



MIPS Datapath and Control Unit Simulator

CSE116

Developed by: Mostafa Hazem | Mohamed El Ghamry

iONEX Group

Tel (+20) 120 724 3513 | 112 300 0478

Website www.ionexgroup.com.eg

Email Technical_support@ionexgroup.com

IMPLEMENTATION

Classes

- CSE116_Project
- RegisterNode
- Registers
- MemoryNode
- Memory

CSE116_Project

This is our main class where programs execution takes place

RegisterNode

The RegisterNode class is used to implement a customized node for registers, it holds both name and value of the register, with setters and getters included

Registers

Registers is an array of RegisterNode which represents our 32 registers with initial zero value

MemoryNode

The MemoryNode class is used to implement our memory node

Memory

Memory is an array of MemoryNode with size of 1024 elements

Methods

- StartAddress
- readInstructions
- loadDataInMem
- printReg
- printIns
- setOpAndFn
- setCtrlUnit
- readRd
- readRs
- readRt
- readImm
- readOffset
- readShamt
- readAddress
- printData
- store
- execute
- printDataPath
- DecToBin
- BinToDec
- printRegValues
- printMem
- storeCode
- printCode
- credits

StartAddress

This method takes the initial program counter entered by the user and assign it to three different static fields (pc, pcStart, and pcPlusFour)

readInstructions

The read instructions method is a method which gives the user the opportunity to select instructions to be executed, also by each instruction this method uses the rest of methods to set opcode and function code, set the control unit signals, define the rs field, rd field, rt field, in case of R-Type, or rs, rt, and immediate fields in I/J-Type. It keeps reading the instructions until user terminates by typing “-1”

loadDataInMem

It asks the user if he wants to load data in memory at the beginning or not, if yes; it takes a register which holds base address, base address, after how many elements is that value assigned, and finally the value the user wish to store

printReg

The print register method prints out to the user the 32 registers names and numbers only

printIns

The print instructions method prints out to the user the valid instructions supported by this mips data path simulator only

setOpAndFn

This method takes two arguments opcode and function code, and sets the opcode and function code correctly to the suitable instruction

setCtrlUnit

This method takes 9 arguments regDest, branch, memRead, memToReg, ALUOp, memWrite, ALUSrc, RegWrite and ALUCtrl; then sets all these to their correct fields

readRs/readRt/readRd/readImm/readShamt/readOffset/readAddress

These methods read the rs, rd, rt, imm, shamt, offset, and address from the users and assigns them to their suitable fields

printData

The following method task is to print the machine code

store

This method takes the machine code and stores it in an array list with number of elements equal to number of instructions entered by user

execute

The following method operates depending on the opcode and function code for each instruction (machine code) in the array list, it adds, subtracts, jumps, ... etc.

