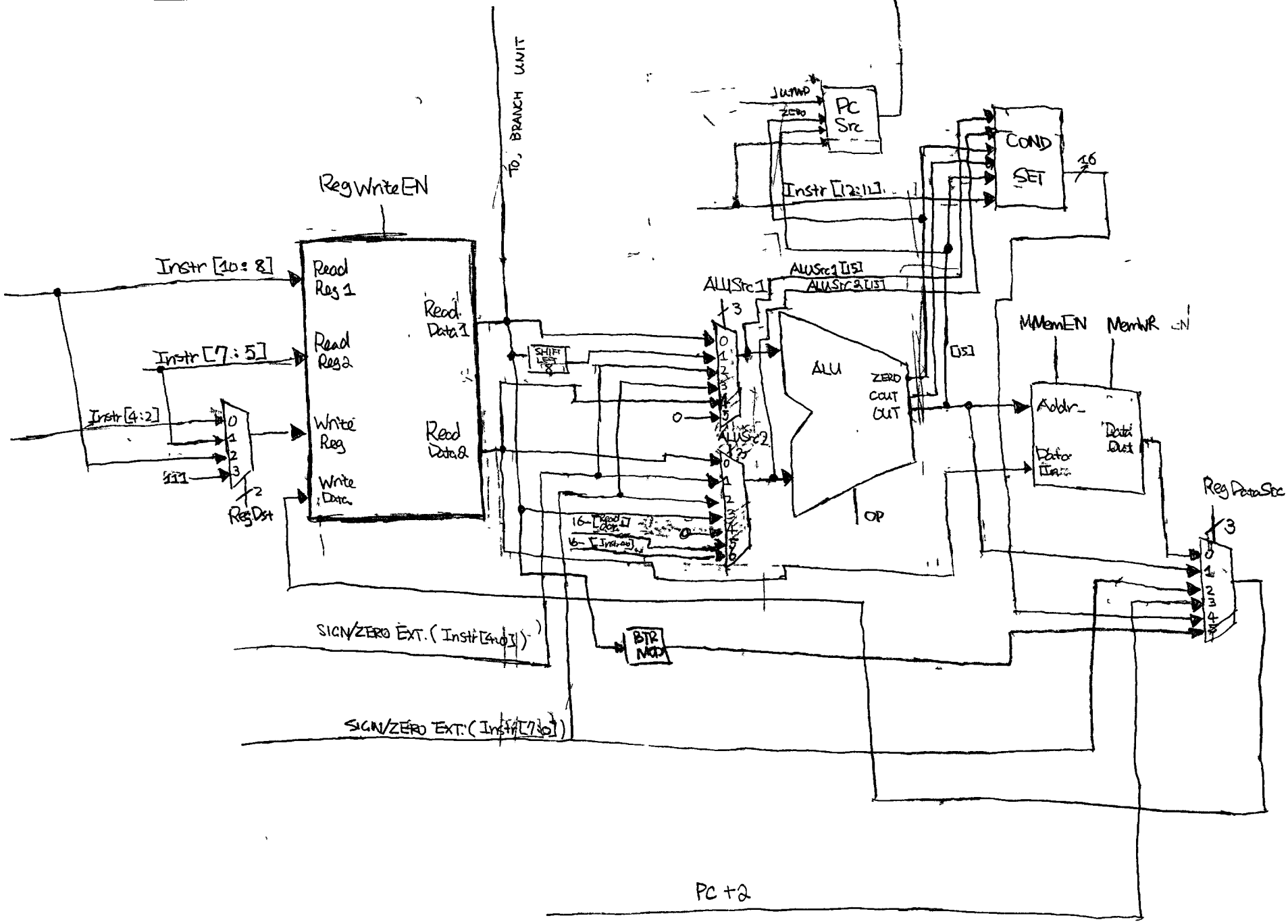
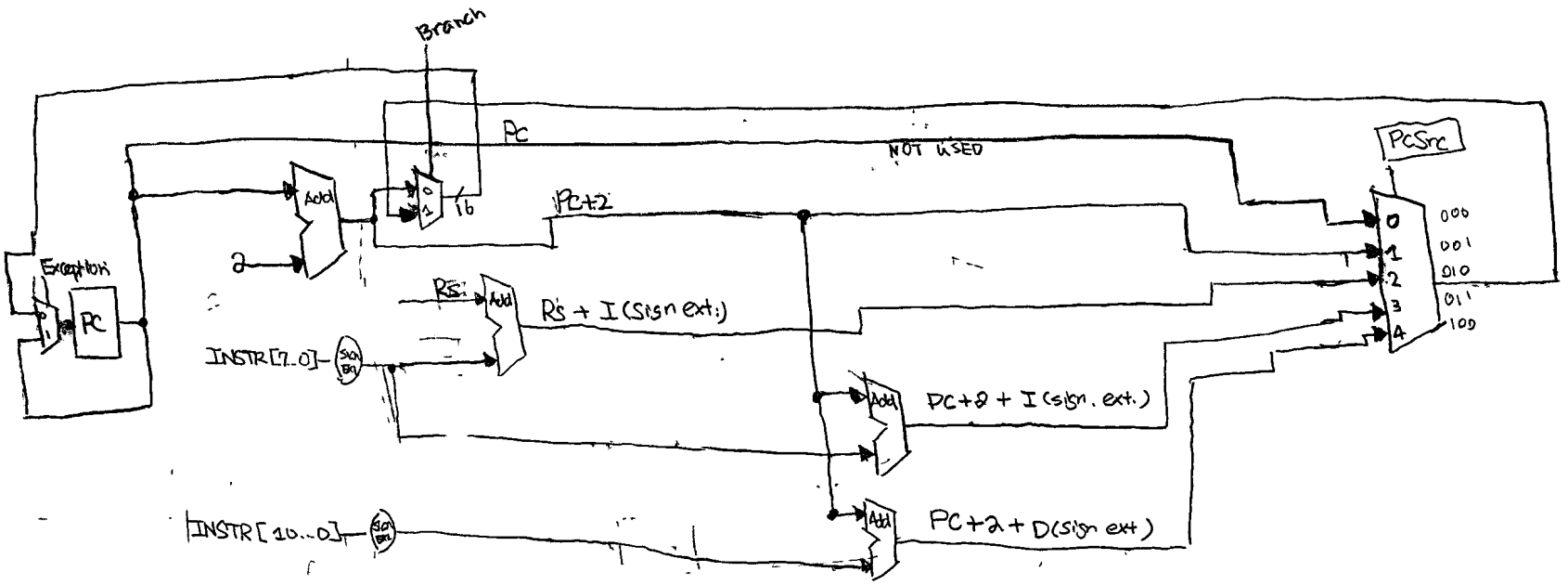
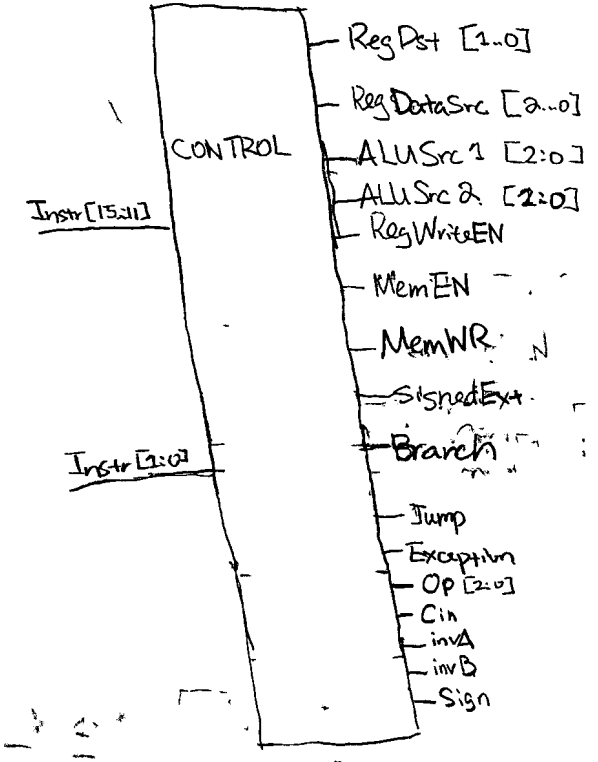
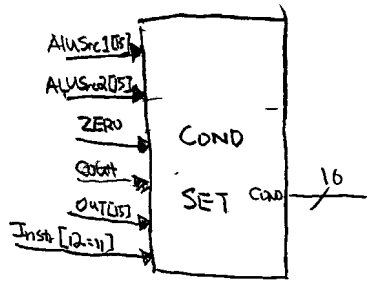
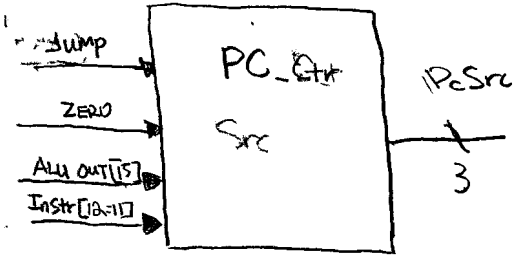

