

CS 301 Final Proj

Generated by Doxygen 1.8.14

Contents

1	Class Index	1
1.1	Class List	1
2	Class Documentation	3
2.1	ALU Class Reference	3
2.1.1	Member Function Documentation	3
2.1.1.1	performOperation()	3
2.2	ControlUnit Class Reference	4
2.2.1	Member Function Documentation	4
2.2.1.1	getMemWrite()	4
2.2.1.2	getRegDest()	4
2.2.1.3	setToZero()	5
2.2.1.4	setValues()	5
2.3	Converter Class Reference	5
2.3.1	Member Function Documentation	5
2.3.1.1	binaryToHex()	5
2.3.1.2	hexify()	6
2.3.1.3	hexToBinary()	6
2.3.1.4	hextoint()	7
2.3.1.5	inttobinary()	7
2.3.1.6	inttohex()	7
2.4	DataMemory Class Reference	8
2.4.1	Constructor & Destructor Documentation	8
2.4.1.1	DataMemory()	8

2.4.2	Member Function Documentation	8
2.4.2.1	dmemPrintFinal()	9
2.4.2.2	getdata()	9
2.4.2.3	writeMem()	9
2.5	Instruction Class Reference	10
2.5.1	Member Function Documentation	10
2.5.1.1	setValues()	10
2.6	InstructionMemory Class Reference	11
2.6.1	Member Function Documentation	11
2.6.1.1	getInstruction()	11
2.6.1.2	printContents()	11
2.7	Multiplexor Class Reference	12
2.7.1	Member Function Documentation	12
2.7.1.1	setFlow()	12
2.8	OpcodeTable Class Reference	12
2.8.1	Constructor & Destructor Documentation	13
2.8.1.1	OpcodeTable()	13
2.8.2	Member Function Documentation	13
2.8.2.1	getFunctField()	13
2.8.2.2	getOpcode()	13
2.8.2.3	getOpcodeField()	13
2.8.2.4	IMMposition()	13
2.8.2.5	isIMMLabel()	14
2.8.2.6	RDposition()	14
2.8.2.7	RSposition()	14
2.8.2.8	RTposition()	14
2.9	Parser Class Reference	14
2.9.1	Constructor & Destructor Documentation	15
2.9.1.1	Parser()	15
2.9.2	Member Function Documentation	15

2.9.2.1	isFormatCorrect()	15
2.10	ProgramCounter Class Reference	15
2.10.1	Detailed Description	16
2.10.2	Member Function Documentation	16
2.10.2.1	getCurrentAddress()	16
2.10.2.2	moveAddressTo()	16
2.11	RegisterEntry Struct Reference	16
2.12	RegisterFile Class Reference	17
2.12.1	Constructor & Destructor Documentation	17
2.12.1.1	RegisterFile() [1/2]	17
2.12.1.2	RegisterFile() [2/2]	17
2.12.2	Member Function Documentation	17
2.12.2.1	getNum()	17
2.12.2.2	writeReg()	18
2.13	ShiftLeftTwo Class Reference	18
2.13.1	Detailed Description	18
2.14	SignExtend Class Reference	18
2.15	Stimulation Class Reference	19
2.15.1	Member Function Documentation	19
2.15.1.1	run()	19
2.16	Tester Class Reference	32
2.16.1	Member Function Documentation	32
2.16.1.1	run()	33
Index		47

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

ALU	3
ControlUnit	4
Converter	5
DataMemory	8
Instruction	10
InstructionMemory	11
Multiplexor	12
OpcodeTable	12
Parser	14
ProgramCounter	
This h creates the guidelines for the program counter	15
RegisterEntry	16
RegisterFile	17
ShiftLeftTwo	
Used to shift the offset field to the left by two, making it a word offset	18
SignExtend	18
Stimulation	19
Tester	32

Chapter 2

Class Documentation

2.1 ALU Class Reference

Public Member Functions

- void **setInput_1** (string in_1)
- void **setInput_2** (string in_2)
- void **setOperation** (string op)
- void [performOperation](#) ()
- string **getResult** ()

2.1.1 Member Function Documentation

2.1.1.1 performOperation()

```
void ALU::performOperation ( )
```

Add : add, addi

Subtract : sub

Compare : beq

The documentation for this class was generated from the following files:

- ALU.h
- ALU.cpp

2.2 ControlUnit Class Reference

Public Member Functions

- void [setValues](#) (std::string opcode)
- void [setToZero](#) ()
- int [getRegDest](#) ()
- int [getJump](#) ()
- int [getBranch](#) ()
- int [getMemRead](#) ()
- int [getmemToReg](#) ()
- int [getMemWrite](#) ()
- int [getAluSrc](#) ()
- int [getRegWrite](#) ()
- int [getAluOp0](#) ()
- int [getAluOp1](#) ()
- void [printControl](#) ()

Method that prints out the contents of the Control.

