Building a Pipelined Processor

RISC V

Part 1

RV32IM Instructions

And

Pipeline Datapath

Damsy De Silva (E/16/069)

Shirly Ekanayake (E/16/094)

Buddhi Perera (E/16/276)

Contents

1. Instruction Encoding Formats

Instructions in RISC-V ISA can be divided into 4 main categories depending on their formats.

1. R-Type
2. I-Type
3. S-Type
4. U-Type

S-Type and U-Type instructions can be further categorized according to the way the immediate value is handled. Figure 1 shows the instruction formats in RISC-V ISA.

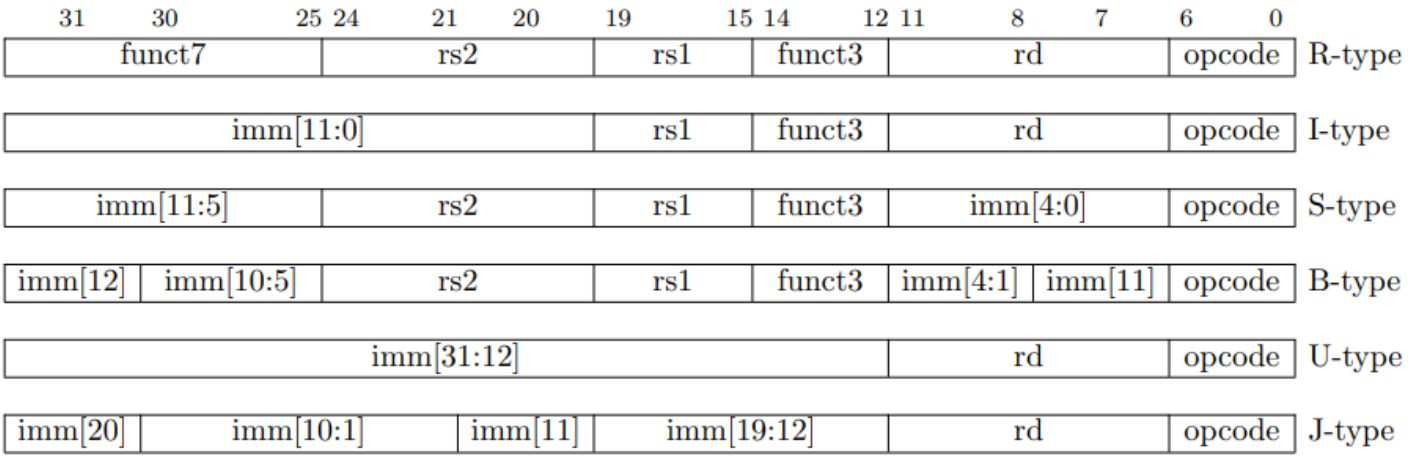

FIGURE 1 : RISC-V Instruction Formats

2. Instructions in RV32IM

2.1. Integer Computational Instructions

2.1.1. Register - Immediate Instructions

Instruction	Instruction Name	Example	Description	Instruction Type
ADDI	Add immediate Unsigned	ADDI x1, x2, 3	Reg[x1] <- Reg[x2] + 3	I - Type
SLTI	Set less than Immediate	SLTI x1, x2, 3	if (Regs[x2] > 3): Regs[x1] <- 1 else: Regs[x1] <- 0 (Signed Comparison)	I - Type
SLTIU	Set less than Immediate Unsigned	SLTI x1, x2, 3	if (Regs[x2] > 3): Regs[x1] <- 1 else: Regs[x1] <- 0 (Unsigned Comparison)	I - Type
ANDI	Logical AND immediate	ANDI x1, x2, 3	Reg[x1] <- Reg[x2] & 3	I - Type
ORI	Logical OR immediate	ORI x1, x2, 3	Reg[x1] <- Reg[x2] | 3	I - Type
XORI	Logical XOR immediate	XORI x1, x2, 3	Reg[x1] <- Reg[x2] ^ 3	I - Type
SLLI	Logical Left shift Immediate	SLLI x1, x2, 3	Reg[x1] <- Reg[x2] << 3	I - Type
SRLI	Logical right Shift Immediate	SRLI x1, x2, 3	Reg[x1] <- Reg[x2] >> 3 (zeros are shifted into the upper bits)	I - Type
SRAI	Arithmetic Right Shift	SRAI x1, x2, 3	Reg[x1] <- Reg[x2] >> 3 (Original Sign bit is copied into the vacated upper bits)	I - Type
LUI	Load Upper Immediate	LUI x1, 3	Reg[x1] <- $[3]_{20\ bits}[0]_{12\ bits}$ immediate value in the top 20 bits & lowest 12 bits with zeros	U - Type
AUIPC	Add upper immediate to PC	ALUPC x1, 3	Reg[x1]<-PC+$[[3]_{20\ bits}[0]_{12}]$	U - Type

