ipPROCESS

ARCHITECTURE SPECIFICATION

32-bit uDLX Core Processor

Universidade Federal da Bahia

Versão: 1.0

GNU LGPL License

This file is part of uDLX (micro-DeLuX) soft IP-core.

uDLX is free soft IP-core: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

uDLX soft core is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with uDLX. If not, see

<http://www.gnu.org/licenses/>.

Hist???rico de Revis???es

Date	Description	Author(s)
04/27/2014	Conception	Jo???o Carlos Bittencourt
04/30/2014	Instruction layout description	Jo???o Carlos Bittencourt
05/09/2014	• Text revision; • Update diagrams and instruction layout; • Update instruction fetch I/O definitions; • Missing pictures inclusion; • Include memory access and write back pin/port definitions;	Jo???o Carlos Bittencourt
04/13/2014	Missing pictures	Jo???o Carlos Bittencourt
04/15/2014	• Fix instruction fetch pin definitions; • Include instruction fetch datapath; • Include pipeline registers definitions; • Fix memory access stage pin/port definitions; • Include write back data path;	Jo???o Carlos Bittencourt
04/15/2014	Add execute block diagram	Igo Amauri Luz
04/16/2014	• Add branch prediction signal to ID and IF blocks and pin definitions; • Add table for pin definitions in Execute stage • Add branch prediction signal in pipeline registers definitions; • Include architecture interface figure;	Jo???o Carlos Bittencourt

CONTENTS

1. Introduction

1.1. Purpose

The main purpose of this document is to define specifications of a uDLX implementation and to provide a full overview of the design. This specifications defines all implementation parameters that composes the general uDLX requirements and specification. This definitions include processor operation modes, instruction set (ISA) and internal registers characteristics. This document also include detailed information of pipeline stages architecture, buses and other supplemental units.

1.2. Document Outline Description

This document is outlined as follow:

- Section 2: This section presents the core processor block diagram, Pin/Port definitions and global parameters and configuration directives.
- Section 3: This section presents the μDLX instruction layout and specifications.
- Section 4: This section presents a description of each pipeline stage block, including pin definitions, signals and internal datapath.

1.3. Acronyms and Abbreviations

Along this and other documents part of this project, it will be recurrent the usage of some acronyms and abbreviations. In order to keep track of this elements the Table 1 presents a set of abbreviations used and its corresponding meaning.

Table 1: Acronym and descriptions of elements in this document.

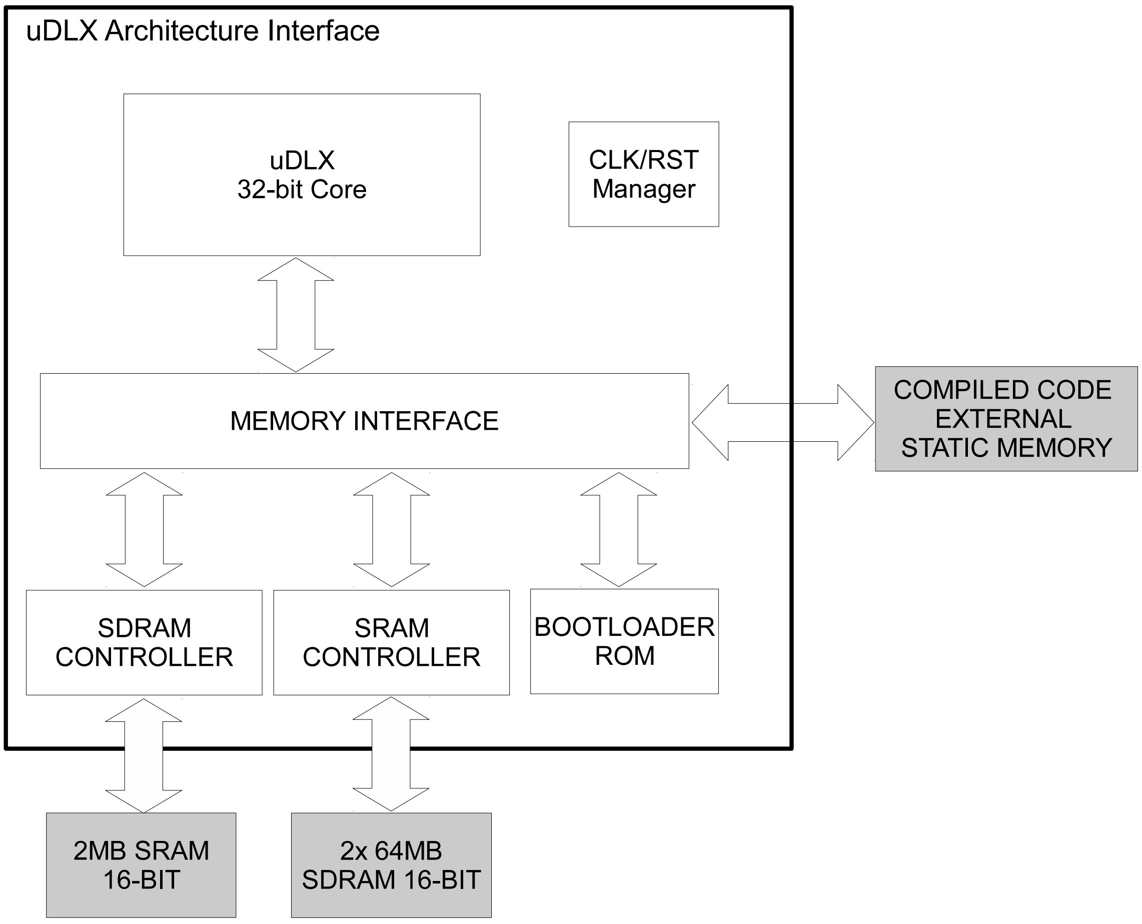
Acronym	Description
RISC	Reduced Instruction Set Computer
GPR	General Purpose Registers
FPGA	Field Gate Programmable Array
GPPU	General Purpose Processing Unit
SDRAM	Synchronous Dynamic Random Access Memory
HDL	Hardware Description Language
RAW	Read After Write
CPU	Central Processing Unit
ISA	Instruction Set Architecture
ALU	Arithmetic and Logic Unit
PC	Program Counter
RFlags	Flags Register
Const	Constant
BPM	Branch Prediction Buffer

2. Architecture Overview

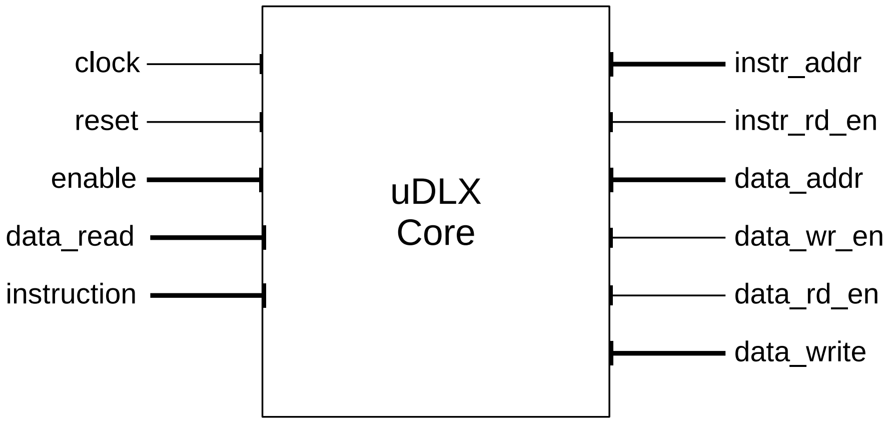
2.1. Interface Architecture

The μDLX architecture interface is composed by the following components.

