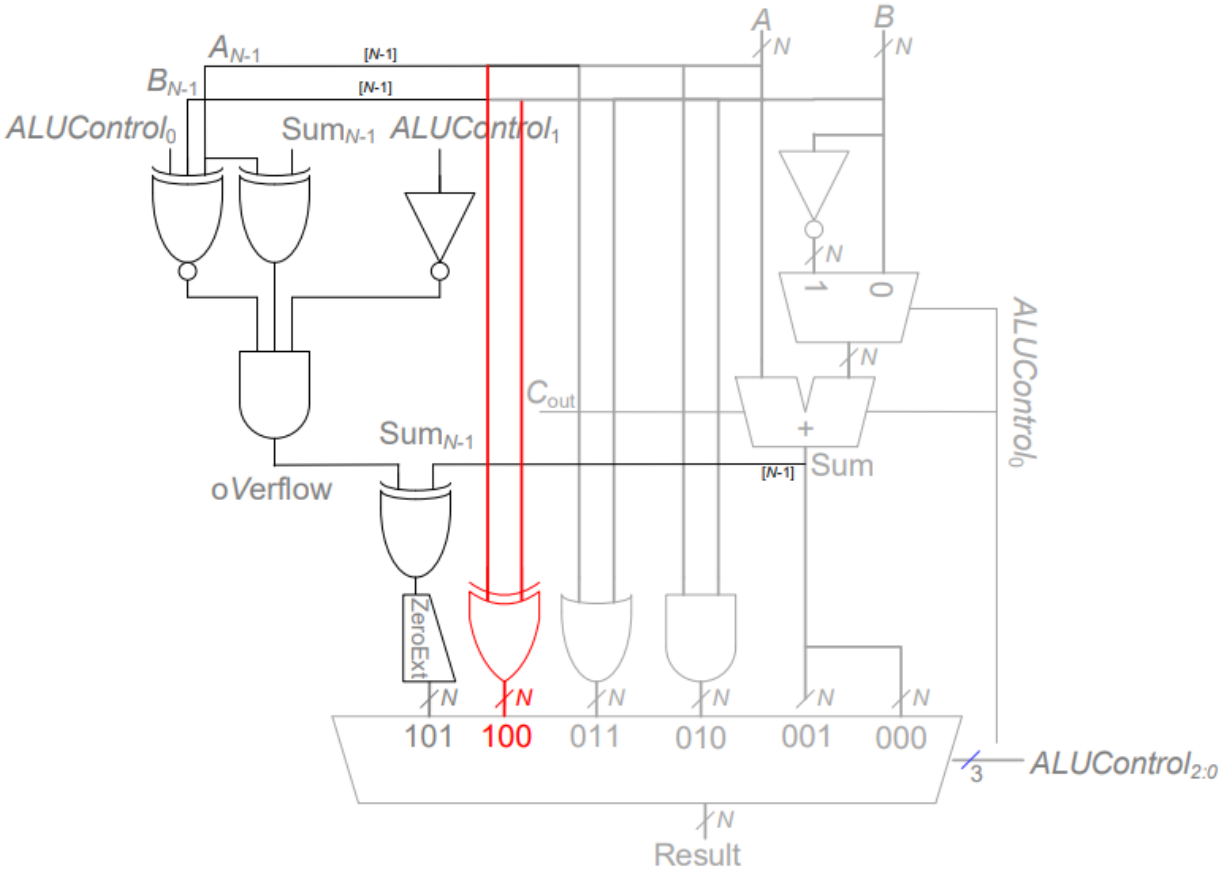
Exercise 7.3

(a) `xor`

The datapath does not require any changes to its interfaces. Only the ALU needs to be modified: we add another input to the multiplexer and N 2-bit XOR gates within the ALU. We also update the ALU Decoder truth table / logic. The Main Decoder truth table need not be updated because it already supports R-type instructions. These changes are shown below.