printDataPath

Prints out to the user the Datapath of the mips code

DecToBin

Converts from decimal to binary

BinToDec

Converts the binary to decimal

printRegValues

This method prints all the registers and their values after execution

printMem

This method prints the values of the memory

storeCode / printCode

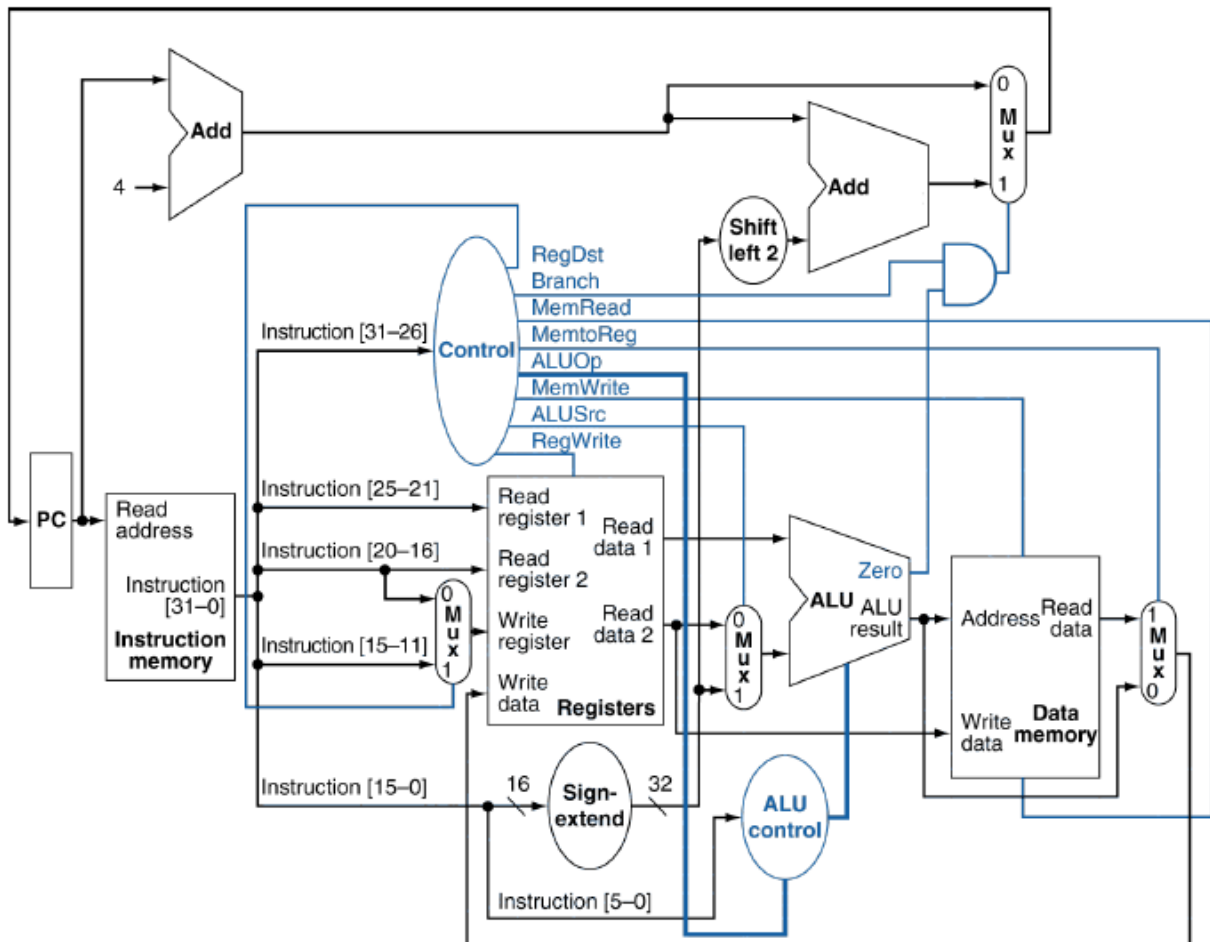
Those methods store the mips code and prints it out as a bunch

credits

prints out to the user the credits of the following simulator

DATAPATH

Appendix: MIPS datapath (single cycle implementation)



We have only added to the control unit one more signal, it is the jump signal.