Protected Attributes

- bool **regDest**
- bool **jump**
- bool **branch**
- bool **memRead**
- bool **memToReg**
- bool **aluOp0**
- bool **aluOp1**
- bool **memWrite**
- bool **aluSrc**
- bool **regWrite**

2.2.1 Member Function Documentation

2.2.1.1 [getMemWrite\(\)](#)

```
int ControlUnit::getMemWrite ( )
```

```
std::string ControlUnit::getAluOp() { return aluOp; }
```

2.2.1.2 [getRegDest\(\)](#)

```
int ControlUnit::getRegDest ( )
```

The following accessors return 1 if the control is set to true and 0 if false.

2.2.1.3 setToZero()

```
void ControlUnit::setToZero ( )
```

Method that sets all values to 0, must be done prior to each instruction

2.2.1.4 setValues()

```
void ControlUnit::setValues (
    std::string opcode )
```

Jump instruction

R-Type

I-Type Addi

I think? this means add

LW

SW

BEQ

Represents 01

The documentation for this class was generated from the following files:

- ControlUnit.h
- ControlUnit.cpp

2.3 Converter Class Reference

Static Public Member Functions

- static int [hextoint](#) (std::string s)
- static string [inttohex](#) (int x)
- static string [hexToBinary](#) (string hex)
- static string [binaryToHex](#) (string binary)
- static std::string [hexify](#) (std::string s)
- static std::string [inttobinary](#) (int a)

2.3.1 Member Function Documentation

2.3.1.1 binaryToHex()

```
static string Converter::binaryToHex (
    string binary ) [inline], [static]
```

Converts a given binary value to hex

Parameters

<i>binary</i>	binary value to be changed to a hex value
---------------	---

Returns

hex hexadecimal conversion of the provided binary value

string that will hold final hex string to return

string to extend input string so length is divisible by 4

string that will hold 4 bit chunks of input string

Extend inputted string to be of a length divisible by 4

append extender to front of binary string

Loop through 4 bit chunks of binary string, appending to hex string

2.3.1.2 hexify()

```
static std::string Converter::hexify (
    std::string s ) [inline], [static]
```

Checking if it already has "0x" in the begging of the string

Parameters

<i>s</i>	string to be converted to it's hex equivalent
----------	---

Returns

mystring the hexadecimal conversion of the provided string

2.3.1.3 hexToBinary()

```
static string Converter::hexToBinary (
    string hex ) [inline], [static]
```

Converts a given hex value to binary

Parameters

<i>hex</i>	hex value to be changed to a binary value
------------	---

Returns

bin binary conversion of the provided hex value

Get the next char in the hex

Convert the next char in the hex to the appropriate 4-bit binary representation

2.3.1.4 hextoint()

```
static int Converter::hextoint (
    std::string s ) [inline], [static]
```

Converts a given hex value to an int

Parameters

s	hex input to be changed to an int value
----------	---

Returns

integer conversion of the provided hex value

2.3.1.5 inttobinary()

```
static std::string Converter::inttobinary (
    int a ) [inline], [static]
```

Converts given int value into binary

Parameters

a	integer value to be converted to it's binary equivalent
----------	---

Returns

bin thebinary conversion of the provided integer

2.3.1.6 inttohex()

```
static string Converter::inttohex (
    int x ) [inline], [static]
```

Converts a given int value to hex

Parameters

x	integer value to be changed to a hex value
---	--

Returns

s hexadecimal conversion of the provided integer value

The documentation for this class was generated from the following file:

- Converter.h

2.4 DataMemory Class Reference

Public Member Functions

- [DataMemory](#) ()
Default constructor.
- [DataMemory](#) (std::string filename)
- std::string [getdata](#) (std::string address)
- void [dmemPrint](#) ()
Prints the data memory to console.
- void [dmemPrintFinal](#) (string memOutputFile)
- std::string [writeMem](#) (string address, string val)

