CS 301 FinalProject

Generated by Doxygen 1.8.14

Contents

Chapter 1

Class Index

1.1 Class List

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

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 ALUControl Class Reference

```
#include <ALUControl.h>
```

Static Public Member Functions

- static std::string getOperation (int aluop1, int aluop0, std::string functCode)

2.2.1 Detailed Description

SignExtend Class Basel Arafat, Nicholas Biffis,Vincent Camp & Will Saada Computer Orginization CS 301 Spring 2018

2.2.2 Member Function Documentation

2.2.2.1 getOperation()

```
static std::string ALUControl::getOperation (
            int aluop1,
            int aluop0,
            std::string functCode )  [inline], [static]
```

SLT, ADD, SUB, SLT

Result gets "equal" or "not equal"

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

- ALUControl.h

2.3 ConfigurationParser Class Reference

```
#include <ConfigurationParser.h>
```

Public Member Functions

- **ConfigurationParser** (std::string s)
- void Parseit ()
- std::string **getConfigurationfile** ()
- std::string **getprogramInputFile** ()
- std::string **getdataMemoryFile** ()
- std::string **getregisterFile** ()
- std::string **getoutputMode** ()
- std::string **getoutputFile** ()
- bool **getdebugMode** ()
- bool **getprintMemoryContents** ()
- bool **getwriteToFile** ()

2.3.1 Detailed Description

ConfigurationFile Parsr Class Basel Arafat, Nicholas Biffis,Vincent Camp & Will Saada Computer Orginization CS 301 Spring 2018

2.3.2 Member Function Documentation

2.3.2.1 Parseit()

```
void ConfigurationParser::Parseit ( )
```

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.

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

- ConfigurationParser.h
- ConfigurationParser.cpp

2.4 ControlUnit Class Reference

```
#include <ControlUnit.h>
```

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**

2.4.1 Detailed Description

Control Unit Class Basel Arafat, Nicholas Biffis,Vincent Camp & Will Saada Computer Orginization CS 301 Spring 2018 The Control Unit class stores a variables for each of the control fields in out simple mips processor. The set values function is called given an opcode to initialize the values for each new instruction. After each instruction the set to zero method resets the control object such that all fiels are false.

2.4.2 Member Function Documentation

2.4.2.1 getRegDest()

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

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

2.4.2.2 setToZero()

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

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

2.4.2.3 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

2.5 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 inttobinarry (int a)

2.5.1 Member Function Documentation

2.5.1.1 binaryToHex()

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

Converts a given binary value to hex

Parameters

binary	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

2.5.1.2 hexify()

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

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

Parameters

s	string to be converted to it's hex equivalent

Returns

mystring the hexadeciaml conversion of the provided string

2.5.1.3 hexToBinary()

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

Converts a given hex value to binary

Parameters

hex	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

2.5.1.4 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

2.5.1.5 inttobinarry()

```
static std::string Converter::inttobinarry (
            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

2.5.1.6 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

2.6 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)

2.6.1 Constructor & Destructor Documentation

2.6.1.1 DataMemory()

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

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

Parameters

filename	file that will be checked for syntactic correctness

2.6.2 Member Function Documentation

2.6.2.1 dmemPrintFinal()

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

Prints the data memory to the Output file

Parameters

memOutputFile	file that data memory will be written to

2.6.2.2 getdata()

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

Given an Address, returns the data associated with that address

Parameters

theAddress	specified address to gather data from

Returns

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

2.6.2.3 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

address	address to be written to
val	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

2.7 Instruction Class Reference

```
#include <Instruction.h>
```

Public Member Functions

- Instruction (Opcode op, Register rs, Register rt, Register rd, int imm)
- void setValues (Opcode op, Register rs, Register rt, Register rd, int imm)
- Opcode getOpcode ()

  Returns the various fields for the Instruction.
- Register **getRS** ()
- Register **getRD** ()
- Register **getRT** ()
- int **getImmediate** ()
- string getString ()
- void setEncoding (string s)
- string getEncoding ()

2.7.1 Detailed Description

Instruction Class Basel Arafat, Nicholas Biffis,Vincent Camp & Will Saada Computer Orginization CS 301 Spring 2018 This class provides an internal representation for a MIPS assembly instruction. Any of the fields can be queried. Additionally, the class stores a 32 bit binary encoding of the MIPS instruction.

2.7.2 Constructor & Destructor Documentation

2.7.2.1 Instruction()

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

You can specify all the fields to initialize the Instruction

Parameters

op	current instruction opcode
rs	current instruction register source
rt	current instruction register source 2
rd	current instruction register destination
imm	current instruction immediate value

2.7.3 Member Function Documentation

2.7.3.1 getEncoding()

