

This ULU

The *ULU.08 – Adder & ALU* can be used as an adder and ALU.

Used parts

Only standard parts are used:

1x casing 80 x 50 x 20mm;
4x 2mm signal connector;
4x black O-ring 9 x 5 x 2mm;
4x 4-bit data connector;
4x colored O-ring 8 x 5 x 1.5mm;
1x power connector;
14x 10K pull-up resistor;
2x 1-pole ON-ON switch;
5x micro (G6K-2F-Y-5VDC) relay;

5x fly back diode (1N4148);
1x Arduino Nano;
1x 5-LED bar graph;
2x M3 M/F standoff;
2x M3 8mm bolt;
2x M3 lock nut;
1x 20 x 8-hole prototype board;
1x 11 x 5-hole prototype board;
5x power diode (1N5817).

Construction

The standard ULU specifications are applicable as specified in the datasheet ULU.00 – Common specifications.

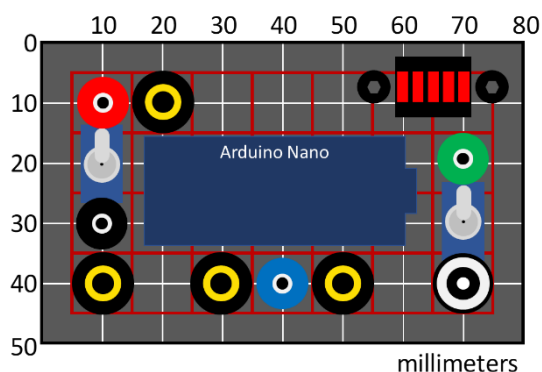


Figure 1 – Drill guide

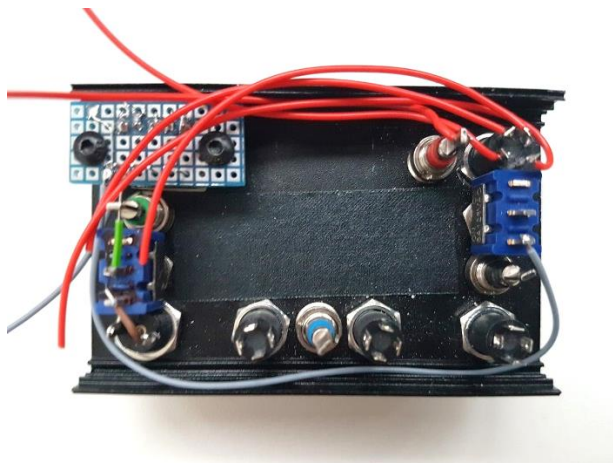


Figure 2 – ULU construction

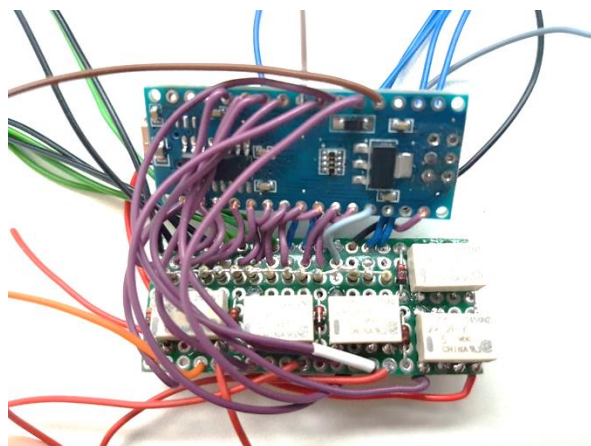


Figure 3 – Top side PCB

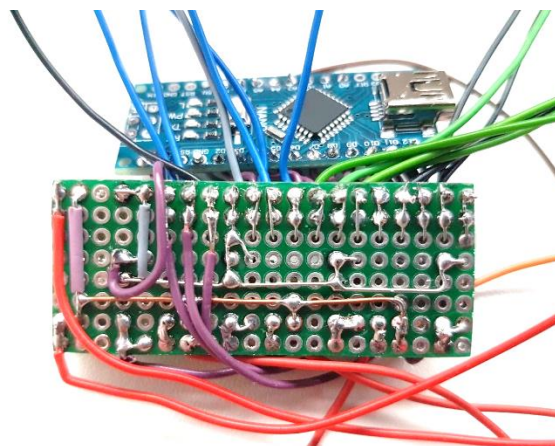


Figure 4 – Bottom side PCB

	Val	Bits	Binary	2-complement
0	-8	1000		1000
1	-7	1001		1001
2	-6	1010		1010
3	-5	1011		1011
4	-4	1100		1100
5	-3	1101		1101
6	-2	1110		1110
7	-1	1111		1111
8	0	0000	0000	0000
9	1	0001	0001	0001
10	2	0010	0010	0010
11	3	0011	0011	0011
12	4	0100	0100	0100
13	5	0101	0101	0101
14	6	0110	0110	0110