2.4.1 Constructor & Destructor Documentation

2.4.1.1 DataMemory()

```
DataMemory::DataMemory (  
    std::string filename )
```

Makes sure the file given is opened correctly and identifies the delimiter

Parameters

<i>filename</i>	file that will be checked for syntactic correctness
-----------------	---

2.4.2 Member Function Documentation

2.4.2.1 dmemPrintFinal()

```
void DataMemory::dmemPrintFinal (
    string memOutputFile )
```

Prints the data memory to the Output file

Parameters

<i>memOutputFile</i>	file that data memory will be written to
----------------------	--

2.4.2.2 getdata()

```
std::string DataMemory::getdata (
    std::string address )
```

Given an Address, returns the data associated with that address

Parameters

<i>theAddress</i>	specified address to gather data from
-------------------	---------------------------------------

Returns

mem[theAddress] the data associated with the specified address in the data memory

2.4.2.3 writeMem()

```
std::string DataMemory::writeMem (
    string address,
    string val )
```

Given an address and a value, will write the value within the specified data memory address

Parameters

<i>address</i>	address to be written to
<i>val</i>	value to be stored into the specified address

Returns

temp the value originally stored within the specified address

The documentation for this class was generated from the following files:

- DataMemory.h
- DataMemory.cpp

2.5 Instruction Class Reference

Public Member Functions

- [Instruction](#) (Opcode op, Register rs, Register rt, Register rd, int imm)
You can specify all the fields to initialize the [Instruction](#).
- void [setValues](#) (Opcode op, Register rs, Register rt, Register rd, int imm)
Allows you to specify all the fields of the [Instruction](#).
- Opcode [getOpcode](#) ()
Returns the various fields for the [Instruction](#).
- Register [getRS](#) ()
- Register [getRD](#) ()
- Register [getRT](#) ()
- int [getImmediate](#) ()
- string [getString](#) ()
Returns a string which represents all of the fields.
- void [setEncoding](#) (string s)
Stores the 32 bit binary encoding of MIPS instruction passed in.
- string [getEncoding](#) ()
Returns string representing the 32 binary encoding of MIPS instruction.

2.5.1 Member Function Documentation

2.5.1.1 setValues()

```
void Instruction::setValues (
    Opcode op,
    Register rs,
    Register rt,
    Register rd,
    int imm )
```

Allows you to specify all the fields of the [Instruction](#).

You can specify all the fields to initialize the [Instruction](#). if(!((imm & 0xFFFF0000) << 1)) ///
make sure it has nothing in upper 16 bits myImmediate = imm;

The documentation for this class was generated from the following files:

- Instruction.h
- Instruction.cpp

2.6 InstructionMemory Class Reference

Public Member Functions

- [InstructionMemory](#) ()
Default constructor.
- **InstructionMemory** (std::string filename)
- [getInstruction](#) (std::string address)
- bool [isValidInstruction](#) (std::string theAddress)
Checks to make sure that a valid instruction is given.
- void [printContents](#) ()

2.6.1 Member Function Documentation

2.6.1.1 [getInstruction\(\)](#)

```
Instruction InstructionMemory::getInstruction (
    std::string theAddress )
```

Given an address, will get the instruction associated with the specified address

Parameters

<i>theAddress</i>	Address given to access and get the associated Instruction
-------------------	--

Returns

[Instruction](#) associated with given address

2.6.1.2 [printContents\(\)](#)

```
void InstructionMemory::printContents ( )
```

Initializes iterator and iterates through myInstructionMapping, gathering the contents at each [Instruction](#) address and printing them out

The documentation for this class was generated from the following files:

- InstructionMemory.h
- InstructionMemory.cpp

2.7 Multiplexor Class Reference

Public Member Functions

- `std::string setFirstInput (std::string firstInput)`
Set's input at point that coincides with 0 on the picture.
- `std::string setSecondInput (std::string secondInput)`
Sets input at 1 that coincides with 1 on the picture.
- `std::string mux ()`
calls mux and returns the result chosed
- `void setFlow (int flow)`
- `int getFlow ()`
Returns signal for testing.

Protected Attributes

- `std::string firstInput`
- `std::string secondInput`
- `int flow`
0 or 1, based on value in the picture of the processor.

2.7.1 Member Function Documentation

2.7.1.1 setFlow()

```
void Multiplexor::setFlow (
    int flow )
```

will be used by the Control Unit, which sends true or false based on whether or not the mux is needed.

