theWickedWebDev/8-bit-computer

# Archimedes8 CPU

## USER MANUAL

V0.1

# General Purpose Registers & Transfer Register

## General Purpose Registers

There are a total of 6 GPRS: four 8-bit ones and two 16-bit general purpose registers.

The 8-bit registers (r), are named as follows: A, C, D and E. You are able to treat some of the 8-bit registers like a 16-bit one by doing: AC, CD, or DE. Any of the 8-bit registers can be asserted to the LSB of the 16-bit bus. All registers can be transferred from one to another(s), although some may require more clock cycles to perform. This special transfer is done internally by direct r to r transfer, or by way of the Transfer Register (TX).

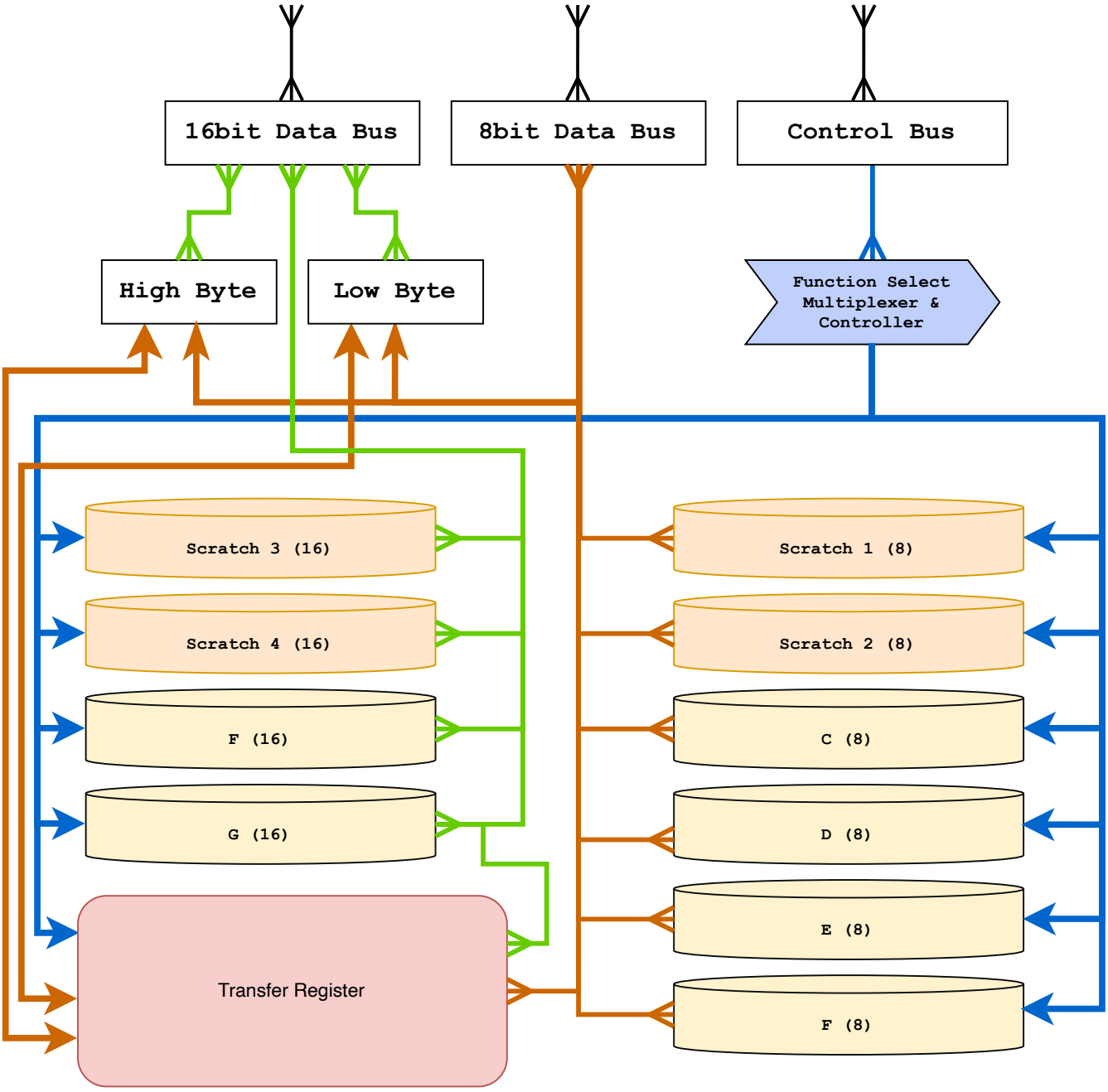
The two 16-bit registers ($r^{16}$) are F and G. The 16-bit registers include an INC and DEC function which does not set flags.

In addition to these GPRs, which are all user controlled/available. The CPU contains additional internal registers that are used to assist with the execution of instructions. These registers are as follows: $S1^{8}$, $S2^{8}$, $S3^{16}$ and $S4^{16}$. They are not available to the user.

Lastly, there are 3 8-bit output registers, designated as O1, O2, and O3, which are asserted to the “outside world” to be integrated with any way the user sees fit.

# GENERAL PURPOSE REGISTERS

Block Diagram



# 8-bit Register Loads & Assertions

| Instruction | Destination | Source |
|---|---|---|
| mov a, c | 0xf0 | 0x41 |
| mov a, d | 0xf0 | 0x42 |
| mov a, e | 0xf0 | 0x43 |
| mov a, s2 | 0xf0 | 0x45 |
| mov a, s1 | 0xf0 | 0x44 |
| mov a, txl | 0xf0 | 0x46 |
| mov a, txm | 0xf0 | 0x47 |
| mov c, a | 0xf1 | 0x40 |
| mov c, d | 0xf1 | 0x42 |
| mov c, e | 0xf1 | 0x43 |
| mov c, s1 | 0xf1 | 0x44 |
| mov c, s2 | 0xf1 | 0x45 |
| mov c, txl | 0xf1 | 0x46 |
| mov c, txm | 0xf1 | 0x47 |
| mov d, a | 0xf2 | 0x40 |
| mov d, c | 0xf2 | 0x41 |
| mov d, e | 0xf2 | 0x43 |
| mov d, s1 | 0xf2 | 0x44 |
| mov d, s2 | 0xf2 | 0x45 |
| mov d, txl | 0xf2 | 0x46 |
| mov d, txm | 0xf2 | 0x47 |
| mov e, a | 0xf3 | 0x40 |
| mov e, c | 0xf3 | 0x41 |
| mov e, d | 0xf3 | 0x42 |
| mov e, s1 | 0xf3 | 0x44 |
| mov e, s2 | 0xf3 | 0x45 |
| mov e, txl | 0xf3 | 0x46 |
| mov e, txm | 0xf3 | 0x47 |

| Instruction | Destination | Source |
|---|---|---|
| mov s1, a | 0xf4 | 0x40 |
| mov s1, c | 0xf4 | 0x41 |
| mov s1, d | 0xf4 | 0x42 |
| mov s1, e | 0xf4 | 0x43 |
| mov s1, s2 | 0xf4 | 0x45 |
| mov s1, txl | 0xf4 | 0x46 |
| mov s1, txm | 0xf4 | 0x47 |
| mov s2, a | 0xf4 | 0x40 |
| mov s2, c | 0xf4 | 0x41 |
| mov s2, d | 0xf4 | 0x42 |
| mov s2, e | 0xf4 | 0x43 |
| mov s2, s1 | 0xf4 | 0x44 |
| mov s2, txl | 0xf4 | 0x46 |
| mov s2, txm | 0xf4 | 0x47 |
| mov txl, a | 0xf6 | 0x40 |
| mov txl, c | 0xf6 | 0x41 |
| mov txl, d | 0xf6 | 0x42 |
| mov txl, e | 0xf6 | 0x43 |
| mov txl, s1 | 0xf6 | 0x44 |
| mov txl, s2 | 0xf6 | 0x45 |
| mov txm, a | 0xf7 | 0x40 |
| mov txm, c | 0xf7 | 0x41 |
| mov txm, d | 0xf7 | 0x42 |
| mov txm, e | 0xf7 | 0x43 |
| mov txm, s1 | 0xf7 | 0x44 |
| mov txm, s2 | 0xf7 | 0x45 |

| Instruction | Destination | Source |
|---|---|---|
| mov a, * | 0xf0 | 0x0 |
| mov c, * | 0xf1 | 0x0 |
| mov d, * | 0xf2 | 0x0 |
| mov e, * | 0xf3 | 0x0 |
| mov s1, * | 0xf4 | 0x0 |
| mov s2, * | 0xf5 | 0x0 |
| mov txl, * | 0xf6 | 0x0 |
| mov txh, * | 0xf7 | 0x0 |

## 16-bit GPR to GPR

| Instruction | Destination | Source |
|---|---|---|
| mov f, g | 0xf | 0x81 |
| mov f, s3 | 0xf | 0x82 |
| mov f, s4 | 0xf | 0x83 |
| mov f, tx | 0xf | 0x48 |
| mov g, f | 0x1f | 0x80 |
| mov g, s3 | 0x1f | 0x82 |
| mov g, s4 | 0x1f | 0x83 |
| mov g, tx | 0x1f | 0x48 |
| mov s3, f | 0x2f | 0x80 |
| mov s3, g | 0x2f | 0x81 |
| mov s3, s4 | 0x2f | 0x83 |
| mov s3, tx | 0x2f | 0x48 |
| mov s4, f | 0x3f | 0x80 |
| mov s4, g | 0x3f | 0x81 |
| mov s4, s3 | 0x3f | 0x82 |
| mov s4, tx | 0x3f | 0x48 |
| mov f, ac | 0xf | 0xca |
| mov f, ad | 0xf | 0xcb |
| mov f, ae | 0xf | 0xcc |
| mov f, as1 | 0xf | 0xcd |
| mov f, as2 | 0xf | 0xce |
| mov f, ca | 0xf | 0xd1 |
| mov f, cd | 0xf | 0xd3 |
| mov f, ce | 0xf | 0xd4 |
| mov f, cs1 | 0xf | 0xd5 |
| mov f, cs2 | 0xf | 0xd6 |
| mov f, da | 0xf | 0xd9 |
| mov f, dc | 0xf | 0xdb |
| mov f, de | 0xf | 0xdc |
| mov f, ds1 | 0xf | 0xdd |
| mov f, ds2 | 0xf | 0xde |
| mov f, ea | 0xf | 0xe1 |
| mov f, ec | 0xf | 0xe3 |
| mov f, ed | 0xf | 0xe4 |
| mov f, es1 | 0xf | 0xe5 |
| mov f, es2 | 0xf | 0xe6 |
| mov f, s1a | 0xf | 0xe9 |
| mov f, s1c | 0xf | 0xeb |
| mov f, s1d | 0xf | 0xec |
| mov f, s1e | 0xf | 0xed |
| mov f, s1s2 | 0xf | 0xee |
| mov f, s2a | 0xf | 0xf1 |
| mov f, s2c | 0xf | 0xf3 |
| mov f, s2d | 0xf | 0xf4 |
| mov f, s2e | 0xf | 0xf5 |
| mov f, s2s1 | 0xf | 0xf6 |
| mov g, ac | 0x1f | 0xca |
| mov g, ad | 0x1f | 0xcb |
| mov g, ae | 0x1f | 0xcc |
| mov g, as1 | 0x1f | 0xcd |
| mov g, as2 | 0x1f | 0xce |
| mov g, ca | 0x1f | 0xd1 |
| mov g, cd | 0x1f | 0xd3 |
| mov g, ce | 0x1f | 0xd4 |
| mov g, cs1 | 0x1f | 0xd5 |
| mov g, cs2 | 0x1f | 0xd6 |
| mov g, da | 0x1f | 0xd9 |
| mov g, dc | 0x1f | 0xdb |
| mov g, de | 0x1f | 0xdc |
| mov g, ds1 | 0x1f | 0xdd |
| mov g, ds2 | 0x1f | 0xde |
| mov g, ea | 0x1f | 0xe1 |
| mov g, ec | 0x1f | 0xe3 |
| mov g, ed | 0x1f | 0xe4 |
| mov g, es1 | 0x1f | 0xe5 |
| mov g, es2 | 0x1f | 0xe6 |
| mov g, s1a | 0x1f | 0xe9 |
| mov g, s1c | 0x1f | 0xeb |
| mov g, s1d | 0x1f | 0xec |
| mov g, s1e | 0x1f | 0xed |
| mov g, s1s2 | 0x1f | 0xee |
| mov g, s2a | 0x1f | 0xf1 |
| mov g, s2c | 0x1f | 0xf3 |
| mov g, s2d | 0x1f | 0xf4 |
| mov g, s2e | 0x1f | 0xf5 |
| mov g, s2s1 | 0x1f | 0xf6 |

| Instruction | Destination | Source |
|---|---|---|
| mov s3, ac | 0x2f | 0xca |
| mov s3, ad | 0x2f | 0xcb |
| mov s3, ae | 0x2f | 0xcc |
| mov s3, as1 | 0x2f | 0xcd |
| mov s3, as2 | 0x2f | 0xce |
| mov s3, ca | 0x2f | 0xd1 |
| mov s3, cd | 0x2f | 0xd3 |
| mov s3, ce | 0x2f | 0xd4 |
| mov s3, cs1 | 0x2f | 0xd5 |
| mov s3, cs2 | 0x2f | 0xd6 |
| mov s3, da | 0x2f | 0xd9 |
| mov s3, dc | 0x2f | 0xdb |
| mov s3, de | 0x2f | 0xdc |
| mov s3, ds1 | 0x2f | 0xdd |
| mov s3, ds2 | 0x2f | 0xde |
| mov s3, ea | 0x2f | 0xe1 |
| mov s3, ec | 0x2f | 0xe3 |
| mov s3, ed | 0x2f | 0xe4 |
| mov s3, es1 | 0x2f | 0xe5 |
| mov s3, es2 | 0x2f | 0xe6 |
| mov s3, s1a | 0x2f | 0xe9 |
| mov s3, s1c | 0x2f | 0xeb |
| mov s3, s1d | 0x2f | 0xec |
| mov s3, s1e | 0x2f | 0xed |
| mov s3, s1s2 | 0x2f | 0xee |
| mov s3, s2a | 0x2f | 0xf1 |
| mov s3, s2c | 0x2f | 0xf3 |
| mov s3, s2d | 0x2f | 0xf4 |
| mov s3, s2e | 0x2f | 0xf5 |
| mov s3, s2s1 | 0x2f | 0xf6 |
| mov s4, ac | 0x3f | 0xca |
| mov s4, ad | 0x3f | 0xcb |
| mov s4, ae | 0x3f | 0xcc |
| mov s4, as1 | 0x3f | 0xcd |
| mov s4, as2 | 0x3f | 0xce |
| mov s4, ca | 0x3f | 0xd1 |
| mov s4, cd | 0x3f | 0xd3 |
| mov s4, ce | 0x3f | 0xd4 |
| mov s4, cs1 | 0x3f | 0xd5 |
| mov s4, cs2 | 0x3f | 0xd6 |
| mov s4, da | 0x3f | 0xd9 |
| mov s4, dc | 0x3f | 0xdb |
| mov s4, de | 0x3f | 0xdc |
| mov s4, ds1 | 0x3f | 0xdd |
| mov s4, ds2 | 0x3f | 0xde |
| mov s4, ea | 0x3f | 0xe1 |
| mov s4, ec | 0x3f | 0xe3 |
| mov s4, ed | 0x3f | 0xe4 |
| mov s4, es1 | 0x3f | 0xe5 |
| mov s4, es2 | 0x3f | 0xe6 |
| mov s4, s1a | 0x3f | 0xe9 |
| mov s4, s1c | 0x3f | 0xeb |
| mov s4, s1d | 0x3f | 0xec |
| mov s4, s1e | 0x3f | 0xed |
| mov s4, s1s2 | 0x3f | 0xee |
| mov s4, s2a | 0x3f | 0xf1 |
| mov s4, s2c | 0x3f | 0xf3 |
| mov s4, s2d | 0x3f | 0xf4 |
| mov s4, s2e | 0x3f | 0xf5 |
| mov s4, s2s1 | 0x3f | 0xf6 |
| mov tx, ac | 0xf8 | 0xca |
| mov tx, ad | 0xf8 | 0xcb |
| mov tx, ae | 0xf8 | 0xcc |
| mov tx, as1 | 0xf8 | 0xcd |
| mov tx, as2 | 0xf8 | 0xce |
| mov tx, ca | 0xf8 | 0xd1 |
| mov tx, cd | 0xf8 | 0xd3 |
| mov tx, ce | 0xf8 | 0xd4 |
| mov tx, cs1 | 0xf8 | 0xd5 |
| mov tx, cs2 | 0xf8 | 0xd6 |
| mov tx, da | 0xf8 | 0xd9 |
| mov tx, dc | 0xf8 | 0xdb |
| mov tx, de | 0xf8 | 0xdc |
| mov tx, ds1 | 0xf8 | 0xdd |
| mov tx, ds2 | 0xf8 | 0xde |
| mov tx, ea | 0xf8 | 0xe1 |