- μ**DLX 32-bit Core:** The core four-deep pipeline processor.
- **Memory Interface:** Provides a middle layer between the core processor and the external memories. This interface also controls the bootloader process.
- **SDRAM Controller:** Provides the interface for controlling the external SDRAM.
- **SRAM Controller:** Provides the interface for controlling the external SRAM.



2.2. Block Diagram



2.3. Pin/Port Definitions

Name	Length	Direction	Description
clock	1	input	CPU core clock
reset	1	input	CPU core reset
instruction	32	input	SRAM instruction data
data_read	32	input	SDRAM read data
instr_addr	20	input	SRAM address
instr_rd_en	1	output	SRAM read enable
data_addr	13	output	SDRAM address
data_wr_en	1	output	SDRAM write enable
data_rd_en	1	output	SDRAM read enable
data_write	32	output	SDRAM write data

2.4. Parameters and Configurations

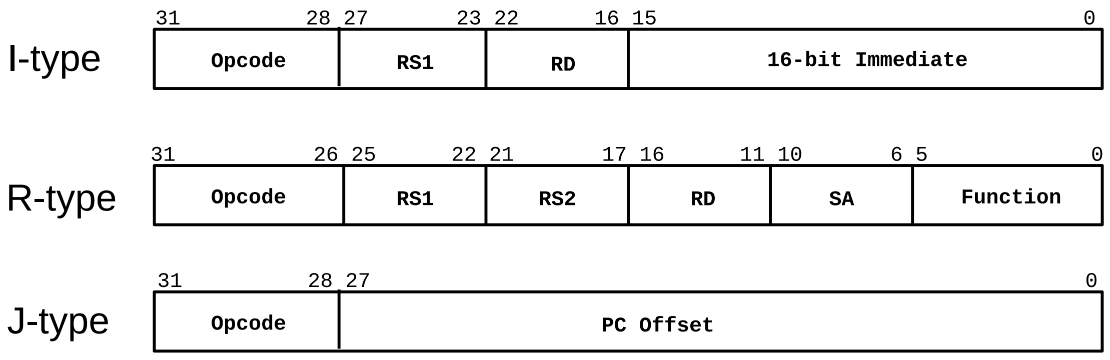
Name	Value	Description

3. Instructions Layout

In order to be the most close to the DLX and also be as compatible as possible to MIPS R2000 and R3000 architecture few instructions were added to the instruction set. Instructions that are not present in DLX and MIPS architecture are added in not used OPCODES.

3.1. Instructions Set

DLX instruction structure has 5 types of instructions, the floating point instructions were removed. The 3 DLX instructions types supported in the project are shown in figure below.



NOP instruction has zero in all bits and will not be mapped to an instruction type.

3.2. I-type Instruction

Immediate instructions use the immediate data to address a load and store operation, make arithmetic operations and make brachs using Some arithmetic immediate instructions were added because they make easier assembly programming. The conditional branch operations BEQZ and BNEQZ were added to enable some compiler compatibility and future core improvement.

OPCODE	Mneumonic	Operation
100011/0x23	lw	$R_D = mem$
101011/0x2b	sw	$mem = R_D$
101010/0x2A	brfl	???
001000/0x08	addi	$R_D = R_S1 + Sext(imm)$
001010/0x10	subi	$R_D = R_S1 - Sext(imm)$
001101/0x12	andi	$R_D = R_S1??Sext(imm)$
001101/0x13	ori	$R_D = R_S1??Sext(imm)$
000100/0x04	beqz	$PC = PC + 4 + (R_S1 = 0?Sext(imm) : 0)$
000101/0x05	bnez	??
000111/0x16	jr	$PC = R_S1$

Only LW, SW, BRFL, and JR instructions are in the p?oject requirements, the other were added to make compatibility and some codes easier.

3.3. R-type Instruction

This instructions realize registers operations, most operations are arithmetic. All arithmetic opcodes are zero, differing only in the function value.

OPCODE	Mneumonic	Operation
100000/0x20	add	$R_D = R_S1 + R_S2$
100010/0x22	sub	$R_D = R_S1 - R_S2$
100100/0x24	and	$R_D = R_S1 and R_S2$
100101/0x25	or	$R_D = R_S1 or R_S2$
011000/0x18	mult	$R_D = R_S1 x R_S2$
011010/0x1A	div	$R_D = R_S1/R_S2$
011100/0x1C	cmp	$R_D = R_S1 cmp R_S2$
011101/0x1D	not	$R_D = R_S1 not R_S2$

The MULT, DIV, CMP, and NOT instructions were added in DLX instruction set

3.4. J-type Instruction

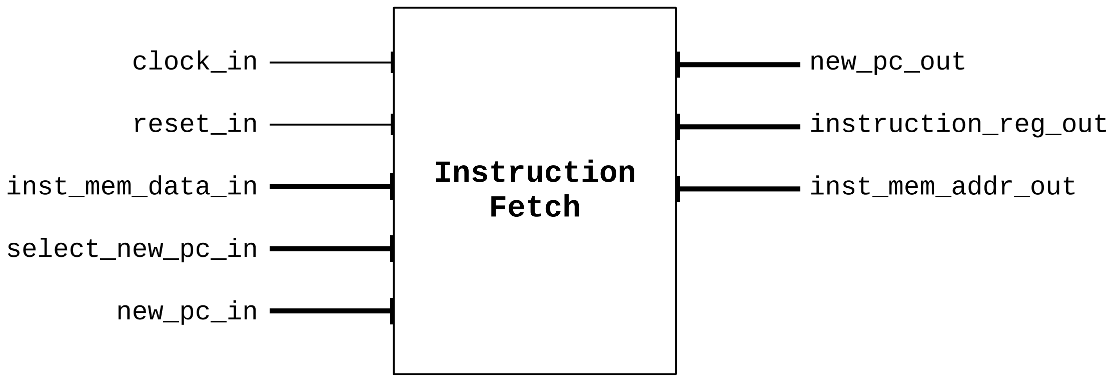
Instructions related with instruction memory jump. The instructions are JPC (J),RET (RFE), CALL (TRAP).

