PROJECT REPORT

Parsa KafshduziBukani 810102501
Milad Mazaheri 810102585

CA2

Project Description

هدف از این تمرین کامپیوتری پیاده‌سازی Single-Cycle پردازنده‌ی RISC-V با محموعه دستورات زیر است:

- R-Type: add, sub, and, or, slt
- I-Type: lw, addi, ori, slti, jalr
- S-Type: sw
- J-Type: jal
- B-Type: beq, bne
- U-Type: lui

برای تست پردازنده‌ی خود، یک برنامه بنویسید که بزرگترین عنصر یک آرایه‌ی ۲۰ عنصری از اعداد بدون علامت ۳۲ بیتی و اندیس آن را پیدا کند.

RISC-V Reference

RV32I Base Integer Instructions

Inst	Name	FMT	Opcode	funct3	funct7	Description (C)	Note
add	ADD	R	0110011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0110011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0110011	0x4	0x00	rd = rs1 ^ rs2	
or	OR	R	0110011	0x6	0x00	rd = rs1 | rs2	
and	AND	R	0110011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0110011	0x1	0x00	rd = rs1 << rs2	
srl	Shift Right Logical	R	0110011	0x5	0x00	rd = rs1 >> rs2	
sra	Shift Right Arith*	R	0110011	0x5	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011	0x2	0x00	rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011	0x3	0x00	rd = (rs1 < rs2)?1:0	zero-extends
addi	ADD Immediate	I	0010011	0x0		rd = rs1 + imm	
xori	XOR Immediate	I	0010011	0x4		rd = rs1 ^ imm	
ori	OR Immediate	I	0010011	0x6		rd = rs1 | imm	
andi	AND Immediate	I	0010011	0x7		rd = rs1 & imm	
slli	Shift Left Logical Imm	I	0010011	0x1	imm[5:11]=0x00	rd = rs1 << imm[0:4]	
srli	Shift Right Logical Imm	I	0010011	0x5	imm[5:11]=0x00	rd = rs1 >> imm[0:4]	
srai	Shift Right Arith Imm	I	0010011	0x5	imm[5:11]=0x20	rd = rs1 >> imm[0:4]	msb-extends
slti	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
sltiu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≥	B	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	I	1100111	0x0		rd = PC+4; PC = rs1 + imm	
lui	Load Upper Imm	U	0110111			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	
ecall	Environment Call	I	1110011	0x0	imm=0x0	Transfer control to OS	
ebreak	Environment Break	I	1110011	0x0	imm=0x1	Transfer control to debugger	

32-bit RISC-V Instruction Format

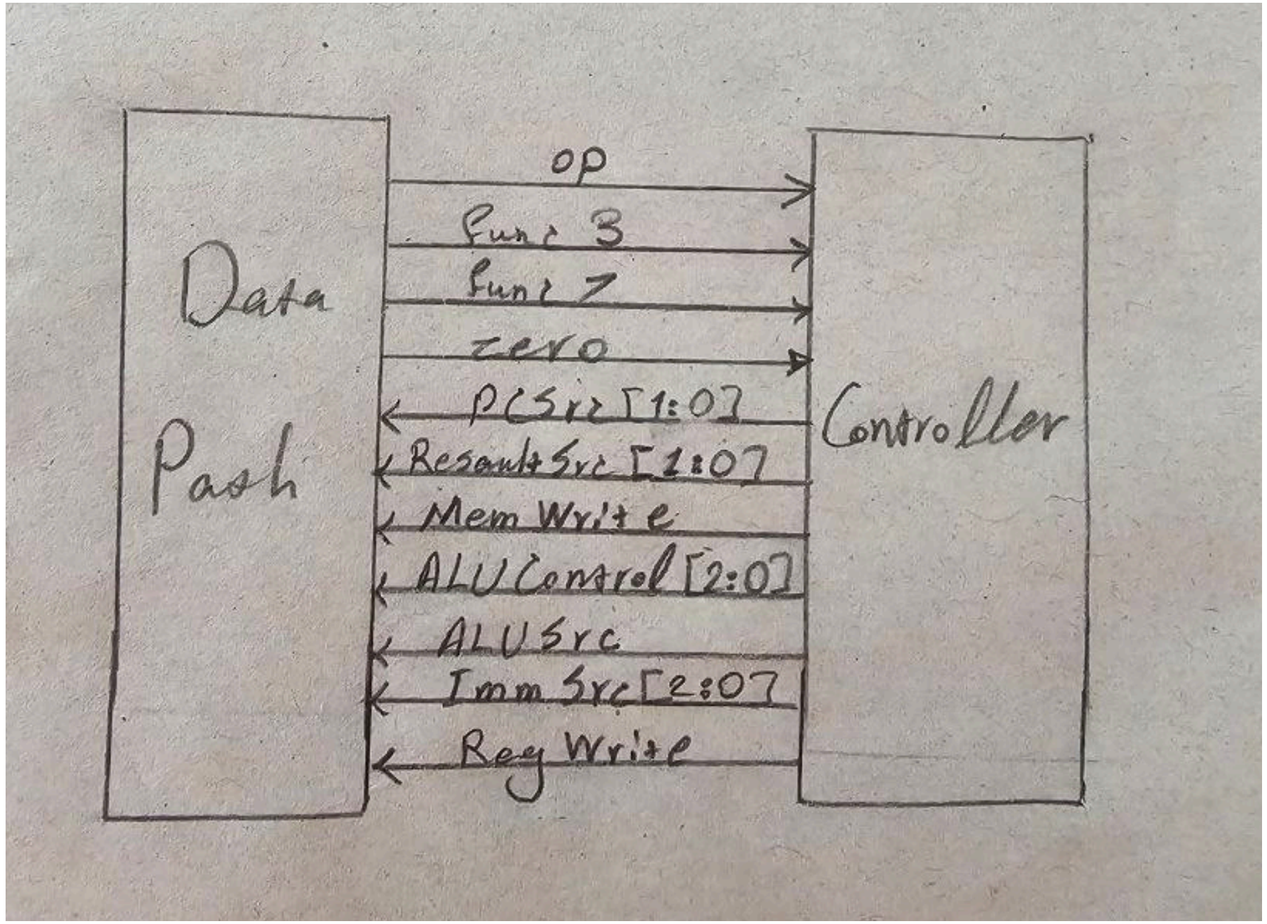
Instruction Formats	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Register/register	funct7							rs2					rs1					funct3			rd					opcode						
Immediate	imm[11:0]												rs1					funct3			rd					opcode						
Upper Immediate	imm[31:12]																				rd		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

