RISC v3

Data memory is 16 bit words and word addressed
Instruction memory is 16 bit words, word addressed but the code uses byte addresses and converts to words – which is inconsistent

Changing it to use word addresses for instructions too – so PC increments by 1 each cycle

instruction_memory.v

```
14        wire [3 : 0] rom_addr = pc[3 : 0];
```

datapath.v

```
26        wire [11:0] jump_shift;
37        assign pc2 = pc_current + 16'd1;
44        assign jump_shift = instr[11:0];
75        assign PC_beq = pc2 + ext_im[15:0];
76        assign PC_bne = pc2 + ext_im[15:0];
91        assign PC_j = {pc2[15:12],jump_shift};
```

Instruction	**Mnemonic**	**Action**
Load word	LD ws, offset(rs1)	ws := mem16[rs1 + offset]
Store word	ST rs2, offset(rs1)	mem16[rs1 + offset] := rs2
Add	ADD ws, rs1, rs2	ws := rs1 + rs2
Subtract	SUB ws, rs1, rs2	ws := rs1 - rs2
Invert (1s complement)	INV ws, rs1	ws := !rs1
Logical Shift Left	LSL ws, rs1, rs2	ws := rs1 << rs2
Logical Shift Right	LSR ws, rs1,rs2	ws := rs1 >> rs2
Bitwise AND	AND ws, rs1, rs2	ws := rs1 & rs2
Bitwise OR	OR ws, rs1, rs2	ws := rs1 | rs2
Set on Less Than	SLT ws, rs1, rs2	ws := 1 if rs1 < rs2 ws := 0 if rs1 >= rs2
Branch on Equal	BEQ rs1, rs2, offset	PC := PC + 1 + offset if rs1 != rs2
Branch on Not Equal	BNE rs1, rs2, offset	PC := PC + 1+ offset if rs1 == rs2
Jump	JMP offset	PC := PC[15:13], inst[11:0]}

OP	Instruction
0000	Load word
0001	Store word
0002	Add
0003	Subtract
0004	Invert (1s complement)
0005	Logical Shift Left
0006	Logical Shift Right
0007	Bitwise AND
0008	Bitwise OR
0009	Set on Less Than
0011	Branch on Equal
0012	Branch on Not Equal
0013	Jump

