Design and Synthesis of a 32-bit RISC Processor

CS39001: Computer Organization and Architecture Laboratory

Group 74: Battula Hari Lakshman Prasad (23CS10009), Sayon Sujit Mondal (23CS30063)

10th September 2025

Contents

1 Instruction Format and Encoding

1.1 R-Type Instructions

opcode	**rs**	**rt**	**rd**	**unused**	**funct**
6 bits	5 bits	5 bits	5 bits	7 bits	4 bits

Table 1: R-type Instruction Format

Instruction	**Opcode**	**Funct**	**Description**
ADD	000000	0000	Add two registers: rd=rs+rt
SUB	000000	0001	Subtract two registers: rd=rs-rt
INC	000000	0010	Increment a register: rd=rs+1
DEC	000000	0011	Decrement a register: rd=rs-1
SLT	000000	0100	Set on Less Than: rd=(rs<rt)?1:0
SGT	000000	0101	Set on Greater Than: rd=(rs>rt)?1:0
HAM	000000	0110	Hamming Distance: rd=hamming(rs)
OR	000000	1000	Bitwise OR: rd=rs$\lor$rt
XOR	000000	1001	Bitwise XOR: rd=rs$\oplus$rt
NOR	000000	1010	Bitwise NOR: rd=$\neg$(rs$\lor$rt)
AND	000000	1011	Bitwise AND: rd=rs$\land$rt
NOT	000000	1100	Bitwise NOT: rd=$\neg$rs
SL	000000	1101	Shift Left Logical: rd=rs$\ll$rt
SRL	000000	1110	Shift Right Logical: rd=rs$\gg$rt
SRA	000000	1111	Shift Right Arithmetic: rd=rs$\ggg$rt
MOVE	110000	0000	Move register: rd=rs
CMOV	110001	0001	rd = (rs < rt) ? rs : rt

Table 2: Opcodes and function codes for R-type instructions

1.2 I-Type Instructions

opcode	**rs**	**rt**	**immediate**
6 bits	5 bits	5 bits	16 bits

Table 3: I-type Instruction Format

Instruction	Opcode	Description
ADDI	010000	Add immediate: rt=rs+immediate
SUBI	010001	Subtract immediate: rt=rs-immediate
INCI	010010	Increment immediate: rt=rs+1
DECI	010011	Decrement immediate: rt=rs-1
SLTI	010100	Set on Less Than Immediate: rt=(rs<immediate)?1:0
SGTI	010101	Set on Greater Than Immediate: rt=(rs>immediate)?1:0
HAMI	010110	Hamming Immediate: rt=hamming(rs,immediate)
LUI	010111	Load Upper Immediate: rt=immediate$\ll$16
ORI	011000	Bitwise OR immediate: rt=rs$\vee$immediate
XORI	011001	Bitwise XOR immediate: rt=rs$\oplus$immediate
NORI	011010	Bitwise NOR immediate: rt=$\neg$(rs$\vee$immediate)
ANDI	011011	Bitwise AND immediate: rt=rs$\wedge$immediate
NOTI	011100	Bitwise NOT Immediate: rt=$\neg$immediate
SLI	011101	Shift Left Immediate: rt=rs$\ll$immediate
SRLI	011110	Shift Right Immediate: rt=rs$\gg$immediate
SRAI	011111	Shift Right Arithmetic Immediate: rt=rs$\ggg$immediate
BZ	100000	Branch on Zero: if (rs=0) then PC=PC+4+immediate$\times$4
BMI	100001	Branch on Minus: if (rs<0) then PC=PC+4+immediate$\times$4
BPL	100010	Branch on Plus: if (rs$\geq$0) then PC=PC+4+immediate$\times$4
LD	101000	Load a word from memory: rt=Memory[rs+immediate]
ST	101001	Store a word to memory: Memory[rs+immediate]=rt

Table 4: Opcodes for I-type instructions

1.3 J-Type Instructions

opcode	target address
6 bits	26 bits

Table 5: J-type Instruction Format

Instruction	Opcode	Description
BR	100011	Unconditional Branch: PC=target address

Table 6: Opcodes for J-type instructions

1.4 Program Control Instructions

opcode	Don't Care
6 bits	26 bits

Table 7: Program Control Instruction Format

Instruction	Opcode	Description
HALT	111000	Halt the processor
NOP	111001	No Operation

Table 8: Opcodes for program control instructions

2 Register Usage Convention

Register	**Function**	**Register Number**
\$R0	Zero Register	00000
\$R1-\$R15	General Purpose Registers	00001-01111
\$sp	Stack Pointer	10000

Table 9: 17 32-bit GPRs in Register File

Register	**Function**
\$pc	Program Counter

Table 10: SPRs

3 Datapath

The datapath consists of the following components:

- 32-bit general-purpose registers.
- ALU for arithmetic and logic operations.
- Program Counter (PC) and incrementer for the same
- Control Unit: A hardwired control unit that generates control signals.
- Memory: Byte-addressable memory, supporting 32-bit aligned access.