Modified ALU operations to support `xor`

$ALUControl_{2:0}$	Function
000	add
001	subtract
010	and
011	or
100	**xor**
101	SLT



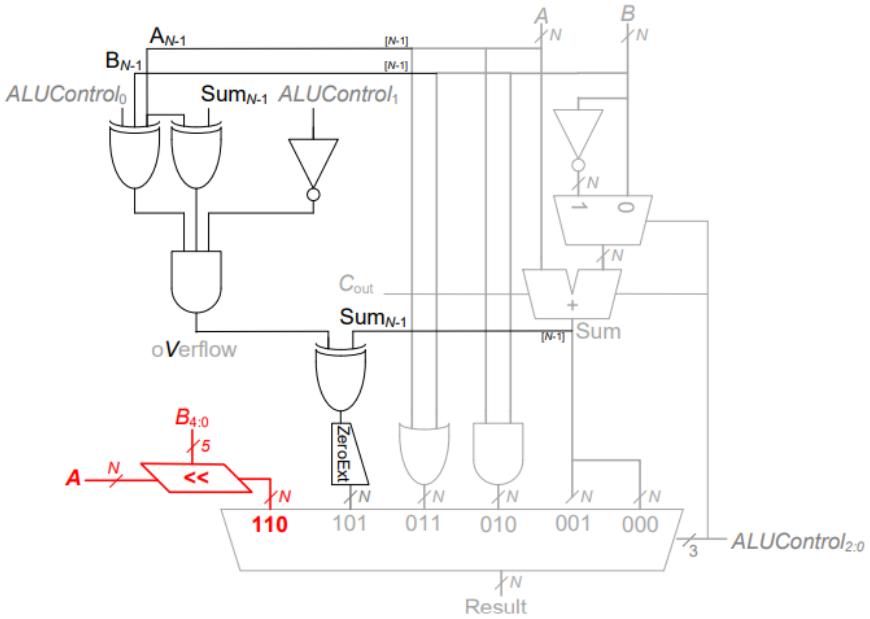
Modified ALU Decoder truth table to support `xor`

ALUOp	funct3	op_5, $funct7_5$	*ALUControl*	Instruction
00	x	x	000 (add)	`lw, sw`
01	x	x	001 (subtract)	`beq`
10	000	00, 01, 10	000 (add)	`add, addi`
	000	11	001 (subtract)	`sub`
	010	x	101 (set less than)	`slt, slti`
	100	**x**	**100 (xor)**	**`xor, xori`**
	110	x	011 (or)	`or, ori`
	111	x	010 (and)	`and, andi`

(b) `sll`

The overall datapath (interfaces and units) need not be changed. We only modify the ALU and the ALU Decoder, as shown below. We add a shifter and expand the multiplexer inside the ALU.

Modified ALU to support `sll`



Modified ALU operations to support `sll`

$ALUControl_{2:0}$	Function
000	add
001	subtract
010	and
011	or
101	SLT
110	**sll**

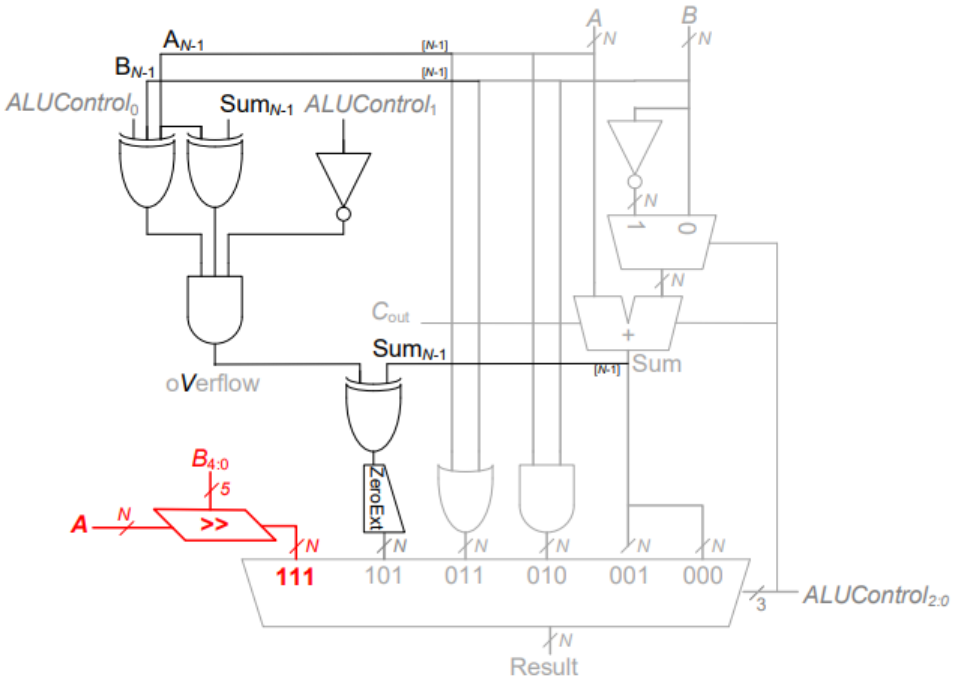
Modified ALU Decoder truth table to support `sll`