TABLE 1 : Register - Immediate Instructions

31 20	19 15	14 12	11 7	6 0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
I-immediate[11:0]	src	ADDI/SLTI[U]	dest	OP-IMM
I-immediate[11:0]	src	ANDI/ORI/XORI	dest	OP-IMM

31 25	24 20	19 15	14 12	11 7	6 0
imm[11:5]	imm[4:0]	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	shamt[4:0]	src	SLLI	dest	OP-IMM
0000000	shamt[4:0]	src	SRLI	dest	OP-IMM
0100000	shamt[4:0]	src	SRAI	dest	OP-IMM

FIGURE 2 : Register - Immediate Instruction encoding formats

2.1.2. Register - Register Instructions

Instruction	Instruction Name	Example	Description	Instruction Type
ADD	Addition	ADD x1, x2, x3	Reg[x1] <- Reg[x2] + Reg[x3]	R - Type
SLT	Set less than	SLT x1, x2, x3	if (Regs[x2]>Regs[x3]): Regs[x1]<-1 else: Regs[x1] <- 0 (Signed Comparison)	R - Type
SLTU	Set less than unsigned	SLTU x1, x2, x3	if (Regs[x2] > Regs[x3]): Regs[x1] <- 1 else: Regs[x1] <- 0 (Signed Comparison)	R - Type
AND	Bitwise logical AND	AND x1, x2, x3	Reg[x1] <- Reg[x2] & Regs[x3]	R - Type
OR	Bitwise logical OR	OR x1, x2, x3	Reg[x1] <- Reg[x2] | Regs[x3]	R - Type
XOR	Bitwise logical XOR	XOR x1, x2, x3	Reg[x1] <- Reg[x2] ^ Regs[x3]	R - Type
SLL	Logical Left Shift	SLL x1, x2, x3	Reg[x1] <- Reg[x2] << Regs[x3]	R - Type
SRL	Logical Right Shift	SRL x1, x2, x3	Reg[x1] <- Reg[x2] >> 3 (zeros are shifted into the upper bits)	R - Type
SUB	Subtraction	SUB x1, x2, x3	Reg[x1] <- Reg[x2] - Reg[x3]	R - Type
SRA	Arithmetic Right shift	SRA x1,x2,x3	Reg[x1] <- Reg[x2] >> Reg[x3] (Original Sign bit is copied into the vacated upper bits)	R - Type

