BUILDING A BASIC RV32IM PIPELINE

PART O1

GROUP - 03

E/18/058 - DE ALWIS K.K.M.

E/18/170 - KARUNARATHNA W.K.

E/18/203 - MADHUSANKA K.G.A.S.

TABLE OF CONTENTS

1. INTRODUCTION

RV32I Instruction Set Architecture is a 32-bit ISA for the RISC-V instruction set architecture family. It is the base integer ISA for RISC-V and includes a set of instructions that perform basic operations on 32-bit integers.

RV32I has 32 registers, each 32 bits wide. A special purpose register x0 is hardwired to the constant value zero and cannot be modified. The remaining 31 registers (x1 - x31) are general-purpose registers that can be used for storing data and address pointers. Another important special purpose register is the program counter (pc) which the CPU uses to remember the memory address where its program instructions are located.

2. BASE INSTRUCTION FORMATS

There are four core instruction formats in the base RV32I Instruction set architecture. They are *R type, I type, S type and U type*. All are fixed 32-bits in length. There are further two types of instruction formats which are *B type and J type* based on the handling of immediate values. Below is the diagram of how these types of instruction formats are encoded.

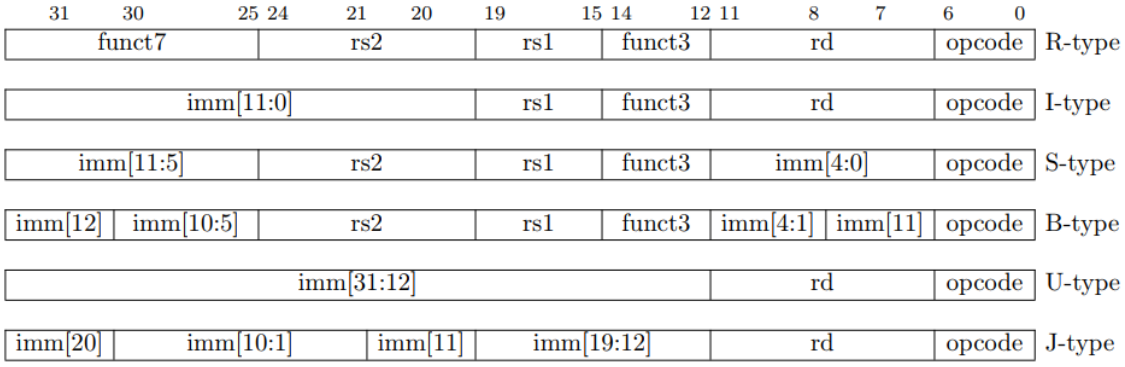

Figure 1: RISC-V base instruction formats

All instructions have their *opcode* in bits 0-6 and when they include an *rd register* it will be specified in bits 7-11, and *rs1 register* in bits 15-19, and *rs2 register* in bits 20-24, and so on. This has a seemingly strange impact on the placement of any immediate operands.

When immediate operands are present in an instruction, they are placed in the remaining unused bits. However, they are organized such that the sign bit is always in bit 31 and the remaining bits placed so as to minimize the number of places any given bit is located in different instructions.