The documentation for this class was generated from the following files:

- Multiplexor.h
- Multiplexor.cpp

2.8 OpcodeTable Class Reference

Public Member Functions

- `OpcodeTable ()`
Initializes all the fields for every instruction in Opcode enum.
- `Opcode getOpcode (string str)`
- `int numOperands (Opcode o)`
Given an Opcode, returns number of expected operands.
- `int RSposition (Opcode o)`
- `int RTposition (Opcode o)`
- `int RDposition (Opcode o)`
- `int IMMposition (Opcode o)`
- `bool isIMMLabel (Opcode o)`
- `InstType getInstType (Opcode o)`
Given an Opcode, returns instruction type.
- `string getOpcodeField (Opcode o)`
- `string getFunctField (Opcode o)`
- `string name (Opcode o)`

2.8.1 Constructor & Destructor Documentation

2.8.1.1 OpcodeTable()

```
OpcodeTable::OpcodeTable ( )
```

Initializes all the fields for every instruction in Opcode enum.

```
myArray[UNDEFINED].name = "undefined"; myArray[UNDEFINED].numOps = -1; myArray[UNDEFINED].rdPos =
-1; myArray[UNDEFINED].rsPos = -1; myArray[UNDEFINED].rtPos = -1; myArray[UNDEFINED].immPos = -1; my↵
Array[UNDEFINED].op_field = ""; myArray[UNDEFINED].funct_field = "";
```

2.8.2 Member Function Documentation

2.8.2.1 getFunctField()

```
string OpcodeTable::getFunctField (
    Opcode o )
```

Given an Opcode, returns a string representing the binary encoding of the function field.

2.8.2.2 getOpcode()

```
Opcode OpcodeTable::getOpcode (
    string str )
```

Given a valid MIPS assembly mnemonic, returns an Opcode which represents a template for that instruction.

2.8.2.3 getOpcodeField()

```
string OpcodeTable::getOpcodeField (
    Opcode o )
```

Given an Opcode, returns a string representing the binary encoding of the opcode field.

2.8.2.4 IMMposition()

```
int OpcodeTable::IMMposition (
    Opcode o )
```

Given an Opcode, returns the position of IMM field. If field is not appropriate for this Opcode, returns -1.

2.8.2.5 isIMMLabel()

```
bool OpcodeTable::isIMMLabel (
    Opcode o )
```

Given an Opcode, returns true if instruction expects a label in the instruction. See "J".

2.8.2.6 RDposition()

```
int OpcodeTable::RDposition (
    Opcode o )
```

Given an Opcode, returns the position of RD field. If field is not appropriate for this Opcode, returns -1.

2.8.2.7 RSposition()

```
int OpcodeTable::RSposition (
    Opcode o )
```

Given an Opcode, returns the position of RS/RT/RD/IMM field. If field is not appropriate for this Opcode, returns -1.

Given an Opcode, returns the position of RS field. If field is not appropriate for this Opcode, returns -1.

2.8.2.8 RTposition()

```
int OpcodeTable::RTposition (
    Opcode o )
```

Given an Opcode, returns the position of RT field. If field is not appropriate for this Opcode, returns -1.

The documentation for this class was generated from the following files:

- Opcode.h
- Opcode.cpp

2.9 Parser Class Reference

Public Member Functions

- [Parser](#) (string filename)
- bool [isFormatCorrect](#) ()
- [Instruction getNextInstruction](#) ()
Iterator that returns the next [Instruction](#) in the list of Instructions.

Static Public Member Functions

- static string [cvtInt2Bin](#) (int number, size_t length)

2.9.1 Constructor & Destructor Documentation

2.9.1.1 Parser()

```
Parser::Parser (
    string filename )
```

Specify a text file containing MIPS assembly instructions. Function checks syntactic correctness of file and creates a list of Instructions. No opcode but operands

invalid opcode specified

2.9.2 Member Function Documentation

2.9.2.1 isFormatCorrect()

```
bool Parser::isFormatCorrect ( ) [inline]
```

Returns true if the file specified was syntactically correct. Otherwise, returns false.

