

Computation II: embedded system design (5EIB0)

[Dashboa...](#) / [My cours...](#) / [5EIB0](#) / [Final Exam 2024-04-...](#) / [Part 2a - 2024-04-12 Adding Custom Instruction \(Selective Biasing Calculation - ...](#)

 **Available from:**

 **Due date:**

 **Requested files:**

Type of work:

Introduction

The code below performs selctive biasing. Selective biasing checks an image pixel-by-pixel. All the pixels above a threshold are biased down (subtracted) by 100 whereas all the pixels below the threshold are biased up (added) by 100. The C code below handles this with an if-statement resulting in many processor instructions.

```
#define WIDTH  32
#define HEIGHT 32

void main(void)
{
    int a, b, result;
    unsigned char *buf_i = (unsigned char*)0x401000, *buf_o = (unsigned char*)0x402000;

    for (a = 1; a < HEIGHT - 1; a++)
    {
        for (b = 1; b < WIDTH - 1; b++)
        {
            /*biasing*/
            result = buf_i[a * WIDTH + b];
            if(result > 128) buf_o[a * WIDTH + b] = result-100;
            else buf_o[a * WIDTH + b] = result+100;

        }
    }
}
```

To reduce the number of instructions, the C code is modified to use a custom instruction (biasing), giving the following C code:

```
#define WIDTH  32
#define HEIGHT 32
#define biasing(p, q) ((p) + ((q) - *(int *) 0x12344321)) //HEX Code: 0x32

void main(void)
{
    int a, b, result, threshold;
    unsigned char *buf_i = (unsigned char*)0x401000, *buf_o = (unsigned char*)0x402000;

    for (a = 1; a < HEIGHT - 1; a++)
    {
        for (b = 1; b < WIDTH - 1; b++)
        {
            /*biasing*/
            result = buf_i[a * WIDTH + b];
            buf_o[a * WIDTH + b] = biasing(result, threshold);

        }
    }
}
```

Add hardware support for the custom instruction (**biasing**) in the current mMIPS implementation by modifying all or some of the provided files (ctrl.v, aluctrl.v and alu.v). You can infer the functionality of **biasing** by comparing the C codes above.

Note: The biasing function code is 0x32.

Debug

For debugging the design, you can use the "**\$display**" command in the RTL code to print out the values of interest. **Note: "\$display" will only work in the debug mode of VPL.**

Requested files

ctrl.v


```

104 //Determine the output
105 case (Opcode)
106 0: // R-format instruction: check functioncode
107     case (FunctionCode)
108         'h8: // Instruction: Jr
109             begin
110                 RegDst      = 2'b01;
111                 Target      = 2'b10;
112                 ALUSrc      = 1'b0;
113                 MemtoReg    = 2'b00;
114                 RegWrite    = 1'b0;
115                 MemRead     = 2'b00;
116                 MemWrite    = 2'b00;
117                 Branch      = 2'b11;
118                 ALUop       = 5'b00010;
119                 SignExtend  = 1'b1;
120                 HiLowWrite  = 1'b0;
121                 AluSel      = 2'b00;
122             end
123         'h9: // Instruction Jalr
124             begin
125                 RegDst      = 2'b01;
126                 Target      = 2'b10;
127                 ALUSrc      = 1'b0;
128                 MemtoReg    = 2'b00;
129                 RegWrite    = 1'b1;
130                 MemRead     = 2'b00;
131                 MemWrite    = 2'b00;
132                 Branch      = 2'b11;
133                 ALUop       = 5'b00010;
134                 SignExtend  = 1'b1;
135                 HiLowWrite  = 1'b0;
136                 AluSel      = 2'b11;
137             end
138         'h10: // Instruction: Move hi register
139             begin
140                 RegDst      = 2'b01;
141                 Target      = 2'b00;
142                 ALUSrc      = 1'b0;
143                 MemtoReg    = 2'b00;
144                 RegWrite    = 1'b1;
145                 MemRead     = 2'b00;
146                 MemWrite    = 2'b00;
147                 Branch      = 2'b00;
148                 ALUop       = 5'b00010;
149                 SignExtend  = 1'b1;
150                 HiLowWrite  = 1'b0;
151                 AluSel      = 2'b10;
152             end
153         'h12: // Instruction: Move lo register
154             begin
155                 RegDst      = 2'b01;
156                 Target      = 2'b00;
157                 ALUSrc      = 1'b0;
158                 MemtoReg    = 2'b00;
159                 RegWrite    = 1'b1;
160                 MemRead     = 2'b00;
161                 MemWrite    = 2'b00;
162                 Branch      = 2'b00;
163                 ALUop       = 5'b00010;
164                 SignExtend  = 1'b1;
165                 HiLowWrite  = 1'b0;
166                 AluSel      = 2'b01;
167             end
168         'h19: // Instruction: Multiply unsigned
169             begin
170                 RegDst      = 2'b00; //No destination
171                 Target      = 2'b00;
172                 ALUSrc      = 1'b0;
173                 MemtoReg    = 2'b00;
174                 RegWrite    = 1'b1;
175                 MemRead     = 2'b00;
176                 MemWrite    = 2'b00;
177                 Branch      = 2'b00;
178                 ALUop       = 5'b00010;
179                 SignExtend  = 1'b1;
180                 HiLowWrite  = 1'b1;
181                 AluSel      = 2'b00;
182             end
183         default: // Others
184             begin
185                 RegDst      = 2'b01;
186                 Target      = 2'b00;
187                 ALUSrc      = 1'b0;
188                 MemtoReg    = 2'b00;
189                 RegWrite    = 1'b1;
190                 MemRead     = 2'b00;
191                 MemWrite    = 2'b00;
192                 Branch      = 2'b00;
193                 ALUop       = 5'b00010;
194                 SignExtend  = 1'b1;
195                 HiLowWrite  = 1'b0;
196                 AluSel      = 2'b00;
197             end
198     endcase
199 2: // Instruction: J
200     begin
201         RegDst      = 2'b00;
202         Target      = 2'b01;
203         ALUSrc      = 1'b0;
204         MemtoReg    = 2'b00;
205         RegWrite    = 1'b0;
206         MemRead     = 2'b00;
207         MemWrite    = 2'b00;
208         Branch      = 2'b11;
209         ALUop       = 5'b00010;
210         SignExtend  = 1'b1;
211         HiLowWrite  = 1'b0;
212         AluSel      = 2'b00;
213     end

