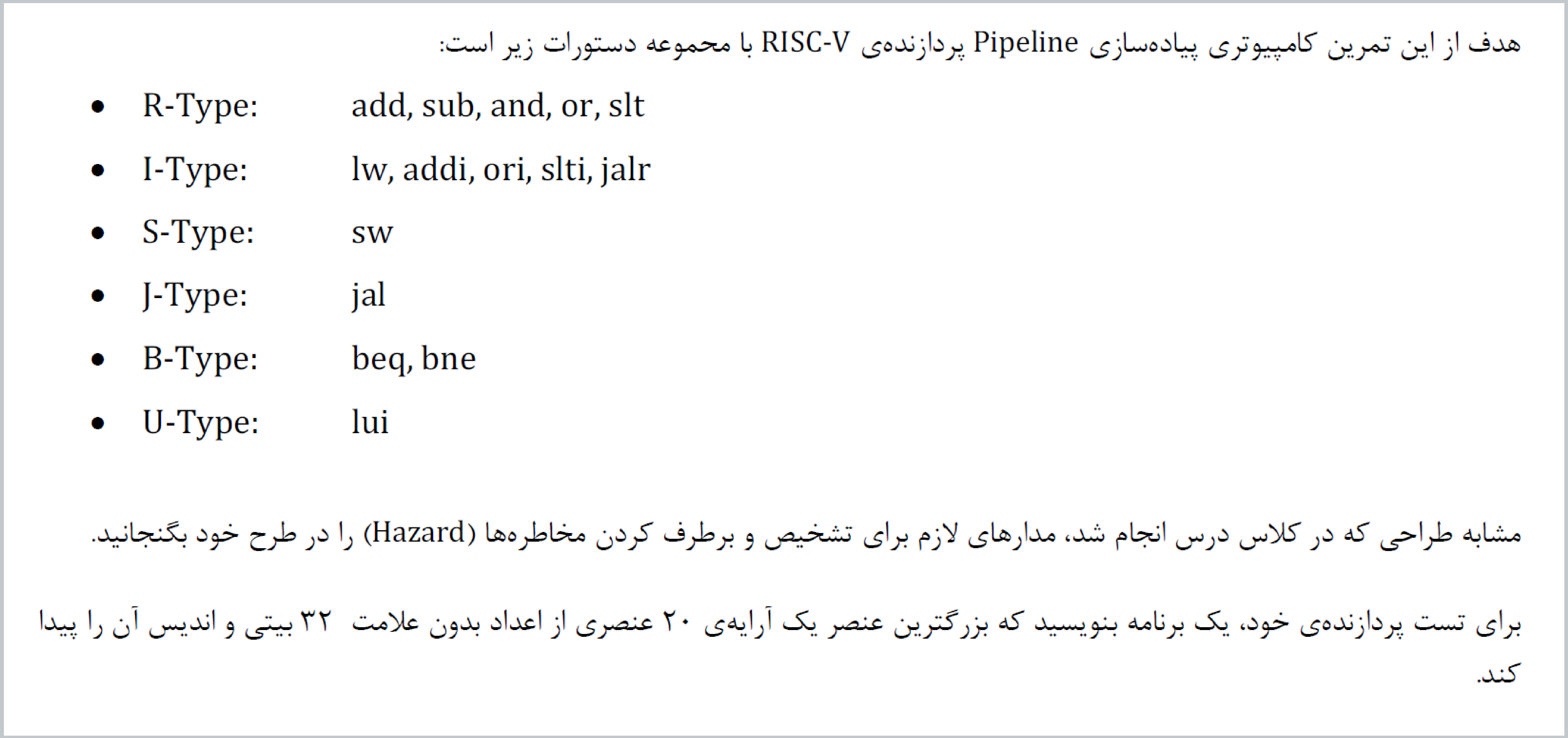
Project Report

Parsa KafshduziBukani 810102501
Milad Mazaheri 810102585

CA4

Project Description



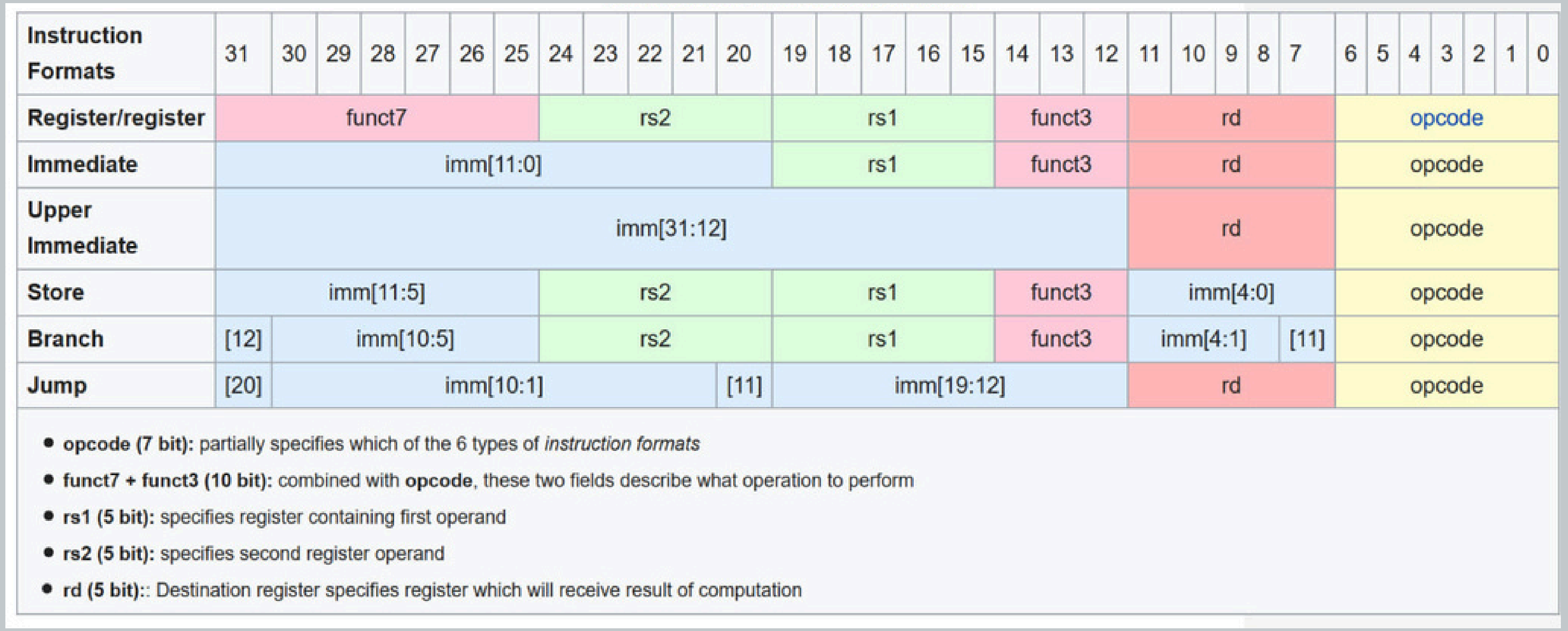
هدف از این تمرین کامپیوتری پیاده‌سازی Pipeline پردازنده‌ی RISC-V با مجموعه دستورات زیر است:

• 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

مشابه طراحی که در کلاس درس انجام شد، مدارهای لازم برای تشخیص و برطرف کردن مخاطره‌ها (Hazard) را در طرح خود بگنجانید.