ALUOp	funct3	op_5, $funct7_5$	*ALUControl*	Instruction
00	x	x	000 (add)	`lw, sw`
01	x	x	001 (subtract)	`beq`
10	000	00, 01, 10	000 (add)	`add, addi`
	000	11	001 (subtract)	`sub`
	001	**x**	**110 (shift left logical)**	**`sll, slli`**
	010	x	101 (set less than)	`slt, slti`
	110	x	011 (or)	`or, ori`
	111	x	010 (and)	`and, andi`

(c) `srl`

The overall datapath (interfaces and units) need not be changed. We only modify the ALU and the ALU Decoder, as shown below. We add a shifter and expand the multiplexer inside the ALU.

Modified ALU to support `srl`



Modified ALU operations to support `srl`

$ALUControl_{2:0}$	Function
000	add
001	subtract
010	and
011	or
101	SLT
111	**srl**

Modified ALU Decoder truth table to support `srl`

ALUOp	funct3	op_5, $funct7_5$	*ALUControl*	Instruction
00	x	xx	000 (add)	`lw, sw`
01	x	xx	001 (subtract)	`beq`
10	000	00, 01, 10	000 (add)	`add, addi`
	000	11	001 (subtract)	`sub`
	001	**x0**	**111 (shift right logical)**	**`srl, srli`**
	010	xx	101 (set less than)	`slt, slti`
	110	xx	011 (or)	`or, ori`
	111	xx	010 (and)	`and, andi`

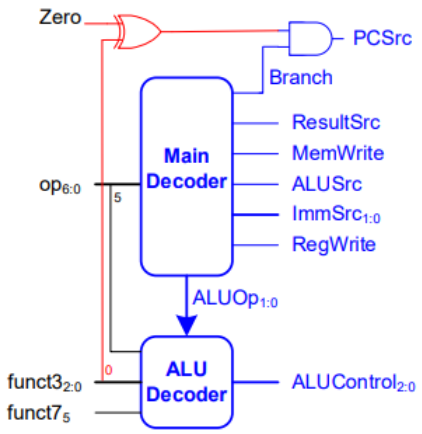
(d) `bne`

`bne` is the opposite of `beq`. `beq` and `bne` can be identified by **func3**$_0$, which is high when `bne` is the instruction. To implement, we simply need to change the control unit to branch when *Zero* is 0 and `bne` is the instruction or when *Zero* is 1 and `beq` is the instruction. This is easily achieved with *Zero* XOR **func3**$_0$.

Main Decoder truth table enhanced to support `bne`

Instruction	Opcode	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	Jump
`Lw`	0000011	1	00	1	0	01	0	00	0
`Sw`	0100011	0	01	1	1	xx	0	00	0
R-type	0110011	1	xx	0	0	00	0	10	0
`beq`**`/bne`**	1100011	0	10	0	0	xx	1	01	0
`Addi`	0010011	1	00	0	0	00	0	10	0
`Jal`	1101111	1	11	x	0	10	0	xx	1

Enhanced control unit for `bne`



Exercise 7.4

(a) `lui`

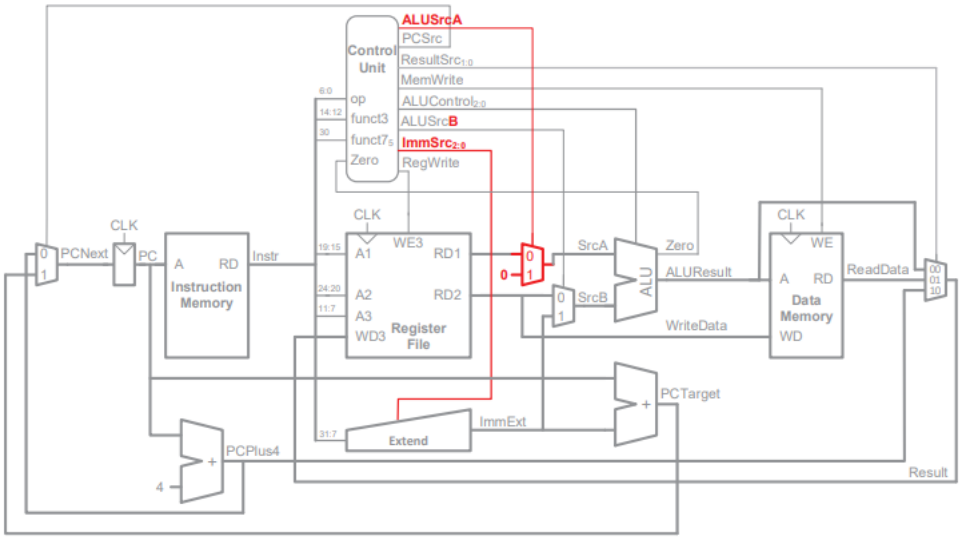
First, we update the immediate Extend unit to support `lui`.

