

***Lahore University of Management Sciences***

***School of Science and Engineering***

EE 320 Project Report

**Project Name:**

***MIPS Single Cycle Processor***

***(8-Bit Architecture)***

**Project Developers:**

*Kamran Ali 2013-10-0174*

*Muhammad Saqlain 2013-10-0170*

*Waleed Hassan 2013-10-0142*

*Rai Ammad Khan 2013-10-0205*

**Group NO: 16**

============================================================

***Table of Contents*** **============================================================**

[**1.0**](http://www-ee.eng.hawaii.edu/%7Esasaki/EE361/Fall99/ChrisChan/Report.html#1.0)**OBJECTIVE**

[2.0](http://www-ee.eng.hawaii.edu/%7Esasaki/EE361/Fall99/ChrisChan/Report.html#2.0)   A BRIEF INTRODUCTION

[3.0](http://www-ee.eng.hawaii.edu/%7Esasaki/EE361/Fall99/ChrisChan/Report.html#3.0)    INSTRUCTION FORMAT

[4.0](http://www-ee.eng.hawaii.edu/%7Esasaki/EE361/Fall99/ChrisChan/Report.html#3.0)    DESIGN

**4.1 Instructions**

**4.2 Design Behind the Instruction Decoding and the Implementation:**

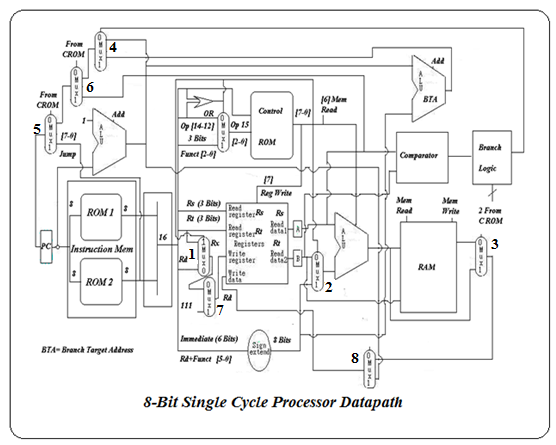
[5.0](http://www-ee.eng.hawaii.edu/%7Esasaki/EE361/Fall99/ChrisChan/Report.html#4.0)    IMPLEMENTATION

[5.1](http://www-ee.eng.hawaii.edu/%7Esasaki/EE361/Fall99/ChrisChan/Report.html#4.1)   User Interface

[5.2](http://www-ee.eng.hawaii.edu/%7Esasaki/EE361/Fall99/ChrisChan/Report.html#4.2)   Datapath

[5.3](http://www-ee.eng.hawaii.edu/%7Esasaki/EE361/Fall99/ChrisChan/Report.html#4.3)   Execution

**6.0 More Efficient Implementation**



***1.0 OBJECTIVE:***

The objective of the project is to develop the operation of MIPS Single Cycle processor. In this project the [Datapath](http://www-ee.eng.hawaii.edu/%7Esasaki/EE361/Fall99/ChrisChan/Report.html#Datapath) and Control units of the processor for implementation of the MIPS instruction set are constructed, and an implementation of a subset of the core [MIPS instruction](http://www-ee.eng.hawaii.edu/%7Esasaki/EE361/Fall99/ChrisChan/Report.html#MIPS%20Instructions) set is also included. And of course after doing the project the following main points would be clearly understood.

* 1. The main modules of the MIPS processor.
  2. The [function](http://www-ee.eng.hawaii.edu/%7Esasaki/EE361/Fall99/ChrisChan/Report.html#function_of_each_module) of each module.
  3. The three types of MIPS:
     1. I-type
     2. R-type
     3. J-type
  4. The four classes of MIPS:
     1. The memory-reference instructions load word (***lw***) and store word (***sw***)
     2. The arithmetic-logical instruction ***and, or, add,*** ***sub, ori, andi,*** and ***addi***
     3. The branch instructions ***beq*** and ***bne***.
     4. The jump instruction ***j,jal*** and ***jr***
  5. Modules required when running a particular instruction of each type of MIPS

***2.0 A brief Introduction:***

**MIPS** (originally an acronym for **Microprocessor without Interlocked Pipeline Stages**) is an [instruction set architecture](http://en.wikipedia.org/wiki/Instruction_set_architecture) (ISA) developed by [MIPS Technologies](http://en.wikipedia.org/wiki/MIPS_Technologies). The early MIPS architectures were 32-bit, and later versions were 64-bit. Multiple revisions of the MIPS instruction set exist, including MIPS I, MIPS II, MIPS III, MIPS IV, MIPS V, MIPS32, and MIPS64. The current revisions are MIPS32 (for 32-bit implementations) and MIPS64 (for 64-bit implementations). MIPS32 and MIPS64 define a control register set as well as the instruction set.

