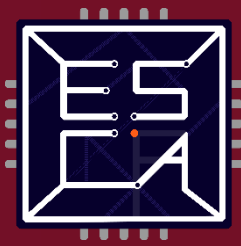
2025-01-15



PID Control in RISCV Vector extension

강준석 석사 과정

List

1.RISCV Vector extension

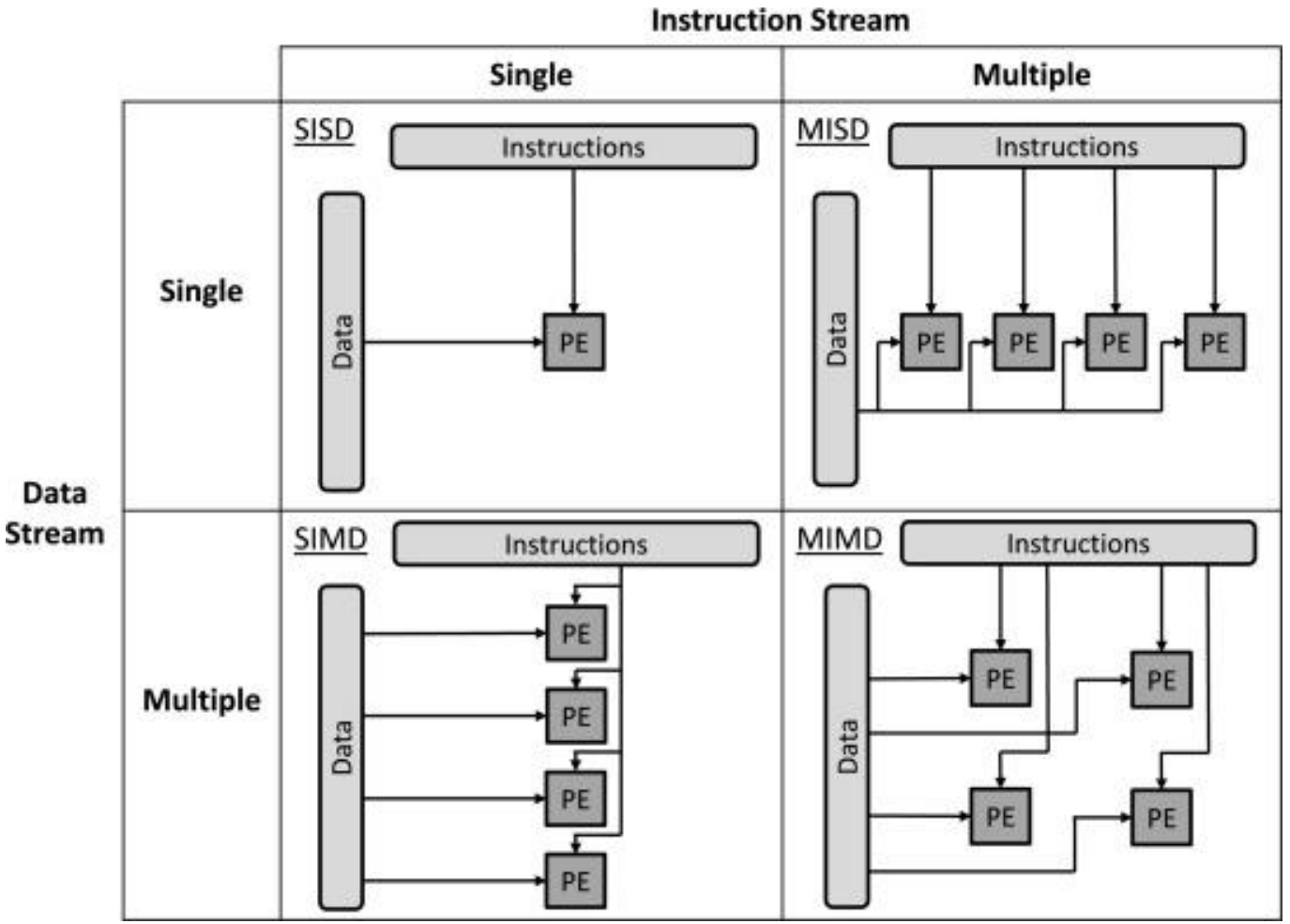
2.The concept of PID Control

3.PID LD, ST custom instruction

4.SPIKE-isa simulation

5.Future works

RISCV Vector extension



RISCV Vector extension

Terminology

- VLEN: Vector register length in bits(bit)
- ELEN: Largest Element Width in bits(bit)

HW

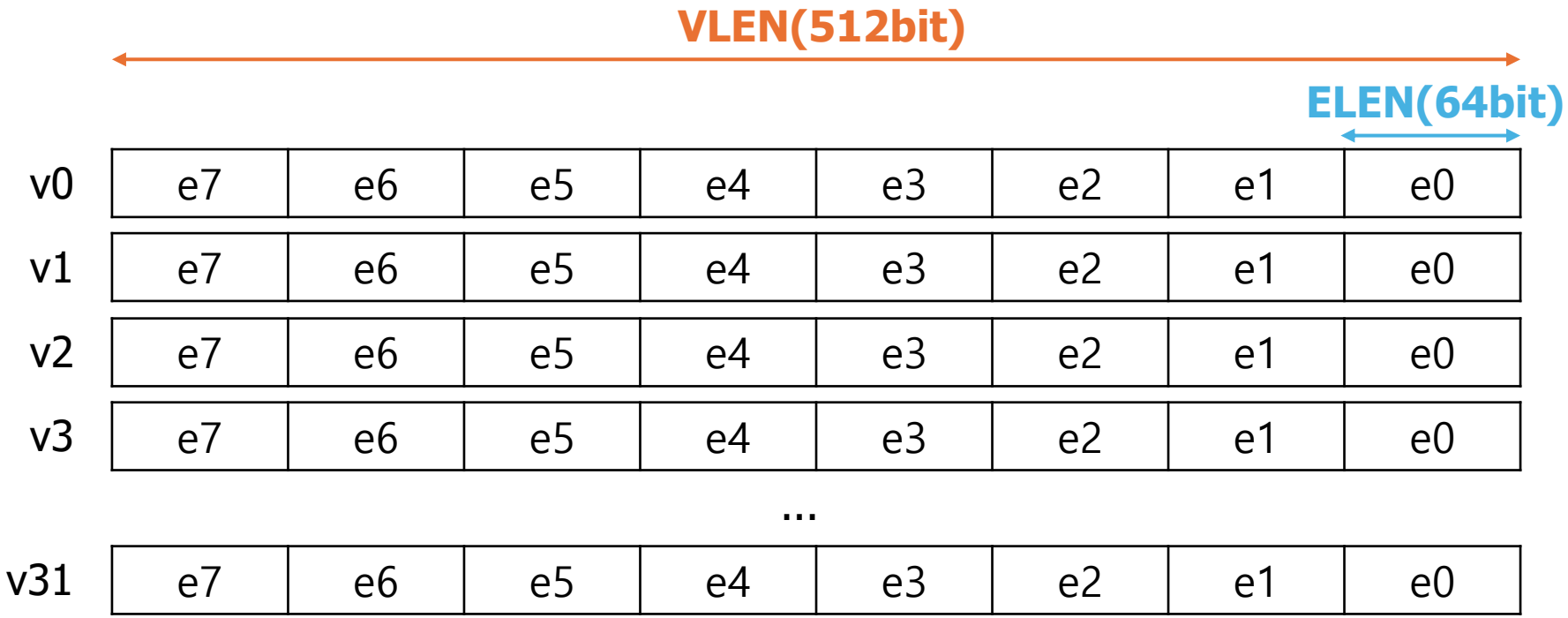
- SEW: Selected Element Width in bits(bit)
- LMUL: Register grouping multiple(#1/8~8)
- EEW: Effective Element Width
- EMUL: Effective LMUL
- VLMAX: The max num of Vector element(#)
- VL(AVL): The num of Application Vector element(#)

SW

RISCV Vector extension - Register

- VLEN: Vector register length in bits(bit)
- ELEN: Largest Element Width in bits(32 or 64bit)

➡ HW Fixed



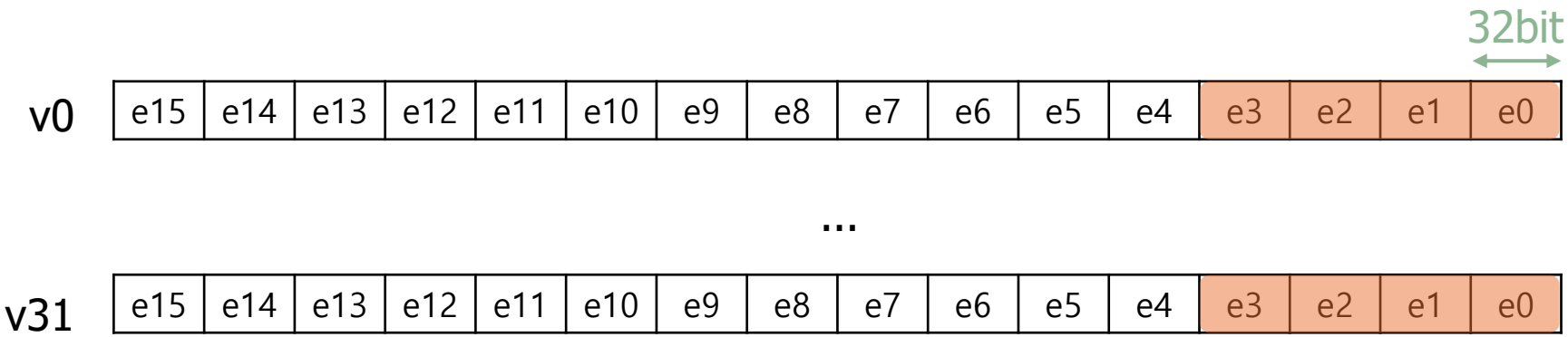
*v0~v31: vector register
*e0~e7: element

ESCA
Embedded Systems & Computer Architecture Labs

RISCV Vector extension - Setting

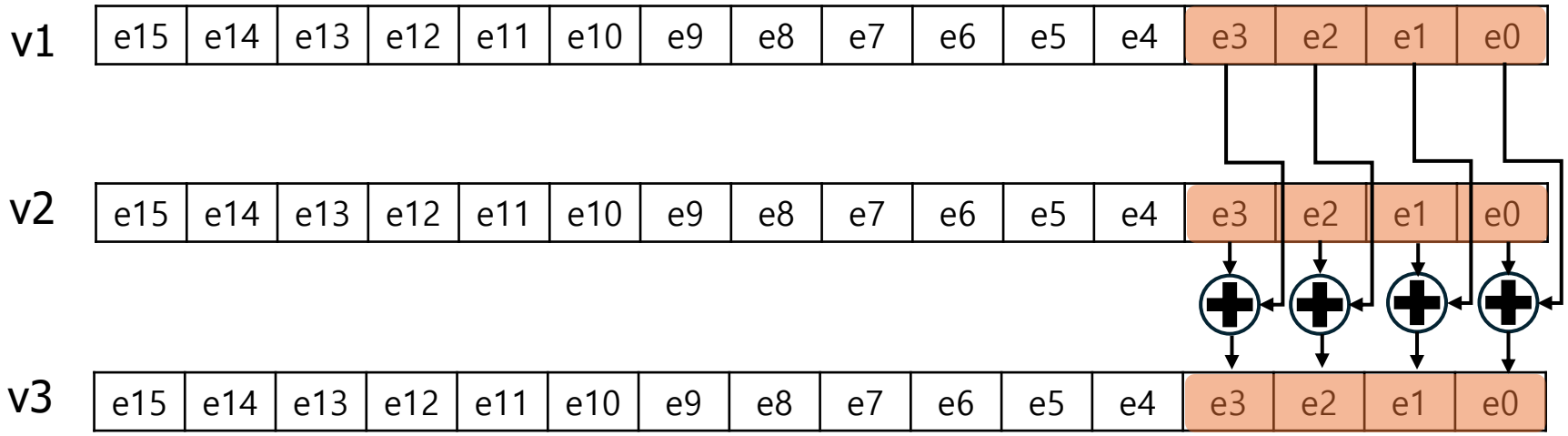
- VL : 4 // The num of Application Vector element
- SEW(<=ELEN) : 32bit // Selected Element Width in bits
- LMUL : 1 // Register grouping multiple
- vsetivli x10, 4, e32, m1, ta, ma

