Computer Organization and Architecture
(CS39001)

ASSIGNMENT - 6

KGP-miniRISC processor

Group - 56

Utsav Mehta (20CS10069)
Vibhu (20CS10072)

A. INSTRUCTION FORMAT AND ENCODING -

Instruction format (opcode and func-code for various instructions) -

- Each instruction holds a 32-bit address in memory/data.
- The encoding (opcode and func-code) of each instruction is mentioned below in its instruction format-

1. R-Format Instructions -

opcode	rs	rt	shamt	insignificant	func
6-bits	5-bits	5-bits	5-bits	6-bits	5-bits

Instruction	opcode	func
add	000000	00000
comp		00001
and	000001	00000
xor		00001
shll	000010	00000
shrl		00001
shllv		00010
shrlv		00011
shra		00100
shrav		00101
diff	001111	00000

2. I-Format Instructions -

opcode	rs	insignificant	imm (immediate)
6-bits	5-bits	5-bits	16-bits

Instruction	opcode	func
addi	000011	-
compi	000100	-

3. Memory Access Instructions -

opcode	rs	insignificant	imm (immediate)
6-bits	5-bits	5-bits	16-bits

Instruction	opcode	func
lw	000101	-
sw	000110	-

4. J-Format Instructions (Conditional Jump to Label depending on Register Value) -

opcode	rs	insignificant	L (label address)
6-bits	5-bits	5-bits	16-bits

Instruction	opcode	func
bltz	000111	-
bz	001000	-
bnz	001001	-

5. J-Format Instructions (Unconditional Jump to Address provided by Register Value) -

opcode	rs	insignificant
6-bits	5-bits	16 bits

Instruction	opcode	func
br	001010	-

6. J-Format Instructions (Jump to label based on a condition or unconditionally) -

opcode	L (label address)
6-bits	26-bits

Instruction	opcode	func
b	001011	-
bl	001100	-
bcy	001101	-
bncy	001110	-

B. LOOK UP TABLE FOR CONTROL SIGNALS -

Instr	opcode	func	Reg Dst	Reg Write	Mem Read	Mem Write	MemTo Reg	ALU src	ALU op	ALU sel	Branch	Jump Addr	Lbl Sel
add	000000	00000	00	1	0	0	10	0	00001	0	0	-	
comp	000000	00001	00	1	0	0	10	0	00101	1	0	-	
and	000001	00000	00	1	0	0	10	0	00010	0	0	-	-
xor	000001	00001	00	1	0	0	10	0	00011	0	0	-	-
shll	000010	00000	00	1	0	0	10	1	01010	0	0	-	-
shrl	000010	00001	00	1	0	0	10	1	01000	0	0	-	-
shllv	000010	00010	00	1	0	0	10	0	01010	0	0	-	-
shrlv	000010	00011	00	1	0	0	10	0	01000	0	0	-	-
shra	000010	00100	0	1	0	0	10	1	01001	0	0	-	-
shrav	000010	00101	00	1	0	0	10	0	01001	0	0	-	-
addi	000011	-	00	1	0	0	10	1	00001	0	0	-	-
compi	000100	-	00	1	0	0	10	1	00101	1	0	-	-
lw	000101	-	01	1	1	0	01	1	10101	0	0	-	-
sw	000110	-	-	0	0	1	-	1	10101	0	0	-	-
bltz	000111	-	-	0	0	0	-	-	00000	-	1	0	1
bz	001000	-	-	0	0	0	-	-	00000	-	1	0	1
bnz	001001	-	-	0	0	0	-	-	00000	-	1	0	1
br	001010	-	-	0	0	0	-	-	00000	-	1	1	-
b	001011	-	-	0	0	0	-	-	00000	-	1	0	0
bl	001100	-	10	1	0	0	00	-	00000	-	1	0	0
bcy	001101	-	-	0	0	0	-	-	00000	-	1	0	0
bncy	001110	-	-	0	0	0	-	-	00000	-	1	0	0
diff	001111	00000	-	1	0	0	10	0		0	0	-	-

Definition and Uses of the Control Signals used above -

- RegDst
  - Used to select the register to which data is to be written
  - 00 => Write to rs
  - 01 => Write to rt
  - 10 => Write to $ra
  - 11 => <NOT USED>

- RegWrite
  - Determines if data is to be written
  - 1 => Write data
  - 0 => Don't write

- MemRead
  - Determines if data is to be loaded from register
  - 1 => Load value
  - 0 => Don't load value

- MemWrite
  - Determines if data is to be stored to a register
  - 1 => Store value
  - 0 => Don't store value

- MemToReg
  - Determines value to be written to register
  - 00 => Program counter, used for bl instruction
  - 01 => Store data from memory, used for sw instruction
  - 10 => Store computed data, used for various outputsfrom ALU
  - 11 => <NOT USED>

- ALUsrc -
  - Determines one of the inputs to the ALU
  - 0 => Data from registers
  - 1 => Sign extended inputs

- ALUsel -
  - Determines if an input to ALU is corresponding to complement or not
  - 1 => Complement operation
  - 0 => Any other operation