| Instruction | Destination | Source |
|---|---|---|
| mov tx, ec | 0xf8 | 0xe3 |
| mov tx, ed | 0xf8 | 0xe4 |
| mov tx, es1 | 0xf8 | 0xe5 |
| mov tx, es2 | 0xf8 | 0xe6 |
| mov tx, s1a | 0xf8 | 0xe9 |
| mov tx, s1c | 0xf8 | 0xeb |
| mov tx, s1d | 0xf8 | 0xec |
| mov tx, s1e | 0xf8 | 0xed |
| mov tx, s1s2 | 0xf8 | 0xee |
| mov tx, s2a | 0xf8 | 0xf1 |
| mov tx, s2c | 0xf8 | 0xf3 |
| mov tx, s2d | 0xf8 | 0xf4 |
| mov tx, s2e | 0xf8 | 0xf5 |
| mov tx, s2s1 | 0xf8 | 0xf6 |
| mov g, cs1 | 0x1f | 0xd5 |
| mov g, cs2 | 0x1f | 0xd6 |
| mov g, da | 0x1f | 0xd9 |
| mov g, dc | 0x1f | 0xdb |
| mov g, de | 0x1f | 0xdc |
| mov g, ds1 | 0x1f | 0xdd |
| mov g, ds2 | 0x1f | 0xde |
| mov g, ea | 0x1f | 0xe1 |
| mov g, ec | 0x1f | 0xe3 |
| mov g, ed | 0x1f | 0xe4 |
| mov g, es1 | 0x1f | 0xe5 |
| mov g, es2 | 0x1f | 0xe6 |
| mov g, s1a | 0x1f | 0xe9 |
| mov g, s1c | 0x1f | 0xeb |
| mov g, s1d | 0x1f | 0xec |
| mov g, s1e | 0x1f | 0xed |
| mov g, s1s2 | 0x1f | 0xee |
| mov g, s2a | 0x1f | 0xf1 |
| mov g, s2c | 0x1f | 0xf3 |
| mov g, s2d | 0x1f | 0xf4 |
| mov g, s2e | 0x1f | 0xf5 |
| mov g, s2s1 | 0x1f | 0xf6 |
| mov s3, ac | 0x2f | 0xca |
| mov s3, ad | 0x2f | 0xcb |
| mov s3, ae | 0x2f | 0xcc |
| mov s3, as1 | 0x2f | 0xcd |
| mov s3, as2 | 0x2f | 0xce |
| mov s3, ca | 0x2f | 0xd1 |
| mov s3, cd | 0x2f | 0xd3 |
| mov s3, ce | 0x2f | 0xd4 |
| mov s3, cs1 | 0x2f | 0xd5 |
| mov s3, cs2 | 0x2f | 0xd6 |
| mov s3, da | 0x2f | 0xd9 |
| mov s3, dc | 0x2f | 0xdb |
| mov s3, de | 0x2f | 0xdc |
| mov s3, ds1 | 0x2f | 0xdd |
| mov s3, ds2 | 0x2f | 0xde |
| mov s3, ea | 0x2f | 0xe1 |
| mov s3, ec | 0x2f | 0xe3 |
| mov s3, ed | 0x2f | 0xe4 |
| mov s3, es1 | 0x2f | 0xe5 |
| mov s3, es2 | 0x2f | 0xe6 |
| mov s3, s1a | 0x2f | 0xe9 |
| mov s3, s1c | 0x2f | 0xeb |
| mov s3, s1d | 0x2f | 0xec |
| mov s3, s1e | 0x2f | 0xed |
| mov s3, s1s2 | 0x2f | 0xee |
| mov s3, s2a | 0x2f | 0xf1 |
| mov s3, s2c | 0x2f | 0xf3 |
| mov s3, s2d | 0x2f | 0xf4 |
| mov s3, s2e | 0x2f | 0xf5 |
| mov s3, s2s1 | 0x2f | 0xf6 |
| mov s4, ac | 0x3f | 0xca |
| mov s4, ad | 0x3f | 0xcb |
| mov s4, ae | 0x3f | 0xcc |
| mov s4, as1 | 0x3f | 0xcd |
| mov s4, as2 | 0x3f | 0xce |
| mov s4, ca | 0x3f | 0xd1 |
| mov s4, cd | 0x3f | 0xd3 |
| mov s4, ce | 0x3f | 0xd4 |
| mov s4, cs1 | 0x3f | 0xd5 |
| mov s4, cs2 | 0x3f | 0xd6 |

| Instruction | Destination | Source |
|---|---|---|
| mov s4, da | 0x3f | 0xd9 |
| mov s4, dc | 0x3f | 0xdb |
| mov s4, de | 0x3f | 0xdc |
| mov s4, ds1 | 0x3f | 0xdd |
| mov s4, ds2 | 0x3f | 0xde |
| mov s4, ea | 0x3f | 0xe1 |
| mov s4, ec | 0x3f | 0xe3 |
| mov s4, ed | 0x3f | 0xe4 |
| mov s4, es1 | 0x3f | 0xe5 |
| mov s4, es2 | 0x3f | 0xe6 |
| mov s4, s1a | 0x3f | 0xe9 |
| mov s4, s1c | 0x3f | 0xeb |
| mov s4, s1d | 0x3f | 0xec |
| mov s4, s1e | 0x3f | 0xed |
| mov s4, s1s2 | 0x3f | 0xee |
| mov s4, s2a | 0x3f | 0xf1 |
| mov s4, s2c | 0x3f | 0xf3 |
| mov s4, s2d | 0x3f | 0xf4 |
| mov s4, s2e | 0x3f | 0xf5 |
| mov s4, s2s1 | 0x3f | 0xf6 |
| mov tx, ac | 0xf8 | 0xca |
| mov tx, ad | 0xf8 | 0xcb |
| mov tx, ae | 0xf8 | 0xcc |
| mov tx, as1 | 0xf8 | 0xcd |
| mov tx, as2 | 0xf8 | 0xce |
| mov tx, ca | 0xf8 | 0xd1 |
| mov tx, cd | 0xf8 | 0xd3 |
| mov tx, ce | 0xf8 | 0xd4 |
| mov tx, cs1 | 0xf8 | 0xd5 |
| mov tx, cs2 | 0xf8 | 0xd6 |
| mov tx, da | 0xf8 | 0xd9 |
| mov tx, dc | 0xf8 | 0xdb |
| mov tx, de | 0xf8 | 0xdc |
| mov tx, ds1 | 0xf8 | 0xdd |
| mov tx, ds2 | 0xf8 | 0xde |
| mov tx, ea | 0xf8 | 0xe1 |
| mov tx, ec | 0xf8 | 0xe3 |
| mov tx, ed | 0xf8 | 0xe4 |
| mov tx, es1 | 0xf8 | 0xe5 |
| mov tx, es2 | 0xf8 | 0xe6 |
| mov tx, s1a | 0xf8 | 0xe9 |
| mov tx, s1c | 0xf8 | 0xeb |
| mov tx, s1d | 0xf8 | 0xec |
| mov tx, s1e | 0xf8 | 0xed |
| mov tx, s1s2 | 0xf8 | 0xee |
| mov tx, s2a | 0xf8 | 0xf1 |
| mov tx, s2c | 0xf8 | 0xf3 |
| mov tx, s2d | 0xf8 | 0xf4 |
| mov tx, s2e | 0xf8 | 0xf5 |
| mov tx, s2s1 | 0xf8 | 0xf6 |

## GPR Increment / Decrement

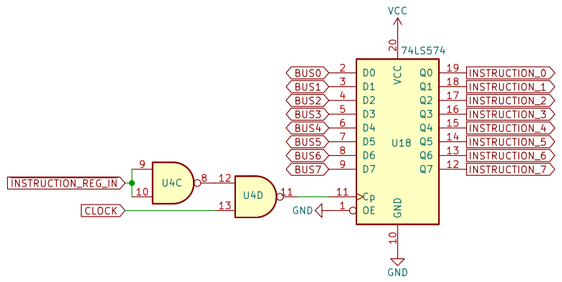
| Instruction | Destination | Source |
|---|---|---|
| inc f | 0x9f | 0x0 |
| inc g | 0xbf | 0x0 |
| inc s3 | 0x5f | 0x0 |
| inc s4 | 0x7f | 0x0 |
| dec f | 0x8f | 0x0 |
| dec g | 0xaf | 0x0 |
| dec s3 | 0x4f | 0x0 |
| dec s4 | 0x6f | 0x0 |

## 16-bit Register Loads*

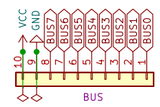
| Instruction | Destination | Source |
|---|---|---|
| mov f, * | 0xf | 0x0 |
| mov g, * | 0x1f | 0x0 |
| mov s3, * | 0x2f | 0x0 |
| mov s4, * | 0x3f | 0x0 |
| mov tx, * | 0xf8 | 0x0 |

# CONTROL LOGIC UNIT

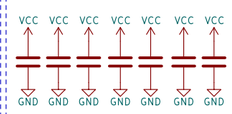
## Instruction Register



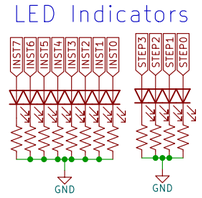

## BUS CONNECTION



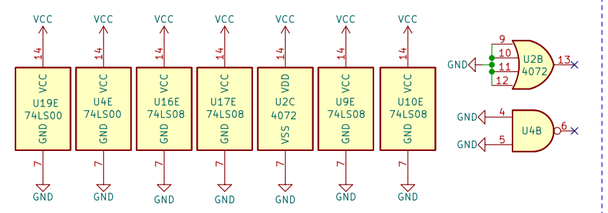

## Smoothing Caps



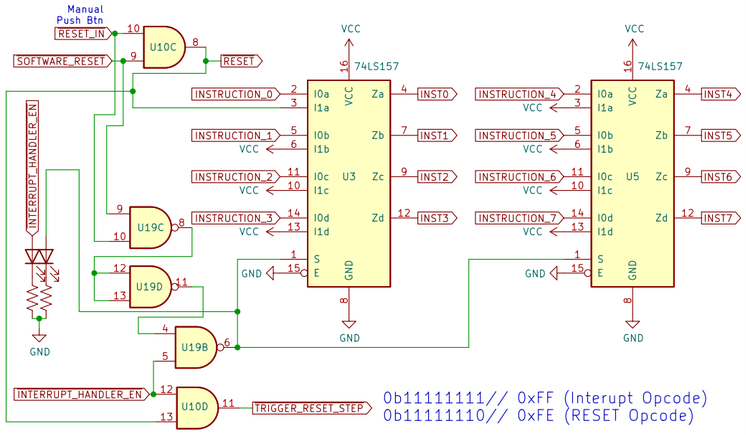

## LED Indicators




## Logic Gate Power

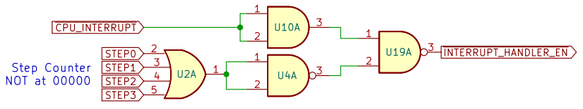



## INSTRUCTION ADDRESS MUX



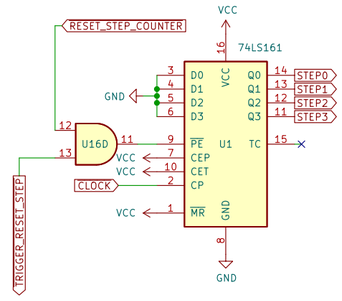

## ENABLE INTERRUPT HANDLER REQUEST

True when following conditions are met:
1. Step Counter is at 0x0
2. At least 1 interrupt is requested
3. The current instruction is not an enter interrupt one.



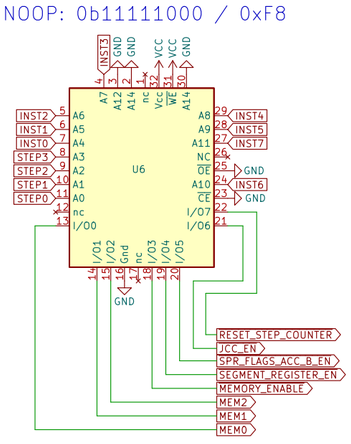

## STEP COUNTER

Each instruction can have up to 16 steps



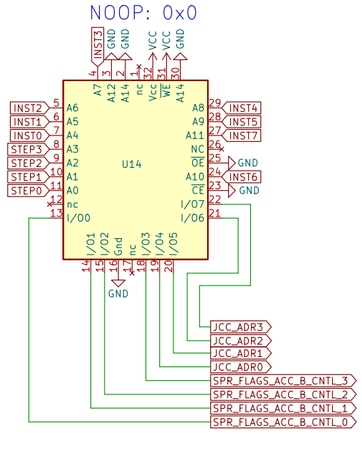

## Memory / SPR JCC / RST STEP

NOOP: 0b11111000 / 0xF8



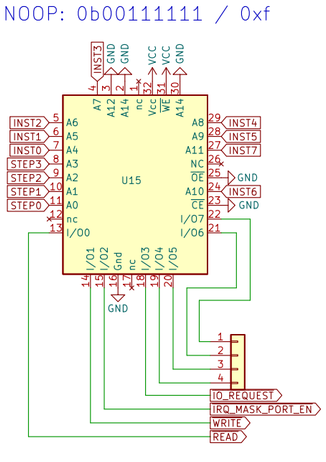

## SPR / FLAGS OPERAND / JCC

NOOP: 0x0



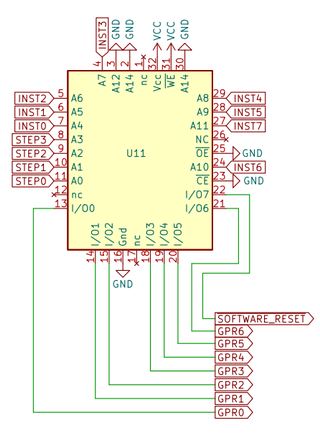

## I/O / INTERRUPT

NOOP: 0b00111111 / 0xf



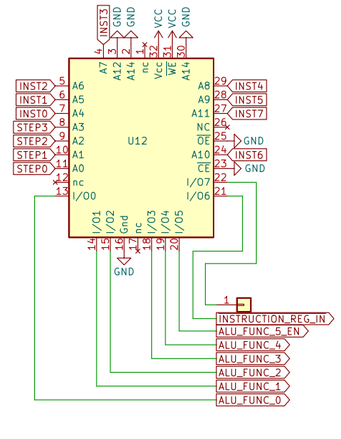

## GPR & RESET

NOOP: 0b10000000 / 0x80

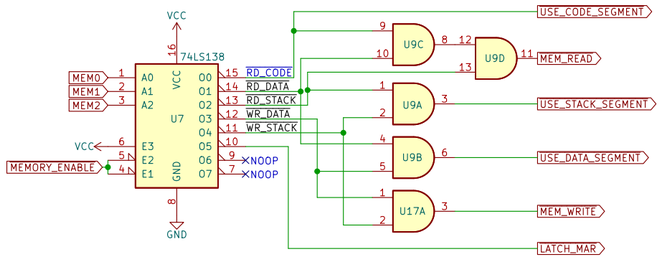



## ALU & INST REG

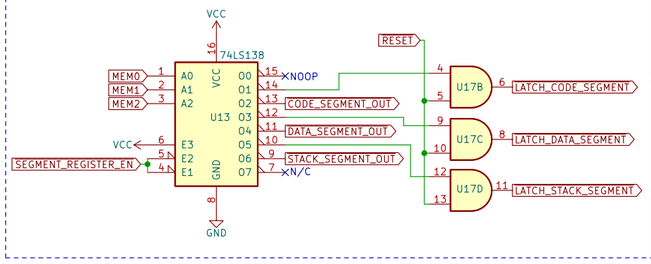
NOOP: 0b01000000 / 0x40



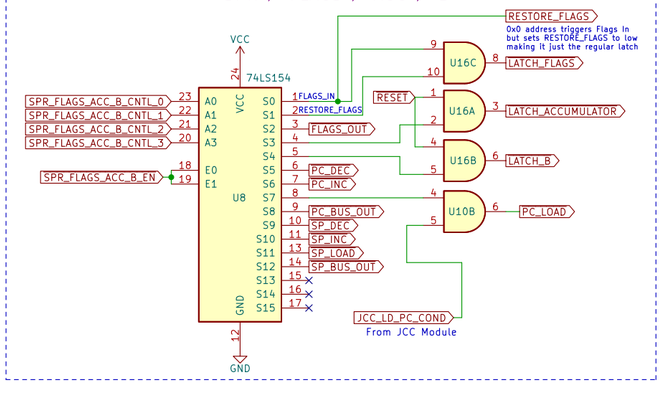

## READ / WRITE MEMORY DECODE LOGIC



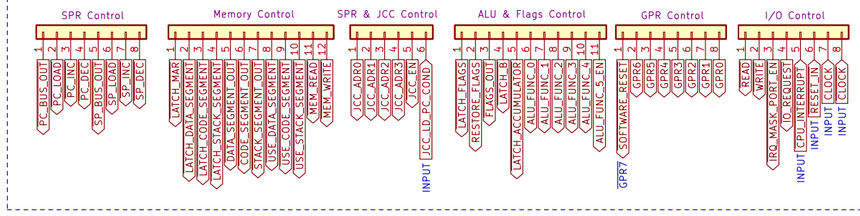

## READ / WRITE SEGMENT DECODE LOGIC




## SPR / FLAGS / ACC / B




## VARIOUS CONTROL LINES




Interrupt Request Logic
Operation Counter
Instruction Register
EEPROM Microcode
**theWickedWebDev/8-bit-computer**

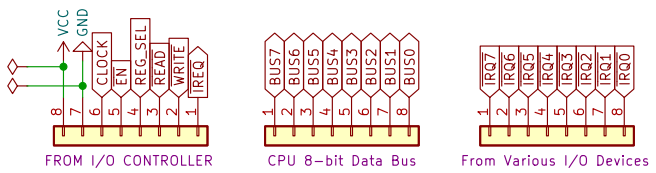
Sheet: /
File: control-unit.kicad_sch

**Title: Control Logic Unit**

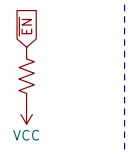
| Size: User | Date: 2022-02-10 | **Rev: v6** |
|---|---|---|
| KiCad E.D.A. eeschema (6.0.0-0) | | Id: 1/1 |

