GP8B

Datasheet

-

Version 5.x

Copyright (C) 2020 Guillaume Guillet
Licensed under CERN-OHL-W v2 or later

This source describes Open Hardware and is licensed under the CERN-OHL-W v2 or later.

You may redistribute and modify this documentation and make products using it under the terms of the CERN-OHL-W v2 (https:/cern.ch/cern-ohl). This documentation is distributed WITHOUT ANY EXPRESS OR IMPLIED WARRANTY, INCLUDING OF MERCHANTABILITY, SATISFACTORY QUALITY AND FITNESS FOR A PARTICULAR PURPOSE.

Please see the CERN-OHL-W v2 for applicable conditions.

Table of contents

Description....3
  Features....3
CodeG....4
Blocks details....6
  Bus....6
  Instruction execution....9
    Sequencer....9
    Simple instructions....11
    Complex instructions....12
  Arguments....13
Fabrication....14
  Getting the PCB....14
  Getting the materials....14
  Soldering....14
Changes....15

Description

The GP8B or G_Processor8Bits is a home-made, from scratch, with no micro-controller, only made with logic circuits, 8bits processor.

The goal of this project is to create an 8bit computer community with the help of custom mecanical and electronic standards to build a nice and compatible environnement.

Features

- 64 kByte of RAM maximum.

- Using **ALUminium**[*1] in a CPLD for an 8bits ALU (arithmetic-logic unit).

- A simple SPI implementation.

- 16 writable bits for peripheral communication.

- 16 readable bits for peripheral communication.

- Following the custom standard **SPS1**[*2].

- Following the **CodeG binary rev1**[*2] standard.

- Accessible debug pins and Jtag for the CPLD.

- 50Mhz theorical maximum frequency.

[*1] ALUminium is an VHDL 8bits ALU code that you can find here :

https://github.com/JonathSpirit/ALUminium

[*2] You can find the standard here :

https://github.com/JonathSpirit/GComputer_standard

CodeG

GP8B execute the revision 1 of the homemade binary language called CodeG :

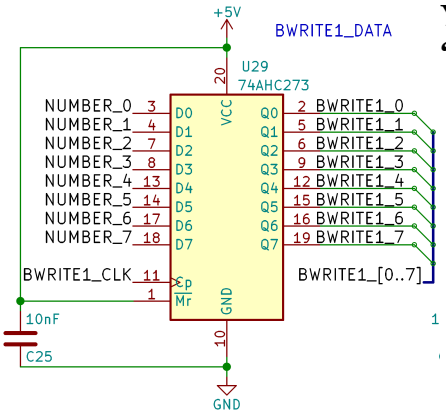
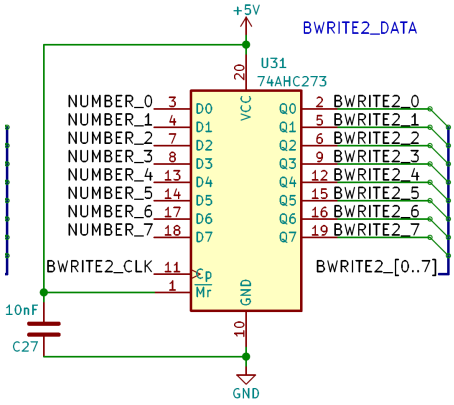
CodeG binary revision 1			
Instruction selection			
Opcode	**Hex**	**Name**	**Description**
xxx0’0000	**0x00**	BWRITE1_CLK	Apply value to bus “write 1”.
xxx0’0001	**0x01**	BWRITE2_CLK	Apply value to bus “write 2”.
xxx0’0010	**0x02**	BPCS_CLK	Apply value to bus “peripheral CS”.
xxx0’0011	**0x03**	OPLEFT_CLK	Apply value to the left operation.
xxx0’0100	**0x04**	OPRIGHT_CLK	Apply value to the right operation.
xxx0’0101	**0x05**	OPCHOOSE_CLK	Apply value for choosing the operation.
xxx0’0110	**0x06**	PERIPHERAL_CLK	Send a clock pulse to the peripheral.
xxx0’0111	**0x07**	BJMPSRC1_CLK	Apply value to bus “jump source 1”.
xxx0’1000	**0x08**	BJMPSRC2_CLK	Apply value to bus “jump source 2”.
xxx0’1001	**0x09**	BJMPSRC3_CLK	Apply value to bus “jump source 3”.
xxx0’1010	**0x0A**	JMPSRC_CLK	Jump to the address of “jump source”.
xxx0’1011	**0x0B**	BRAMADD1_CLK	Apply value to bus “RAM address 1”.
xxx0’1100	**0x0C**	BRAMADD2_CLK	Apply value to bus “RAM address 2”.
xxx0’1101	**0x0D**	SPI_CLK	Send a clock pulse for the SPI.
xxx0’1110	**0x0E**	BCFG_SPI_CLK	Apply value to the config bus SPI.
xxx0’1111	**0x0F**	STICK	Simple tick, do nothing (delay).
xxx1’0000	**0x10**	IF	Conditional instruction.
xxx1’0001	**0x11**	IFNOT	Conditional instruction inverted.
xxx1’0010	**0x12**	RAMW	Write value to the actual address of the memory.
xxx1’0011	**0x13**	UOP	Undefined operation.
xxx1’0100	**0x14**	UOP	Undefined operation.
xxx1’0101	**0x15**	UOP	Undefined operation.
xxx1’0110	**0x16**	UOP	Undefined operation.
xxx1’0111	**0x17**	LTICK	Long tick, do nothing (delay).