```

```

207         MemWrite   = 2'b00;
208         Branch     = 2'b11;
209         ALUOp      = 5'b00010;
210         SignExtend = 1'b1;
211         HiLoWrite  = 1'b0;
212         AluSel     = 2'b00;
213     end
214 3: // Instruction: Jal
215     begin
216         RegDst      = 2'b10;
217         Target      = 2'b01;
218         ALUSrc      = 1'b0;
219         MemtoReg    = 2'b00;
220         RegWrite    = 1'b1;
221         MemRead     = 2'b00;
222         MemWrite    = 2'b00;
223         Branch      = 2'b11;
224         ALUOp       = 5'b00010;
225         SignExtend  = 1'b1;
226         HiLoWrite   = 1'b0;
227         AluSel      = 2'b11;
228     end
229 4: // Instruction: BEQ
230     begin
231         RegDst      = 2'b00;
232         Target      = 2'b00;
233         ALUSrc      = 1'b0;
234         MemtoReg    = 2'b00;
235         RegWrite    = 1'b0;
236         MemRead     = 2'b00;
237         MemWrite    = 2'b00;
238         Branch      = 2'b01;
239         ALUOp       = 5'b00001;
240         SignExtend  = 1'b1;
241         HiLoWrite   = 1'b0;
242         AluSel      = 2'b00;
243     end
244 5: // Instruction: BNE
245     begin
246         RegDst      = 2'b00;
247         Target      = 2'b00;
248         ALUSrc      = 1'b0;
249         MemtoReg    = 2'b00;
250         RegWrite    = 1'b0;
251         MemRead     = 2'b00;
252         MemWrite    = 2'b00;
253         Branch      = 2'b10;
254         ALUOp       = 5'b00001;
255         SignExtend  = 1'b1;
256         HiLoWrite   = 1'b0;
257         AluSel      = 2'b00;
258     end
259 9: // Instruction: ADDIU
260     begin
261         RegDst      = 2'b00;
262         Target      = 2'b00;
263         ALUSrc      = 1'b1;
264         MemtoReg    = 2'b00;
265         RegWrite    = 1'b1;
266         MemRead     = 2'b00;
267         MemWrite    = 2'b00;
268         Branch      = 2'b00;
269         ALUOp       = 5'b00011;
270         SignExtend  = 1'b1;
271         HiLoWrite   = 1'b0;
272         AluSel      = 2'b00;
273     end
274 10: // Instruction: SLTI
275     begin
276         RegDst      = 2'b00;
277         Target      = 2'b00;
278         ALUSrc      = 1'b1;
279         MemtoReg    = 2'b00;
280         RegWrite    = 1'b1;
281         MemRead     = 2'b00;
282         MemWrite    = 2'b00;
283         Branch      = 2'b00;
284         ALUOp       = 5'b00111;
285         SignExtend  = 1'b1;
286         HiLoWrite   = 1'b0;
287         AluSel      = 2'b00;
288     end
289 11: // Instruction: SLTUI
290     begin
291         RegDst      = 2'b00;
292         Target      = 2'b00;
293         ALUSrc      = 1'b1;
294         MemtoReg    = 2'b00;
295         RegWrite    = 1'b1;
296         MemRead     = 2'b00;
297         MemWrite    = 2'b00;
298         Branch      = 2'b00;
299         ALUOp       = 5'b01000;
300         SignExtend  = 1'b1;
301         HiLoWrite   = 1'b0;
302         AluSel      = 2'b00;
303     end
304 12: // Instruction: ANDI
305     begin
306         RegDst      = 2'b00;
307         Target      = 2'b00;
308         ALUSrc      = 1'b1;
309         MemtoReg    = 2'b00;
310         RegWrite    = 1'b1;