# INTERRUPT REQUEST HANDLER

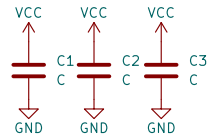
## Connections



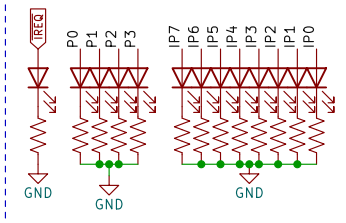

## PULL U/D



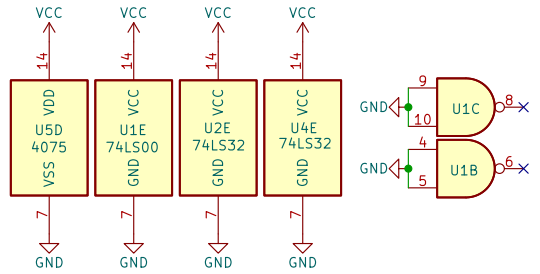

## Smoothing Caps



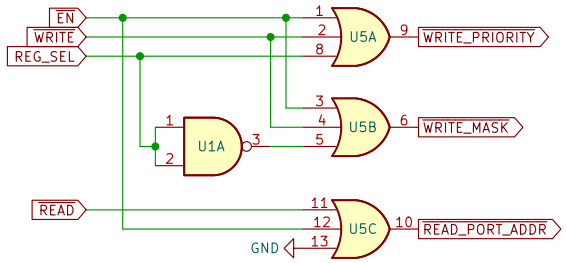

## LED Indicators




## Logic Gate Power



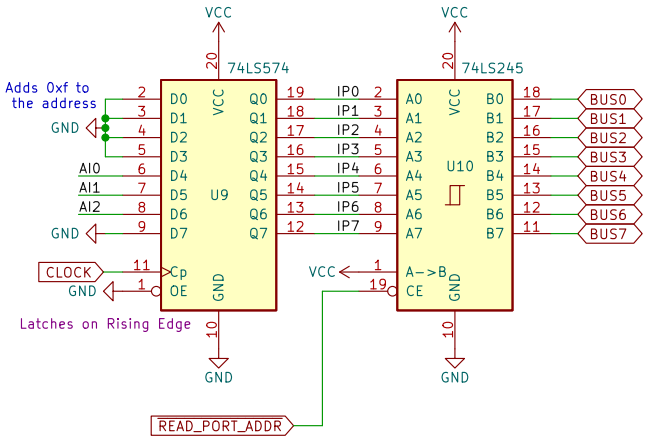

## Decode Logic




## Interrupt Latch

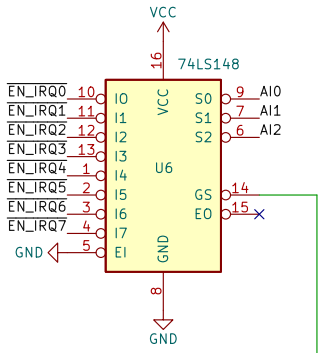
This latch ensures that when a port address is read, it is synchonrous with the clock, in case there is a change in interrupts at the same moment,which would cause an incorrect ISR to be run

### Read Port Address



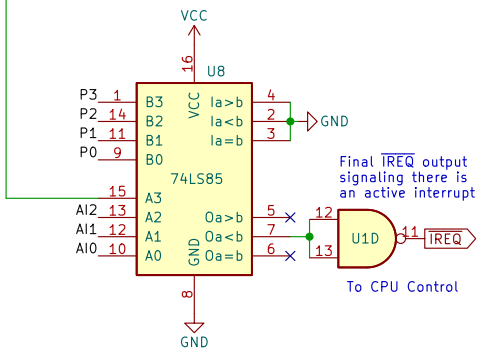

## 8-to-3 Priority Decoder

Active LOW (inputs & outputs)




### Comparator

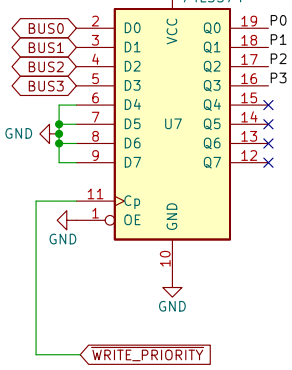
Request Interrupt if IRQn > Priority Level




## Priority Level Register

Latches on Rising Edge

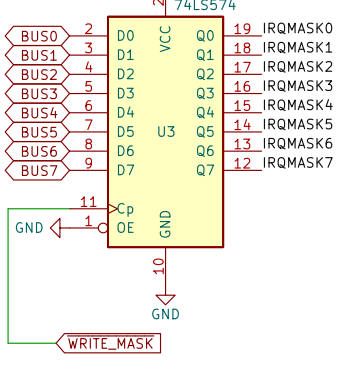
PORT_HNDLR_EN and IRQ_HANDLER_WRITE must both be LOW to latch




## Interrupt Mask

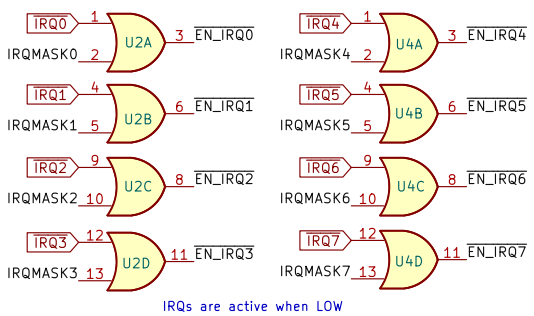
Latches on Rising Edge

IRQ_MASK_WRITE and IRQ_HANDLER_WRITE must both be LOW to latch




## Disables Interrupts via Masks

When IRQMASK[*] is set HIGH, the corresponding interrupt request is disabled




IRQs are active when LOW

## MASK LOGIC

Active LOW IRQn
Active HIGH MASKn

If either of these are HIGH then then that interrupt is disabled/inactive

IREQ's are OR'ed with the mask bits

## IRQ Example Assembly

```
IRQPORT: 0x0
IRQMASKPORT: 0x1

SETUP_IRQ:
    # Enables two IREQs
    OUT IRQMASKPORT, 0b0111111

    # Sets priority level to all
    OUT IRQPORT, 0x8

IRQ:
    PUSH CS
    IN S1, IRQPORT
    JMPF (0:s1)
```

## Interrupt Vector Table

$0:00f0: IRQ0
$0:00f1: IRQ1
$0:00f2: IRQ2
$0:00f3: IRQ3
$0:00f4: IRQ4
$0:00f5: IRQ5
$0:00f6: IRQ6
$0:00f7: IRQ7

## PRIORITY LOGIC

IRQ7 – Highest Priority
IRQ0 – Lowest Priority

If the curent Interrupt address from the Priority Decoder output has a priority higher than the Priority Level, then an IREQ is triggered

*Priority Examples*

A value of 8 written to the priority latch will enable all interrupts.
A value of 5 written to the priority latch will enable IQR0 – IRQ4. IRQ5 – IRQ7 will be disabled.
A value of 3 written to the priority latch will enable IQR0 – IRQ2. IRQ3 – IRQ7 will be disabled.
A value of 0 written to the priority latch will disable all interrupts.

## Pinout Description

IREQ – (output) Signals to the CPU there is an interrupt

WRITE – (input) Trigers a latch on a register. Used in conjunction with INT_PORT_REQ and either PORT_HNDLR_EN or IRQ_MASK_WRITE

READ – (input) Triggers a latch on the Interrupt Port Register and asserts its value onto the bus. Only when PORT_HNDLR_EN is LOW.

ADDR[4,5] – (input) Used to select Mask or Priority registers

INT_PORT_EN – (input) Enable all functionality on LOW signal

w/ Priority Level Register & Mask Register
**theWickedWebDev/8-bit-computer**

Sheet: /
File: new-interrupt-handler.kicad_sch

***Title: Interrpt Request Handler Module***

| | | |
|---|---|---|
| Size: User | Date: 2022-02-10 | **Rev: v6** |
| KiCad E.D.A. eeschema (6.0.0-0) | | Id: 1/1 |

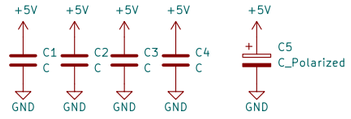
# MEMORY MODULE

RAM & ROM w/ battery backup power

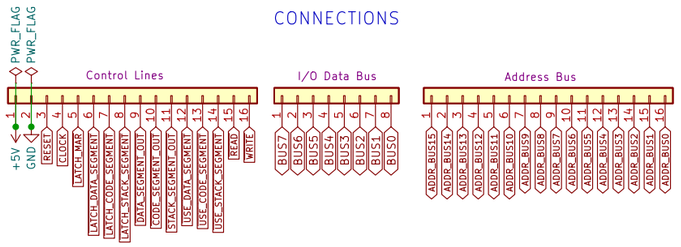
32K ROM | 1M RAM

ROM $0000 – $7FFF
RAM $8000 – $FFFFF

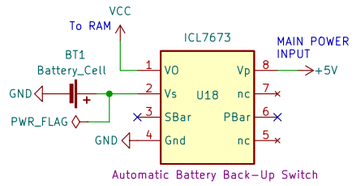
## Smoothing Cap's



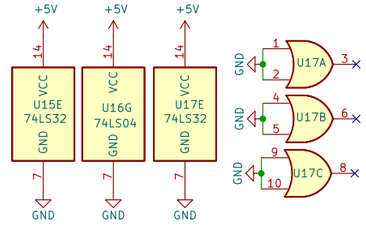

## CONNECTIONS



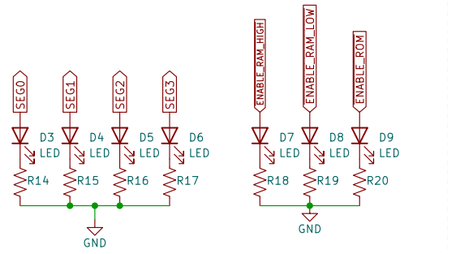

## Backup Battery Power Switching




## Logic Gate Power and Unused

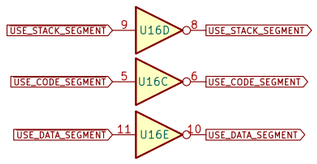



## LED INDICATORS

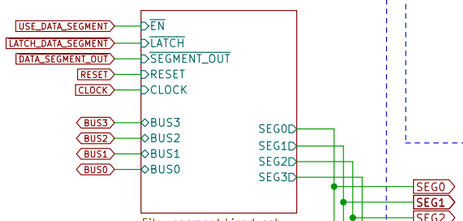



## MEMORY SEGMENTS

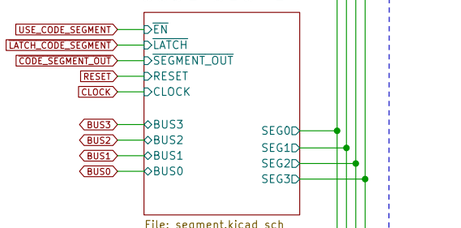
Make active LOW



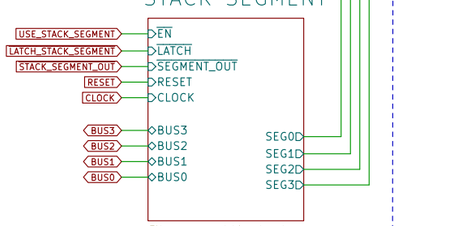

### DATA SEGMENT




### CODE SEGMENT



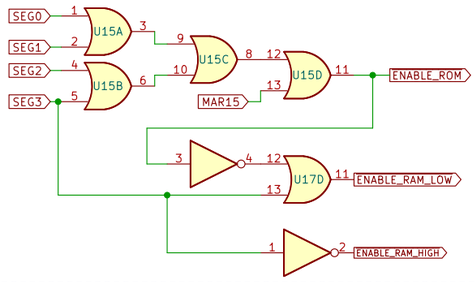

### STACK SEGMENT



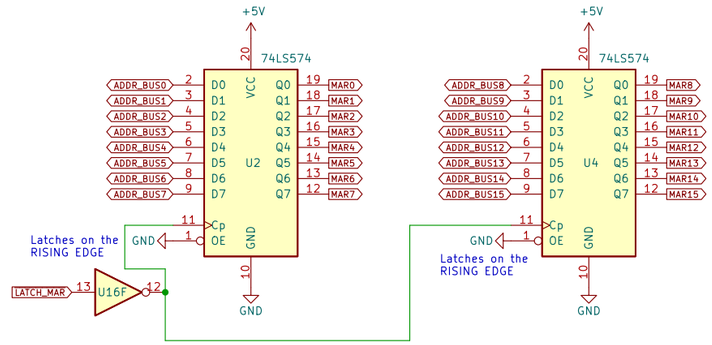

## MEMORY CHIP ENABLE LOGIC

Logic Explaination

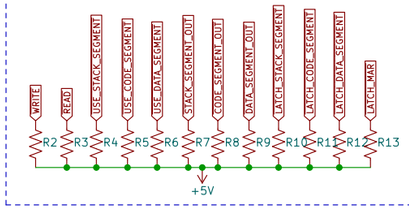
ALL LOW inputs, ROM Enable
SEG3 LOW, At least one HIGH: RAM Enable
SEG3 HIGH, rest dont care: RAM Enable HIGH



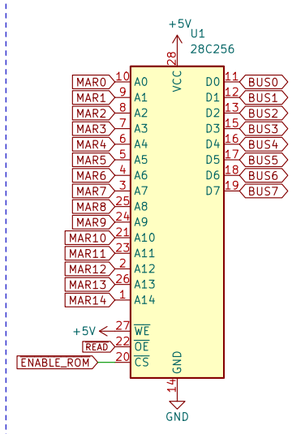

## MEMORY ADDRESS REGISTER



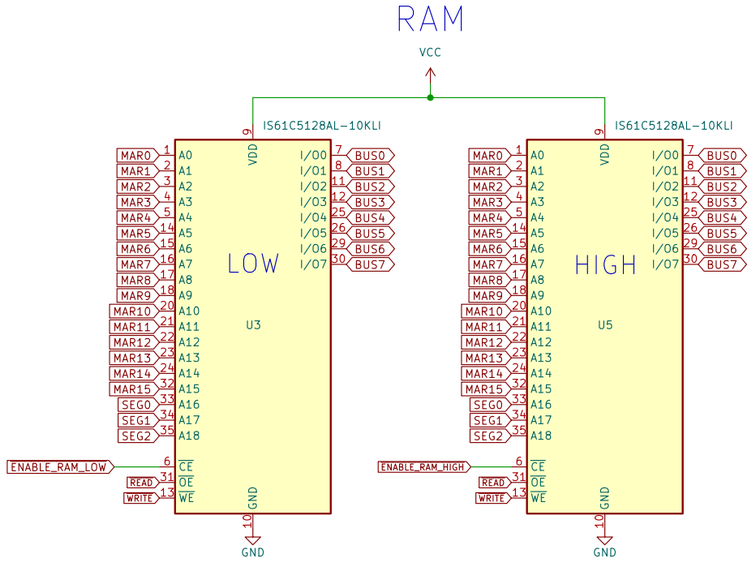

## PULL UPS




## ROM




## RAM



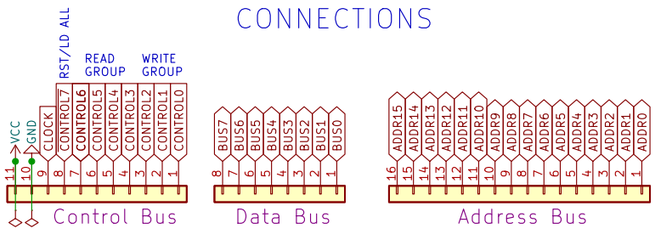

Data preserved with backup battery
**theWickedWebDev/8-bit-computer**

| | | |
|---|---|---|
| Sheet: / | | |
| File: RAM-MODULE.kicad_sch | | |
| ***Title: Memory Module*** | | |
| Size: A3 | Date: 2021-12-02 | **Rev: 3** |
| KiCad E.D.A. eeschema (6.0.0-0) | | Id: 1/4 |