- VLMAX=VLEN/SEW*LMUL=512/32*1=16



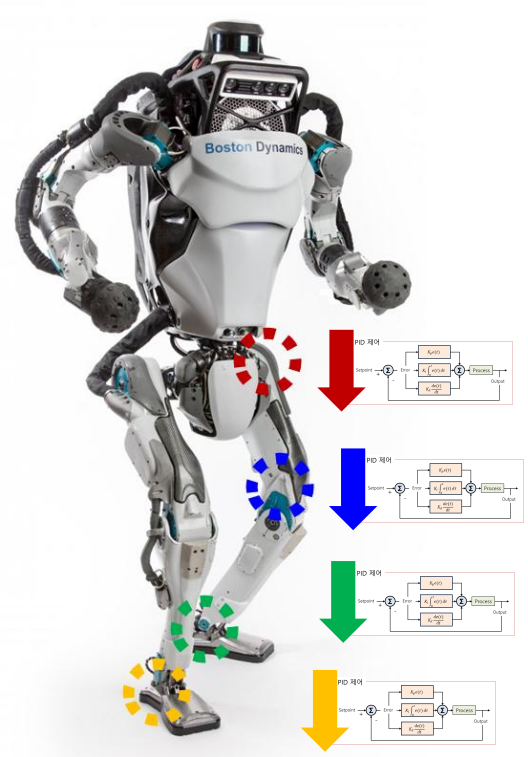
RISCV Vector extension

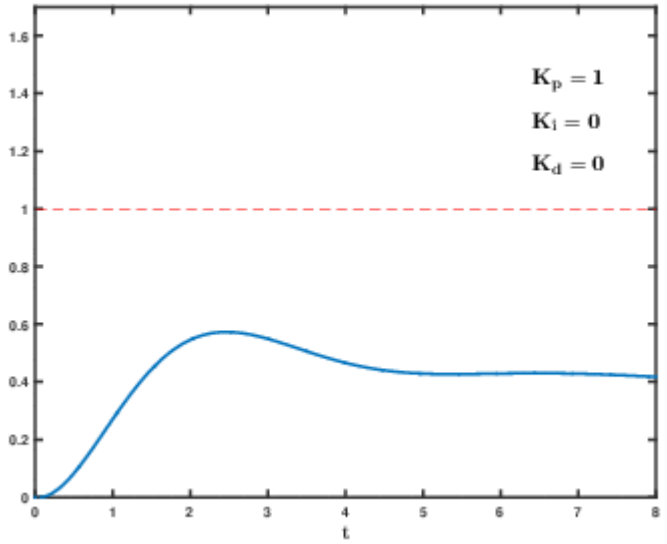
vsetivli x10, 4, e32, m1, ta, ma
vadd.vv v3, v1, v2

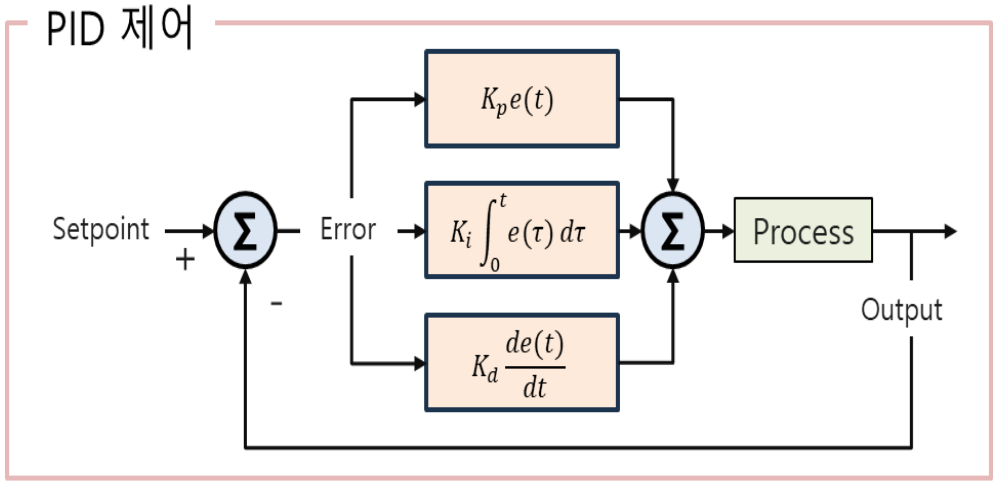


PID Control

PID: Proportional Integral Derivative

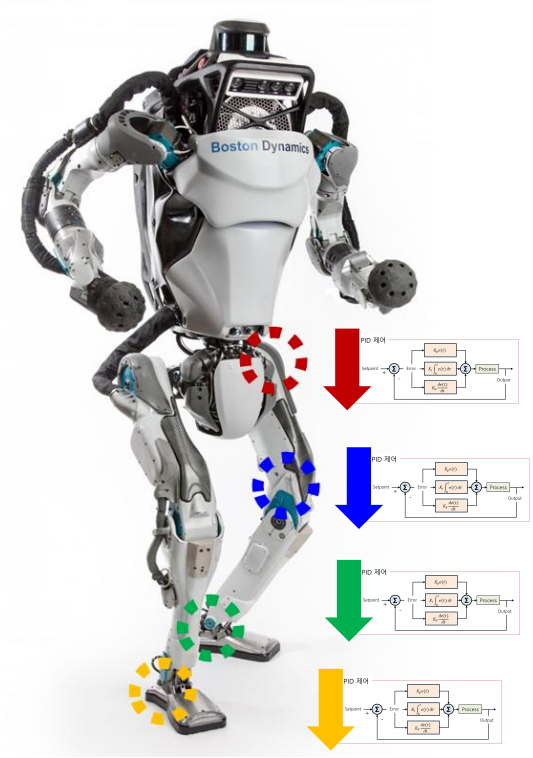






ESCA
Embedded Systems & Computer Architecture Labs

PID Control




```
struct pid {
    double kp;
    double ki;
    double kd;
    double setpoint;
    double integral, prev_error;
    double current_angle;
    double prev_time;
} actuators[4];
```

*field: the elements in the struct
*segment: actuator

RISC-V Scalar processor

```
double computePID(){
pidProportional();
pidIntegral();
pidDeravitive();
}
```

actuators[0]
actuators[1]
actuators[2]
actuators[3]

serial

Long Time

RISC-V Vector processor

```
double computeVPID(){
  VPIDLOAD.v
  VPID.vv
  VPIDSTORE.v
}
```

actuators[0] actuators[1] actuators[2] actuators[3]

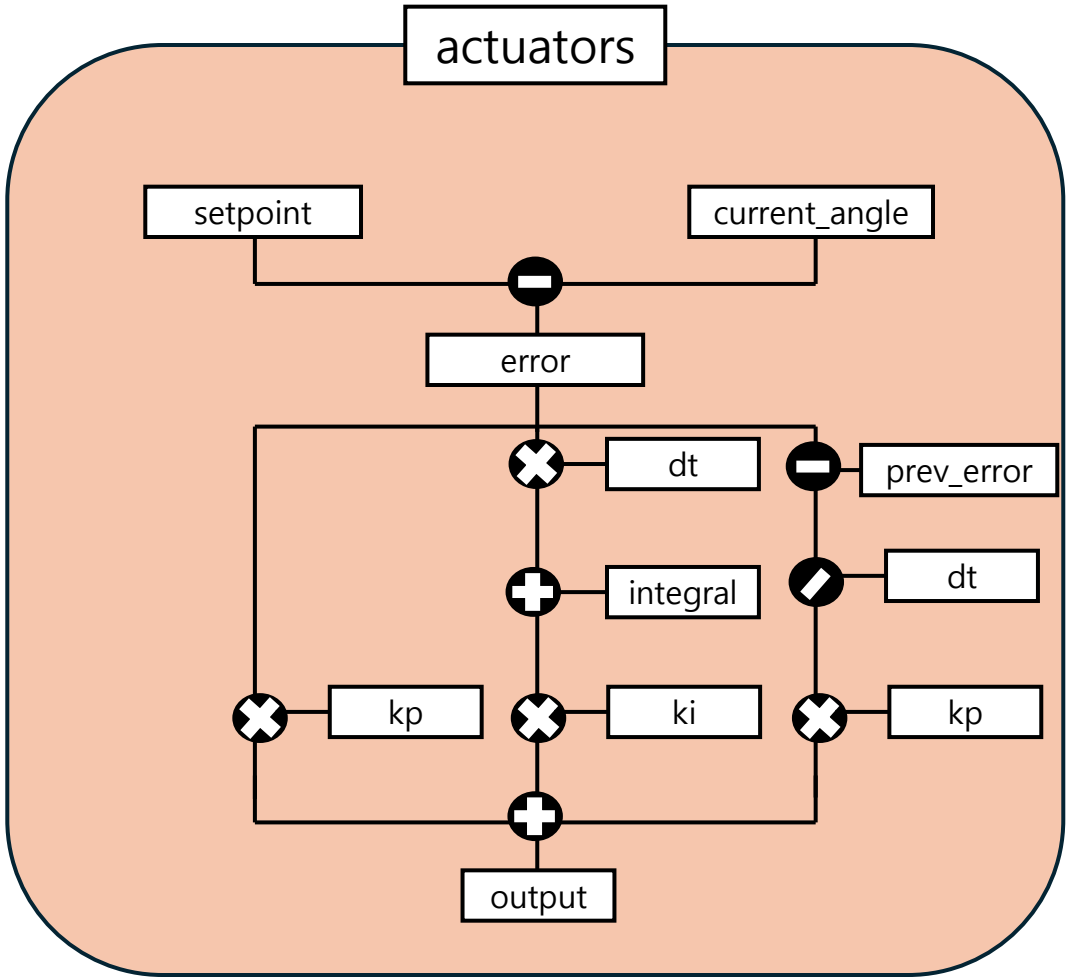
parallel

Short Time

ESCA
Embedded Systems & Computer Architecture Labs

PID Control

```c
struct pid {
    double kp;
    double ki;
    double kd;
    double setpoint;
    double integral, prev_error;
    double current_angle;
    double prev_time;
};

double computePID(struct pid *actuator){
        double current_time = millis();                //get current time
        double dt = (double)(current_time - actuator->prev_time);//compute time elapsed from previous computation

        double error = actuator->setpoint - actuator->current_angle;       // determine error
        actuator->integral += error * dt;                 // compute integral
        double deravitive = (error - actuator->prev_error)/dt;    // compute derivative

        double output = actuator->kp * error
        + actuator->ki * actuator->integral
        + actuator->kd * deravitive; //PID output

        actuator->prev_error = error;                     //remember current error
        actuator->prev_time = current_time;               //remember current time

        return output;                          //have function return the PID output
}
```



ESCA
Embedded Systems & Computer Architecture Labs

PID Control RVV Load/Store - vle, vlse

- vle64.v vd, (rs1) // vd=destination vector, rs1=&base addr

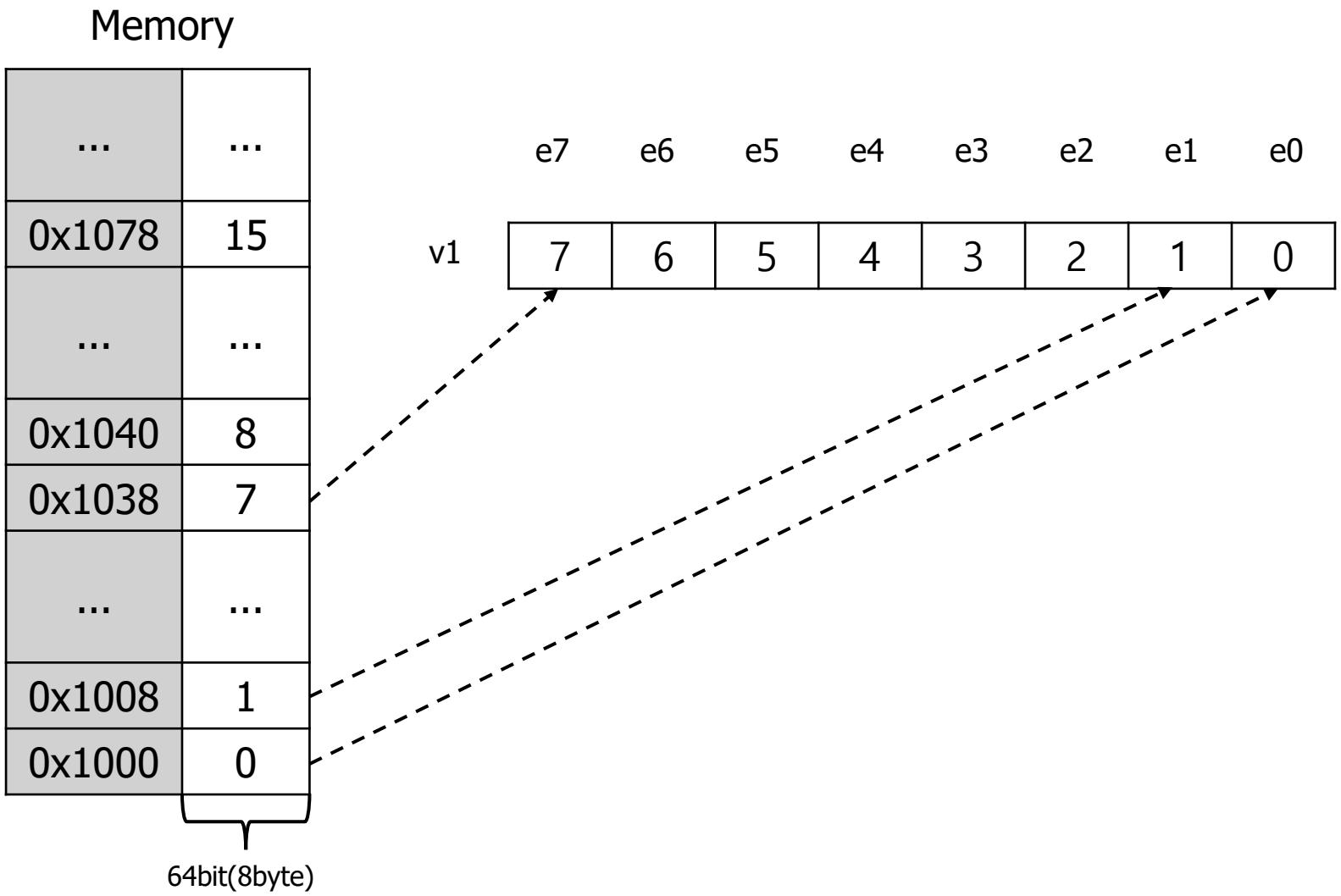
*bring the data until vl

- vlse64.v vd, (rs1), rs2 // rs2=stride

PID Control RVV Load/Store - vle

- vle64.v v1, (a1)

