Github: theWickedWebDev/8-bit-computer

Trilobyte

http://trilobyte.io/

User Manual V0.1

2022 Trilobyte Computer // Stephen Young stephenyoung7267@gmail.com

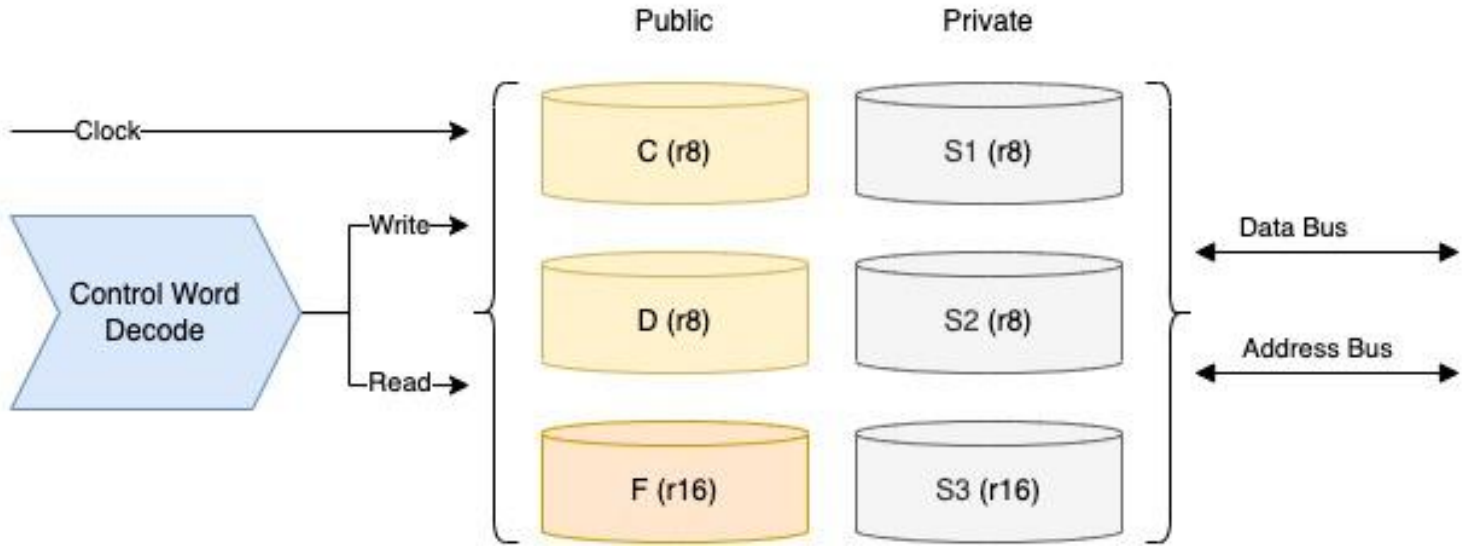
Architecture Overview

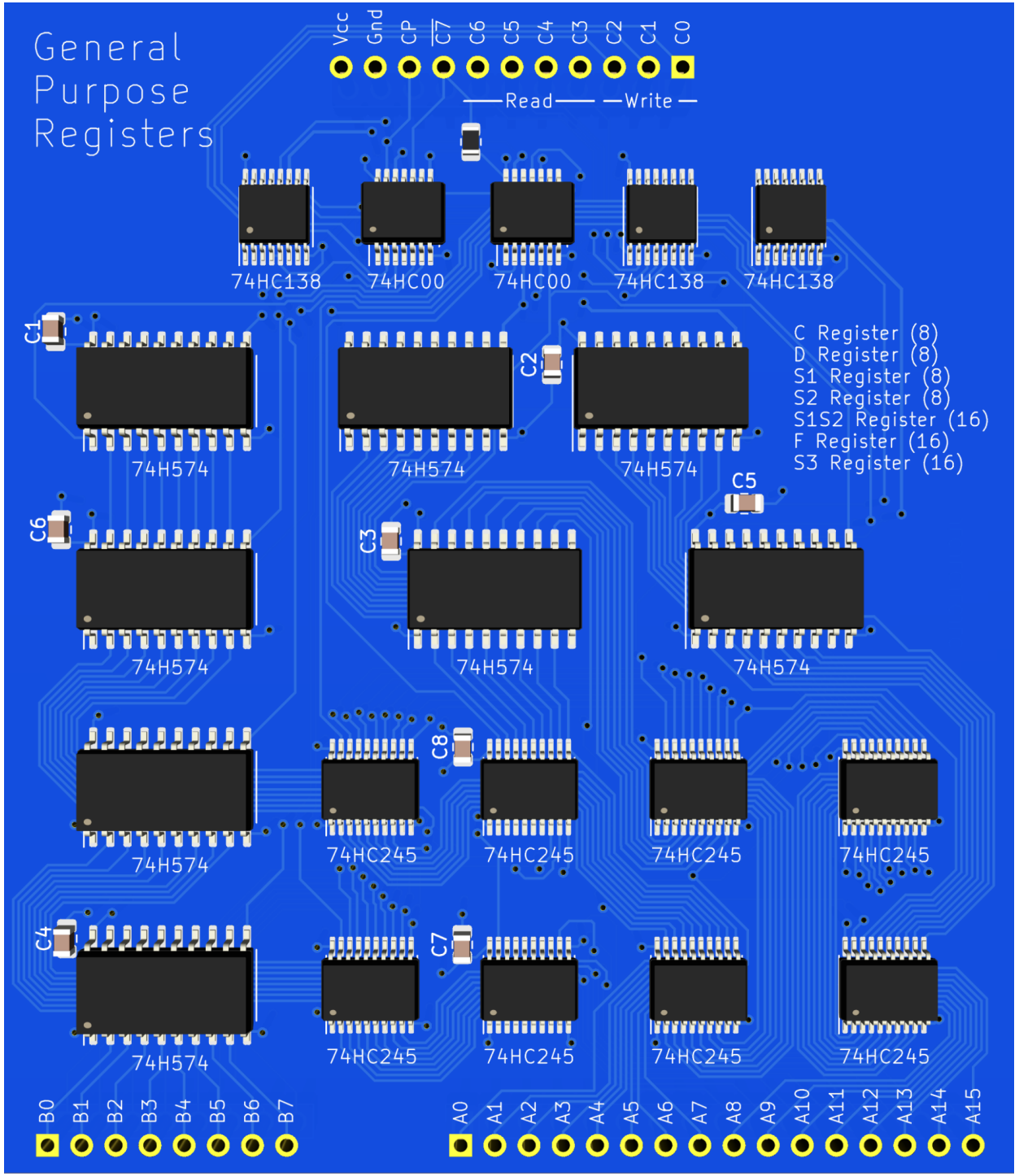
- **CISC/RISC** Hybrid
- **~1Mhz** clock speed (Maybe? Not actually sure yet)
- **256 x 8-bit Opcodes**
    - Performs microcode operations
- **8-bit arithmetic and logic functions**
- **Memory**: 2 megabytes of addressable memory using 16bit addresses with 5-bit segments.
- **Interrupts:** 6 maskable hardware interrupts & 2 maskable timer interrupts
- **15 I/O devices**
    - VGA, 640 x 480
    - Midi / Audio Out
    - Serial Port
    - Can use ROM cartridges **(**games, programs, data…)
    - DAC - Digital to Analog Converter
    - ADC - Analog to Digital Converter
    - Keyboard
- **Data bus (8-bit)**
    - ALU
    - Memory
    - I/O
    - General Purpose Registers
- **Address bus (16-bit)**
    - Program Counter
    - Stack Pointer
    - Memory Address Register
    - General Purpose Registers
- **Control line bus**: Asserted by the decoded microcode instructions and used by all components.

General Purpose Registers (GPR)

The Trilobyte CPU contains **two 8-bit general-purpose registers** and **one 16-bit general-purpose counter register** that is available for user control. The GPR module itself contains an additional two 8-bit scratch registers and one 16-bit scratch register.

Power: **5V Vcc**




General
Purpose
Registers
Vcc
Gnd
CP
C7
C6
C5
C4
C3
C2
C1
C0
Read
Write
74HC138
74HC00
74HC00
74HC138
74HC138
C1
C2
C3
C4
C5
C6
C7
C8
74H574
74H574
74H574
74H574
74H574
74H574
74H574
74HC245
74HC245
74HC245
74HC245
74HC245
74HC245
74HC245
74HC245
C Register (8)
D Register (8)
S1 Register (8)
S2 Register (8)
S1S2 Register (16)
F Register (16)
S3 Register (16)
B0
B1
B2
B3
B4
B5
B6
B7
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15

Register

Name	Size	Visibility	Mnemonic
C	8-bit	PUBLIC	r, r8
D	8-bit	PUBLIC	r, r8
F	16-bit	PUBLIC	r, r16
S1	8-bit	PRIVATE	r, r8
S2	8-bit	PRIVATE	r, r8
S3	16-bit	PRIVATE	r, r16

Bus Connections

- C, D: Loads and asserts data from/to the 8-bit data bus
- F: Loads and asserts data from/to the 16-bit data bus
- S1: Loads and asserts data from/to the 8-bit data bus, as well as, asserts its data to the LSB of the 16-bit address bus.
- S2: Loads and asserts data from/to the 8-bit data bus, as well as, asserts its data to the MSB of the 16-bit address bus.
- S3: Loads and asserts data from/to the 16-bit data bus, as well as, asserts its MSB, or LSB to the 8-bit data bus.
- S1S2: Asserts register-pair to the 16-bit data bus

Control & Instructions

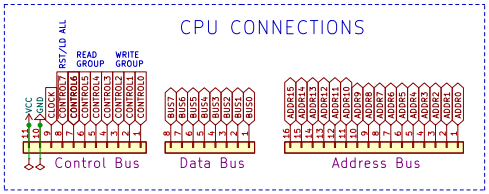
Description	C7	Control [0..6]	Hex Code
Reset / Load ALL	0	0000_000	0x0
Load C	1	0000_001	0x81
Load D	1	0000_010	0x82
Load S1	1	0000_011	0x83
Load S2	1	0000_100	0x84
Load F	1	0000_101	0x85
Load S3	1	0000_110	0x86
Assert C	1	0001_000	0x88
Assert D	1	0010_000	0x90
Assert S1	1	0011_000	0x98
Assert S2	1	0100_000	0xa0
Assert S1S2	1	0101_000	0xa8
Assert F	1	0110_000	0xb0