CodeG binary revision 1			
Argument selection			
Opcode	**Hex**	**Name**	**Description**
000x'xxxx	**0x00**	SRCVALUE	The value stored to the source.
001x'xxxx	**0x20**	BREAD1	The value of the read bus 1.
010x'xxxx	**0x40**	BREAD2	The value of the read bus 2.
011x'xxxx	**0x60**	OPRESULT	The value of the operation result (ALU).
100x'xxxx	**0x80**	RAMVALUE	The value of the processor memory.
101x'xxxx	**0xA0**	SPI	The value of the SPI.
110x'xxxx	**0xC0**	EXT_1	External value 1.
111x'xxxx	**0xE0**	EXT_2	External value 2.

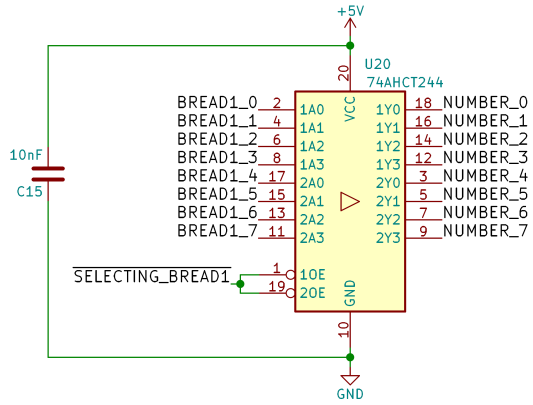
Blocks details

Bus

Details of essential/fonctionnal bus are listed here :

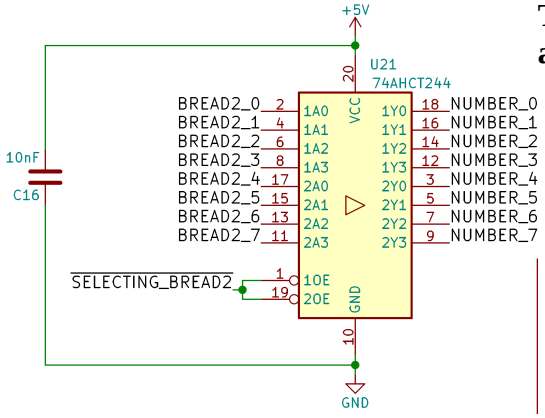
Bus name	Size (bits)	Direction	Description
BWRITE1	**8**	**out**	Bus for external peripheral communication.
	**A D-latch chip control the write bus 1.** **You can apply a value with the instruction “BWRITE1_CLK”.**		
BWRITE2	**8**	**out**	Bus for external peripheral communication.
	**A D-latch chip control the write bus 2.** **You can apply a value with the instruction “BWRITE2_CLK”.**		

Bus name	Size (bits)	Direction	Description
BREAD1	**8**	**in**	Bus for external peripheral communication.