15	7	0111	0111	0111
16	8	1000	1000	
17	9	1001	1001	
18	10	1010	1010	
19	11	1011	1011	
20	12	1100	1100	
21	13	1101	1101	
22	14	1110	1110	
23	15	1111	1111	

Figure 5 – Used binary representations

Code	Input
1	0 0 0 1 Right carry
2	0 0 1 0 Right 0
3	0 0 1 1 Right 1
4	0 1 0 0 Right 2
5	0 1 0 1 Right 3
6	0 1 1 0 Left 0
7	0 1 1 1 Left 1
8	1 0 0 0 Left 2
9	1 0 0 1 Left 3
10	1 0 1 0 Operator 0
11	1 0 1 1 Operator 1
12	1 1 0 0 Operator 2
13	1 1 0 1 Operator 3
14	1 1 1 0 Latch
15	1 1 1 1 2 complement

Figure 6 – Solder check code

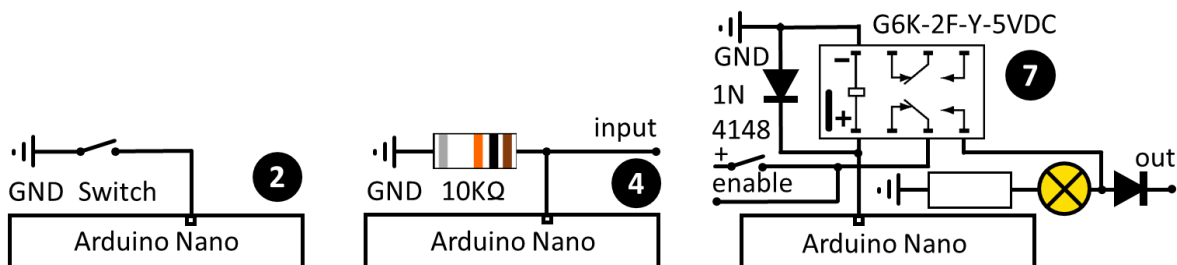


Figure 7 – Used Arduino interfaces

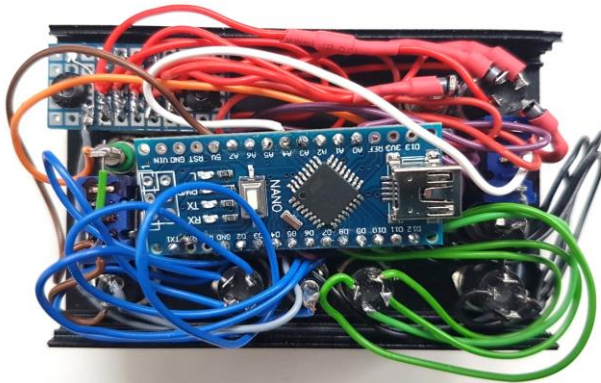


Figure 8 – ULU inside



Figure 9 – Finished ULU

	Port	Con.	Rest.	Func.	Interface	Signal
1.	D1	□	O	O	7	Overflow
2.	D0	-	O	-		n.u.
3.	D2	④		I	4	Right.1
4.	D3	④		I	4	Right.2
5.	D4	④		I	4	Right.3
6.	D5	④		I	4	Left.0
7.	D6	④		I	4	Left.1
8.	D7	④		I	4	Left.2
9.	D8	④		I	4	Left.3
10.	D9	④		I	4	Action.0
11.	D10	④		I	4	Action.1
12.	D11	④		I	4	Action.2
13.	D12	④		I	4	Action.3
14.	D13		O	L		Heartbeat
15.	A0	□		O	7	Output.3
16.	A1	□		O	7	Output.2
17.	A2	□		O	7	Output.1
18.	A3	□		O	7	Output.0
19.	A4	⏏		I	2	2-complement
20.	A5	④		I	4	Right.0
21.	A6	●	I	I	4	Latch
22.	A7	●	I	I	4	Carry
23.	+5V	⊙	I	I	0	DC +5V
24.	GND	⊙	I	I	0	GND