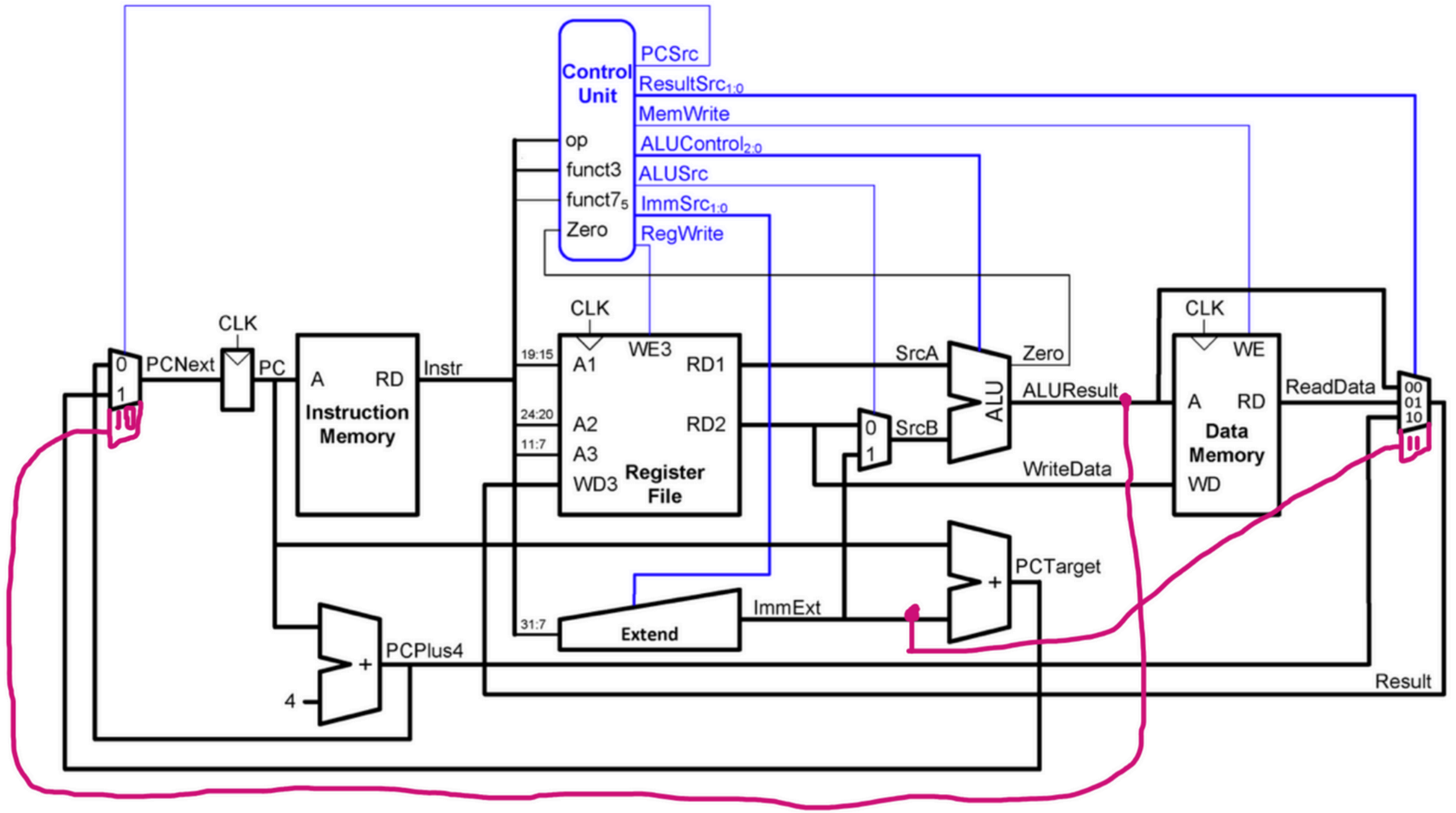
ImmSrc

ImmSrc	ImmExt	Input Type
000	**{ 20{ Ins[31] } , Ins[31:20] }**	I-Type
001	**{ 20{ Ins[31] } , Ins[31:25] , Ins[11:7] }**	S-Type
010	**{ 19{ Ins[31] } , Ins[31] , Ins[7] , Ins[30:25] , Ins[11:8] , 1'b0 }**	B-Type
011	**{ 11{ Ins[31] } , Ins[31] , Ins[19:12] , Ins[20] , Ins[30:21] , 1'b0 }**	J-Type
100	**{ Ins[31:12] , 12'b0 }**	U-Type

Internal Signals



Data Path


Control Unit
PCSrc
ResultSrc1:0
MemWrite
ALUControl2:0
ALUSrc
ImmSrc1:0
RegWrite
op
funct3
funct75
Zero
CLK
PCNext
PC
A
RD
Instruction Memory
Instr
19:15
24:20
11:7
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
CLK
WE
A
RD
Data Memory
WD
ReadData
00
01
10
PCTarget
ImmExt
31:7
Extend
PCPlus4
4
Result

Controller

```
7'b1101111: // J_Type
    begin
        RegWrite = 1;
        MemWrite = 0;
        PCSrc = 2'b01;
        ALUSrc = 1'bx;
        AluOp = 2'bxx;
        ResultSrc = 2'b10;
        ImmSrc = 3'b100;
    end
```

```
7'b0110111: // U_Type (Lui)
    begin
        RegWrite = 1;
        MemWrite = 0;
        PCSrc = 2'b00;
        ALUSrc = 1'bx;
        AluOp = 2'bxx;
        ResultSrc = 2'b11;
        ImmSrc = 3'b011;
    end
```

```
7'b1100011: // B-Type
    begin
        RegWrite = 0;
        MemWrite = 0;
        ALUSrc = 0;
        AluOp = 2'b01;
        ImmSrc = 3'b010;