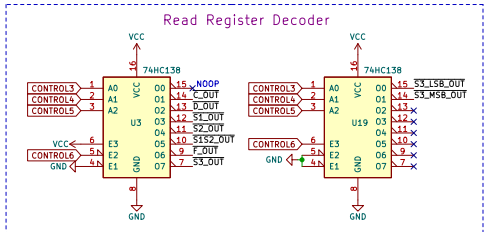
Assert S3	1	0111_000	0xb8
Assert S3L	1	1000_000	0xc0
Assert S3H	1	1001_000	0xc8
MOV C, D	1	0010_001	0x91
MOV C, S1	1	0011_001	0X99
MOV C, S2	1	0100_001	0xa1
MOV C, S3L	1	1000_001	0xc1
MOV C, S3H	1	1001_001	0xc9
MOV D, C	1	0001_010	0x8a
MOV D, S1	1	0011_010	0x9a
MOV D, S2	1	0100_010	0xa2
MOV D, S3L	1	1000_010	0xc2
MOV D, S3H	1	1001_010	0xca
MOV S1, C	1	0001_011	0x8b
MOV S1, D	1	0010_011	0x9a

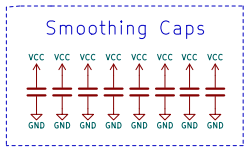
MOV S1, S2	1	0100_011	0x93

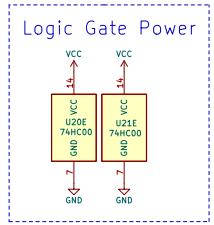
GENERAL PURPOSE REGISTER ARRAY

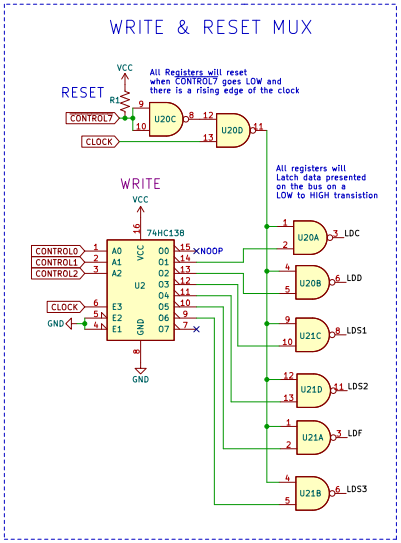
8bit & 16bit C^8, D^8, $S1^8$, $S2^8$, $S1S2^{16}$, F^{16}, $S3^{16}$

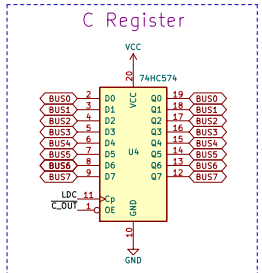


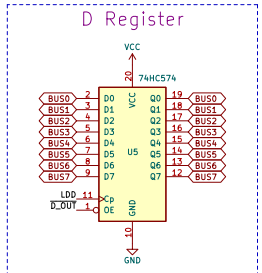


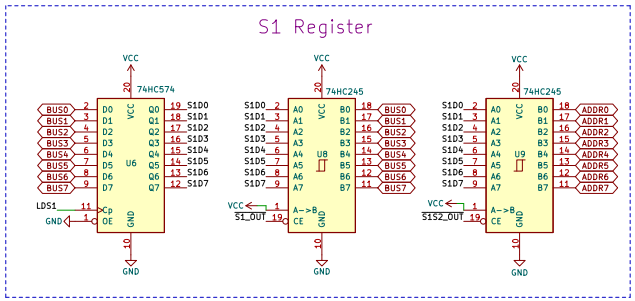


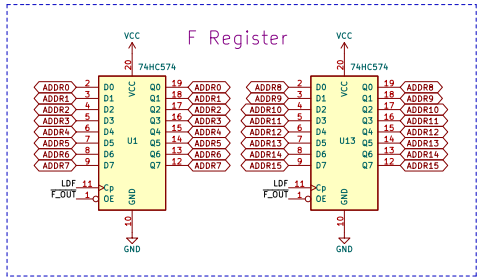


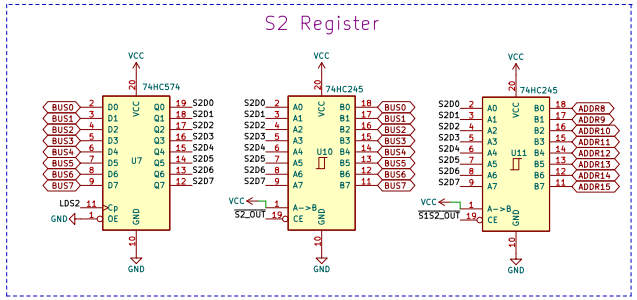


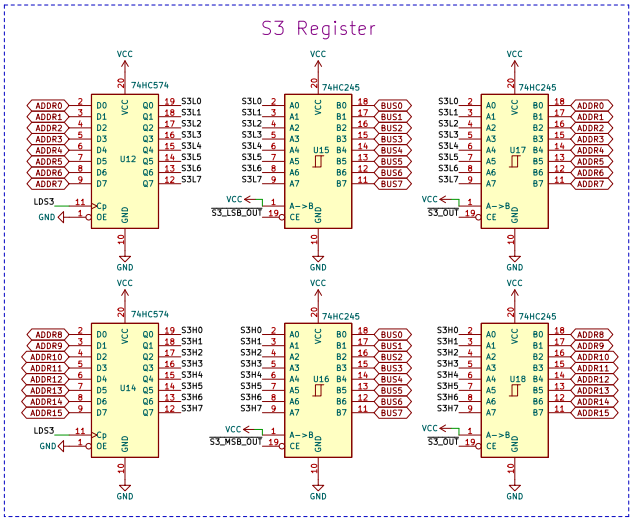












Control Word Truth Table

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.

GPR <-> External

MOV R, *	MOV *, R
0x81: LDC	0x80: NOOP
0x82: LDD	0x88: CO
0x83: LDS1	0x90: DO
0x84: LDS2	0x98: S1OB
0x85: LDF	0xa0: S2OB
0x86: LDS3	0xa8: S1S2OA
	0xb0: FO
	0xb8: S3O
	0xc0: S3LOB
	0xc8: S3HOB

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)

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. kicad (6.0.0-0)		Id: 1/1

Parts & Components List

Digital Logic, IC's

Part #	Description	Qty	Datasheet	Link
74HC574	8-Bit, Edge-Triggered, flip-flops	7	DATA	Mouser
74HC245	8-bit, Tri-State Transceiver	8	DATA	Mouser
74HC00	Quad, 2-input NAND gates	2	DATA	Mouser
74HC138	3 to 8 line, decoder	2	DATA	Mouser
-	Ceramic Capacitor	8	-	-
-	Resistor (Pull U/D)	1	-	-
-	Header Pins	-	-	-

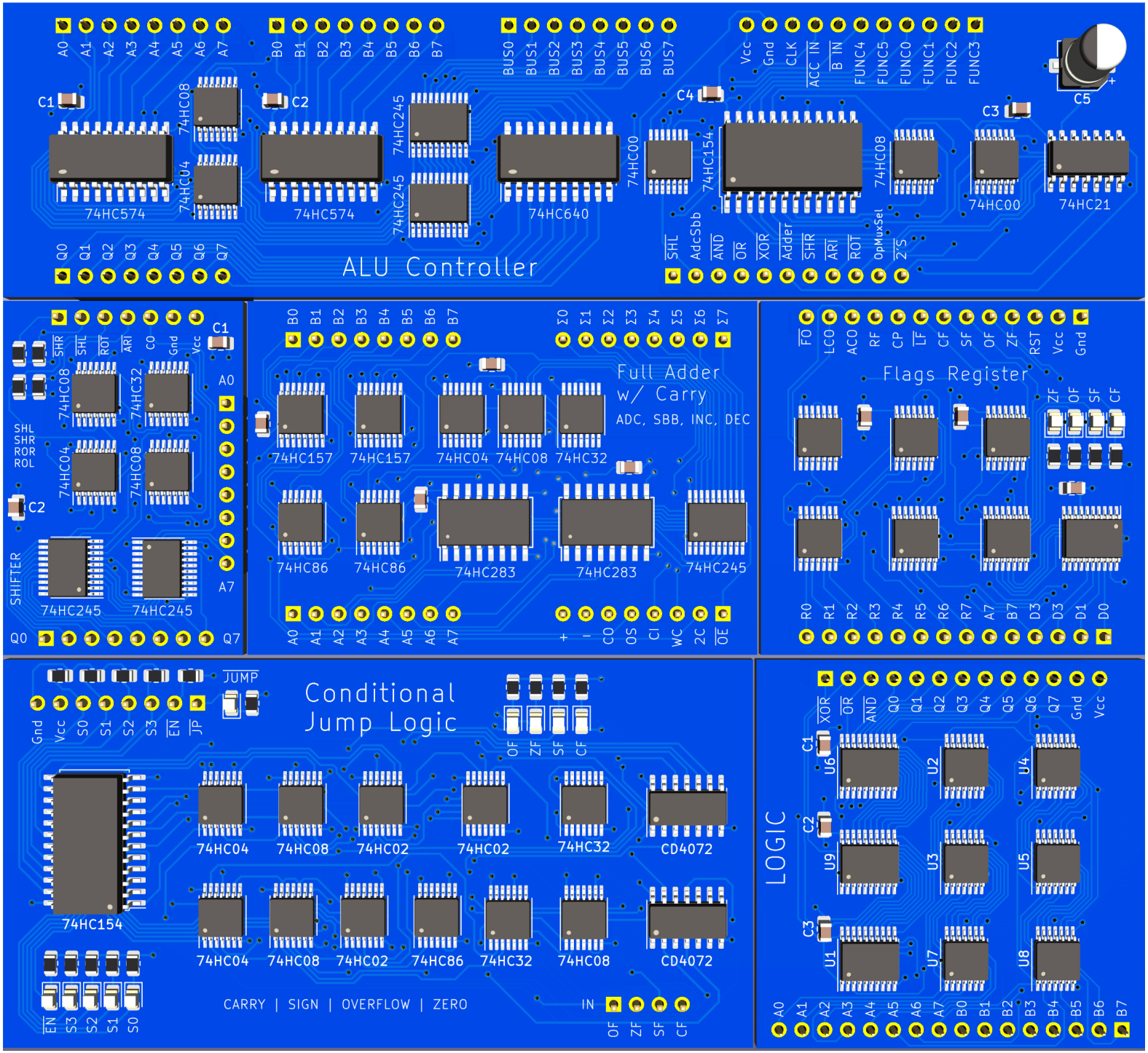
Arithmetic Logic Unit (ALU)

This portion of the CPU is responsible for all of the arithmetic, logic, and boolean functionality. It includes a *flags register* as well as a *conditional jump logic* module. Using a *6-bit control word*, and the *A and B* data busses, it can perform *22 different functions*. It is connected to the main 8-bit data bus to transfer values to the accumulator and operand registers as well as output the result of the given function. It is also used to restore the flags register during an interrupt return.