```

```

310         regWrite = 1'b1;
311         MemRead   = 2'b00;
312         MemWrite  = 2'b00;
313         Branch    = 2'b00;
314         ALUOp     = 5'b00100;
315         SignExtend = 1'b0;
316         HiLoWrite = 1'b0;
317         AluSel    = 2'b00;
318     end
319 13: // Instruction: ORI
320     begin
321         RegDst    = 2'b00;
322         Target    = 2'b00;
323         ALUSrc    = 1'b1;
324         MemtoReg  = 2'b00;
325         RegWrite  = 1'b1;
326         MemRead   = 2'b00;
327         MemWrite  = 2'b00;
328         Branch    = 2'b00;
329         ALUOp     = 5'b00101;
330         SignExtend = 1'b0;
331         HiLoWrite = 1'b0;
332         AluSel    = 2'b00;
333     end
334 14: // Instruction: XORI
335     begin
336         RegDst    = 2'b00;
337         Target    = 2'b00;
338         ALUSrc    = 1'b1;
339         MemtoReg  = 2'b00;
340         RegWrite  = 1'b1;
341         MemRead   = 2'b00;
342         MemWrite  = 2'b00;
343         Branch    = 2'b00;
344         ALUOp     = 5'b00110;
345         SignExtend = 1'b0;
346         HiLoWrite = 1'b0;
347         AluSel    = 2'b00;
348     end
349 15: // Instruction: LUI
350     begin
351         RegDst    = 2'b00;
352         Target    = 2'b00;
353         ALUSrc    = 1'b1;
354         MemtoReg  = 2'b00;
355         RegWrite  = 1'b1;
356         MemRead   = 2'b00;
357         MemWrite  = 2'b00;
358         Branch    = 2'b00;
359         ALUOp     = 5'b01001;
360         SignExtend = 1'b1;
361         HiLoWrite = 1'b0;
362         AluSel    = 2'b00;
363     end
364 32: //Instruction: LB
365     begin
366         RegDst    = 2'b00;
367         Target    = 2'b00;
368         ALUSrc    = 1'b1;
369         MemtoReg  = 2'b10;
370         RegWrite  = 1'b1;
371         MemRead   = 2'b10;
372         MemWrite  = 2'b00;
373         Branch    = 2'b00;
374         ALUOp     = 5'b00000;
375         SignExtend = 1'b1;
376         HiLoWrite = 1'b0;
377         AluSel    = 2'b00;
378     end
379 35: // Instruction: LW
380     begin
381         RegDst    = 2'b00;
382         Target    = 2'b00;
383         ALUSrc    = 1'b1;
384         MemtoReg  = 2'b01;
385         RegWrite  = 1'b1;
386         MemRead   = 2'b01;
387         MemWrite  = 2'b00;
388         Branch    = 2'b00;
389         ALUOp     = 5'b00000;
390         SignExtend = 1'b1;
391         HiLoWrite = 1'b0;
392         AluSel    = 2'b00;
393     end
394 40: // Instruction: SB
395     begin
396         RegDst    = 2'b00;
397         Target    = 2'b00;
398         ALUSrc    = 1'b1;
399         MemtoReg  = 2'b00;
400         RegWrite  = 1'b0;
401         MemRead   = 2'b00;
402         MemWrite  = 2'b10;
403         Branch    = 2'b00;
404         ALUOp     = 5'b00000;
405         SignExtend = 1'b1;
406         HiLoWrite = 1'b0;
407         AluSel    = 2'b00;
408     end
409 43: // Instruction: SW
410     begin
411         RegDst    = 2'b00;
412         Target    = 2'b00;
413         ALUSrc    = 1'b1;