TABLE 2 : Register - Register Instructions

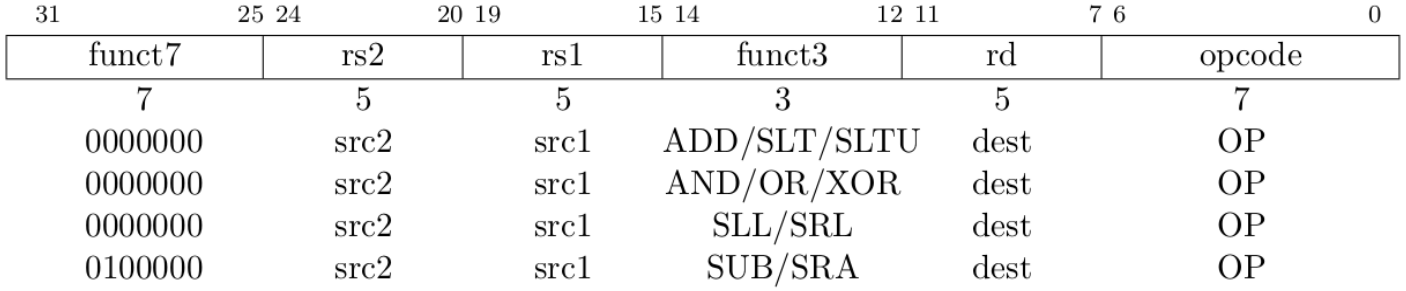

FIGURE 3 : Register - Register Instruction Encoding Format

2.2. Control Transfer Instructions

Instruction	Instruction Name	Example	Description	Instruction Type
JAL	Jump and Link	JAL x1, offset	Regs[x1] <- PC+4; PC <- PC + (offset << 1)	J - Type
JALR	Jump and link register	JALR x1, x2,offset	Regs[x1] <- PC+4; PC<-Regs[x2]+(offset<<1)	I - Type
BEQ	Branch Equal	BEQ x1, x2,offset	if(Regs[x1]== Regs[x2]): PC<-PC+(offset<< 1)	B - Type
BNE	Branch Not Equal	BNE x1, x2,offset	if(Regs[x1] != Regs[x2]): PC <- PC+(offset<< 1)	B - Type
BLT	Branch Less Than	BLT x1, x2,offset	if(Regs[x1] < Regs[x2]): PC <- PC+(offset<< 1)	B - Type
BLTU	Branch Less Than unsigned	BLTU x1, x2,offset	if(Regs[x1] < Regs[x2]): PC <- PC+(offset<< 1) (Unsigned comparison)	B - Type
BGE	Branch Greater than or equal	BGE x1, x2,offset	if(Regs[x1] > Regs[x2]): PC <- PC+ (offset<< 1)	B - Type
BGEU	Branch Greater than or Equal Unsigned	BGEU x1, x2,offset	if(Regs[x1] > Regs[x2]): PC <- PC+ (offset<< 1) (Unsigned comparison)	B - Type