Power: 5V Vcc

Functionality

- Add w/ Carry
- Subtract w/ Borrow
- Invert
- Shift Left / Right
- Arithmetic Shift Left / Right
- Rotate Left / Right
- Compare / Test
- AND / NAND
- OR / NOR
- XOR / XNOR


ALU Controller
74HC574
74HC574
74HC640
74HC00
74HC21
74HC08
74HC04
74HC245
74HC154
Full Adder
w/ Carry
ADC, SBB, INC, DEC
74HC157
74HC86
74HC283
SHIFTER
Flags Register
Conditional
Jump Logic
74HC02
74HC32
CD4072
LOGIC
CARRY | SIGN | OVERFLOW | ZERO

Control & Instructions

ƒ	Description	Flags Affected	Hex Code
A	Asserts accumulator to the data bus		0x20
INC	Increment, A + 1	CF, SF, OF, ZF	0x25
DEC	Decrement, A – 1	CF, SF, OF, ZF	0x26
ADD	Addition, A + B	CF, SF, OF, ZF	0x21
ADC	Addition w/ Carry, A + B + Carry Flag	CF, SF, OF, ZF	0x22
SUB	Subtract, A – B	CF, SF, OF, ZF	0x23
SBB	Subtract w/ Carry, A – B – Carry Flag	CF, SF, OF, ZF	0x24
NOT	Invert A, ~A		0x30
AND	Boolean AND, A & B	SF, ZF, OF: Cleared, CF: Cleared	0x27
NAND	Boolean NAND, ~(A & B)	SF, ZF, OF: Cleared, CF: Cleared	0x35
OR	Boolean OR, A | B	SF, ZF, OF: Cleared, CF: Cleared	0x28

NOR	Boolean NOR, ~(A | B)	SF, ZF, OF: Cleared, CF: Cleared	0x38
XOR	Boolean XOR, A ^ B	SF, ZF, OF: Cleared, CF: Cleared	0x29
XNOR	Boolean XNOR, ~(A ^ B)	SF, ZF, OF: Cleared, CF: Cleared	0x39
SHL	Shift Left, A << 1	CF: Bit shifted out, SF, ZF, OF	0x2a
SHR	Shift Right, A >> 1	CF: Bit shifted out, SF, ZF, OF	0x2b
ASL	Signed Arithmetic Shift Left, A * 2	CF: Bit shifted out, SF, ZF, OF	0x2c
ASR	Signed Arithmetic Shift Riht, A / 2	CF: Bit shifted out, SF, ZF, OF	0x2d
ROR	Rotate Right	CF: Bit shifted out, SF, ZF, OF	0x2e
ROL	Rotate Left	CF: Bit shifted out, SF, ZF, OF	0x2f
CMP	Compare (SUB) - Without flags		0x23*
TST	Test (AND) - Without flags		0x27*

Accumulator and Operand Registers (8-bit)

- Latch Accumulator – Loads the value present on the data bus into the Accumulator (A) Register to be used by the various ALU functions.

- Latch Operand – Loads the value present on the data bus into the Operand (B) Register to be used by the various ALU functions.

Arithmetic Module

This module is built with two 4-bit ***Full Adders*** to perform addition on two 8-bit values. It uses **two's-complement** for Subtraction by XOR'ing the operand (B) with 0xff and adding 1. The accumulator can be incremented or decremented.

Arithmetic Control Truth Table

Instruction	Mux	Two's Comp	Carry In
A + B	0	0	0
A – B	0	1	1
A + B + Carry	0	0	Carry Flag Value
A – B – Carry	0	1	Carry Flag Value
INC (A + 1)	1	1	1
DEC (A – 1)	1	0	0

Arithmetic Examples

Instruction	A	B	CF	Result
ADD	0b0011	0b0001	n.c.	0b0100
ADC	0b0011	0b0001	1	0b0101
SUB	0b0011	0b0001	n.c.	0b0010
SBB	0b0011	0b0001	1	0b0001
INC	0b0011	n.c.	n.c.	0b0100
DEC	0b0011	n.c.	n.c.	0b0010

Logic Gate Module

This module is responsible for executing ***logical bitwise operations***. It contains only the AND, OR, and NOR functions. To acheive NAND, NOR, and XNOR the result is inverted at the ALU control unit.

Logic Gate Examples

Instruction	A	B	Result
AND	0b0101	0b0011	0b0001
NAND*	0b0101	0b0011	0b1110
OR	0b0101	0b0011	0b0111
NOR*	0b0101	0b0011	0b1000
XOR	0b0101	0b0011	0b0110
XNOR*	0b0101	0b0011	0b1001

Logic Gate Examples

**1 – Uses same function, however, the invert control bit is set to active. ie. and => nand*

Shift & Rotate Module

This module provides the ability to shift bits to the left or right. Can be done ***logically*** or ***arithmetically***. The logical shift is unsigned (ignores MSB), whereas Arithmetic Shift is signed, and maintains the most significant bit. This is the equivalent of multiplying/dividing by 2.

Shift & Rotate Truth Table

Instruction	SHR	SHL	Rotate	Arithmetic
SHL	1	0	1	1
SHR	0	1	1	1
ASL	1	0	1	0
ASR	0	1	1	0
ROR	0	1	0	1
ROL	1	0	0	1

Shift & Rotate Examples

Instruction	Operand	Result
SHL	0b10001001	0b00010010
SHR	0b10001001	0b01000100
ASL	0b10001001	0b10010010
ASR	0b10001001	0b11000100
ROR	0b10001001	0b11000100
ROL	0b10001001	0b00010011

Conditional Jump Logic

This module provides all of the conditions for calculating ***jumps*** based on ALU Flags. It uses a 4-bit control bus with enable, which is connected to the main CPU's microcode control logic, the ALU flags and outputs a single control line which is LOW if a jump is active.

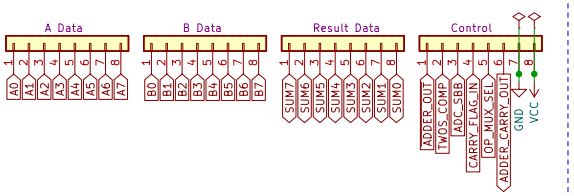
Instruction	Description	Flags	Hex Code
jp	Jump	n/a	0x0
jle / jng	Jump if Less Than or Equal / Jump Not Greater	ZF = 1 or SF <> OF	0x1
jg / jnle	Jump if Greater / Jump if Not Less Than or Equal	ZF = 0 and SF = OF	0x2
jge / jnl	Jump if Greater Than or Equal / Jump if Not Lower	SF = OF	0x3
jl / jnge	Jump if Less Than / Jump if Not Greater Than or Equal	SF <> OF	0x4
ja / jnbe	Jump if Above / Jump if Not Below	CF = 0 and ZF = 0	0x5
jbe / jna	Jump if Below / Jump if Not Above	CF = 1 or ZF = 1	0x6
jnb / jae / jnc	Jump if not below / Jump if above or equal / Jump if not carry	CF = 0	0x7

jb / jnae / jc	Jump if below / Jump if not above or equal / Jump if carry	CF = 1	0x8
jne / jnz	Jump if not equal / Jump if not zero	ZF = 0	0x9
je / jz	Jump if equal / Jump if zero	ZF = 1	0xa
jns	Jump if not sign	SF = 0	0xb
js	Jump if sign	SF = 1	0xc
jno	Jump if not overflow	OF = 0	0xd
jo	Jump if overflow	OF = 1	0xe

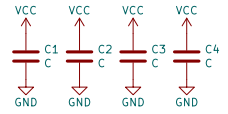
Arithmetic Module

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

CPU Connections

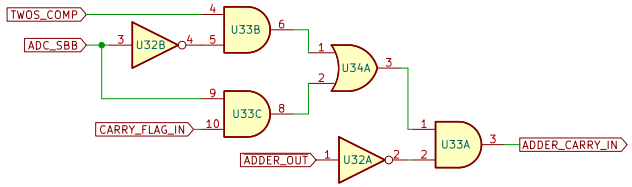


Decoupling



Arithmetic Carry In Logic

a) If twos complement, or
b) if ADC or SBB, and current carry flag status is active