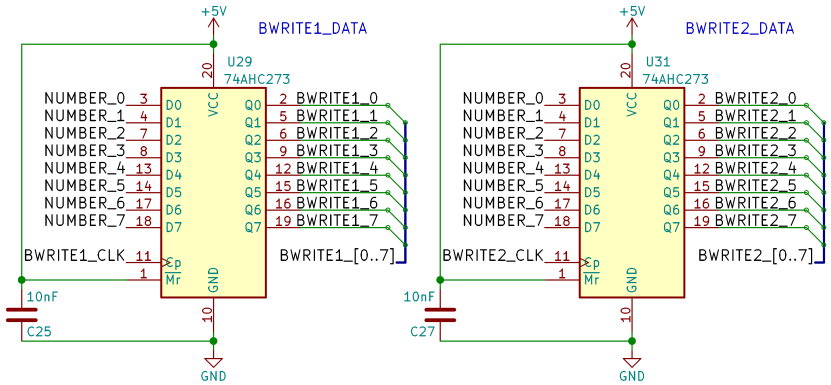
The readable bus 1 can be directly applied as an argument into the “NUMBER” bus.

BREAD2	8	in	Bus for external peripheral communication.



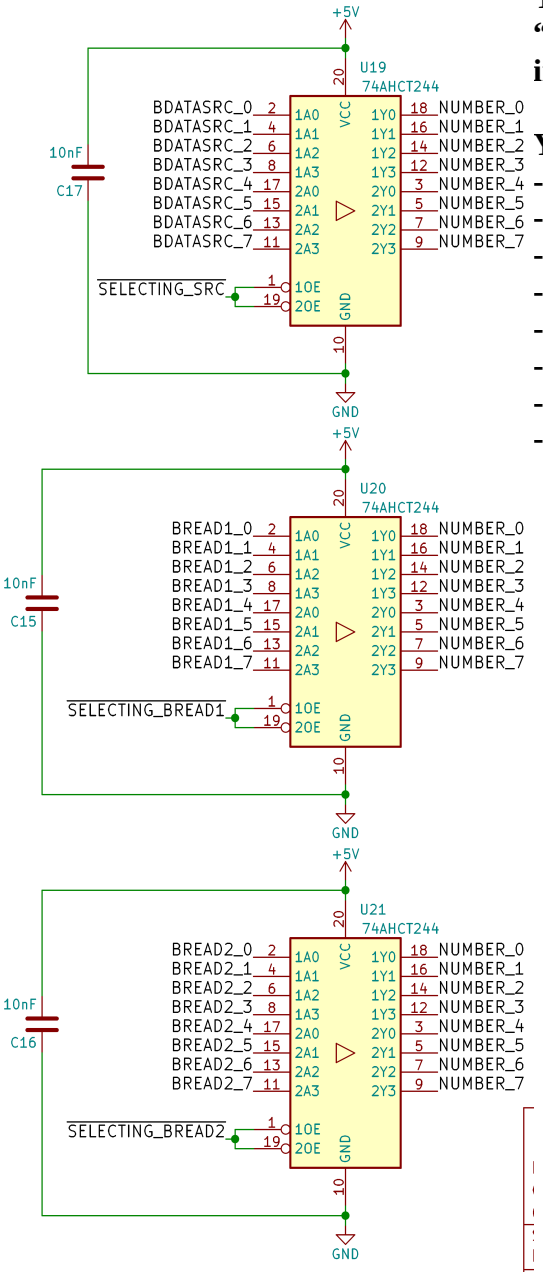
The readable bus 2 can be directly applied as an argument into the “NUMBER” bus.