TABLE 3 : Control Transfer Instructions

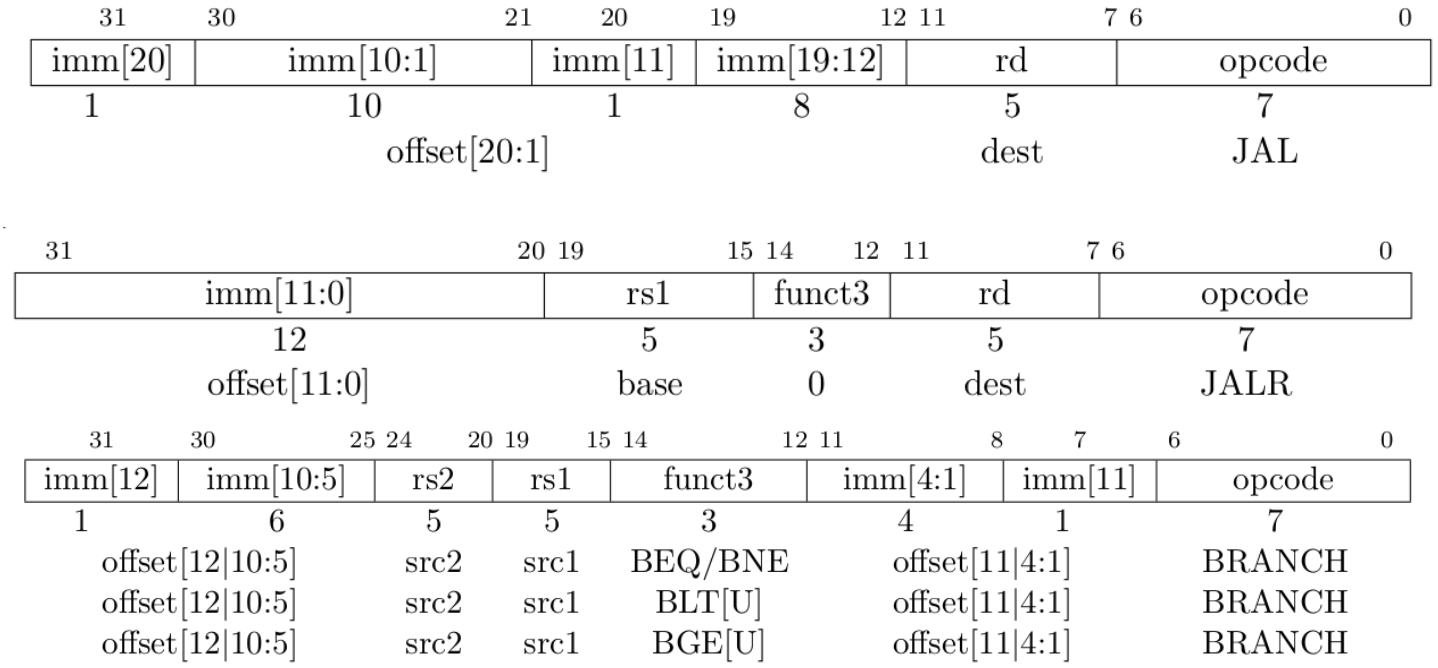

FIGURE 4 : Control Transfer Instruction Encoding Formats

2.3. Load and Store Instructions

Instruction	Instruction Name	Example	Description	Instruction Type
LW	Load Word(32bit)	Lw x1, 60(x2)	Regs[x1]<- Mem[60+Regs[x2]]$_{32\ bit}$	I - Type
LH	Load Half word (16 bit)	LH x1, 60(x2)	Regs[x1]<- [Mem[60+Regs[x2]]]$_{16\ bit}$ Sign extend to 32 bits	I - Type
LHU	Load half word Unsigned	LHU x1, 60(x2)	Regs[x1]<- [Mem[60+Regs[x2]]]$_{16\ bit}$ Zero extend to 32 bits	I - Type
LB	Load Byte (8 bit)	LB x1, 60(x2)	Regs[x1]<- [Mem[60+Regs[x2]]]$_{8\ bit}$ Sign extend to 32 bits	I - Type
LBU	Load Byte Unsigned	LBU x1, 60(x2)	Regs[x1]<- [Mem[60+Regs[x2]]]$_{8\ bit}$ Zero extend to 32 bits	I - Type
SW	Store Word	SW x1, 60(x2)	[Mem[60+Regs[x2]]] <- Reg[x1]$_{32\ bit}$	S - Type
SH	Store Half word	SH x1, 60(x2)	[Mem[60+Regs[x2]]] <- Reg[x1]$_{16\ bit}$	S - Type
SB	Store Byte	SB x1, 60(x2)	[Mem[60+Regs[x2]]] <- Reg[x1]$_{8\ bit}$	S - Type

