Design and Synthesis of a 32-bit RISC Processor

CS39001: Computer Organization and Architecture Laboratory

Group 2:

Parag Mahadeo Chimankar (23CS10049),
Harshit Singhal (23CS10025)

1. Instruction Encoding

The ISA uses a fixed 32-bit instruction length, a hallmark of RISC architectures that simplifies instruction fetching and decoding. Instructions are categorized into R-Type, I-Type, J-Type, and Program Control formats.

Assumptions and Justifications

- **Fixed-Length Instructions (32-bit):** Simplifies the Program Counter (PC) increment logic (always PC + 4) and makes instruction fetching and decoding hardware simpler and faster.
- **Load-Store Architecture:** Only load (LD) and store (ST) instructions can access data memory.
- **Register File:** A bank of 16 general-purpose 32-bit registers is used. A smaller register file is simpler to implement and can have faster access times.
- **Opcode:** The most significant 6 bits (instruction[31:26]) are always the opcode. This allows the control unit to start decoding and generating control signals immediately after an instruction is fetched.

1.1. R-Type (Register-Type) Instructions

R-type instructions are used for operations performed between registers.

Format:

opcode	rs	rt	rd	Don't care	func
6 bits [31:26]	5 bits [25:21]	5 bits [20:16]	5 bits [15:11]	6 bits [10:6]	5 bits [4:0]

- **opcode (6 bits):** Primary operation code.
- **rs (5 bits):** Source register 1.
- **rt (5 bits):** Source register 2.
- **rd (5 bits):** Destination register.
- **funct (6 bits):** Function code to differentiate R-type instructions.

Justification: This format efficiently encodes two source and one destination register, ideal for three-operand instructions.

Using a **funct** field allows for many register-based operations under a single opcode.

Instruction Set for R-type:

Instruction	**Usage**	**Opcode**	**Function**
ADD	ADD rd,rs,rt	000000	00001
SUB	SUB rd,rs,rt	000000	00010
AND	AND rd,rs,rt	000000	00011
OR	OR rd,rs,rt	000000	00100
XOR	XOR rd,rs,rt	000000	00101
NOR	NOR rd,rs,rt	000000	00110
SL	SL rd,rs,rt	000000	00111
SRL	SRL rd,rs,rt	000000	01000
SRA	SRA rd,rs,rt	000000	01001
SLT	SLT rd,rs,rt	000000	01010
SGT	SGT rd,rs,rt	000000	01011
NOT	NOT rd,rt	000000	01100
INC	INC rd,rt	000000	01101
DEC	DEC rd,rt	000000	01110
HAM	HAM rd,rt	000000	01111

MOVE	MOVE rd,rs	010100	10000
CMOV	CMOV rd,rs,rt	010101	10001

1.2. I-Type (Immediate-Type) Instructions

I-type instructions are for operations involving an immediate value, memory access, and conditional branches.

Format:

opcode	rs	rt	immediate
6 bits [31:26]	5 bits [25:21]	5 bits [20:16]	16 bits [15:0]

- **rs (5 bits):** Source register (base for memory, comparison for branch).
- **rt (5 bits):** Source/Destination register.
- **immediate (16 bits):** A 16-bit constant value.

Justification: This versatile format allows constants to be used directly, provides an offset for memory access, and a target offset for branches.

Instruction Set:

Instruction	**Usage**	**Opcode**
ADDI	ADDI rt,rs,imm	000001
SUBI	SUBI rt,rs,imm	000010
ANDI	ANDI rt,rs,imm	000011
ORI	ORI rt,rs,imm	000100
XORI	XORI rt,rs,imm	000101
NORI	NORI rt,rs,imm	000110
SLI	SLI rt,rs,imm	000111
SRLI	SRLI rt,rs,imm	001000

SRAI	SRAI rt,rs,imm	001001
SLTI	SLTI rt,rs,imm	001010
SGTI	SGTI rt,rs,imm	001011
NOTI	NOTI rt,imm	001100
INCI	INCI rt,imm	001101
DECI	DECI rt,imm	001110
HAMI	HAMI rt,imm	001111
LUI	LUI rt,imm	010000
LD	LD rt,imm(rs)	010001
ST	ST rt,imm(rs)	010010
BMI	BMI rs,imm	100001
BPL	BPL rs,imm	100010
BZ	BZ rs,imm	100011

1.3. J-Type (Jump-Type) Instructions

J-type instructions are used for unconditional jumps.

Format:

opcode	**immediate**
6 bits [31:26]	26 bits [25:0]

- **immediate (26 bits):** A 26-bit address fragment.

Justification: This format maximizes the address range for jumps.

