Computer Organisation and Architecture Laboratory

Assignment - 6
Design of a KGP-miniRISC Processor

Lab Report

Group - 28
Members:
Rohit Kumar Prajapati (20CS30041)
Sai Sahan Talabattula (20CS30055)

Instruction Format used:

The following instruction formats have been used for different operations:

★ For general operations like Arithmetic, logic and shift operations:

Op Code	rs	rt	shamt	Func
6 bits	6 bits	6 bits	8 bits	6 bits

Since the total bit-size on which we are working is 32. So, the maximum amount of shift that can take place is 31 bits. Therefore, out of 8 bits of shamt, the higher 3 bits are redundant. The lower 5 bits will be used. Higher 3 bits are all set to zero.

★ For Immediate / load or store word:

Op Code	rs	rt	Constant or address
6 bits	6 bits	6 bits	14 bits

Since, the size of the address or constant is 14 bits. So, the allowed range of value of immediate is:

-2^{14} to $2^{14} - 1$

, i.e., -8192 to 8191

★ For Branching Operations:

Op Code	rs	address	Func
6 bits	6 bits	14 bits	6 bits

OPCODES and Func-Codes Used for different instructions:

The required codes are given in following table (Table 1):

Class	Instruction	Op Code	Func Code
Arithmetic	Add Comp Add Immediate Complement Immediate	000001 000001 111100 111101	000000 000001 - -
Logic	AND XOR	000010 000010	000000 000001
Shift	Shift Left Logical Shift Right Logical Shift Left Logical Variable Shift Right Logical Variable Shift Right Arithmetic Shift Right Arithmetic Variable	000011 000011 000011 000011 000011 000011	000000 000001 000010 000011 000100 000101
Memory	Load Word Store Word	111110 111111	- -
Branch	Unconditional Branch Branch register Branch on less than 0 Branch on flag zero Branch on flag not zero Branch and link Branch on Carry Branch on No Carry	000101 000100 000100 000100 000100 000110 000101 000101	000000 000000 000001 000010 000011 000000 000001 000010
Complex	Diff	000111	000000

Table 1: OpCodes and Func Codes used for different instructions.

DataPath Designed by us:

The datapath designed by us for the required mini-RISC processor is shown in the following figure (Fig.1).
The control lines and the data lines are represented by the dotted and solid lines respectively.