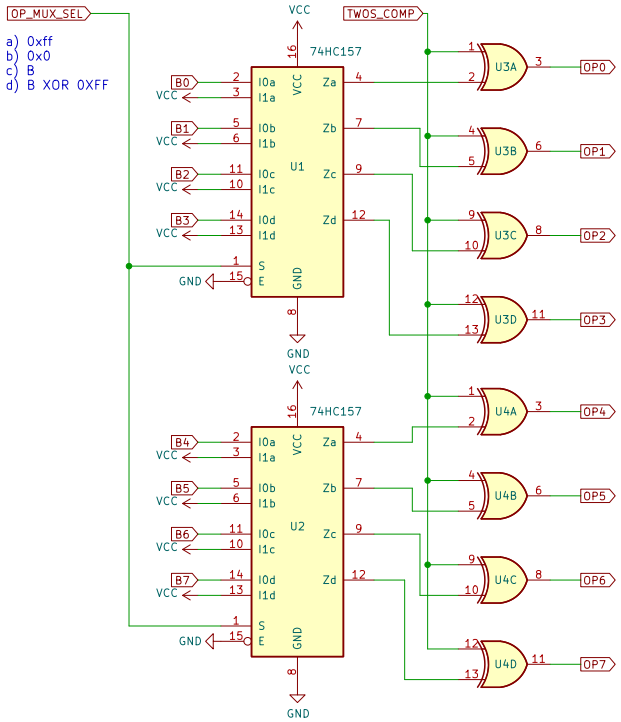


Control Word Decoding

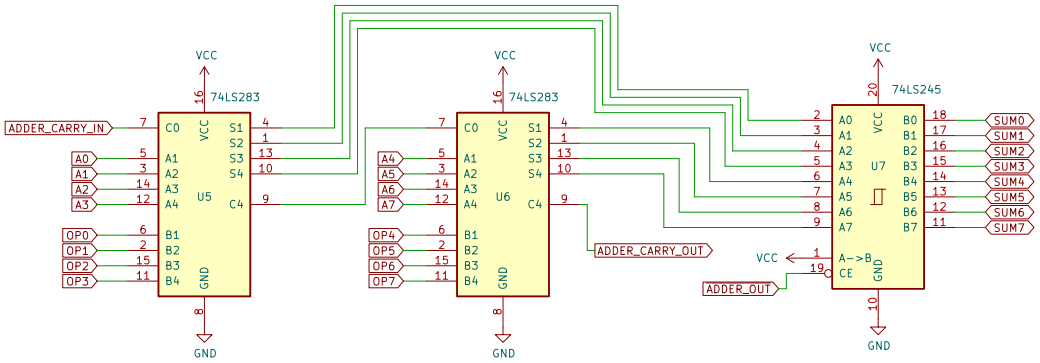
DEC A: MUX_SEL: 1, TC: 0, CI: 0
A – B: MUX_SEL: 0, TC: 1, CI: 1
A + B: MUX_SEL: 0, TC: 0, CI: 0
INC A: MUX_SEL: 1, TC: 1, CI: 1
A – B – CI: MUX_SEL: 0, TC: 1, CI: ?
A + B + Cf: MUX_SEL: 0, TC: 0, CI: ?

Operand Multiplexer

a) 0xff
b) 0x0
c) B
d) B XOR 0XFF

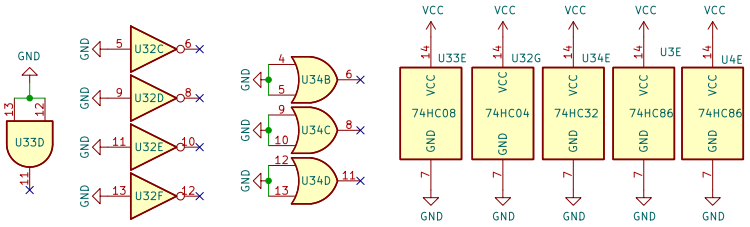


Full Adder w/ Carry



NOTE: Should have used 74HC283

Logic Power & Unused Gates



ADD / SUB / ADC / SBB / INC / DEC

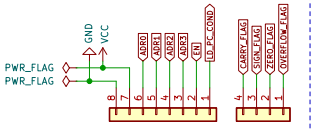
Sheet: /
File: Arithmetic.kicad_sch

Title: Arithmetic Module

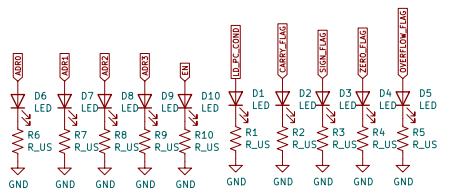
Size: User	Date:	Rev: 3
KiCad E.D.A. kicad (6.0.0-0)		Id: 1/1

ALU Conditional Jump Logic

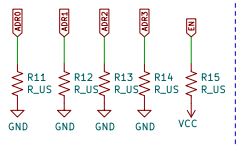
CPU Connections



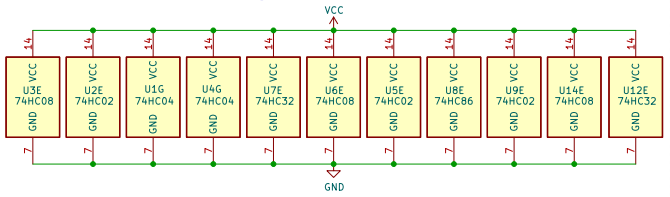
LED Indicators



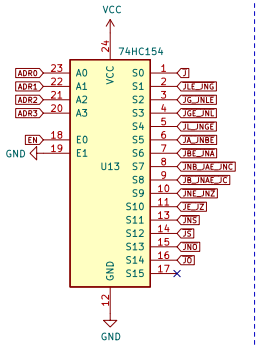
Pull U/D



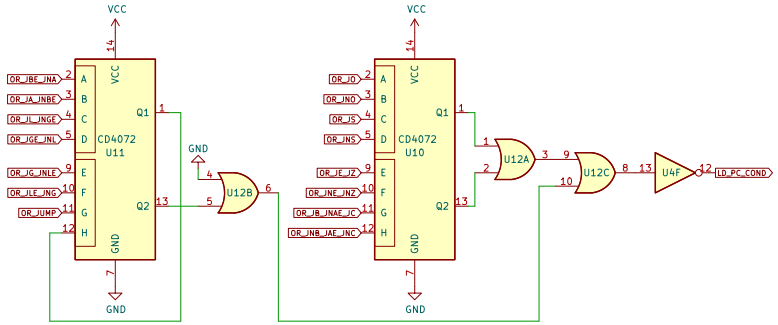
Logic Gates Power



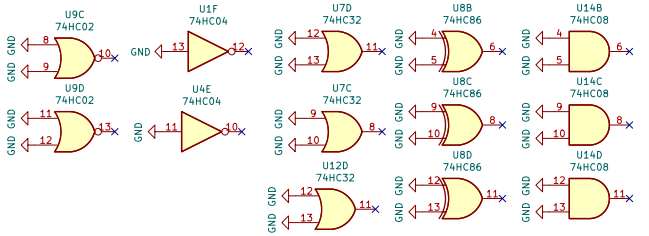
Jumps MUX



FINAL OR GATE FOR PC_LOAD control line

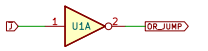


Unused Logic Gates



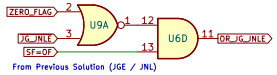
J — 0000

Unconditional Jump



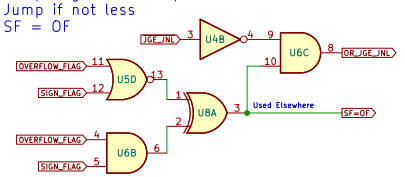
JG / JNLE — 0010

Jump if greater
Jump if not less or equal
ZF = 0 and SF = OF



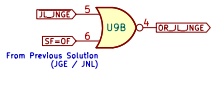
JGE / JNL — 0011

Jump if greater or equal
Jump if not less
SF = OF



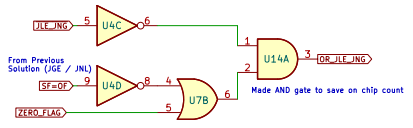
JL / JNGE — 0100

Jump if less
Jump if not greater or equal
SF <> OF



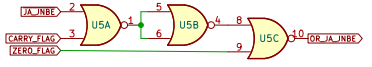
JLE / JNG — 0001

Jump if less or equal Jump if not greater
ZF = 1 or SF <> OF



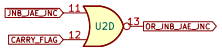
JA / JNBE — 0101

Jump if above
Jump if not below or equal
CF = 0 and ZF = 0



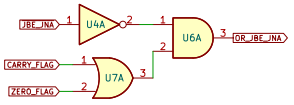
JNB / JAE / JNC — 0111

Jump if not below
Jump if above or equal
Jump if not carry
CF=0



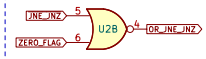
JBE / JNA — 0110

Jump if below or equal
Jump if not above
CF = 1 or ZF = 1



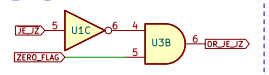
JNE / JNZ — 1001

Jump if not equal
Jump if not zero
ZF=0



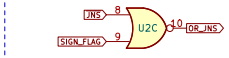
JE / JZ — 1010

Jump if equal,
Jump if zero
ZF=1



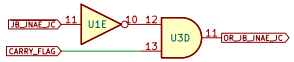
JS — 1011

Jump if NOT SIGN
SF=0



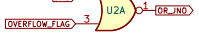
JB / JNAE / JC — 1000

Jump if below
Jump if not above or equal
jump if Carry
CF = 1



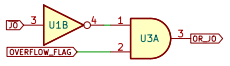
JNO — 1101

Jump if NOT OVERFLOW
OF=0



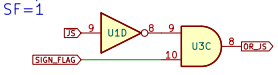
JO — 1110

Jump if OVERFLOW
OF=1



JS — 1100

Jump if SIGN
SF=1



Trilobyte CPU	
Sheet: /	
File: control-unit.kicad_sch	
Title: Conditional Jump Logic – ALU	
Size: User	Date: 2022-07-19
KiCad E.D.A. kicad (6.0.0-0)	Rev: v2
	Id: 1/1

Parts & Components List

Digital Logic, IC's

Part #	Description	Qty	Datasheet	Link
74HC574	8-Bit, Edge-Triggered, flip-flops	2	DATA	Mouser
74HC245	8-bit, Tri-State Transceiver	9	DATA	Mouser
74HC640	8-bit, Tri-State Transceiver w/ inverted outputs	1	DATA	Mouser
74HC154	4 to 16 line, decoder	2	DATA	Mouser
74HC00	Quad, 2-input NAND gates	2	DATA	Mouser
74HC04	6, 1-input, NOT (invert) gates	5	DATA	Mouser
74HC08	Quad, 2-input AND gates	11	DATA	Mouser
74HC21	Dual, 4-input AND gates	1	DATA	Mouser
74HC32	Quad, 2-input OR gates	7	DATA	Mouser
74HC86	Quad, 2-input XOR gates	6	DATA	Mouser
CD4072	Dual, 4-input OR gates	2	DATA	Mouser