NUMBER	8	in/out	Main argument bus.



The “NUMBER” bus is applied almost everywhere. With it, you can control most of the register like the writeable bus 1 or 2.

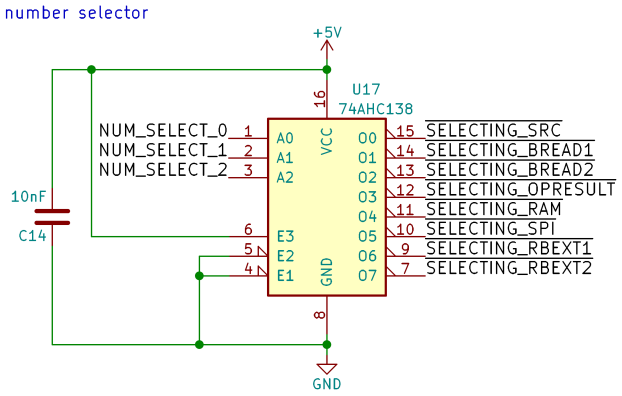
NUMBER	8	in/out	Main argument bus.



You can choose the value that goes into the "NUMBER" bus by changing the last 3bits of the instruction code.

You can choose 8 differents value :

- **The source.**
- **BREAD1**
- **BREAD2**
- **Operation result**
- **RAM**
- **SPI**
- **External bus 1**
- **External bus 2**