3. RV32IM INSTRUCTIONS

Usage Template		Type	Description	Detailed Description
add	rd, rs1, rs2	R	Add	rd ← rs1 + rs2, pc ← pc+4
addi	rd, rs1, imm	I	Add Immediate	rd ← rs1 + imm_i, pc ← pc+4
and	rd, rs1, rs2	R	And	rd ← rs1 ∧ rs2, pc ← pc+4
andi	rd, rs1, imm	I	And Immediate	rd ← rs1 ∧ imm i, pc ← pc+4
auipc	rd, imm	U	Add Upper Immediate to PC	rd ← pc + imm_u, pc ← pc+4
beq	rs1, rs2, offset	B	Branch Equal	pc ← pc + ((rs1==rs2) ? (offset<<1) : 4)
bge	rs1, rs2, offset	B	Branch Greater or Equal	pc ← pc + ((rs1>=rs2) ? (offset<<1) : 4)
bgeu	rs1, rs2, offset	B	Branch Greater or Equal Unsigned	pc ← pc + ((rs1>=rs2) ? (offset<<1) : 4)
blt	rs1, rs2, offset	B	Branch Less Than	pc ← pc + ((rs1<rs2) ? (offset<<1) : 4)
bltu	rs1, rs2, offset	B	Branch Less Than Unsigned	pc ← pc + ((rs1<rs2) ? (offset<<1) : 4)
bne	rs1, rs2, offset	B	Branch Not Equal	pc ← pc + ((rs1!=rs2) ? (offset<<1) : 4)
ecall		I	Environment Call	Transfer Control to Debugger
ebreak		I	Environment Break	Transfer Control to Operating System
jal	rd, offset	J	Jump And Link	rd ← pc+4, pc ← pc+ (offset<<1)
jalr	rd,rs1,offset	I	Jump And Link Register	rd ← pc+4, pc ← rs1 + (offset<<1)
lb	rd, imm(rs1)	I	Load Byte (8 bits)	rd ← sx(m8(rs1+imm i)), pc ← pc+4
lbu	rd, imm(rs1)	I	Load Byte Unsigned	rd ← zx(m8(rs1+imm i)), pc ← pc+4
lh	rd, imm(rs1)	I	Load Halfword (16 bits)	rd ← sx(m16(rs1+imm i)), pc ← pc+4
lhu	rd, imm(rs1)	I	Load Halfword Unsigned	rd ← zx(m16(rs1+imm i)), pc ← pc+4
lui	rd, imm	U	Load Upper Immediate	rd ← imm u, pc ← pc+4
lw	rd, imm(rs1)	I	Load Word (32 bits)	rd ← sx(m32(rs1+imm i)), pc ← pc+4
or	rd, rs1, rs2	R	Or	rd ← rs1 ∨ rs2, pc ← pc+4
ori	rd, rs1, imm	I	Or Immediate	rd ← rs1 ∨ imm i, pc ← pc+4
sb	rs2, imm(rs1)	S	Store Byte	m8(rs1+imm_s) ← rs2[7:0], pc ← pc+4
sh	rs2, imm(rs1)	S	Store Halfword	m16(rs1+imm_s) ← rs2[15:0], pc ← pc+4
sll	rd, rs1, rs2	R	Shift Left Logical	rd ← rs1 << rs2, pc ← pc+4
slli	rd, rs1, shamt	I	Shift Left Logical Immediate	rd ← rs1 << shamt_i, pc ← pc+4
slt	rd, rs1, rs2	R	Set Less Than	rd ← (rs1 < rs2) ? 1 : 0, pc ← pc+4
slti	rd, rs1, imm	I	Set Less Than Immediate	rd ← (rs1 < imm i) ? 1 : 0, pc ← pc+4
sltiu	rd, rs1, imm	I	Set Less Than Immediate Unsigned	rd ← (rs1 < imm i) ? 1 : 0, pc ← pc+4
sltu	rd, rs1, rs2	R	Set Less Than Unsigned	rd ← (rs1 < rs2) ? 1 : 0, pc ← pc+4
sra	rd, rs1, rs2	R	Shift Right Arithmetic	rd ← rs1 >> rs2, pc ← pc+4
srai	rd, rs1, shamt	I	Shift Right Arithmetic Immediate	rd ← rs1 >> shamt_i, pc ← pc+4
srl	rd, rs1, rs2	R	Shift Right Logical	rd ← rs1 >> rs2, pc ← pc+4
srli	rd, rs1, shamt	I	Shift Right Logical Immediate	rd ← rs1 >> shamt_i, pc ← pc+4
sub	rd, rs1, rs2	R	Subtract	rd ← rs1 - rs2, pc ← pc+4
sw	rs2, imm(rs1)	S	Store Word	m32(rs1+imm_s) ← rs2[31:0], pc ← pc+4
xor	rd, rs1, rs2	R	Exclusive Or	rd ← rs1 ⊕ rs2, pc ← pc+4
xori	rd, rs1, imm	I	Exclusive Or Immediate	rd ← rs1 ⊕ imm i, pc ← pc+4
mul	rd, rs1, rs2	R	Multiplication	rd ← rs1 X rs2, pc ← pc+4
mulh	rd, rs1, rs2	R	Multiplication High	rd ← (rs1 s×s rs2) >>s XLEN
mulhu	rd, rs1, rs2	R	Multiplication on High Unsigned	rd ← (rs1 s×s rs2) >>s XLEN
mulhsu	rd, rs1, rs2	R	Multiplication on High Signed Unsigned	rd ← (rs1 s×s rs2) >>s XLEN
div	rd, rs1, rs2	R	Division	rd ← rs1 / rs2, pc ← pc+4
divu	rd, rs1, rs2	R	Unsigned Division	rd ← rs1 /$_u$ rs2, pc ← pc+4
rem	rd, rs1, rs2	R	Remainder	rd ← rs1 % rs2, pc ← pc+4
remu	rd, rs1, rs2	R	Remainder Unsigned	rd ← rs1 %$_u$ rs2, pc ← pc+4