Enhanced *ImmSrc* encoding to support `lui`

ImmSrc	*ImmExt*	Type	Description
000	{{20{*Instr*[31]}}, *Instr*[31:20]}	I	12-bit signed immediate
001	{{20{*Instr*[31]}}, *Instr*[31:25], *Instr*[11:7]}	S	12-bit signed immediate
010	{{20{*Instr*[31]}}, *Instr*[7], *Instr*[30:25], *Instr*[11:8], 1'b0}	B	13-bit signed immediate
011	{{12{*Instr*[31]}}, *Instr*[19:12], *Instr*[20], *Instr*[30:21], 1'b0}	J	21-bit signed immediate
100	**{*Instr*[31:12], 12'b0}**	**U**	**20-bit signed immediate**

Next, we modify the datapath by increasing the width of the *ImmSrc* control signal to 3 bits and by making 0 an option for the ALU's top input (*SrcA*).

Enhanced datapath to support `lui`



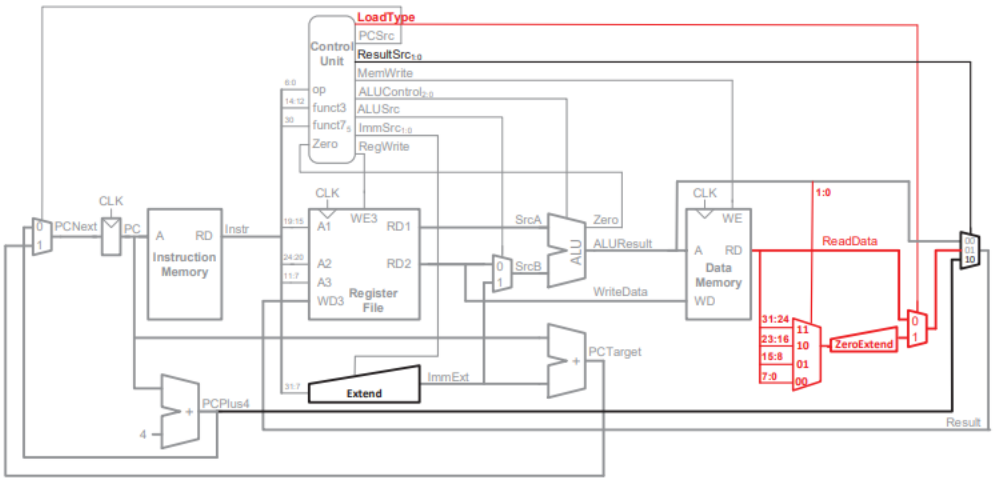
Main Decoder truth table enhanced to support `lui`

Instruction	Opcode	RegWrite	ImmSrc	ALUSrcA	ALUSrcB	MemWrite	ResultSrc	Branch	ALUOp	Jump
`lw`	0000011	1	000	0	1	0	01	0	00	0
`sw`	0100011	0	001	0	1	1	xx	0	00	0
R-type	0110011	1	xxx	0	0	0	00	0	10	0
`beq`	1100011	0	010	0	0	0	xx	1	01	0
I-type ALU	0010011	1	000	0	0	0	00	0	10	0
`jal`	1101111	1	011	x	x	0	10	0	xx	1
`lui`	**0110111**	**1**	**100**	**1**	**1**	**0**	**00**	**0**	**xx**	**0**

(c) `lbu`

To implement `lbu` we create a load/store unit (LSU) within the controller that outputs a new signal *LoadType*. The LU takes in **funct3** and outputs *LoadType*. When *LoadType* is 0, it is an `lbu` instruction, and a zero-extended byte of the *ReadData* bus (selected using the two least significant bits of the address) is sent to the ResultSrc multiplexer. Otherwise, *ReadData* is sent. We add the following to the datapath: the new signal, *LoadType*, zero-extension unit, 4:1 multiplexer to select the byte within the *ReadData* word, and a LoadType multiplexer.