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.

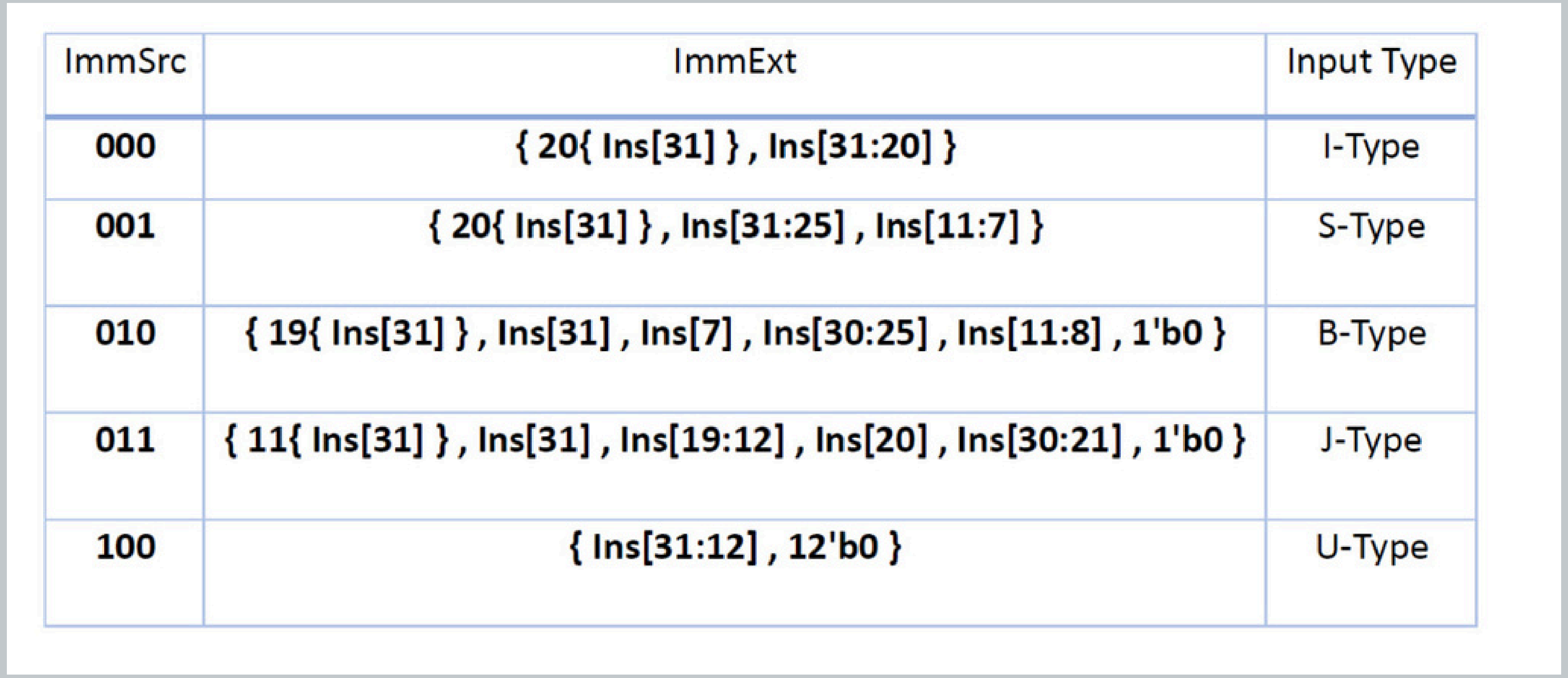
32-bit RISC-V Instruction Formats



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
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						

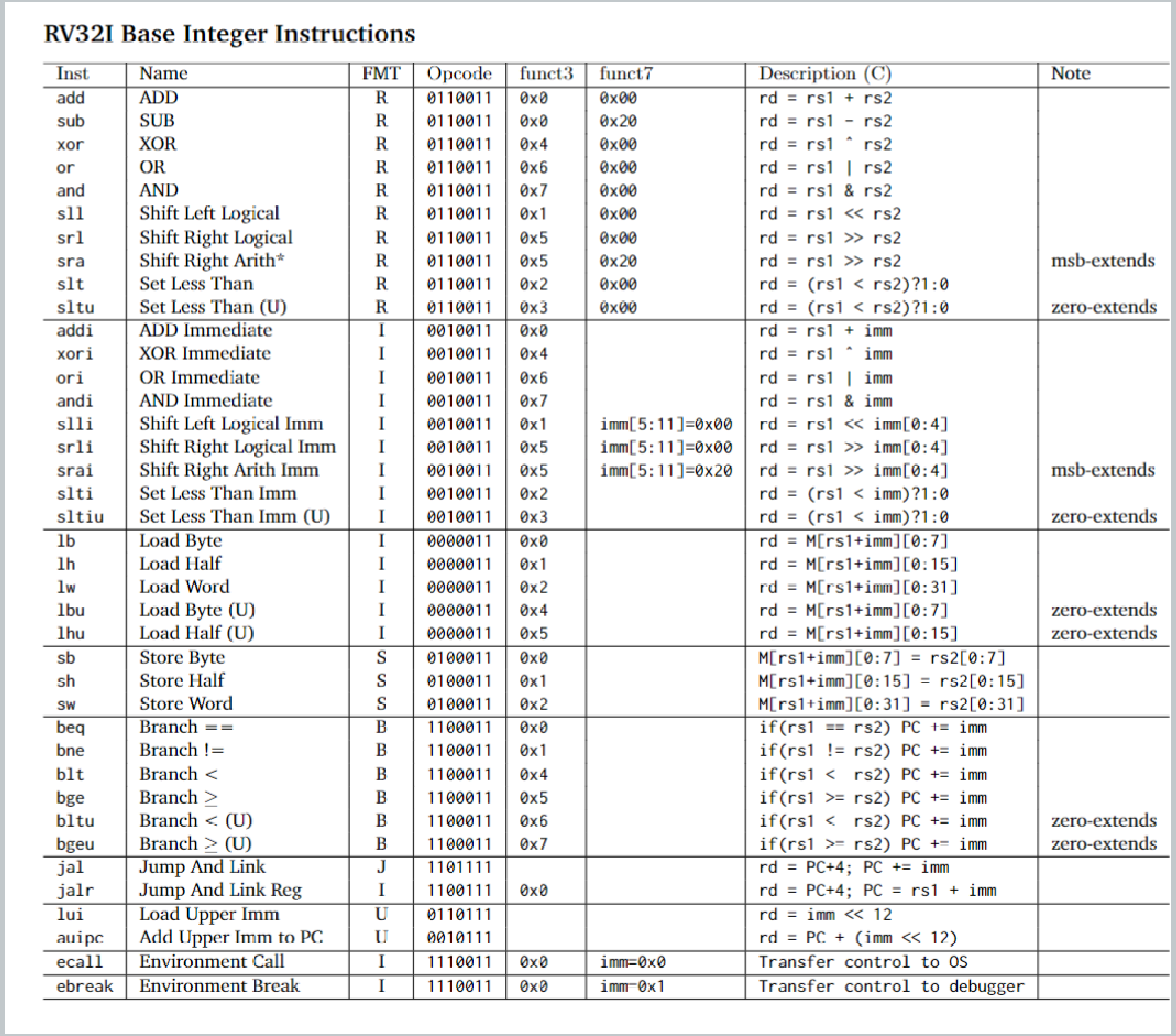
- **opcode (7 bit):** partially specifies which of the 6 types of *instruction formats*
- **funct7 + funct3 (10 bit):** combined with **opcode**, these two fields describe what operation to perform
- **rs1 (5 bit):** specifies register containing first operand
- **rs2 (5 bit):** specifies second register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