Table 1: RV32IM Instructions

3.1 Integer Computational Instructions

3.1.1 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

31 12	11 7	6 0
imm[31:12]	rd	opcode
20	5	7
U-immediate[31:12]	dest	LUI
U-immediate[31:12]	dest	AUIPC

Figure 2: Register-Immediate instruction encoding formats

3.1.2 Register - Register Instructions

31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	src2	src1	ADD/SLT/SLTU	dest	OP
0000000	src2	src1	AND/OR/XOR	dest	OP
0000000	src2	src1	SLL/SRL	dest	OP
0100000	src2	src1	SUB/SRA	dest	OP

Figure 3: Register-Register instruction encoding formats

3.2 Control Transfer Instructions

3.2.1 Unconditional Jumps

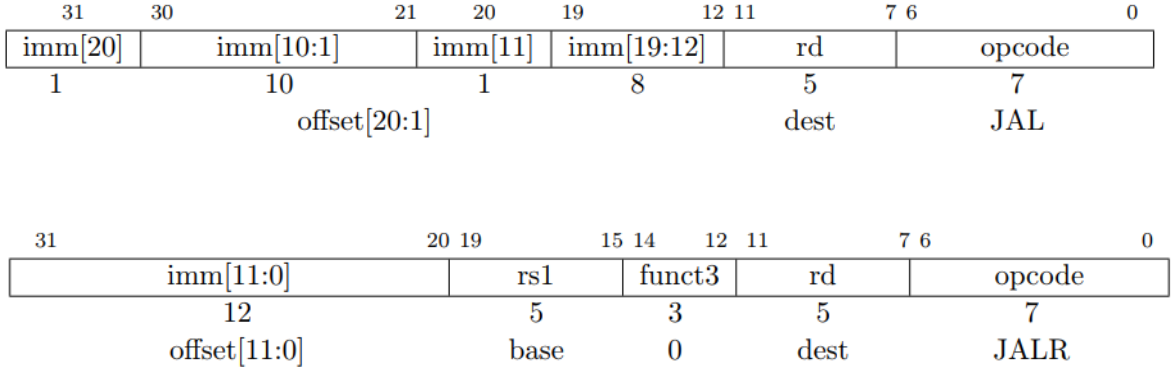

Figure 4: Unconditional Jumps instruction encoding formats

3.2.1 Conditional Branches

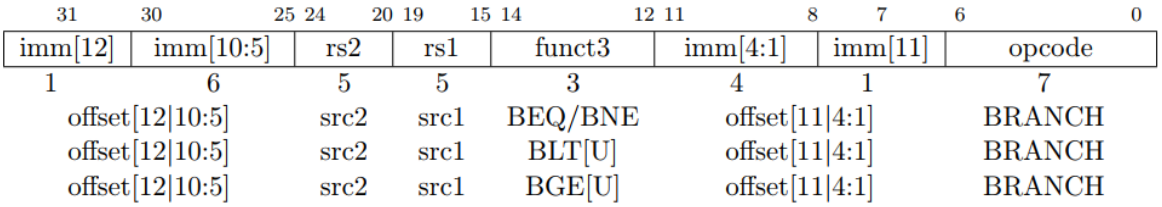

Figure 5: Conditional Branches instruction encoding formats

3.2 Load and Store Instructions

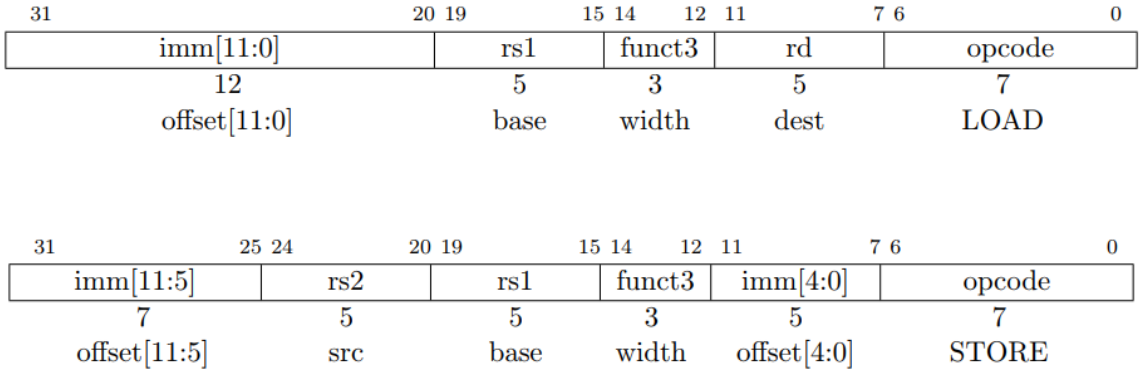

Figure 6: Load and Store instruction encoding formats

3.3 Multiplication / Division Instructions

31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
MULDIV	multiplier	multiplicand	MUL/MULH[[S]U]	dest	OP
MULDIV	multiplier	multiplicand	MULW	dest	OP-32

31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
MULDIV	divisor	dividend	DIV[U]/REM[U]	dest	OP
MULDIV	divisor	dividend	DIV[U]W/REM[U]W	dest	OP-32

Figure 7: Multiplication and Division instruction encoding formats

3.4 Memory Ordering Instructions

Memory ordering instructions are used to ensure that memory accesses are executed in a specific order.

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. It provides explicit synchronization between writes to instruction memory and instruction fetches on the same hart.