74LS283	4-bit Binary Adder w/ Fast Carry	2	DATA	Mouser
Note: This is LS series, which has a HIGH output voltage of 3.6V, which feeds a 74HC245 that has an input HIGH voltage of 3.15V (at 4.5V VCC). I should have used 74HC283, however, it seems to be working fine for me - but this is def a point to evaluate with any issues				
74HC157	Quad 2-input multiplexer	3	DATA	Mouser
74HC173	Quad D-Type Flip-Flop, ThreeState	1	DATA	Mouser
	Polarized, Electrolyic Capacitor	1		
	Ceramic Capacitor	14		
	Resistor (LED)	14		
	Resistor (Pull U/D)	9		
	Light Emitting Diodes (LED)	14		
	Header Pins	-		

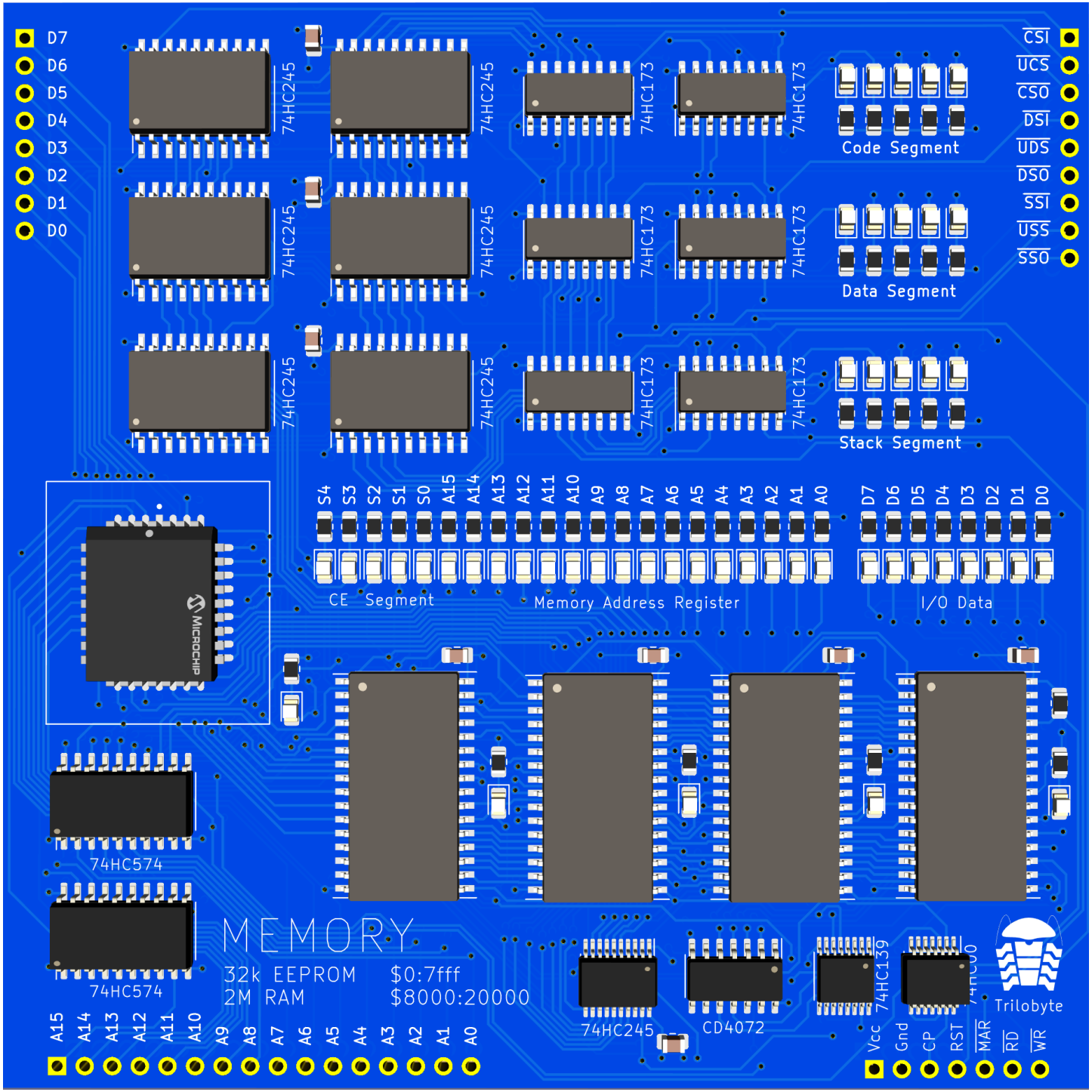
Memory (RAM/ROM)

The module provides the CPU with *2MB of volatile memory* and *32k of persistent EEPROM* memory. You are able to address locations larger than 16-bit by making use of the *memory segment registers*. With these, you can access the full 19-bits of address space and use multiple RAM chips..

Power: **5V Vcc**

Overview

- 1x EEPROM
  - 32k
  - 150ns
- 4x RAM
  - 512k
  - 25ns
- 3 Segment Registers
  - Code
  - Data
  - Stack
- Memory Address Register


D7
D6
D5
D4
D3
D2
D1
D0
74HC245
74HC173
Code Segment
Data Segment
Stack Segment
CSI
UCS
CSO
DSI
UDS
DSO
SSI
USS
SSO
S4 S3 S2 S1 S0 A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0
D7 D6 D5 D4 D3 D2 D1 D0
CE Segment
Memory Address Register
I/O Data
MICROCHIP
74HC574
MEMORY
32k EEPROM $0:7fff
2M RAM $8000:20000
74HC245
CD4072
74HC139
Vcc Gnd CP RST MAR RD WR
Trilobyte

Control & Instructions

Read & Write

Address	Segment [0..4]	Read	Write	Use Code	Use Data	Use Stack	Description
n.c.	n.c.	1	1	1	1	1	NOP
<= 0x7ffff	0x0	0	n.c.	0	1	1	Read Code EEPROM
<= 0x7ffff	0x0	0	n.c.	1	0	1	Read Data EEPROM
0x0...0x20000	0x1...0x1f	0	1	0	1	1	Read Code RAM
0x0...0x20000	0x1...0x1f	0	1	1	0	1	Read Data RAM
0x0...0x20000	0x1...0x1f	0	1	1	1	0	Read Stack RAM
>= 0x8000	n.c.	0	1	0	1	1	Read Code RAM
>= 0x8000	n.c.	0	1	1	0	1	Read Data RAM
>= 0x8000	n.c.	0	1	1	1	0	Read Stack RAM
0x0...0x20000	0x1...0x1f	1	0	1	0	1	Write Data RAM

>= 0x8000	n.c.	1i	0	1	0	1	Write Data RAM
0x0…0x20000	0x1…0x1f	1	0	1	1	0	Write Stack RAM
>= 0x8000	n.c.	1	0	1	1	0	Write Stack RAM

Memory Segment & Address Register

CSI	CSO	DSI	DSO	SSI	SSO	MAR	CLK	Description
1	1	1	1	1	1	1	_/	NOP
0	1	1	1	1	1	1	_/	Latches Code Segment
1	1	0	1	1	1	1	_/	Latches Data Segment
1	1	1	1	0	1	1	_/	Latches Stack Segment
1	0	1	1	1	1	1	_/	Assert Code Segment
1	1	1	0	1	1	1	_/	Assert Data Segment
1	1	1	1	1	0	1	_/	Assert Stack Segment
1	1	1	1	1	1	0	_/	Latch Memory Address Register

Memory Map

There is more detailed information about how Interrupts, Resets and the Entrypoint are handled can be found on the following sections: Instruction Cycle & Microcode section

Description	Address Range	Bytes	Location
Interrupt Vector Addresses [0..7]	$0:00000 … $0:00007	8	ROM
BIOS / PROGRAM	$0:00008 … $0:07fff	32k	ROM
Working Ram	$0:08000 … $0:fffff	491k	RAM 0
	$1:80000 … $1:fffff	524k	RAM 1
	$2:80000 … $2:fffff	524k	RAM 2
Working Ram / Stack	$3:80000 … $3:fffff	524k	RAM 3

Addressing Memory

Address Bits

Active Segment	Memory Address Register

RAM						EEPROM														
S4	S3	S2	S1	S0	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0

Code Segment (CS)

This is active when fetching the next instruction. It is only changed by a far jmp instruction and cannot be manipulated by mov.

```
// Example
jmp 0x2:0x8d32;
```

Data Segment (DS)

This is active during any read's or write's to Memory (excluding program code). This can be set using mov.

```
// Example
// Write
mov ds, 0x4;
// Active (uses the data segment value)
mov a, $8f40;
```

Stack Segment (SS)

This is active when executing a PUSH or POP instruction to and from the stack. This can be set using mov ss, 0xf

```
// Example
// Writes (sets the stack segment value)
mov ss, 0x2
pop ss, 0x2
// Active (uses the stack segment value)
push a;
pop a;
```

MEMORY MODULE w/segments

32K ROM | 2M RAM

ROM $0000 – $7FFF
RAM $8000 – $200000

Contents of this module include the following:
- 16-bit Memory Address Register (MAR)
- Memory Segments: Data, Code, Stack segments (DS, CS, SS)
- EEPROM & RAM

ROM address space is 15bits wide
RAM address space is 19bits wide

Since there is only 16bits in the MAR for addressing, the following logic explains how chip are selected:

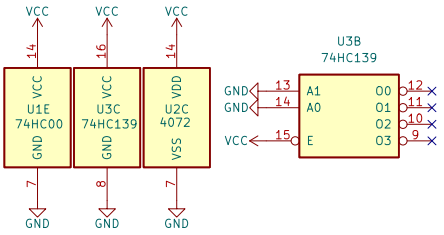
- If any segment is HIGH, or A15 is HIGH then EEPROM is disabled
- Otherwise, EEPROM is active and uses READ only
- The SEGMENT[0..2]'s are used to fill address space 16, 17, & 18
- SEGMENT[3,4] is decoded into 4 values and is used to target different RAM chips

Segment registers are loaded from the shared 8bit bus and can also be asserted back onto the BUS. A reason for asserting is so you can {push ds} onto the stack to store them during interrupts

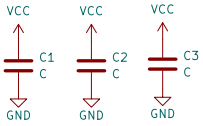
Addressing Key:
S: SEGMENT[3,4]
s: SEGMENT[0..2]
a: MAR[0..15]

0b_SS_sss_aaaa_aaaa_aaaa_aaaa

Logic Gate Power

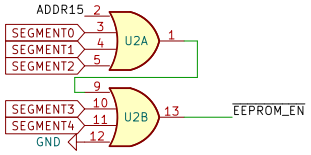


Smoothing Caps

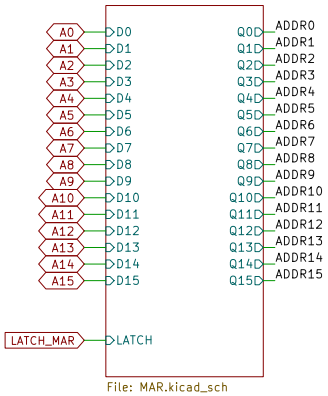


EEPROM ENABLE LOGIC

If any of the following is HIGH
segment[0..4] or A16
Then EEPROM access is disabled

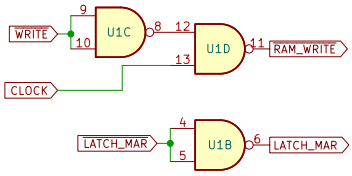


Memory Address Register



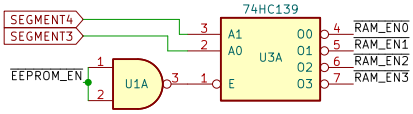
File: MAR.kicad_sch

Control Logic Active High/Lows

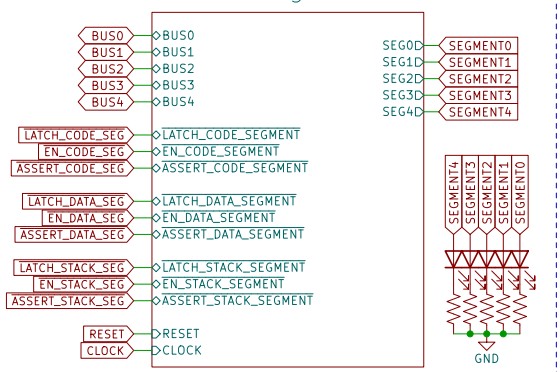


RAM SELECT LOGIC

If EEPROM is HIGH (disabled) then this MUX is enabled. It then uses SEGMENT[3,4] to choose a RAM enable line

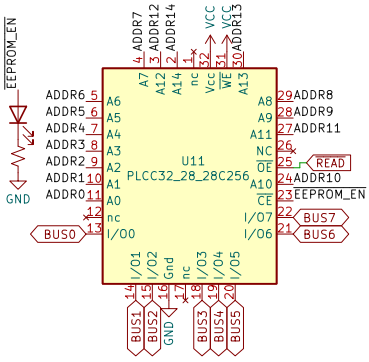


Data Segments

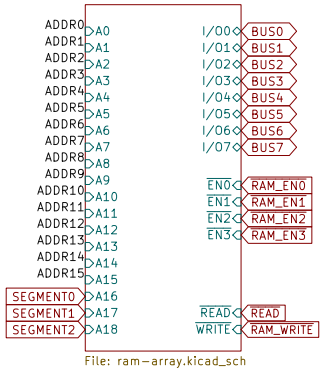


File: data-segments.kicad_sch

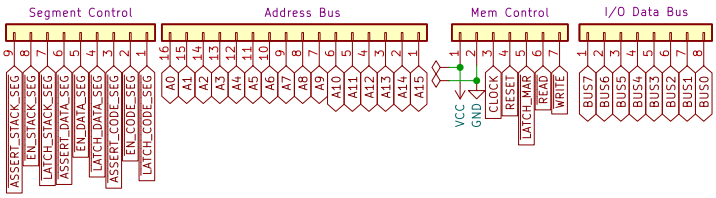
EEPROM



RAM Array



File: ram-array.kicad_sch



RAM $8000 – $20000
ROM $0000 – $7fff
32K ROM | 2M RAM
Trilobyte – theWickedWebDev/8-bit-computer

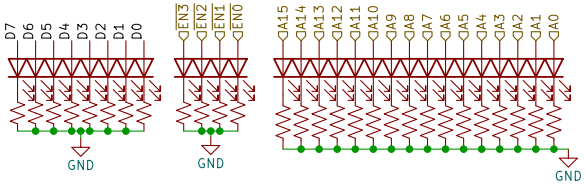
Sheet: /
File: memory.kicad_sch

Title: Memory Module

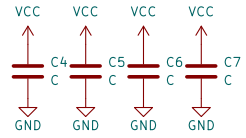
Size: User	Date: 2022-03-11	**Rev: 1.6**
KiCad E.D.A. kicad (6.0.0-0)		Id: 1/7

RAM Array Module

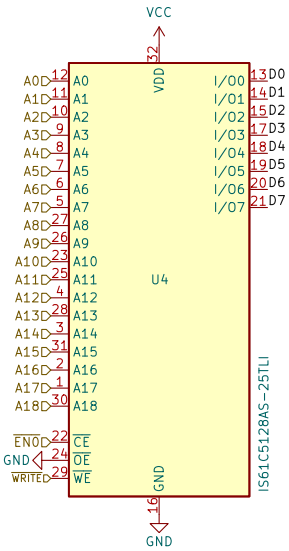
LED Indicators



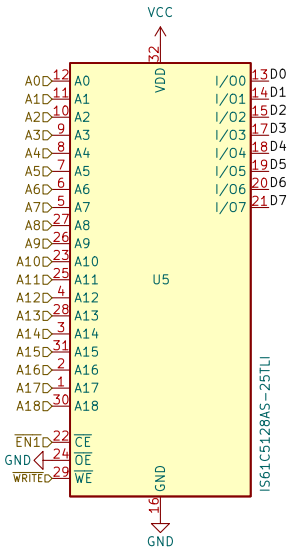
Smoothing Caps



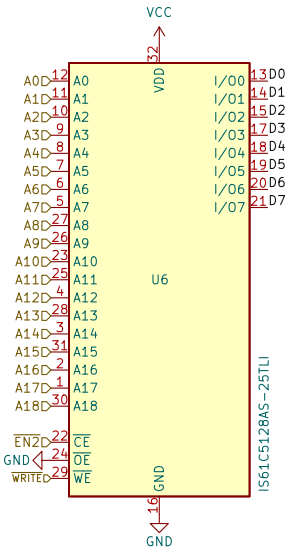
SRAM 512k 0



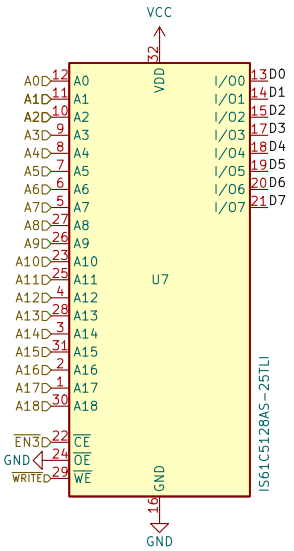
SRAM 512k 1



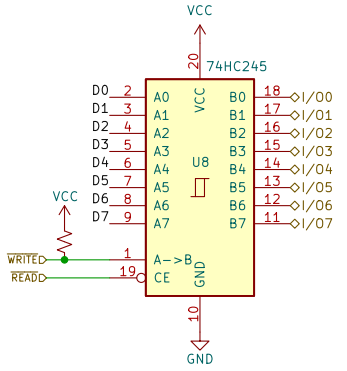
SRAM 512k 2



SRAM 512k 3

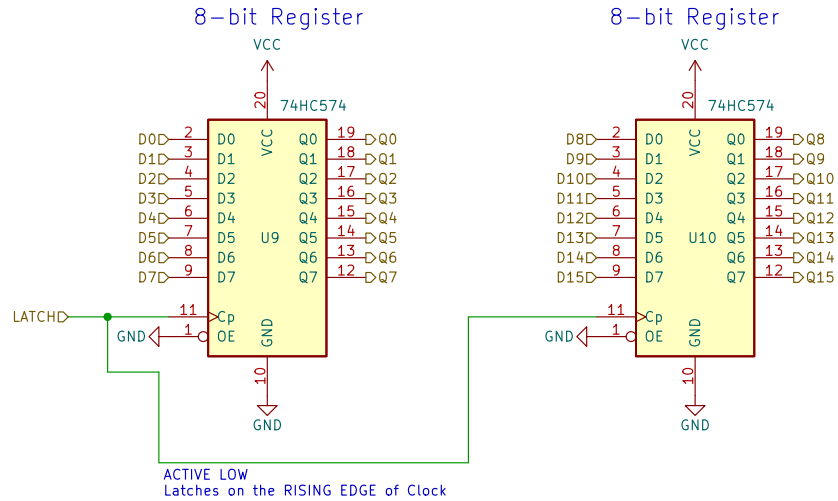


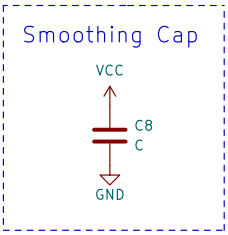
Data BUS Tranceiver



Sheet: /RAM Array/		
File: ram-array.kicad_sch		
Title:		
Size: User	Date:	**Rev:**
KiCad E.D.A. kicad (6.0.0-0)		Id: 2/7