```
string Instruction::getEncoding ( )  [inline]
```

Returns string representing the 32 binary encoding of MIPS instruction

Returns

myEncoding string representing the 32 binary encoding of MIPS instruction

2.7.3.2 getString()

```
string Instruction::getString ( )
```

Returns a string which represents all of the fields

Returns

s.str() a string representing all of the fields

2.7.3.3 setEncoding()

```
void Instruction::setEncoding (
            string s )  [inline]
```

Stores the 32 bit binary encoding of MIPS instruction passed in

Parameters

s	MIPS instruction

2.7.3.4 setValues()

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

Allows you to specify all the fields of the Instruction

Parameters

op	current instruction opcode
rs	current instruction register source
rt	current instruction register source 2
rd	current instruction register destination
imm	current instruction immediate value

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

- Instruction.h
- Instruction.cpp

2.8 InstructionMemory Class Reference

Public Member Functions

- InstructionMemory ()

  Default constructor.
- InstructionMemory (std::string filename)
- Instruction getInstruction (std::string address)
- bool isValidInstruction (std::string theAddress)
- void printContents ()
- void **imemPrintFinal** (string OutputFile)

2.8.1 Constructor & Destructor Documentation

2.8.1.1 InstructionMemory()

```
InstructionMemory::InstructionMemory (
            std::string filename )
```

Accesses and parses through the Instruction memory

Parameters

filename	file to be accessed and store given instructions

2.8.2 Member Function Documentation

2.8.2.1 getInstruction()

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

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

Parameters

theAddress	Address given to access and get the associated Instruction

Returns

Instruction associated with given address

2.8.2.2 isValidInstruction()

```
bool InstructionMemory::isValidInstruction (
             std::string theAddress )
```

Checks to make sure that a valid instruction is given

Parameters

theAddress	Address given to check if it's instruction is valid

Returns

false if the instruction at the given address is invalid, and true otherwise

2.8.2.3 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

2.9 Multiplexor Class Reference

```
#include <Multiplexor.h>
```

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 chosen
- 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.

2.9.1 Detailed Description

Multiplexor Class Basel Arafat, Nicholas Biffis,Vincent Camp & Will Saada Computer Orginization CS 301 Spring 2018

2.9.2 Member Function Documentation

2.9.2.1 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

2.10 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)
- int RSposition (Opcode o)
- int RTposition (Opcode o)
- int RDposition (Opcode o)
- int IMMposition (Opcode o)
- bool isIMMLabel (Opcode o)
- InstType getInstType (Opcode o)
- string getOpcodeField (Opcode o)
- string getFunctField (Opcode o)
- string name (Opcode o)

2.10.1 Constructor & Destructor Documentation

2.10.1.1 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 = "";

2.10.2 Member Function Documentation

2.10.2.1 getFunctField()

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

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

Parameters

opcode	of the current instruction

Returns

string representing the binary encoding of the funct field

2.10.2.2 getInstType()