Input, Output, Led, SPI, Toggle switch, Rotary switch

Figure 10 – Pinout Arduino Nano

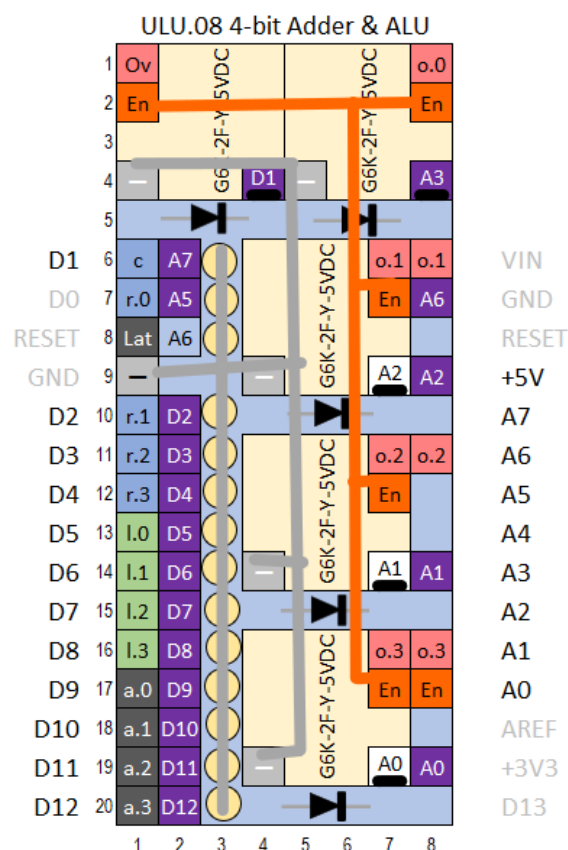


Figure 11 – Layout resistor & relay PCB

Construction is straight forward. The main PCB is very dense packaged. First the resistors and connection wires for the sockets are soldered. Then the connecting wires on the PCB are put in place. Finally, the relays are soldered. Be sure that the fly-back diodes are properly soldered to both pins of the relay coils.

After that the Arduino can be connected to the main PCB. Be sure that both PCB's can fold like a book, for easy access (repairs) of both PCB's.

The LED-bar PCB is a tiny PCB that contains the resistors for this bar, as well as the bar itself. The resistors are placed on an angle, to avoid touching the casing. The output of the relays is first routed to this PCB and then via a diode to the corresponding output socket. Make sure the mounting holes in the PCB are drilled after the holes in the casing are drilled. Otherwise, it is hard to line them up correctly.

The solder check program can be used to test the soldering. The output will be encoded according to the provided input conform Figure 6.

Usage

The available actions (Figure 13) are classified in 2-bit, 3-bit and 4-bit actions. When no action is given (0000) the ULU will act as an adder. The output is only available to both output connectors when the enable switch is set upwards or a 1 is provided to the enable socket. The ULU will use the binary representation, is the 2-complement switch is downwards. Otherwise, it is using the 2-complement encoding. A command is available (7) to convert the left input from binary to 2-complement encoding.

This ULU has two operation modes: latched and regular. The ULU starts in the regular mode. When a signal is put on the latching socket, the latching mode is enabled. It will stay in latching mode until a reset is enforced by removing the power. The latching mode can be used to feedback the output to the right input. It will preserve the output from the first calculation and store it for use in the second calculation. When the latch signal is going from 1 to 0, the results of the current calculation will become available for the output, but only if the output is enabled.