Immediate Extension Formats



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

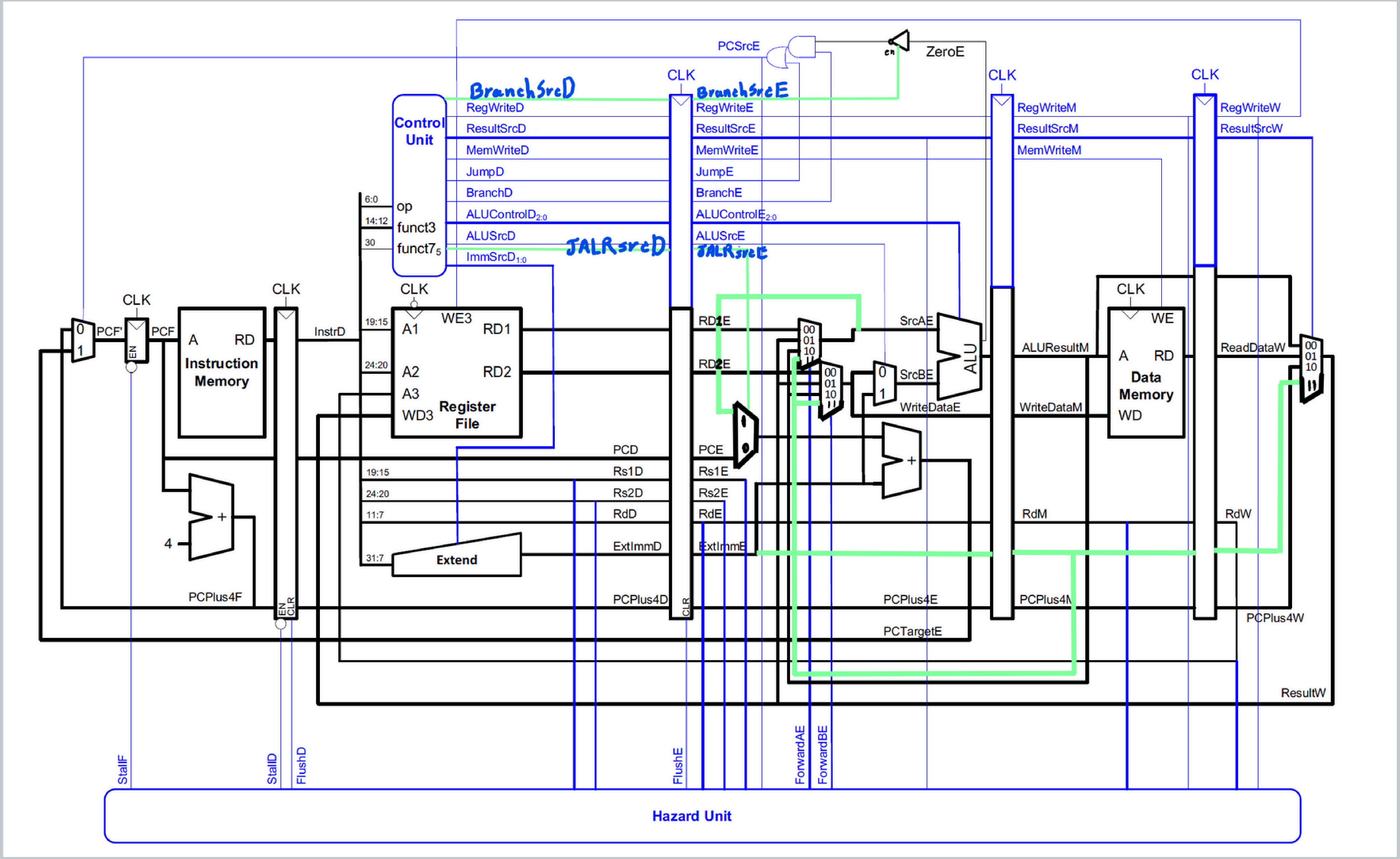
RISC-V Reference



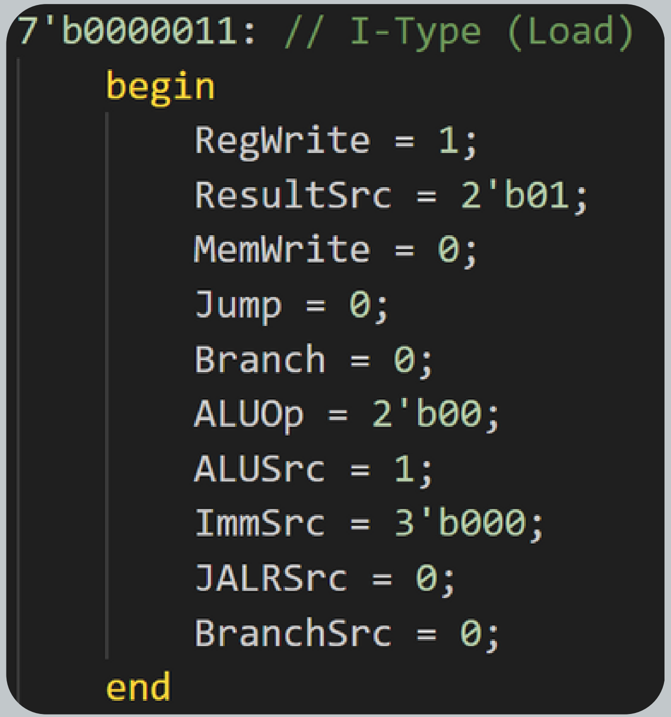
RV32I Base Integer Instructions

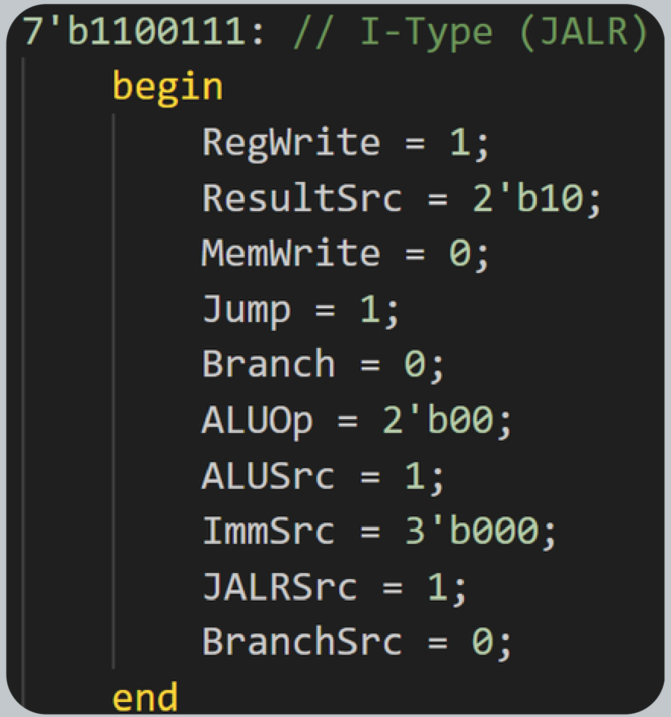
Inst	Name	FMT	Opcode	funct3	funct7	Description (C)	Note
add	ADD	R	0110011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0110011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0110011	0x4	0x00	rd = rs1 ˆ rs2	
or	OR	R	0110011	0x6	0x00	rd = rs1 | rs2	
and	AND	R	0110011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0110011	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 ˆ 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	