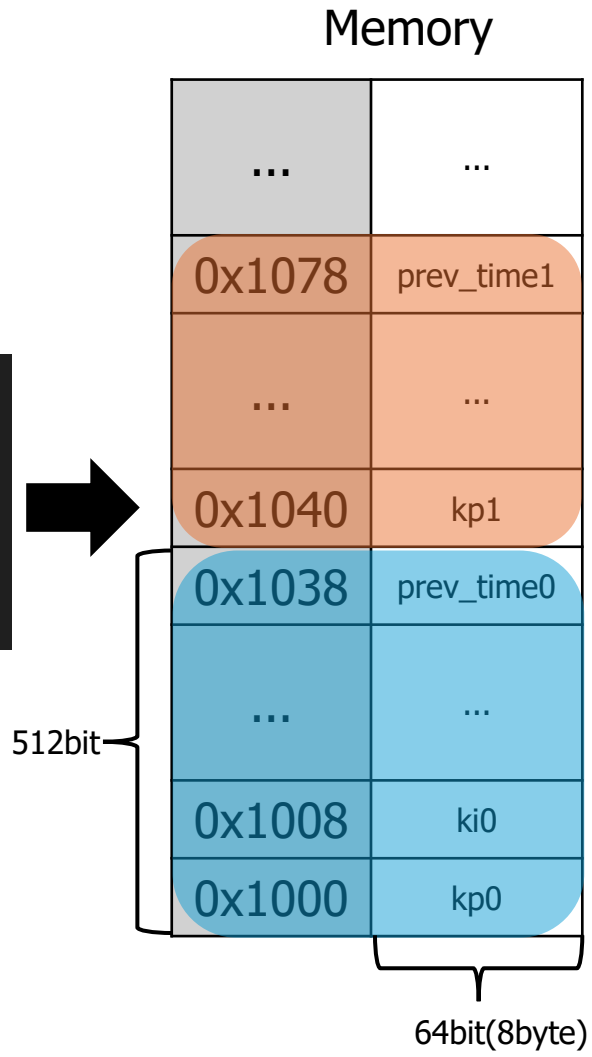
#a1=0x1000(base addr)



ESCA
Embedded Systems & Computer Architecture Labs

PID Control RVV Load/Store - vle

```
struct pid {
    double kp;
    double ki;
    double kd;
    double setpoint;
    double integral, prev_error;
    double current_angle;
    double prev_time;
} actuators[4];
```

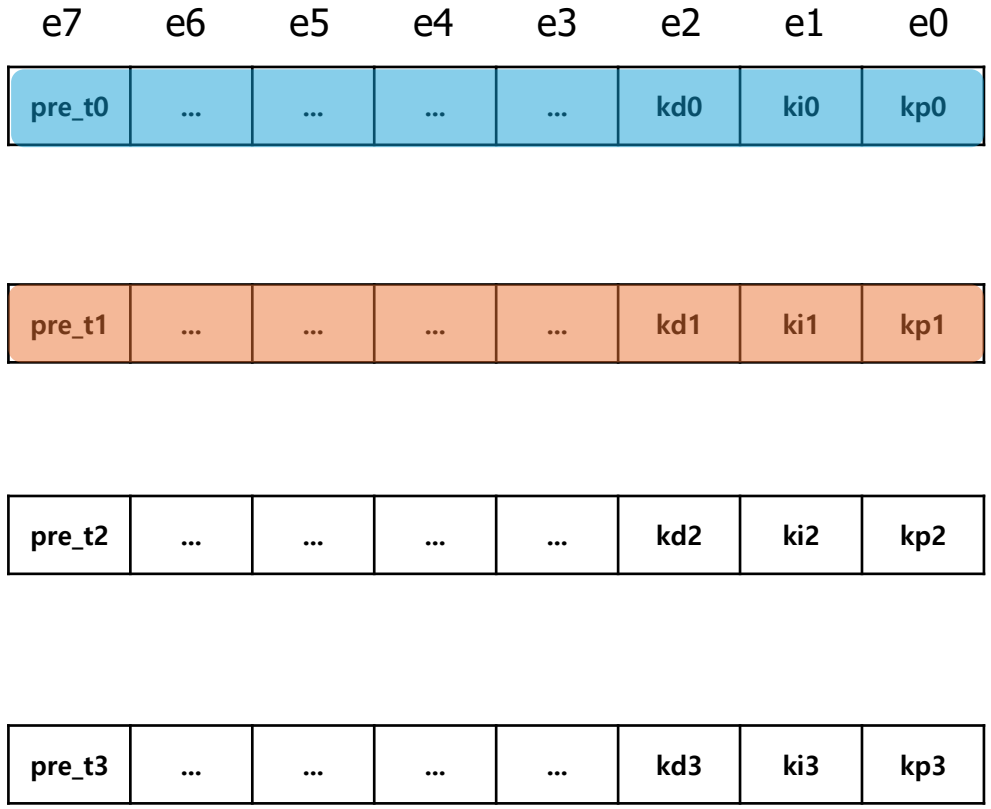


actuators[1]

actuators[0]

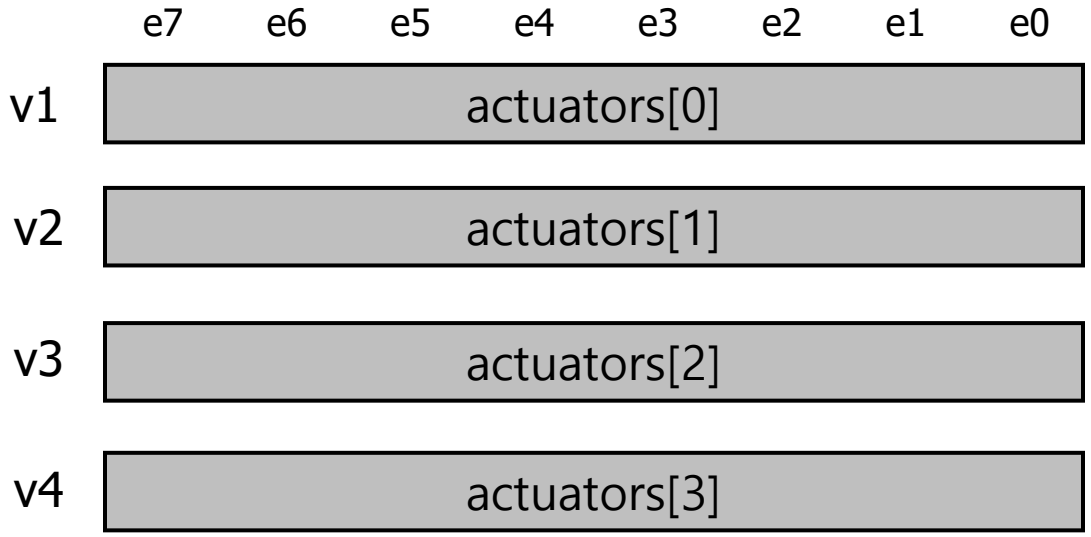
```
vle64.v v1, (a1) #a1=0x1000
addi a1, a1, 64
```

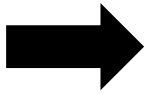
x4

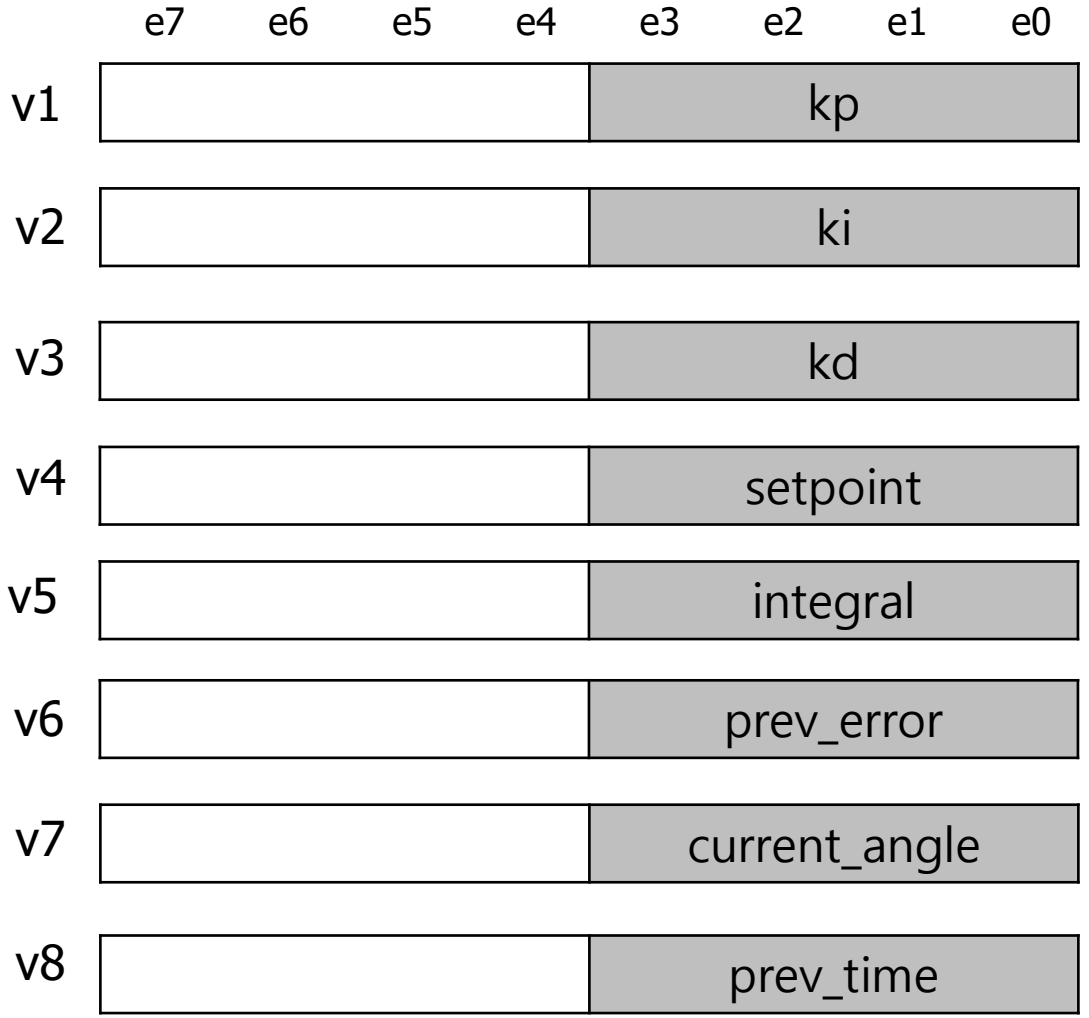


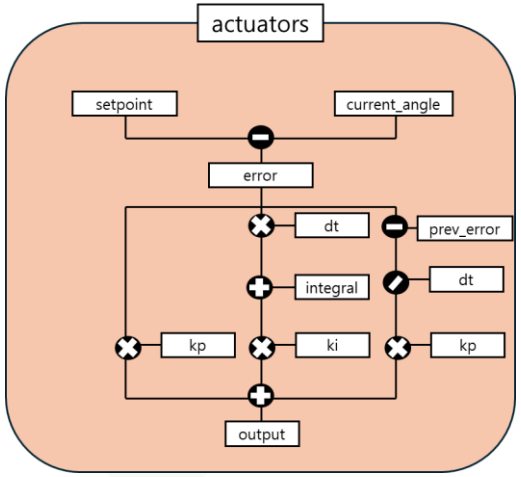
ESCA
Embedded Systems & Computer Architecture Labs