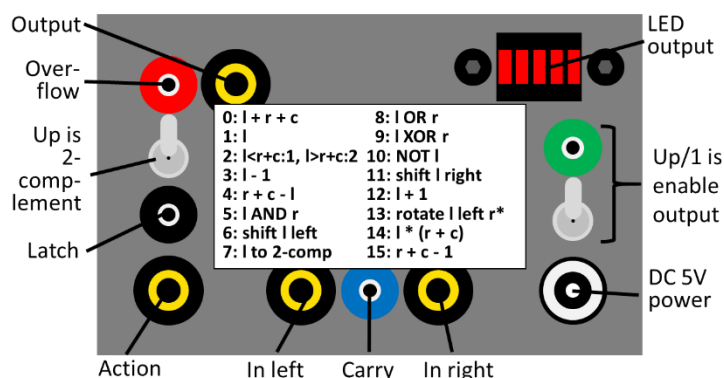


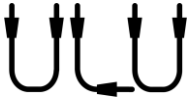
Figure 12 – Controls and connectors

Code	Operation
0	0 0 0 0 l + r + c
1	0 0 0 1 l
2	0 0 1 0 l < r + c:1, l > r + c:2
3	0 0 1 1 l - 1
4	0 1 0 0 r + c - l
5	0 1 0 1 l AND r
6	0 1 1 0 shift l left
7	0 1 1 1 l to 2-comp
8	1 0 0 0 l OR r
9	1 0 0 1 l XOR r
10	1 0 1 0 NOT l
11	1 0 1 1 shift l right
12	1 1 0 0 l + 1
13	1 1 0 1 rotate l left r*
14	1 1 1 0 l * (r + c)
15	1 1 1 1 r + c - 1

Figure 13 – Available actions

The following actions can be performed:

1. Add left, right and carry.
2. Pass through left unchanged.
3. If left is less than right plus carry, the result is 1. If greater the result is 2. If equal the result is 0.
4. Subtract left from right plus carry.
5. Perform an AND between left and right.
6. Shift left one position to the right
7. Convert left from binary to 2-complement coding.
8. Perform an OR between left and right.
9. Perform an XOR between left and right.
10. NOT left.
11. Shift left one position to the right
12. Increment left by one.
13. Rotate left right positions.
14. Multiply left with right plus carry
15. Right plus carry minus one