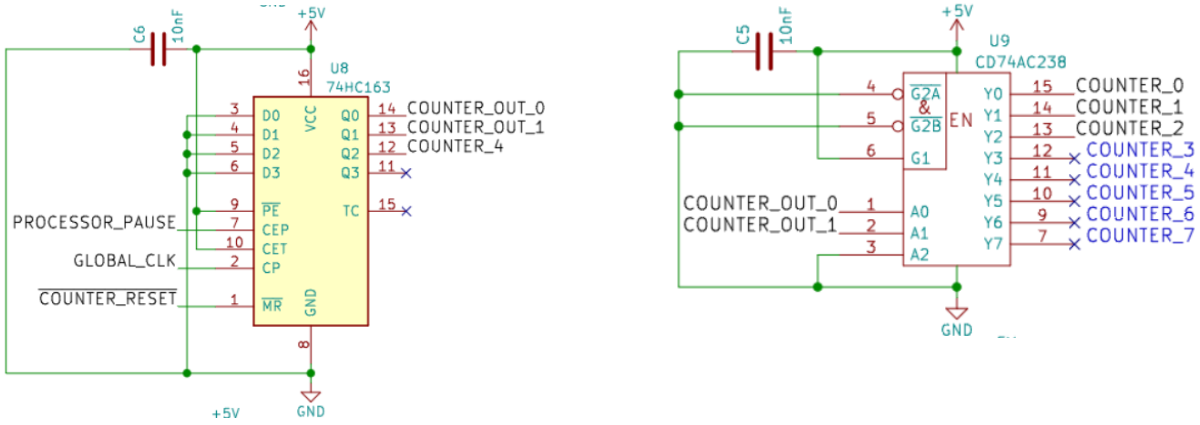


Instruction execution

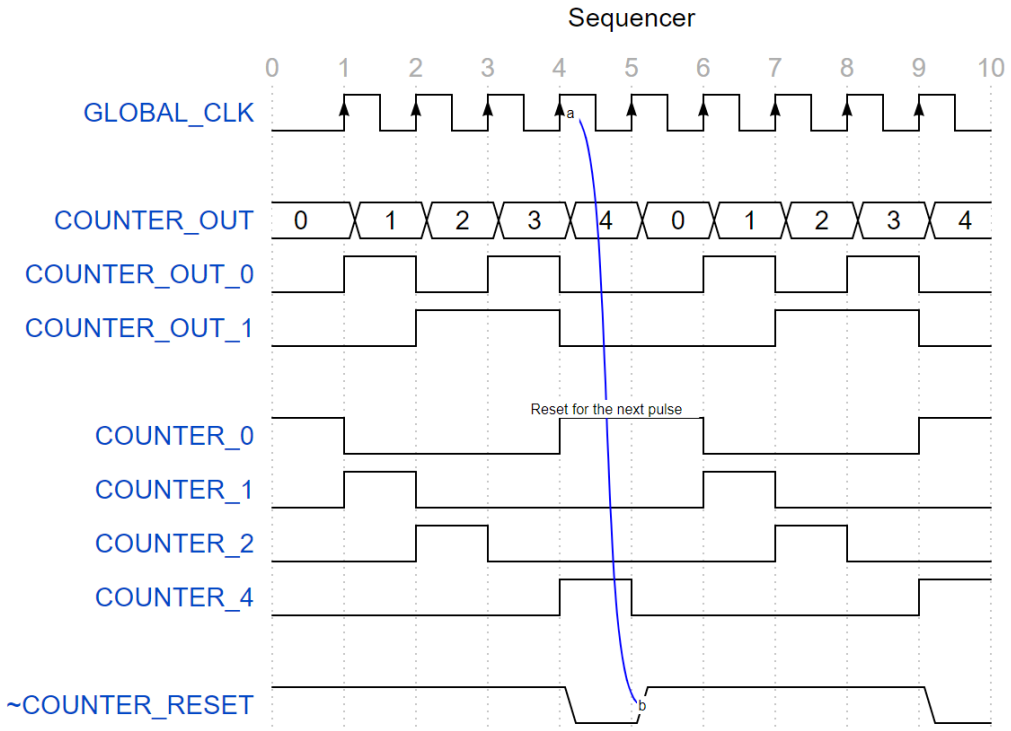
Sequencer

The first main component is a 4bits counter that will receive the “GLOBAL_CLK” of the motherboard.

The goal of this chip is to provide, with the help of a demux, all the pulse required for executing instruction sequentially.



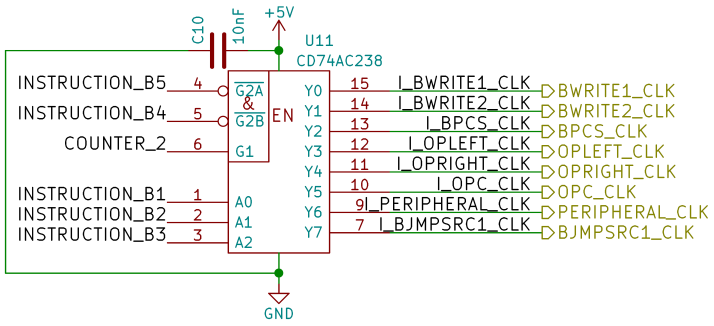
The 4bits count from 0 to 4, the fifth clock reset the counter to 0:

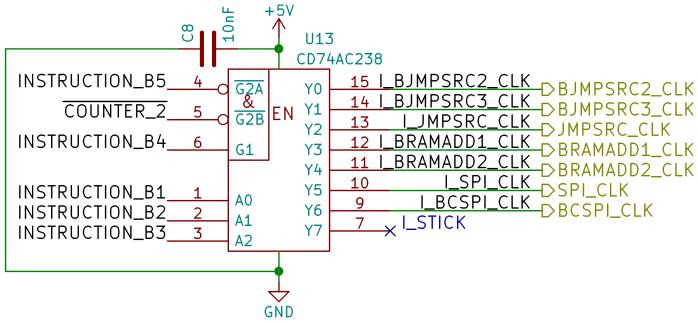


Every pulse from the sequencer represent a stat/action:

Sequence value	Action/Stat	Counter reset?	Next address?	Instruction latched?
0	**Sync bit**	**No**	**No**	**No**
At this stat, no instruction are executed and the SYNC_BIT is pulled high, this serve only for syncing with an interface like a debugger and prepare the execution of the next instruction.				
1	**Instruction set**	**No**	**Yes**	**Yes**
A pulse have been made to ‘U7’ (8bit D-Latch) to save the instruction to execute and the next address is requested (the argument).				
2	**Execution**	**Yes/No**	**Yes/No**	**No**
“Simple instruction”: One pulse is made to execute the instruction and a reset/next address is requested. **“Complex instruction”:** The first part of the instruction is executed with no reset: **IF/IFNOT:** If the argument is valid (>0 or =0), then the next instruction is skipped (usually a JMPSRC_CLK instruction) by requesting the next address. **RAMW:** The “output enable” pin is pulled high for write preparation. Noting is executed for UOP/LTICK instructions.				
3	**Waiting**	**No**	**No**	**No**
At this stat, the processor do nothing (this stat is only useful for write preparation to the memory).				
4	**RAM Write or END**	**Yes**	**Yes**	**No**
Depending on the instruction: **RAMW:** The RAM write the requested data by pulling low the “write enable” pin. At the end, a reset and the next address is requested.				

Simple instructions



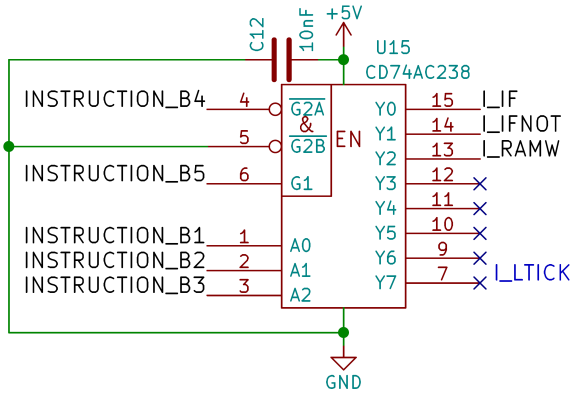


Simple instruction consist of just one pulse and an argument.

Every simple instruction is executed in 3 rising-edges from GLOBAL_CLK.

Opcode	Hex	Name	Description
xxx0'0000	**0x00**	BWRITE1_CLK	Apply value to bus "write 1".
xxx0'0001	**0x01**	BWRITE2_CLK	Apply value to bus "write 2".
xxx0'0010	**0x02**	BPCS_CLK	Apply value to bus "peripheral CS".
xxx0'0011	**0x03**	OPLEFT_CLK	Apply value to the left operation.
xxx0'0100	**0x04**	OPRIGHT_CLK	Apply value to the right operation.
xxx0'0101	**0x05**	OPCHOOSE_CLK	Apply value for choosing the operation.
xxx0'0110	**0x06**	PERIPHERAL_CLK	Send a clock pulse to the peripheral.
xxx0'0111	**0x07**	BJMPSRC1_CLK	Apply value to bus "jump source 1".
xxx0'1000	**0x08**	BJMPSRC2_CLK	Apply value to bus "jump source 2".
xxx0'1001	**0x09**	BJMPSRC3_CLK	Apply value to bus "jump source 3".
xxx0'1010	**0x0A**	JMPSRC_CLK	Jump to the address of "jump source".
xxx0'1011	**0x0B**	BRAMADD1_CLK	Apply value to bus "RAM address 1".
xxx0'1100	**0x0C**	BRAMADD2_CLK	Apply value to bus "RAM address 2".
xxx0'1101	**0x0D**	SPI_CLK	Send a clock pulse for the SPI.
xxx0'1110	**0x0E**	BCFG_SPI_CLK	Apply value to the config bus SPI.
xxx0'1111	**0x0F**	STICK	Simple tick, do nothing (delay).

Complex instructions



Complex instructions are instructions that takes 5 rising-edges from GLOBAL_CLK to execute.