- ALUop -
    - Determines the type of output provided by the ALU
    - x1xxx => Shift operation
    - x1x1x => Left shift operation
    - x1x0x => Right shift operation
    - x1xx0 => Logical shift operation
    - x1xx1 => Arithmetic shift operation
    - xx1xx => Complement operation
    - 0xxxx => R/I Format instruction
    - 1xxxx => Memory instruction

- Branch -
    - Determines if the operation is branch operation
    - 1 => Branch operation
    - 0 => Any other operation

- JumpAddr -
    - Determines type of address to be used during branch instruction
    - 1 => br instruction
    - 0 => Other branch instructions

- LblSel -
    - Determines the size of the label to be used to branch
    - 1 => Branch to 15 bit address
    - 0 => Branch to 26 bit address

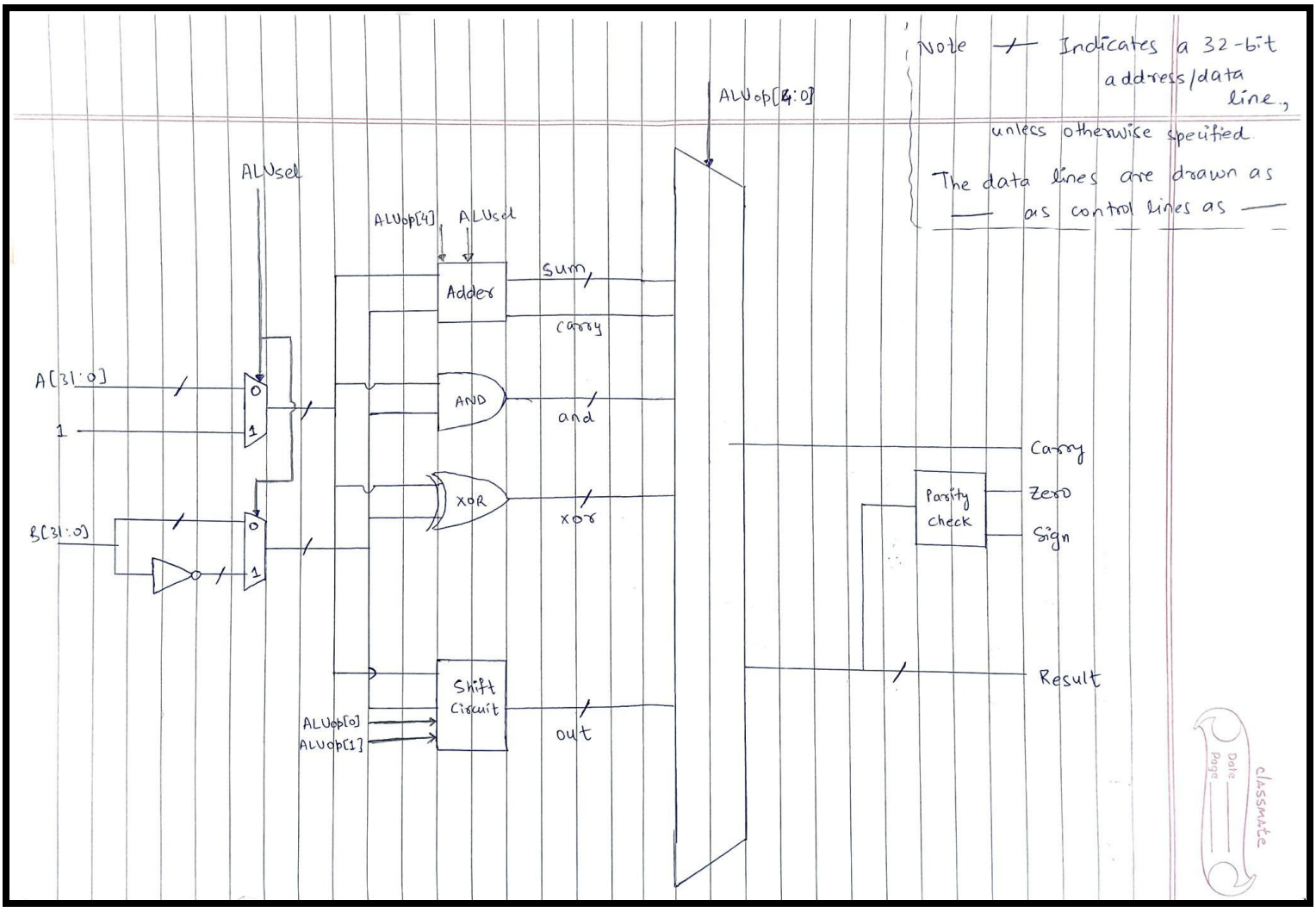
C. LOOK UP TABLE FOR JUMP CONTROL -

Instr	opcode	zero	sign	carry	validJump
bltz	000111	0	1	-	1
bz	001000	1	0	-	1
bnz	001001	0	-	-	1
br	001010	-	-	-	1
b	001011	-	-	-	1
bl	001100	-	-	-	1
bcy	001101	-	-	1	1
bncy	001110	-	-	0	1