PID Control RVV Load/Store







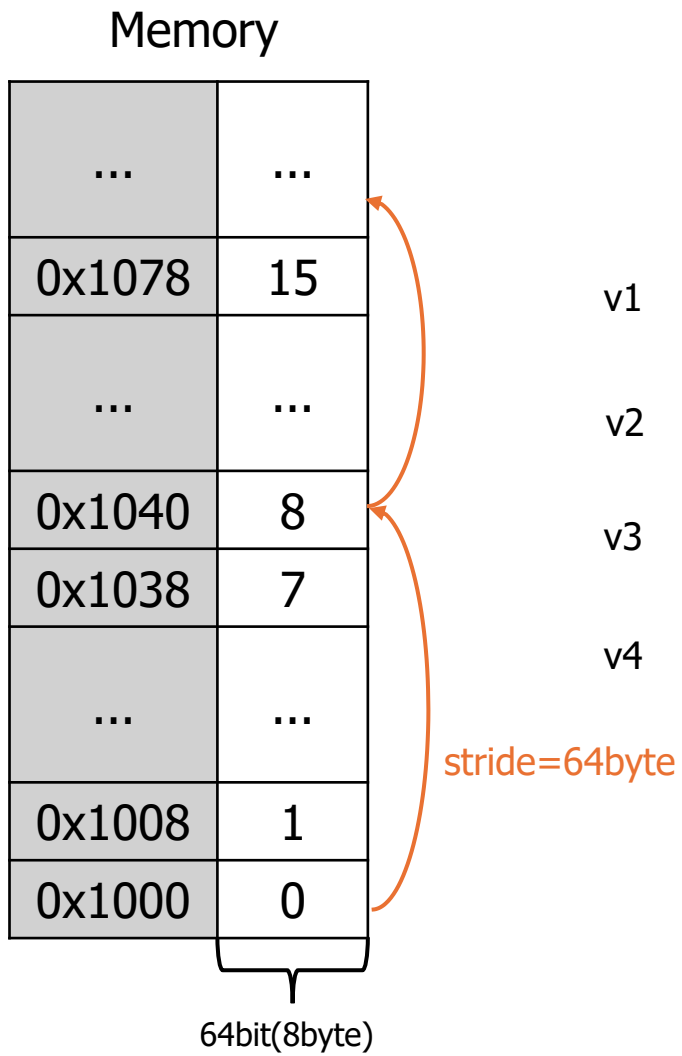


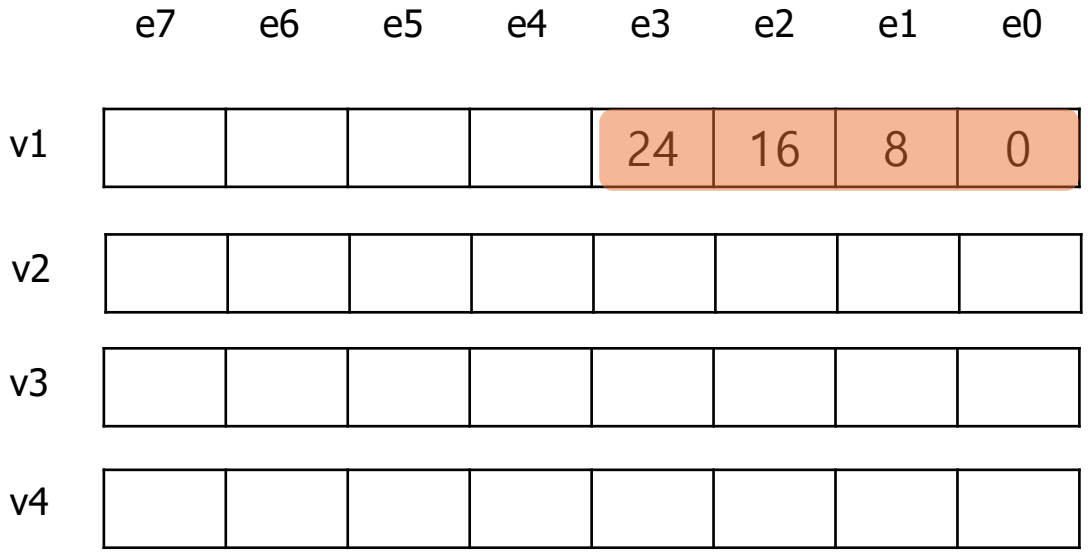
➡ To compute with the fields, each vector must have same field.

RVV Load/Store - vlse

- vsetivli x10, 4, e64, m1, ta, ma
- vlse64.v v1, (a1), a2

#a1=0x1000
#a2=64(byte)



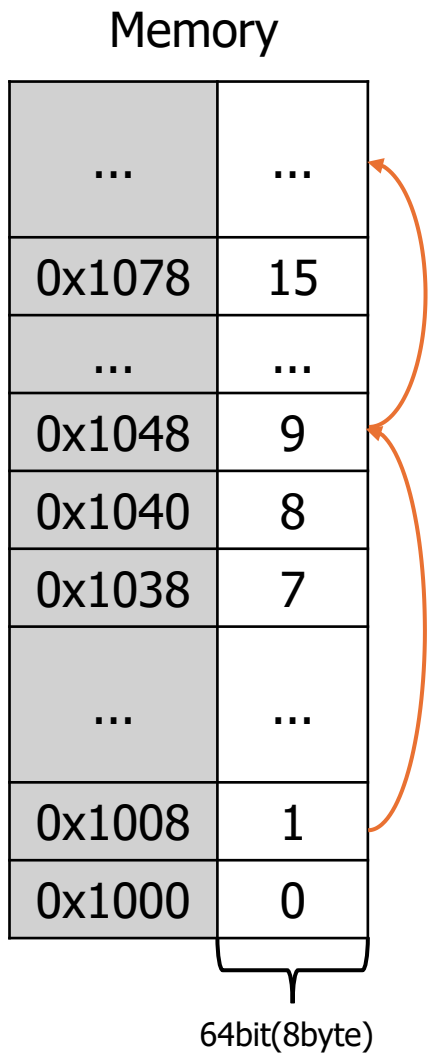


ESCA
Embedded Systems & Computer Architecture Labs

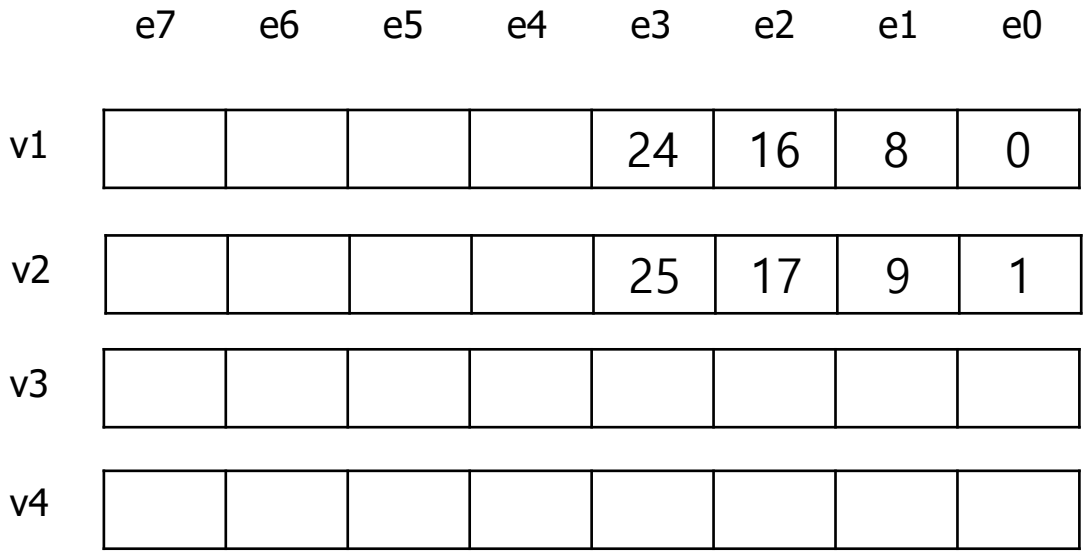
RVV Load/Store - vlse

- addi a1, a1, 8
- vlse64.v v2, (a1), a2

#a1=0x1008
#a2=64



stride=64byte

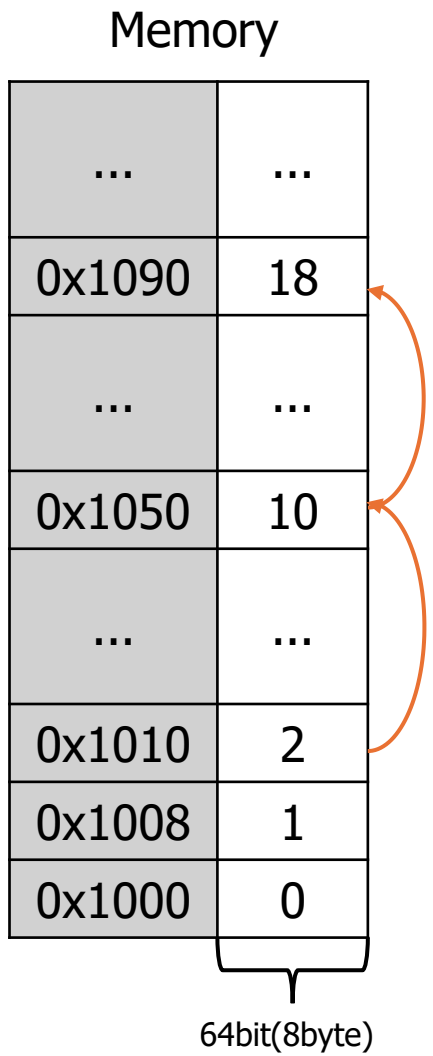


ESCA
Embedded Systems & Computer Architecture Labs

RVV Load/Store - vlse

- addi a1, a1, 8
- vlse64.v v3, (a1), a2

#a1=0x1010
#a2=64

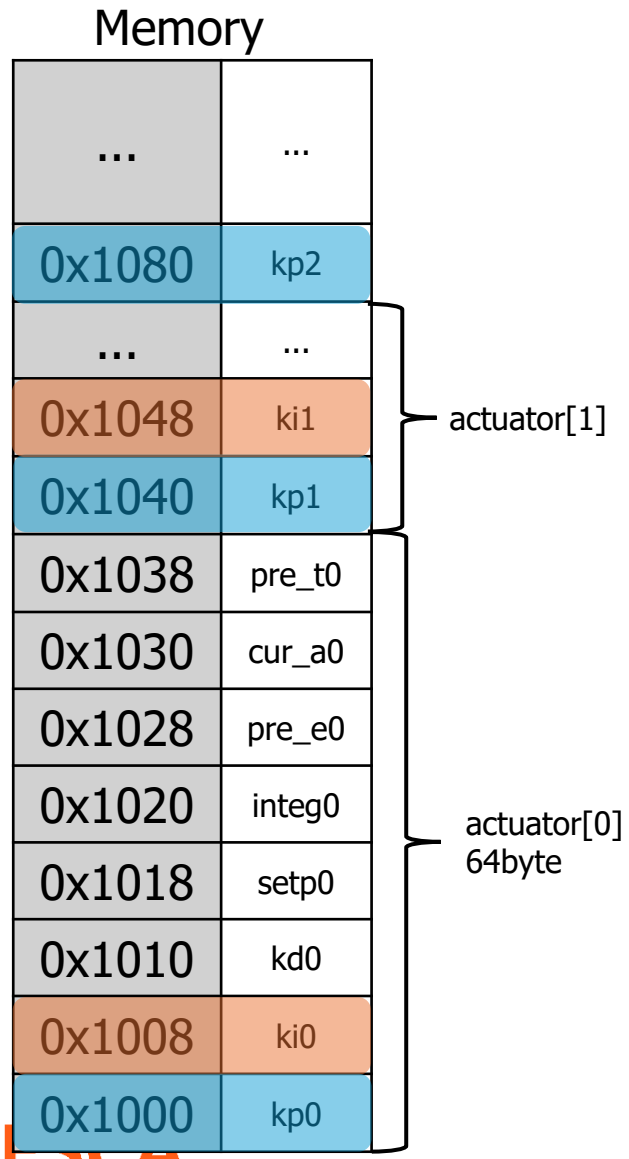


stride=64byte

	e7	e6	e5	e4	e3	e2	e1	e0
v1					24	16	8	0
v2					25	17	9	1
v3					26	18	10	2
v4								

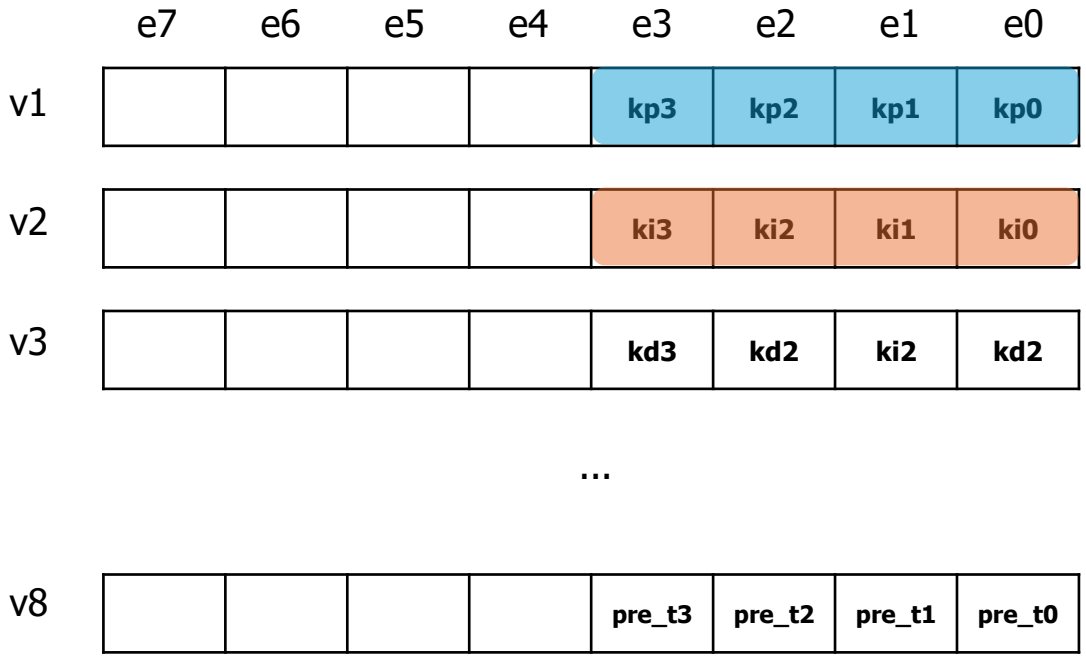
ESCA
Embedded Systems & Computer Architecture Labs