Instruction Set:

Instruction	**Usage**	**Opcode**
BR	BR imm	100000

1.4. Program Control Instructions

These instructions manage the flow and state of the processor.

Format:

opcode	**Don't Care**
6 bits [31:26]	26 bits [25:0]

Instruction Set:

Instruction	**Opcode**	**Description**
HALT	100100	Pause instruction reading until INT signal
NOP	100101	No Operation
CALL	100110	Write current PC Address into $ret

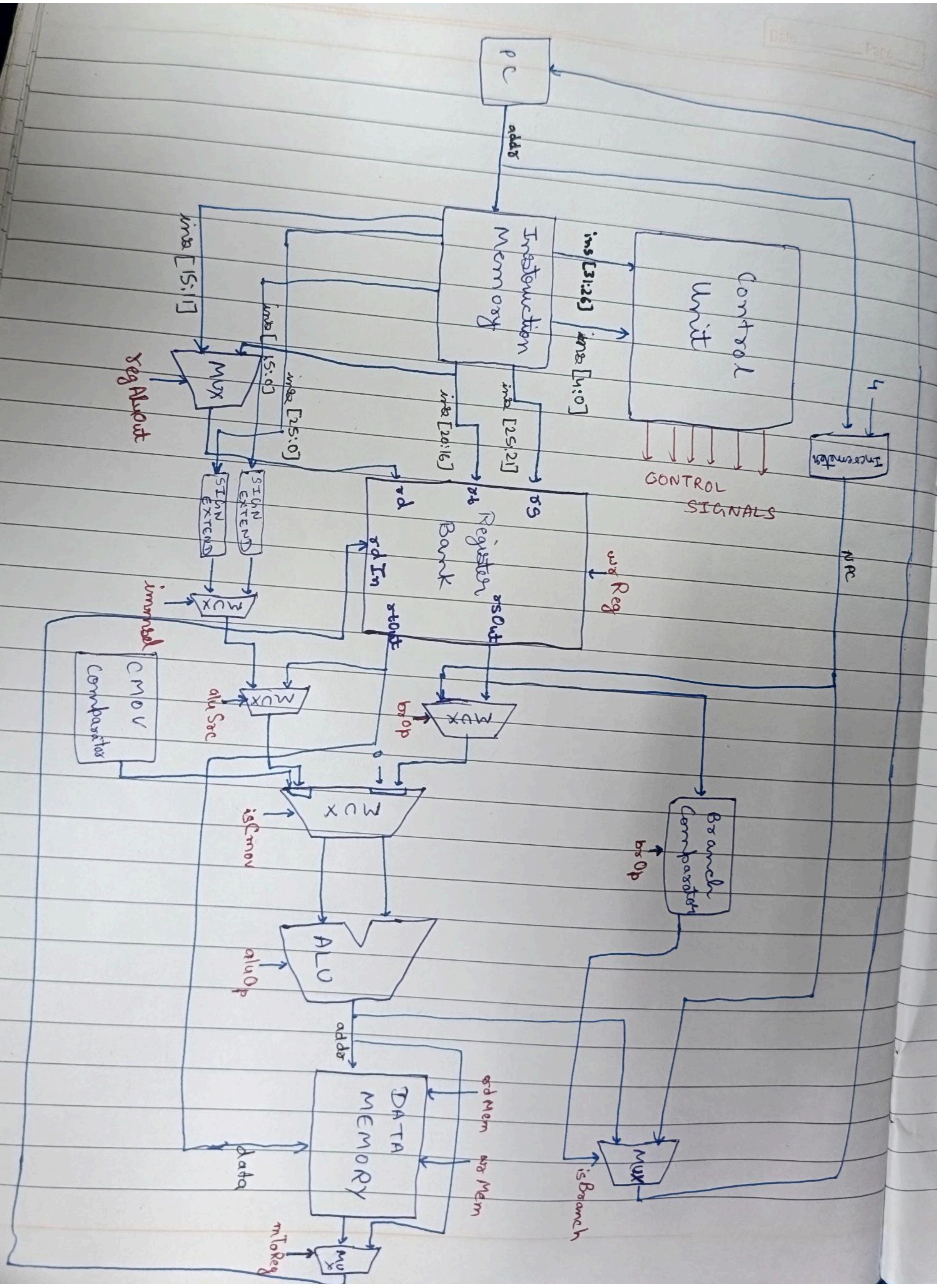
2. Register Usage Convention

Register	**Function**	**Register Number**
$R0 Zero Register $R1-$R15 $ret $pc	$R0 Zero Register General Purpose Registers Return Address $pc Program Counter	00000 00001-01111 10000