OPCODE	Mneumonic	Oper
000010/0x02	jpc(j)	$PC = PC + 4$
111110/0x3e	call(trap)	$trap = 1; EPC = PC; PC = SISR; ESR = SR; ECA = maskedCA;$
111111/0x3f	ret(rfe)	$SR = ESR; PC = EPC$

4. Architecture Description

4.1. Instruction Fetch

4.1.1. *Block Diagram*



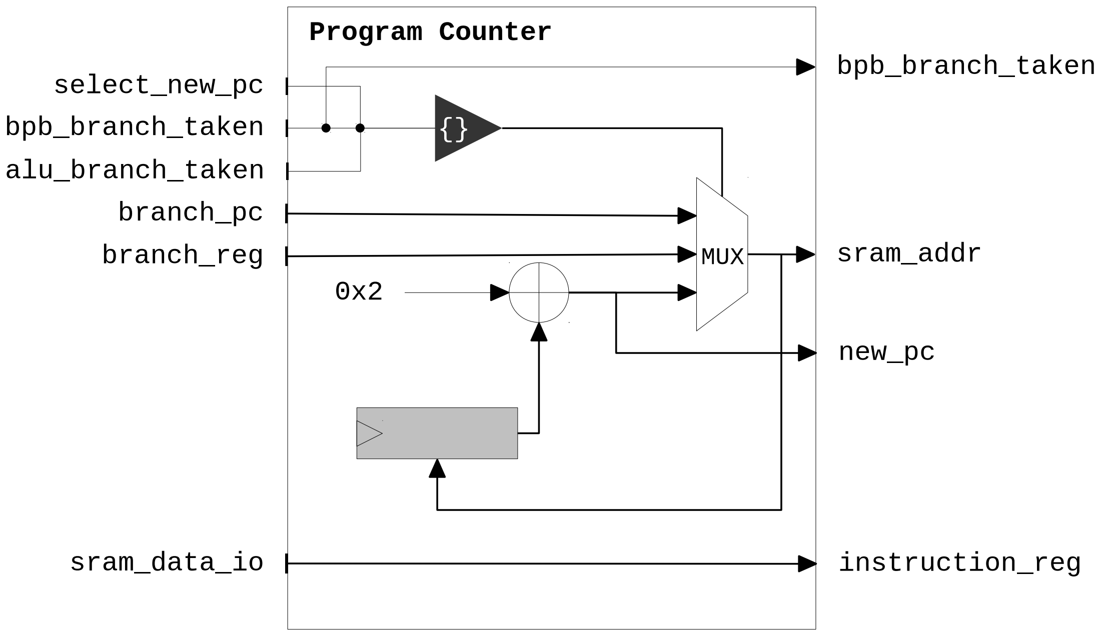
4.1.2. *Pin/Port Definitions*

Name	Length	Direction	Description
clock	1	input	CPU core clock
reset	1	input	CPU core reset
sram_data_io	16	in/out	SRAM data
branch_pc	20	input	Branch address PC relative
branch_reg	20	input	Branch address loaded from registers
select_new_pc	1	input	Signal used for branch not taken
bpb_branch_taken	1	input	Branch prediction buffer result
alu_branch_taken	1	input	Branch result from execution
new_pc	20	output	Updated value of PC
instruction	32	output	CPU core instruction
sram_addr	20	output	SRAM address
sram_we	1	output	SRAM write enable
bpb_branch_taken	1	output	Branch prediction buffer result

4.1.3. *Internal Datapath*

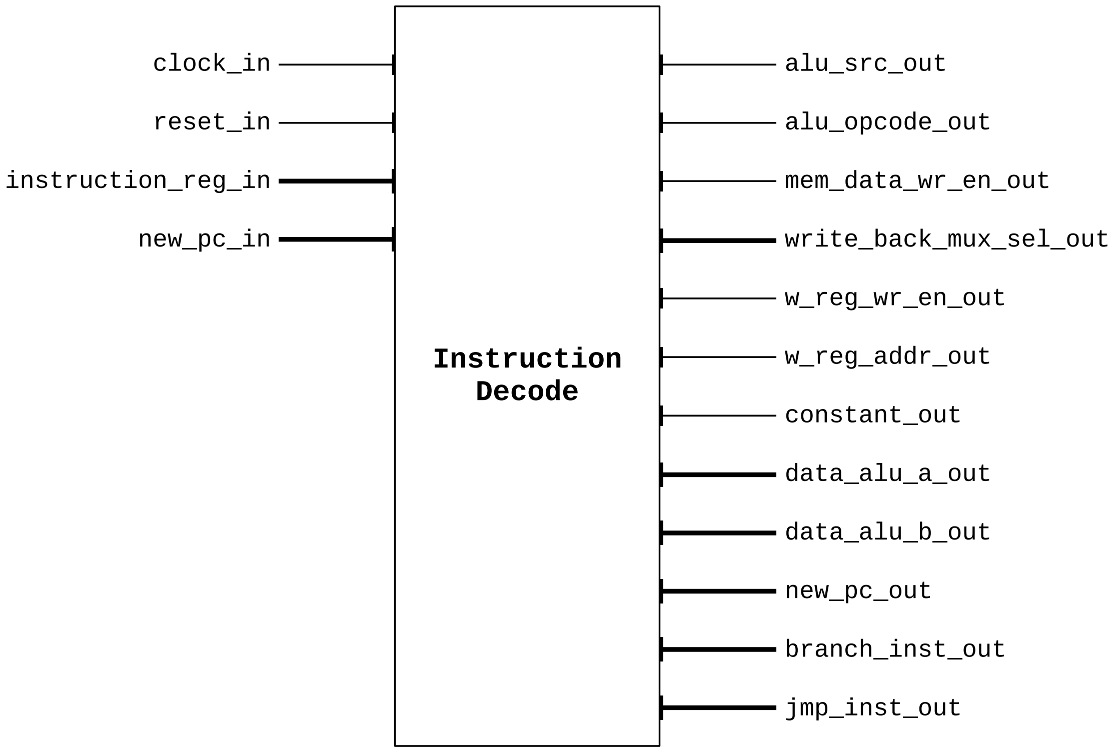
The internal data path is composed by the following components.

Program Counter : During the instruction time of an instruction this is the address of the instruction word. The address of the instruction that occurs during the next instruction time is determined by assigning a value to PC during an instruction time. If no value is assigned to PC during an instruction time by any pseudocode statement, it is automatically incremented by 2 before the next instruction time.