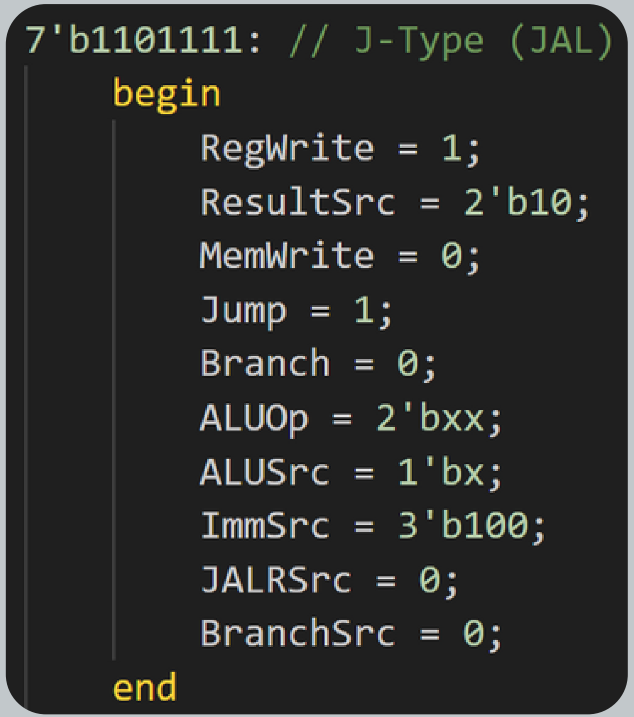
RISC-V Pipelined Processor

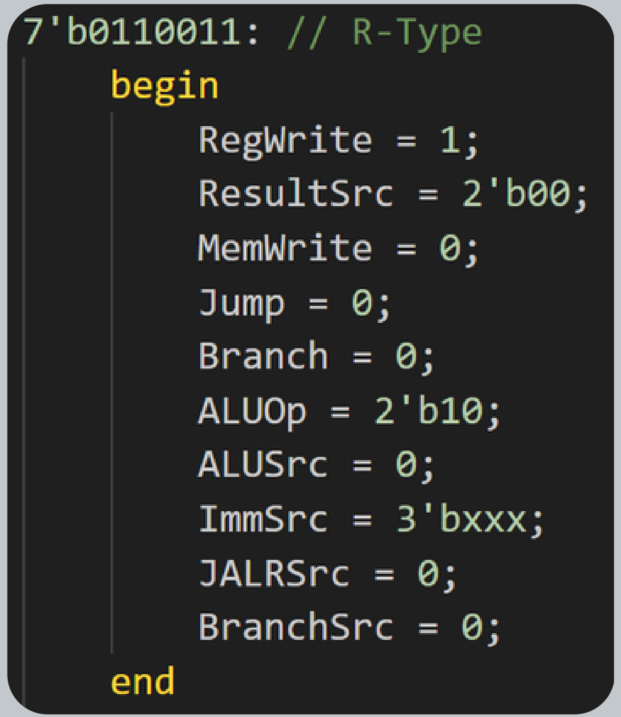


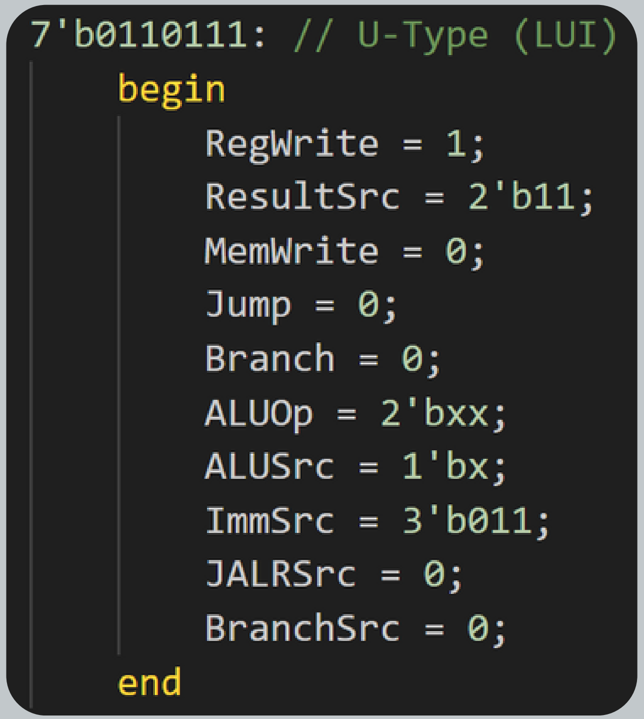
Controller

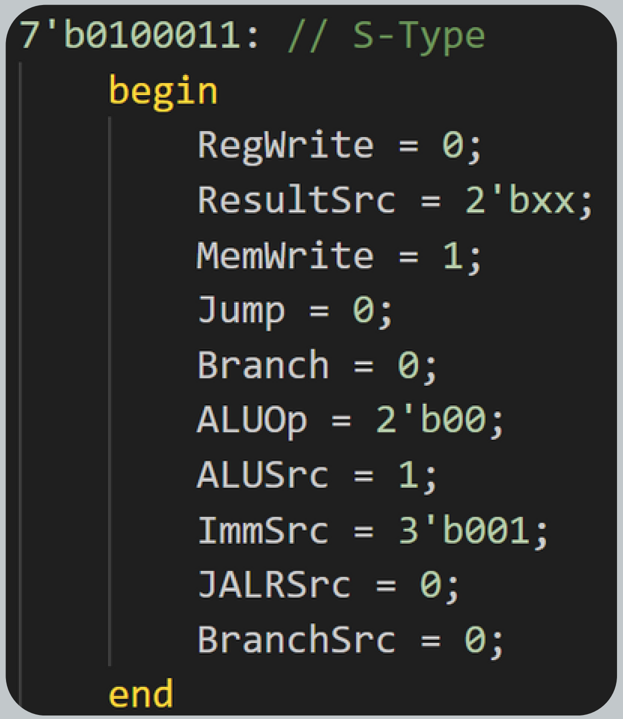

```
7'b0000011: // I-Type (Load)
    begin
        RegWrite = 1;
        ResultSrc = 2'b01;
        MemWrite = 0;
        Jump = 0;
        Branch = 0;
        ALUOp = 2'b00;
        ALUSrc = 1;
        ImmSrc = 3'b000;
        JALRSrc = 0;
        BranchSrc = 0;
    end
```

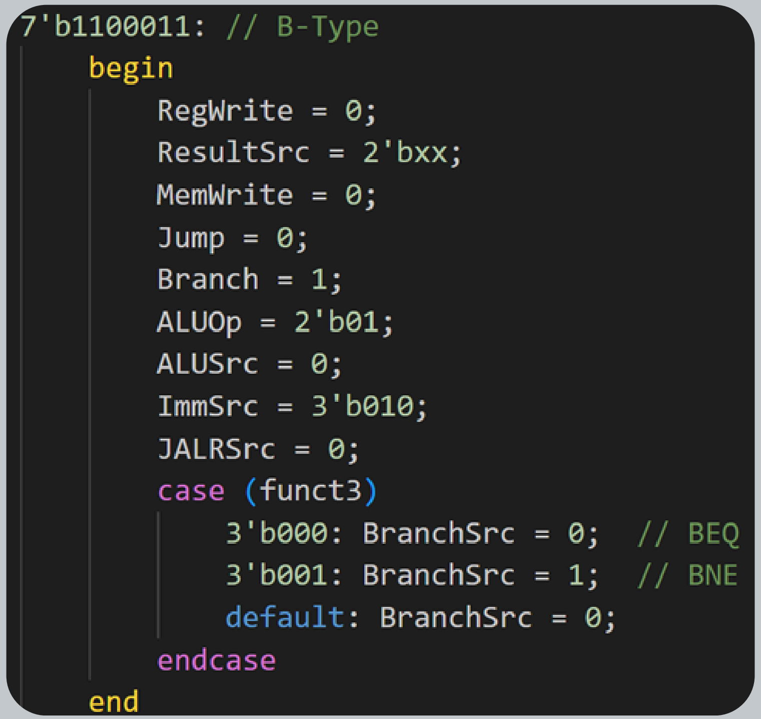

```
7'b1100111: // I-Type (JALR)
    begin
        RegWrite = 1;
        ResultSrc = 2'b10;
        MemWrite = 0;
        Jump = 1;
        Branch = 0;
        ALUOp = 2'b00;