PID Control RVV Load/Store - vlse




```
vlse64.v v1, (a1), a2 #a1=0x1000, a2=64
addi a1, a1, 8
```

x8

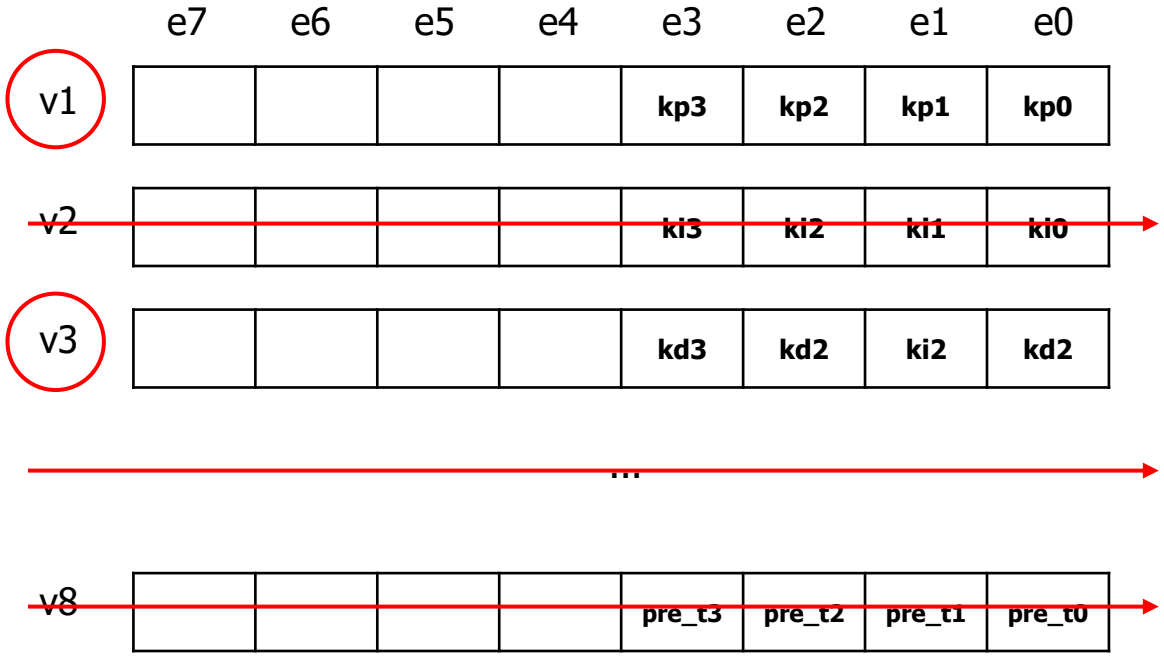


ESCA
Embedded Systems & Computer Architecture Labs

PID Custom Instruction - vpidld/st

- segment-wise load/store is needed in one instruction
- segment-wise masking: Some fields(ex. kp, ki, kd) don't have to update every times. Skipping the vector(field)

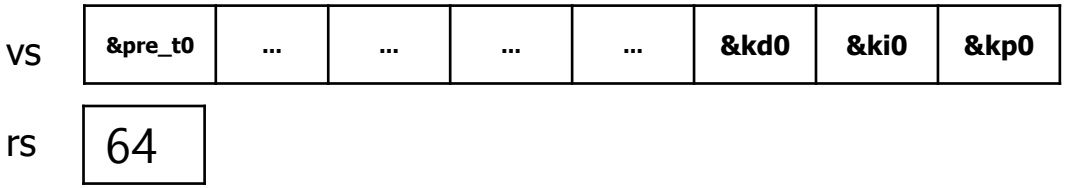
masking vector(v0): 0x101

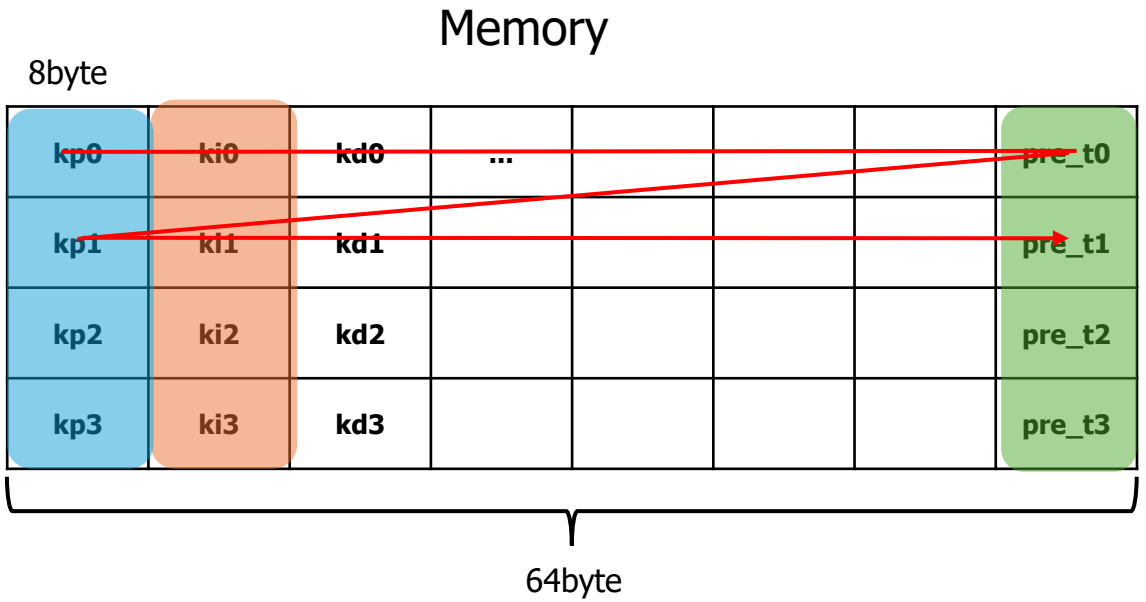


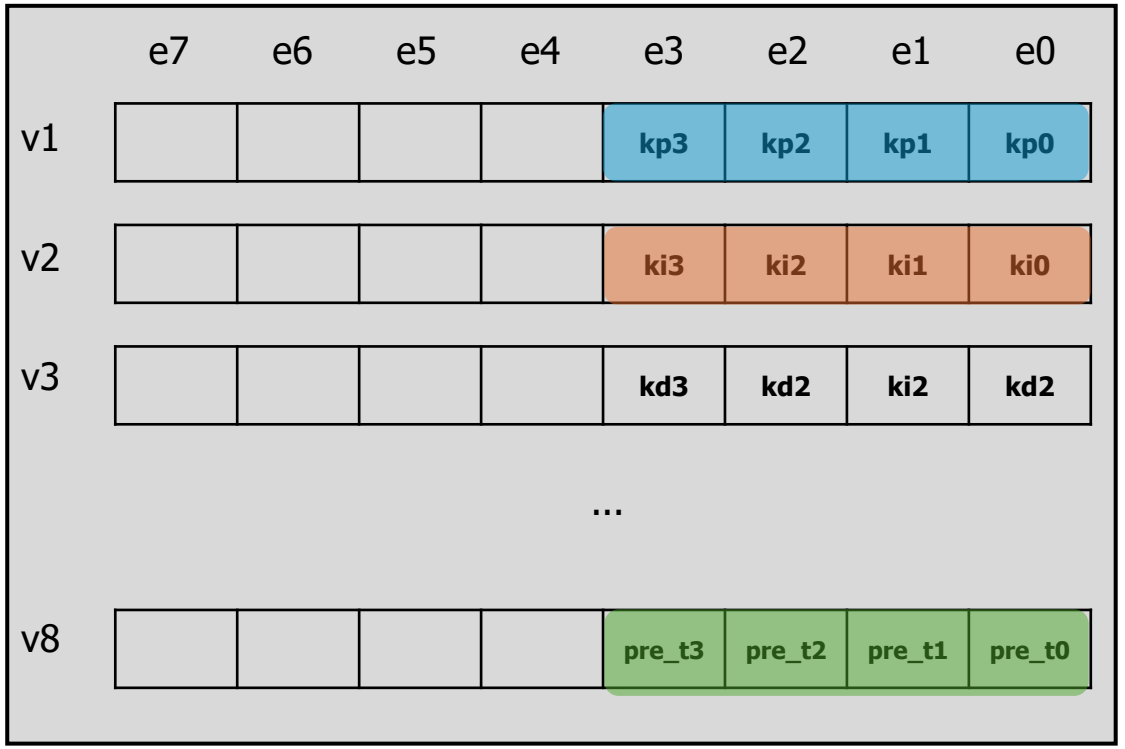
PID Custom Instruction - vpidld/st

vpidldeb<eew>.vx vd, vs, rs // vs=&(actuators[0])

32,64





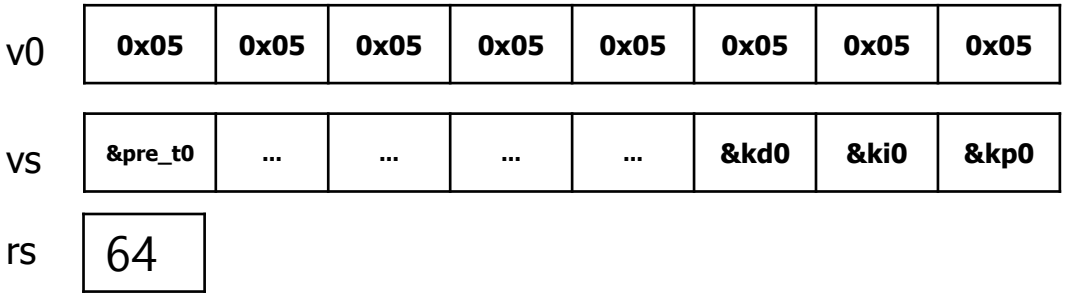


ESCA
Embedded Systems & Computer Architecture Labs

PID Custom Instruction - vpidld/st

vpidldeb<eew>.vx vd, vs, rs, v0.t // masking

32,64

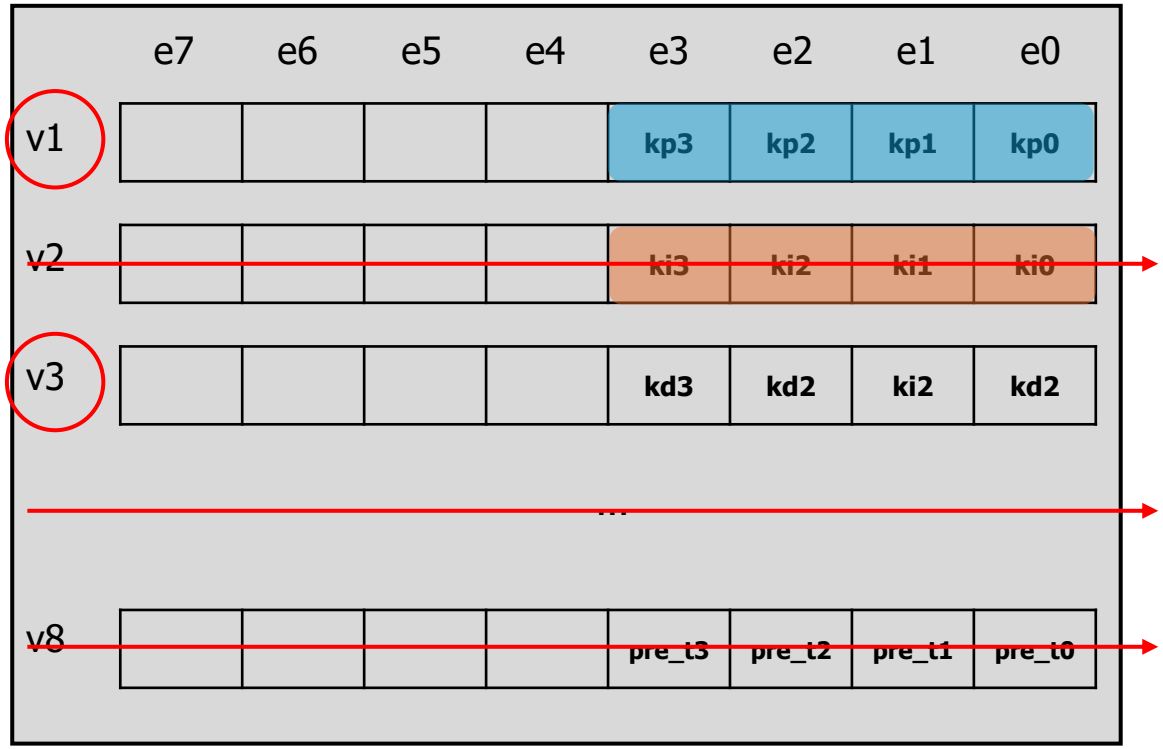


Memory

8byte

kp0	ki0	kd0	...				pre_t0
kp1	ki1	kd1					pre_t1
kp2	ki2	kd2					pre_t2
kp3	ki3	kd3					pre_t3