4.2. Instruction Decode/Register Fetch

4.2.1. Block Diagram



4.2.2. Pin/Port Definitions

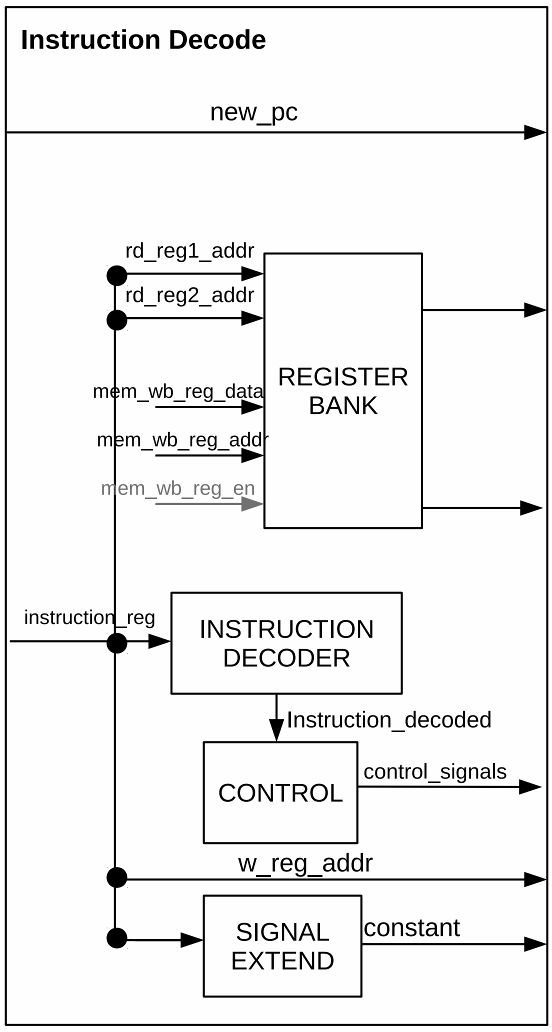
Name	Length	Direction	Description
clock_in	1	input	CPU core clock
reset_in	1	input	CPU core reset
instruction_reg_in	32	input	CPU core instruction
new_pc_in	20	input	Updated value of PC
alu_src_out	1	output	Alu mux source selector
alu_opcode_out	3	output	Alu opcode
mem_data_wr_en_out	1	output	Data memory write enable
write_back_mux_sel_out	1	output	Write back mux selector
w_reg_wr_en_out	1	output	GPR bank write enable signal
w_reg_addr_out	TBD	output	GPR bank write enable signal
constant_out	32	output	32-bit Sign-extended constant
data_alu_a_out	32	output	ALU input A data

continued on next page

continued from previous page			
Name	**Length**	**Direction**	**Description**
data_alu_b_out	32	output	ALU input B data
new_pc_out	20	output	Updated value of PC delayed
branch_inst_out	1	output	Conditional branch instruction
jmp_inst_out	1	output	Incoditional branch instruction

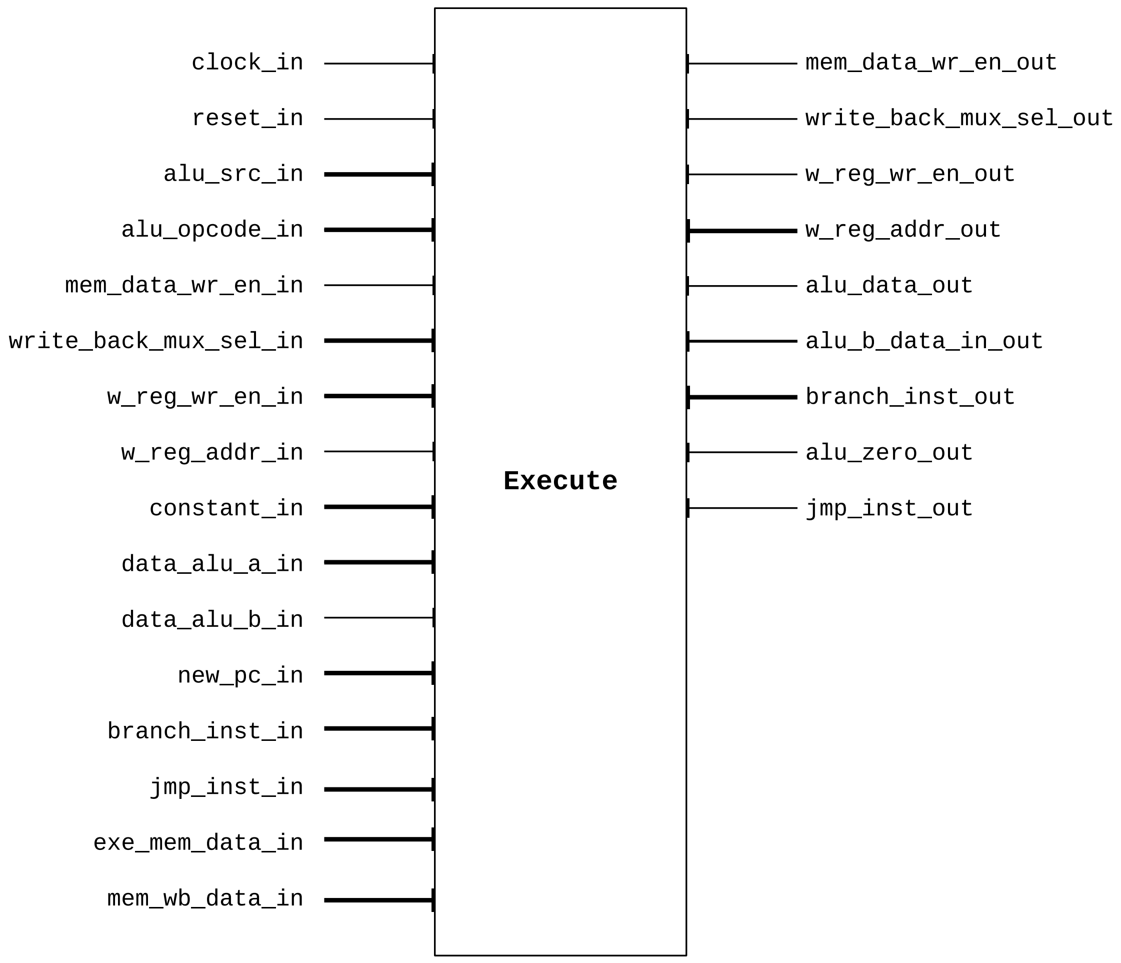
4.2.3. *Internal Datapath*

The internal data path is composed by the following components.