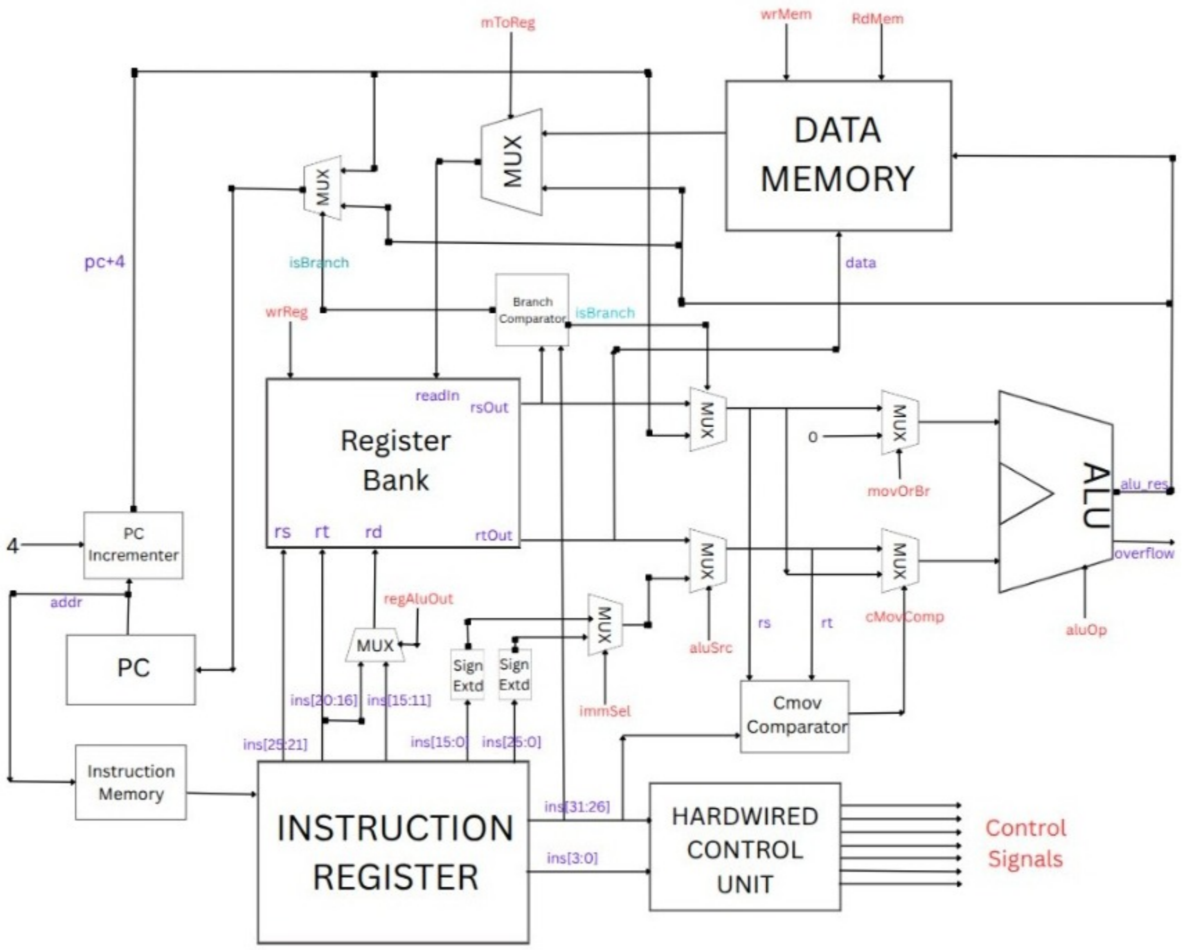

Figure 1: Processor Datapath

4 Control Unit

4.1 aluOp

Opcode	aluOp	Meaning
000000	$ins_{3:0}$	Infer from funct: ADD,...,SRA
010000-011111	$ins_{29:26}$	Infer from opcode: ADDI,...,SRAI
100000-100010	0000	Add (for branch comparison)
101000-101001	0000	ADD (for load, store)
110000	0000	ADD (for MOVE)
110001	0000	ADD (for CMOV)
100011,111000-111001	xxxx	Don't Care (BR, HALT, NOP)

Table 11: Specification of ALU operations

4.2 isBranch

Opcode	isBranch	Meaning
100011	1	BR
100001	1	BMI
100010	1	BPL
100000	1	BZ
other	0	Not a branching instruction

Table 12: Specification of Branch operations

4.3 aluSrc

Opcode	aluSrc	Meaning
000000,110000-110001	1	rt
010000-011111,100000-100010,101000-101001	0	immediate
other	x	Don't Care

Table 13: Selection of 2nd source of ALU input

4.4 regAluOut

Opcode	regAluOut	Meaning
000000,110000-110001	1	rd
010000-010111,101000	0	rt
other	x	Don't Care

Table 14: Selection of Destination Register

4.5 rdMem

Opcode	rdMem	Meaning
101000	1	READ from Memory[rs+immediate]
other	0	Don't Read

Table 15: Read from Memory

4.6 wrMem

Opcode	wrMem	Meaning
101001	1	WRITE to Memory[rs+immediate]
other	0	Don't Write

Table 16: Write to Memory

4.7 wrReg

Opcode	wrReg	Meaning
000000,010000-010111,101000,110000-110001	1	Write to dest reg
other	0	Don't Write