64byte

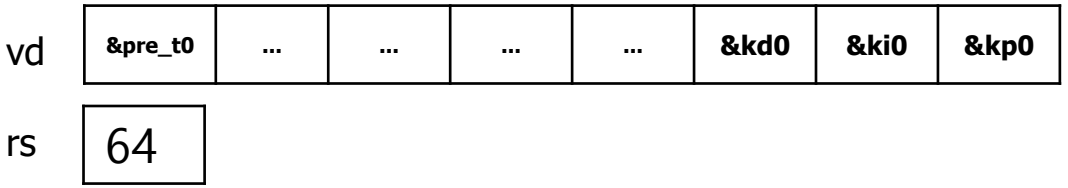


ESCA
Embedded Systems & Computer Architecture Labs

PID Custom Instruction - vpidld/st

vpidsteb<eew>.vx vs, vd, rs, v0.t

32,64

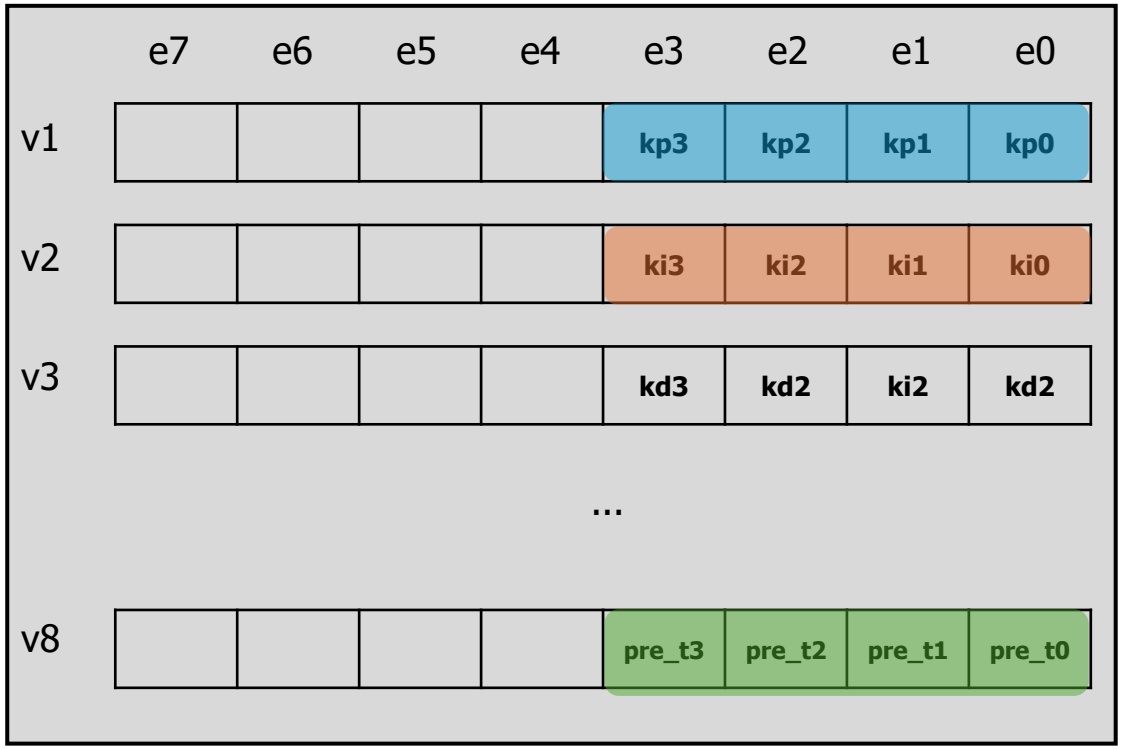


Memory

8byte

kp0	ki0	kd0	...				pre_t0
kp1	ki1	kd1					pre_t1
kp2	ki2	kd2					pre_t2
kp3	ki3	kd3					pre_t3

64byte



ESCA
Embedded Systems & Computer Architecture Labs

PID Custom Instruction - vpidld/st

```
vpidldeb<eew>.vx vd, vs, rs, v0.t
vpidsteb<eew>.vx vs, vd, rs, v0.t
```

1. Definite the Custom Instruction through RISCV-OPCODES

2. Modifying the RISCV Compiler(add the custom instruction)

3. Modifying the SPIKE simulator (add the custom instruction)

*using tool: riscv-gnu-toolchain, riscv-opcodes, riscv-isa-sim

PID Custom Instruction - vpidld/st

1. Definite the Custom Instruction through RISCV-OPCODES

~/riscv-gnu-toolchains/riscv-opcodes/extensions/rv_v

```
453   # Custom Instruction
454   # PID Load/Store          funct6                              funct3        opcode
455   vpidldeb64.vx    31..26=0b010001 vm vs2 rs1 14..12=0x6 vd 6..0=0x57
456   vpidldeb32.vx    31..26=0b010101 vm vs2 rs1 14..12=0x6 vd 6..0=0x57
457   vpidsteb64.vx    31..26=0b010011 vm vs2 rs1 14..12=0x6 vd 6..0=0x57
458   vpidsteb32.vx    31..26=0b010110 vm vs2 rs1 14..12=0x6 vd 6..0=0x57
```

~/riscv-gnu-toolchains/riscv-opcodes/encoding.out.h

```
2144  #define MATCH_VPIDLDEB32_VX 0x54006057
2145  #define MASK_VPIDLDEB32_VX 0xfc00707f
2146  #define MATCH_VPIDLDEB64_VX 0x44006057
2147  #define MASK_VPIDLDEB64_VX 0xfc00707f
2148  #define MATCH_VPIDSTEB32_VX 0x58006057
2149  #define MASK_VPIDSTEB32_VX 0xfc00707f
2150  #define MATCH_VPIDSTEB64_VX 0x4c006057
2151  #define MASK_VPIDSTEB64_VX 0xfc00707f
```

```
3842  DECLARE_INSN(vpidldeb32_vx, MATCH_VPIDLDEB32_VX, MASK_VPIDLDEB32_VX)
3843  DECLARE_INSN(vpidldeb64_vx, MATCH_VPIDLDEB64_VX, MASK_VPIDLDEB64_VX)
3844  DECLARE_INSN(vpidsteb32_vx, MATCH_VPIDSTEB32_VX, MASK_VPIDSTEB32_VX)
3845  DECLARE_INSN(vpidsteb64_vx, MATCH_VPIDSTEB64_VX, MASK_VPIDSTEB64_VX)
```

PID Custom Instruction - vpidld/st

2. Modifying the assembler, disassembler

~/riscv-gnu-toolchain/binutils/include/opcode/riscv-opc.h

```
2144    #define MATCH_VPIDLDEB32_VX 0x54006057
2145    #define MASK_VPIDLDEB32_VX 0xfc00707f
2146    #define MATCH_VPIDLDEB64_VX 0x44006057
2147    #define MASK_VPIDLDEB64_VX 0xfc00707f
2148    #define MATCH_VPIDSTEB32_VX 0x58006057
2149    #define MASK_VPIDSTEB32_VX 0xfc00707f
2150    #define MATCH_VPIDSTEB64_VX 0x4c006057
2151    #define MASK_VPIDSTEB64_VX 0xfc00707f
```

```
3842    DECLARE_INSN(vpidldeb32_vx, MATCH_VPIDLDEB32_VX, MASK_VPIDLDEB32_VX)
3843    DECLARE_INSN(vpidldeb64_vx, MATCH_VPIDLDEB64_VX, MASK_VPIDLDEB64_VX)
3844    DECLARE_INSN(vpidsteb32_vx, MATCH_VPIDSTEB32_VX, MASK_VPIDSTEB32_VX)
3845    DECLARE_INSN(vpidsteb64_vx, MATCH_VPIDSTEB64_VX, MASK_VPIDSTEB64_VX)
```

~/riscv-gnu-toolchain/binutils/opcodes/riscv-opc.c

```
{"vpidldeb32.vx", 0, INSN_CLASS_V, "Vd,Vt,sVm", MATCH_VPIDLDEB32_VX, MASK_VPIDLDEB32_VX, match_opcode, INSN_DREF},
{"vpidldeb64.vx", 0, INSN_CLASS_V, "Vd,Vt,sVm", MATCH_VPIDLDEB64_VX, MASK_VPIDLDEB64_VX, match_opcode, INSN_DREF|INSN_V_EEW64},
{"vpidsteb32.vx", 0, INSN_CLASS_V, "Vd,Vt,sVm", MATCH_VPIDSTEB32_VX, MASK_VPIDSTEB32_VX, match_opcode, INSN_DREF},
{"vpidsteb64.vx", 0, INSN_CLASS_V, "Vd,Vt,sVm", MATCH_VPIDSTEB64_VX, MASK_VPIDSTEB64_VX, match_opcode, INSN_DREF|INSN_V_EEW64},
```

PID Custom Instruction - vpidld/st

3. Modifying the SPIKE simulator

~/riscv-gnu-toolchain/riscv-isa-sim/riscv/v_ext_macros.h

```
#define VPID_LOAD(elt_width, is_mask_ldst) \
  const reg_t vl = is_mask_ldst ? ((P.VU.vl->read() + 7) / 8) : P.VU.vl->read(); \
  const reg_t stride = RS1; \
  const reg_t rd_num = insn.rd(); \
  const reg_t rs2_num = insn.rs2(); \
  const reg_t nf = PID_SEGMENT_NUM; \
  VI_CHECK_LD_INDEX(elt_width); \
  VPID_DUPLICATE_VREG(rs2_num, elt_width); \
  for (reg_t i = 0; i < vl; ++i) { \
    VI_STRIP(i); \
    P.VU.vstart->write(i); \
    for (reg_t fn = 0; fn < nf; ++fn) { \
      VI_SEGMENT_SKIP \
      switch(P.VU.vsew) { \
        case e8: { \
          P.VU.elt<uint8_t>(rd_num + fn, i, true) = MMU.load<uint8_t>(index[fn] + (stride * i)); \
          break; } \
        case e16: { \
          P.VU.elt<uint16_t>(rd_num + fn, i, true) = MMU.load<uint16_t>(index[fn] + (stride * i)); \
          break; } \
        case e32: { \
          P.VU.elt<uint32_t>(rd_num + fn, i, true) = MMU.load<uint32_t>(index[fn] + (stride * i)); \
          break; } \
        case e64: { \
          P.VU.elt<uint64_t>(rd_num + fn, i, true) = MMU.load<uint64_t>(index[fn] + (stride * i)); \
          break; } \
      } \
    } \
  } \
  P.VU.vstart->write(0);
```

SPIKE – ISA Simulation

Spike, the RISC-V ISA Simulator, implements a functional model of one or more RISC-V harts.