ASSUMPTIONS

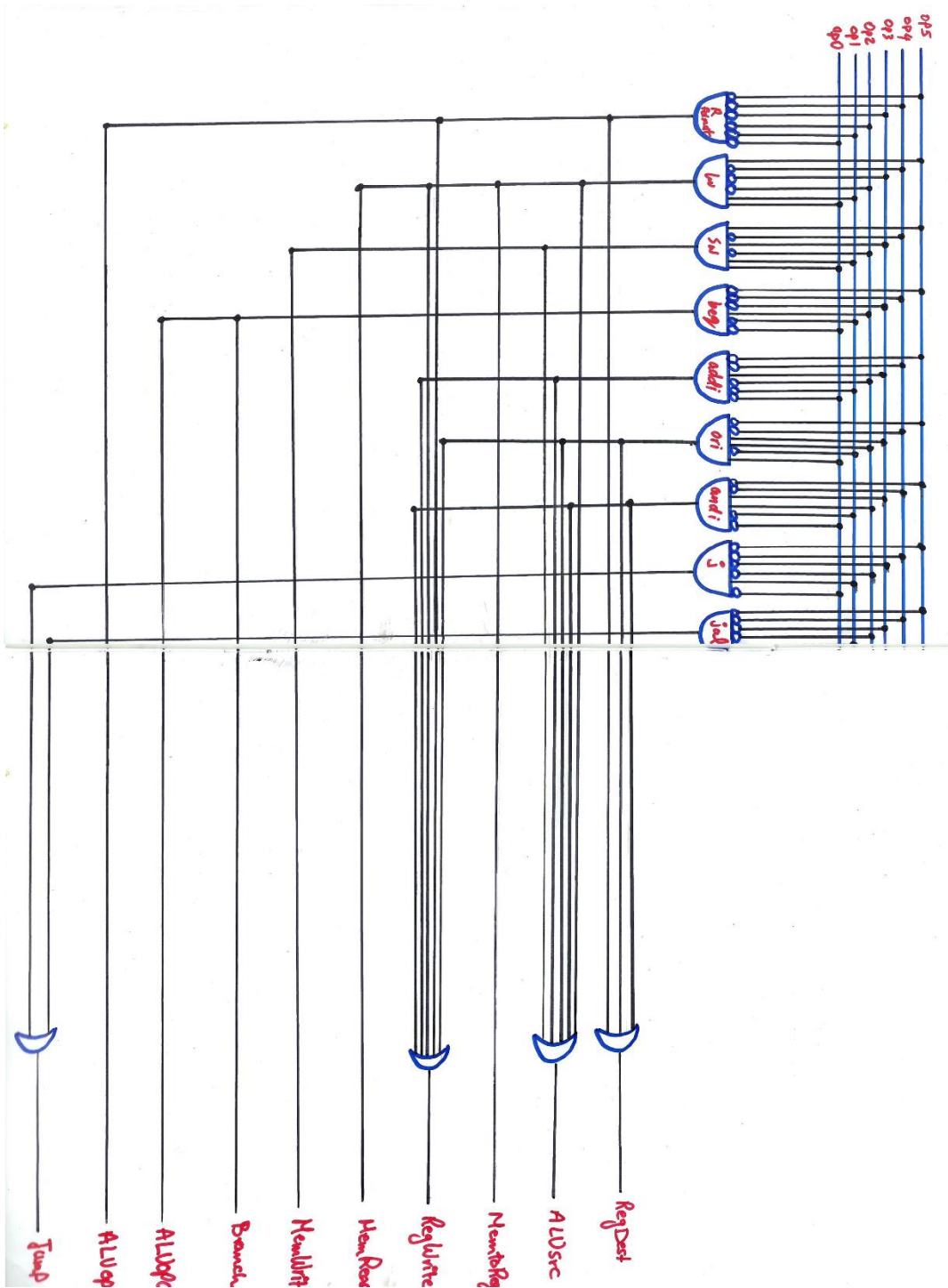
We have assumed in the control unit any don't care value as a zero for simplicity reasons.

Also, we have assumed the following:

`sll` -> `ALUCtrl = 0` | `ori/andi` -> `ALUOp = 0`, `ALUSrc = 1`

`nor` -> `ALUCtrl = 0` | `jr` -> `ALUCtrl = 2`

LOGIC DIAGRAM



USER GUIDE

```
run:
-- MIPS SIMULATOR PROJECT --
```

```
Enter the starting address:
512|
```

```
Do you want to load any data in the memory?
[1] Yes
[2] No
1|
```

```
Select the register that holds the array base address:
```

```
[0] $zero
[2] $v0
[3] $v1
[4] $a0
[5] $a1
[6] $a2
[7] $a3
[8] $t0
[9] $t1
[10] $t2
[11] $t3
[12] $t4
[13] $t5
[14] $t6
[15] $t7
[16] $s0
[17] $s1
[18] $s2
[19] $s3
[20] $s4
[21] $s5
[22] $s6
[23] $s7
[24] $t8
[25] $t9
[29] $sp
[31] $ra
16|
```

```
Base address:
64|
```

```
After how many elements from the base address you want to add the value?
2|
```

```
Enter the value you wish to store in your memory:
1997|
```

Do you want to add any more data?

[1] Yes

[2] No

2|

Program is building...