TABLE 4 : Load and Store Instructions

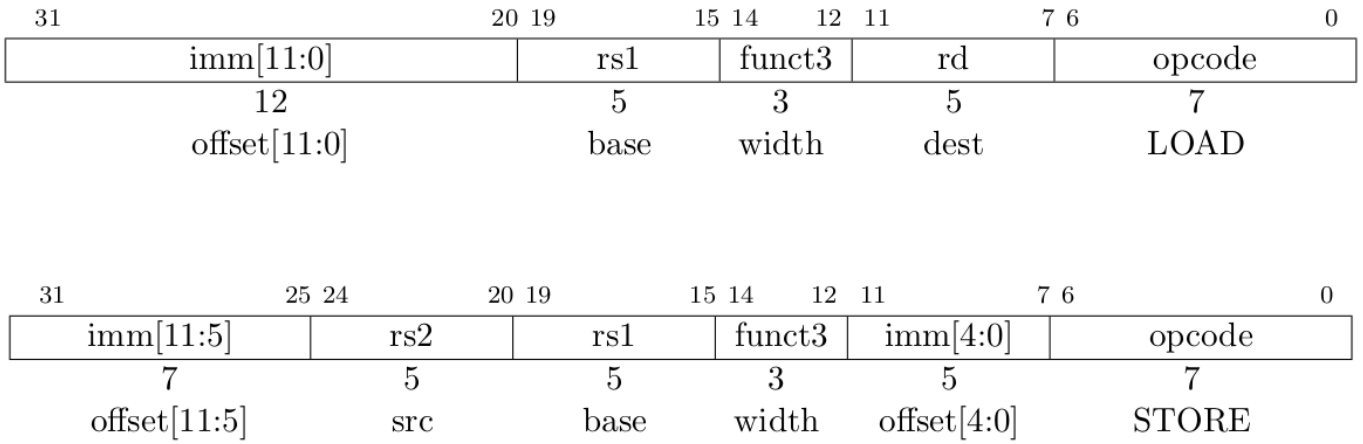

FIGURE 5 : Load and Store Instruction Encoding Formats

2.4. Multiplication / Division Instructions

Instruction	Instruction Name	Example	Description	Instruction Type
MUL	Multiplication	MUL x1,x2,x3	Reg[x1]<-[Reg[x2]*Reg[x3]]$_{\text{lower 32bit}}$	R - Type
MULH	Multiplication High	MULH x1,x2,x3	Reg[x1]<-[Reg[x2]*Reg[x3]]$_{\text{upper32bit}}$ Signed x Signed	R - Type
MULHU	Multiplication High Unsigned	MULHU x1,x2,x3	Reg[x1]<-[Reg[x2]*Reg[x3]]$_{\text{upper32bit}}$ (Unsigned x Unsigned)	R - Type
MULHSU	Multiplication High Signed UnSigned	MULHSU x1,x2,x3	Reg[x1]<-[Reg[x2]*Reg[x3]]$_{\text{upper32bit}}$ (Signed x Unsigned)	R - Type
DIV	Division	DIV x1,x2,x3	Reg[x1]<-[Reg[x2] / Reg[x3]] (Signed Division) Round towards Zero	R - Type
DIVU	Division Unsigned	DIVU x1,x2,x3	Reg[x1]<-[Reg[x2] / Reg[x3]] (Unsigned Division) Round towards Zero	R - Type
REM	Remainder	REM x1,x2,x3	Reg[x1]<-[Reg[x2] % Reg[x3]] Sign of the result = Sign of the dividend	R - Type
REMU	Remainder Unsigned	REMUx1,x2,x3	Reg[x1]<-[Reg[x2] % Reg[x3]] (Unsigned Division)	R - Type

TABLE 5 : Multiplication & Division Instructions

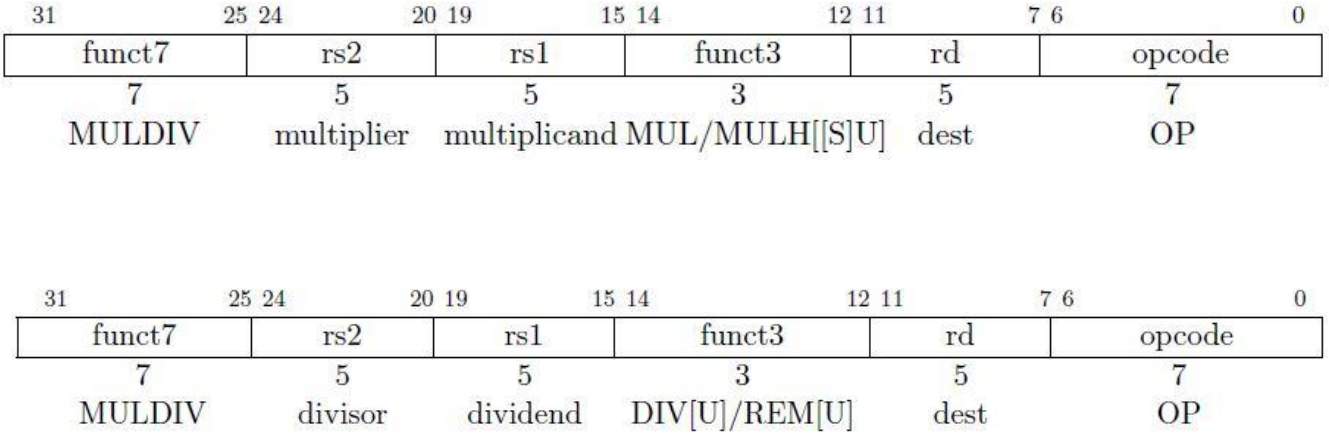

FIGURE 6 : Multiplication & Division Instruction Encoding Formats

2.5. Memory Ordering Instruction

The FENCE instruction is used to order device I/O and memory accesses as viewed by other RISC-V harts and external devices or coprocessors. FENCE instruction that provides explicit synchronization between writes to instruction memory and instruction fetches on the same hart.