```

```

414             MentoReg  = 2'b00;
415             RegWrite  = 1'b0;
416             MemRead   = 2'b00;
417             MemWrite  = 2'b01;
418             Branch    = 2'b00;
419             ALUop      = 5'b00000;
420             SignExtend = 1'b1;
421             HiLoWrite  = 1'b0;
422             AluSel     = 2'b00;
423         end
424         default: //No default case
425             begin
426                 end
427             endcase
428         end
429
430     endmodule
431

```

aluctrl.v

```

1 ///////////////////////////////////////////////////////////////////
2 // ALUCTRL.V
3 //
4 // TU/e Eindhoven University Of Technology
5 // Eindhoven, The Netherlands
6 //
7 // Created: 21-11-2013
8 // Author: Bergmans, G (g.bergmans@student.tue.nl)
9 // Based on work by Sander Stuijk
10 //
11 // Function:
12 //     ALU controller
13 //
14 // Version:
15 //     (27-01-2014): initial version
16 //
17 ///////////////////////////////////////////////////////////////////
18
19 module ALUCTRL(functionCode, ALUOp, Shamt, ALUctrl);
20     input  [5:0]  functionCode;
21     input  [4:0]  ALUOp;
22     input  [4:0]  Shamt;
23     output [5:0]  ALUctrl;
24     reg  [5:0]  ALUctrl;
25
26     always @(functionCode or ALUOp or Shamt)
27         begin : aluctrl_thread
28             case (ALUOp) //synopsys parallel_case
29                 'h0: // Add signed
30                     ALUctrl = 'h2;
31
32                 'h1: // Subtract unsigned
33                     ALUctrl = 'h6;
34
35                 'h2: // R-type instruction, look to functionCode
36                     begin
37                         case (functionCode)
38                             'h0: // SLL
39                                 case (Shamt) //Check shift amount
40                                     1:
41                                         ALUctrl = 'hA;
42                                     2:
43                                         ALUctrl = 'hB;
44                                     8:
45                                         ALUctrl = 'hC;
46                                 default:
47                                     ALUctrl = 'h0;
48                             endcase
49
50                             'h2: // SRL
51                                 case (Shamt) //Check shift amount
52                                     1:
53                                         ALUctrl = 'hD;
54                                     2:
55                                         ALUctrl = 'hE;
56                                     8:
57                                         ALUctrl = 'hF;
58                                 default:
59                                     ALUctrl = 'h0;
60                             endcase
61
62                             'h3: // SRA
63                                 case (Shamt) //Check shift amount
64                                     1:
65                                         ALUctrl = 'h10;
66                                     2:
67                                         ALUctrl = 'h11;
68                                     8:
69                                         ALUctrl = 'h12;
70                                 default:
71                                     ALUctrl = 'h0;
72                             endcase
73
74                             'h10: // Move hi register (nop in ALU)
75                                 ALUctrl = 'h0;
76
77                             'h12: // Move hi register (nop in ALU)
78                                 ALUctrl = 'h0;
79
80                             'h19: // Multiply unsigned
81                                 ALUctrl = 'h13;
82
83                             'h20: // Add signed
84                                 ALUctrl = 'h2;
85
86                             'h21: // Add unsigned
87                                 ALUctrl = 'h3;
88
89                             'h23: // Subtract unsigned
90                                 ALUctrl = 'h6;
91
92                             'h24: // And
93                                 ALUctrl = 'h0;
94
95                             'h25: // Or
96                                 ALUctrl = 'h1;
97
98                             'h26: // Xor
99                                 ALUctrl = 'h4;
100
101                             'h2A: //Set-on-less-than (2's complement)
102                                 ALUctrl = 'h7;