Select your instruction **Enter -1 to terminate**

[1] add

[2] addi

[3] lw

[4] sw

[5] sll

[6] and

[7] andi

[8] or

[9] ori

[10] nor

[11] beq

[12] j

[13] jal

[14] jr

[15] slt

1|

Select rd

[0] \$zero

[2] \$v0

[3] \$v1

[4] \$a0

[5] \$a1

[6] \$a2

[7] \$a3

[8] \$t0

[9] \$t1

[10] \$t2

[11] \$t3

[12] \$t4

[13] \$t5

[14] \$t6

[15] \$t7

[16] \$s0

[17] \$s1

[18] \$s2

[19] \$s3

[20] \$s4

[21] \$s5

[22] \$s6

[23] \$s7

[24] \$t8

[25] \$t9

[29] \$sp

[31] \$ra

8|

Select rs

[0] \$zero

[2] \$v0

[3] \$v1

[4] \$a0

[5] \$a1

[6] \$a2

[7] \$a3

[8] \$t0

[9] \$t1

[10] \$t2

[11] \$t3

[12] \$t4

[13] \$t5

[14] \$t6

[15] \$t7

[16] \$s0

[17] \$s1

[18] \$s2

[19] \$s3

[20] \$s4

[21] \$s5

[22] \$s6

[23] \$s7

[24] \$t8

[25] \$t9

[29] \$sp

[31] \$ra

9|

Select rt

[0] \$zero

[2] \$v0

[3] \$v1

[4] \$a0

[5] \$a1

[6] \$a2

[7] \$a3

[8] \$t0

[9] \$t1

[10] \$t2

[11] \$t3

[12] \$t4

[13] \$t5

[14] \$t6

[15] \$t7

[16] \$s0

[17] \$s1

[18] \$s2

[19] \$s3

[20] \$s4

[21] \$s5

[22] \$s6

[23] \$s7

[24] \$t8

[25] \$t9

[29] \$sp

[31] \$ra

10|

Select your instruction **Enter -1 to terminate**

- [1] add
- [2] addi
- [3] lw
- [4] sw
- [5] sll
- [6] and
- [7] andi
- [8] or
- [9] ori
- [10] nor
- [11] beq
- [12] j
- [13] jal
- [14] jr
- [15] slt
- 1|

Fetching in progress...

Do you want to print a value stored in the memory?

- [1] Yes
- [2] No
- 1

Select the register that holds the array base address:

- [0] \$zero
- [2] \$v0
- [3] \$v1
- [4] \$a0
- [5] \$a1
- [6] \$a2
- [7] \$a3
- [8] \$t0
- [9] \$t1
- [10] \$t2
- [11] \$t3
- [12] \$t4
- [13] \$t5
- [14] \$t6
- [15] \$t7
- [16] \$s0
- [17] \$s1
- [18] \$s2
- [19] \$s3
- [20] \$s4
- [21] \$s5
- [22] \$s6
- [23] \$s7
- [24] \$t8
- [25] \$t9
- [29] \$sp
- [31] \$ra
- 16|

Enter the index to be printed:

2

Value assigned: 1997

Do you want to print any other values stored?

[1] Yes

[2] No

2

Do you want to print the values in all the registers?

[1] Yes

[2] No

1

• Values in Registers

- o \$zero = 0
- o \$at = 0
- o \$v0 = 0
- o \$v1 = 0
- o \$a0 = 0
- o \$a1 = 0
- o \$a2 = 0
- o \$a3 = 0
- o \$t0 = 0
- o \$t1 = 0
- o \$t2 = 0
- o \$t3 = 0
- o \$t4 = 0
- o \$t5 = 0
- o \$t6 = 0
- o \$t7 = 0
- o \$s0 = 64
- o \$s1 = 0
- o \$s2 = 0
- o \$s3 = 0
- o \$s4 = 0
- o \$s5 = 0
- o \$s6 = 0
- o \$s7 = 0
- o \$t8 = 0
- o \$t9 = 0
- o \$k0 = 0
- o \$k1 = 0
- o \$gp = 0
- o \$sp = 0
- o \$fp = 0
- o \$ra = 0

MIPS Code:
add \$t0 \$t1 \$t2

Machine Code is:
00000001001010100100000000100000

• Datapath

- o PC starting address: 512
- o PC+4 adder output: 516
- o Instruction memory output: 000000 01001 01010 01000 00000 100000
- o opcode field of instruction: 000000
- o rs field of instruction: 01001
- o rt field of instruction: 01010
- o rd field of instruction: 01000
- o RegDestination Mux output: 01000
- o Sign-extender output: 000000000000000001000000000100000
- o Shift-2 output: 000000000000000001000000000100000
- o Target address adder: 66180
- o Function code of instruction: 100000
- o Read data 1: 0
- o Read data 2: 0
- o ALU second input: 0
- o ALU output: 0
- o Zero flag: 0
- o Data memory output: N/A
- o MemtoReg Mux output: 0
- o AND gate output: 0

