|  |  |
| --- | --- |
|  | DESIGN AND IMPLEMENTATION OF A RISC PROCESSOR |
|  |  |
| 15/11/2013 | SUBMITTED TO PROF. KARAN KARUN |
|  | *S N MUNENDRA – 12CO72*  *SHIVA SAGAR B – 12CO83*  *SIDDARTHA L K – 12CO91*  *THEJDEEP G - 12CO99* |
|  |  |

DESIGN AND IMPLEMENTATION OF A RISC PROCESSOR

AN ABSTRACT AND LOGISIM IMPLEMENTATION

WHAT DOES RISC MEAN ?

**Reduced instruction set computing**, or **RISC** is a CPU design strategy based on the insight that simplified (as opposed to complex) instructions can provide higher performance if this simplicity enables much faster execution of each instruction. A computer based on this strategy is a *reduced instruction set computer*, also called *RISC*. The opposing architecture is known as complex instruction set computing, i.e. CISC.

Various suggestions have been made regarding a precise definition of RISC, but the general concept is that of a system that uses a small, highly-optimized set of instructions, rather than a more specialized set of instructions often found in other types of architectures. Another common trait is that RISC systems use the load/store architecture  where memory is normally accessed only through specific instructions, rather than accessed as part of other instructions like add. RISC processors use pipelining , and their instruction set allows tight control of their pipeline by the software. Optimizing compilers perform instruction scheduling so as to exploit the parallelism offered by the pipelined data path. RISC processors achieve high performance at low cost. In addition silicon area , design cost and design time are also reduced .

Well-known RISC families include [DEC Alpha](http://en.wikipedia.org/wiki/DEC_Alpha), [AMD 29k](http://en.wikipedia.org/wiki/AMD_29k), [ARC](http://en.wikipedia.org/wiki/ARC_International), [ARM](http://en.wikipedia.org/wiki/ARM_architecture), [Atmel AVR](http://en.wikipedia.org/wiki/Atmel_AVR), [Blackfin](http://en.wikipedia.org/wiki/Blackfin), [Intel i860](http://en.wikipedia.org/wiki/Intel_i860) and [i960](http://en.wikipedia.org/wiki/Intel_i960), [MIPS](http://en.wikipedia.org/wiki/MIPS_architecture), [Motorola 88000](http://en.wikipedia.org/wiki/Motorola_88000), [PA-ISC](http://en.wikipedia.org/wiki/PA-RISC), [Power](http://en.wikipedia.org/wiki/Power_Architecture) (including [PowerPC](http://en.wikipedia.org/wiki/PowerPC)), [SuperH](http://en.wikipedia.org/wiki/SuperH), and[SPARC](http://en.wikipedia.org/wiki/SPARC).

# CHARACTERISTICS AND DESIGN PHILOSOPHY

INSTRUCTION SET

A common misunderstanding of the phrase "reduced instruction set computer" is the mistaken idea that instructions are simply eliminated, resulting in a smaller set of instructions. In fact, over the years, RISC instruction sets have grown in size, and today many of them have a larger set of instructions than many CISC CPUs.[[19]](http://en.wikipedia.org/wiki/Reduced_instruction_set_computing#cite_note-19)[[20]](http://en.wikipedia.org/wiki/Reduced_instruction_set_computing#cite_note-20) Some RISC processors such as the [PowerPC](http://en.wikipedia.org/wiki/PowerPC) have instruction sets as large as the CISC [IBM](http://en.wikipedia.org/wiki/IBM) [System/370](http://en.wikipedia.org/wiki/System/370), for example; conversely, the DEC [PDP-8](http://en.wikipedia.org/wiki/PDP-8)—clearly a CISC CPU because many of its instructions involve multiple memory accesses—has only 8 basic instructions and a few extended instructions.

The term "reduced" in that phrase was intended to describe the fact that the amount of work any single instruction accomplishes is reduced—at most a single data memory cycle—compared to the "complex instructions" of CISC CPUs that may require dozens of data memory cycles in order to execute a single instruction.[[21]](http://en.wikipedia.org/wiki/Reduced_instruction_set_computing#cite_note-21) In particular, RISC processors typically have separate instructions for I/O and data processing.

HARDWARE UTILIZATION

For any given level of general performance, a RISC chip will typically have far fewer [transistors](http://en.wikipedia.org/wiki/Transistor) dedicated to the core logic which originally allowed designers to increase the size of the register set and increase internal parallelism.

Other features that are typically found in RISC architectures are:

* Uniform instruction format, using a single word with the opcode in the same bit positions in every instruction, demanding less decoding;
* Identical [general purpose registers](http://en.wikipedia.org/wiki/General_purpose_register), allowing any register to be used in any context, simplifying compiler design (although normally there are separate [floating point](http://en.wikipedia.org/wiki/Floating_point) registers);
* Simple [addressing modes](http://en.wikipedia.org/wiki/Addressing_mode), with complex addressing performed via sequences of arithmetic and/or load-store operations;
* Few data types in hardware, some CISCs have [byte](http://en.wikipedia.org/wiki/Byte) [string](http://en.wikipedia.org/wiki/String_(computer_science)) instructions, or support [complex numbers](http://en.wikipedia.org/wiki/Complex_number); this is so far unlikely to be found on a RISC.