The core MIPS instruction set architecture detemines many aspects of the implementation, which includes an integer arithmetic-logic instruction, the memory-reference instructions, and the branch instructions. Much of what needs to be done to implement these instructions is the same, independent of the exact class of instruction. We have tried to keep our implementation and instruction format as close to that of MIPS architecture.

***3.0 Instruction Format:***

The instruction formats for all types are given below.

***R-Type:***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| opcode | rs | rt | rd | funct |

4 Bits 3 Bits 3 Bits 3 Bits 3 Bits

***I-Type:***

|  |  |  |  |
| --- | --- | --- | --- |
| opcode | rs | rt | Immediate |

4 Bits 3 Bits 3 Bits 6 Bits

***J-Type:***

|  |  |  |
| --- | --- | --- |
| opcode | Irrelevant 4 bits | Address |

3 Bits 8 Bits

***4.0 Design:***

The design of this processor includes the following modules.

***4.1 Instructions:***

|  |  |  |  |
| --- | --- | --- | --- |
| Instruction | Op-code | Funct | Control Rom |
| ADD | 0000 | 001 | 11100110 |
| SUB | 0000 | 000 | 11011000 |
| AND | 0000 | 010 | 11101101 |
| OR | 0000 | 011 | 111110X1 |
| ADDI | 0001 | NA | 11100110 |
| ANDI | 0010 | NA | 11101101 |
| OR | 0011 | NA | 111110X1 |
| LW | 0100 | NA | 11100110 |
| SW | 0101 | NA | 00100110 |
| BNE | 1000 | NA | 01100000 |
| BEQ | 1001 | NA | 01110000 |
| JR | 1100 | NA | 01000100 |
| JUMP | 1010 | NA | 01001000 |
| JAL | 1011 | NA | 10001011 |
|  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Instruction*** | ***MUX1*** | ***MUX2*** | ***MUX3*** | ***MUX4*** | ***MUX5*** | ***MUX6*** | ***MUX7*** | ***MUX8*** | ***Reg write*** | ***Mem read*** |
| ***ADD*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***1*** | ***1*** |
| ***SUB*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***1*** | ***1*** |
| ***AND*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***1*** | ***1*** |
| ***OR*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***1*** | ***1*** |
| ***JR*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***1*** | ***0*** | ***0*** | ***0*** | ***1*** |
| ***ADDI*** | ***1*** | ***1*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***1*** | ***1*** |
| ***ANDI*** | ***1*** | ***1*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***1*** | ***1*** |
| ***ORI*** | ***1*** | ***1*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***1*** | ***1*** |
| ***BNE*** | ***1*** | ***1*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***1*** |
| ***BEQ*** | ***1*** | ***1*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***1*** |
| ***LW*** | ***1*** | ***1*** | ***1*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***1*** | ***1*** |
| ***SW*** | ***1*** | ***1*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** | ***0*** |
| ***J*** | ***1*** | ***1*** | ***0*** | ***0*** | ***1*** | ***0*** | ***0*** | ***0*** | ***0*** | ***1*** |
| ***JAL*** | ***1*** | ***1*** | ***0*** | ***0*** | ***1*** | ***0*** | ***1*** | ***1*** | ***1*** | ***1*** |

***4.2 Design Behind the Instruction Decoding and the Implementation:***

When the instruction reached the Decode stage, first of all we divide it into three parts, according to the instruction format given in the report above.

1. R-type
2. I-type
3. J-type

We are using **ONE ROM** as our main control. Here is how we designed that,

1. Most significant 2 bits are Reg-write and Mem-write respectively.
2. The other 6 bits are very frugally and efficiently used to do 2 things at one time.
3. Control the ALU.
4. A) The bits 5, 4 are used for branch (Branch/Not branch and BNE/BEQ).