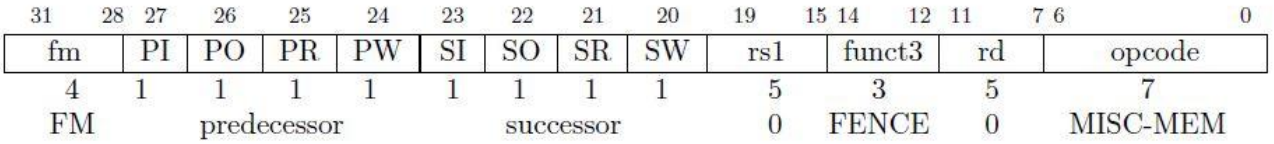

FIGURE 7 : Memory Ordering Instruction Encoding Format

2.6. Environment Call and Breakpoints

SYSTEM instructions are used to access system functionality that might require privileged access and are encoded using the I-type instruction format
The ECALL instruction is used to make a service request to the execution environment.
The EBREAK instruction is used to return control to a debugging environment.

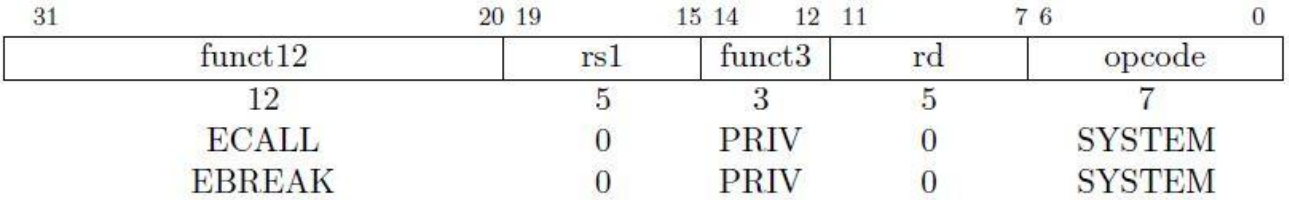

FIGURE 8 : System Instructions Encoding Format

3. Opcodes

The OPCODES used in the RISC - V manual were used as the OPCODES to implement the RV32IM.

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
fm | pred | succ		rs1	000	rd	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK

RV32M Standard Extension

0000001	rs2	rs1	000	rd	0110011	MUL
0000001	rs2	rs1	001	rd	0110011	MULH
0000001	rs2	rs1	010	rd	0110011	MULHSU
0000001	rs2	rs1	011	rd	0110011	MULHU
0000001	rs2	rs1	100	rd	0110011	DIV
0000001	rs2	rs1	101	rd	0110011	DIVU
0000001	rs2	rs1	110	rd	0110011	REM
0000001	rs2	rs1	111	rd	0110011	REMU