        ALUSrc = 1;
        ImmSrc = 3'b000;
        JALRSrc = 1;
        BranchSrc = 0;
    end
```

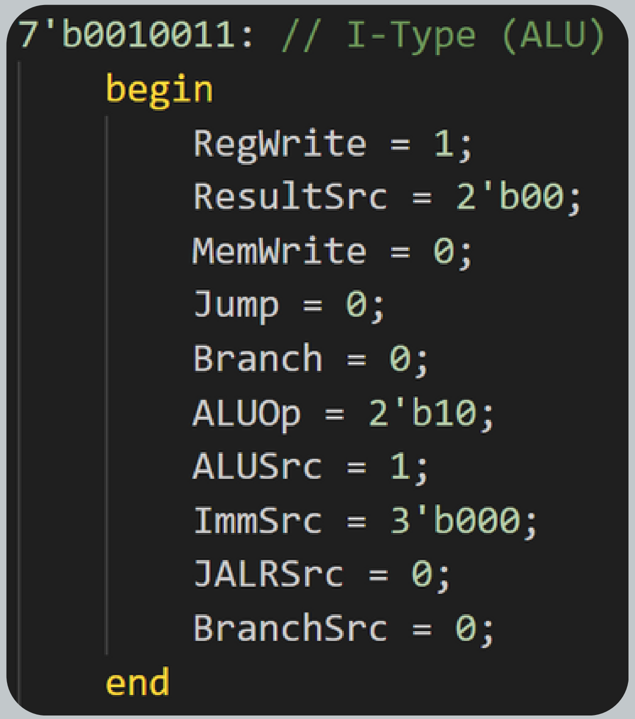

```
7'b1101111: // J-Type (JAL)
    begin
        RegWrite = 1;
        ResultSrc = 2'b10;
        MemWrite = 0;
        Jump = 1;
        Branch = 0;
        ALUOp = 2'bxx;
        ALUSrc = 1'bx;
        ImmSrc = 3'b100;
        JALRSrc = 0;
        BranchSrc = 0;
    end
```


```
7'b0110011: // R-Type
    begin
        RegWrite = 1;
        ResultSrc = 2'b00;
        MemWrite = 0;
        Jump = 0;
        Branch = 0;
        ALUOp = 2'b10;
        ALUSrc = 0;
        ImmSrc = 3'bxxx;
        JALRSrc = 0;
        BranchSrc = 0;
    end
```