B) 3, 2, 1, 0 are used for controlling MUXES 5, 6, 7, 8 respectively.

As **Load Word** is only one instruction which used both ALU and there is a need to control a MUX after the memory only in this instruction, we have implemented a MINTERM OF THE OPCODE. The load word instruction is at 0100 op-code. So, this MINTERM is implemented and that mux lets the memory value to be selected to go into the register to be written.

MUXes 1, 2 are operated by the OR of all the OP-CODE BITS. Thus we have implemented an OR of all the op-code bits and the output controls the muxes.

Now we further define some bits. The most significant bit of the instruction or of the op-code is given the name ALU-OP. The purpose of this bit is to make sure following things,

1. **As we are using only one ROM, we have to make sure the MUXES 5,6,7,8 controlled by the ROM and,**
2. **The branch bits which control the branch logic and thus the branch/not branch decision don’t interfere with the proper instruction execution when the control the ALU. So, this ALUop bit is “ANDED” with those controls and then sent to the muxes, and the branch decision is also ANDED with this BEFORE sending to control the Branch Target selection Mux.**
3. **For this reason, as we will show, we have stored the controls of those instructions which use ALU in the 0000 – 0111 locations of the ROM, to keep ALUOP = 0.**
4. **THIS DECODING IS DONE ON 2ND DECODER WHICH ALSO CONTROLS THE RAM ACORDING TO THE “NOT LOGIC” OF THE READ/WRITE BUFFERS AND “G” INPUT WHICH HAS BEEN DISCUSSED IN THE PROPOSAL AS WELL.**

The SIGN EXTENSION is done by **ANDING** the **NOT** of the ALU MODE (Which tells us that the instruction is not LOGIC, but arithmetic) with the last bit of the 6 bit immediate. And thus **CONCATENATE** WITH THE 6-BIT immediate to get 8 bit sign extended output.

**The Register used in the counter is high edge triggered, RAM is low edge triggered and the register file is also high edge triggered. The NOT of the clock is provided to the Program Counter register and the CLOCK “ANDED” WITH REG-WRITE to the register file and the CLOCK “OR” WITH REG-WRITE to the RAM.**

**Thus when the clock is low the instruction is FETCHED. When it goes high, the Write Back process occurs updating the register file and carrying out any memory operation etc.**

**ONE MORE THING which is another efficiency of this design IS THAT WE HAVE LUMPED THE:**

1. **Add and addi on one location of the of the control ROM (of course the need the same ALU and register control and the I-type muxes are already being controlled with the help of OR of all the opcode bits)**
2. **And and Andi;**
3. **Or and Ori;**
4. **And to of course to get to this design without any interference, the subtract instruction is placed on 0000 location of the control ROM.**

**THUS WHEN WE have 0000 opcode, we use the op-code MS bit and the funct bits to address control ROM location. And when we have opcode 0001, 0010, 0011 we use opcode bits to address the control ROM. The address lines of the ROM are 4.**

This implementation is done by a mux which gives the **funct** bits out or the **least** **three** bits of **the op-code. And this mux of course is on the MAIN DECODER and is controlled by the OR of ALU bits.**

***5.0 Implementaion:***

The implementation of this processor requires the following components.

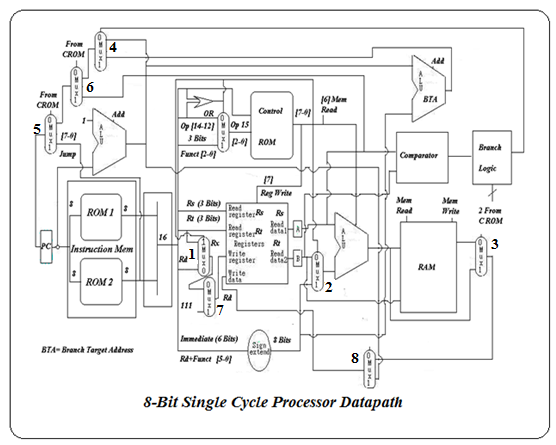
* ***8-bit registers***
* ***Read Only Memory (ROM) ICs***
* ***AND, OR, XOR and Inverter ICs***
* ***Full adder ICs***
* ***8-bit ALU ICs***
* ***8-bit Buffer ICs***
* ***8-bit Register file***
* ***Random Access Memory (RAM) ICs***
* ***Power Supply***
* ***Male pin connectors***
* ***Data Busses***
* ***Vero Boards***
* ***PCB***
* ***LEDs***

***5.1 User Interface:***

When the clock is low the instruction is fetched and executed in all the hardware when it is high the write back process is completed. LEDs are there at every step where we can check our instruction beign fetched, decoded and executed at every hardware level.

