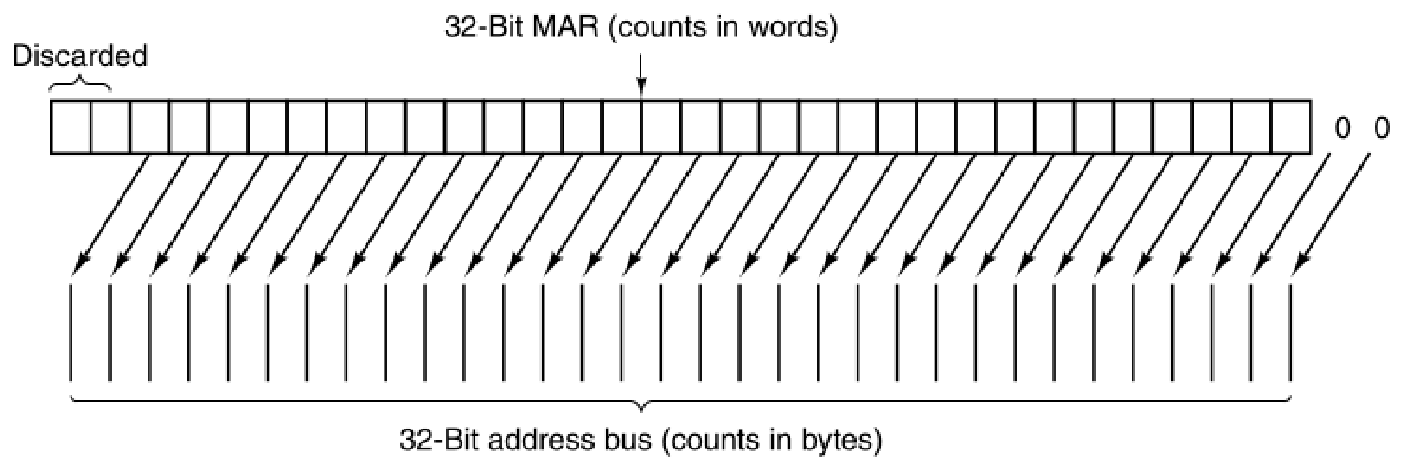
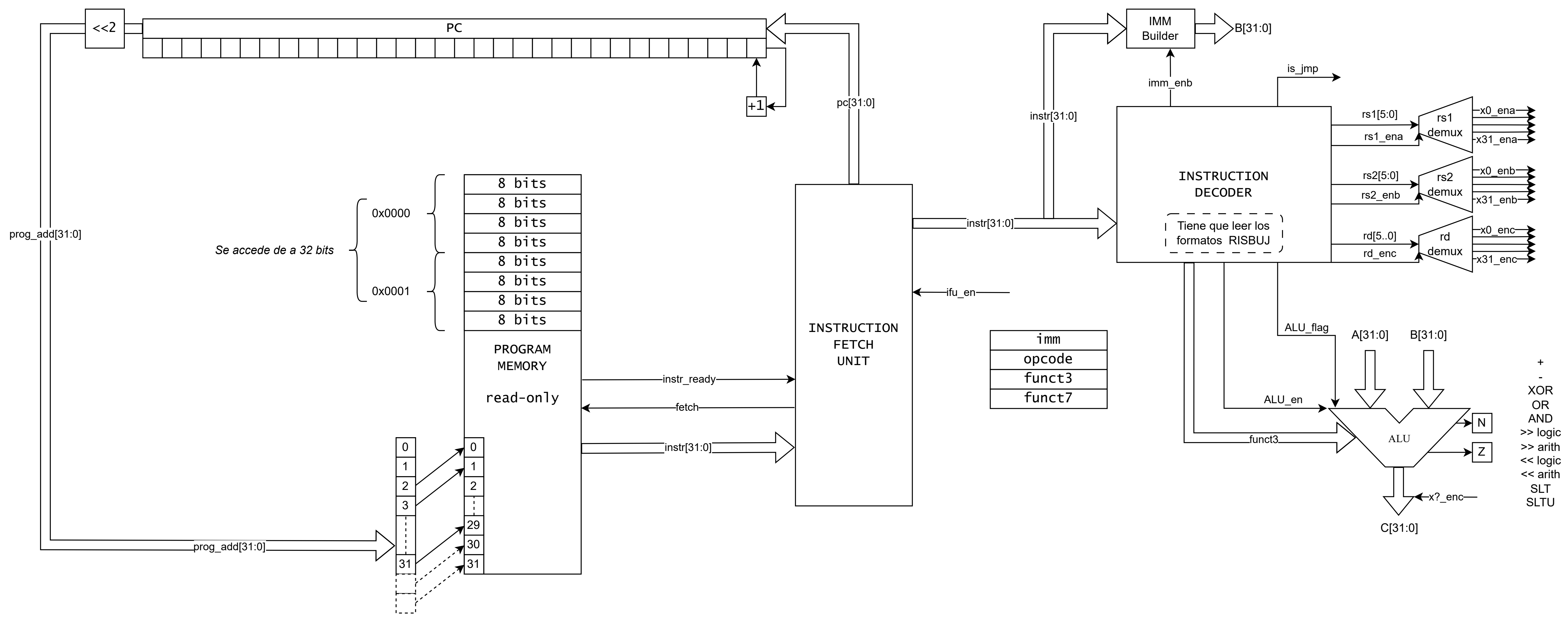