4.3. Execute/Address Calculate

4.3.1. Block Diagram



4.3.2. Pin/Port Definitions

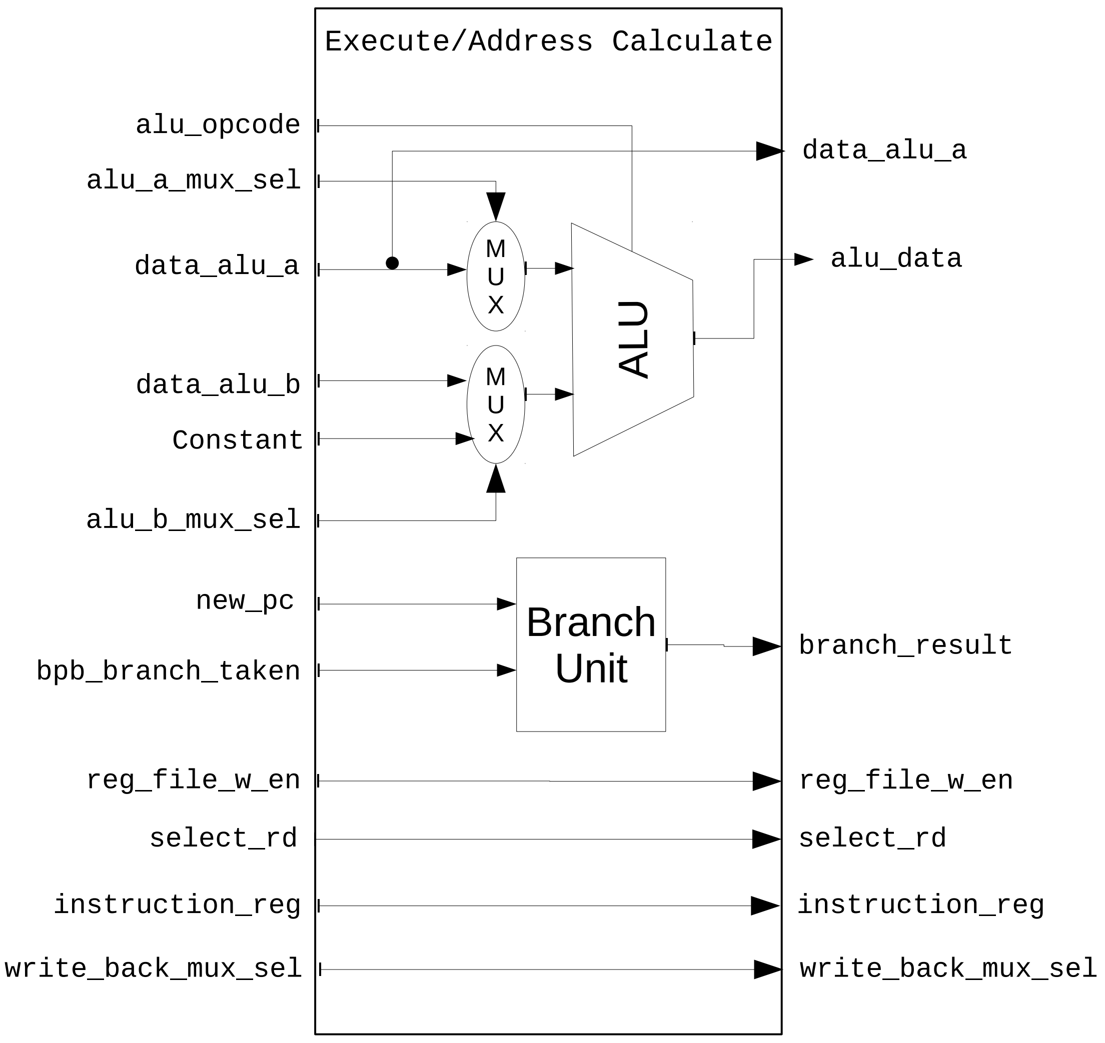
Name	Length	Direction	Description
clock	1	input	CPU core clock
reset	1	input	CPU core reset
alu_opcode	3	input	ALU opperation code
data_alu_a	32	input	ALU input A data
data_alu_b	32	input	ALU input B data
alu_a_mux_sel	TBD	input	ALU input A data select
alu_b_mux_sel	TBD	input	ALU input B data select
instruction_reg	32	input	CPU core instruction

continued on next page

continued from previous page			
Name	**Length**	**Direction**	**Description**
constant	32	input	32-bit Sign-extended constant
write_back_mux_sel	TBD	input	Write back mux select
reg_file_w_en	1	input	GPR bank write enable
select_rd	TBD	input	TBD
bpb_branch_taken	1	input	Branch prediction buffer result
new_pc	20	input	Updated value of PC
data_alu_a	32	output	ALU input A data
alu_data	32	output	ALU data output
instruction_reg	32	output	CPU core instruction
write_back_mux_sel	TBD	output	Write back mux select
reg_file_w_en	1	output	GPR bank write enable
select_rd	TBD	output	TBD
branch_result	1	output	Branch result after flag over ALU execution check

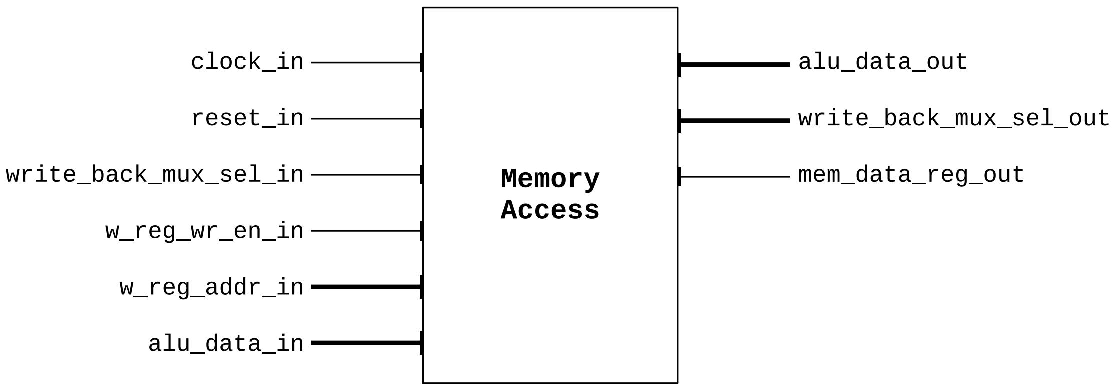
4.3.3. Internal Datapath

The internal data path is composed by the following components.


Execute/Address Calculate
alu_opcode
alu_a_mux_sel
data_alu_a
data_alu_b
Constant
alu_b_mux_sel
new_pc
bpb_branch_taken
reg_file_w_en
select_rd
instruction_reg
write_back_mux_sel
MUX
MUX
ALU
Branch Unit
data_alu_a
alu_data
branch_result
reg_file_w_en
select_rd
instruction_reg
write_back_mux_sel

4.4. Memory Access

4.4.1. Block Diagram

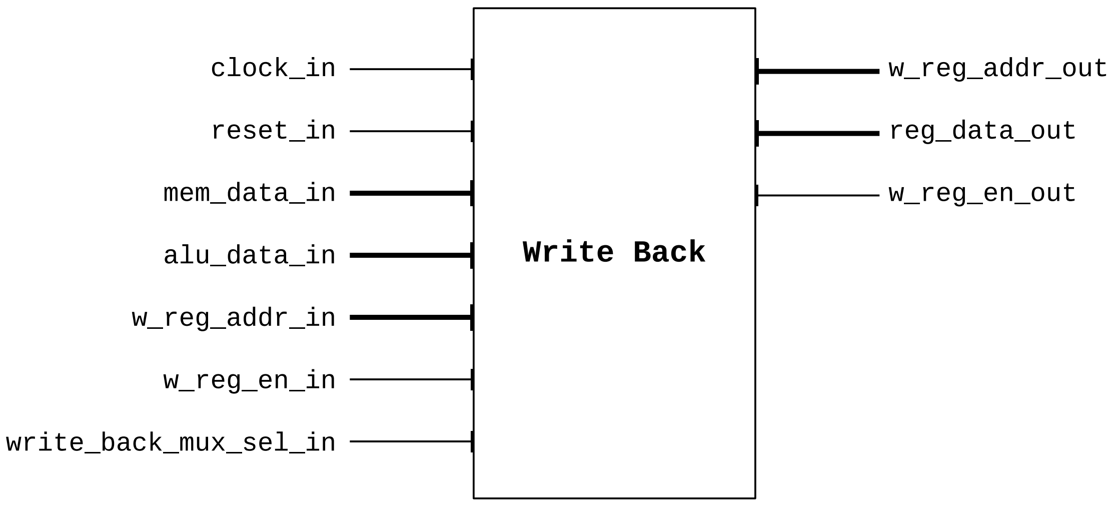


4.4.2. Pin/Port Definitions

Name	Length	Direction	Description
clock	1	input	CPU core clock
reset	1	input	CPU core reset
sdram_dara_ready	1	input	SDRAM data ready control
sdram_w_en	1	input	SDRAM write enable
sdram_addr	13	input	SDRAM read/write address
sdram_data_io	32	input	SDRAM I/O data
alu_data	32	input	ALU data output
select_rd	TBD	input	Select data to be writen in GPR bank
reg_file_w_en	4	input	GPR bank write enable signal
write_back_mux_sel	TBD	input	Write back mux select
mem_data	32	output	Memory output data
sdram_addr	13	output	SDRAM read/write address
sdram_w_en	1	output	SDRAM write enable
select_rd	TBD	output	Select data to be writen in GPR bank
reg_file_w_en	4	output	GPR bank write enable signal
write_back_mux_sel	TBD	output	Write back mux select
alu_data	32	output	ALU data output