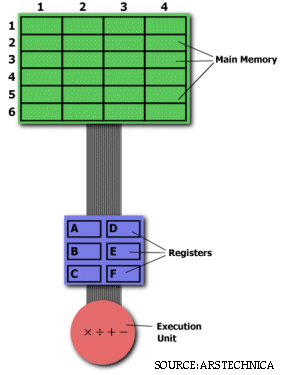
Exceptions abound, of course, within both CISC and RISC.

# RISC v/s CISC

The simplest way to examine the advantages and disadvantages of RISC architecture is by contrasting it with it's predecessor: CISC (Complex Instruction Set Computers) architecture.

**Multiplying Two Numbers in Memory**

Given below is a diagram representing the storage scheme for a generic computer. The main memory is divided into locations numbered from (row) 1: (column) 1 to (row) 6: (column) 4. The execution unit is responsible for carrying out all computations. However, the execution unit can only operate on data that has been loaded into one of the six registers (A, B, C, D, E, or F). Let's say we want to find the product of two numbers - one stored in location 2:3 and another stored in location 5:2 - and then store the product back in the location 2:3.



**The CISC Approach**   
The primary goal of CISC architecture is to complete a task in as few lines of assembly as possible. This is achieved by building processor hardware that is capable of understanding and executing a series of operations. For this particular task, a CISC processor would come prepared with a specific instruction (we'll call it "MULT"). When executed, this instruction loads the two values into separate registers, multiplies the operands in the execution unit, and then stores the product in the appropriate register. Thus, the entire task of multiplying two numbers can be completed with one instruction:

MULT 2:3, 5:2

MULT is what is known as a "complex instruction." It operates directly on the computer's memory banks and does not require the programmer to explicitly call any loading or storing functions. It closely resembles a command in a higher level language. For instance, if we let "a" represent the value of 2:3 and "b" represent the value of 5:2, then this command is identical to the C statement "a = a \* b."

One of the primary advantages of this system is that the compiler has to do very little work to translate a high-level language statement into assembly. Because the length of the code is relatively short, very little RAM is required to store instructions. The emphasis is put on building complex instructions directly into the hardware.

**The RISC Approach**  
RISC processors only use simple instructions that can be executed within one clock cycle. Thus, the "MULT" command described above could be divided into three separate commands: "LOAD," which moves data from the memory bank to a register, "PROD," which finds the product of two operands located within the registers, and "STORE," which moves data from a register to the memory banks. In order to perform the exact series of steps described in the CISC approach, a programmer would need to code four lines of assembly:

LOAD A, 2:3  
LOAD B, 5:2  
PROD A, B  
STORE 2:3, A

At first, this may seem like a much less efficient way of completing the operation. Because there are more lines of code, more RAM is needed to store the assembly level instructions. The compiler must also perform more work to convert a high-level language statement into code of this form.

|  |  |
| --- | --- |
| **CISC** | **RISC** |
| Emphasis on hardware | Emphasis on software |
| Includes multi-clock complex instructions | Single-clock, reduced instruction only |
| Memory-to-memory: "LOAD" and "STORE" incorporated in instructions | Register to register: "LOAD" and "STORE" are independent instructions |
| Small code sizes, high cycles per second | Low cycles per second, large code sizes |
| Transistors used for storing complex instructions | Spends more transistors on memory registers |

However, the RISC strategy also brings some very important advantages. Because each instruction requires only one clock cycle to execute, the entire program will execute in approximately the same amount of time as the multi-cycle "MULT" command. These RISC "reduced instructions" require less transistors of hardware space than the complex instructions, leaving more room for general purpose registers. Because all of the instructions execute in a uniform amount of time (i.e. one clock), pipelining is possible.

Separating the "LOAD" and "STORE" instructions actually reduces the amount of work that the computer must perform. After a CISC-style "MULT" command is executed, the processor automatically erases the registers. If one of the operands needs to be used for another computation, the processor must re-load the data from the memory bank into a register. In RISC, the operand will remain in the register until another value is loaded in its place.

**The Performance Equation**  
The following equation is commonly used for expressing a computer's performance ability:

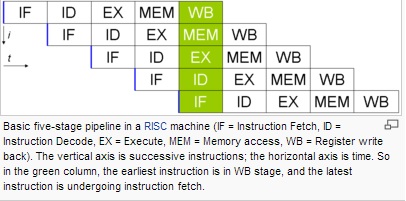
http://www-cs-faculty.stanford.edu/~eroberts/courses/soco/projects/risc/risccisc/options/performanceeq.gif

The CISC approach attempts to minimize the number of instructions per program, sacrificing the number of cycles per instruction. RISC does the opposite, reducing the cycles per instruction at the cost of the number of instructions per program.