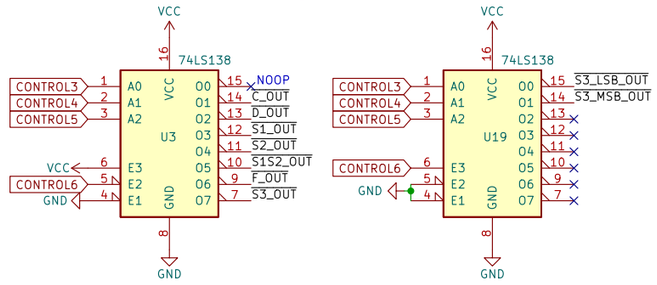
# GENERAL PURPOSE REGISTER ARRAY

## 8bit & 16bit $C^8$, $D^8$, $S1^8$, $S2^8$, $S1S2^{16}$, $F^{16}$, $S3^{16}$

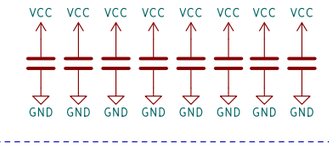
### CONNECTIONS



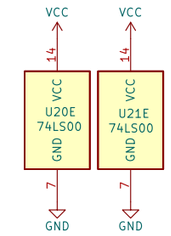

### Read Register Decoder



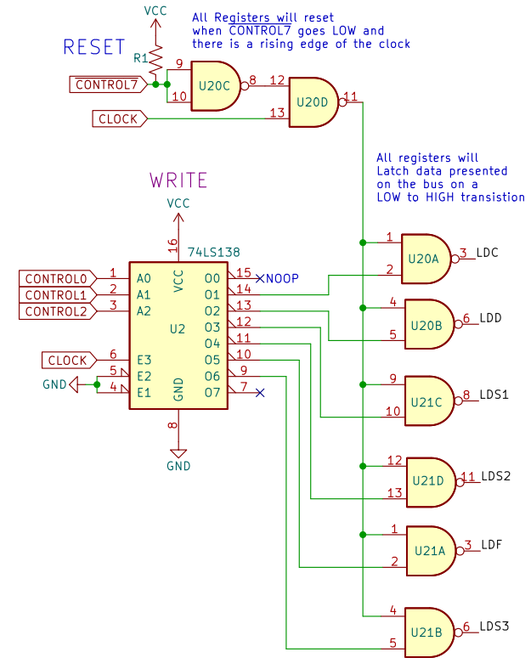

### Smoothing Caps



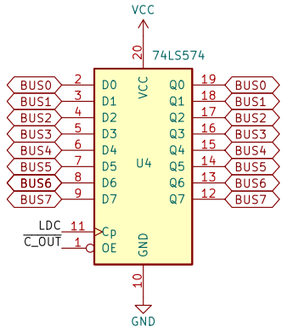

### Logic Gate Power



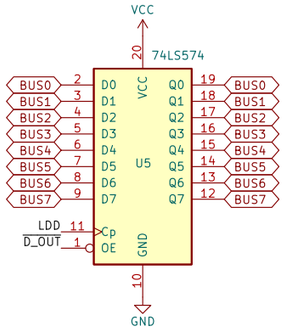

### WRITE & RESET MUX



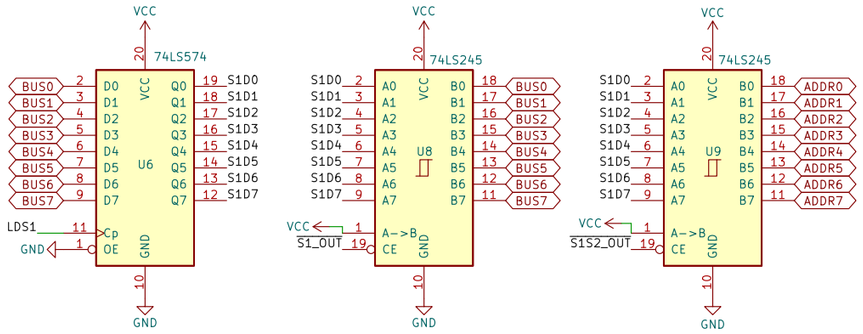

### C Register



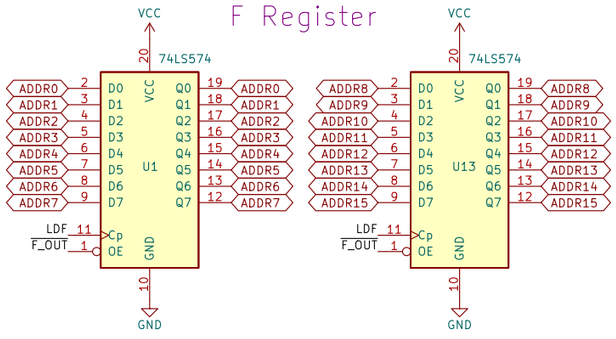

### D Register



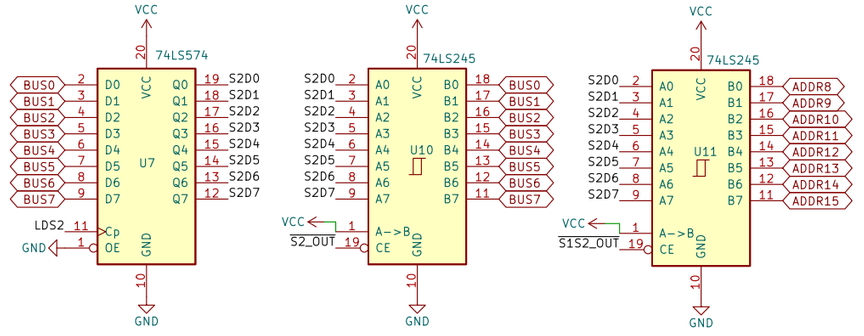

### S1 Register



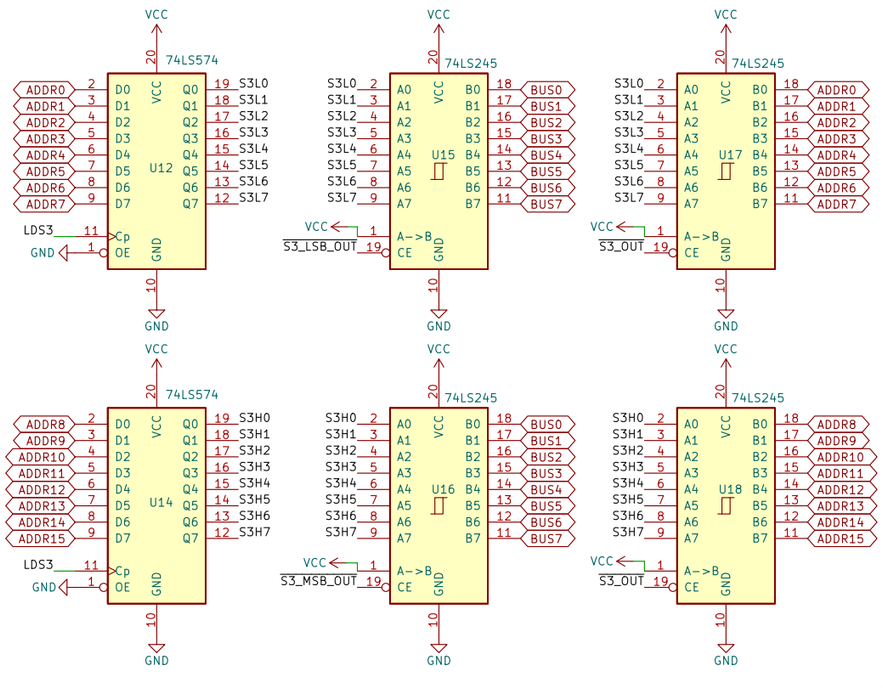

### F Register




### S2 Register




### S3 Register




### GPR <–> External

**MOV R, ***

0x81: LDC
0x82: LDD
0x83: LDS1
0x84: LDS2
0x85: LDF
0x86: LDS3

**MOV *, R**

0x80: NOOP
0x88: CO
0x90: DO
0x98: S1OB
0xa0: S2OB
0xa8: S1S2OA
0xb0: FO
0xb8: S3O
0xc0: S3L0B
0xc8: S3H0B

### GPR to GPR Control Words

MOV C, D – 0x91 (0b1_0010_001)
MOV C, S1 – 0x99 (0b1_0011_001)
MOV C, S2 – 0xa1 (0b1_0100_001)
MOV C, S3LSB – 0xc1 (0b1_1000_001)
MOV C, S3MSB – 0xc9 (0b1_1001_001)
MOV D, C – 0x8a (0b1_0001_010)
MOV D, S1 – 0x9a (0b1_0011_010)
MOV D, S2 – 0xa2 (0b1_0100_010)
MOV D, S3LSB – 0xc2 (0b1_1000_010)
MOV D, S3MSB – 0xca (0b1_1001_010)
MOV S1, C – 0x8b (0b1_0001_011)
MOV S1, D – 0x9a (0b1_0010_011)
MOV S1, S2 – 0x93 (0b1_0100_011)
MOV S1, S3LSB – 0xc3 (0b1_1000_011)
MOV S1, S3MSB – 0xcb (0b1_1001_011)
MOV S2, C – 0x8c (0b1_0001_100)
MOV S2, D – 0x94 (0b1_0010_100)
MOV S2, S1 – 0x9c (0b1_0011_100)
MOV S2, S3LSB – 0xc4 (0b1_1000_100)
MOV S2, S3MSB – 0xcc (0b1_1001_100)
MOV F, S3 – 0xbd (0b1_0111_101)
MOV F, S1S2 – 0xad (0b1_0101_101)
MOV S3, F – 0xb6 (0b1_0110_110)
MOV S3, S1S2 – 0xae (0b1_0101_110)

### Control Word Truth Table

To just ASSERT or LOAD a value to the data bus to be used with external modules, just select a single value from either READ or WRITE

To do a GPR to GPR transfer just select a READ and WRITE control word and then OR them together.

To do a RESET, $\overline{\text{CONTROL7}}$ is set LOW and all data buses are set to 0x0, which causes each register to load the value presented on their data bus.


**theWickedWebDev/8-bit-computer**

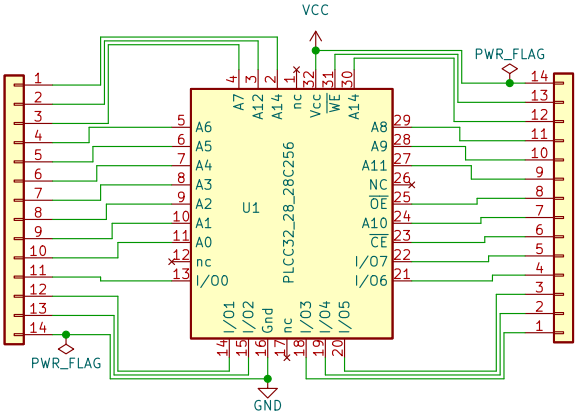
Sheet: /
File: Minified-Gpr.kicad_sch

***Title: General Purpose Register Array***

| Size: User | Date: 2022-02-10 | **Rev: v2** |
|---|---|---|
| KiCad E.D.A. eeschema (6.0.0-0) | | Id: 1/1 |

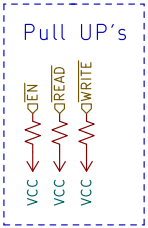
# PLCC to THT Adapter

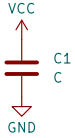
Allows PLCC EEPROM's to be programmed with
a generic IC Programmer (THT)

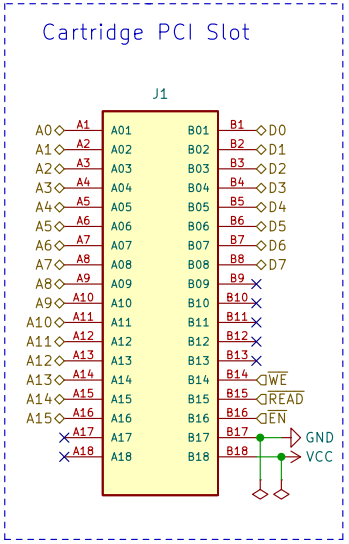



| | | |
|---|---|---|
| theWickedWebDev/8-bit-computer | | |
| Sheet: / | | |
| File: PLCC32-Adapter.kicad_sch | | |
| **Title: PLCC to THT Adapter** | | |
| Size: A5 | Date: 2022-02-10 | **Rev: v1** |
| KiCad E.D.A. eeschema (6.0.0-0) | | Id: 1/1 |

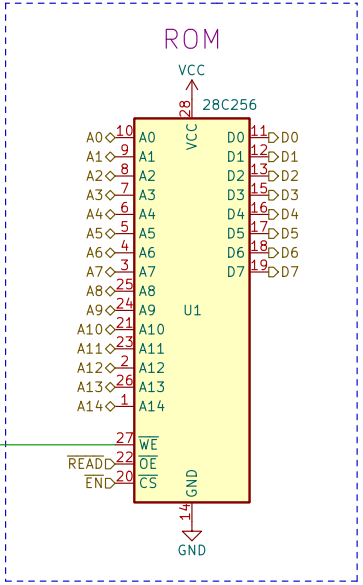
# ROM Cartridge





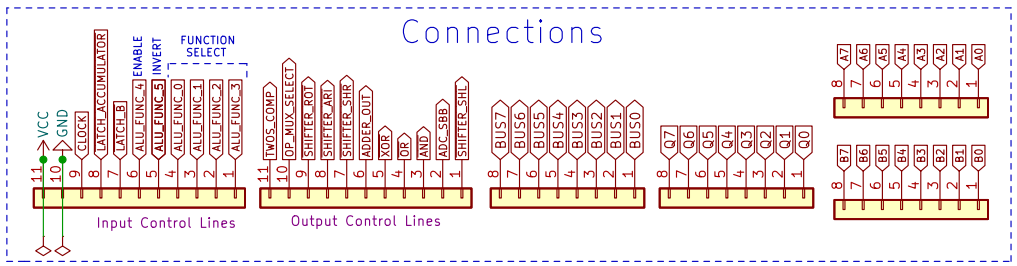



| | | |
|---|---|---|
| theWickedWebDev/8-bit-dev | | |
| Sheet: / | | |
| File: ROM-Cartridge.kicad_sch | | |
| **Title: Programable ROM/RAM Cartridge w/ Battery Backup** | | |
| Size: A5 | Date: 2022-01-11 | **Rev: 3** |
| KiCad E.D.A. eeschema (6.0.0-0) | | Id: 1/1 |

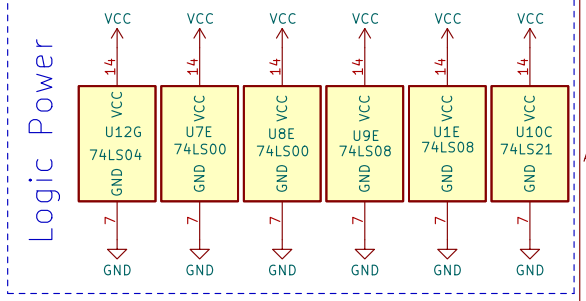
# ALU Control Module

| | | |
|---|---|---|
| A + 1 | AND | SHL |
| A − 1 | NAND | SHR |
| A + B | OR | ASL |
| A − B | NOR | ASR |
| A | XOR | ROR |
| NOT A | XNOR | ROL |

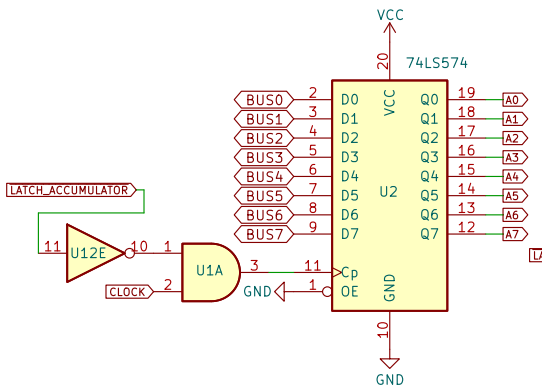
## Connections



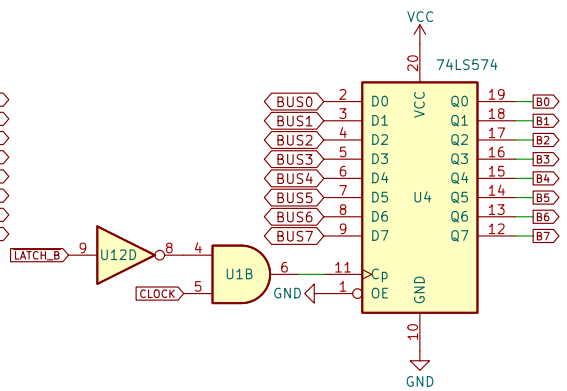

## Logic Power



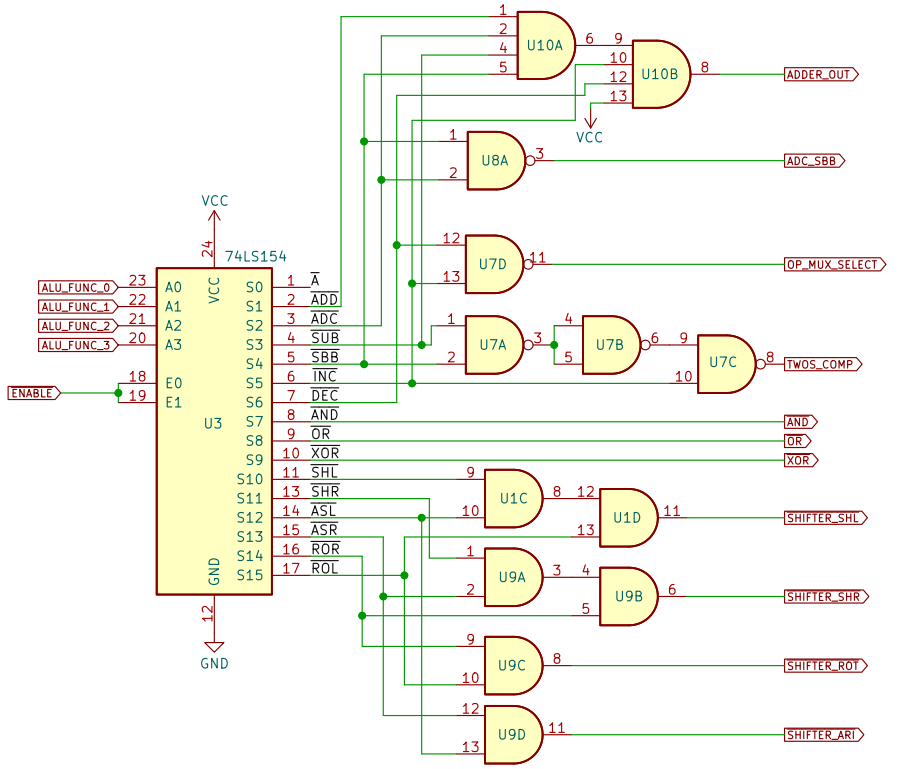

## ACCUMULATOR (A)



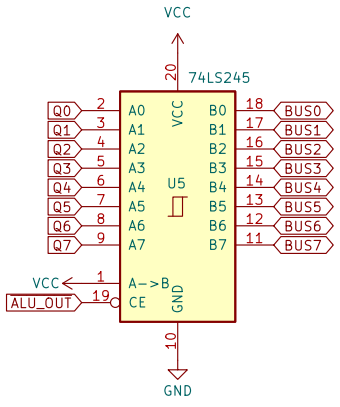

## OPERAND (B)



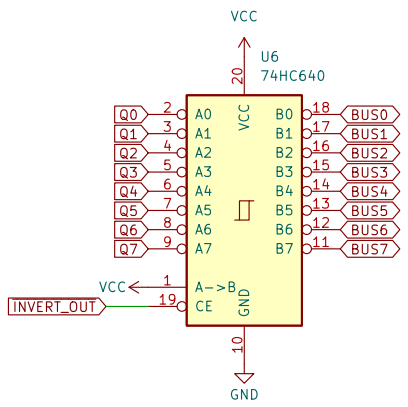

## FUNCTION MULTIPLEXER



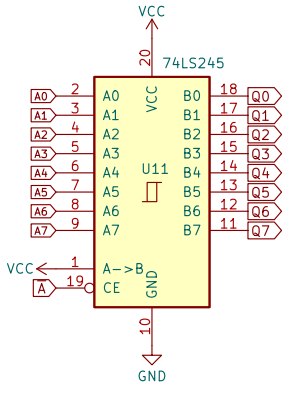

## ALU OUTPUT



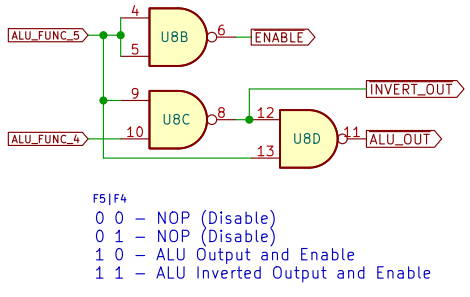

## INVERTED ALU OUTPUT




## PASS A



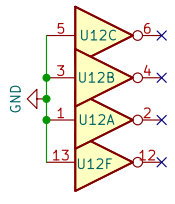

## Enable and Invert Control



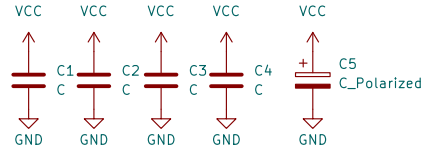

F5|F4

0 0 – NOP (Disable)
0 1 – NOP (Disable)
1 0 – ALU Output and Enable
1 1 – ALU Inverted Output and Enable