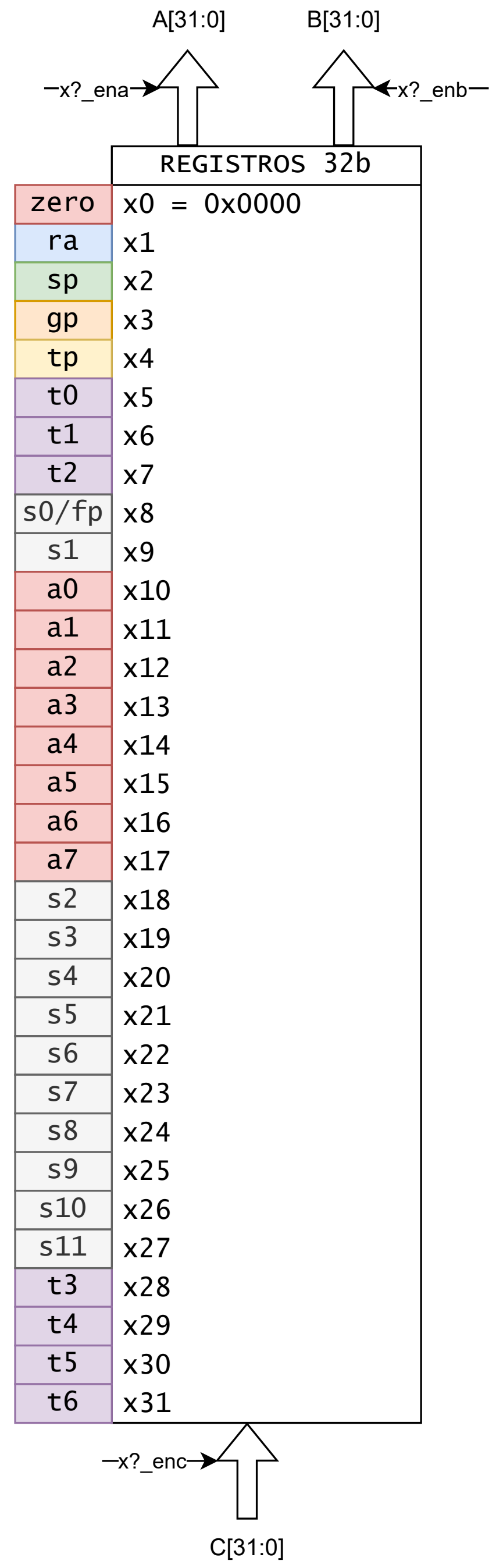
Fig. 4-4.