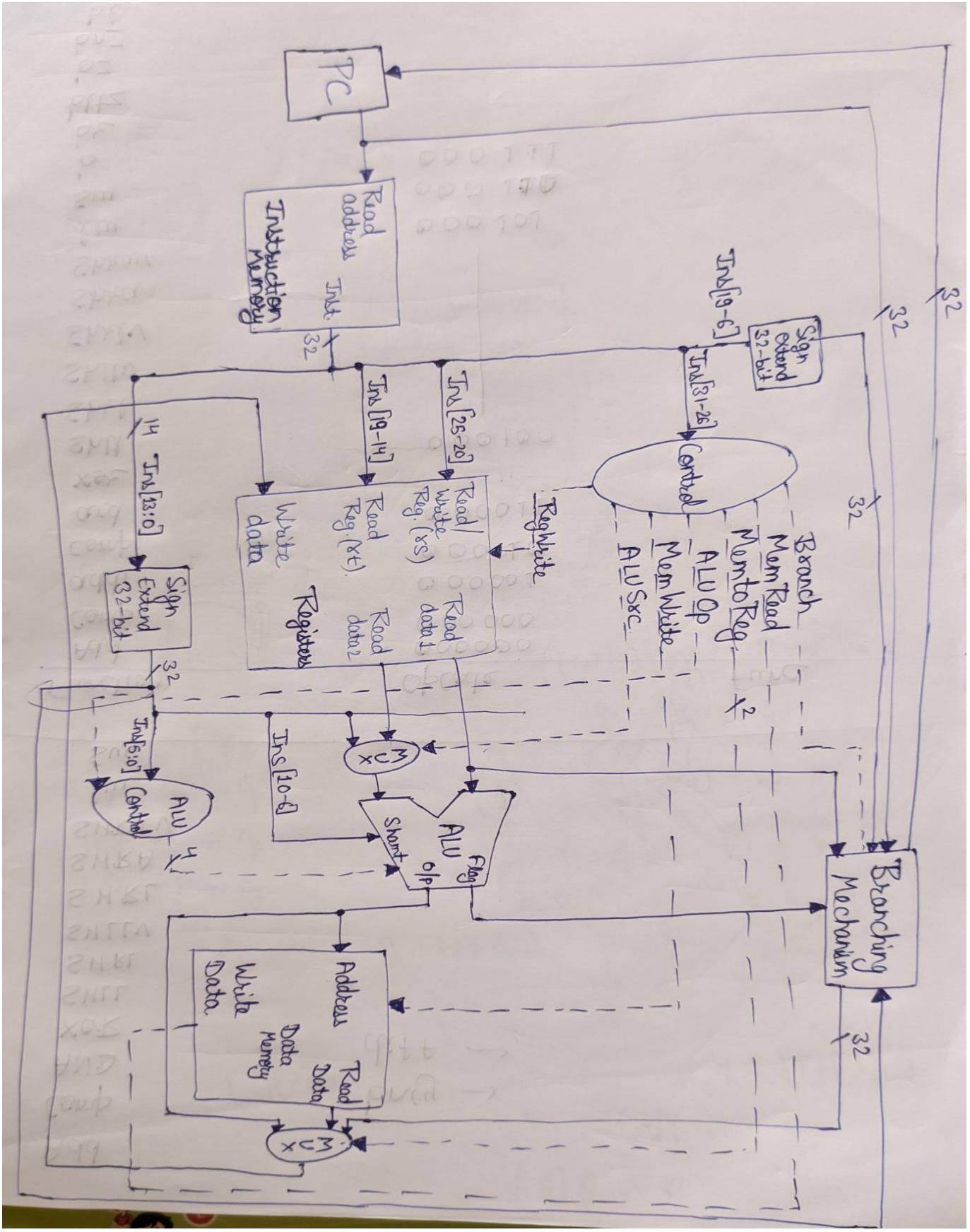

Fig. 1: DATAPATH for the given ISA

TRUTH TABLE FOR CONTROLLER SIGNALS:

The truth table for the controller signals as a function of the opcode and the function code in the instruction format is shown in Table 2:

Instr	OpCode	Func Code	Branch	MemRead	MemtoReg	ALUOp	MemWrite	ALUSrc	RegWrite
add	000001	000000	00	0	00	001	0	1	10
comp	000001	000001	00	0	00	001	0	1	10
addi	111100	-	00	0	00	101	0	0	10
compi	111101	-	00	0	00	110	0	0	10
and	000010	000000	00	0	00	010	0	1	10
xor	000010	000001	00	0	00	010	0	1	10
shll	000011	000000	00	0	00	011	0	1	10
shrl	000011	000001	00	0	00	011	0	1	10
shllv	000011	000010	00	0	00	011	0	1	10
shrlv	000011	000011	00	0	00	011	0	1	10
shra	000011	000100	00	0	00	011	0	1	10
shrav	000011	000101	00	0	00	011	0	1	10
lw	111110	-	00	1	01	101	0	0	11
sw	111111	-	00	1	00	101	1	0	00
b	000101	000000	10	0	00	000	0	1	00
br	000100	000000	01	0	00	000	0	1	00
bltz	000100	000001	01	0	00	000	0	1	00
bz	000100	000010	01	0	00	000	0	1	00
bnz	000100	000011	01	0	00	000	0	1	00
bl	000110	000000	11	0	10	000	0	1	01
bcy	000101	000001	10	0	00	000	0	1	00
bncy	000101	000010	10	0	00	000	0	1	00
diff	000111	000000	00	0	00	010	0	1	10

Table 2: Truth Table for Control Signals

*ALUOp along with FuncCode allows ALU to decide which operation to perform
*ALUSrc - Which operand to choose for ALU (immediate or from register)
*WriteReg - Higher bit of WriteReg tells whether to write or not, and the lower bit of WriteReg indicates whether to write to (rs) or (rt). And if WriteReg == 01 , then it is a special case which indicates to write to reg 31.
*MemtoReg - What to write to the register, the ALU result, the memory read, or the value of program counter + 4.
*Branch - Whether branch has been enabled or not. If 00 it has not been enabled. Else 01 corresponds to branches using opcode 4, 10 corresponds to branches using opcode 5 and 11 corresponds to branches using opcode 6.