FIGURE 9 : RV32IM OPCODES and Instruction Set Formats

4. Pipeline Datapath

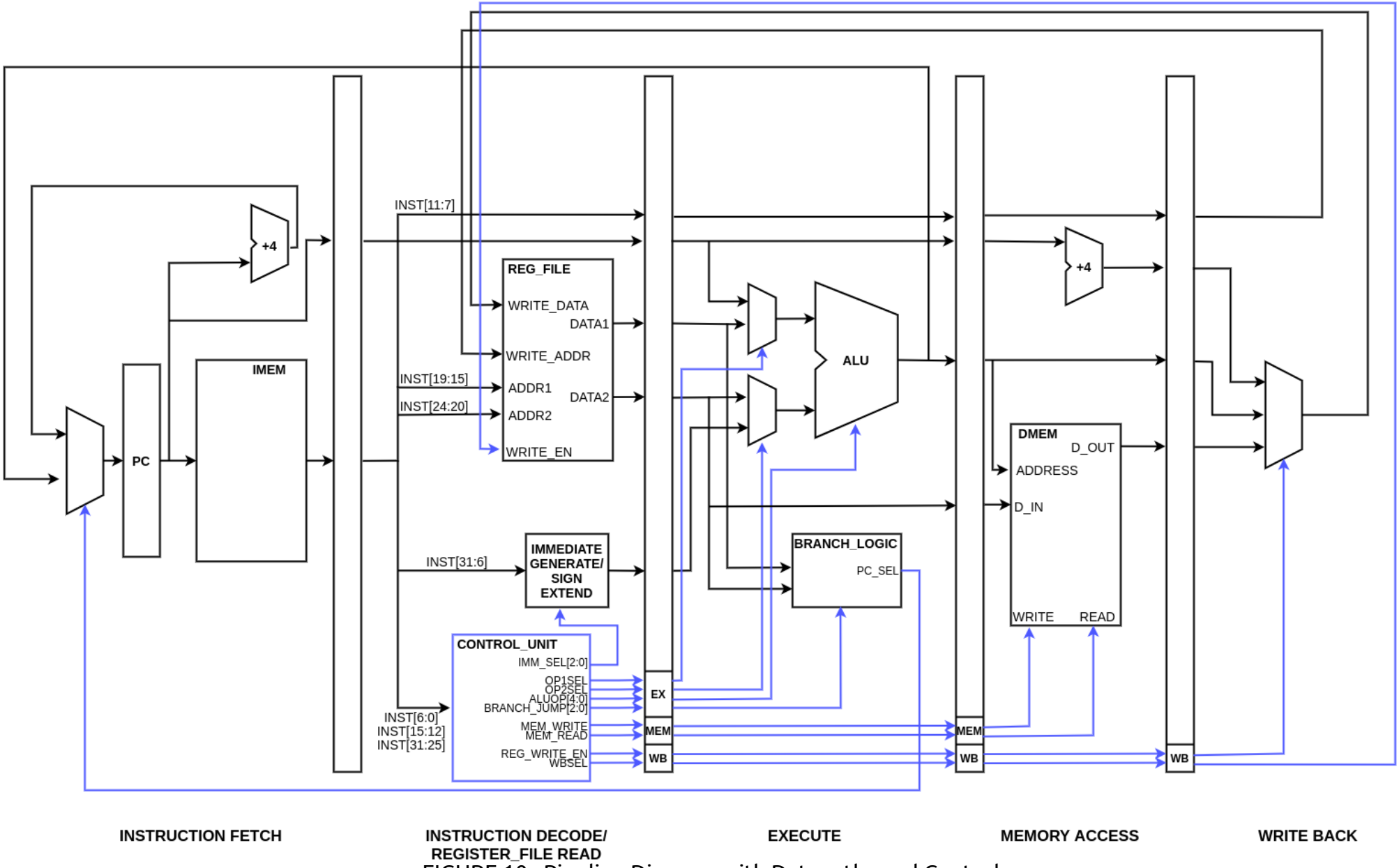

FIGURE 10 : Pipeline Diagram with Datapaths and Controls

5. Hardware Units

- Instruction Memory
- Register File (32 x 32 bit registers)
- ALU (Multiplication and shifting hardware included)
- Data Memory
- Control Unit
- Branch logic (For comparing and generating the control signal related to branch and jump instruction)
- Immediate selecting hardware (Hardware to select the correct immediate format and for sign extending)

6. Control Signals

- ALUOP - ALU operating selection
- REG_WRITE_EN - Enable the writing to the Register File
- PC_SEL - Select the source to the PC register
- IMM_SEL - Select the correct format of the immediate
- OP1SEL - Select the ALU operand 1 source
- OP2SEL - Select the ALU operand 2 source
- MEM_WRITE - Enable writing to the data memory
- MEM_READ - Enable reading from the data memory
- WB_SEL - Select the write back value source
- BRANCH_JUMP - Select the branch comparing type and the jump