```
InstType OpcodeTable::getInstType (
             Opcode o )
```

Given an Opcode, returns instruction type.

Parameters

opcode	of the current instruction

Returns

the type of instruction

2.10.2.3 getOpcode()

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

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

Parameters

str	MIPS assembly mnemonic

Returns

opcode for the specified mnemonic

2.10.2.4 getOpcodeField()

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

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

Parameters

opcode	of the current instruction

Returns

string representing the binary encoding of the opcode

2.10.2.5 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.

Parameters

opcode	of the current instruction

Returns

the position of the IMM field

2.10.2.6 isIMMLabel()

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

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

Parameters

opcode	of the current instruction

Returns

true/false as to whether the instruction expects label

2.10.2.7 name()

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

Given an opcode return the name of the instruction associated with the opcode

Parameters

opcode	of the current instruction

Returns

name of instruction

2.10.2.8 numOperands()

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

Given an Opcode, returns number of expected operands.

Parameters

opcode	for current instruction

Returns

number of operands for the specified opcode

2.10.2.9 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.

Parameters

opcode	of the current instruction

Returns

the position of the RD field

2.10.2.10 RSposition()

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

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

Parameters

opcode	of the current instruction

Returns

the position of the RS/RT/RD/IMM fields respectively

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

Parameters

opcode	of the current instruction

Returns

the position of the RS field

2.10.2.11 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.

Parameters

opcode	of the current instruction

Returns

the position of the RS field

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

- Opcode.h
- Opcode.cpp

2.11 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)

2.11.1 Constructor & Destructor Documentation

2.11.1.1 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

2.11.2 Member Function Documentation

2.11.2.1 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

2.12 ProgramCounter Class Reference

This h creates the guidlines 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)

2.12.1 Detailed Description

This h creates the guidlines for the program counter.

ProgramCounter Class Basel Arafat, Nicholas Biffis,Vincent Camp & Will Saada Computer Orginization CS 301 Spring 2018forward declarations needed can go below

2.12.2 Member Function Documentation

2.12.2.1 getCurrentAddress()

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

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

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

Returns

string Returns current address stored in the program counter

2.12.2.2 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

Parameters

string	New address being moved to memory

Returns

string New address, used for testing

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

- ProgramCounter.h
- ProgramCounter.cpp

2.13 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

2.14 RegisterFile Class Reference

Public Member Functions

- RegisterFile ()

    Default constructor.
- RegisterFile (string registerFile)
- Register getNum (string reg)
- std::string readReg (string reg)
- std::string writeReg (string reg, string value)
- void printContents ()

    Prints contents of the register file.
- void **PrintFinal** (std::string regOutputFile)

2.14.1 Constructor & Destructor Documentation

2.14.1.1 RegisterFile() [1/2]

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

Default constructor.

Register Table for access

2.14.1.2 RegisterFile() [2/2]

```
RegisterFile::RegisterFile (
              string registerFile )
```

Checks to make sure file is opened correctly, and establishes the delimiter

Parameters

registerFile	file given to be evaluated for syntacitc correctness

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

2.14.2 Member Function Documentation

2.14.2.1 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.

Parameters

reg	register to get set number equivalent of

Returns

NumRegisters[i].number if the register corresponded with a number and 32 otherwise

2.14.2.2 printContents()

```
void RegisterFile::printContents ( )
```

Prints contents of the register file.

Prints contents of register file.

2.14.2.3 readReg()

```
std::string RegisterFile::readReg (
             string reg )
```

Given a string representing a MIPS register operand, returns the value associated with said register. If the string is not a valid register, returns the number of registers

Parameters

reg	register to be read from

Returns

myRegister[reg] the value stored at the given register

2.14.2.4 writeReg()

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

Given a string representing a MIPS register operand and a specified value, stores the value within said register.

Parameters

reg	register to be written to
value	value to be stored within the specified address

Returns

temp[reg] the value originally stored in the specified address

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

- RegisterFile.h
- RegisterFile.cpp

2.15 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

2.15.1 Detailed Description

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

ShiftleftTwo Class Basel Arafat, Nicholas Biffis,Vincent Camp & Will Saada Computer Orginization CS 301 Spring 2018

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

- ShiftLeftTwo.h

2.16 SignExtend Class Reference

```
#include <SignExtend.h>
```

Static Public Member Functions

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

2.16.1 Detailed Description

SignExtend Class Basel Arafat, Nicholas Biffis,Vincent Camp & Will Saada Computer Orginization CS 301 Spring 2018

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

- SignExtend.h

2.17 Stimulation Class Reference

Public Member Functions

- **Stimulation** (string filename)
- void **getFiles** ()
- void run ()

2.17.1 Member Function Documentation

2.17.1.1 run()

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

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

}

/// ///
/// ///
///

Parser is run through instruction memory to initializes 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

Converstion 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

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 writting 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 exstended 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

Index