```



```

104         'h2B: //Set-on-less-than (unsigned)
105             ALUctrl = 'h8;
106
107         default:
108             ALUctrl = 'h0;
109     endcase
110 end
111 'h3: // Add unsigned
112     ALUctrl = 6'b000011;
113
114 'h4: // And
115     ALUctrl = 6'b000000;
116
117 'h5: // Or
118     ALUctrl = 6'b000001;
119
120 'h6: // Xor
121     ALUctrl = 6'b000100;
122
123 'h7: //Slt
124     ALUctrl = 6'b000111;
125
126 'h8: //Sltu
127     ALUctrl = 6'b001000;
128
129 'h9: //Load upper immediate
130     ALUctrl = 6'b001001;
131
132     default:
133         ALUctrl = 6'b000000;
134 endcase
135 end
136
137 endmodule
138

```

alu.v

```

1 ///////////////////////////////////////////////////////////////////
2 // ALU.V
3 //
4 // TU/e Eindhoven University Of Technology
5 // Eindhoven, The Netherlands
6 //
7 // Created: 21-11-2013
8 // Author: Bergmans, G (g.bergmans@student.tue.nl)
9 // Based on work by Sander Stuijk
10 //
11 // Function:
12 //   Arithmetic Logic Unit
13 //
14 // Version:
15 //   (27-01-2014): initial version
16 //
17 ///////////////////////////////////////////////////////////////////
18
19 module ALU(ctrl, a, b, r, r2, z);
20     input    [5:0]  ctrl;
21     input    [31:0] a;
22     input    [31:0] b;
23     output   [31:0] r;
24     reg      [31:0] r;
25     output   [31:0] r2;
26     reg      [31:0] r2;
27     output   [0:0]  z;
28     reg      [0:0]  z;
29     reg      [31:0] s;
30     reg      [31:0] t;
31     reg signed [31:0] s_int;
32     reg signed [31:0] t_int;
33     reg      [31:0] result;
34     reg      [31:0] result_hi;
35     reg      [0:0]  sign;
36     reg signed [63:0] c;
37     reg      [0:0]  zero;
38
39     always @(ctrl or a or b)
40         begin : alu_thread
41
42             //Read the inputs
43             s      = a;
44             t      = b;
45             s_int   = s;
46             t_int   = t;
47             result  = 0;
48             result_hi = 0;
49
50             // Calculate result using selected operation
51             case (ctrl)
52                 'h0:    // And
53                     result = s & t;
54
55                 'h1:    // Or
56                     result = s | t;
57
58                 'h2:    // Add signed
59                     result = s_int + t_int;
60
61                 'h3:    // Add unsigned
62                     result = s + t;
63
64                 'h4:    // Xor
65                     result = s ^ t;
66
67                 'h6:    // Subtract signed
68                     result = s - t;
69
70                 'h7:    // Set-on-less-than
71                     if (s_int < t_int)
72                         result = 1;
73                     else
74                         result = 0;
75
76                 'h8:    // Set-on-less-than unsigned
77                     if (s < t)
78                         result = 1;
79                     else
80                         result = 0;
81
82                 'h9:    // Load upper immediate
83                     result = (t << 16);
84
85                 'hA:    // SLL (1 bit)
86                     result = (t << 1);
87
88                 'hB:    // SLL (2 bit)
89                     result = (t << 2);
90
91                 'hC:    // SLL (8 bit)
92                     result = (t << 8);
93
94                 'hD:    // SRL (1 bit)
95                     result = (t >> 1);
96
97                 'hE:    // SRL (2 bit)
98                     result = (t >> 2);
99
100                'hF:    // SRL (8 bit)
101                    result = (t >> 8);
102
103                'h10:   // SRA (1 bit)

```

```

104         begin
105             sign = t[31:31];
106             result = (t >> 1);
107             result[31:31] = sign;
108         end
109
110     'h11: // SRA (2 bit)
111     begin
112         sign = t[31:31];
113         result = (t >> 2);
114         result[31:30] = {sign, sign};
115     end
116
117     'h12: // SRA (8 bit)
118     begin
119         sign = t[31:31];
120         result = (t >> 8);
121         result[31:24] = {sign, sign, sign, sign, sign, sign, sign, sign};
122     end
123
124     'h13: //Multu
125     begin
126         c = s * t;
127         result = c[31:0];
128         result_hi = c[63:32];
129     end
130
131     default: //No default case: invallid opcode!
132     begin
133     end
134 endcase
135
136 // Calculate zero output
137 if (result == 0)
138     zero = 1;
139 else
140     zero = 0;
141
142 // Write results to output
143 r = result;
144 r2 = result_hi;
145 z = zero;
146 end
147
148 endmodule
149

```