***5.2 Datapath:***

The datapath for this 8 bit single cycle processor is shown below with the control muxes.

******

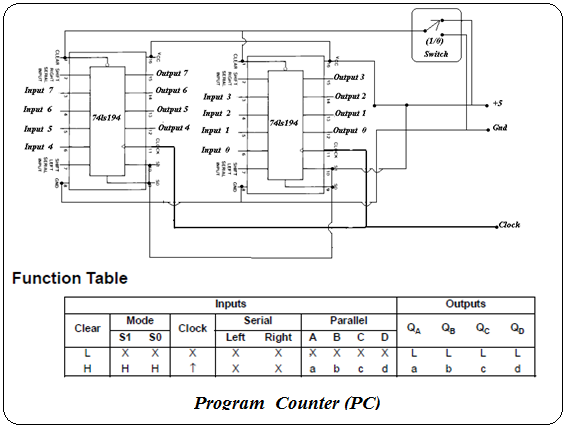
***5.3 Execution:***

In order to execute we first divide this datapath into following parts.

1. *PC*
2. *Instruction Memory*
3. *Instruction Decode*
4. *Register File*
5. *ALU*
6. *Sign Extention*
7. *Comparator*
8. *Branch Logic*
9. *Memory Reference Instructions Control (lw,sw)*
10. *Muxes*
11. *Branch Target Address Calculator*

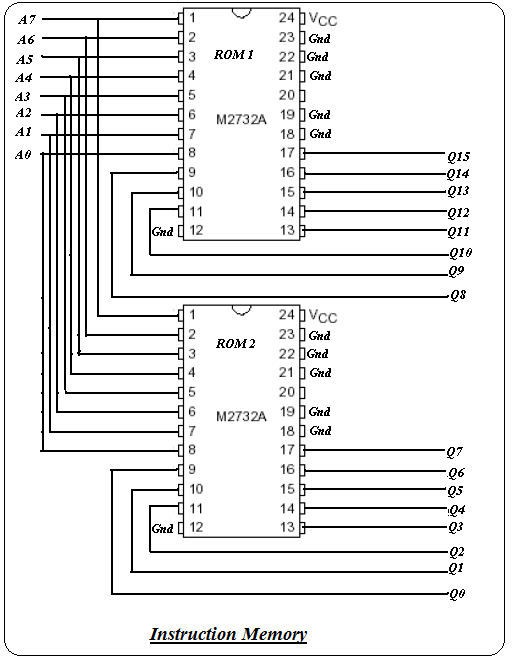
* ***PC:***

The program counter consists of two 4-bit shift registers (bidirectional) whose IC is 74ls194. We use a switch in order to reset. The datapath of PC is given below. The function table is also shown below.



* ***Instruction Memory:***

The instruction memory cosists of two ROMs. The address lines of both ROMs are shorted and they are 8-bit wide. The other 4 most bits are grounded. As the data is stored in the ROMs in bytes so our instruction memory output is 16-bit wide. The layout of it is shown below.