TO BRANCH UNIT
JUMP
ZERO
PC Src
COND
SET
16
Instr [12:11]
RegWriteEN
Instr [10:8]
Instr [7:5]
Instr [4:2]
111
Read Reg 1
Read Reg 2
Write Reg
Write Data
Read Data 1
Read Data 2
RegDst
SHIFT LEFT 1
ALUSrc1
ALUSrc2
3
ALUSrc1 [5]
ALUSrc2 [5]
ALU
ZERO
COUT
OUT
OP
[5]
MMemEN
MemWR
Addr
Data In
Data Out
RegDataSrc
SIGN/ZERO EXT. (Instr[4:0])
BTR MOD
SIGN/ZERO EXT. (Instr[7:0])
Pc +2


Branch
Pc
NOT USED
PCSrc
Add
2
0
1
16
PC+2
Exception
1
PC
RS
Add
Rs + I (sign ext.)
INSTR[7..0]
Sign Ext
000
001
010
011
100
0
1
2
3
4
Add
PC+2 + I (sign. ext.)
INSTR[10..0]
Sign Ext
Add
PC+2 + D(sign ext)

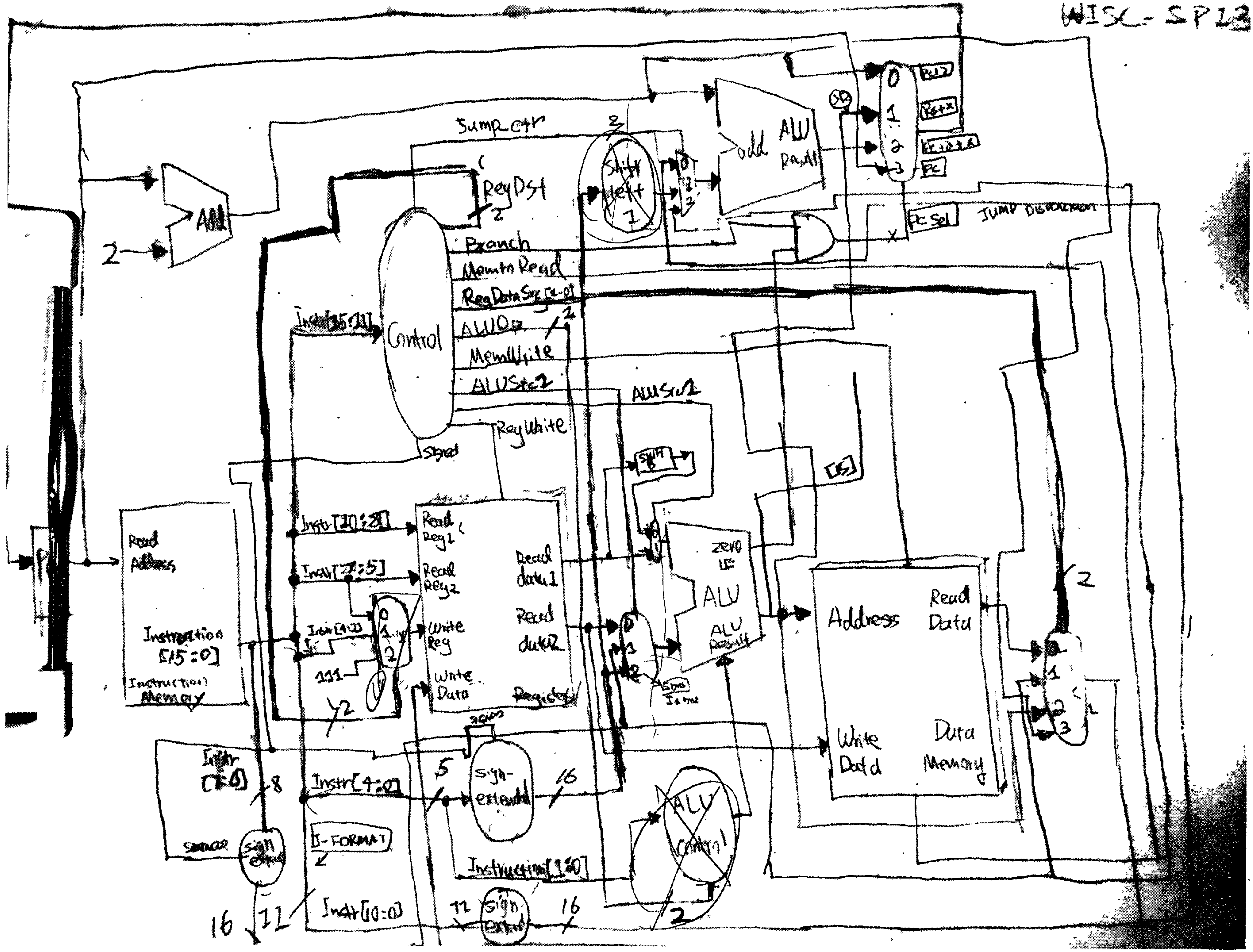






PC Op
0 — Hold PC
1 — PC+2
2 — PC+2+D
3 — Rs+I
4 — P If (true) PC+2+I

WISC-SP13



Add
2
Read Address
Instruction [15:0]
Instruction Memory
Control
Instr[15:11]
RegDst
Branch
MemtoRead
RegDataSrc
ALUOp
MemWrite
ALUSrc2
RegWrite
Signed
Jump ctr
Shift left 1
add ALU Result
PC Sel
JUMP DISPLACEMENT
PC
ALUSrc1
Read Reg1
Read Reg2
Write Reg
Write Data
Read data1
Read data2
Registers
Instr[10:8]
Instr[7:5]
Instr[4:2]
111
sign-extend
Instr[4:0]
Instr[7:0]
Instruction[1:0]
I-FORMAT
Instr[10:0]
sign extend
Zero
ALU
ALU Result
ALU Control