4.5. Write Back

4.5.1. Block Diagram

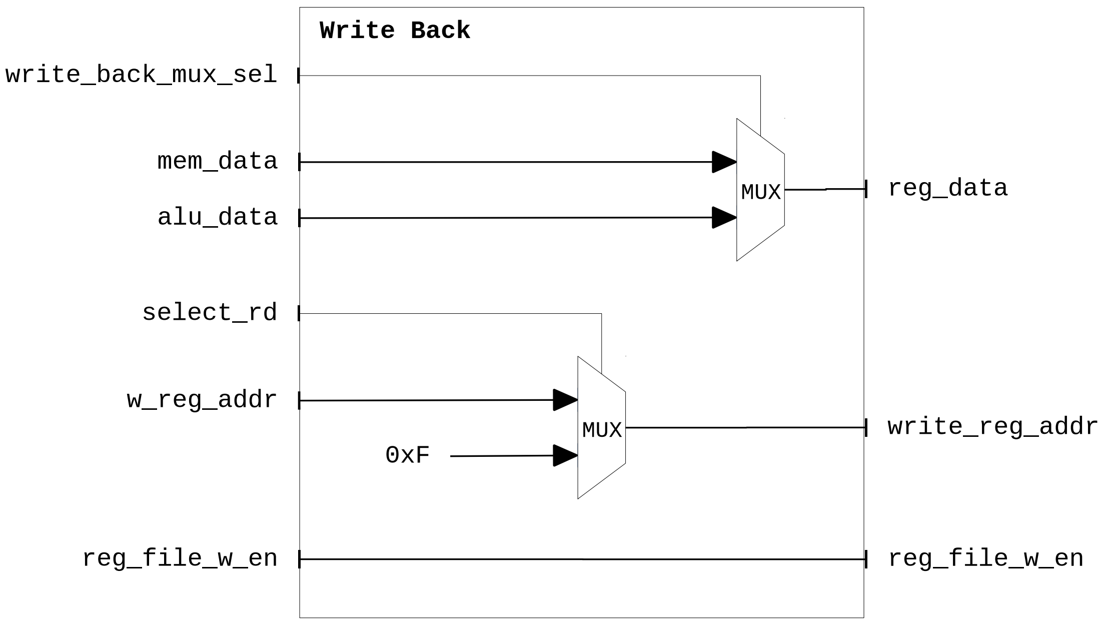


4.5.2. Pin/Port Definitions

Name	Length	Direction	Description
clock	1	input	CPU core clock
reset	1	input	CPU core reset
mem_data	32	input	SDRAM data output
alu_data	32	input	ALU data output
w_file_w_en	4	input	GPR bank write enable signal
w_reg_addr	1	input	GPR bank destiny address
write_back_mux_sel	TBD	input	Write back mux select
select_rd	TBD	input	Select data to be writen in GPR bank
w_reg_addr	4	output	GPR bank destiny address
reg_data	32	output	GPR bank write data
reg_file_w_en	1	output	GPR bank write enable signal

4.5.3. Internal Datapath

The internal data path is composed by the following components.


Write Back
write_back_mux_sel
mem_data
alu_data
MUX
reg_data
select_rd
w_reg_addr
0xF
MUX
write_reg_addr
reg_file_w_en
reg_file_w_en

4.6. Pipeline Register Description

4.6.1. *Instruction Fetch/Instruction Decode*

Name	Length	Description
new_pc	20	Stores the next program counter value.
instruction_reg	32	Stores the intruction word.
bpb_branch_taken	1	Stores BPB result.

4.6.2. *Instruction Decode/Execute*

Name	Length	Description
new_pc	20	Stores the next program counter value.
data_alu_reg_a	32	Stores the value of ALU input port A.
data_alu_reg_b	32	Stores the value of ALU input port B.
constant	32	Stores the signed extended integer constant.
instruction_reg	32	Stores the intruction word.
select_rd_reg	1	TBD
reg_file_w_en_reg	1	Stores the signal to enable GPR write back.
write_back_mux_sel_reg	TBD	Stores the select signal for write back Multiplexer.
alu_opcode	3	Stores the ALU opperation code.
select_mux_alu_a	TBD	Stores the ALU input data select signal
select_mux_alu_b	TBD	Stores the ALU input data select signal
bpb_branch_taken	1	Stores BPB result.

4.6.3. *Execute/Memory Access*

Name	Length	Description
instruction_reg	32	Stores the intruction word.
select_rd_reg	1	TBD
reg_file_w_en_reg	1	Stores the signal to enable GPR write back.
write_back_mux_sel_reg	TBD	Stores the select signal for write back Multiplexer.

continued on next page

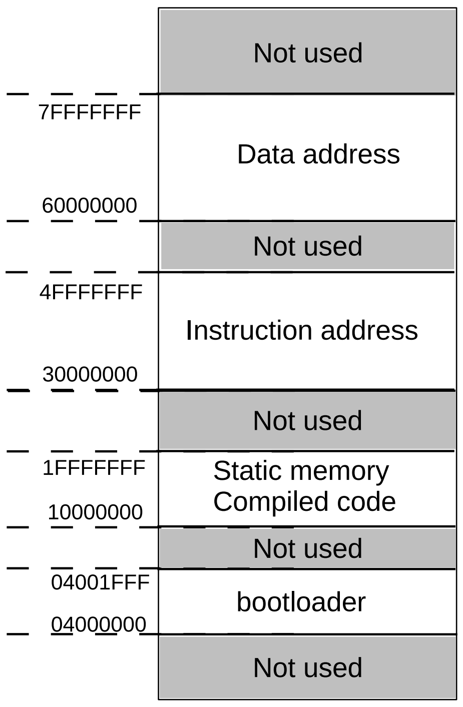
continued from previous page		
Name	**Length**	**Description**
data_alu_a	32	Stores the ALU input data A for memory addressing.
alu_data_reg	32	Stores the ALU output data.