## Unused




## Smoothing Caps



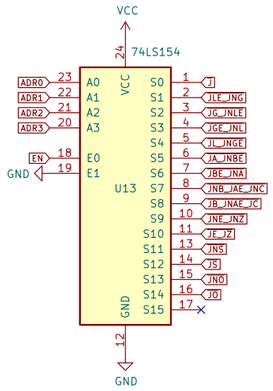

Sheet: /
File: Control_and_Output.kicad_sch

**Title:**

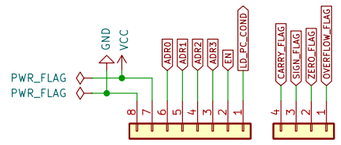
Size: User | Date: | **Rev:**
KiCad E.D.A. eeschema (6.0.0-0) | Id: 1/1

# ALU Conditional Jump Logic

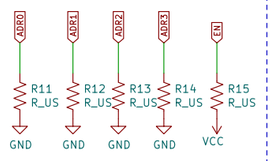
## Jump and Conditional Jump MUX



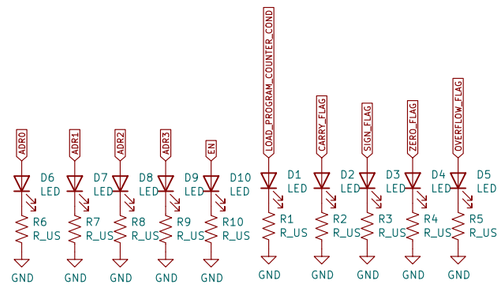

## Connectors



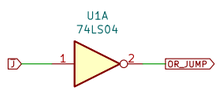

## Pull U/D




## LED Indicators

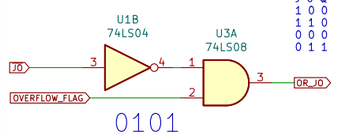



## J — 0000 — Unconditional Jump



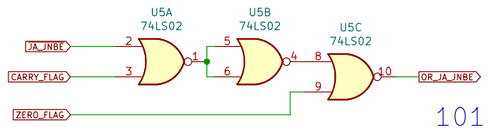

## JO — Jump if OVERFLOW OF=1 — 0101

| J | O | Q |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 1 | 0 |
| 0 | 0 | 0 |
| 0 | 1 | 1 |



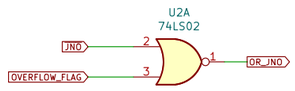

## JA / JNBE — Jump if above / Jump if not below or equal — CF = 0 and ZF = 0 — 1010

| J | C | Z | Q |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 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 | 0 |



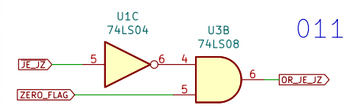

## 0001 — JNO — Jump if NOT OVERFLOW OF=0

| J | O | Q |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 1 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |



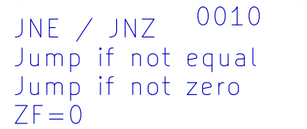

## JE / JZ — Jump if equal, Jump if zero — ZF=1 — 0110

| J | Z | Q |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 1 | 0 |
| 0 | 0 | 0 |
| 0 | 1 | 1 |



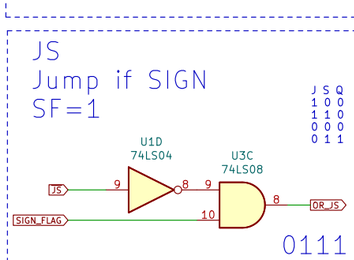

## JNE / JNZ — Jump if not equal, Jump if not zero — ZF=0 — 0010

| J | Z | Q |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 1 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |



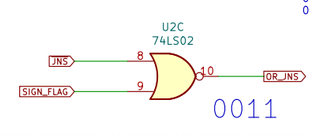

## JS — Jump if SIGN SF=1 — 0111

| J | S | Q |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 1 | 0 |
| 0 | 0 | 0 |
| 0 | 1 | 1 |



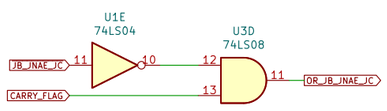

## JS — Jump if NOT SIGN SF=0 — 0011

| J | S | Q |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 1 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |



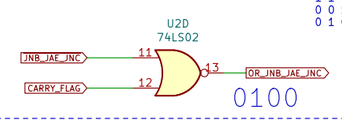

## JB / JNAE / JC — Jump if below / Jump if not above or equal / jump if Carry — CF = 1 — 1000

| J | C | Q |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 1 | 0 |
| 0 | 0 | 0 |
| 0 | 1 | 1 |



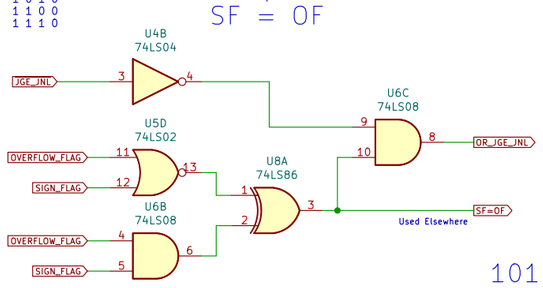

## JNB / JAE / JNC — Jump if not below / Jump if above or equal / Jump if not carry — CF=0 — 0100

| J | C | Q |
|---|---|---|
| 1 | 0 | 0 |
| 1 | 1 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |



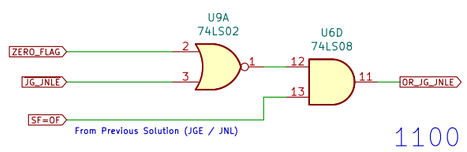

## JGE / JNL — Jump if greater or equal / Jump if not less — SF = OF — 1011

| J | O | S | Q |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |



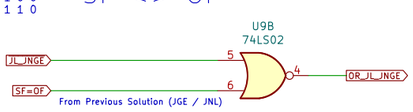

## JG / JNLE — Jump if greater / Jump if not less or equal — ZF = 0 and SF = OF — 1100

| J | S | O | Z | Q |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

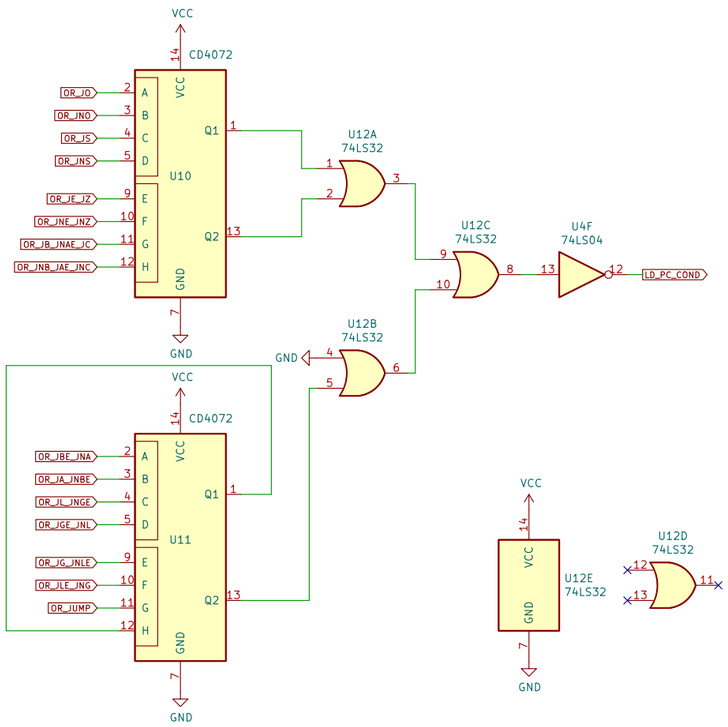



## JL / JNGE — Jump if less / Jump if not greater or equal — SF <> OF — 1101

| J | O | S | Q |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

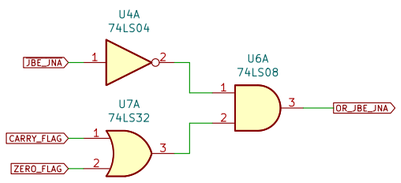



## FINAL OR GATE FOR PC_LOAD control line

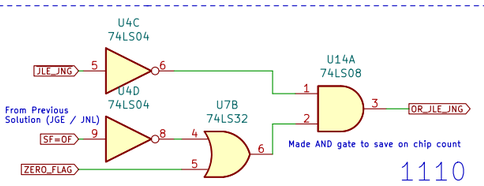



## JBE / JNA — Jump if below or equal / Jump if not above — CF = 1 or ZF = 1 — 1001

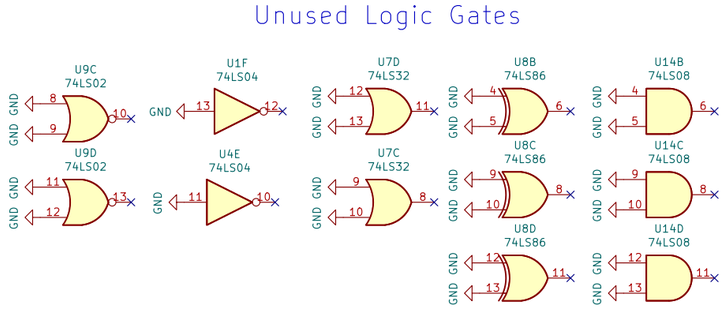
| J | C | Z | Q |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |



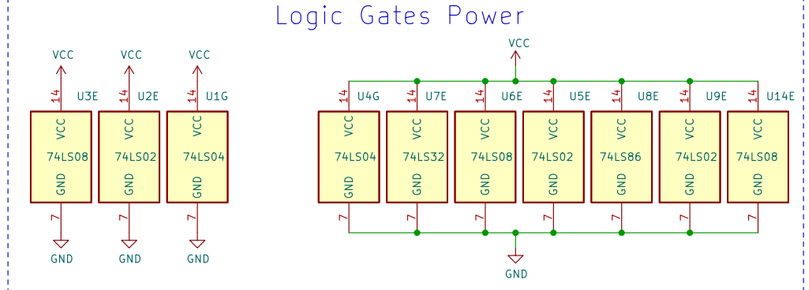

## JLE / JNG — Jump if less or equal / Jump if not greater — ZF = 1 or SF <> OF — 1110




## Unused Logic Gates




## Logic Gates Power




Sheet: /
File: control-unit.kicad_sch

**Title:**

Size: User | Date: | **Rev:**

KiCad E.D.A. eeschema (6.0.0-0) | Id: 1/1

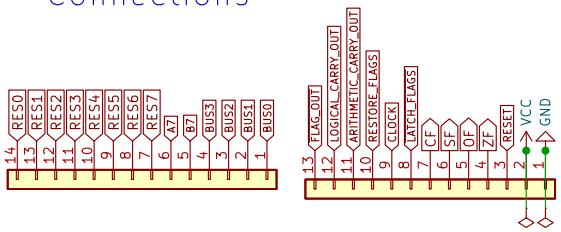
# FLAGS REGISTER

$\overline{\text{LATCH\_FLAGS}}$ – A LOW signal will store the data asserted from the multiplexer into the Flags Register (FR)

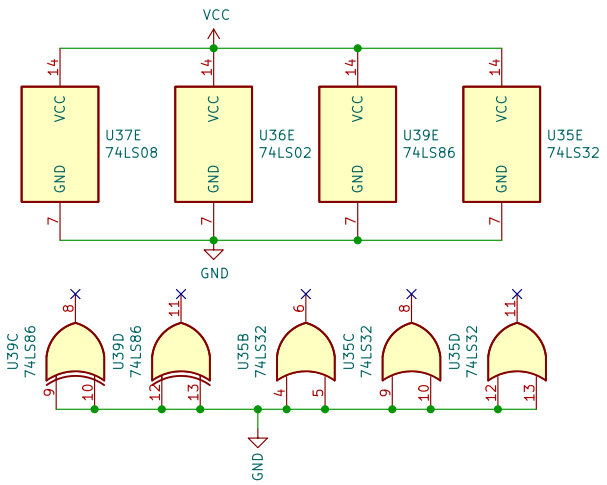
- $\overline{\text{RESTORE}}$: LOW, uses signals from ALU
- RESTORE: HIGH, uses signal asserted on data bus

$\overline{\text{FLAG\_OUT}}$ – Assers the current flags statuses onto the Data bus, typically used to push it onto the stack to handle an ISR

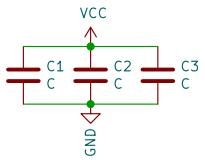
## Connections




## Logic Gate Power

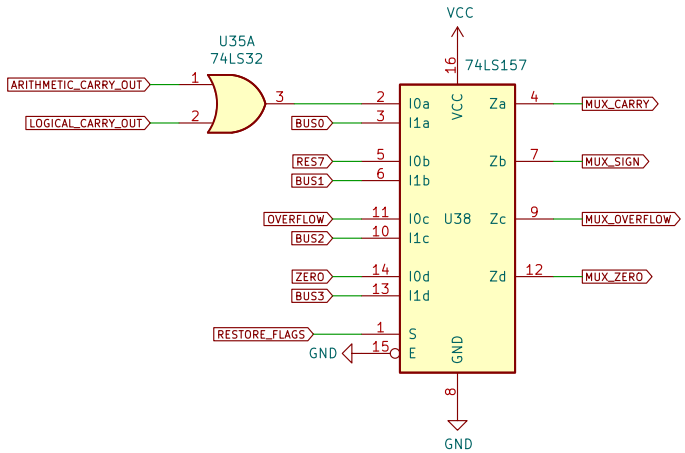



## Smoothing Caps

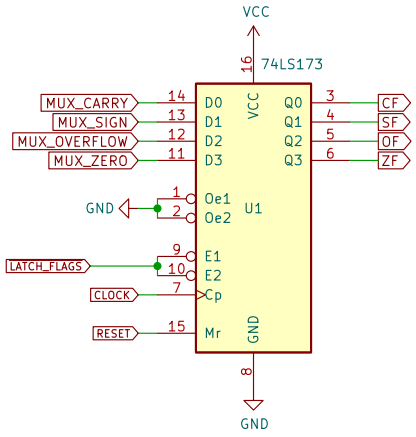



## Source Multiplexer

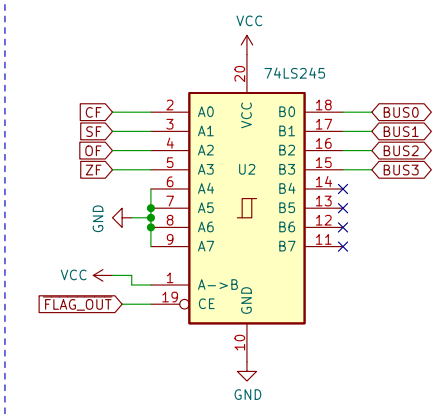
Flags come directly from ALU, or, from the flag/data bus to restore flags from the stack or another location



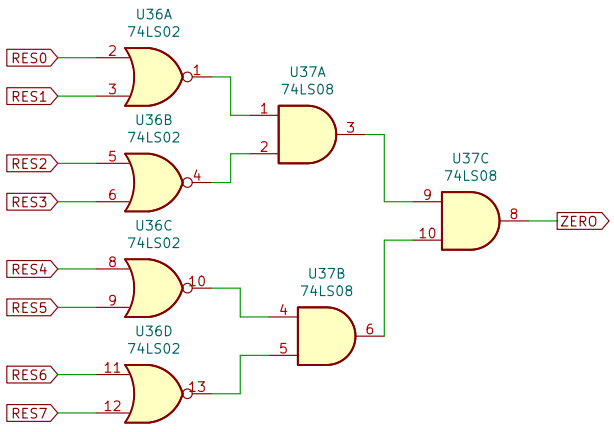

## REGISTER



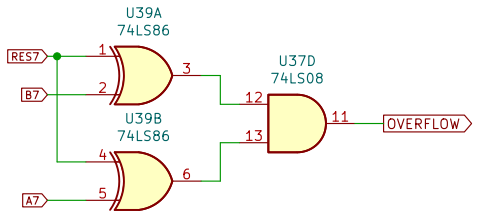

## Bus Connection



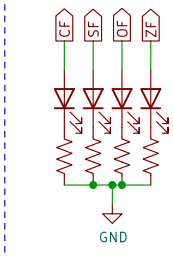

## Zero




## OVERFLOW




## Leds




For storing and asserting current flag statuses from ALU

**theWickedWebDev/8-Bit-Computer**

Sheet: /

File: Flags Register.kicad_sch

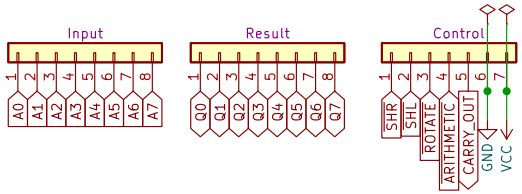
***Title: Flags Register***

| Size: User | Date: 2022-01-03 | **Rev: 3** |
|---|---|---|
| KiCad E.D.A. eeschema (6.0.0-0) | | Id: 1/1 |