Spike supports the following RISC-V ISA features:

- RV32I and RV64I base ISAs, v2.1
- RV32E and RV64E base ISAs, v1.9
- Zifencei extension, v2.0
- Zicsr extension, v2.0
- Zicntr extension, v2.0
- M extension, v2.0
- A extension, v2.1
- B extension, v1.0
- F extension, v2.2
- D extension, v2.2
- Q extension, v2.2
- C extension, v2.0
- Zbkb, Zbkc, Zbkx, Zknd, Zkne, Zknh, Zksed, Zksh scalar cryptography extensions (Zk, Zkn, and Zks groups), v1.0
- Zkr virtual entropy source emulation, v1.0
- V extension, v1.0 (*requires a 64-bit host*)

SPIKE – ISA Simulation

```
(base) jun311k@Junseok:~/riscv-gnu-toolchain/riscv-isa-sim/test$ riscv64-unknown-elf-gcc -march=rv64gcv -o test test.c
(base) jun311k@Junseok:~/riscv-gnu-toolchain/riscv-isa-sim/test$ riscv64-unknown-elf-objdump -D -S test > test.dump
```

- c code → elf → spike

option file

```
(base) jun311k@Junseok:~/riscv-gnu-toolchain/riscv-isa-sim/test$ spike -d --isa=rv64gcv_zvl512b_zve64d pk test
(spike) vreg 0 0
VLEN=512 bits; ELEN=64 bits
```

```
(base) jun311k@Junseok:~/riscv-gnu-toolchain/riscv-isa-sim/test$ spike -d --isa=rv64gcv_zvl8192b_zve32f pk test1_pid_ld_st
error: bad --isa option 'rv64gcv_zvl8192b_zve32f'. Spike does not currently support VLEN > 4096b
```

- rv64→RISC-V 64bit architecture(register value, memory address value is 64bit)
- g(General Purpose Extension Set): IMAFD(Integer, Multiplier, Atomic, Single FP, Double FP)
- c(Compressed Instruction Extension): Compress the instruction into 16bit
- v(Vector Extension)
- zvl: setting VLEN(32b~4096b)
- zve: V extension for Embedded processors. setting XLEN(32x, 32f, 64x, 64f, 64d)

*In spike the maximum ELEN is provided.(64bit)

SPIKE – ISA Simulation

```
(base) jun311k@Junseok:~/riscv-gnu-toolchain/riscv-isa-sim/test$ spike -d --isa=rv64gcv_zvl512b_zve64d pk test1_pid_ld_st
(spike) reg 0
zero: 0x0000000000000000  ra: 0x0000000000000000  sp: 0x0000000000000000  gp: 0x0000000000000000
  tp: 0x0000000000000000  t0: 0x0000000000000000  t1: 0x0000000000000000  t2: 0x0000000000000000
  s0: 0x0000000000000000  s1: 0x0000000000000000  a0: 0x0000000000000000  a1: 0x0000000000000000
  a2: 0x0000000000000000  a3: 0x0000000000000000  a4: 0x0000000000000000  a5: 0x0000000000000000
  a6: 0x0000000000000000  a7: 0x0000000000000000  s2: 0x0000000000000000  s3: 0x0000000000000000
  s4: 0x0000000000000000  s5: 0x0000000000000000  s6: 0x0000000000000000  s7: 0x0000000000000000
  s8: 0x0000000000000000  s9: 0x0000000000000000 s10: 0x0000000000000000 s11: 0x0000000000000000
  t3: 0x0000000000000000  t4: 0x0000000000000000  t5: 0x0000000000000000  t6: 0x0000000000000000
```

```
(base) jun311k@Junseok:~/riscv-gnu-toolchain/riscv-isa-sim/test$ spike -d --isa=rv64gcv_zvl512b_zve
64d pk test1_pid_ld_st
(spike) vreg 0 v0
VLEN=512 bits; ELEN=64 bits
v0  : [7]: 0x0000000000000000  [6]: 0x0000000000000000  [5]: 0x0000000000000000  [4]: 0x00000000000
00000  [3]: 0x0000000000000000  [2]: 0x0000000000000000  [1]: 0x0000000000000000  [0]: 0x0000000000
000000
```

- vlen=512b, elen=64bit (basically # of elements=512/64=8)

SPIKE – ISA Simulation

```
00000000000101d8 <main>:
   101d8:   1141                addi  sp,sp,-16
   101da:   e406                sd ra,8(sp)
   101dc:   e022                sd s0,0(sp)
   101de:   0800                addi  s0,sp,16
   101e0:   67c9                lui   a5,0x12
   101e2:   77078713            addi  a4,a5,1904 # 12770 <actuators>
   101e6:   87818693            addi  a3,gp,-1928 # 12970 <vec6>
   101ea:   22018813            addi  a6,gp,544 # 13318 <current_time>
   101ee:   0d8072d7            vsetvli  t0,zero,e64,m1,ta,ma
   101f2:   853a                mv a0,a4
   101f4:   85b6                mv a1,a3
   101f6:   8342                mv t1,a6
   101f8:   00030603            lb a2,0(t1) # 101ce <frame_dummy+0x10>
   101fc:   0205f007            vle64.v  v0,(a1)
   10200:   9601b057            vsll.vi  v0,v0,3
   10204:   020540d7            vadd.vx  v1,v0,a0
   10208:   0d8072d7            vsetvli  t0,zero,e64,m1,ta,ma
   1020c:   04000793            li a5,64
   10210:   eaf57607            vlsseg8e64.v   v12,(a0),a5
   10214:   4781                li a5,0
   10216:   853e                mv a0,a5
   10218:   60a2                ld ra,8(sp)
   1021a:   6402                ld s0,0(sp)
   1021c:   0141                addi  sp,sp,16
   1021e:   8082                ret
```

```
(spike) until pc 0 10210
(spike)
core   0: 0x0000000000010210 (0xeaf57607) vlsseg8e64.v v12, (a0), a5
```

- until pc 0 10210

→execute from pc=0x0 to pc=0x10210

SPIKE – ISA Simulation

```
(spike) until pc 0 10206
(spike)
core   0: 0x0000000000010206 (0x4617e257) vpidldeb64.vx v4, v1, a5
(spike) vreg 0
VLEN=512 bits; ELEN=64 bits
```

actuators

kp	ki						pre_t
0x00	0x01	...	...	...	...	...	0x07
0x10	0x11	...	...				0x17
0x20	0x21	...	...				0x27
0x30	0x31	...	...				0x37

```
v4  : [7]: 0x0000000000000000  [6]: 0x0000000000000000  [5]: 0x0000000000000000  [4]: 0x0000000000000000
00  [3]: 0x0000000000000030  [2]: 0x0000000000000020  [1]: 0x0000000000000010  [0]: 0x0000000000000000

v5  : [7]: 0x0000000000000000  [6]: 0x0000000000000000  [5]: 0x0000000000000000  [4]: 0x0000000000000000
00  [3]: 0x0000000000000031  [2]: 0x0000000000000021  [1]: 0x0000000000000011  [0]: 0x0000000000000001

v6  : [7]: 0x0000000000000000  [6]: 0x0000000000000000  [5]: 0x0000000000000000  [4]: 0x0000000000000000
00  [3]: 0x0000000000000032  [2]: 0x0000000000000022  [1]: 0x0000000000000012  [0]: 0x0000000000000002

v7  : [7]: 0x0000000000000000  [6]: 0x0000000000000000  [5]: 0x0000000000000000  [4]: 0x0000000000000000
00  [3]: 0x0000000000000033  [2]: 0x0000000000000023  [1]: 0x0000000000000013  [0]: 0x0000000000000003

v8  : [7]: 0x0000000000000000  [6]: 0x0000000000000000  [5]: 0x0000000000000000  [4]: 0x0000000000000000
00  [3]: 0x0000000000000034  [2]: 0x0000000000000024  [1]: 0x0000000000000014  [0]: 0x0000000000000004

v9  : [7]: 0x0000000000000000  [6]: 0x0000000000000000  [5]: 0x0000000000000000  [4]: 0x0000000000000000
00  [3]: 0x0000000000000035  [2]: 0x0000000000000025  [1]: 0x0000000000000015  [0]: 0x0000000000000005

v10 : [7]: 0x0000000000000000  [6]: 0x0000000000000000  [5]: 0x0000000000000000  [4]: 0x0000000000000000
00  [3]: 0x0000000000000036  [2]: 0x0000000000000026  [1]: 0x0000000000000016  [0]: 0x0000000000000006

v11 : [7]: 0x0000000000000000  [6]: 0x0000000000000000  [5]: 0x0000000000000000  [4]: 0x0000000000000000
00  [3]: 0x0000000000000037  [2]: 0x0000000000000027  [1]: 0x0000000000000017  [0]: 0x0000000000000007
```

ESCA
Embedded Systems & Computer Architecture Labs

SPIKE – ISA Simulation

- vlse64.v

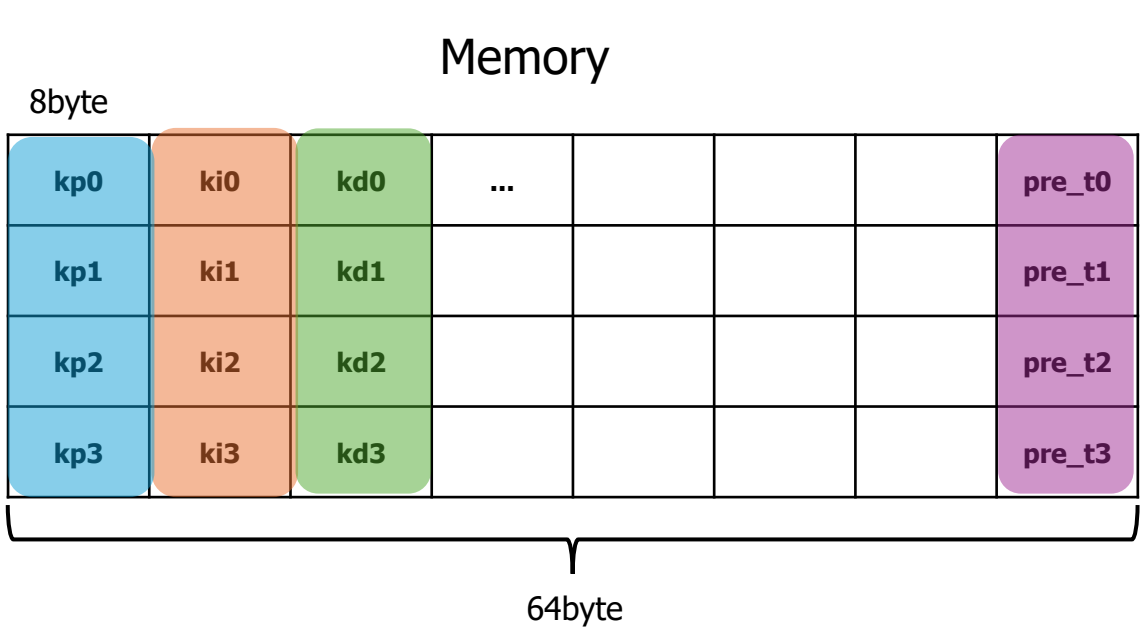
```
00000000000101d8 <main>:
   101d8:   1141                addi  sp,sp,-16
   101da:   e406                sd ra,8(sp)
   101dc:   e022                sd s0,0(sp)
   101de:   0800                addi  s0,sp,16
   101e0:   67cd                lui   a5,0x13
   101e2:   80078793            addi  a5,a5,-2048 # 12800 <actuators>
   101e6:   0d8072d7            vsetvli  t0,zero,e64,m1,ta,ma
   101ea:   853e                mv a0,a5
   101ec:   85aa                mv a1,a0
   101ee:   04000713            li a4,64
   101f2:   0ae5f087            vlse64.v v1,(a1),a4
   101f6:   05a1                addi  a1,a1,8
   101f8:   0ae5f107            vlse64.v v2,(a1),a4
   101fc:   05a1                addi  a1,a1,8
   101fe:   0ae5f187            vlse64.v v3,(a1),a4
   10202:   05a1                addi  a1,a1,8
   10204:   0ae5f207            vlse64.v v4,(a1),a4
   10208:   05a1                addi  a1,a1,8
   1020a:   0ae5f287            vlse64.v v5,(a1),a4
   1020e:   05a1                addi  a1,a1,8
   10210:   0ae5f307            vlse64.v v6,(a1),a4
   10214:   05a1                addi  a1,a1,8
   10216:   0ae5f387            vlse64.v v7,(a1),a4
   1021a:   05a1                addi  a1,a1,8
   1021c:   0ae5f407            vlse64.v v8,(a1),a4
   10220:   4781                li a5,0
   10222:   853e                mv a0,a5
   10224:   60a2                ld ra,8(sp)
   10226:   6402                ld s0,0(sp)
   10228:   0141                addi  sp,sp,16
   1022a:   8082                ret
```