* ***Instruction decode:***

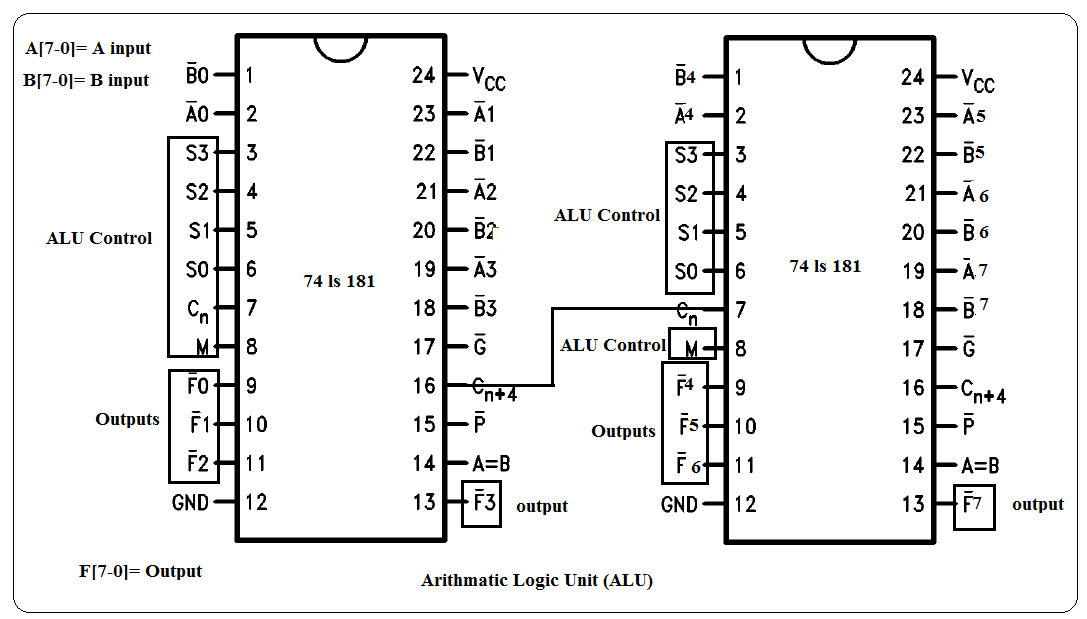
The instruction decoding is done on vero board by segregating the fields of the instruction set architecture. These fields are then givento the respective hardwares.

* ***Register File:***

The register file consists of buffers and registors. In this processor we use 8-bit register file. The data input and output is also 8-bit wide. There are three types of address select lines rs (3-bits), rt (3-bits) and rd(3-bits). When we want to write data into some register we apply a clock edge.

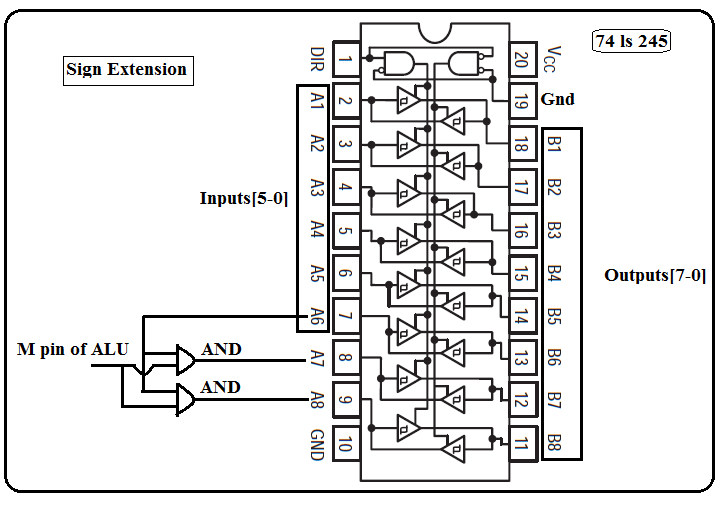
* ***ALU:***

The arithmetic logic unit (ALU) is shown below.ALU controls are common. The controls for arithmatic operation are provided from the control ROM.



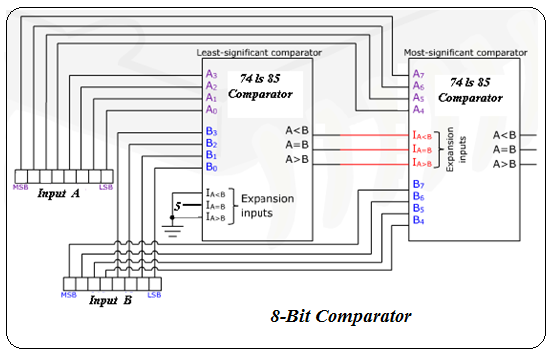
* ***Sign Extention:***

The sign extension circuit is shown below.



* ***Comparator:***

The comparator circuit consists of two 4-bit comparator ICs 74ls85. They are cascaded in such a way that when the inputs are equal then the output (A=B) is high. This is done by making input (A=B) of least significant comparator high and making the other inputs (A<B) and (A>B) low.