4.6.4. *Memory Access/Write Back*

Name	Length	Description
instruction_reg	32	Stores the intruction word.
select_rd_reg	1	TBD
reg_file_w_en_reg	1	Stores the signal to enable GPR write back.
write_back_mux_sel_reg	TBD	Stores the select signal for write back Multiplexer.
mem_data_reg	32	Stores the memory output data.
alu_data_reg	32	Stores the ALU output data.
w_reg_addr_reg	4	Stores the GPR data write address.

4.7. Memory and Device Interface

The memory and device interface is responsible for memory mapping to the processor. Depending on the address accessed a specific memory will be accessed by the memory device interface. If it is a read operation in the next clock cycle the read data will be directed to the DLX input read data port.

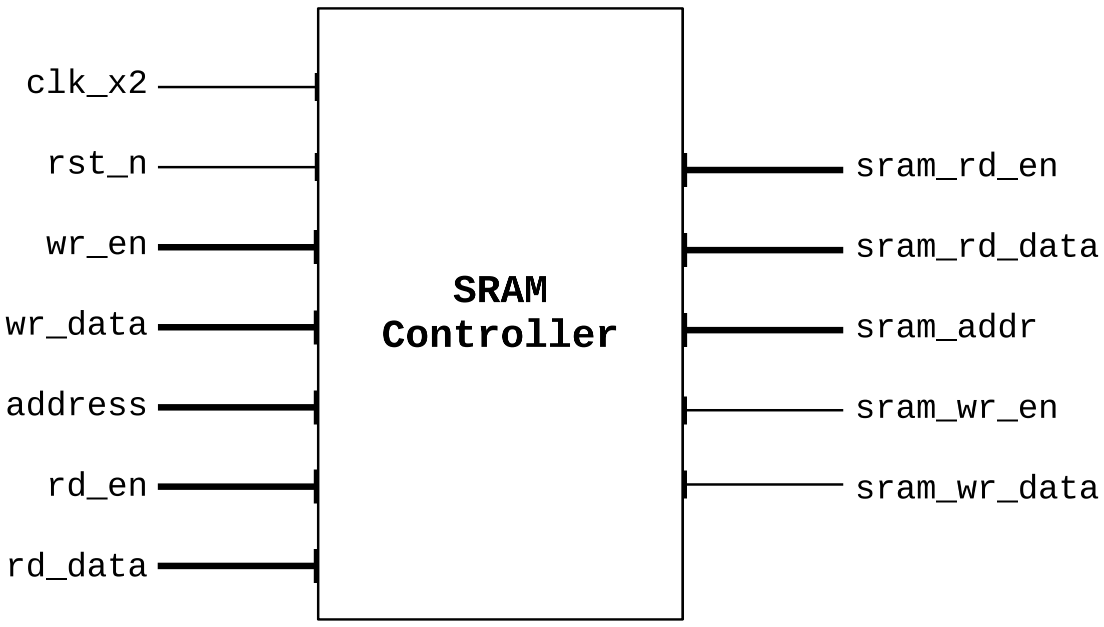


The memory map shown was created with separate address to show that devices have no relation between them and to show that the memory used in this project will be a small fraction for the 4GBytes possible.

4.8. SRAM Controller

The SRAM will store the instructions, as the SRAM has only 16 bits of word and an instruction has 32 bits the read and write operation must be done in a faster clock domain to avoid stopping the microprocessor. The SRAM has instruction code this way this memory is not written in normal operation. The SRAM will be written by the processor just during the bootloader code is running and during this operation no read will be done. Not having read and write simultaneously make easier the control not needing to deal with write and read operations at the same time.

4.8.1. Block Diagram



4.8.2. Pin/Port Definitions

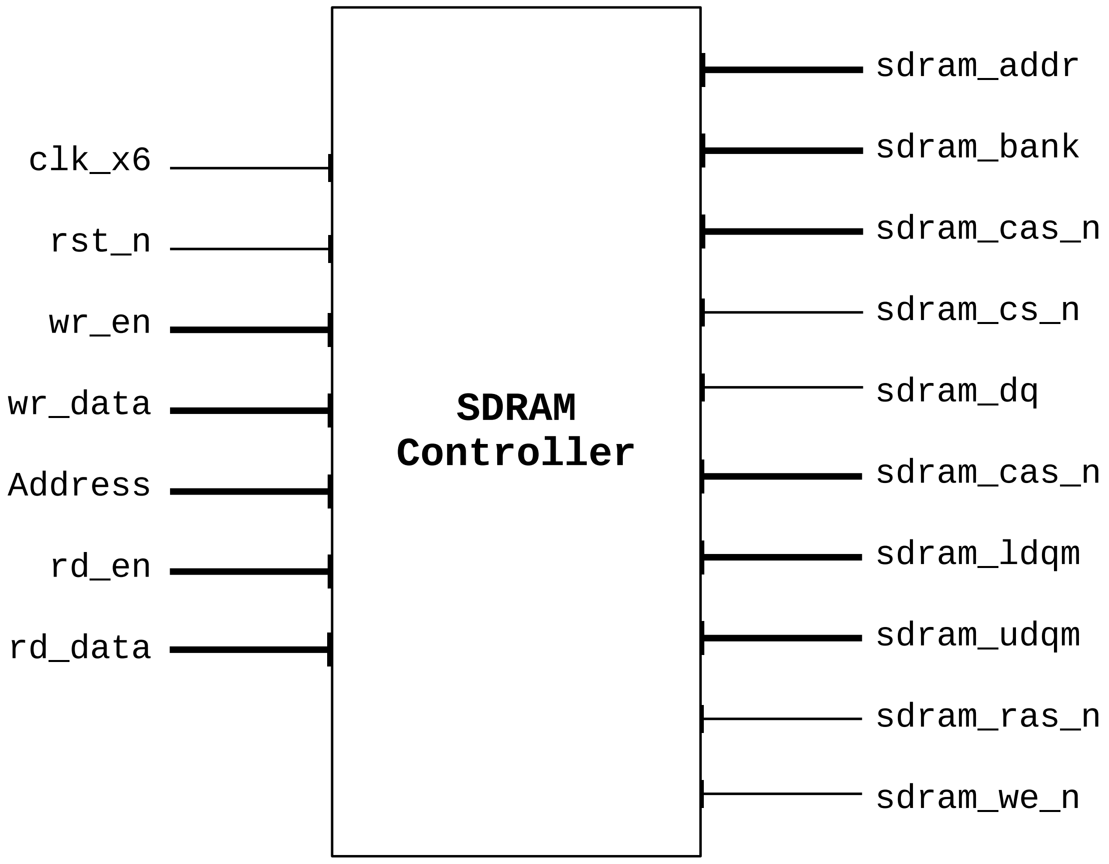
Name	Length	Direction	Description
clk_x2	1	input	SRAM clock that is twice the DLX clock
rst_n	1	input	System asynchronous reset
wr_en	1	input	Data write enable
wr_data	32	input	Data to be written in the memory
address	24	input	Memory address from the DLX
rd_en	1	input	Data read enable
rd_data	32	output	Read data from SRAM to the DLX
sram_rd_en	1	output	???
sram_rd_data	16	output	???
sram_addr	10	output	Memory address that goes to the SRAM
sram_wr_en	1	output	???
sram_wr_data	16	output	???