Address
Read Data
Write Data
Data Memory
16
11
8
5

WISC-SP13 Instruction Set Architecture

On this page... (hide)

1. Instruction Summary

Instruction Format	Syntax	Semantics
00000 xxxxxxxxxxx	HALT	Cease instruction issue, dump memory state to file
00001 xxxxxxxxxxx	NOP	None
01000 sss ddd iiiii	ADDI Rd, Rs, immediate	Rd <- Rs + I(sign ext.)
01001 sss ddd iiiii	SUBI Rd, Rs, immediate	Rd <- I(sign ext.) - Rs
01010 sss ddd iiiii	XORI Rd, Rs, immediate	Rd <- Rs XOR I(zero ext.)
01011 sss ddd iiiii	ANDNI Rd, Rs, immediate	Rd <- Rs AND ~I(zero ext.)
10100 sss ddd iiiii	ROLI Rd, Rs, immediate	Rd <- Rs <<(rotate) I(lowest 4 bits)
10101 sss ddd iiiii	SLLI Rd, Rs, immediate	Rd <- Rs << I(lowest 4 bits)
10110 sss ddd iiiii	RORI Rd, Rs, immediate	Rd <- Rs >>(rotate) I(lowest 4 bits)
10111 sss ddd iiiii	SRLI Rd, Rs, immediate	Rd <- Rs >> I(lowest 4 bits)
10000 sss ddd iiiii	ST Rd, Rs, immediate	Mem[Rs + I(sign ext.)] <- Rd
10001 sss ddd iiiii	LD Rd, Rs, immediate	Rd <- Mem[Rs + I(sign ext.)]
10011 sss ddd iiiii	STU Rd, Rs, immediate	Mem[Rs + I(sign ext.)] <- Rd Rs <- Rs + I(sign ext.)
11001 sss xxx ddd xx	BTR Rd, Rs	Rd[bit i] <- Rs[bit 15-i] for i=0..15