Definition and Uses of the Control Signals used above -

- validJump = 0 => For any other instructions (or combinations of zero/ sign/ carry) not mentioned above
- validJump is always 1 for unconditional branch instructions
- validJump is 1 when sign is 1 (-ve number) and zero is 0 (number is non-zero) – for bltz
- validJump is 1 when sign is 0 and zero is 0 (number is zero) – for bz
- validJump is 1 when zero is 0 – for bnz
- validJump is 1 when carry is 1 – for bcy
- validJump is 0 when carry is 0 – for bncy

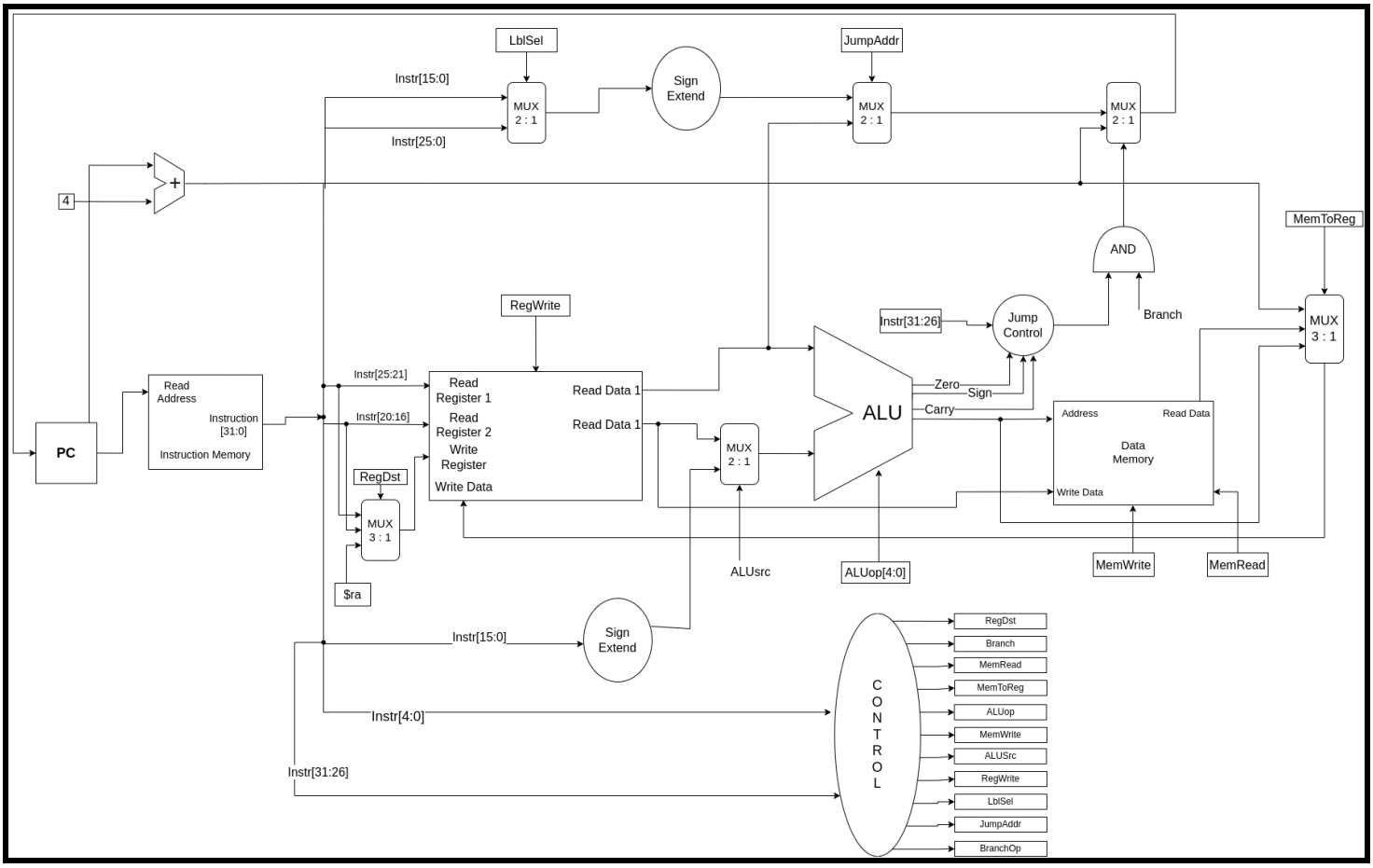
D. Arithmetic Logic Unit



The corresponding verilog files used for generating the ALU unit are -

ALU_unit.v	mux_32_to_1.v	shifter.v	adder_32_bit.v
CLA_16_bit.v	CLA_4_bit_augmented.v		look_ahead_unit.v

E. Data Path



The corresponding verilog files used for generating the DATAPATH are -

ALU_unit.v	d_flip_flop.v	PC_incremented.v	sign_extend.v
mux_32_to_1.v	mux_32x3_to_1.v	mux_4x3_to_1.v	jump_control.v
program_counter.v	instruction_decoder.v	branch_unit.v	diff.v
encoder.v	register.v	datapath.v	control_unit.v