The documentation for this class was generated from the following files:

- Parser.h
- Parser.cpp

2.10 ProgramCounter Class Reference

This h creates the guidelines for the program counter.

```
#include <ProgramCounter.h>
```

Public Member Functions

- [ProgramCounter](#) ()
This creates the program counter object.
- **ProgramCounter** (std::string address)
- std::string [getCurrentAddress](#) ()
- std::string [moveAddressTo](#) (std::string newAddress)

2.10.1 Detailed Description

This h creates the guidelines for the program counter.

forward declarations needed can go below

2.10.2 Member Function Documentation

2.10.2.1 getCurrentAddress()

```
std::string ProgramCounter::getCurrentAddress ( )
```

getCurrentAddress will return the current address of the program counter as a string

2.10.2.2 moveAddressTo()

```
std::string ProgramCounter::moveAddressTo (
    std::string newAddress )
```

moveAddressTo will move the address in the PC to a given point, will will be used for j type and branch instructions. This method will be called by control??

moveAddressTo will move the address in the PC to a given point, will will be used for j type and branch instructions

The documentation for this class was generated from the following files:

- ProgramCounter.h
- ProgramCounter.cpp

2.11 RegisterEntry Struct Reference

Public Attributes

- std::string **name**
- Register **number**
- std::string **value**

The documentation for this struct was generated from the following file:

- RegisterFile.h

2.12 RegisterFile Class Reference

Public Member Functions

- [RegisterFile](#) ()
- [RegisterFile](#) (string regsiterFile)
- Register [getNum](#) (string reg)
- std::string [readReg](#) (string reg)
- std::string [writeReg](#) (string reg, string value)
- void [printContents](#) ()

2.12.1 Constructor & Destructor Documentation

2.12.1.1 RegisterFile() [1/2]

```
RegisterFile::RegisterFile ( )
```

Register Table for access

2.12.1.2 RegisterFile() [2/2]

```
RegisterFile::RegisterFile (
    string regsiterFile )
```

Makes sure the file is opened correctly

Loop should run until eof().

creates string and saves each line to input

puts instruction in

increments number of instructions

2.12.2 Member Function Documentation

2.12.2.1 getNum()

```
Register RegisterFile::getNum (
    string reg )
```

Given a string representing a MIPS register operand, returns the number associated with that register. If string is not a valid register, returns NumRegisters.

2.12.2.2 writeReg()

```
std::string RegisterFile::writeReg (
    string reg,
    string value )
```

```
for(int i = 0; i < 2*NumRegisters; i++){ if(myRegisters[i].name == reg){ myRegisters[i].value = val; return myRegisters[i].value; } } return "";
```

The documentation for this class was generated from the following files:

- RegisterFile.h
- RegisterFile.cpp

2.13 ShiftLeftTwo Class Reference

Used to shift the offset field to the left by two, making it a word offset.

```
#include <ShiftLeftTwo.h>
```

Static Public Member Functions

- static std::string **Shift** (std::string offsetField)
shifts the offset field to the left by two bits, making it a word offset

2.13.1 Detailed Description

Used to shift the offset field to the left by two, making it a word offset.

The documentation for this class was generated from the following file:

- ShiftLeftTwo.h

2.14 SignExtend Class Reference

Static Public Member Functions

- static std::string **Extend** (std::bitset< 16 > sign_extend_val)

The documentation for this class was generated from the following file:

- SignExtend.h

2.15 Stimulation Class Reference

Public Member Functions

- void `run` ()

2.15.1 Member Function Documentation

2.15.1.1 `run()`

```
void Stimulation::run ( )
```

The following code will read the config file

initialized varibales

opens file then make sure it was successful

loop to run once for each config

loops through string until it finds equals sets pointer to j

gets the part of the input after the equals sign

this block of code adds the values from the config file to the appropriate variables.

```
int used to store # of instructions, must be <= 100  
}
```

```
_____/||| _____/||| _____  
_____/||| _____/||| _____  
_____/|||
```

[Parser](#) is run through instruction memory to initialize instructions. Initializes the instruction memory with the input file.

```
Instruction i = im->getInstruction("0x04000008"); string s = i.getString(); cout<<s<<endl;
```

Sets first address at the start and creates Program Counter Object

Creates controlunit object.

build 5 Multiplexors

only ADD

ADD and ALU Result

ALU and ALU Result

Loop should run until end of program, ends when the instruction memory gets to an invalid program.

If the user chose to use single step mode, this code asks the user to press y to continue, will continuously run until user enters y

FETCH Retrives address from the instruction memory as a string of 1s/0s.