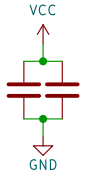
# Shift & Rotate Module

## Logical / Arithmetic

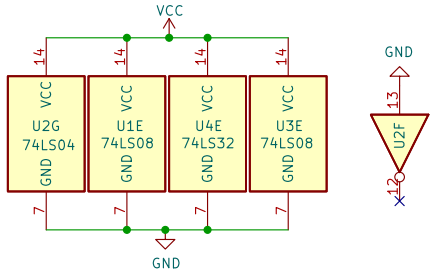
### Conections



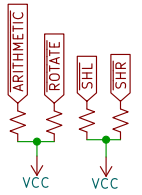

### Cap's



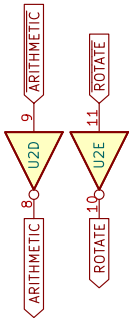

### Logic Power



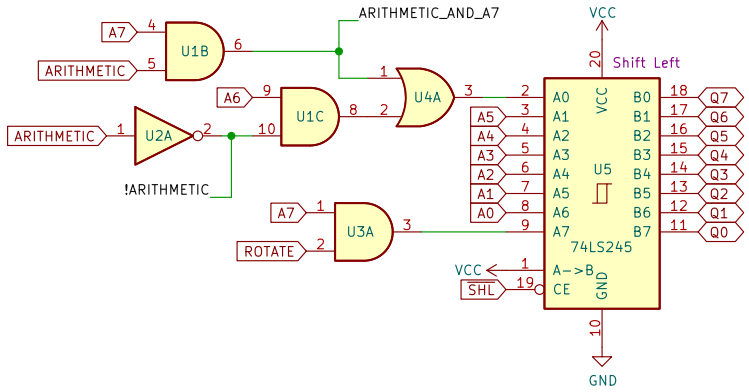

### Pull UP



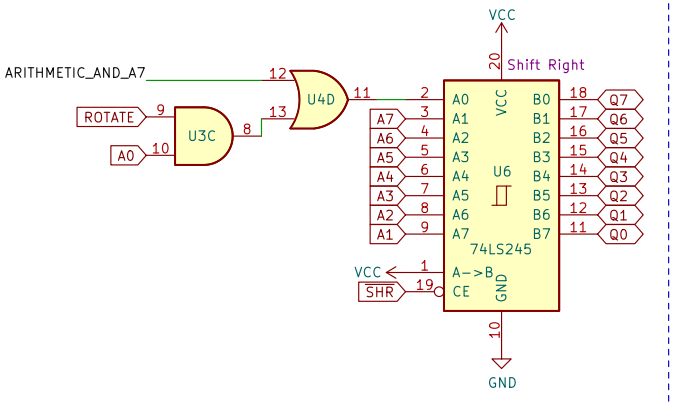

### MAKE ACTIVE LOWS



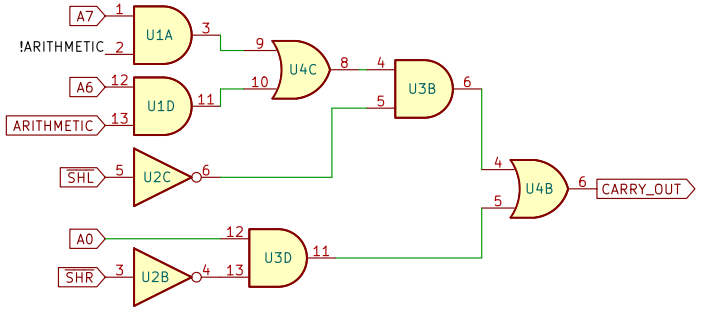

### SHIFT LEFT




### SHIFT RIGHT




### CARRY OUT LOGIC




Rotate Enable / Carry Out

**theWickedWebDev/8-bit-computer**

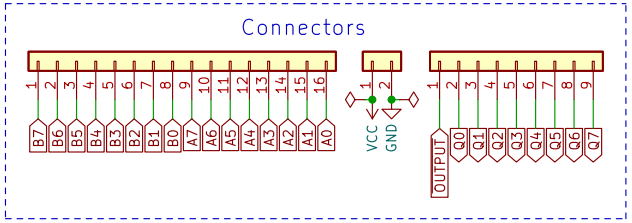
Sheet: /

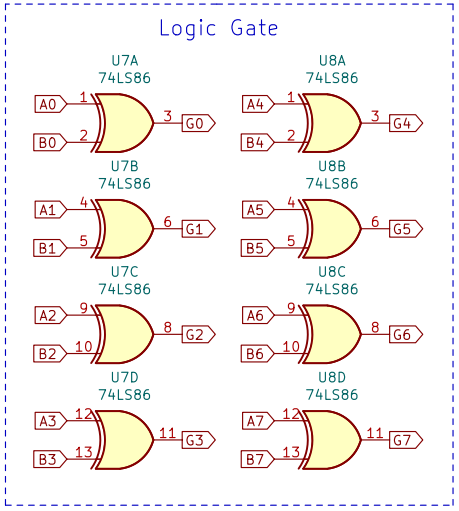
File: Shifter-v6.kicad_sch

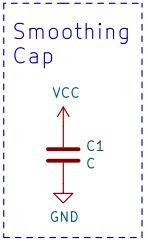
***Title: Arithmetic & Logical Shifter Module***

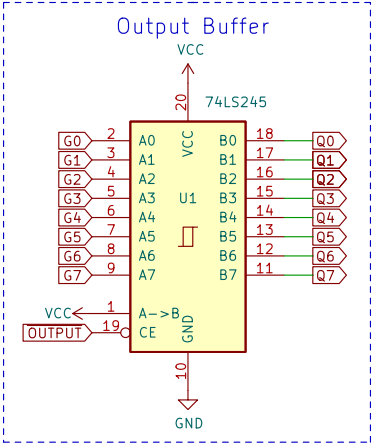
| | | |
|---|---|---|
| Size: A4 | Date: 2022-01-16 | **Rev: 4** |
| KiCad E.D.A. eeschema (6.0.0-0) | | Id: 1/1 |

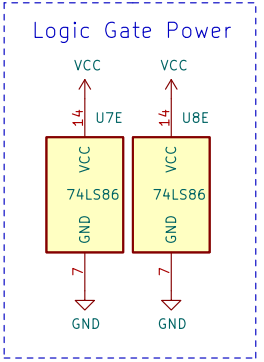
# 8bit Logic Gate

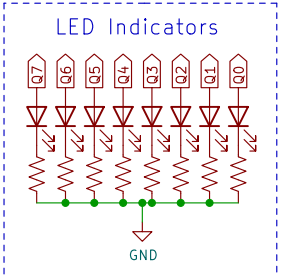



Sheet: /
File: Modules.kicad_sch

**Title: 8bit Logic Gate (ALU)**

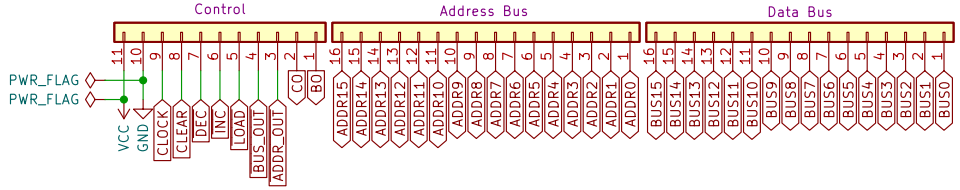
| Size: A5 | Date: | Rev: 3 |
|---|---|---|
| KiCad E.D.A. eeschema (6.0.0-0) | | Id: 1/1 |

# 16bit Address Register

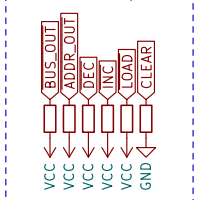
These registers provide functionality to different components
such as the Stack Pointer, Program Counter, or any 16bit
GPR. They have the ability to INC or DEC without using the ALU.

| CLK | $\overline{DEC}$ | $\overline{INC}$ | $\overline{LOAD}$ | $\overline{BUS}$ | $\overline{ADDR}$ | |
|---|---|---|---|---|---|---|
| x | H | H | H | H | H | – NOOP |
| / | H | H | L | x | x | – Load |
| / | L | H | x | x | x | – Decrement |
| / | H | L | x | x | x | – Increment |

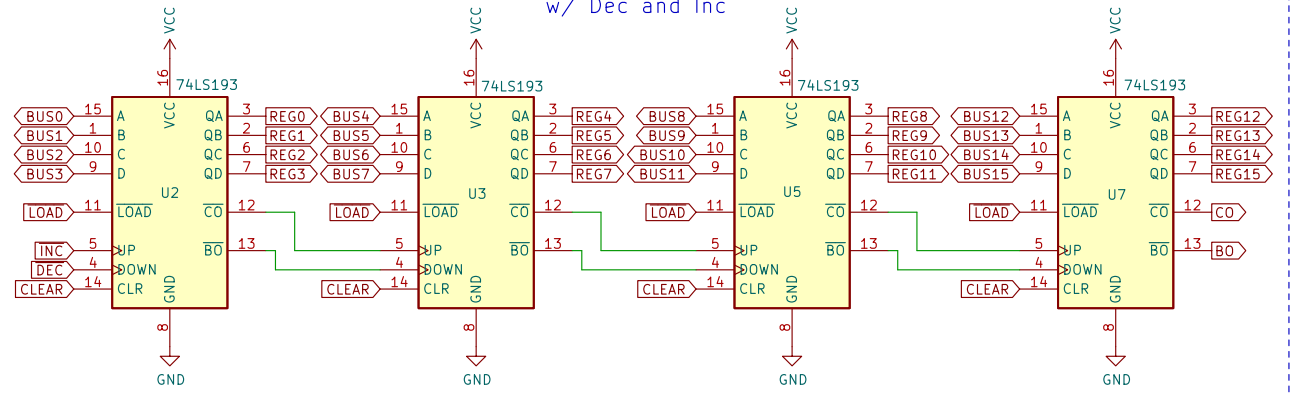
## Connections



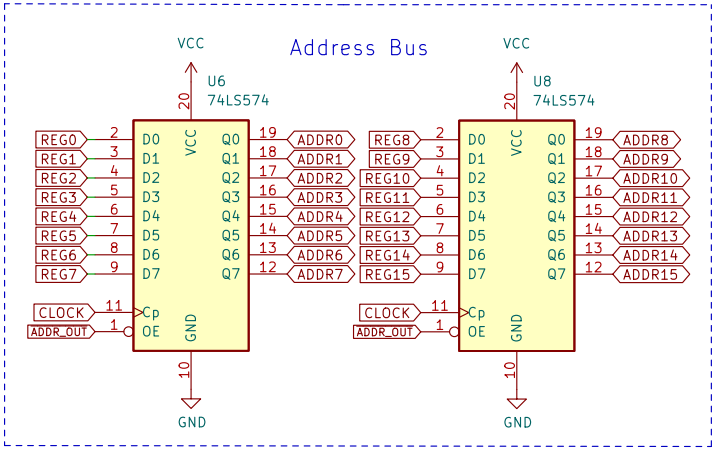

## Pull U/D

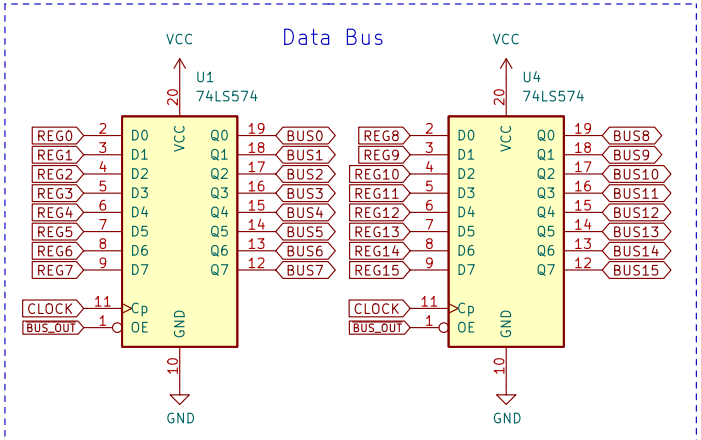



## 16bit Register
w/ Dec and Inc

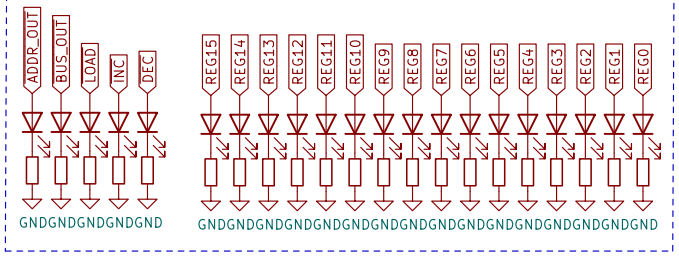



## Output Buffers

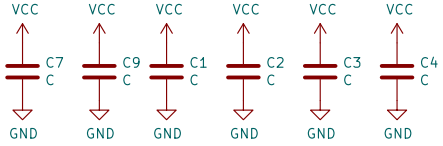



## LED Indicator's




## Smoothing Caps




Sheet: /
File: address-register-smd.kicad_sch

**Title: 16 Bit Address Register w/ INC and DEC**

| Size: A4 | Date: | **Rev: 3** |
|---|---|---|
| KiCad E.D.A. eeschema (6.0.0-0) | | Id: 1/1 |

# Special Purpose Registers

# Special Purpose Registers

## Index Registers

There are three, 4-bit, index registers that serve as the base of 16-bit memory addresses. Data Segment(DS), Code Segment(CS) and Stack Pointer Segment (SS).

**Code Segment (CS):**

Code segment is active while fetching instructions from memory through the program counter. This register is unavailable to the user directly, but is changed during a far-jump instruction.

```
jmp $5:04fs
```

**Data Segment (DS):**

Data segment is active while performing any general reads and writes from memory. This register is controlled only by the user.

```
mov ds, 0x3
```

**Stack Pointer Segment (SS):**

The stack pointer segment is active while performing PUSH, POP, CALL, RTI or RET instructions.
Note: If the SS register gets to 0x0, then a stack overflow interrupt will be triggered.

```
mov ss, 0x2
```

## Stack Pointer

The stack pointer (SP) holds the 16-bit address of the current top of the stack. The SP can point to addresses within the range of: $0:8000 - $f:ffff
It operates by using the last in, first out (LIFO) method. Data can be pushed onto the top of the stack from registers or immediates and can be popped off back onto registers. As data is pushed onto the stack, the SP's value will be decremented; and when data is popped off the stack, the point will be incremented.

## Program Counter

The program counter (PC) (or current instruction register) contains a 16-bit address of the current instruction being fetched from memory. The PC is incremented after its contents are transferred to the address bus. During a jump, the value of the PC is loaded from the address bus, and not incremented. The final instruction address is composed of the CS register value and the PC  register.

```
$cs:pc
```

## Timer Interrupt Register

This timer interrupt register (TIR) contains an 8-bit value that is used to divide the main clock that triggers the timer interrupt. The clock is initially divided by 256 beforehand.

```
Timer Freq. = Main Clock Freq. / 256 / TIR
```

## Interrupt Mask Register

The interrupt mask register (IMR) is a 4-bit register that will enable or disable all interrupts.

## Flags Register

The flags register is used with the ALU to provide specific conditions met while performing ALU operations.

In addition to flags being set during an ALU operation, the flag register can be asserted and loaded directly through the data bus as well. This will be done while returning from an interrupt by means of popping off the stack.

### Zero Flag (ZF)

This flag is set to 1 if the result of an operation is equal to 0

**Carry/Borrow Flag (CF)**

CF is set to 1 when either of the following instructions caused a a carry: ADD, ADC, SUB, SBB, SHL, SHR, ROR, ROL

**Overflow Flag (OF)**

This flag signifies when there was an overflow due to an addition or subtraction operation.

**Sign Flag (SF)**

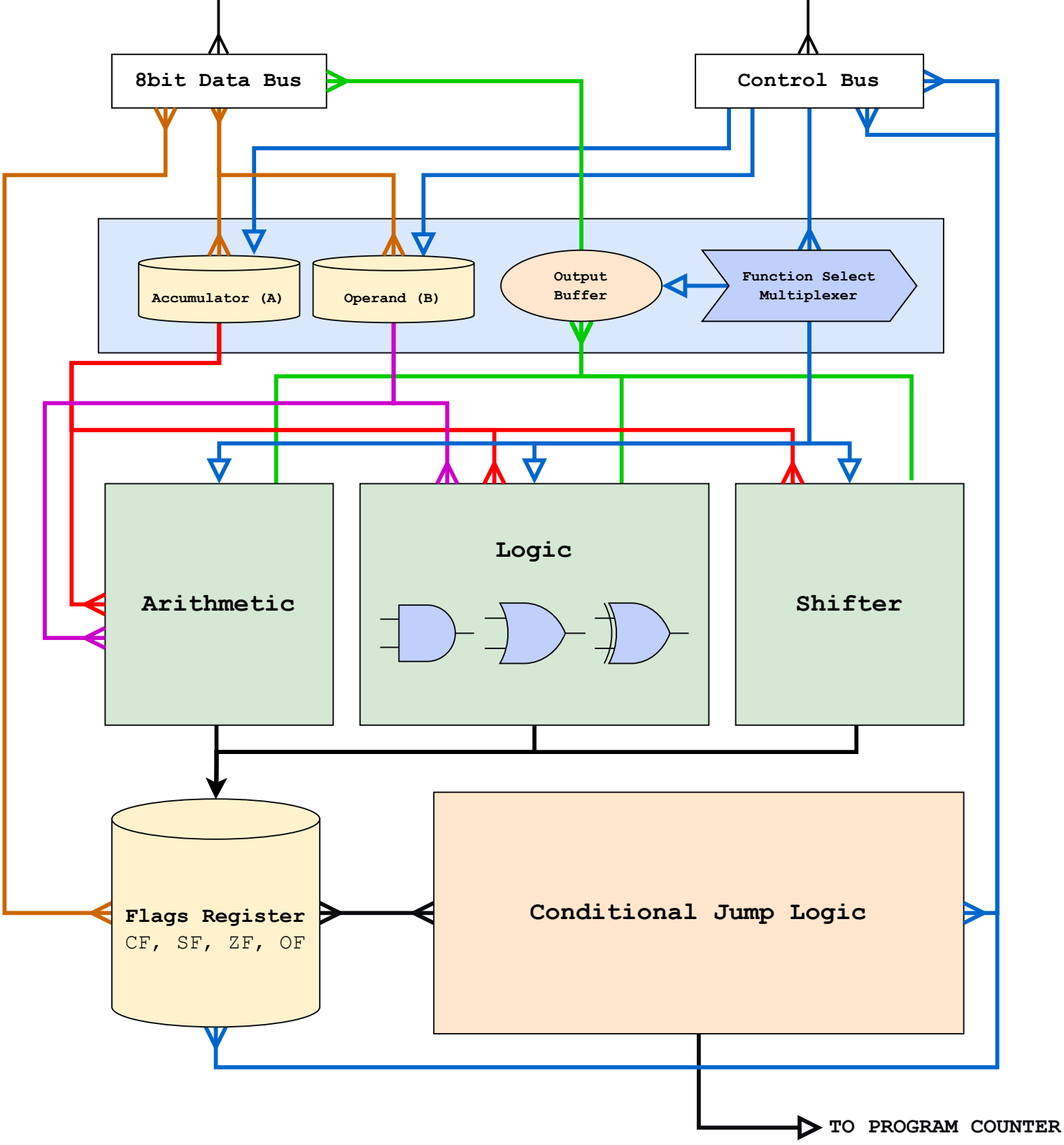
This flag is set to match the MSB of the ALU result, and is used for performing operations on signed integers.