Arduino Nano Solder check

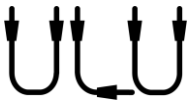
```
/* ULU.08 Adder & ALU - solder check */
/* CC BY-NC-SA Jeroen Brinkman */

bool pressed;

#define OVER 1
#define R1 2
#define R2 3
#define R3 4
#define L0 5
#define L1 6
#define L2 7
#define L3 8
#define AC0 9
#define AC1 10
#define AC2 11
#define AC3 12
#define HEART 13
#define O3 A0
#define O2 A1
#define O1 A2
#define O0 A3
#define TWOCO A4
#define R0 A5
#define LATCH A6
#define RC A7

void setup() {
  pinMode(RC, INPUT);
  pinMode(R0, INPUT);
  pinMode(R1, INPUT);
  pinMode(R2, INPUT);
  pinMode(R3, INPUT);
  pinMode(L0, INPUT);
  pinMode(L1, INPUT);
  pinMode(L2, INPUT);
  pinMode(L3, INPUT);
  pinMode(O0, OUTPUT);
  pinMode(O1, OUTPUT);
  pinMode(O2, OUTPUT);
  pinMode(O3, OUTPUT);
  pinMode(HEART, OUTPUT); // blinking led showing the programma's heartbeat
  pinMode(OVER, OUTPUT);
  pinMode(AC0, INPUT);
  pinMode(AC1, INPUT);
  pinMode(AC2, INPUT);
  pinMode(AC3, INPUT);
  pinMode(LATCH, INPUT);
  pinMode(TWOCO, INPUT_PULLUP);
  digitalWrite(O0, HIGH); delay(300); digitalWrite(O0, LOW); delay(300);
  digitalWrite(O1, HIGH); delay(300); digitalWrite(O1, LOW); delay(300);
  digitalWrite(O2, HIGH); delay(300); digitalWrite(O2, LOW); delay(300);
  digitalWrite(O3, HIGH); delay(300); digitalWrite(O3, LOW); delay(300);
  digitalWrite(OVER, HIGH); delay(300); digitalWrite(OVER, LOW); delay(300);
}

void loop(){
  digitalWrite(HEART, (millis() / 1000) % 2); //1s heartbeat for the onboard led
  pressed = false;
  if ((analogRead(RC) > 500 ? HIGH : LOW) == HIGH) { // 0001
    digitalWrite(O3, LOW); //3
    digitalWrite(O2, LOW); //2
    digitalWrite(O1, LOW); //1
    digitalWrite(O0, HIGH); //0
    pressed = true;
  }
  if (digitalRead(R0) == HIGH) { // 0010
    digitalWrite(O3, LOW); //3
    digitalWrite(O2, LOW); //2
    digitalWrite(O1, HIGH); //1
    digitalWrite(O0, LOW); //0
    pressed = true;
  }
  if (digitalRead(R1) == HIGH) { // 0011
    digitalWrite(O3, LOW); //3
    digitalWrite(O2, LOW); //2
    digitalWrite(O1, HIGH); //1
    digitalWrite(O0, HIGH); //0
    pressed = true;
  }
  if (digitalRead(R2) == HIGH) { // 0100
    digitalWrite(O3, LOW); //3
    digitalWrite(O2, HIGH); //2
    digitalWrite(O1, LOW); //1
    digitalWrite(O0, LOW); //0
    pressed = true;
  }
  if (digitalRead(R3) == HIGH) { // 0101
    digitalWrite(O3, LOW); //3
    digitalWrite(O2, HIGH); //2
    digitalWrite(O1, LOW); //1
    digitalWrite(O0, HIGH); //0
    pressed = true;
  }
  if (digitalRead(L0) == HIGH) { // 0110
    digitalWrite(O3, LOW); //3
    digitalWrite(O2, HIGH); //2
    digitalWrite(O1, HIGH); //1
```



```
digitalWrite(O0, LOW); //0
pressed = true;
}
if (digitalRead(L1) == HIGH) { // 0111
digitalWrite(O3, LOW); //3
digitalWrite(O2, HIGH); //2
digitalWrite(O1, HIGH); //1
digitalWrite(O0, HIGH); //0
pressed = true;
}
if (digitalRead(L2) == HIGH) { // 1000
digitalWrite(O3, HIGH); //3
digitalWrite(O2, LOW); //2
digitalWrite(O1, LOW); //1
digitalWrite(O0, LOW); //0
pressed = true;
}
if (digitalRead(L3) == HIGH) { // 1001
digitalWrite(O3, HIGH); //3
digitalWrite(O2, LOW); //2
digitalWrite(O1, LOW); //1
digitalWrite(O0, HIGH); //0
pressed = true;
}
if (digitalRead(AC0) == HIGH) { // 1010
digitalWrite(O3, HIGH); //3
digitalWrite(O2, LOW); //2
digitalWrite(O1, HIGH); //1
digitalWrite(O0, LOW); //0
pressed = true;
}
if (digitalRead(AC1) == HIGH) { // 1011
digitalWrite(O3, HIGH); //3
digitalWrite(O2, LOW); //2
digitalWrite(O1, HIGH); //1
digitalWrite(O0, HIGH); //0
pressed = true;
}
if (digitalRead(AC2) == HIGH) { // 1100
digitalWrite(O3, HIGH); //3
digitalWrite(O2, HIGH); //2
digitalWrite(O1, LOW); //1
digitalWrite(O0, LOW); //0
pressed = true;
}
if (digitalRead(AC3) == HIGH) { // 1101
digitalWrite(O3, HIGH); //3
digitalWrite(O2, HIGH); //2
digitalWrite(O1, LOW); //1
digitalWrite(O0, HIGH); //0
pressed = true;
}
if ((analogRead(LATCH) > 500 ? HIGH : LOW) == HIGH) { // 1110
digitalWrite(O3, HIGH); //3
digitalWrite(O2, HIGH); //2
digitalWrite(O1, HIGH); //1
digitalWrite(O0, LOW); //0
pressed = true;
}
if (digitalRead(TWOCO) == LOW) { // 1111
digitalWrite(O3, HIGH); //3
digitalWrite(O2, HIGH); //2
digitalWrite(O1, HIGH); //1
digitalWrite(O0, HIGH); //0
pressed = true;
}
if (!pressed) { // 0000
digitalWrite(O3, LOW); //3
digitalWrite(O2, LOW); //2
digitalWrite(O1, LOW); //1
digitalWrite(O0, LOW); //0
}
}
```