Discarded
32-Bit MAR (counts in words)
0 0
32-Bit address bus (counts in bytes)





	REGISTROS 32b
zero	x0 = 0x0000
ra	x1
sp	x2
gp	x3
tp	x4
t0	x5
t1	x6
t2	x7
s0/fp	x8
s1	x9
a0	x10
a1	x11
a2	x12
a3	x13
a4	x14
a5	x15
a6	x16
a7	x17
s2	x18
s3	x19
s4	x20
s5	x21
s6	x22
s7	x23
s8	x24
s9	x25
s10	x26
s11	x27
t3	x28
t4	x29
t5	x30
t6	x31

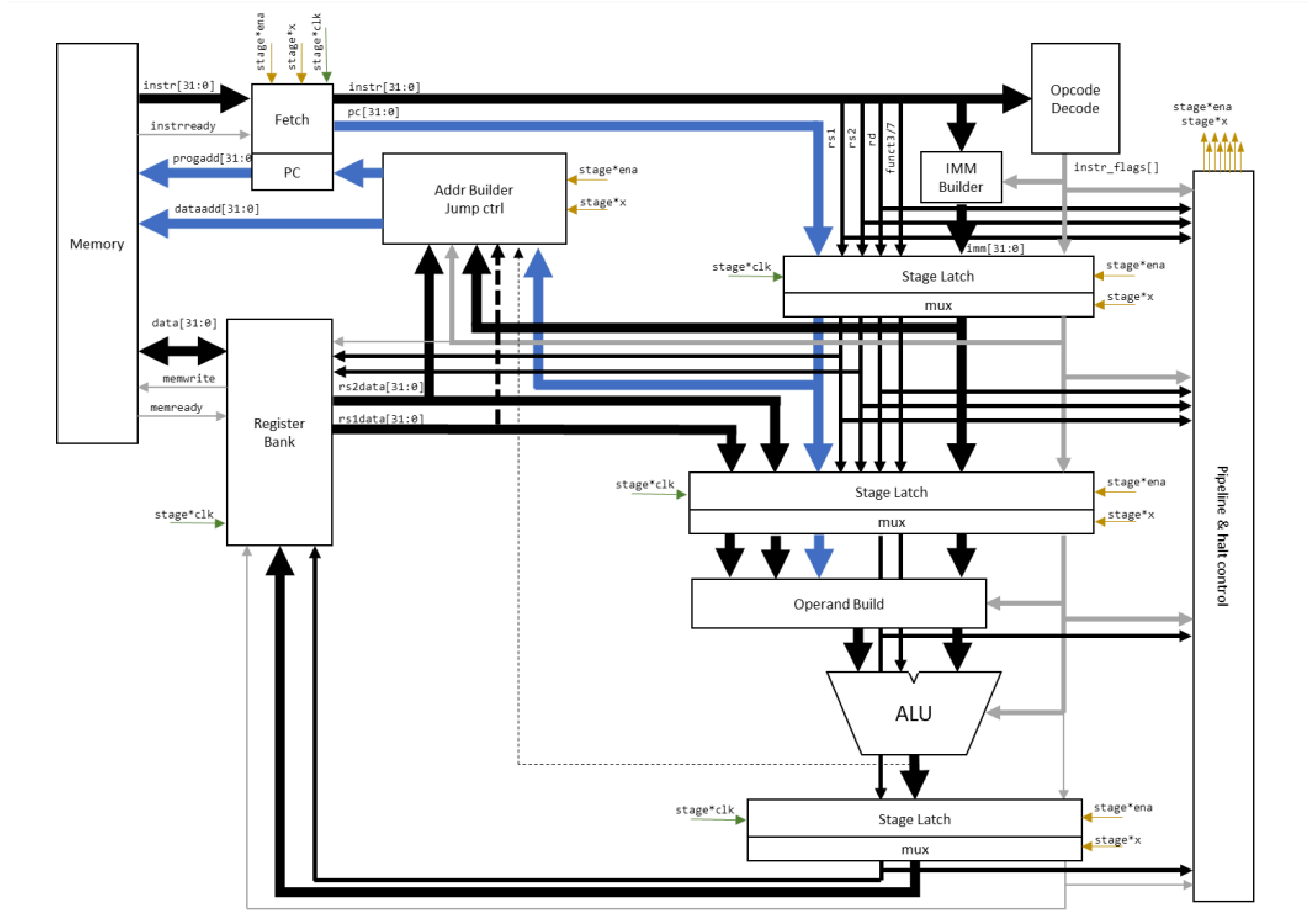
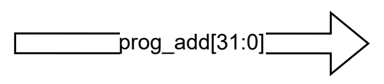


Figura 2.1 – Prototipo de arquitectura con pipeline de 5 etapas



R-type					
funct7	rs2	rs1	funct3	rd	opcode

I-type				
imm[11:0]	rs1	funct3	rd	opcode

S-type					
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode

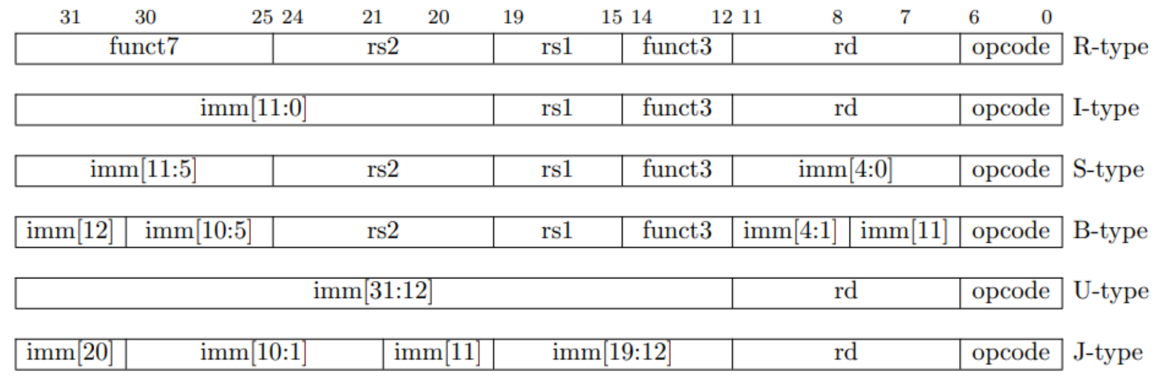
S-type							
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode

U-type		
imm[31:12]	rd	opcode

J-type					
imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode

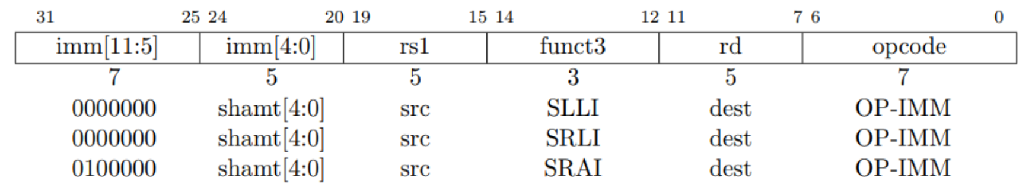
opcode	funct3	Instr Format
LUI	xxx	U
AUIPC	xxx	U
JAL	xxx	J
JALR	xxx	I
LOAD	xxx	I
OP-IMM	not[001,101]	I
OP-IMM	001 or 101	R
OP	xxx	R
STORE	xxx	S
BRANCH	xxx	B
MISC-MEM	xxx	I [N/A]
SYSTEM	xxx	I [N/A]

Tabla 24 – Formato de instrucciones

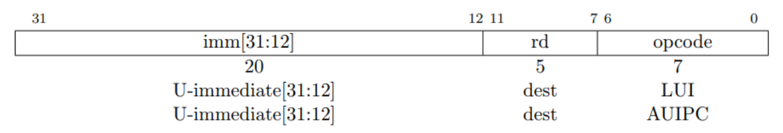


31 30 25	24 21 20	19 15	14 12	11 8 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type

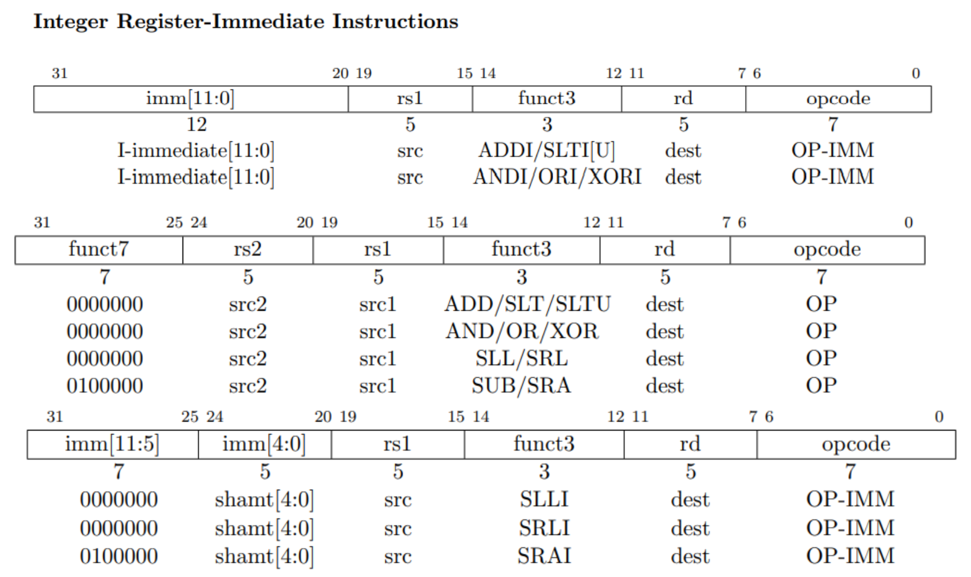
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1] | imm[11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20]	imm[10:1]	imm[11]	imm[19:12]		rd	opcode	J-type



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



Integer 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
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

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

A consultar:

1. Sobre puertos de lectura y escritura, y simbiosis con banco de registros
2. asd

PUERTOS DE LECTURA?
PUERTO DE ESCRITURA?

We note that various microarchitectural techniques exist to dynamically convert unpredictable short forward branches into internally predicated code to avoid the cost of flushing pipelines on a branch mispredict [6, 10, 9] and have been implemented in commercial processors [17]. The simplest techniques just reduce the penalty of recovering from a mispredicted short forward branch by only flushing instructions in the branch shadow instead of the entire fetch pipeline, or by fetching instructions from both sides using wide instruction fetch or idle instruction fetch slots. More complex techniques for out-of-order cores add internal predicates on instructions in the branch shadow, with the internal predicate value written by the branch instruction, allowing the branch and following instructions to be executed speculatively and out-of-order with respect to other code [17].