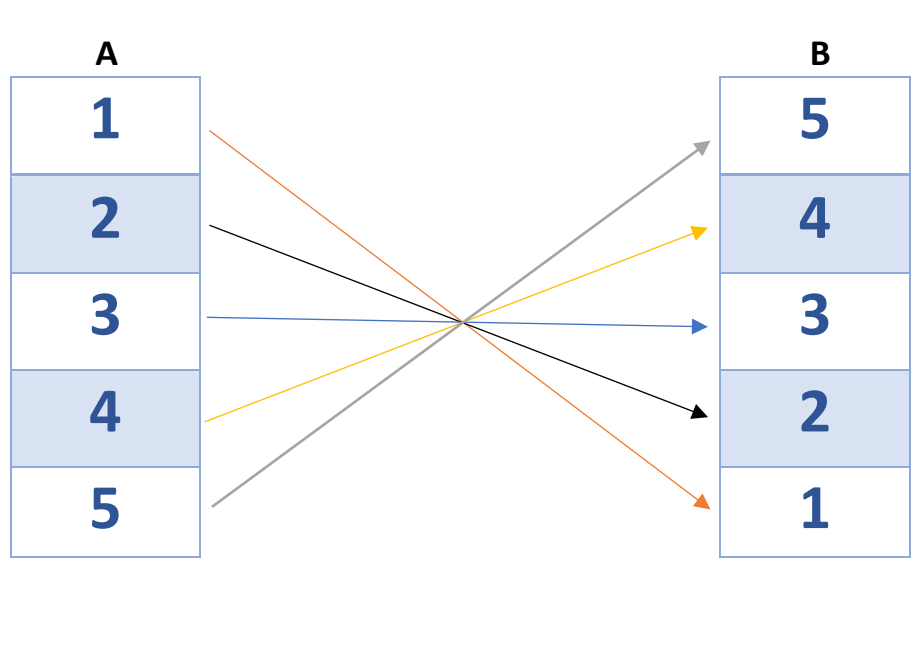
• Control signals

- o RegDest: 1
- o Branch: 0
- o MemRead: 0
- o MemtoReg: 0
- o ALUOp: 10
- o MemWrite: 0
- o ALUSrc: 0
- o RegWrite: 1
- o ALU control output: 0010

Credits #####
MIPS SIMULATOR v1.0 ##
Created by: ##
Mohamed El Ghamry ##
Mostafa Hazem ##
~(iONEX Group)~ ##
#####

LIST OF PROGRAMS

Reversing an array program

	<pre>addi \$t0 \$s0 0 addi \$t1 \$zero 4 sll \$t1 \$t1 2 add \$t1 \$t1 \$s1 #loop (-6 from beq): lw \$t2 0(\$t0) sw \$t2 0(\$t1) addi \$t0 \$t0 4 addi \$t1 \$t1 -4 slt \$t3 \$t1 \$s1 beq \$t3 \$zero -6</pre>
<p>A is a 5-integer array whose starting address is in \$s0, the values 1,5 are stored in A in ascending order.</p> <p>Assume that B is another 5-integer array whose starting address is in \$s1</p> <p>\$s0 holds the base address which is 1000 \$s1 holds the base address which is 2000</p> <p>First, we will load the values 1,2,3,4,5 in the memory respectively starting from the base address 1000 which is stored in \$s0, and then we will load the base address 2000 into \$s1, before starting writing the program</p> <p>the values stored in array A will be stored in reverse order in array B</p>	

Program to test the rest of instructions

500	ori \$t0 \$zero 7	# \$t0 = 7
504	or \$t1 \$t0 \$t1	# \$t1 = 7
508	jal 130	# 520/4 = 130
512	nor \$t4 \$t2 \$t1	# \$t4 = -8
516	j 133	# 532/4 = 133
520	andi \$t2 \$t0 6	# \$t2 = 6
524	and \$t3 \$t2 \$t1	# \$t3 = 6
528	jr \$ra	# \$ra = 512
532	addi \$t5 \$t0 10	# \$t5 = 17
536	#end	# pc+4

ROLES

- Mostafa Hazem:
DecToBin / DecToBinStr / BinToDec / BinToDecStr / Flip / twosComplement /
printRegValues / printMem / storeCode / printCode
Debugging: R-Type Instructions (add, and, or, nor, jr, sll, slt)
- Mohamed El Ghamry:
Store / Execute / printDataPath / printData / readRs / readRd / readRt / readImm /
readAddress / readOffset / setCtrlUnit / setOpAndFn / printReg / printIns / StartAddress
/ LoadDataInMem / readInstructions
Debugging: the rest of instructions (lw, sw, j, jal, beq, ori, andi, addi)