INSTRUCTION	PC_SEL	IMM_SEL	OP1SEL	OP2SEL	ALUOP	MEM_WRITE	MEM_READ	REG_WRITE_EN	WB_SEL	BRANCH_JUMP
LUI	PC + 4	U	*	IMM	FORWARD	0	0	1	ALU	000
AUIPC	PC + 4	U	PC	IMM	ADD	0	0	1	ALU	000
JAL	ALU	J	PC	IMM	ADD	0	0	1	PC + 4	111
JALR	ALU	I	DATA1	IMM	ADD	0	0	1	PC + 4	111
BEQ	PC + 4 / ALU	B	PC	IMM	ADD	0	0	0	*	001
BNE	PC + 4 / ALU	B	PC	IMM	ADD	0	0	0	*	010
BLT	PC + 4 / ALU	B	PC	IMM	ADD	0	0	0	*	011
BGE	PC + 4 / ALU	B	PC	IMM	ADD	0	0	0	*	100
BLTU	PC + 4 / ALU	B	PC	IMM	ADD	0	0	0	*	101
BGEU	PC + 4 / ALU	B	PC	IMM	ADD	0	0	0	*	110
LB	PC + 4	I	DATA1	IMM	ADD	0	1	1	MEM	000
LH	PC + 4	I	DATA2	IMM	ADD	0	1	1	MEM	000
LW	PC + 4	I	DATA3	IMM	ADD	0	1	1	MEM	000
LBU	PC + 4	I	DATA4	IMM	ADD	0	1	1	MEM	000
LHU	PC + 4	I	DATA5	IMM	ADD	0	1	1	MEM	000
SB	PC + 4	S	DATA6	IMM	ADD	1	0	0	*	000
SH	PC + 4	S	DATA7	IMM	ADD	1	0	0	*	000
SW	PC + 4	S	DATA8	IMM	ADD	1	0	0	*	000
ADDI	PC + 4	I	DATA1	IMM	ADD	0	0	1	ALU	000
SLTI	PC + 4	I	DATA1	IMM	SLT	0	0	1	ALU	000
SLTIU	PC + 4	IU	DATA1	IMM	SLTU	0	0	1	ALU	000
XORI	PC + 4	I	DATA1	IMM	XOR	0	0	1	ALU	000
ORI	PC + 4	I	DATA1	IMM	OR	0	0	1	ALU	000
ANDI	PC + 4	I	DATA1	IMM	AND	0	0	1	ALU	000
SLLI	PC + 4	SFT	DATA1	IMM	SLL	0	0	1	ALU	000
SRLI	PC + 4	SFT	DATA1	IMM	SRL	0	0	1	ALU	000
SRAI	PC + 4	SFT	DATA1	IMM	SRA	0	0	5	ALU	000

ADD	PC + 4	*	DATA1	DATA2	ADD	0	0	1	ALU	000
SUB	PC + 4	*	DATA1	DATA2	SUB	0	0	1	ALU	000
SLL	PC + 4	*	DATA1	DATA2	SLL	0	0	1	ALU	000
SLT	PC + 4	*	DATA1	DATA2	SLT	0	0	1	ALU	000
SLTU	PC + 4	*	DATA1	DATA2	SLTU	0	0	1	ALU	000
XOR	PC + 4	*	DATA1	DATA2	XOR	0	0	1	ALU	000
SRL	PC + 4	*	DATA1	DATA2	SRL	0	0	1	ALU	000
SRA	PC + 4	*	DATA1	DATA2	SRA	0	0	1	ALU	000
OR	PC + 4	*	DATA1	DATA2	OR	0	0	1	ALU	000
AND	PC + 4	*	DATA1	DATA2	AND	0	0	1	ALU	000
MUL	PC + 4	*	DATA1	DATA2	MUL	0	0	1	ALU	000
MULH	PC + 4	*	DATA1	DATA2	MULH	0	0	1	ALU	000
MULHSU	PC + 4	*	DATA1	DATA2	MULHSU	0	0	1	ALU	000
MULHU	PC + 4	*	DATA1	DATA2	MULHU	0	0	1	ALU	000
DIV	PC + 4	*	DATA1	DATA2	DIV	0	0	1	ALU	000
DIVU	PC + 4	*	DATA1	DATA2	DIVU	0	0	1	ALU	000
REM	PC + 4	*	DATA1	DATA2	REM	0	0	1	ALU	000
REMU	PC + 4	*	DATA1	DATA2	REMU	0	0	1	ALU	000

TABLE 6 : Instructions and Related Control Signals