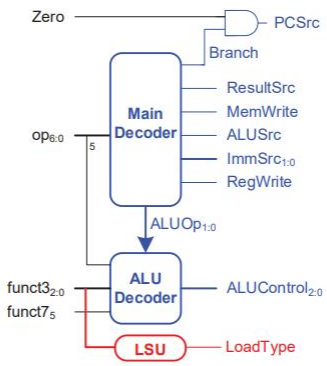
Enhanced datapath to support `lbu`



Main Decoder truth table enhanced to support `lbu`

Instruction	Opcode	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	Jump
`lw` / **`lbu`**	0000011	1	000	1	0	01	0	00	0
`sw`	0100011	0	001	1	1	xx	0	00	0
R-type	0110011	1	xxx	0	0	00	0	10	0
`beq`	1100011	0	010	0	0	xx	1	01	0
I-type ALU	0010011	1	000	0	0	00	0	10	0
`jal`	1101111	1	011	x	0	10	0	xx	1

Enhanced control unit for `lbu`



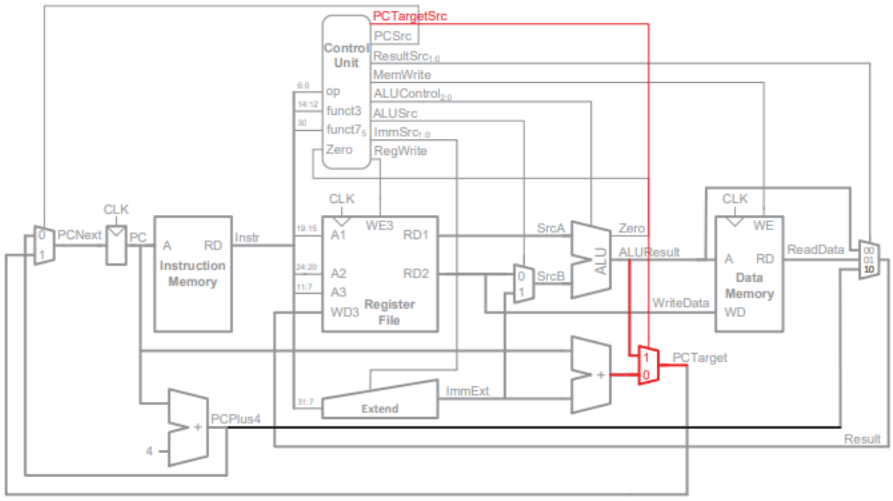
LSU truth table to support `lbu`

funct3	*LoadType*	Instruction
010	0	`lw`
100	1	`lbu`

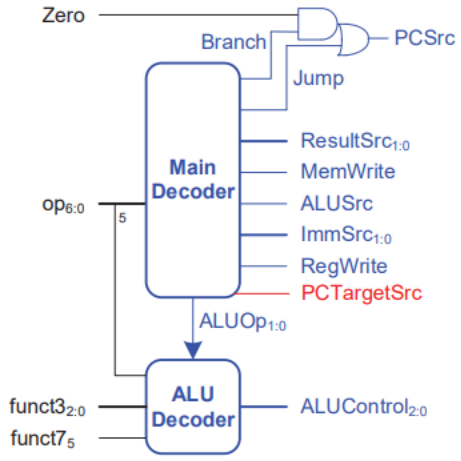
(h) `jalr`

The `jalr` instruction jumps to the target address calculated by adding **rs1** and the sign-extended 12-bit immediate. It also writes PC+4 to **rd**. The datapath already supports these calculations: the ALU can perform **rs1** + *ImmExt*; and PC+4 can be routed to the *Result* signal by the ResultSrc multiplexer. Only the ALU output needs to be routed to the PCTarget signal. So, we add a control signal (*PCTargetSrc*) and a multiplexer to select the correct PCTarget (either from the ALU or the PC-relative adder).