11011 sss ttt ddd 00	ADD Rd, Rs, Rt	Rd <- Rs + Rt
11011 sss ttt ddd 01	SUB Rd, Rs, Rt	Rd <- Rt - Rs
11011 sss ttt ddd 10	XOR Rd, Rs, Rt	Rd <- Rs XOR Rt
11011 sss ttt ddd 11	ANDN Rd, Rs, Rt	Rd <- Rs AND ~Rt
11010 sss ttt ddd 00	ROL Rd, Rs, Rt	Rd <- Rs << (rotate) Rt (lowest 4 bits)
11010 sss ttt ddd 01	SLL Rd, Rs, Rt	Rd <- Rs << Rt (lowest 4 bits)
11010 sss ttt ddd 10	ROR Rd, Rs, Rt	Rd <- Rs >> (rotate) Rt (lowest 4 bits)
11010 sss ttt ddd 11	SRL Rd, Rs, Rt	Rd <- Rs >> Rt (lowest 4 bits)
11100 sss ttt ddd xx	SEQ Rd, Rs, Rt	if (Rs == Rt) then Rd <- 1 else Rd <- 0
11101 sss ttt ddd xx	SLT Rd, Rs, Rt	if (Rs < Rt) then Rd <- 1 else Rd <- 0
11110 sss ttt ddd xx	SLE Rd, Rs, Rt	if (Rs <= Rt) then Rd <- 1 else Rd <- 0
11111 sss ttt ddd xx	SCO Rd, Rs, Rt	if (Rs + Rt) generates carry out then Rd <- 1 else Rd <- 0
01100 sss iiiiiiii	BEQZ Rs, immediate	if (Rs == 0) then PC <- PC + 2 + I(sign ext.)
01101 sss iiiiiiii	BNEZ Rs, immediate	if (Rs != 0) then PC <- PC + 2 + I(sign ext.)
01110 sss iiiiiiii	BLTZ Rs, immediate	if (Rs < 0) then PC <- PC + 2 + I(sign ext.)
01111 sss iiiiiiii	BGEZ Rs, immediate	if (Rs >= 0) then PC <- PC + 2 + I(sign ext.)
11000 sss iiiiiiii	LBI Rs, immediate	Rs <- I(sign ext.)
10010 sss iiiiiiii	SLBI Rs, immediate	Rs <- (Rs << 8) | I(zero ext.)
00100 ddddddddddd	J displacement	PC <- PC + 2 + D(sign ext.)
00101 sss iiiiiiii	JR Rs, immediate	PC <- Rs + I(sign ext.)
00110 ddddddddddd	JAL displacement	R7 <- PC + 2 PC <- PC + 2 + D(sign ext.)
00111 sss iiiiiiii	JALR Rs, immediate	R7 <- PC + 2 PC <- Rs + I(sign ext.)

00010	siic Rs	produce IllegalOp exception. Must provide one source register.
00011 xxxxxxxxxxx	NOP / RTI	PC <- EPC

EXISTING PC ?

2. Formats

WISC-SP13 supports instructions in four different formats: J-format, 2 I-formats, and the R-format. These are described below.

2.1 J-format

The J-format is used for jump instructions that need a large displacement.

J-Format

5 bits	11 bits
Op Code	Displacement

Jump Instructions

The Jump instruction loads the PC with the value found by adding the PC of the next instruction (PC+2, not PC+4 as in MIPS) to the **sign-extended** displacement.

The Jump-And-Link instruction loads the PC with the same value and also saves the address of the next sequential instruction (i.e., PC+2) in the link register R_7.

The syntax of the jump instructions is:

- `J displacement`
- `JAL displacement`

2.2 I-format

I-format instructions use either a destination register, a source register, and a 5-bit immediate value; or a destination register and an 8-bit immediate value. The two types of I-format instructions are described below.

I-format 1 Instructions

I-format 1

5 bits	3 bits	3 bits	5 bits
Op Code	R_s	R_d	Immediate

The I-format 1 instructions include XOR-Immediate, ANDN-Immediate, Add-Immediate, Subtract-Immediate, Rotate-Left-Immediate, Shift-Left-Logical-Immediate, Rotate-Right-Immediate, Shift-Right-Logical-Immediate, Load, Store, and Store with Update.

RS RT

A[15]	B[15]	Out[15]	Sgt
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

A[15] B[15]

0 1 X

X

RS <= RT

RS <= R

RS <= RT

RS - RT <= 0

RT - RS >= 0

RS - RT <= 0

RS <= RT

RT - RS >= 0

Out[15]