Table 17: Write to Register

4.8 mToReg

Opcode	mToReg	Meaning
101000	1	Value from Memory
000000,010000-010111,110000-110001	0	Value from ALUOut
other	x	Don't Care

Table 18: Source of value to be written to dest reg

4.9 immSel

Opcode	immSel	Meaning
010000-011111,100000-100010,101000-101001	0	Sign-extend 16-bit immediate
100011	1	26-bit target address
other	x	Don't Care

Table 19: Selection of immediate value

4.10 movOrBr

Opcode	movOrBr	Meaning
110001	1	CMOV
110000	1	MOVE
100011	1	BRANCH
other	0	Don't Care

Table 20: Specification of Move/Cmov/Br operation

4.11 iscMov

Opcode	isCmov	Meaning
110001	1	CMOV
other	0	NotCMov

Table 21: Specification if opcode is of CMov

4.12 cMovComp

Opcode	CMovComp	Meaning
110001 and isCMov	1	(rs greater than rt) and isCMov
other	0	not cMovComp

Table 22: Result of (CMov Comparator and isCMov)

5 Justification of Design Choices

In designing the instruction set architecture (ISA) and encoding scheme, the following considerations guided our decisions:

5.1 Instruction Formats

The ISA adopts the standard R-type, I-type, and J-type instruction formats, alongside a dedicated format for program control instructions. These formats provide a balance between simplicity and flexibility, enabling efficient instruction decoding and execution. The fixed-width 32-bit instruction encoding ensures straightforward decoding by the control unit. The separation of fields (opcode, register specifiers, immediate, and function codes) facilitates efficient hardware implementation, as each field can be processed independently by the datapath components.

5.2 Opcode Allocation

Opcodes were allocated strategically to optimize instruction decoding and hardware efficiency:

- Frequently used instructions, such as `ADD`, `SUB`, `AND`, and `OR`, share the R-type format with a common opcode (`000000`) and distinct 4-bit function codes. This minimizes opcode usage and simplifies the ALU design by leveraging the function field to differentiate operations.
- Immediate instructions (e.g., `ADDI`, `SUBI`, `LUI`) are assigned unique opcodes in the range `010000` to `011111`, allowing direct support for constants without complicating the decoding logic.
- Branch and jump instructions (`BZ`, `BMI`, `BPL`, `BR`) are assigned opcodes starting with `10` (e.g., `100000`, `100011`) to make them easily distinguishable by the control unit, streamlining branch detection.
- Program control instructions (`HALT`, `NOP`) use high-order opcodes (`111000`, `111001`) to clearly separate them from computational instructions.

5.3 Register Usage Convention

The ISA includes 17 32-bit general-purpose registers: `$R0` (hardwired to zero), `$R1` to `$R15` (general-purpose), and `$sp` (stack pointer). This configuration balances hardware cost with programmer flexibility. The zero register (`$R0`) simplifies operations such as move-immediate (using `ADD` with `$R0`) and comparisons (e.g., `SLT`). The dedicated stack pointer (`$sp`) supports stack-based operations, enhancing the ISA's suitability for procedure calls and memory management.

5.4 Control Signals

The control unit generates signals (`aluOp`, `brOp`, `aluSrc`, `wrMem`, `rdMem`, `wrReg`, `mToReg`, `immSel`, `isCmov`) that directly map instruction fields to datapath actions. For example:

- `aluOp` uses the function field (`ins[3:0]`) for R-type instructions and opcode bits (`ins[29:26]`) for I-type instructions, ensuring precise ALU operation selection.
- `isCmov` uniquely identifies the `CMOV` instruction (`110001`), enabling conditional move operations without additional opcodes.
- `CMovComp` activates when rs is required (rs greater than rt) and when isCMov is active
- `movOrBr` only active for Move, CMov, Branch operation as we need to select 0 for alu operation because the result should be passed as is across ALU.
- `immSel` distinguishes between 16-bit sign-extended immediates for I-type instructions and 26-bit target addresses for J-type instructions, optimizing address calculations.

This design supports a hardwired control implementation, reducing latency compared to microcoded control.

5.5 Memory Model

The ISA employs byte-addressable memory with 32-bit word-aligned access. This choice aligns with standard RISC architectures, simplifying load/store circuitry by restricting accesses to aligned 32-bit words. Instructions like `LD` and `ST` use the `rs+immediate` addressing mode, which is efficient for both data access and hardware implementation.

5.6 Overall Justification

The design adheres to RISC principles: fixed-length 32-bit instructions, simple addressing modes, a load/store architecture, and a compact set of orthogonal instructions. Instructions like `MOVE` and `CMOV` enhance flexibility, while `HAM` and `HAMI` provide specialized functionality for applications requiring Hamming distance calculations. The ISA supports efficient pipelining and FPGA implementation by minimizing decoding complexity and ensuring regular instruction formats. This balance of simplicity and expressiveness makes the processor suitable for a wide range of computational tasks while maintaining hardware efficiency.