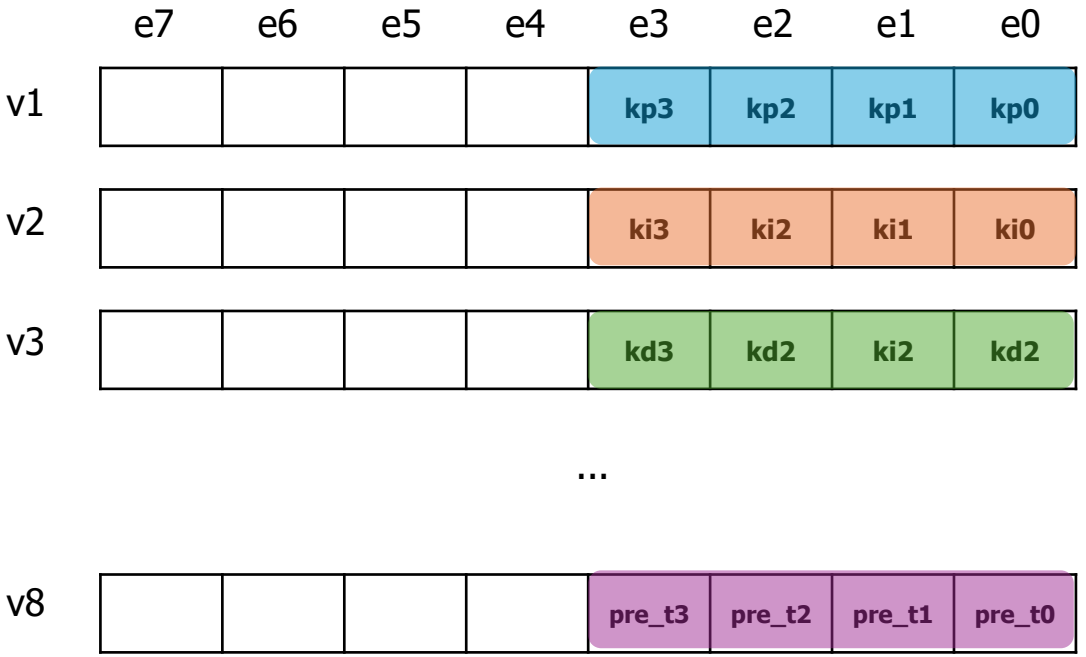
- vpidldeb64.vx

```
00000000000101d8 <main>:
   101d8:   1141                addi  sp,sp,-16
   101da:   e406                sd ra,8(sp)
   101dc:   e022                sd s0,0(sp)
   101de:   0800                addi  s0,sp,16
   101e0:   67d1                lui   a5,0x14
   101e2:   80078713            addi  a4,a5,-2048 # 13800 <actuators>
   101e6:   86018513            addi  a0,gp,-1952 # 13900 <index>
   101ea:   0d8072d7            vsetvli  t0,zero,e64,m1,ta,ma
   101ee:   85ba                mv a1,a4
   101f0:   862a                mv a2,a0
   101f2:   02067007            vle64.v  v0,(a2)
   101f6:   9601b057            vsll.vi  v0,v0,3
   101fa:   0205c0d7            vadd.vx  v1,v0,a1
   101fe:   04000793            li a5,64
   10202:   cd8272d7            vsetivli t0,4,e64,m1,ta,ma
   10206:   4617e257            vpidldeb64.vx  v4,v1,a5
   1020a:   4e17e257            vpidsteb64.vx  v4,v1,a5
   1020e:   67c9                lui   a5,0x12
   10210:   64078513            addi  a0,a5,1600 # 12640 <__errno+0x6>
   10214:   37c000ef            jal   10590 <puts>
   10218:   4781                li a5,0
   1021a:   853e                mv a0,a5
   1021c:   60a2                ld ra,8(sp)
   1021e:   6402                ld s0,0(sp)
   10220:   0141                addi  sp,sp,16
   10222:   8082                ret
```

RVV Load/Store - vlsseg<nf>e<eew>.v

vlsseg8e64.v v1, (a1), a2 #a1=0x1000, a2=64 x1





ESCA
Embedded Systems & Computer Architecture Labs

SPIKE – ISA Simulation

```
(base) jun311k@Junseok:~/riscv-gnu-toolchain/riscv-isa-sim/test$ spike -d --isa=rv64gcv_zvl256b_zve32d
 pk test1_pid_ld_st
(spike) until pc 0 101f4
(spike)
core   0: 0x00000000000101f4 (0xeaf57087) vlsseg8e64.v v1, (a0), a5
(spike) vreg 0
VLEN=256 bits; ELEN=64 bits
v1  : [3]: 0x0000000000000030  [2]: 0x0000000000000020  [1]: 0x0000000000000010  [0]: 0x0000000000000000
v2  : [3]: 0x0000000000000031  [2]: 0x0000000000000021  [1]: 0x0000000000000011  [0]: 0x0000000000000001
v3  : [3]: 0x0000000000000032  [2]: 0x0000000000000022  [1]: 0x0000000000000012  [0]: 0x0000000000000002
v4  : [3]: 0x0000000000000033  [2]: 0x0000000000000023  [1]: 0x0000000000000013  [0]: 0x0000000000000003
v5  : [3]: 0x0000000000000034  [2]: 0x0000000000000024  [1]: 0x0000000000000014  [0]: 0x0000000000000004
v6  : [3]: 0x0000000000000035  [2]: 0x0000000000000025  [1]: 0x0000000000000015  [0]: 0x0000000000000005
v7  : [3]: 0x0000000000000036  [2]: 0x0000000000000026  [1]: 0x0000000000000016  [0]: 0x0000000000000006
v8  : [3]: 0x0000000000000037  [2]: 0x0000000000000027  [1]: 0x0000000000000017  [0]: 0x0000000000000007
```

actuators

kp	ki						pre_t
0x00	0x01	...	...	...	...	...	0x07
0x10	0x11	...	...				0x17
0x20	0x21	...	...				0x27
0x30	0x31	...	...				0x37

ESCA
Embedded Systems & Computer Architecture Labs

SPIKE – ISA Simulation

- vpidldeb64.vx

```
00000000000101d8 <main>:
   101d8:   1141                addi  sp,sp,-16
   101da:   e406                sd ra,8(sp)
   101dc:   e022                sd s0,0(sp)
   101de:   0800                addi  s0,sp,16
   101e0:   67d1                lui   a5,0x14
   101e2:   80078713            addi  a4,a5,-2048 # 13800 <actuators>
   101e6:   86018513            addi  a0,gp,-1952 # 13900 <index>
   101ea:   0d8072d7            vsetvli  t0,zero,e64,m1,ta,ma
   101ee:   85ba                mv a1,a4
   101f0:   862a                mv a2,a0
   101f2:   02067007            vle64.v  v0,(a2)
   101f6:   9601b057            vsll.vi  v0,v0,3
   101fa:   0205c0d7            vadd.vx  v1,v0,a1
   101fe:   04000793            li a5,64
   10202:   cd8272d7            vsetivli t0,4,e64,m1,ta,ma
   10206:   4617e257            vpidldeb64.vx  v4,v1,a5
   1020a:   4e17e257            vpidsteb64.vx  v4,v1,a5
   1020e:   67c9                lui   a5,0x12
   10210:   64078513            addi  a0,a5,1600 # 12640 <__errno+0x6>
   10214:   37c000ef            jal   10590 <puts>
   10218:   4781                li a5,0
   1021a:   853e                mv a0,a5
   1021c:   60a2                ld ra,8(sp)
   1021e:   6402                ld s0,0(sp)
   10220:   0141                addi  sp,sp,16
   10222:   8082                ret
```

- vlsseg8e64.v

```
00000000000101d8 <main>:
   101d8:   1141                addi  sp,sp,-16
   101da:   e406                sd ra,8(sp)
   101dc:   e022                sd s0,0(sp)
   101de:   0800                addi  s0,sp,16
   101e0:   67cd                lui   a5,0x13
   101e2:   80078713            addi  a4,a5,-2048 # 12800 <actuators>
   101e6:   0d8072d7            vsetvli  t0,zero,e64,m1,ta,ma
   101ea:   853a                mv a0,a4
   101ec:   0d8072d7            vsetvli  t0,zero,e64,m1,ta,ma
   101f0:   04000793            li a5,64
   101f4:   eaf57087            vlsseg8e64.v   v1,(a0),a5
   101f8:   4781                li a5,0
   101fa:   853e                mv a0,a5
   101fc:   60a2                ld ra,8(sp)
   101fe:   6402                ld s0,0(sp)
   10200:   0141                addi  sp,sp,16
   10202:   8082                ret
```

Future works - 1

- Other custom instruction - vpid.vv

ex1) vmacc.vv

```
# Integer multiply-add, overwrite addend
vmacc.vv vd, vs1, vs2, vm    # vd[i] = +(vs1[i] * vs2[i]) + vd[i]
vmacc.vx vd, rs1, vs2, vm    # vd[i] = +(x[rs1] * vs2[i]) + vd[i]
```

ex2) FPU

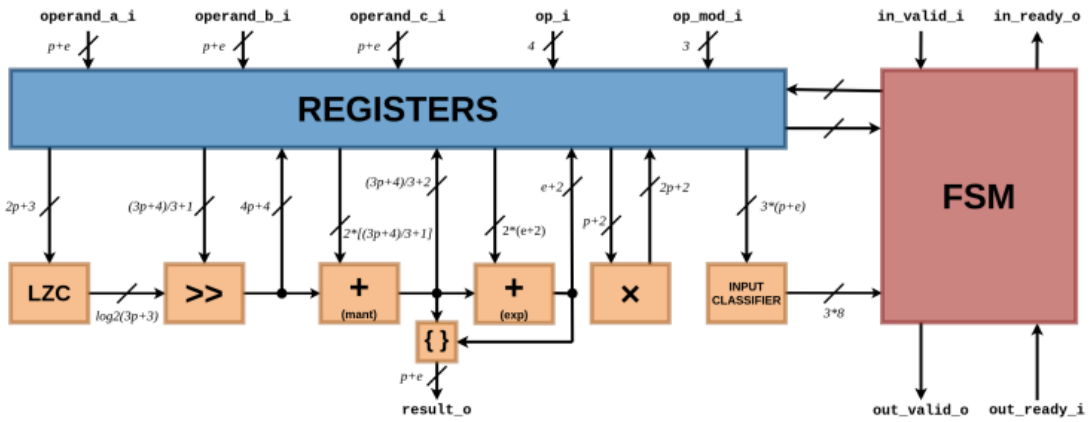


Fig. 1. Tiny-FPU Schematic

- pid computation

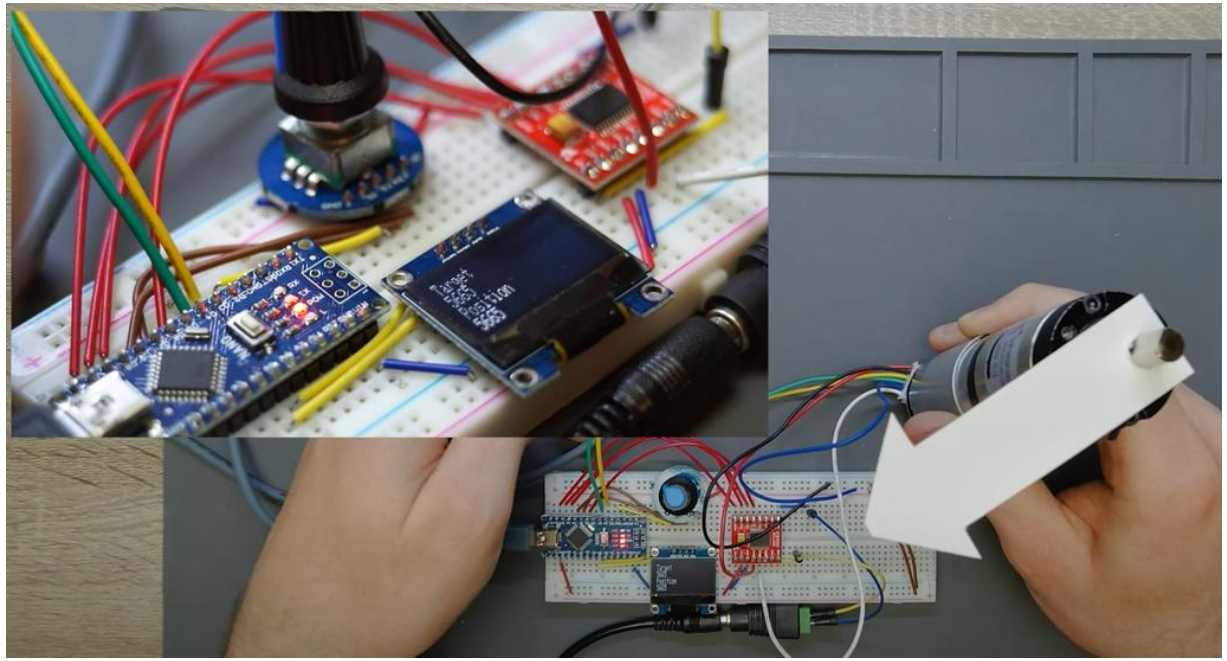
```
; start calculation
vsub.vv v12, v13, v11 ; v12: dt, v13: current_time
vsub.vv v14, v7, v10 ; v14: error
vmacc.vv v8, v14, v12 ; v8: integral
vsub.vv v15, v14, v9; v15: error - prev_error
vdiv.vv v15, v15, v12 ; v15: derivative
vmv.v.i v16, 0 ; v16: output
vmacc.vv v16, v4, v14 ; output += kp*error
vmacc.vv v16, v5, v8 ; output += ki*integral
vmacc.vv v16, v6, v15 ; output +=kd*derivative
```

```
double computeVPID(){
  VPIDLOAD.v
  VPID.vv
  VPIDSTORE.v
}
```

Future works - 2

- Physical experiment Propose

RISCV Scalar/Vector extension Processor FPGA + Encoder + 4 motors
→ Graph



https://www.youtube.com/watch?v=jTIRUXJKMX4

Q&A