*Figure ### Flag Register Bit Positions*

| **Bit** | **3** | **2** | **1** | **0** |
|---|---|---|---|---|
| Name | SF | OF | ZF | CF |

# Arithmetic Logic Unit (ALU)

# & Flags Register


8bit Data Bus
Control Bus
Accumulator (A)
Operand (B)
Output Buffer
Function Select Multiplexer
Arithmetic
Logic
Shifter
Flags Register
CF, SF, ZF, OF
Conditional Jump Logic
TO PROGRAM COUNTER

# Arithmetic Logic Unit and Conditional Jump

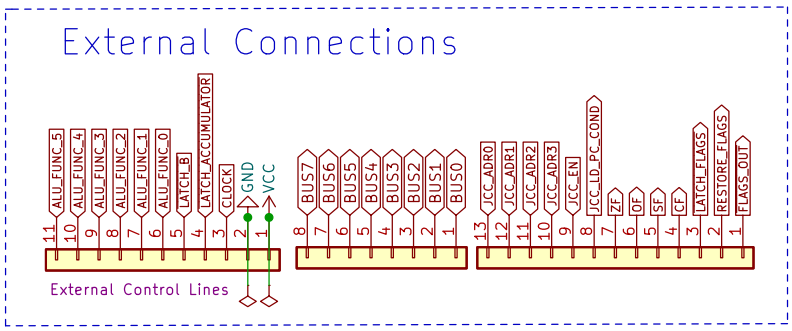
TRUTH TABLE

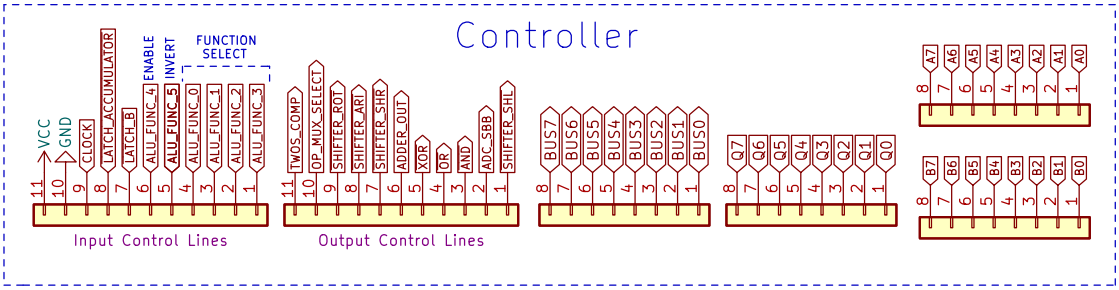
| | | A* | ADD | ADC | SUB | SBB | INC | DEC | AND | OR | XOR | SHL | SHR | ASL | ASR | ROR | ROL | CMP | TEST | NAND | NOR | XNOR | NOT A |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AI | L | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ |
| BI | L | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ | _/ |
| FLAG_IN | L | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| ALU F0 | | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| ALU F1 | | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| ALU F2 | | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| ALU F3 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| (Invert Output) F4 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| (Enable ALU) F5 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

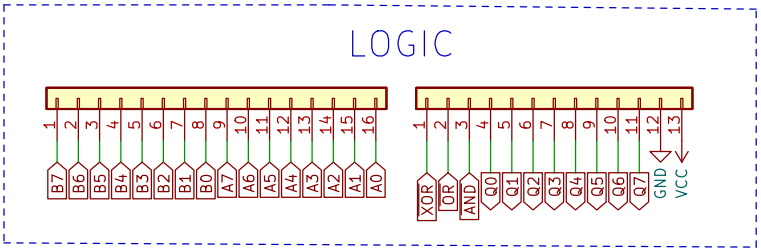
| | | NOP | JP | JLE<br>JNG | JG<br>JNLE | JGE<br>JNL | JL<br>JNGE | JA<br>JNBE | JBE<br>JNA | JNB<br>JAE<br>JNC | JB<br>JNAE<br>JC | JNE<br>JNZ | JE<br>JZ | JNS | JS | JNO | JO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| JCC_EN | L | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| JCC_ADD3 | | x | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| JCC_ADD2 | | x | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| JCC_ADD1 | | x | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| JCC_ADD0 | | x | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

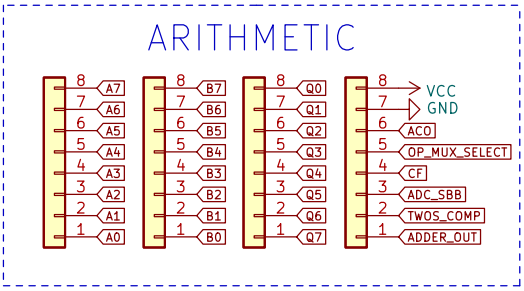
| | | PUSH FLG* | PUSH FLG* |
|---|---|---|---|
| FLAG_OUT | L | L | L |
| RESTORE_FLAGS | H | L | H |

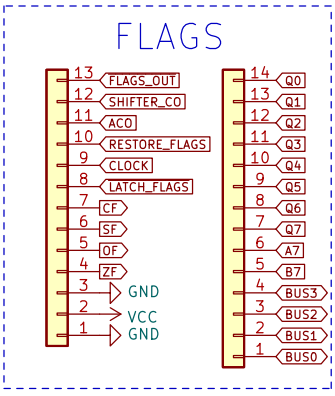
# ALU Backplane

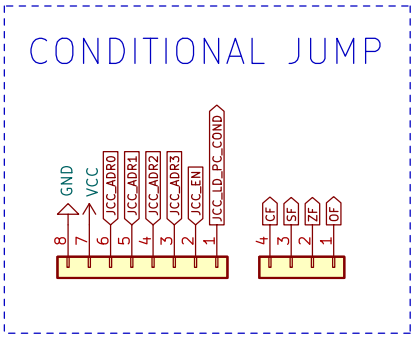


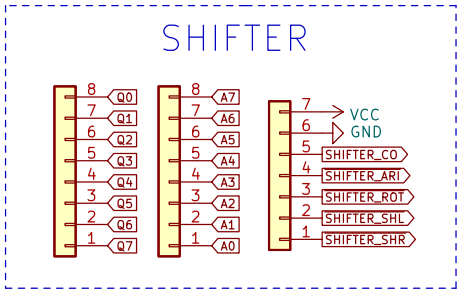



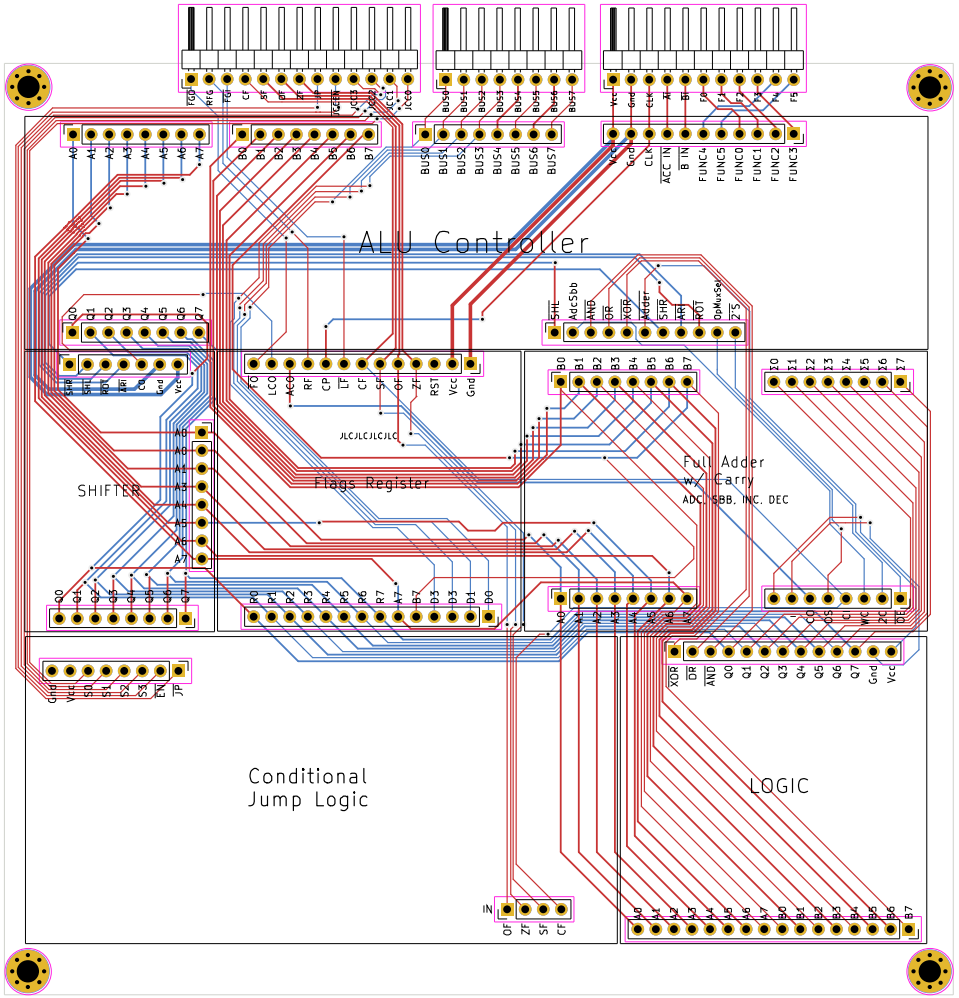




**theWickedWebDev/8-bit-computer**

Sheet: /
File: ALU-Backplane.kicad_sch

***Title: ALU & Conditional Jump Backplane***

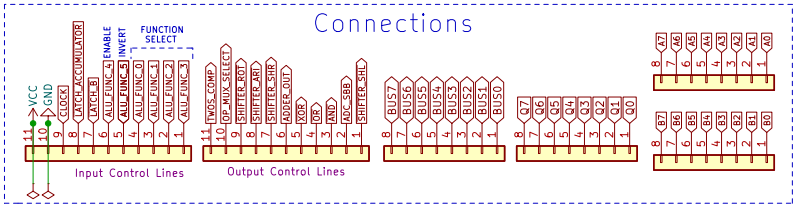
| Size: User | Date: 2022-01-17 | **Rev: 4** |
|---|---|---|
| KiCad E.D.A. kicad (6.0.0-0) | | Id: 1/1 |


ALU Controller
SHIFTER
Flags Register
Full Adder
w/ Carry
ADC, SBB, INC, DEC
Conditional
Jump Logic
LOGIC

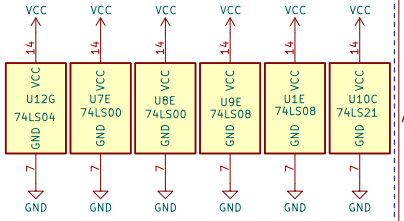
# ALU Control Module

| | | |
|---|---|---|
| A + 1 | AND | SHL |
| A − 1 | NAND | SHR |
| A + B | OR | ASL |
| A − B | NOR | ASR |
| A | XOR | ROR |
| NOT A | XNOR | ROL |

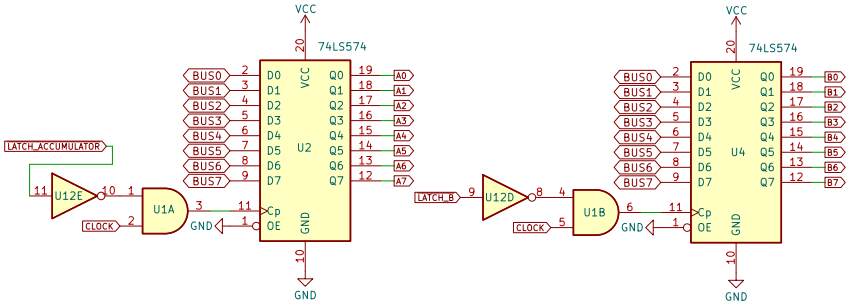
## Connections




## Logic Power



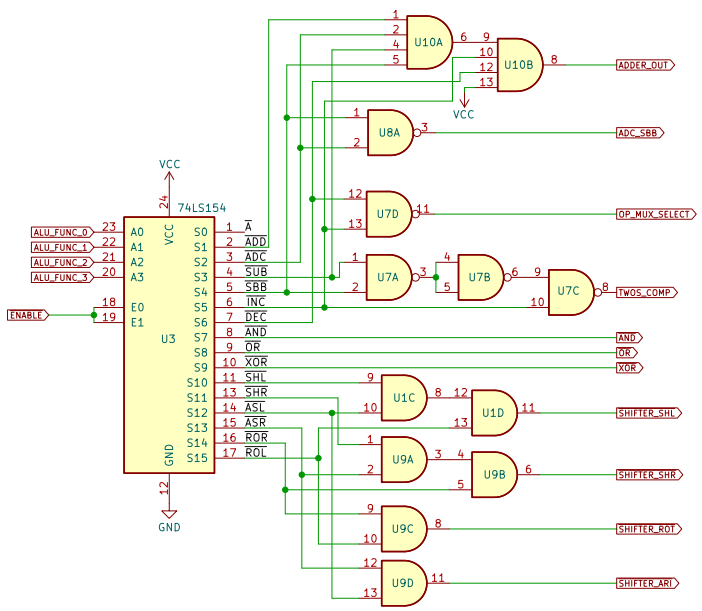

## ACCUMULATOR (A)



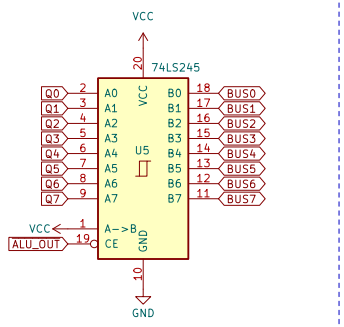

## OPERAND (B)



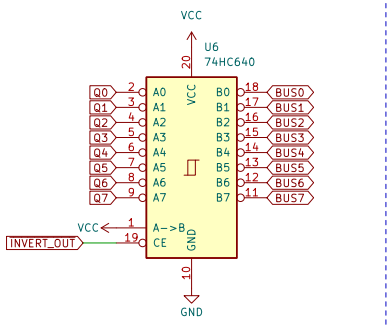

## FUNCTION MULTIPLEXER



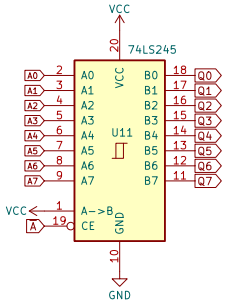

## ALU OUTPUT



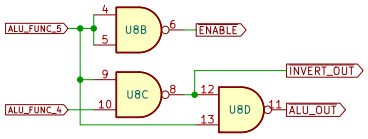

## INVERTED ALU OUTPUT




## PASS A

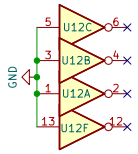



## Enable and Invert Control

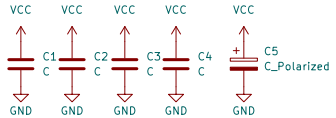



F5|F4
0 0 – NOP (Disable)
0 1 – NOP (Disable)
1 0 – ALU Output and Enable
1 1 – ALU Inverted Output and Enable

## Unused



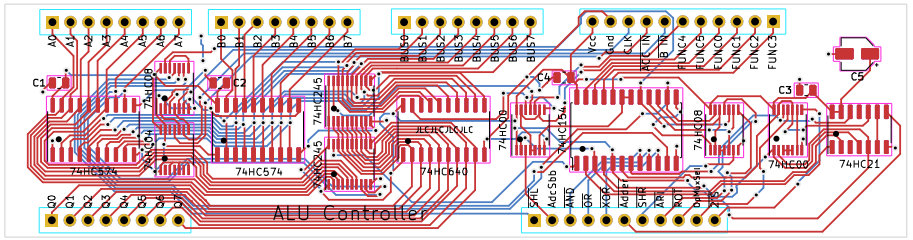

## Smoothing Caps




Sheet: /
File: Control_and_Output.kicad_sch

**Title:**

| Size: User | Date: | **Rev:** |
|---|---|---|
| KiCad E.D.A. kicad (6.0.0-0) | | Id: 1/1 |


ALU Controller
74HC574
74HC574
74HC640
74HC00
74HC21
C1
C2
C3
C4
C5

# Arithmetic Module

8bit Arithmetic Module provides
ADD, ADC, SUB, SBB, INC, DEC

DEC A: MUX_SEL: 1, TC: 0, CI: 0

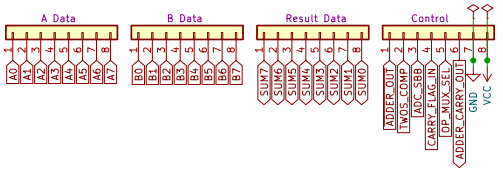
A − B: MUX_SEL: 0, TC: 1, CI: 1

A + B: MUX_SEL: 0, TC: 0, CI: 0

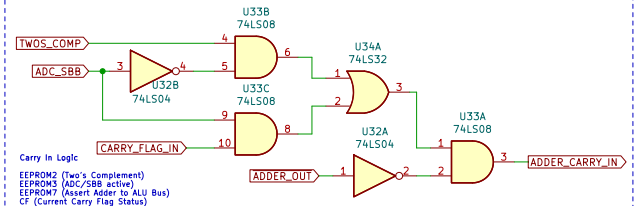
INC A: MUX_SEL: 1, TC: 1, CI: 1

A − B − Ci: MUX_SEL: 0, TC: 1, CI: ?
A + B + Cf: MUX_SEL: 0, TC: 0, CI: ?