31 28	27	26	25	24	23	22	21	20	19 15	14 12	11 7	6 0
fm	PI	PO	PR	PW	SI	SO	SR	SW	rs1	funct3	rd	opcode
4	1	1	1	1	1	1	1	1	5	3	5	7
FM	predecessor				successor				0	FENCE	0	MISC-MEM

Figure 8: Memory Ordering instruction encoding formats

3.5 Environment Call and Breakpoints

In the RV32I instruction set architecture, there are two instructions related to environment calls and breakpoints.

ECALL instruction is used to make a service request to the execution environment.

EBREAK instruction is used to return control to a debugging environment.

31 20	19 15	14 12	11 7	6 0
funct12	rs1	funct3	rd	opcode
12	5	3	5	7
ECALL	0	PRIV	0	SYSTEM
EBREAK	0	PRIV	0	SYSTEM

Figure9:System instruction encoding formats

4. OPCODES

Below are the opcodes assigned to each base instruction which RISC-V ISA used to implement the RV32IM.

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

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

Figure 10: RV32I Base instructions opcodes

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 11: RV32IM Extended instruction opcodes

5. PIPELINE DATAPATH

The basic RV32IM pipeline datapath consists of five stages: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), and Write-Back (WB).

The five pipeline stages are as follows:

1. Instruction Fetch (IF): The program counter (PC) is used to fetch the instruction from the instruction memory (IM). The fetched instruction is stored in the instruction register (IR).

2. Instruction Decode (ID): The instruction register (IR) is decoded to determine the instruction type and its operands. The register file (RF) is read to obtain the values of the operands.