Adds 4 to current address and stores the result.

sets values to false to reset control unit, then calls method to set control values with opcode.

resets values in control unit

mux 5 is set by a combination of branch and the result of [ALU](#)

always goes to read register1

goes to read register 2 and mux1

goes to mux1

gets last15 didgets of instruction

get j type address

function code

Shifts the value to the left (value used for address in jtype)

test for shift left

must wait for result of Mux5

Sends reg2 and reg3 to mux1

write register gets value from mux1 if a writeReg occurs this stores the register to be written to

Conversion to bitset so a conversion to int can be done

Converts from bitset to integer

Readreg accepts decimal value as a string, so we use to string

Converts values from hex (how its stored in register) to binary

test for mux1

sign extend accepts bitset.

second mux decided if imm or register 2 should go to the [ALU](#) extended in bin, val at reg2 is in hex rn

calls second mux to determine second input for alu

The following code acts as the [ALU](#) control for ALU3

SLT, ADD, SUB, SLT

Add

Subtract

SLT instruction, not yet implemented in [ALU](#)

Result gets "equal" or "not equal"

runs for lw and sw

if this runs it is a branch instruction AND the branch condition passed. Basically the AND in the data path.

Not needed but avoids an unused warning for mux5 if no branches

if there is a memory write (sw) it occurs here

valAtReg2 is value to be written address to be written to is alu3 result(needs to be converted to hex)

sends result of the alu to the 3rd multiplexor

runs if op uses a memory read, and sends value to the 3rd multiplexor aluresult needs to be translated to hex

checks to see if it is writing to a register from mux3.

remeber string writeRegister holds in the reg code below should write the given value to the register

so binary can be changed to int

Shifts the previously extended address by 2 bits(needed for b and j)

Add this value to current PC value(This doesnt make sense to me...)

result that is going to program counter

Updates program counter with correct address

prints the control fields, register memory and datamemory after each instruction if printMemoryContents is set to true.

The documentation for this class was generated from the following files:

- Stimulation.h
- Stimulation.cpp

2.16 Tester Class Reference

Public Member Functions

- void `run` ()

2.16.1 Member Function Documentation

2.16.1.1 run()

```
void Tester::run ( )
```

The following code will read the config file

initialized variables

opens file then make sure it was successful

loop to run once for each config

loops through string until it finds equals sets pointer to j

gets the part of the input after the equals sign

this block of code adds the values from the config file to the appropriate variables.


```
int used to store # of instructions, must be <= 100  
}
```

```
_____|//|_____|//|_____|
_____|//|_____|//|_____|
_____|//|
```

[Parser](#) is run through instruction memory to initialize instructions. Initializes the instruction memory with the input file.

```
Instruction i = im->getInstruction("0x04000008"); string s = i.getString(); cout<<s<<endl;
```

Sets first address at the start and creates Program Counter Object

Creates controlunit object.

build 5 Multiplexors

only ADD

ADD and ALU Result

ALU and ALU Result

Loop should run until end of program, ends when the instruction memory gets to an invalid program.

If the user chose to use single step mode, this code asks the user to press y to continue, will continuously run until user enters y

FETCH Retrives address from the instruction memory as a string of 1s/0s.

Adds 4 to current address and stores the result.

sets values to false to reset control unit, then calls method to set control values with opcode.

resets values in control unit

mux 5 is set by a combination of branch and the result of [ALU](#)

always goes to read register1

goes to read register 2 and mux1

goes to mux1

gets last15 didgets of instruction

get j type address

function code

Shifts the value to the left (value used for address in jtype)

test for shift left

must wait for result of Mux5

Sends reg2 and reg3 to mux1

write register gets value from mux1 if a writeReg occurs this stores the register to be written to

Conversion to bitset so a conversion to int can be done

Converts from bitset to integer

Readreg accepts decimal value as a string, so we use to string

Converts values from hex (how its stored in register) to binary

test for mux1

sign extend accepts bitset.

second mux decided if imm or register 2 should go to the [ALU](#) extended in bin, val at reg2 is in hex rn

calls second mux to determine second input for alu

The following code acts as the [ALU](#) control for ALU3

SLT, ADD, SUB, SLT

Add

Subtract

SLT instruction, not yet implemented in [ALU](#)

Result gets "equal" or "not equal"

runs for lw and sw

if this runs it is a branch instruction AND the branch condition passed. Basically the AND in the data path.

