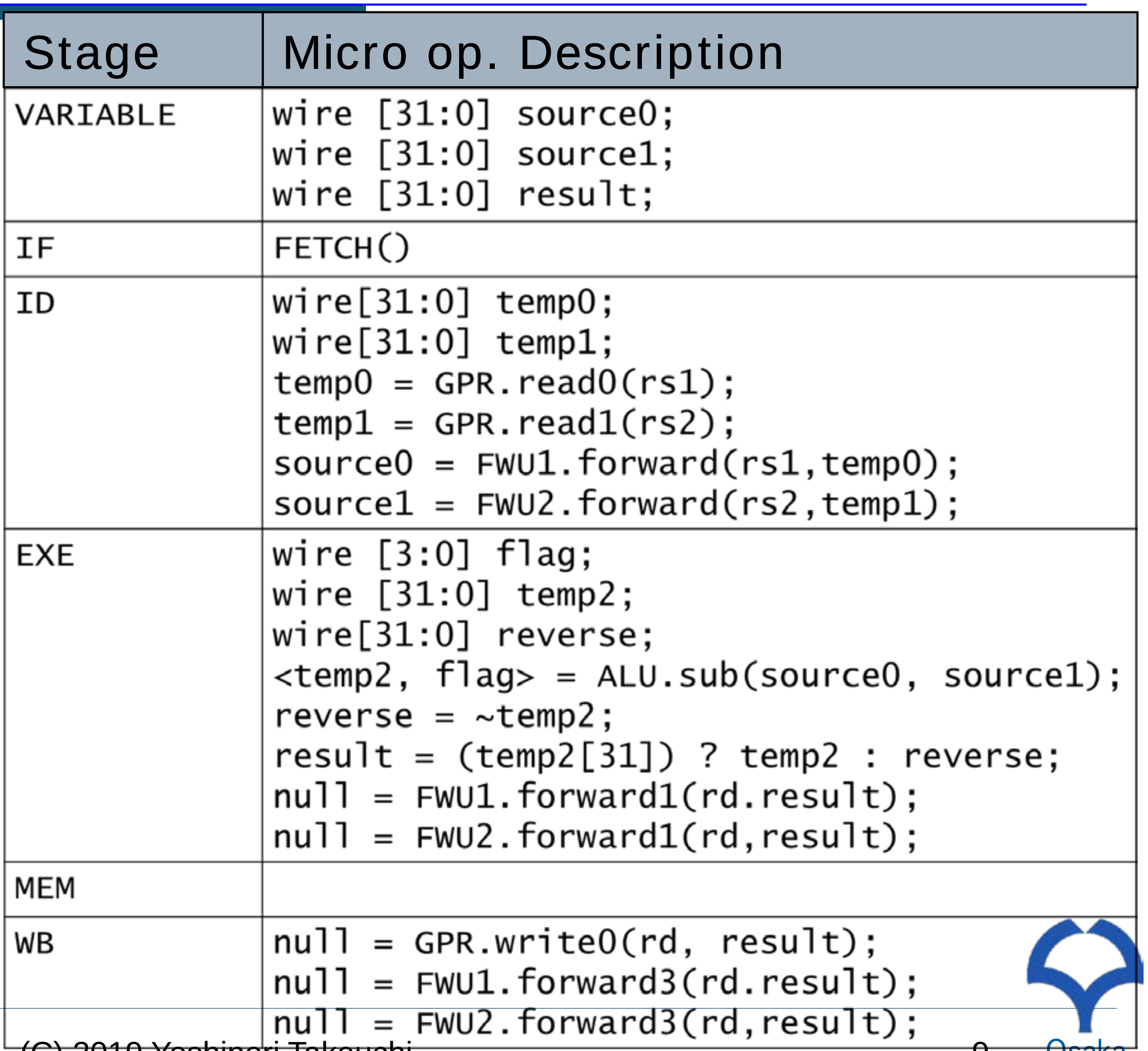
Format	**<u>ABSO rd, rs1, rs2</u>**

Inst. type
<u>R</u>
I
S
U
B
J

MSB	LSB	Field Type	Field Attr	Value
31	25	opcode	binary	**<u>0000000</u>**
24	20	operand	name	rs2
19	15	operand	name	rs1
14	12	opcode	binary	**<u>000</u>**
11	7	operand	name	rd
6	0	opcode	binary	**<u>1010111</u>**