Load word

Op	**rs1**	**ws**	**offset**
4	3	3	6

Store word

Op	**rs1**	**rs2**	**offset**
4	3	3	6

Data processing

Op	**rs1**	**rs2**	**ws**	
4	3	3	3	3

Branch

Op	**rs1**	**rs2**	**offset**
4	3	3	6

Jump

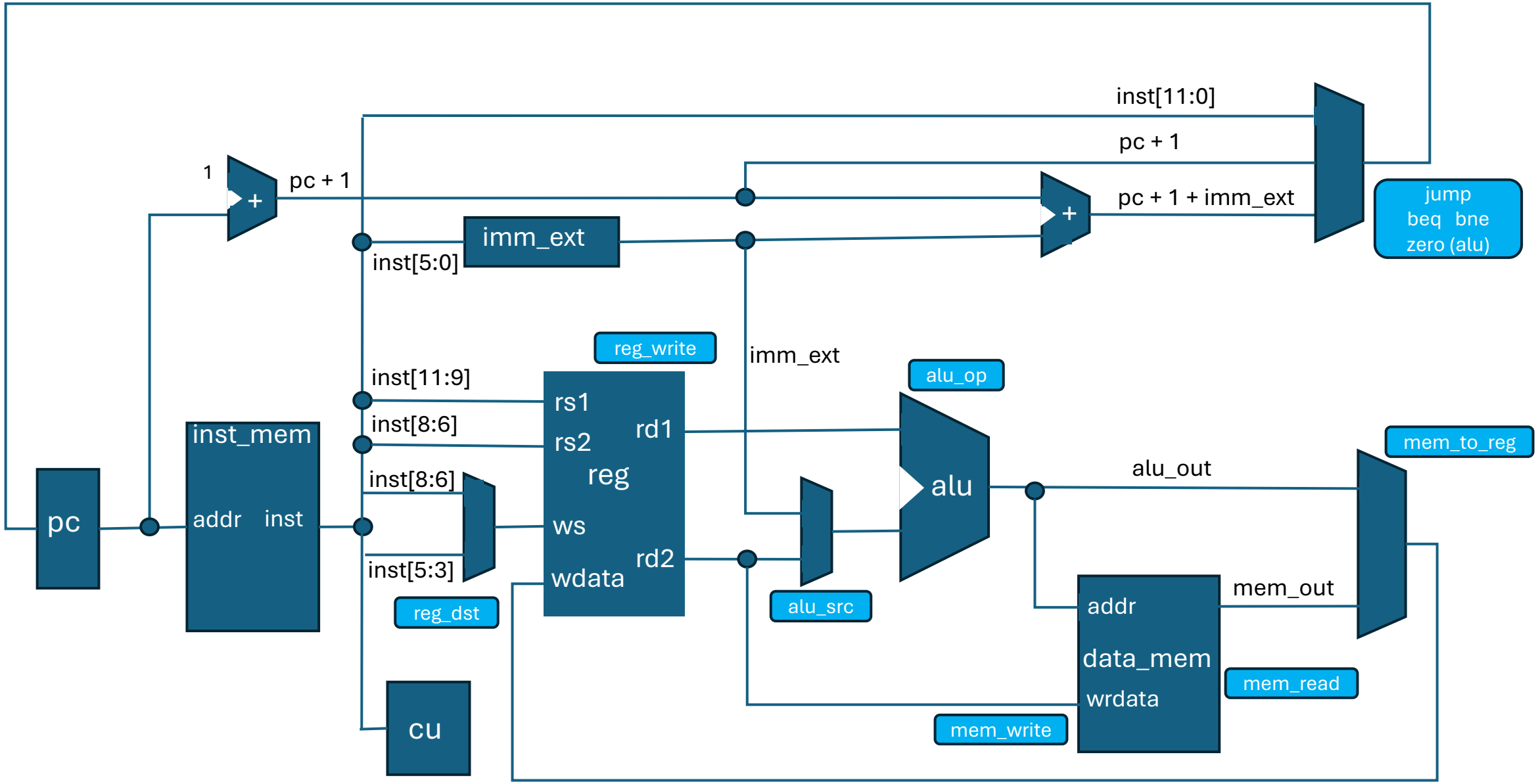
Op	**offset**
4	12

RISC v3

Control signals									
Instruction	**Reg Dst**	**ALUSrc**	**Memto Reg**	**Reg Write**	**Mem Read**	**Mem Write**	**Branch**	**ALUOp**	**Jump**
Data-processing	1	0	0	1	0	0	0	see below	0
LW	0	1	1	1	1	0	0	000	0
SW	0	1	0	0	0	1	0	000	0
BEQ,BNE	0	0	0	0	0	0	1	001	0
J	0	0	0	0	0	0	0	000	1

ALU Op		
Opcode	**ALU Operation**	**ALU Op**
02	ADD	000
03	SUB	001
04	INVERT	010
05	LSL	011
06	LSR	100
07	AND	101
08	OR	110
09	SLT	111

ALU Op			
Opcode	**Instruction**	**ALU Operation**	**ALUop**
00	LW	ADD	000
01	SW	SUB	001
02	ADD	ADD	000
03	SUB	SUB	001
04	INVERT	INVERT	010
05	LSL	LSL	011
06	LSR	LSR	100
07	AND	AND	101
08	OR	OR	110
09	SLT	SLT	111
11	BEQ	SUB	001
12	BNE	SUB	001
13	JMP	ADD	000


inst[11:0]
pc + 1
1
+
pc + 1
+
pc + 1 + imm_ext
jump
beq bne
zero (alu)
imm_ext
inst[5:0]
reg_write
imm_ext
alu_op
inst[11:9]
inst[8:6]
rs1
rs2
reg
rd1
rd2
ws
wdata
inst_mem
addr inst
pc
inst[8:6]
inst[5:3]
reg_dst
alu_src
alu
alu_out
mem_to_reg
addr
data_mem
wrdata
mem_out
mem_read
mem_write
cu