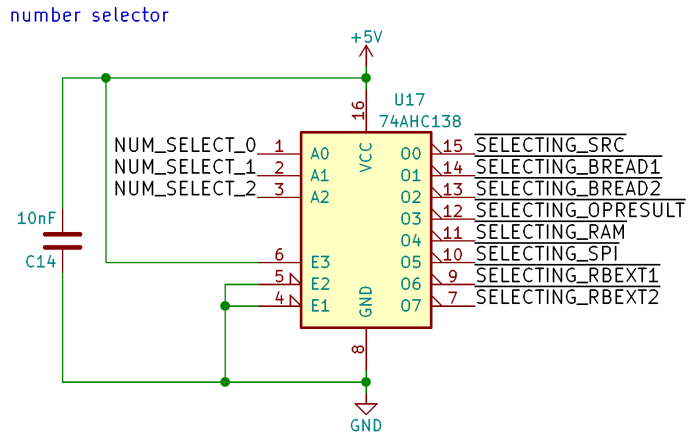
Opcode	Hex	Name	Description
xxx1'0000	**0x10**	IF	Conditional instruction.
xxx1'0001	**0x11**	IFNOT	Conditional instruction inverted.
xxx1'0010	**0x12**	RAMW	Write value to the actual address of the memory.
xxx1'0111	**0x17**	LTICK	Long tick, do nothing (delay).

The RAMW instruction consist by pulling OE low when counter is at 2, and send a WE pulse when counter is at 4.

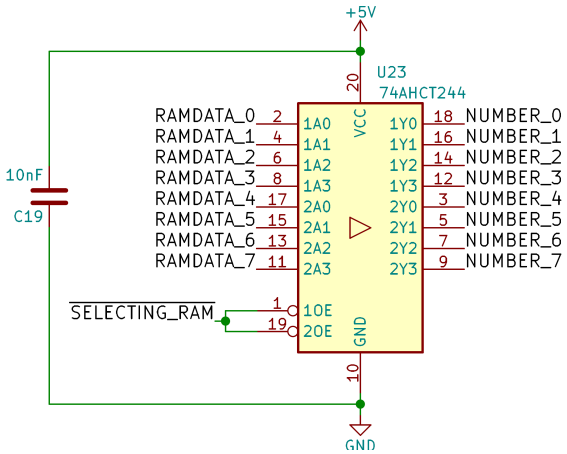
The conditional instruction consist of checking the argument when counter is at 2, if the argument is valid, the next instruction (usually a jump) is skipped else the next instruction is executed.

Arguments

The argument for an operation is selected by changing the last 3bits of the opcode :



The 3bits, once latched, are connected to the demux “U17”.



Every “SELECTING” line control a 8bits buffer with tri-state output and every buffer is connected to the argument bus “NUMBER”.

Opcode	Hex	Name	Description
000x’xxxx	**0x00**	SRCVALUE	The value stored to the source.
001x’xxxx	**0x20**	BREAD1	The value of the read bus 1.
010x’xxxx	**0x40**	BREAD2	The value of the read bus 2.
011x’xxxx	**0x60**	OPRESULT	The value of the operation result (ALU).
100x’xxxx	**0x80**	RAMVALUE	The value of the processor memory.
101x’xxxx	**0xA0**	SPI	The value of the SPI.
110x’xxxx	**0xC0**	EXT_1	External value 1.
111x’xxxx	**0xE0**	EXT_2	External value 2.

Fabrication

Getting the PCB

Fabrication files can be found in the folder : “documents/GP8B_gerbert”.

For JLCPCB or other manufacturers, you can zip the fabrication folder and send it to them directly.

Getting the materials

A bill of materials can be found here : “documents/GP8B_materials”. Note that if you want to change the supplier, you have to make sur that the component is fully compatible mostly for the package.

Soldering

If you solder the board manually, a microscope can be useful for small component like the CPLD.

Follow the bill of materials and the PCB placement file to solder the board.

I suggest you start with the CPLD and finish with the connectors.

Changes

Revision 0:

initial release