Not needed but avoids an unused warning for mux5 if no branches

if there is a memory write (sw) it occurs here

valAtReg2 is value to be written address to be written to is alu3 result(needs to be converted to hex)

sends result of the alu to the 3rd multiplexor

runs if op uses a memory read, and sends value to the 3rd multiplexor aluresult needs to be translated to hex

checks to see if it is writing to a register from mux3.

remember string writeRegister holds in the reg code below should write the given value to the register

so binary can be changed to int

Shifts the previously extended address by 2 bits(needed for b and j)

Add this value to current PC value(This doesn't make sense to me...)

result that is going to program counter

Updates program counter with correct address

prints the control fields, register memory and datamemory after each instruction if printMemoryContents is set to true.

The documentation for this class was generated from the following file:

- Test.cpp

Index

- ALU, [3](#)
 - performOperation, [3](#)
- binaryToHex
 - Converter, [5](#)
- ControlUnit, [4](#)
 - getMemWrite, [4](#)
 - getRegDest, [4](#)
 - setToZero, [4](#)
 - setValues, [5](#)
- Converter, [5](#)
 - binaryToHex, [5](#)
 - hexToBinary, [6](#)
 - hexify, [6](#)
 - hextoint, [7](#)
 - inttobinary, [7](#)
 - inttohex, [7](#)
- DataMemory, [8](#)
 - DataMemory, [8](#)
 - dmemPrintFinal, [8](#)
 - getdata, [9](#)
 - writeMem, [9](#)
- dmemPrintFinal
 - DataMemory, [8](#)
- getCurrentAddress
 - ProgramCounter, [16](#)
- getFunctField
 - OpcodeTable, [13](#)
- getInstruction
 - InstructionMemory, [11](#)
- getMemWrite
 - ControlUnit, [4](#)
- getNum
 - RegisterFile, [17](#)
- getOpcode
 - OpcodeTable, [13](#)
- getOpcodeField
 - OpcodeTable, [13](#)
- getRegDest
 - ControlUnit, [4](#)
- getdata
 - DataMemory, [9](#)
- hexToBinary
 - Converter, [6](#)
- hexify
 - Converter, [6](#)
- hextoint
 - Converter, [7](#)
- IMMposition
 - OpcodeTable, [13](#)
- Instruction, [10](#)
 - setValues, [10](#)
- InstructionMemory, [11](#)
 - getInstruction, [11](#)
 - printContents, [11](#)
- inttobinary
 - Converter, [7](#)
- inttohex
 - Converter, [7](#)
- isFormatCorrect
 - Parser, [15](#)
- isIMMLabel
 - OpcodeTable, [13](#)
- moveAddressTo
 - ProgramCounter, [16](#)
- Multiplexor, [12](#)
 - setFlow, [12](#)
- OpcodeTable, [12](#)
 - getFunctField, [13](#)
 - getOpcode, [13](#)
 - getOpcodeField, [13](#)
 - IMMposition, [13](#)
 - isIMMLabel, [13](#)
 - OpcodeTable, [13](#)
 - RDposition, [14](#)
 - RSposition, [14](#)
 - RTposition, [14](#)
- Parser, [14](#)
 - isFormatCorrect, [15](#)
 - Parser, [15](#)
- performOperation
 - ALU, [3](#)
- printContents
 - InstructionMemory, [11](#)
- ProgramCounter, [15](#)
 - getCurrentAddress, [16](#)
 - moveAddressTo, [16](#)
- RDposition
 - OpcodeTable, [14](#)
- RSposition
 - OpcodeTable, [14](#)
- RTposition
 - OpcodeTable, [14](#)

- RegisterEntry, [16](#)
- RegisterFile, [17](#)
 - getNum, [17](#)
 - RegisterFile, [17](#)
 - writeReg, [17](#)
- run
 - Stimulation, [19](#)
 - Tester, [32](#)
- setFlow
 - Multiplexor, [12](#)
- setToZero
 - ControlUnit, [4](#)
- setValues
 - ControlUnit, [5](#)
 - Instruction, [10](#)
- ShiftLeftTwo, [18](#)
- SignExtend, [18](#)
- Stimulation, [19](#)
 - run, [19](#)
- Tester, [32](#)
 - run, [32](#)
- writeMem
 - DataMemory, [9](#)
- writeReg
 - RegisterFile, [17](#)