3. Execute (EX): The ALU performs the operation specified by the instruction. If the instruction is a branch or jump, the target address is calculated.

4. Memory Access (MEM): If the instruction is a load or store, the data memory (DM) is accessed to read or write data.

5. Write-Back (WB): The result of the operation is written back to the register file (RF).

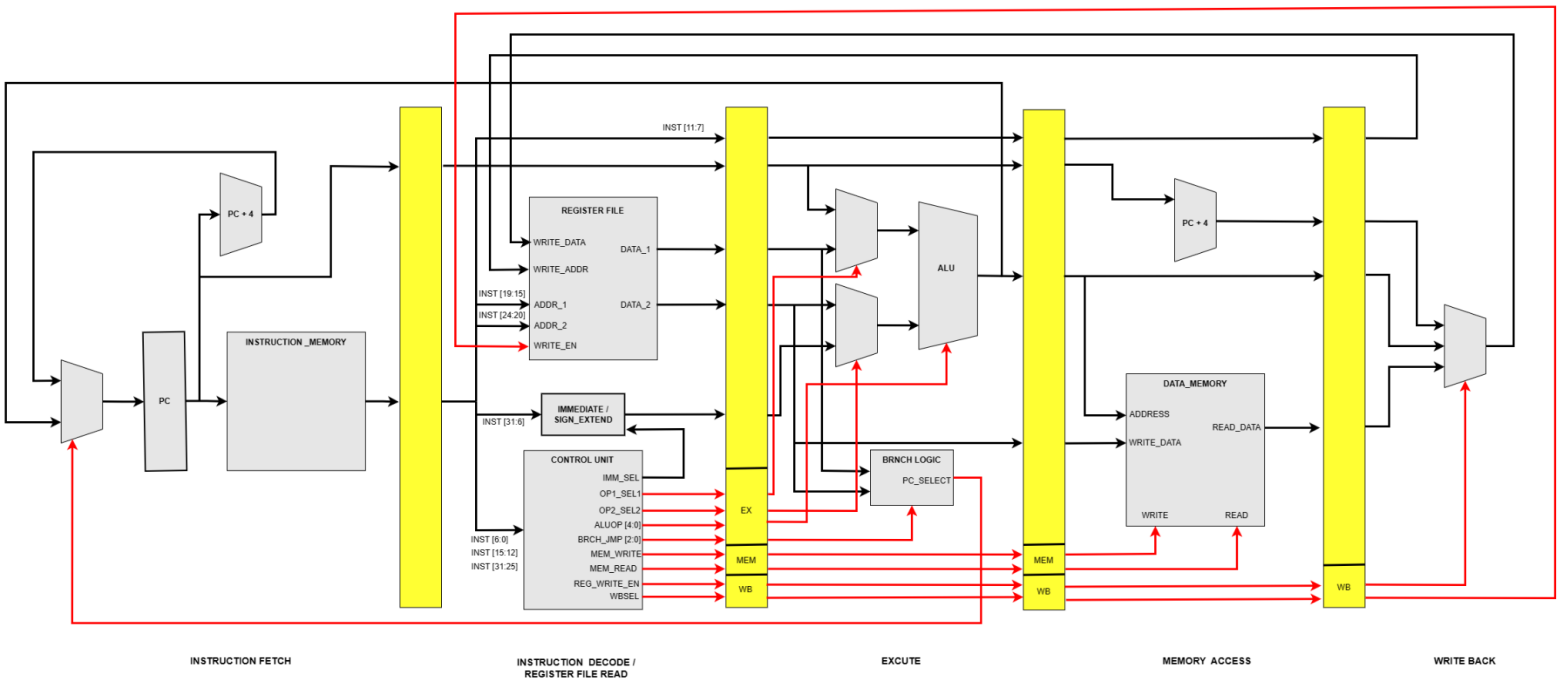

Figure 11: pipeline datapath

Full image of the datapath can be seen here. [Datapath.png]

6. 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.

7. 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

TABLE : INSTRUCTIONS AND RELATED CONTROL SIGNALS

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
AUPIC	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	*	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	MEM	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	*	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	ALU	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	1	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