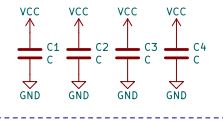
## Connectors



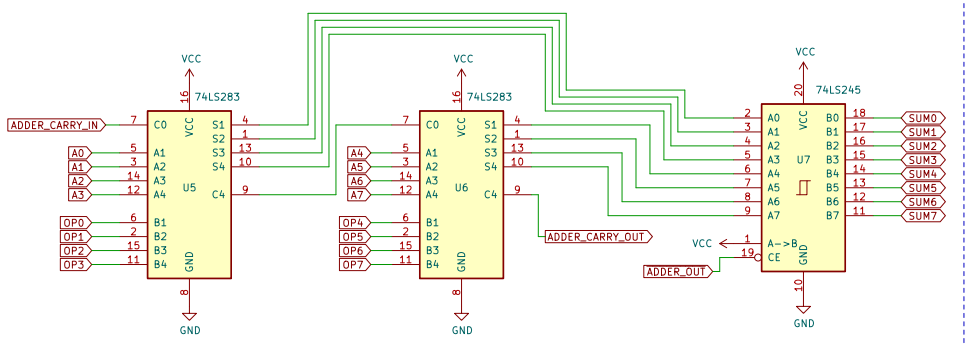

## Arithmetic Carry In Logic



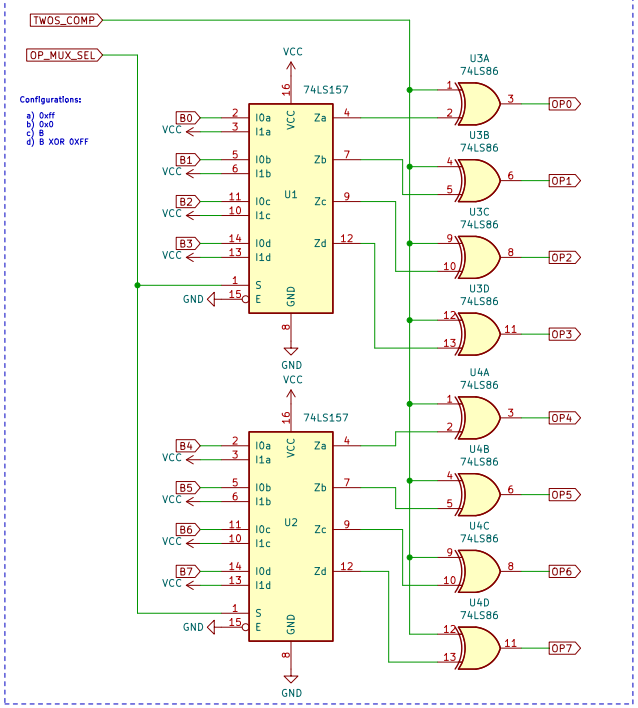

## Smoothing Caps



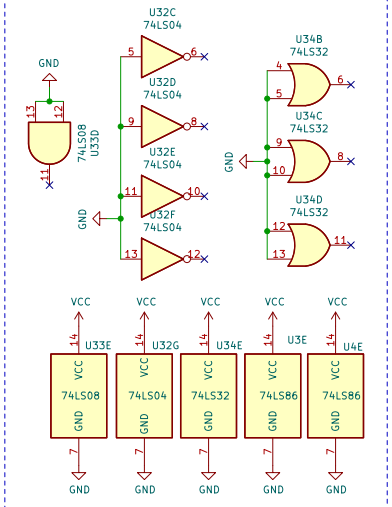

## Full Adder w/ Carry




## Operand Multiplexer




## Logic Power

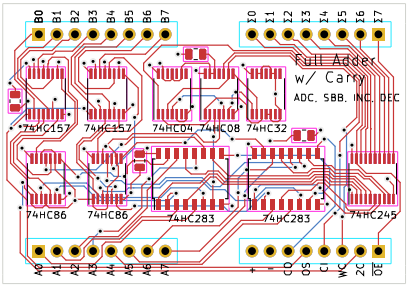



ADD / SUB / ADC / SBB / INC / DEC

Sheet: /
File: Arithmetic.kicad_sch

**Title: Arithmetic Module**

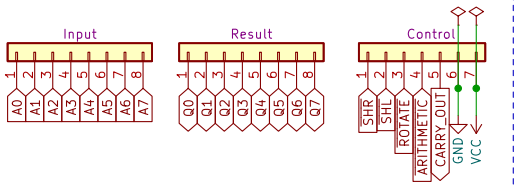
| Size: User | Date: | Rev: 3 |
|---|---|---|
| KiCad E.D.A. kicad (6.0.0−0) | | Id: 1/1 |


B0
B1
B2
B3
B4
B5
B6
B7
I0
I1
I2
I3
I4
I5
I6
I7
Full Adder
w/ Carry
ADC. SBB. INC. DEC
74HC157
74HC157
74HC04
74HC08
74HC32
74HC86
74HC86
74HC283
74HC283
74HC245
A0
A1
A2
A3
A4
A5
A6
A7
+
-
CO
dS
CI
WC
2C
OE

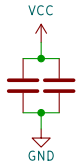
# Shift & Rotate Module

## Logical / Arithmetic

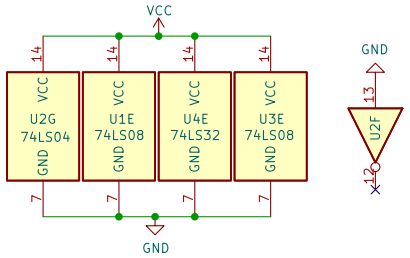
### Conections



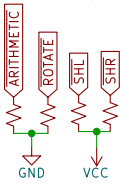

### Cap's



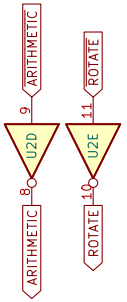

### Logic Power



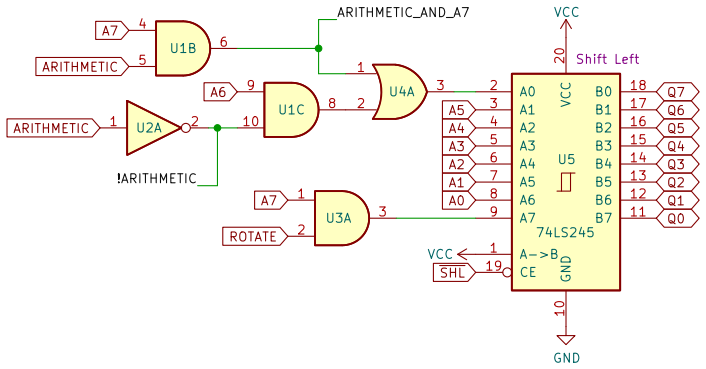

### Pull U/D



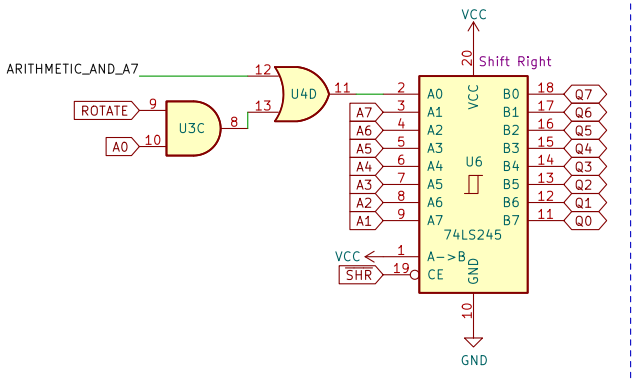

### MAKE ACTIVE LOWS



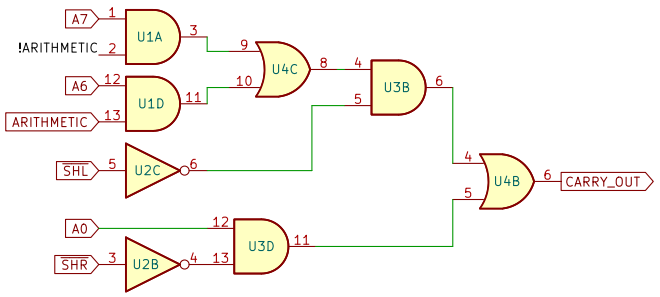

### SHIFT LEFT




### SHIFT RIGHT




### CARRY OUT LOGIC



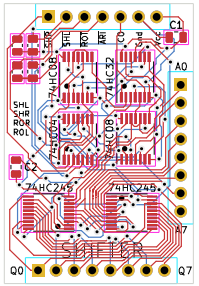

Rotate Enable / Carry Out

**theWickedWebDev/8-bit-computer**

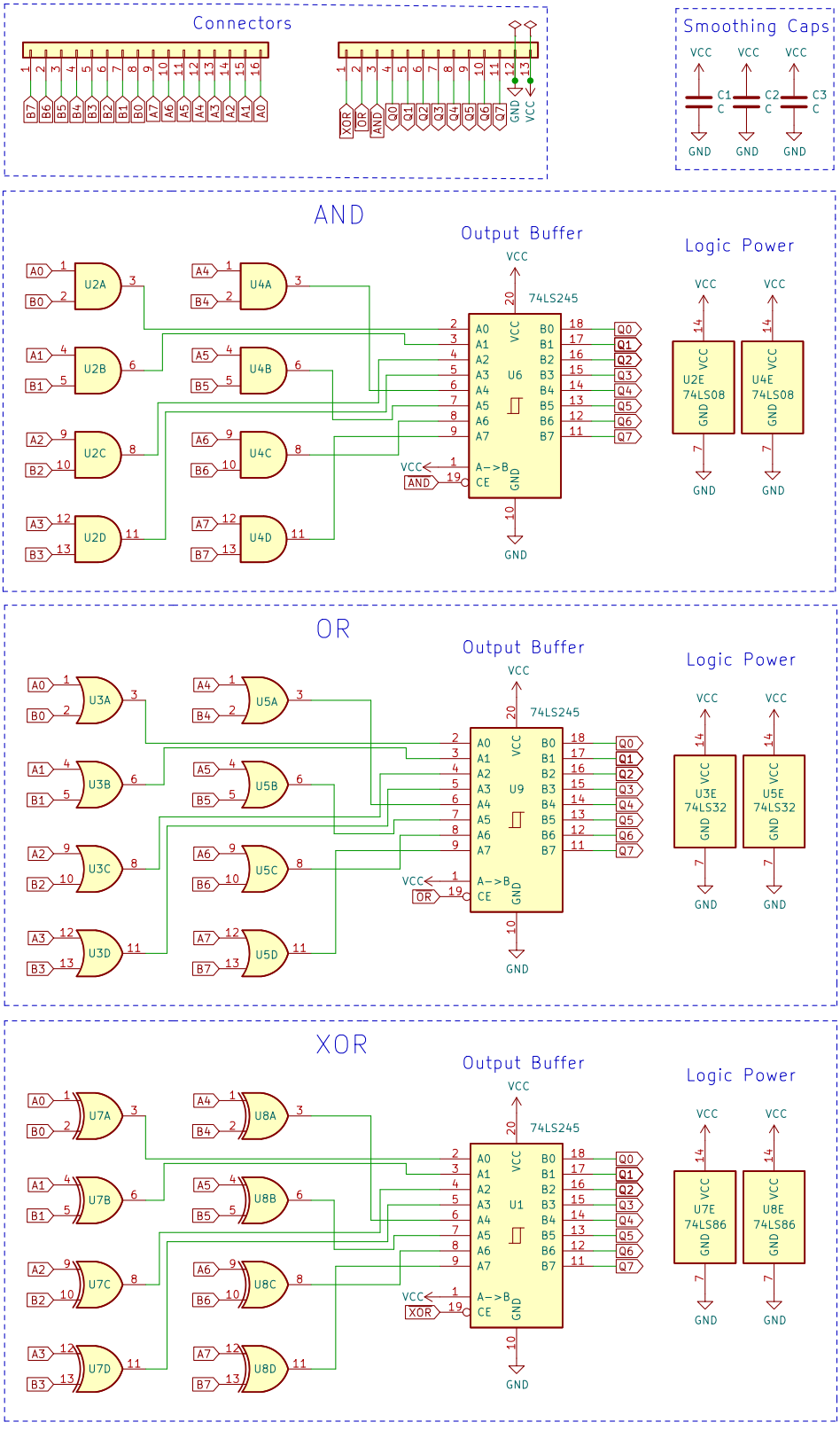
Sheet: /
File: Shifter-v6.kicad_sch

***Title: Arithmetic & Logical Shifter Module***

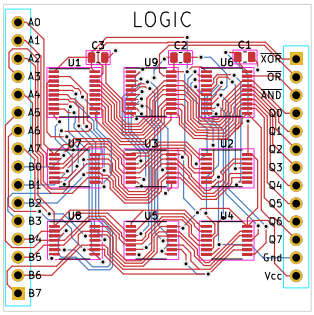
| Size: A4 | Date: 2022-01-16 | **Rev: 4** |
|---|---|---|
| KiCad E.D.A. kicad (6.0.0-0) | | Id: 1/1 |


C1
A0
A7
Q0
Q7
C2
74HC245
74HC245
74HC299
74HC299
74HC299
74HC299
SHL
SHR
RO0
RO7

# 8bit Logic Gate




| | |
|---|---|
| Sheet: / | |
| File: Modules.kicad_sch | |
| **Title: 8bit Logic Gate (ALU)** | |
| Size: User | Date: |
| KiCad E.D.A. kicad (6.0.0-0) | **Rev: 3** |
| | Id: 1/1 |


LOGIC
A0
A1
A2
A3
A4
A5
A6
A7
B0
B1
B2
B3
B4
B5
B6
B7
C3
C2
C1
U1
U9
U6
U7
U3
U2
U8
U5
U4
XOR
OR
AND
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Gnd
Vcc

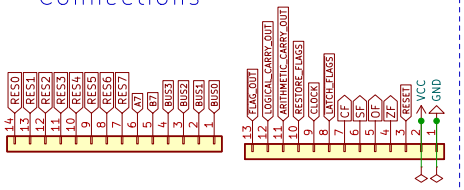
# FLAGS REGISTER

$\overline{\text{LATCH\_FLAGS}}$ – A LOW signal will store the data asserted from the multiplexer into the Flags Register (FR)

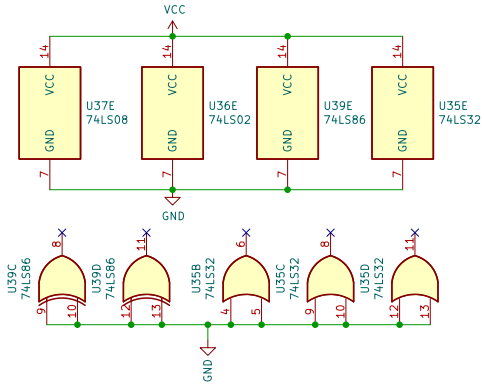
- $\overline{\text{RESTORE}}$: LOW, uses signals from ALU
- RESTORE: HIGH, uses signal asserted on data bus

$\overline{\text{FLAG\_OUT}}$ – Assers the current flags statuses onto the Data bus, typically used to push it onto the stack to handle an ISR

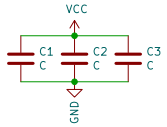
## Connections




## Logic Gate Power

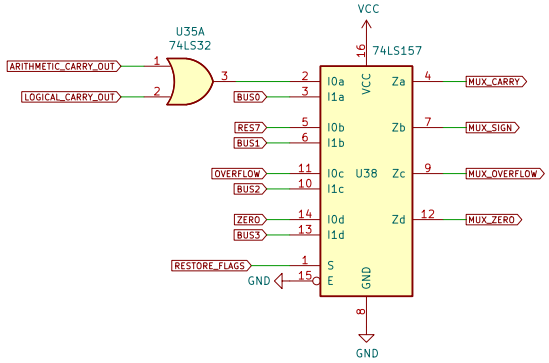



## Smoothing Caps

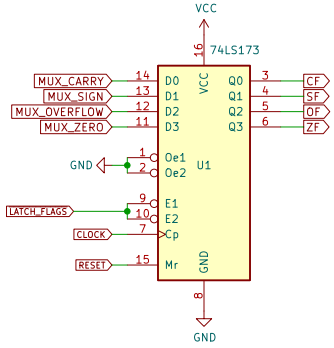



## Source Multiplexer

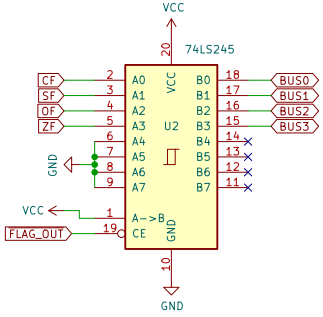
Flags come directly from ALU, or, from the flag/data bus to restore flags from the stack or another location



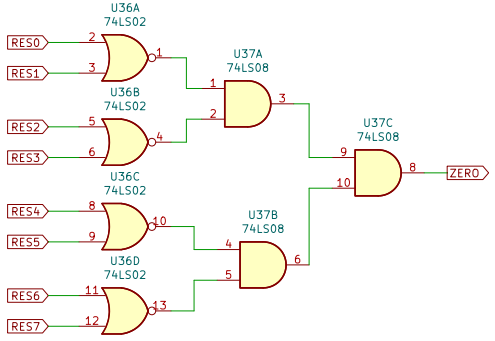

## REGISTER



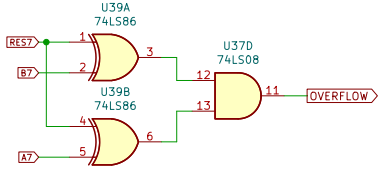

## Bus Connection



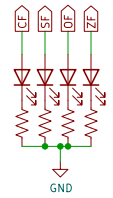

## Zero




## OVERFLOW




## Leds




For storing and asserting current flag statuses from ALU

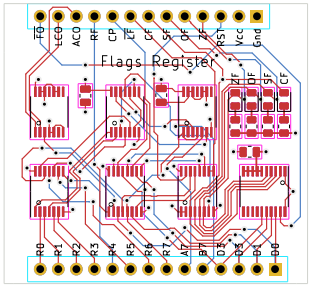
**theWickedWebDev/8-Bit-Computer**

Sheet: /

File: Flags Register.kicad_sch

***Title: Flags Register***

| | | |
|---|---|---|
| Size: User | Date: 2022-01-03 | **Rev: 3** |
| KiCad E.D.A. kicad (6.0.0-0) | | Id: 1/1 |


Flags Register
IEO
LEO
ACO
AF
CP
ZF
CF
SF
OF
IF
RST
Vcc
Gnd
ZF
OF
SF
CF
R0
R1
R2
R3
R4
R5
R6
R7
A7
B7
D7
D6
D5
D4
D0

# Memory

# ROM & RAM

# Input & Output (I/O)

# CPU Control