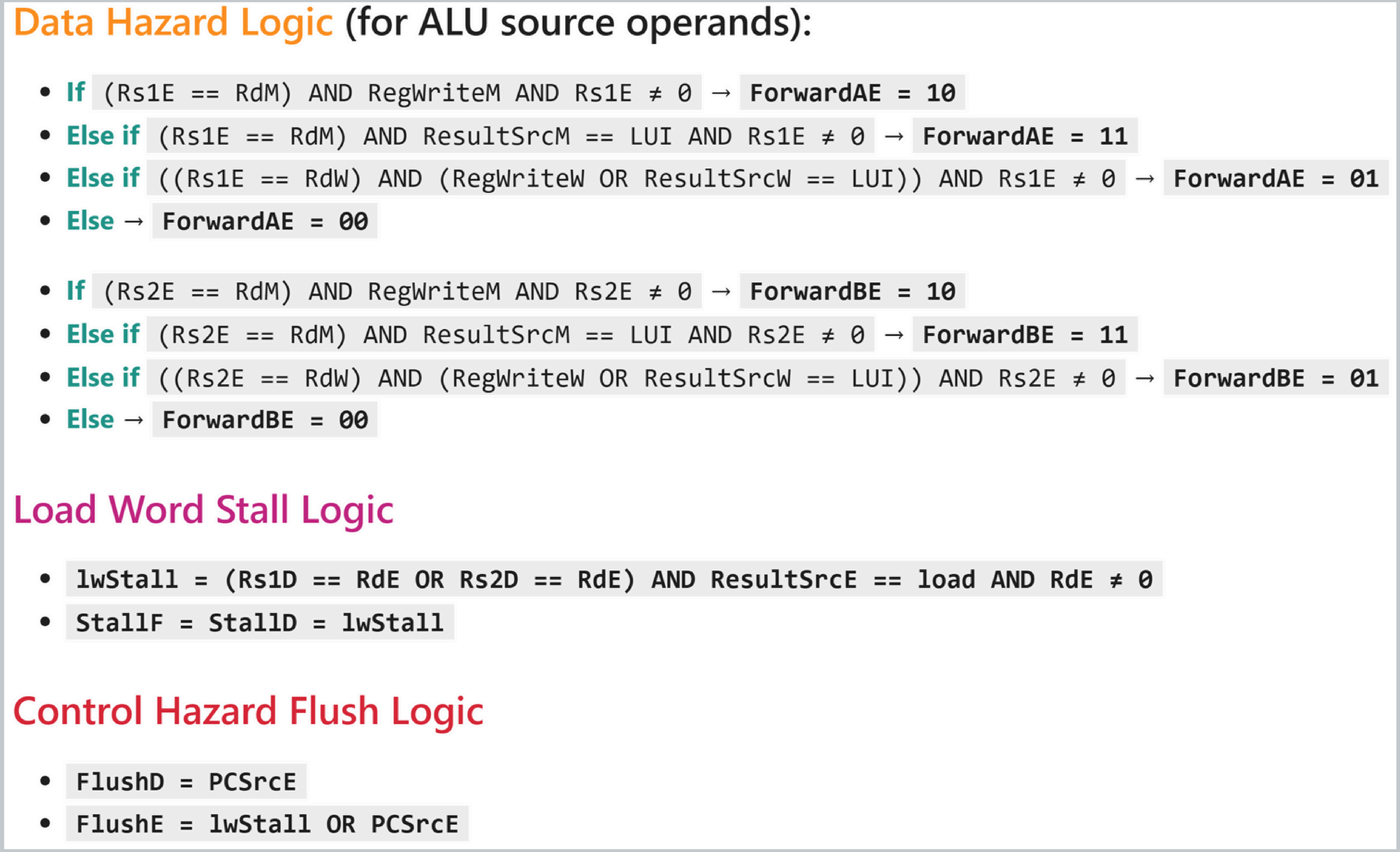

```
7'b0110111: // U-Type (LUI)
    begin
        RegWrite = 1;
        ResultSrc = 2'b11;
        MemWrite = 0;
        Jump = 0;
        Branch = 0;
        ALUOp = 2'bxx;
        ALUSrc = 1'bx;
        ImmSrc = 3'b011;
        JALRSrc = 0;
        BranchSrc = 0;
    end
```


```
7'b0100011: // S-Type
    begin
        RegWrite = 0;
        ResultSrc = 2'bxx;
        MemWrite = 1;
        Jump = 0;
        Branch = 0;
        ALUOp = 2'b00;
        ALUSrc = 1;
        ImmSrc = 3'b001;
        JALRSrc = 0;
        BranchSrc = 0;
    end
```


```
7'b1100011: // B-Type
    begin
        RegWrite = 0;
        ResultSrc = 2'bxx;
        MemWrite = 0;
        Jump = 0;
        Branch = 1;
        ALUOp = 2'b01;
        ALUSrc = 0;
        ImmSrc = 3'b010;
        JALRSrc = 0;
        case (funct3)
            3'b000: BranchSrc = 0;  // BEQ
            3'b001: BranchSrc = 1;  // BNE
            default: BranchSrc = 0;
        endcase
    end
```


```
7'b0010011: // I-Type (ALU)
    begin
        RegWrite = 1;
        ResultSrc = 2'b00;
        MemWrite = 0;
        Jump = 0;
        Branch = 0;
        ALUOp = 2'b10;
        ALUSrc = 1;
        ImmSrc = 3'b000;
        JALRSrc = 0;
        BranchSrc = 0;
    end
```

Hazard Unit



Data Hazard Logic (for ALU source operands):

- **If** `(Rs1E == RdM) AND RegWriteM AND Rs1E ≠ 0` → `ForwardAE = 10`
- **Else if** `(Rs1E == RdM) AND ResultSrcM == LUI AND Rs1E ≠ 0` → `ForwardAE = 11`
- **Else if** `((Rs1E == RdW) AND (RegWriteW OR ResultSrcW == LUI)) AND Rs1E ≠ 0` → `ForwardAE = 01`
- **Else** → `ForwardAE = 00`

- **If** `(Rs2E == RdM) AND RegWriteM AND Rs2E ≠ 0` → `ForwardBE = 10`
- **Else if** `(Rs2E == RdM) AND ResultSrcM == LUI AND Rs2E ≠ 0` → `ForwardBE = 11`
- **Else if** `((Rs2E == RdW) AND (RegWriteW OR ResultSrcW == LUI)) AND Rs2E ≠ 0` → `ForwardBE = 01`
- **Else** → `ForwardBE = 00`