Enhanced datapath to support `jalr`



Main Decoder enhanced to support `jalr`



Main Decoder truth table enhanced to support `jalr`

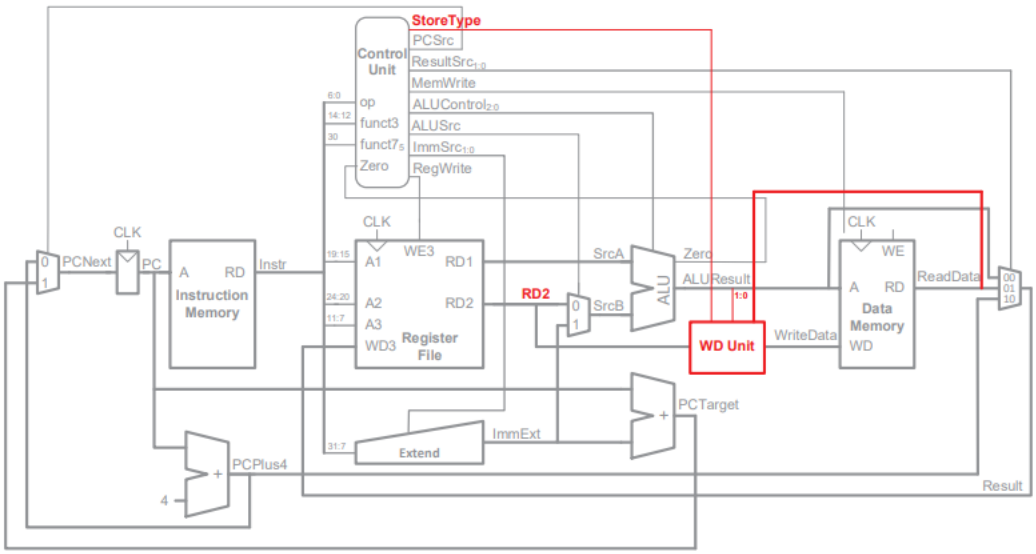
Instruction	Opcode	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	Jump	PCTargetSrc
`lw`	0000011	1	000	1	0	01	0	00	0	x
`sw`	0100011	0	001	1	1	xx	0	00	0	x
R-type	0110011	1	xxx	0	0	00	0	10	0	x
`beq`	1100011	0	010	0	0	xx	1	01	0	0
I-type ALU	0010011	1	000	0	0	00	0	10	0	x
`jal`	1101111	1	011	x	0	10	0	xx	1	0
`jalr`	**1100111**	**1**	**000**	**1**	**0**	**10**	**0**	**xx**	**1**	**1**

(j) `sb`

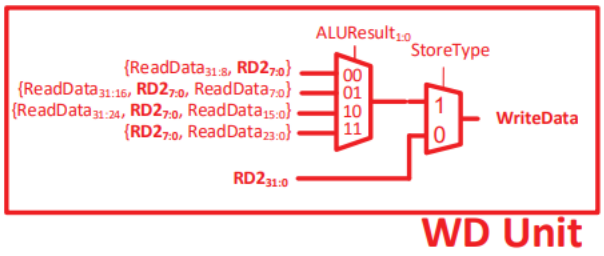
We add a WriteData Unit (WD Unit) to the datapath to write either RD2 or RD2 bit-swizzled with ReadData for sub-word writes. For `sb`, the least significant byte (lsb) of

RD2 (**rs2**'s contents), replaces a byte of data in the *ReadData* bus depending on the byte offset of the memory address, $ALUResult_{1:0}$. We add a *StoreType* output to the control unit to choose either the entire word or the bitswizzled word to write to memory. The updated figures and additional hardware is shown below.

Enhanced datapath to support `sb` (WD Unit shown in next figure)



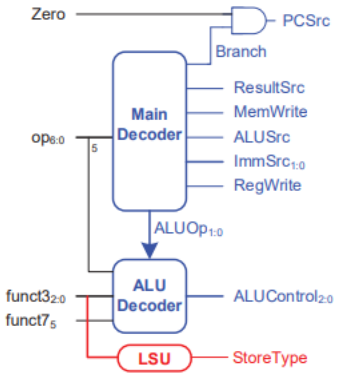
WD Unit



Main Decoder truth table enhanced to support `sb`

Instruction	Opcode	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	Jump
`lw`	0000011	1	000	1	0	01	0	00	0
`sw` / **`sb`**	0100011	0	001	1	1	xx	0	00	0
R-type	0110011	1	xxx	0	0	00	0	10	0
`beq`	1100011	0	010	0	0	xx	1	01	0
I-type ALU	0010011	1	000	0	0	00	0	10	0
`jal`	1101111	1	011	x	0	10	0	xx	1

Enhanced control unit for `sb` : added Load/Store Unit (LSU)



Load/Store Unit (LSU) truth table to support `sb`

funct3	StoreType	Instruction
000	1	`sb`
010	0	`sw`