        ResultSrc = 2'bxx;

        case (funct3)
            3'b000: PCSrc = (zero == 1) ? 2'b01 : 2'b00;  // BEQ
            3'b001: PCSrc = (zero == 0) ? 2'b01 : 2'b00;  // BNE
            default: PCSrc = 2'b00;
        endcase
    end
```

```
7'b0000011: // I_Type (Load)
    begin
        RegWrite = 1;
        MemWrite = 0;
        PCSrc = 2'b00;
        ALUSrc = 1;
        AluOp = 2'b00;
        ResultSrc = 2'b01;
        ImmSrc = 3'b000;
    end
```

```
7'b0100011: // S_Type
    begin
        RegWrite = 0;
        MemWrite = 1;
        PCSrc = 2'b00;
        ALUSrc = 1;
        AluOp = 2'b00;
        ResultSrc = 2'bxx;
        ImmSrc = 3'b001;
    end
```

```
7'b1100111: // I_Type (Jalr)
    begin
        RegWrite = 1;
        MemWrite = 0;
        PCSrc = 2'b10;
        ALUSrc = 1;
        AluOp = 2'b00;
        ResultSrc = 2'b10;
        ImmSrc = 3'b000;
    end
```

```
7'b0010011: // I_Type (Alu)
    begin
        RegWrite = 1;
        MemWrite = 0;
        PCSrc = 2'b00;
        ALUSrc = 1;
        AluOp = 2'b10;
        ResultSrc = 2'b00;
        ImmSrc = 3'b000;
    end
```

```
7'b0110011: // R_Type
    begin
        RegWrite = 1;
        MemWrite = 0;
        PCSrc = 2'b00;
        ALUSrc = 0;
        AluOp = 2'b10;
        ResultSrc = 2'b00;
        ImmSrc = 3'bxxx;
    end
```