Instruction extension on ASIP Meister 2

2. Micro op. descriptions on each pipeline stage



Stage	Micro op. Description

VARIABLE	`wire [31:0] source0;`
	`wire [31:0] source1;`
	`wire [31:0] result;`

IF	`FETCH()`

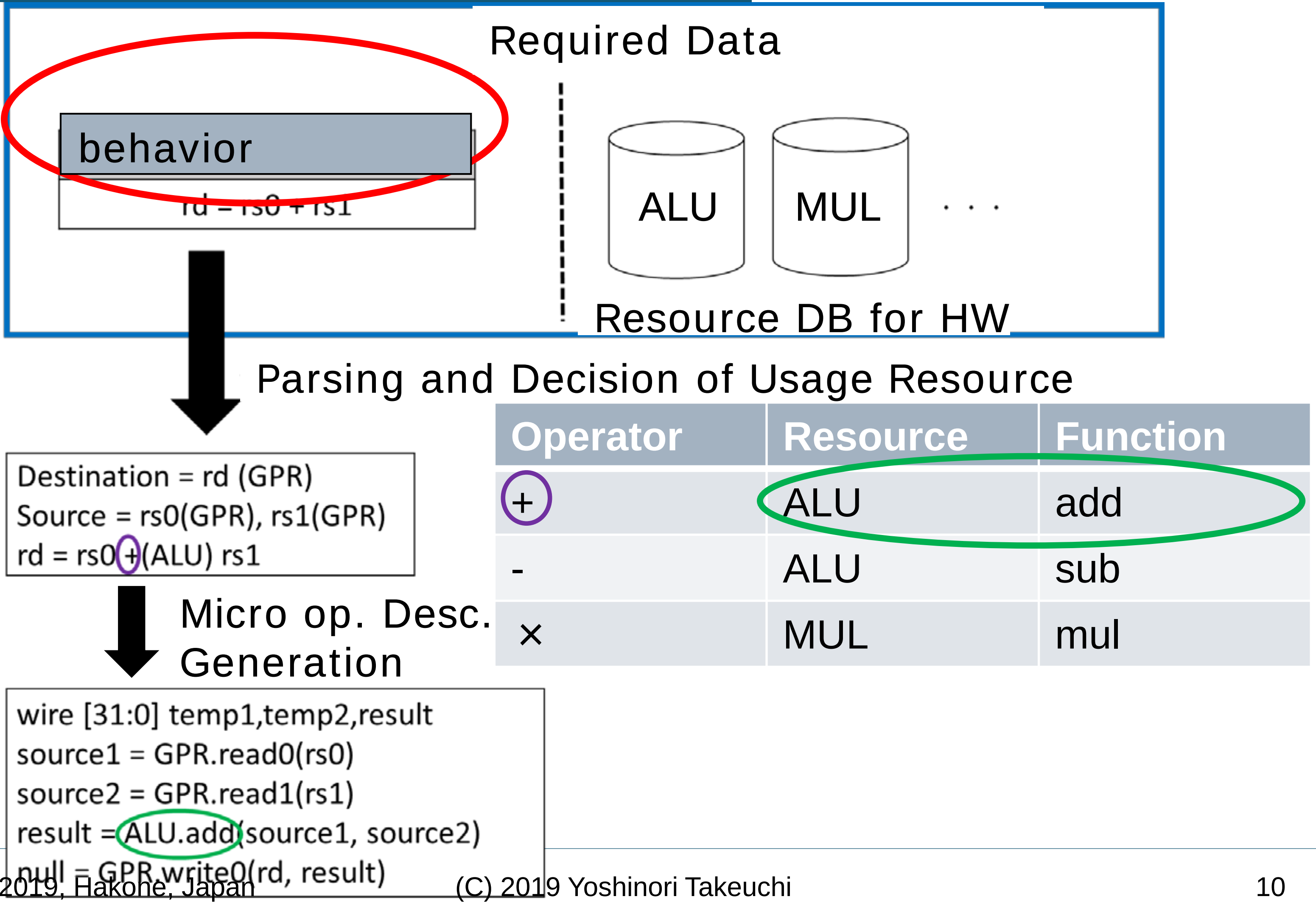
ID	`wire[31:0] temp0;`
	`wire[31:0] temp1;`
	`temp0 = GPR.read0(rs1);`
	`temp1 = GPR.read1(rs2);`
	`source0 = FWU1.forward(rs1,temp0);`
	`source1 = FWU2.forward(rs2,temp1);`

EXE	`wire [3:0] flag;`
	`wire [31:0] temp2;`
	`wire[31:0] reverse;`
	`<temp2, flag> = ALU.sub(source0, source1);`
	`reverse = ~temp2;`
	`result = (temp2[31]) ? temp2 : reverse;`
	`null = FWU1.forward1(rd.result);`
	`null = FWU2.forward1(rd,result);`

MEM	

WB	`null = GPR.write0(rd, result);`
	`null = FWU1.forward3(rd.result);`
	`null = FWU2.forward3(rd,result);`

Micro op. description from Behavior



```
Destination = rd (GPR)
Source = rs0(GPR), rs1(GPR)
rd = rs0 +(ALU) rs1
```

Operator	Resource	Function
+	ALU	add
-	ALU	sub
×	MUL	mul

```
wire [31:0] temp1,temp2,result
source1 = GPR.read0(rs0)
source2 = GPR.read1(rs1)
result = ALU.add(source1, source2)
null = GPR.write0(rd, result)
```

Conclusion and Future Work

- We introduce our challenges for processor instruction extension for RISC-V ISA based configurable processors
- Future work
  - Trials for several ISA extension designs
  - Evaluation by real examples
  - Evaluation from several metrics of designed processors

Acknowledgment

- This work was partly supported by JSPS KAKENHI Grant Number JP17K00077.

Thank you
for your attention!