Computer Architecture Project

MIPS Datapath Simulator

Project By:

Ahmed Malatawy 22-1452

Amina Zoheir 22-1041

Dayna Hany 22-3794

Maged Shalaby 22-4408

1. *Project Summary*

A software simulation of the single cycle MIPS Datapath, also designed to support the extra instructions not supported by the version presented in the lectures of the course. (Instructions supported can be found in the last page).

A visual representation of a MIPS program in execution is presented, simulating the inputs & outputs of the separate stages of each instruction on a drawing of the datapath. An assembler for standard MIPS programs written using any of the supported instructions is also included.

1. *Datapath Used*

Almost identical Datapath as the one in the lecture, for the single cycle Datapath implementation (Lec. 5, Slide 22).

However for the following instructions: *{Jump and Link, Jump, Branch if Equal, Branch if not equal},* we assume the following difference:

To support labels, instead of having an address value that needs to be shifted and offset in the instruction, we keep a separate memory for labels, which maps labels to their corresponding addresses. Each of the aforementioned instructions will then have a pointer to the label map, which will point it to the direct address it should jump or branch to.

Also due to the software architecture used, it could be argued that there is a register file present to keep track of the different fields of an instruction, such as the source and destination registers, etc.. However, it can be assumed that whenever a stage requires an attribute from the instruction, a wire is connecting the instruction coming

**Important Note:** Two separate memories are used for instructions and for data, and both by default start at 0. It was meaningless for the user to specify the starting address for instructions, due to its implementation as an ArrayList<String>.

1. *Control Unit*

No hardware modules or control units were designed, since it was an implementation decision to simulate the Datapath as stages, each stage having its correct inputs and generating the desired outputs. This was done, however, in a fashion that does not directly mimic hardware, meaning that there were no specific classes created for Multiplexers, Adders, etc..

However, a separate control unit was designed, in hopes of having the GUI show the control signals as well, for educational purposes.

1. *Software Architecture*

* **Simulator**: Main class responsible for running loaded instructions one by one. Calls the different methods for running each stage *(Fetch, Decode, Execute, Memory, Write Back*), and keeps track of the different components present, such as the Register File, Data Memory Unit, Instruction Memory Unit, and Label Map. Also keeps track of the current instruction, to allow passing the instruction parameters (type, source and destination registers, etc..), without having to have each stage pass a list of parameters to its following stage.
* **Instruction:** Holds the indices of registers required by this instruction, the format of the instruction, as well as other relevant information that could be required.

**Note:** *The formats used are not exactly the same as the traditional format, they were separated according to the type of instruction, as well as how the instruction is written in MIPS. For instance, “add” and “addi” are both format 0, but are separated by having a boolean “immediate”.*

* **RegisterFile:** Holds an array representing the 32 registers, and allows to read or write to any of them (except writing to the zero register). Register names can be translated into their corresponding index by using the RegisterMapper class.
* **MemoryUnit:** Holds an array of 2^20 bytes by default, for keeping program data.
* **LogicUnit:** Allows for the execution of any arithmetic or logic operations required by an instruction, simulating the ALU (execute stage).
* **Datapath:** The main class responsible for drawing the datapath, using the different components present in the “GUI” package.
* **Exceptions:**  Thrown to handle different hazards, such as memory out of bounds, writing to zero register, etc..

1. *Test Programs*

*Program 1 (Testing Memory Instructions):*

addi s3, zero, 255

sb s3, 0(zero)

lb s2, 0(zero)

lbu s2, 0(zero)

addi s3, zero, 32768

sh s3, 4(zero)

lh s2, 4(zero)

lhu s2, 4(zero)

addi s3, zero, 100000

sw s3, 8(zero)

lw s2, 8(zero)

*Program 2 (Testing loops):*

add s2, s0, s1

lw s2, 4(s2)

sub s2, s2, s0

addi s2, s2, 1

sw s2, 4(s2)

addi s3, zero, 20

addi s2, zero, 0

j FIB

addi s2,s2,1

FIB:

addi s2, s2, 2

bne s2, s3, FIB

lui s2, 1

1. *Simulating Program*

* Import Java Project.
* Run Simulator file.
* Press *Next Step* to execute one stage of the current instruction, *Next Instruction to* execute a whole instruction, or *Run All* to execute the whole program.

*NB: Default memory values are loaded via “Data 1.txt”, and MIPS program is loaded from “Program 1.txt”. This can be changed from the load\_data(), load\_instructions() methods.*

*NB II: GUI is currently not fully complete, however the contents of register file, memory unit are updated correctly, and the program prints to the console the outputs of the execute and memory stages, as well as the contents of registers s2 & s3 after each instruction.*

*VII. Work Division*

* **Ahmed Malatawy:** Responsible for the implementation of the Jump and Branch instructions and their corresponding Datapath.
* **Amina Zoheir:**Responsible for the implementation and design of the Graphical User Interface.
* **Dayna Hany:**Responsible for the implementation of the Logic Unit and the execute stage.
* **Maged Shalaby:**Responsible for the implementation of the Memory Unit & the supported memory instructions.

***Note:*** *Most other classes and functions were implemented in a team setting.*

1. *Instructions Supported*

*add, addi, sub,*

*lw, lh, lhu, lb, lbu, sw, sh, sb, lui,*

*sll, srl, and, andi, or, ori, nor,*

*beq, bne, j, jal, jr,*

*slt*