ALU CONTROL:

The control is decided on the basis of following table entries:

ALUOp	Func_code	ALU_control_signal
000	xxxxxx	0111
001	000000	0000
001	000001	0001
010	000000	0010
010	000001	0011
011	000000	0100
011	000001	0101
011	000010	1100
011	000011	1101
011	000100	0110
011	000101	1110
101	xxxxxx	0000
110	xxxxxx	0001

Table 3: ALU control

And the architecture for ALU is shown in figure 2.
The design can be understood as follows:
The control signal is of 4 bits, out of which, MSB will be used to decide whether to choose input2 or shamt and the rest of the 3 bits to decide the operation as follows:

If (controlSignal[2:0])
- == 000, then add
- == 001, then compliment
- == 010, then and
- == 011, then xor
- == 100, then sll
- == 101, then srl
- == 110, then sra
- == 111, then diff

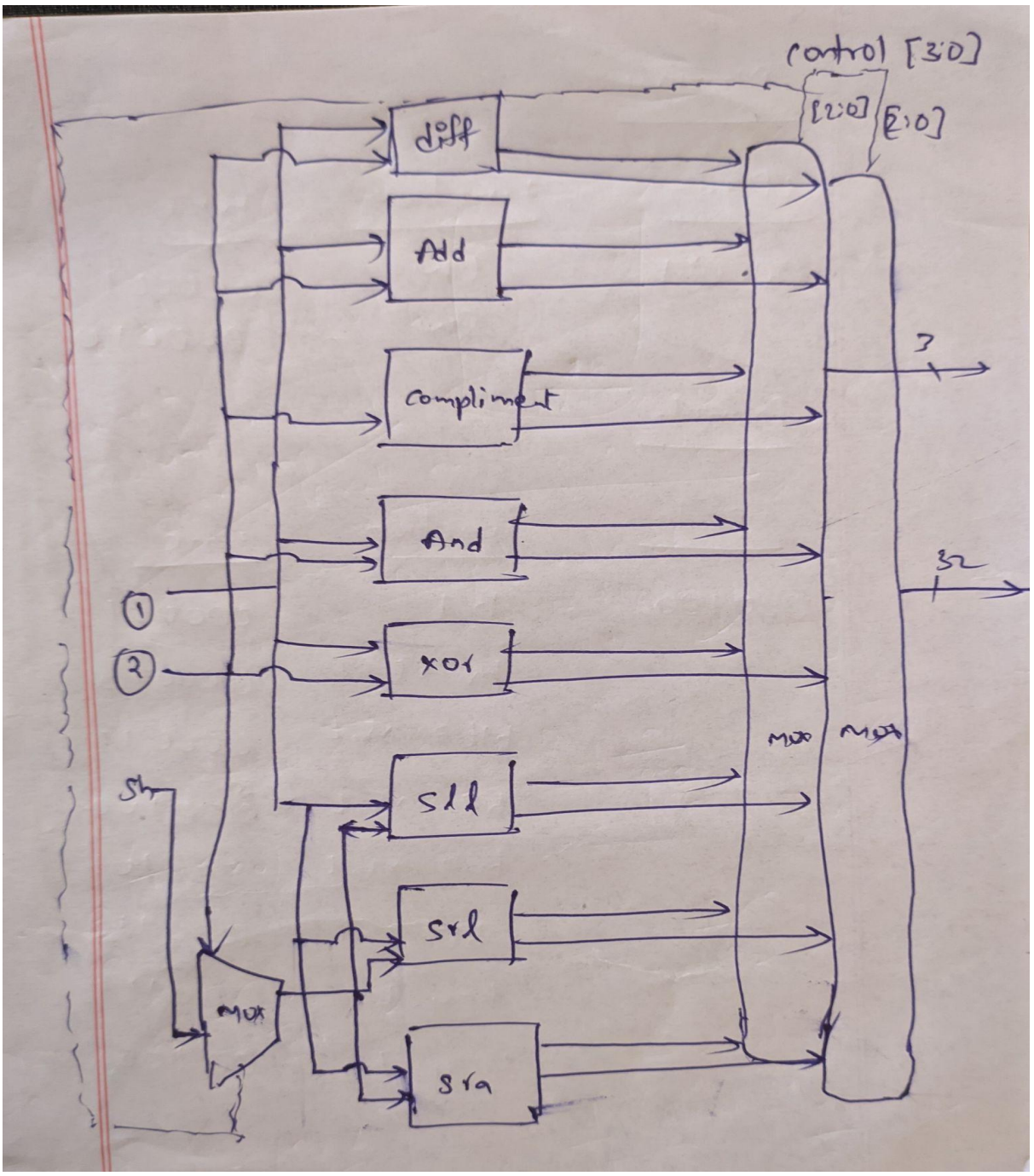

Fig.2: Architecture designed for ALU

Branching Mechanism:

operation	Func_code	Ctrl_signal	Alu_flag	PC_out
b	000000	10	xxx	L
br	000000	01	xxx	reg_rs
bltz	000001	01	x1x	L
bz	000010	01	1xx	L
bnz	000011	01	0xx	L
bl	000000	11	xxx	L
bcy	000001	10	xx1	L
bncy	000010	10	xx0	L
No branching	xxxxxx	00	xxx	(PC)+4

**********THE END**********