Assembly code

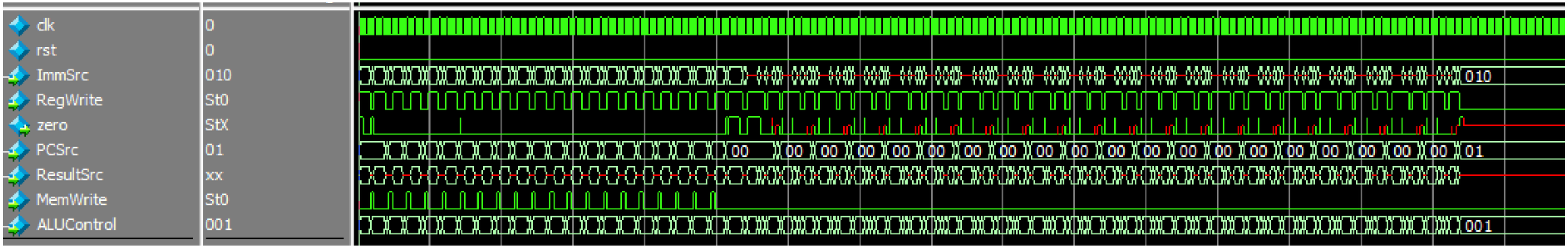
```
addi sp, zero, 0
addi tp, zero, 0
addi t0, zero, 20
addi t1, zero, 5

store_loop:
    sw t1, 0(sp)
    addi sp, sp, 4
    addi t1, t1, 3
    addi tp, tp, 1
    bne tp, t0, store_loop


addi a0, zero, 0
addi a1, zero, 0
addi a2, zero, 0
addi a3, zero, 20

loop:
  beq  a2, a3, done
  add  a4, a2, a2
  add  a4, a4, a4
  add  a5, a0, a4
  lw   a6, 0(a5)
  slt  a7, a1, a6
  beq  a7, zero, skip
  add  a1, a6, zero
skip:
  addi a2, a2, 1
  jal  zero, loop
done:
```

Machine code

```
00000113
00000213
01400293
00500313
00612023
00410113
00330313
00120213
FE5218E3
00000513
00000593
00000613
01400693
02D60463
00C60733
00E70733
00E507B3
0007A803
0105A8B3
00088463
000805B3
00160613
FDDFF06F
```

Simulation Result (1)

Wave Form



Simulation Result (2)

Memory

Memory Data - /riscv_testbench/dut/datapath_inst/dmem/mem - Default

00000000	00000000000000000000000000000101	00000000000000000000000000001000	00000000000000000000000000001011	00000000000000000000000000001110	00000000000000000000000000010001
00000005	00000000000000000000000000010100	00000000000000000000000000010111	00000000000000000000000000011010	00000000000000000000000000011101	00000000000000000000000000100000
0000000a	00000000000000000000000000100011	00000000000000000000000000100110	00000000000000000000000000101001	00000000000000000000000000101100	00000000000000000000000000101111
0000000f	00000000000000000000000000110010	00000000000000000000000000110101	00000000000000000000000000111000	00000000000000000000000000111011	00000000000000000000000000111110
00000014	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000019	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001e	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx			
00000023					

Register File

Memory Data - /riscv_testbench/dut/datapath_inst/regfile/registers - Default

00000000	00000000000000000000000000000000	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	00000000000000000000000001010000	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	00000000000000000000000000010100
00000005	00000000000000000000000000010100	00000000000000000000000001000001	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000a	00000000000000000000000000000000	00000000000000000000000000111110	00000000000000000000000000010100	00000000000000000000000000010100	00000000000000000000000001001100
0000000f	00000000000000000000000001001100	00000000000000000000000000111110	00000000000000000000000000000001	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000014	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000019	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001e	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx			
00000023					

RAM

Memory Data - /riscv_testbench/dut/datapath_inst/imem/RAM - Default

00000000	00000000000000000000000100010011	00000000000000000000001000010011	00000001010000000000001010010011	00000000010100000000001100010011	00000000011000010010000000100011
00000005	00000000010000010000000100010011	00000000000110011000000110010011	00000000000010010000001000010011	11111111001010010000110001100011	00000000000000000000001010010011
0000000a	00000000000000000000010110010011	00000000000000000000011000010011	00000001010000000000011010010011	00000001011010110000001000110011	00000000110001100000011100110011
0000000f	00000000111001110000011100110011	00000000111001010000011110110011	00000000000000111101010000000011	00000001000001011010100010110011	00000000000010001000010001100011
00000014	00000000000010000000010110110011	00000000000101100000011000010011	11111101110111111111000001101111	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000019	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001e	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000023	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000028	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000002d	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000032	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000037	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000003c	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx	
00000041					