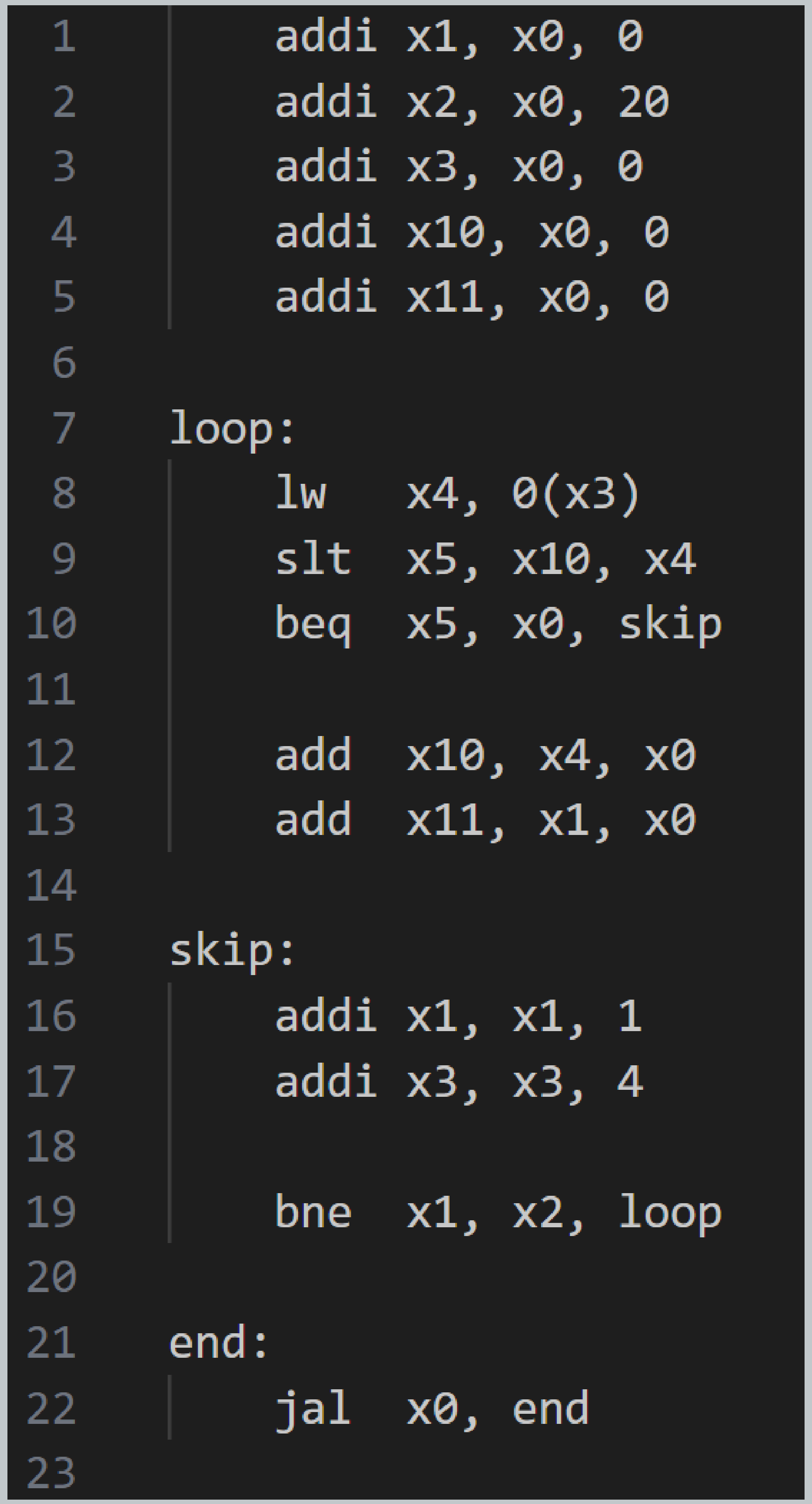
Load Word Stall Logic

- `lwStall = (Rs1D == RdE OR Rs2D == RdE) AND ResultSrcE == load AND RdE ≠ 0`
- `StallF = StallD = lwStall`

Control Hazard Flush Logic

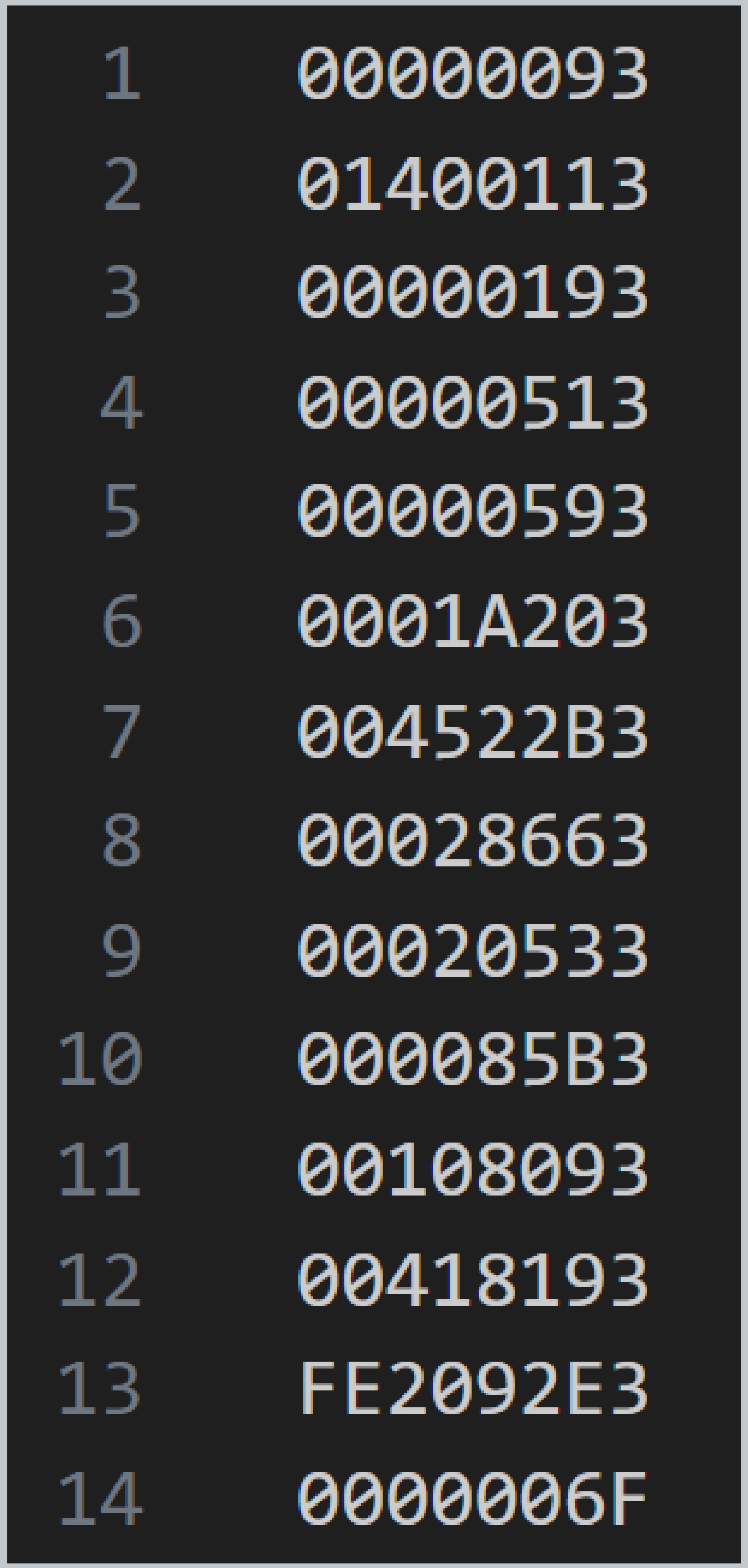
- `FlushD = PCSrcE`
- `FlushE = lwStall OR PCSrcE`