Arduino Nano program

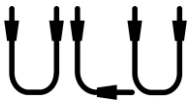
```
/* ULU.08 Adder & ALU - program code */
/* CC BY-NC-SA Jeroen Brinkman */

int i, signa, action, l, r, o, t; // integers
int rc, l0, l1, l2, l3, r0, r1, r2, r3, o0, o1, o2, o3, oc; // bits
bool latchmode, lastlatch, latch, show, twocomp; // booleans

const int conv[24][4] = {{1,0,0,0}, {1,0,0,1}, {1,0,1,0}, {1,0,1,1}, {1,1,0,0}, {1,1,0,1}, {1,1,1,0}, {1,1,1,1}, {0,0,0,0},
{0,0,0,1}, {0,0,1,0}, {0,0,1,1}, {0,1,0,0}, {0,1,0,1}, {0,1,1,0}, {0,1,1,1}, {1,0,0,0}, {1,0,0,1}, {1,0,1,0}, {1,0,1,1},
{1,1,0,0}, {1,1,0,1}, {1,1,1,0}, {1,1,1,1}}; // array

#define BOUNCE 8

#define OVER 1
#define R1 2
#define R2 3
#define R3 4
#define L0 5
#define L1 6
#define L2 7
#define L3 8
```



```
#define AC0 9
#define AC1 10
#define AC2 11
#define AC3 12
#define HEART 13
#define O3 A0
#define O2 A1
#define O1 A2
#define O0 A3
#define TWOCO A4
#define R0 A5
#define LATCH A6
#define RC A7

void setup() {
  pinMode(RC, INPUT);
  pinMode(L0, INPUT);
  pinMode(L1, INPUT);
  pinMode(L2, INPUT);
  pinMode(L3, INPUT);
  pinMode(R0, INPUT);
  pinMode(R1, INPUT);
  pinMode(R2, INPUT);
  pinMode(R3, INPUT);
  pinMode(O0, OUTPUT);
  pinMode(O1, OUTPUT);
  pinMode(O2, OUTPUT);
  pinMode(O3, OUTPUT);
  pinMode(HEART, OUTPUT); // blinking led showing the programma's heartbeat
  pinMode(OVER, OUTPUT);
  pinMode(AC0, INPUT);
  pinMode(AC1, INPUT);
  pinMode(AC2, INPUT);
  pinMode(AC3, INPUT);
  pinMode(LATCH, INPUT);
  pinMode(TWOCO, INPUT_PULLUP);

  /* Initialize variables */
  lastlatch = false; latchmode = false; o3 = 0; o2 = 0; o1 = 0; o0 = 0; oc = 0;

  /* Every ULU with an Arduino introduces itself. This one uses a pulse count */
  digitalWrite(O3, HIGH); delay (100); digitalWrite(O3, LOW); delay (200);
  digitalWrite(O3, HIGH); delay (100); digitalWrite(O3, LOW); delay (200);
  digitalWrite(O3, HIGH); delay (100); digitalWrite(O3, LOW); delay (200);
  digitalWrite(O3, HIGH); delay (100); digitalWrite(O3, LOW); delay (200);
  digitalWrite(O3, HIGH); delay (100); digitalWrite(O3, LOW); delay (200);
  digitalWrite(O3, HIGH); delay (100); digitalWrite(O3, LOW); delay (200);
  digitalWrite(O3, HIGH); delay (100); digitalWrite(O3, LOW); delay (200);
  digitalWrite(O3, HIGH); delay (100); digitalWrite(O3, LOW); delay (200);
  digitalWrite(O3, HIGH); delay (100); digitalWrite(O3, LOW); delay (200);
}

void loop(){
  digitalWrite(HEART, (millis() / 1000) % 2); //1s heartbeat for the onboard led

  /* Read all input */
  signa = 0;
  for (i = 0; i < BOUNCE; i++){signa += (analogRead(LATCH) > 500 ? 1 : 0); delay(1);} // eliminate bouncing
  latch = (signa > (i / 2));

  action = ((digitalRead(AC0) == HIGH ? 1 : 0) + (digitalRead(AC1) == HIGH ? 2 : 0) + (digitalRead(AC2) == HIGH ? 4 : 0) +
(digitalRead(AC3) == HIGH ? 8 : 0));
  twocomp = (digitalRead(TWOCO) == LOW);

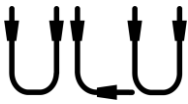
  l0 = digitalRead(L0); l1 = digitalRead(L1); l2 = digitalRead(L2); l3 = digitalRead(L3);
  r0 = digitalRead(R0); r1 = digitalRead(R1); r2 = digitalRead(R2); r3 = digitalRead(R3); rc = (analogRead(RC) > 500 ? 1 :
0);
  l = l0 + (l1 * 2) + (l2 * 4) + (l3 * 8);
  r = r0 + (r1 * 2) + (r2 * 4) + (r3 * 8);
  if (twocomp && (action != 7)){
    if (l > 7) l = l - 16;
    if (r > 7) r = r - 16;
  }

  /* Write current output */
  show = false; latchmode = latchmode || latch;
  if (!latchmode) show = true;
  if (!latch && lastlatch) show = true;
  lastlatch = latch;

  if (show){
    digitalWrite(OVER, (oc == 1) ? HIGH : LOW);
    digitalWrite(O3, (o3 == 1) ? HIGH : LOW);
    digitalWrite(O2, (o2 == 1) ? HIGH : LOW);
    digitalWrite(O1, (o1 == 1) ? HIGH : LOW);
    digitalWrite(O0, (o0 == 1) ? HIGH : LOW);
  }

  /* Process input and start the ALU */
  o = 99; oc = 0;
  switch (action) {
    case 0: // r + l + rc
      o = l + r + rc;
      break;
    case 1: // l
      o = r;
      break;
    case 2: // l=r+c, l<r+c, l>r+c, l=0
      o3 = (l = r + rc) ? 0 : 1; o2 = (l < r + rc) ? 0 : 1; o1 = (l > r + rc) ? 0 : 1; o0 = (l = 0) ? 0 : 1;
      break;
    case 3: // l - 1