MEMORY ADDRESS REGISTER

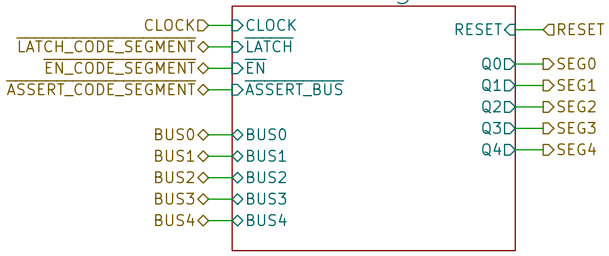




Sheet: /Memory Address Register/		
File: MAR.kicad_sch		
Title:		
Size: A5	Date:	**Rev:**
KiCad E.D.A. kicad (6.0.0-0)		Id: 3/7

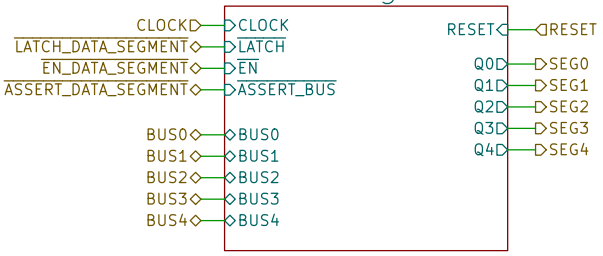
Segments

Code Segment



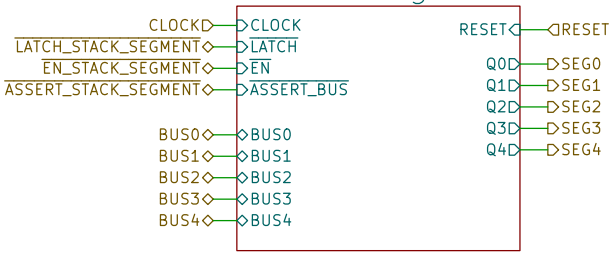
File: segment-register.kicad_sch

Data Segment



File: segment-register.kicad_sch

Stack Segment



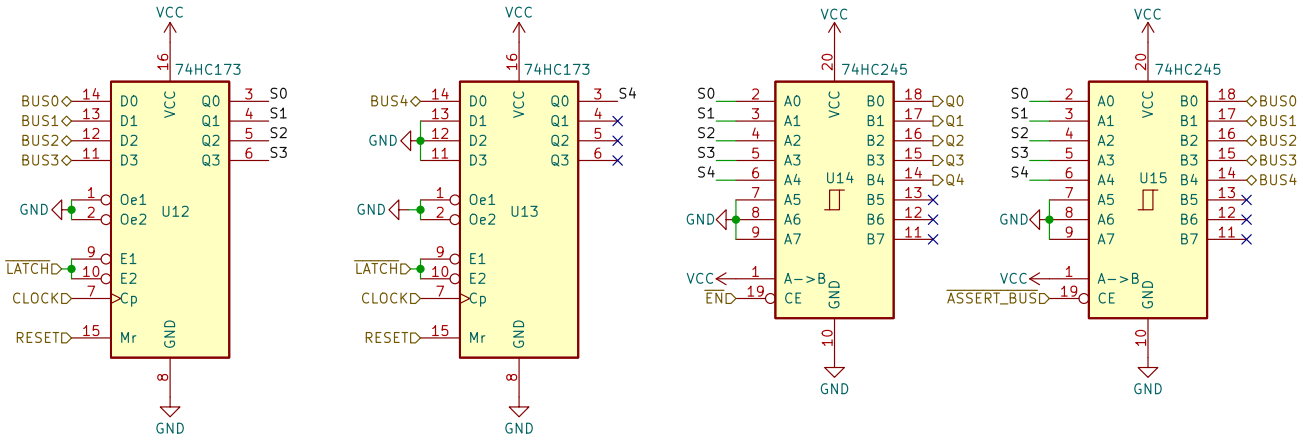
File: segment-register.kicad_sch

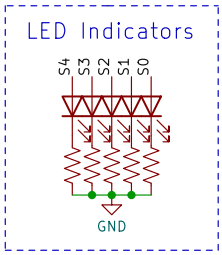
Sheet: /Data Segments/
File: data-segments.kicad_sch

Title:

Size: User	Date:	**Rev:**
KiCad E.D.A. kicad (6.0.0-0)		Id: 5/7

Segment Registers: Data, Code & Stack





Sheet: /Data Segments/Data Segment/		
File: segment-register.kicad_sch		
Title:		
Size: A5	Date:	**Rev:**
KiCad E.D.A. kicad (6.0.0-0)		Id: 6/7

Parts & Components List

Part #	Description	Qty	Datasheet	Link
IS61C5128AS-25TLI	512K x 8 HIGH-SPEED CMOS STATIC RAM	4	DATA	Mouser
AT28C256	256K (32K x 8) Paged Parallel EEPROM	1	DATA	Mouser
74HC574	8-Bit, Edge-Triggered, flip-flop's	2	DATA	Mouser
74HC173	Quad D-Type Flip-Flop	2	DATA	Mouser
74HC245	8-bit, Tri-State Transceiver	7	DATA	Mouser
CD4072	Dual 4-Input OR gates	1	DATA	Mouser
74HC139	4 to 16 line, decoder	1	DATA	Mouser
74HC00	Dual 4-input NAND gates	1	DATA	Mouser
-	Ceramic Capacitor	8	-	-
-	Resistor (LED)	49	-	-
-	Light Emitting Diodes (LED)	49	-	-
-	PLCC Socket, 32 pin - MOUSER	1	-	-
-	Header Pins	-	-	-

Instruction Cycle, Fetch

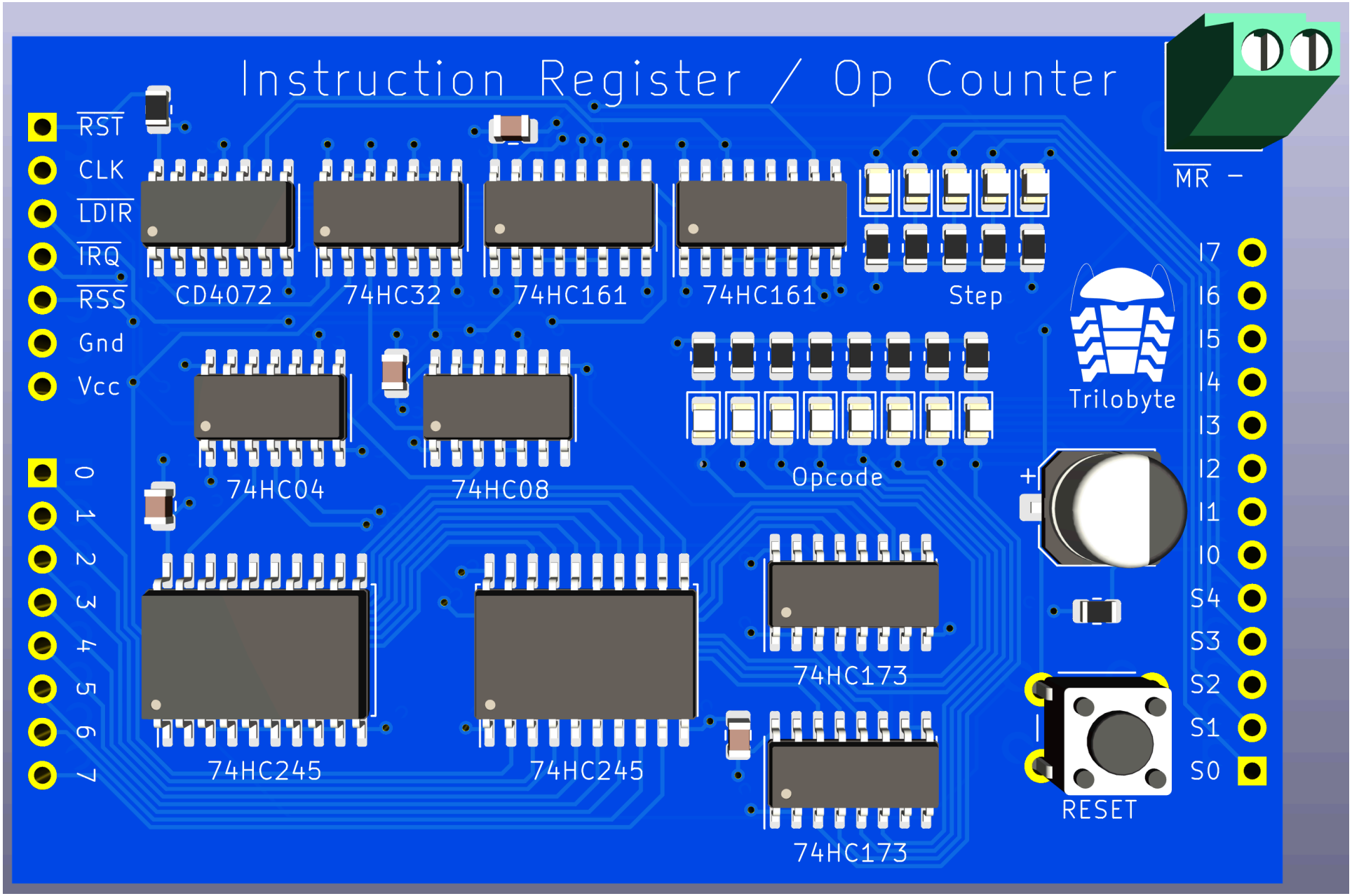
This module consists of a few parts: an 8-bit register, a 5-bit binary counter, and logic gates for multiplexing the input source. Together, their values combine to form the microcode address.

Inputs to this module include the 8-bit *memory bus*, the *interrupt request flag*, and a *reset line*.