Assembly code



```
    addi x1, x0, 0
    addi x2, x0, 20
    addi x3, x0, 0
    addi x10, x0, 0
    addi x11, x0, 0

loop:
    lw   x4, 0(x3)
    slt  x5, x10, x4
    beq  x5, x0, skip

    add  x10, x4, x0
    add  x11, x1, x0

skip:
    addi x1, x1, 1
    addi x3, x3, 4

    bne  x1, x2, loop

end:
    jal  x0, end

```

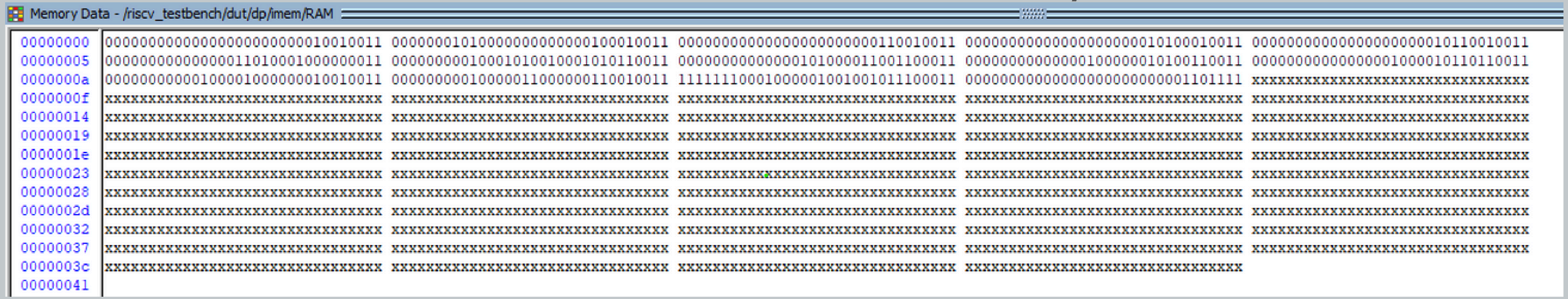
Machine code



```
00000093
01400113
00000193
00000513
00000593
0001A203
004522B3
00028663
00020533
000085B3
00108093
00418193
FE2092E3
0000006F
```

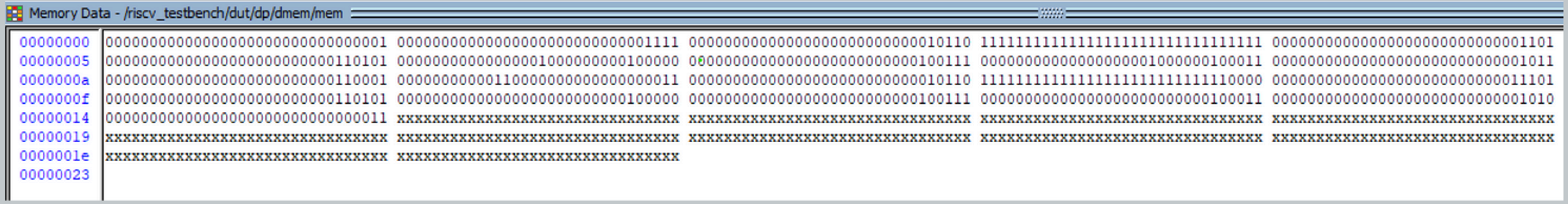
Simulation Results

Instruction Memory



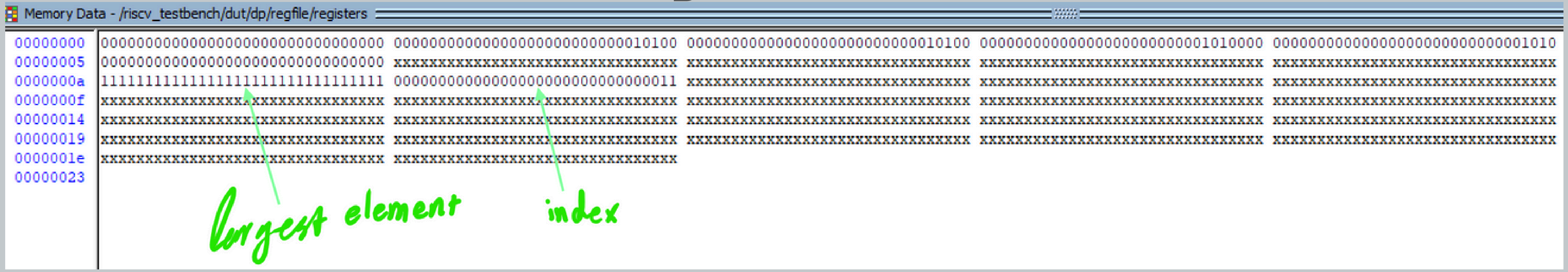
```
Memory Data - /riscv_testbench/dut/dp/imem/RAM
00000000 00000000000000000000000010010011 00000001010000000000000100010011 00000000000000000000000110010011 00000000000000000000010100010011 00000000000000000000010110010011
00000005 00000000000000011010001000000011 00000000010001010010001010110011 00000000000000101000011001100011 00000000000000100000010100110011 00000000000000001000010110110011
0000000a 00000000000100001000000010010011 00000000010000011000000110010011 11111110001000001001001011100011 00000000000000000000000001101111 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000f xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000014 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx 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
```

Data Memory



```
Memory Data - /riscv_testbench/dut/dp/dmem/mem
00000000 00000000000000000000000000000001 00000000000000000000000000001111 00000000000000000000000000010110 11111111111111111111111111111111 00000000000000000000000000001101
00000005 00000000000000000000000000110101 00000000000000001000000000100000 00000000000000000000000000100111 00000000000000000001000000100011 00000000000000000000000000001011
0000000a 00000000000000000000000000110001 00000000000110000000000000000011 00000000000000000000000000010110 11111111111111111111111111110000 00000000000000000000000000011101
0000000f 00000000000000000000000000110101 00000000000000000000000000100000 00000000000000000000000000100111 00000000000000000000000000100011 00000000000000000000000000001010
00000014 00000000000000000000000000000011 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000019 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001e xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000023
```

Register File



```
Memory Data - /riscv_testbench/dut/dp/regfile/registers
00000000 00000000000000000000000000000000 00000000000000000000000000010100 00000000000000000000000000010100 00000000000000000000000001010000 00000000000000000000000000001010
00000005 00000000000000000000000000000000 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000a 11111111111111111111111111111111 00000000000000000000000000000011 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000000f xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000014 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000019 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0000001e xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
00000023
```