```



```
        o = 1 - 1;
        break;
    case 4: // r + c - 1
        o = r + rc - 1;
        break;
    case 5: // 1 AND r
        o3 = 13 && r3; o2 = 12 && r2; o1 = 11 && r1; o0 = 10 && r0;
        break;
    case 6: // shift left r, c
        o3 = 12; o2 = 11; o1 = 10; o0 = 0;
        break;
    case 7: // 1 to 2-comp
        // has already been done
        break;
    case 8: // OR
        o3 = 13 || r3; o2 = 12 || r2; o1 = 11 || r1; o0 = 10 || r0;
        break;
    case 9: // XOR
        o3 = 13 ^ r3; o2 = 12 ^ r2; o1 = 11 ^ r1; o0 = 10 ^ r0;
        break;
    case 10: // Not 1
        o3 = (13 == 1 ? 0 : 1); o2 = (12 == 1 ? 0 : 1); o1 = (11 == 1 ? 0 : 1); o0 = (10 == 1 ? 0 : 1);
        break;
    case 11: // shift r right
        o3 = 0; o2 = 13; o1 = 12; o0 = 11;
        break;
    case 12: // 1 + 1
        o = 1 + 1;
        break;
    case 13: // rotate r left 1x When 1 is zero, rotate just once
        o3 = 12; o2 = 11; o1 = 10; o0 = 13;
        for (i = 0; i < r - 1; i++){
            t = o2; o2 = o1; o1 = o0; o0 = o3; o3 = t;
        }
        break;
    case 14: // 1 * (r + c)
        o = 1 * (r + rc);
        break;
    case 15: // rotate right a
        o = 1 - r;
        break;
    }
    if (o != 99){
        if (twocomp){
            if (o > 7) {
                o = 0; oc = 1;
            }
            if (o < -8) {
                o = 0; oc = 1;
            }
        } else {
            if (o < 0) {
                o = 0; oc = 1;
            }
            if (o > 15) {
                o = o - 16; oc = 1;
            }
        }
        o0 = conv[o+8][3]; o1 = conv[o+8][2]; o2 = conv[o+8][1]; o3 = conv[o+8][0];
    }
}
```