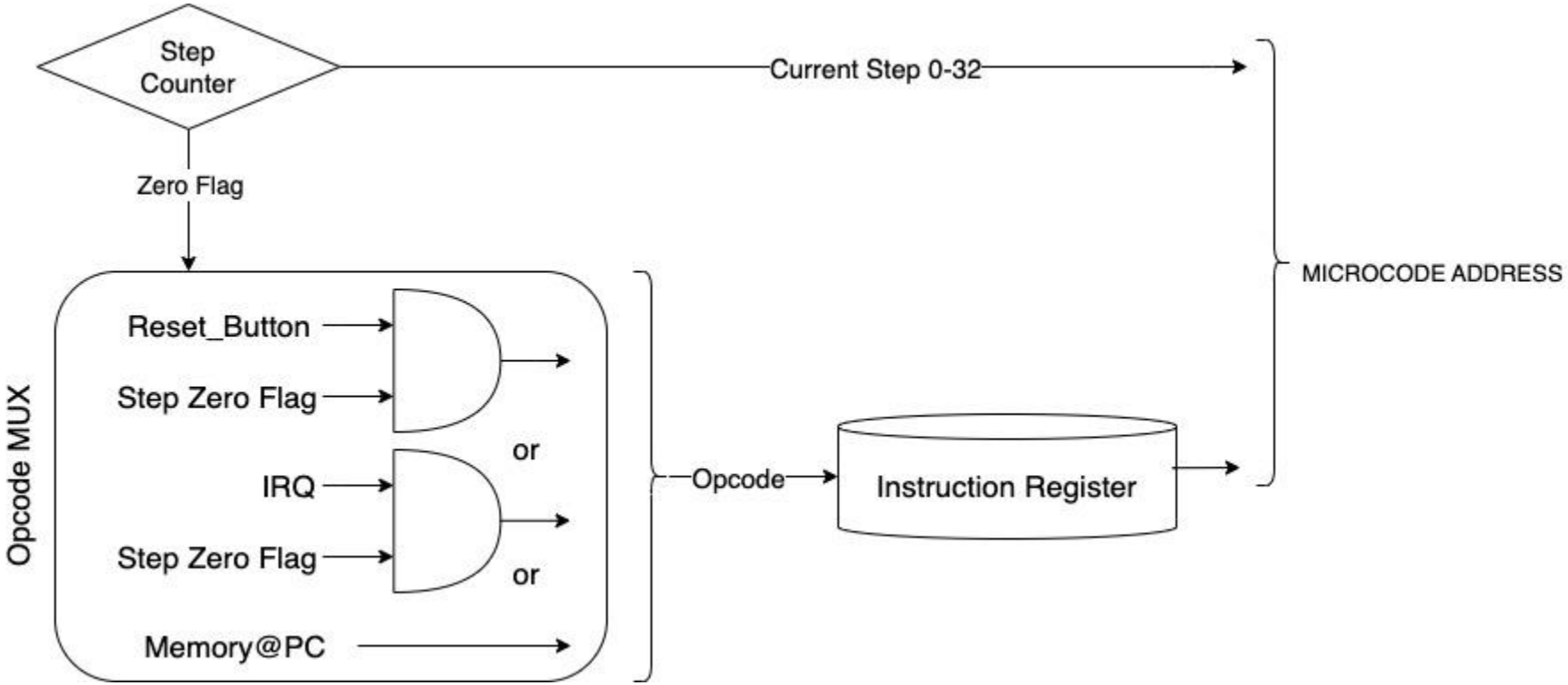
A momentary push-button is built into the board as another source for a CPU reset. Connected to this button is also a resistor-capacitor circuit that will fire off a reset signal once on power up.

- Instruction Register
- Step Counter
- Opcode Multiplexer
- Hardwired Reset Vector
- Hardwired IRQ Addresses

Power: 5V Vc


Instruction Register / Op Counter
RST
CLK
LDIR
IRQ
RSS
Gnd
Vcc
0 1 2 3 4 5 6 7
CD4072
74HC32
74HC161
74HC161
Step
MR
I7
I6
I5
I4
I3
I2
I1
I0
S4
S3
S2
S1
S0
Trilobyte
74HC04
74HC08
Opcode
74HC245
74HC245
74HC173
74HC173
RESET

Block Diagram



Instruction Register

Current Instruction Register

This is an *8-bit register* that stores the current instruction from the data bus (typically from memory).

The input of the instruction register has three potential values: a *hardwired reset vector,* an *interrupt request vector,* or an *instruction opcode*.

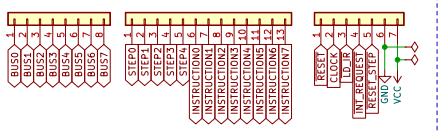
The reset or interrupt request addresses will only be latched when there is an active interrupt or the reset line was pulled low, as well as the *step zero flag* (SZF) is active. Otherwise, the instruction register will load its value from the data bus (memory).

Step Counter

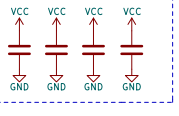
The step counter makes up the lower *5-bits* of the microcode address. This gives every instruction the ability to use up to *32 steps* to complete its cycle. Most instructions though however have much fewer steps needed. To avoid wasted clocks or NOPs, each instruction's microcode should include setting the *reset step counter* control line.

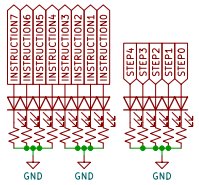
Instruction / Step Counter

CPU Connections



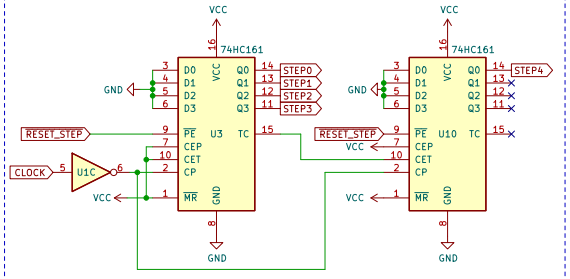
Smoothing Caps



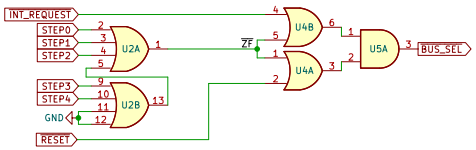


STEP COUNTER

Each instruction can have up to 32 steps

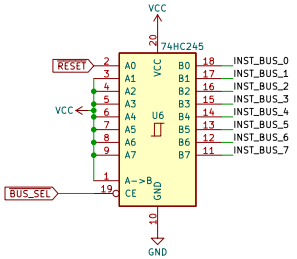


Bus Select Logic

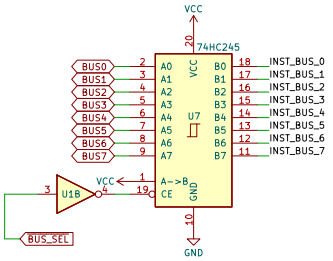


Instruction Bus

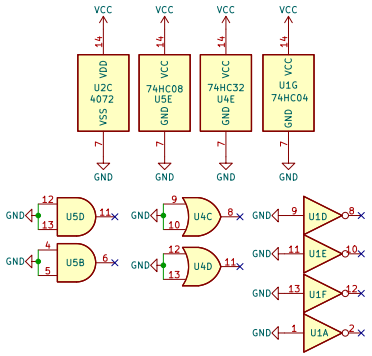
Hardwired Addresses Bus



Data Bus



Logic Gate Power



An instruction address can come from two places:
1. 8-bit Memory/Data Bus
2. Hardwired Address

When the Step counter is at 0x0, a $\overline{ZF}$ (zero flag) is set to active and is OR'ed with the inputs from INT_REQUEST AND RESET

If either $\overline{INT\_REQUEST}$ or $\overline{RESET}$ are 'low', and Zero Flag is LOW, then LD_INT_VECTOR is LOW. This causes the Instruction Source to switch to the hardwired addresses.

When a hardwired address is used, it is 0xff or 0xfe with the LSB being the current value of $\overline{RESET}$.

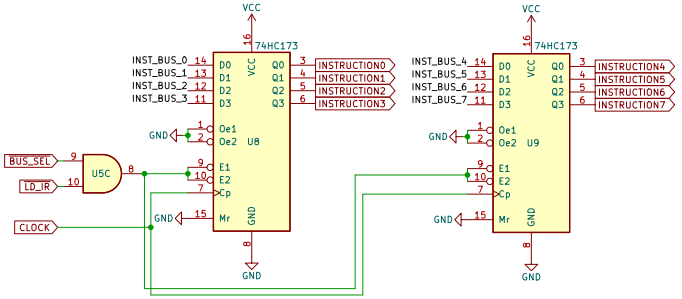
$\overline{RESET}$ Has the higher priority than $\overline{INT\_REQUEST}$ so if both are active, then reset operation is executed

0b11111111// 0xFF (Interupt Opcode)
0b11111110// 0xFE (RESET Opcode)

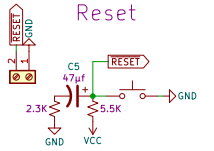
TRUTH TABLE

ZF	RESET	IRQ	SEL	DESC
1	nc	nc	1	Bus Data Asserted
0	1	1	1	Bus Data Asserted
0	0	nc	0	0xFE Asserted (Reset)
0	1	0	0	0xFF Asserted (IRQ Handler

Instruction Register



Reset



Reset RC circuit starts at logic level LOW for about a half a second. This ensures the CPU enters the reset mode on power-up

Trilobyte CPU	
Sheet: /	
File: inst-step-irq-decode.kicad_sch	
Title: Instruction Register & Step Counter	
Size: User	Date: 2022-07-20
KiCad E.D.A. kicad (6.0.0-0)	Rev: v1.0.0
	Id: 1/1

Parts & Components List

Part #	Description	Qty	Datasheet	Link
74HC173	8-Bit, Edge-Triggered, flip-flops	2	DATA	Mouser
74HC245	8-bit, Tri-State Transceiver	2	DATA	Mouser
74HC161	4-bit, Binary Counter	2	DATA	Mouser
74HC08	Quad, 2-input AND gates	11	DATA	Mouser
74HC32	Quad, 2-input OR gates	7	DATA	Mouser
74HC04	6, 1-input, NOT (invert) gates	5	DATA	Mouser
CD4072	Dual, 4-input OR gates	2	DATA	Mouser
-	Momentary Push Button	1	-	-
-	Pull Up Resistor	1	-	-
-	Capacitor (for switch)	1	-	-
-	Decoupling Capacitors	4	-	-
-	LEDs / Resistors	13	-	-