3. Datapath Schematic

Datapath Components and Flow

1. **Instruction Fetch:**
   - The **Program Counter (PC)** holds the address of the current instruction and sends it to **Instruction Memory**.
   - An **Incrementer** computes PC + 4 to point to the next instruction.
2. **Instruction Decode & Register Read:**
   - The instruction's rs and rt fields are sent to the **Register Bank** to read rsOut and rtOut.
3. **Execution:**
   - The **ALU** receives rsOut as its first operand.
   - The aluSrc MUX selects the second operand: rtOut for R-type or the immediate value for I-type.
   - aluOp tells the ALU which operation to perform.
4. **Memory Access:**
   - For LD and ST, the ALU result is used as an address for **Data Memory**.
   - rdMem or wrMem are asserted to read from or write to memory.
5. **Write Back:**
   - The mToReg MUX selects which result to write back to the register file: the ALU result or data from memory.
   - wrReg must be asserted to enable the write.
6. **PC Update:**
   - The PC is updated to PC + 4, or a branch/jump target address if required.


PC
addr
Instruction Memory
ins[31:26]
ins[4:0]
Control Unit
4
Incrementer
CONTROL SIGNALS
ins[25:21]
ins[20:16]
ins[15:11]
ins[15:0]
ins[25:0]
MUX
regAluOut
rs
rt
rd
Register Bank
rdIn
rsOut
rtOut
wrReg
SIGN EXTEND
SIGN EXTEND
immSel
CMOV Comparator
aluSrc
brOp
nPC
Branch Comparator
isCmov
ALU
aluOp
addr
DATA MEMORY
rdMem
wrMem
data
isBranch
mToReg

4. Hardwired Control Unit Design

The hardwired control unit is a combinational logic circuit that generates control signals from the instruction's opcode and function code.

4.1. aluOp (ALU Operation)

Opcode	aluOp	Meaning
000000	ins[3:0]-1	Infer from funct: ADD...HAM
000001-001111	ins[29:26]-1	Infer from opcode: ADDI...HAMI
010000	1111	LUI
010001, 010010	0000	ADD (for load/store address calc)
010100, 010101	0000	ADD (for move)
100000-100011	0000	ADD (for branch)
other	XXXX	Don't Care

4.2. brOp (Branch Operation)

Opcode	brOp	Meaning
100000	000	BR (Unconditional Branch)
100001	001	BMI (Branch on Minus)
100010	010	BPL (Branch on Plus)
100011	011	BZ (Branch on Zero)
other	100	Not a branching instruction

4.3. aluSrc (ALU Source 2 Select)

Opcode	**aluSrc**	**Meaning (Selects second ALU operand)**
000000	1	rt register
other	0	immediate value

4.4. regAluOut (Destination Register Select)

Opcode	**regAluOut**	**Meaning (Selects rd or rt as destination)**
000000, 010100-010101	1	rd
000001-010001	0	rt
other	X	Don't Care

4.5. rdMem (Memory Read)

Opcode	**rdMem**	**Meaning**
010001	1	READ from #imm(R[rs]) (LD)
other	0	Don't Read

4.6. wrMem (Memory Write)

Opcode	**wrMem**	**Meaning**
010010	1	WRITE to #imm(R[rs]) (ST)
other	0	Don't Write

4.7. wrReg (Register Write)

Opcode	**wrReg**	**Meaning**

000000-010001, 010100-010101	1	Write to destination register
other	0	Don't Write

4.8. mToReg (Memory to Register MUX)

Opcode	**mToReg**	**Meaning (Source for register write)**
000000-010000, 010100-010101	0	Value from ALUOut
010001	1	Value from Load Memory Data (LMD)
other	X	Don't Care

4.9. immSel (Immediate Selection)

Opcode	**immSel**	**Meaning (Sign extension mode)**
000001-010010, 100001-100011	0	(ins[15])^16 ## ins[15:0] (Sign Extend I-Type)
100000	1	ins[25:0] ## 00 (J-Type)
other	X	Don't Care

4.10. isCmov (Conditional Move)

Opcode	**isCmov**	**Meaning**
010101	1	CMOV
other	0	not CMOV

Key Control Signals Summary

- **wrReg**: Asserted for instructions that write to a register.
- **aluSrc**: Selects the second ALU operand (rtOut or immediate).
- **mToReg**: Selects the data source for register write-back (ALU or memory).
- **rdMem / wrMem**: Enables for memory read/write.
- **brOp**: Specifies the condition for the branch comparator.
- **aluOp**: Controls the ALU's operation.
- **isCmov**: Enables the conditional move functionality.

This design provides a solid foundation for a functional 32-bit RISC processor, with clear separation between the datapath that moves and transforms data, and the control unit that directs its operation.