As the SRAM has half word the controller works with a clock twice faster enabling read two address, concatenating and sending to the DLX in time.

4.9. SDRAM Controller

The SDRAM controller will be very simple and optimized to read or write one word that will be requested by the processor. There will be no burst, just one word write or read operation.

4.9.1. *Block Diagram*



4.9.2. *Pin/Port Definitions*

Name	Length	Direction	Description
clk_x6	1	input	SDRAM controller input clock. 6 times faster than DLX clock
rst_n	1	input	System asynchronous reset
wr_en	1	input	Data write enable
wr_data_in	32	input	Data to be written in the memory
address	24	input	Address of write or read operation
rd_en	1	input	Data read enable

continued on next page

continued from previous page			
Name	**Length**	**Direction**	**Description**
rd_data	32	output	Data read from the SDRAM to the memory interface
sdram_addr	12	output	SDRAM data address
sdram_bank	2	output	The SDRAM selected Bank
sdram_cas_n	1	output	SDRAM CAS
sdram_cs_n	1	output	SDRAM chip select
sdram_dq	32	output	SDRAM read data, 2 bus of 16 bits, one for each memory
sdram_ldqm	1	output	???
sdram_udqm	1	output	???
sdram_ras_n	1	output	SRAM RAS
sdram_we_n	1	output	SDRAM write enable

4.10. Forwarding Unit

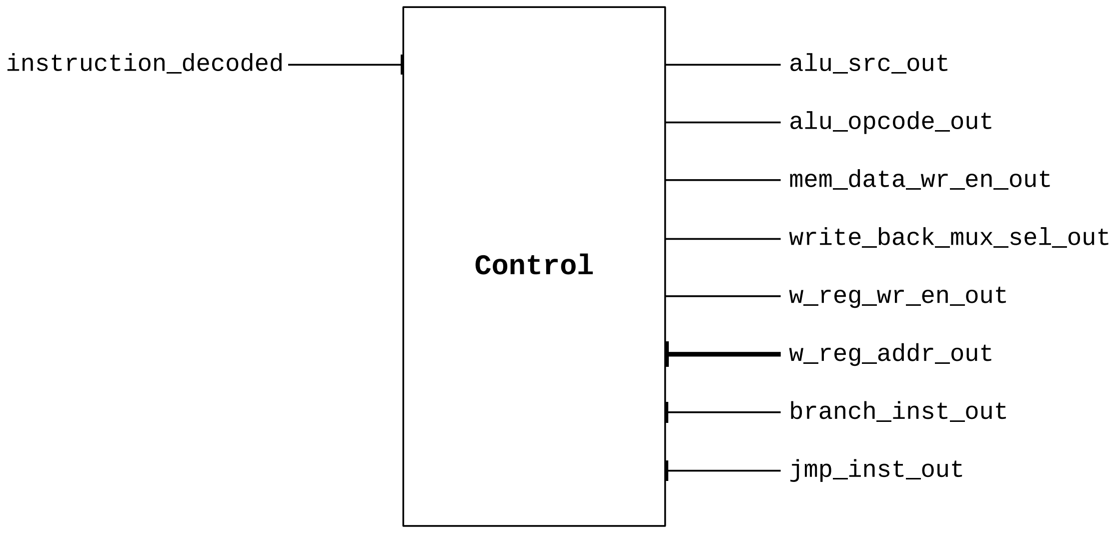
TBD in further releases.

4.11. Branch Prediction Buffer

TBD in further releases.

4.12. Control Micro-instructions Description

4.12.1. Block Diagram

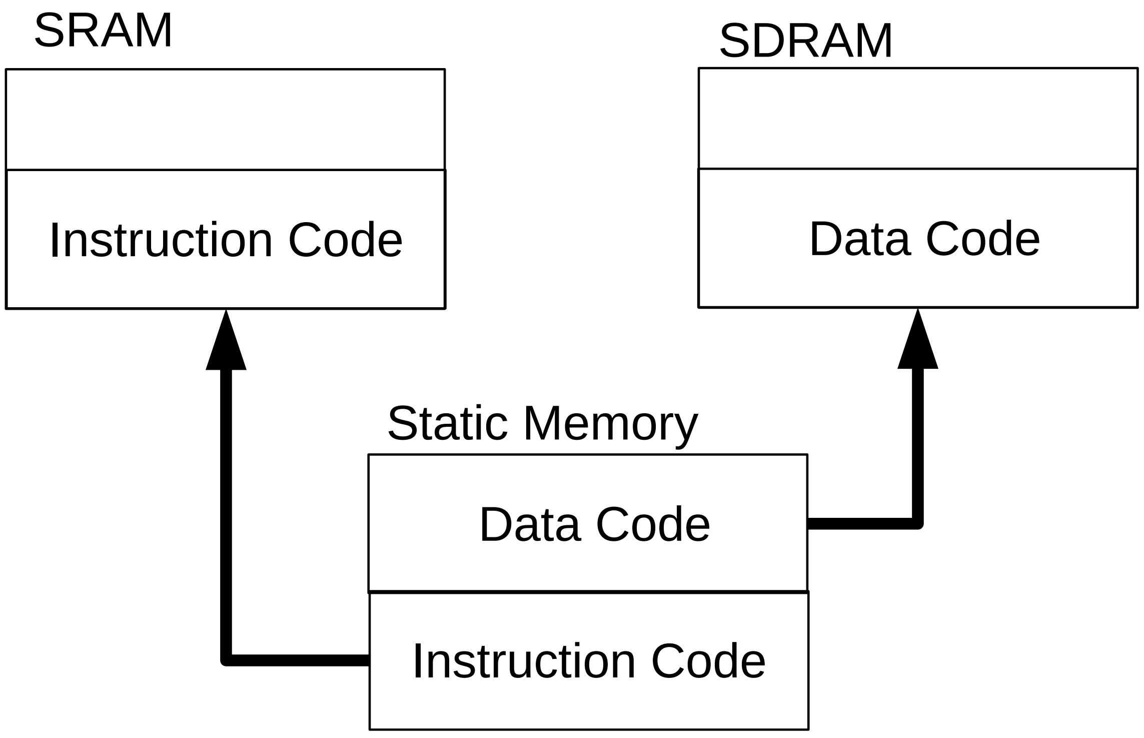


4.12.2. *Pin/Port Definitions*

Name	Length	Direction	Description
instruction_decoded	1	input	???
alu_src_out_n	1	output	???
alu_opcode_out	1	output	???
mem_data_wr_en_out	1	output	???
write_back_mux_sel_out	1	output	???
w_reg_wr_en_out	1	output	???
w_reg_addr_out	1	output	???
branch_inst_out	12	output	???
jmp_inst_out	1	output	???

4.13. **Bootloader**

The bootloader is the ROM memory has the code responsible to initialize the processor and write the code that will be executed in the correct address and consequently each memory device. The ROM must have the DLX reset PC register address. When the system is reseted the processor will read instructions from the ROM address. The code inside the ROM will copy the instruction and data code from a static memory to the instruction memory SRAM and the data memory SDRAM.



Usually data in code is read-only data and it is common in compiled C code. Probably there will be only instruction code to be loaded during bootloader operation.