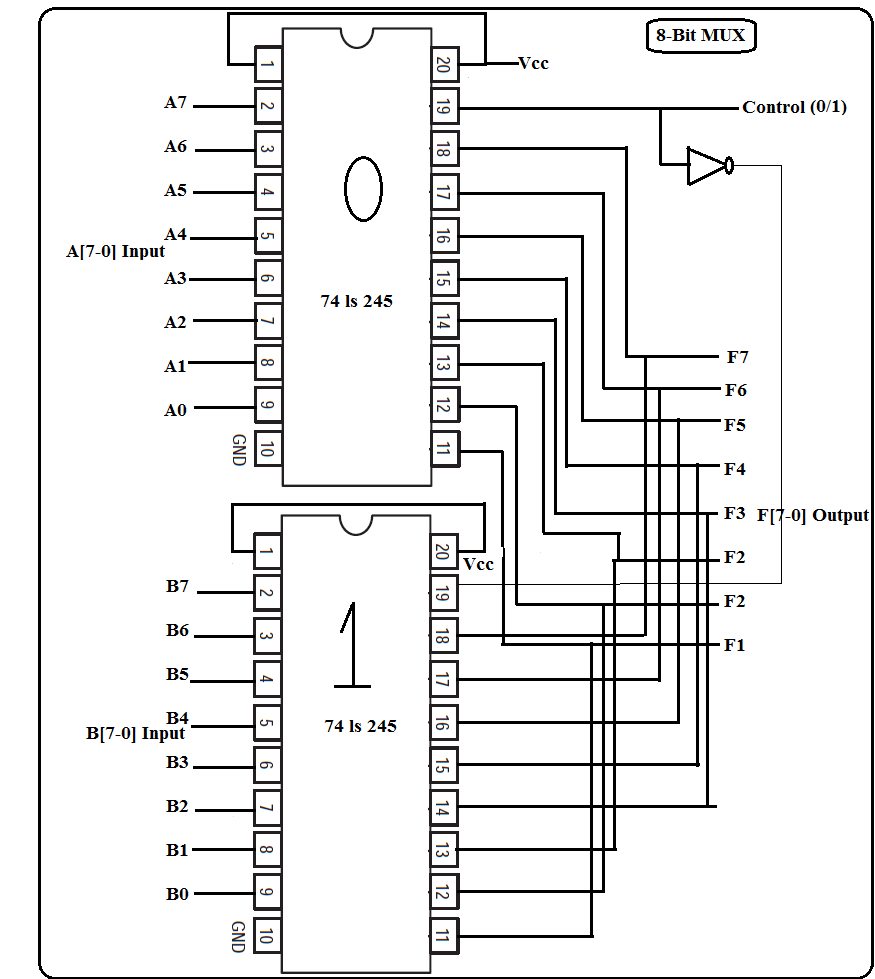
* ***Branch Logic:***

The branch logic for the control of MUX whose inputs are branch taget address and PC+1 is given below.

******

* ***MUX:***

8-bit MUX is designed using buffers. Implementation of this design on vero board is easier then using MUX ICs. The circuit is shown below.



***6.0 More Efficient Implementation:***

The main difference between a multi-cycle and single cycle MIPS processor is the fact  that the multi-cycle implementation takes more than one clock cycle to execute an instruction, but the exact number of cycles needed to execute one instruction can vary depending on the type of instruction. Although the single cycle implementation may appear on the surface to be a faster, more efficient data path, the multi-cycle data path actually has many more advantages. The multi cycle implementation allows a functional unit (such as an Arithmetical Logic Unit or ALU) to be used more than once per instruction provided it's used on different clock cycles. This in turn reduces the amount of hardware required.

Allowing the processor to take multiple cycles to execute an instruction is actually an advantage. For a single cycle processor, it would take a fixed amount of time to execute an instruction, no matter how simple it is. So a simple addition instruction will take as long as a more complicated memory access or memory write instruction. This is not the case in the multi-cycle processor. If the instruction is a relatively simple instruction, then it takes the processor fewer cycles and therefore less time to execute. If the instruction is relatively more complicated, it can use more cycles, requiring a small amount of time as the single cycle processor to execute. The ability to allow instructions to have varying numbers of clock cycles to execute and the reuse of functional units are the major advantages of the multi-cycle processor.

We were planning to do some pipelining but time was limited. However we have designed our processor to be as efficient as possible at every level of hardware implementation.