RISC v2

Instruction	**Mnemonic**	**Action**
Load word	LD ws, offset(rs1)	ws := mem16[rs1 + offset]
Store word	ST rs2, offset(rs1)	mem16[rs1 + offset] := rs2
Add	ADD ws, rs1, rs2	ws := rs1 + rs2
Subtract	SUB ws, rs1, rs2	ws := rs1 - rs2
Invert (1s complement)	INV ws, rs1	ws := !rs1
Logical Shift Left	LSL ws, rs1, rs2	ws := rs1 << rs2
Logical Shift Right	LSR ws, rs1,rs2	ws := rs1 >> rs2
Bitwise AND	AND ws, rs1, rs2	ws := rs1 & rs2
Bitwise OR	OR ws, rs1, rs2	ws := rs1 | rs2
Set on Less Than	SLT ws, rs1, rs2	ws := 1 if rs1 < rs2 ws := 0 if rs1 >= rs2
Branch on Equal	BEQ rs1, rs2, offset	PC := PC + 2 + offset << 1 if rs1 != rs2
Branch on Not Equal	BNE rs1, rs2, offset	PC := PC + 2 + offset << 1 if rs1 == rs2
Jump	JMP offset	PC := {PC[15:13], (offset <<1)}

OP	Instruction
0000	Load word
0001	Store word
0002	Add
0003	Subtract
0004	Invert (1s complement)
0005	Logical Shift Left
0006	Logical Shift Right
0007	Bitwise AND
0008	Bitwise OR
0009	Set on Less Than
0011	Branch on Equal
0012	Branch on Not Equal
0013	Jump

Load word

Op	**rs1**	**ws**	**offset**
4	3	3	6

Store word

Op	**rs1**	**rs2**	**offset**
4	3	3	6

Data processing

Op	**rs1**	**rs2**	**ws**	
4	3	3	3	3

Branch

Op	**rs1**	**rs2**	**offset**
4	3	3	6

Jump

Op	**offset**
4	12

RISC v2

Control signals									
Instruction	**Reg Dst**	**ALUSrc**	**Memto Reg**	**Reg Write**	**Mem Read**	**Mem Write**	**Branch**	**ALUOp**	**Jump**
Data-processing	1	0	0	1	0	0	0	see below	0
LW	0	1	1	1	1	0	0	000	0
SW	0	1	0	0	0	1	0	000	0
BEQ,BNE	0	0	0	0	0	0	1	001	0
J	0	0	0	0	0	0	0	000	1

ALU Op		
Opcode	**ALU Operation**	**ALU Op**
02	ADD	000
03	SUB	001
04	INVERT	010
05	LSL	011
06	LSR	100
07	AND	101
08	OR	110
09	SLT	111

ALU Op			
Opcode	**Instruction**	**ALU Operation**	**ALUop**
00	LW	ADD	000
01	SW	SUB	001
02	ADD	ADD	000
03	SUB	SUB	001
04	INVERT	INVERT	010
05	LSL	LSL	011
06	LSR	LSR	100
07	AND	AND	101
08	OR	OR	110
09	SLT	SLT	111
11	BEQ	SUB	001
12	BNE	SUB	001
13	JMP	ADD	000

RISC v1

Control signals									
Instruction	**Reg Dst**	**ALUSrc**	**Memto Reg**	**Reg Write**	**Mem Read**	**Mem Write**	**Branch**	**ALUOp**	**Jump**
Data-processing	1	0	0	1	0	0	0	00	0
LW	0	1	1	1	1	0	0	10	0
SW	0	1	0	0	0	1	0	10	0
BEQ,BNE	0	0	0	0	0	0	1	01	0
J	0	0	0	0	0	0	0	00	1

ALU Control				
ALUOp	Opcode (hex)	ALUcnt	ALU Operation	Instruction
10	xxxx	000	ADD	LW,SW
01	xxxx	001	SUB	BEQ,BNE
00	0002	000	ADD	D-type: ADD
00	0003	001	SUB	D-type: SUB
00	0004	010	INVERT	D-type: INVERT
00	0005	011	LSL	D-type: LSL
00	0006	100	LSR	D-type: LSR
00	0007	101	AND	D-type: AND
00	0008	110	OR	D-type: OR
00	0009	111	SLT	D-type: SLT