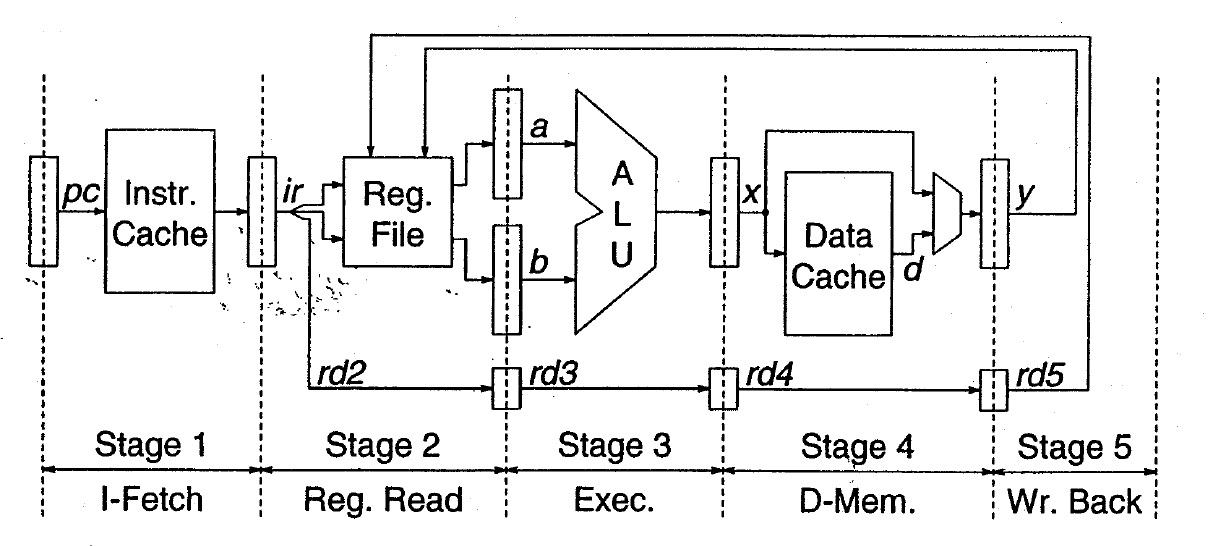
# PIPELINING IN RISC

Pipelining is the principal method to achieve high performance in the operation of a processor. RISC proved that pipelining can be applied to low-end systems provided that the instruction set is simplified. Pipelining, a standard feature in RISC processors, is much like an assembly line. Because the processor works on different steps of the instruction at the same time, more instructions can be executed in a shorter period of time.

The basic five stage RISC Pipeline



The simplified data path of the 5 stage RISC pipeline is :



The five stages of the RISC pipeline are :

1. Instruction Fetch

The Instruction [Cache](http://en.wikipedia.org/wiki/Cache_(computing)) on these machines had a latency of one cycle, meaning that if the instruction was in the cache, it would be ready on the next [clock cycle](http://en.wikipedia.org/wiki/Clock_cycle). During the Instruction Fetch stage, a 32-bit instruction was fetched from the cache.

The [Program Counter](http://en.wikipedia.org/wiki/Program_Counter), or PC, is a register responsible for holding the address of the current instruction. It feeds into the PC predictor which then sends the[Program Counter](http://en.wikipedia.org/wiki/Program_Counter) (PC) to the Instruction Cache to read the current instruction. At the same time, the PC predictor predicts the address of the next instruction by incrementing the PC by 4 (all instructions were 4 bytes long).

1. Instruction Decode

At the time the register file was read, instruction issue logic in this stage determined if the pipeline was ready to execute the instruction in this stage. If not, the issue logic would cause both the Instruction Fetch stage and the Decode stage to stall. On a stall cycle, the stages would prevent their initial flip-flops from accepting new bits.

If the instruction decoded was a branch or jump, the target address of the branch or jump was computed in parallel with reading the register file. The branch condition is computed after the register file is read, and if the branch is taken or if the instruction is a jump, the PC predictor in the first stage is assigned the branch target, rather than the incremented PC that has been computed.

1. Instruction Execute

The Execute stage is where the actual computation occurs. Typically this stage consists of an Arithmetic and Logic Unit, and also a bit shifter. It may also include a multiple cycle multiplier and divider.

The Arithmetic and Logic Unit is responsible for performing boolean operations (and, or, not, nand, nor, xor, xnor) and also for performing integer addition and subtraction. Besides the result, the ALU typically provides status bits such as whether or not the result was 0, or if an overflow occurred.

The bit shifter is responsible for shift and rotations.

1. Memory Access

If data memory needs to be accessed, it is done so in this stage.

During this stage, single cycle latency instructions simply have their results forwarded to the next stage. This forwarding ensures that both single and two cycle instructions always write their results in the same stage of the pipeline, so that just one write port to the register file can be used, and it is always available.

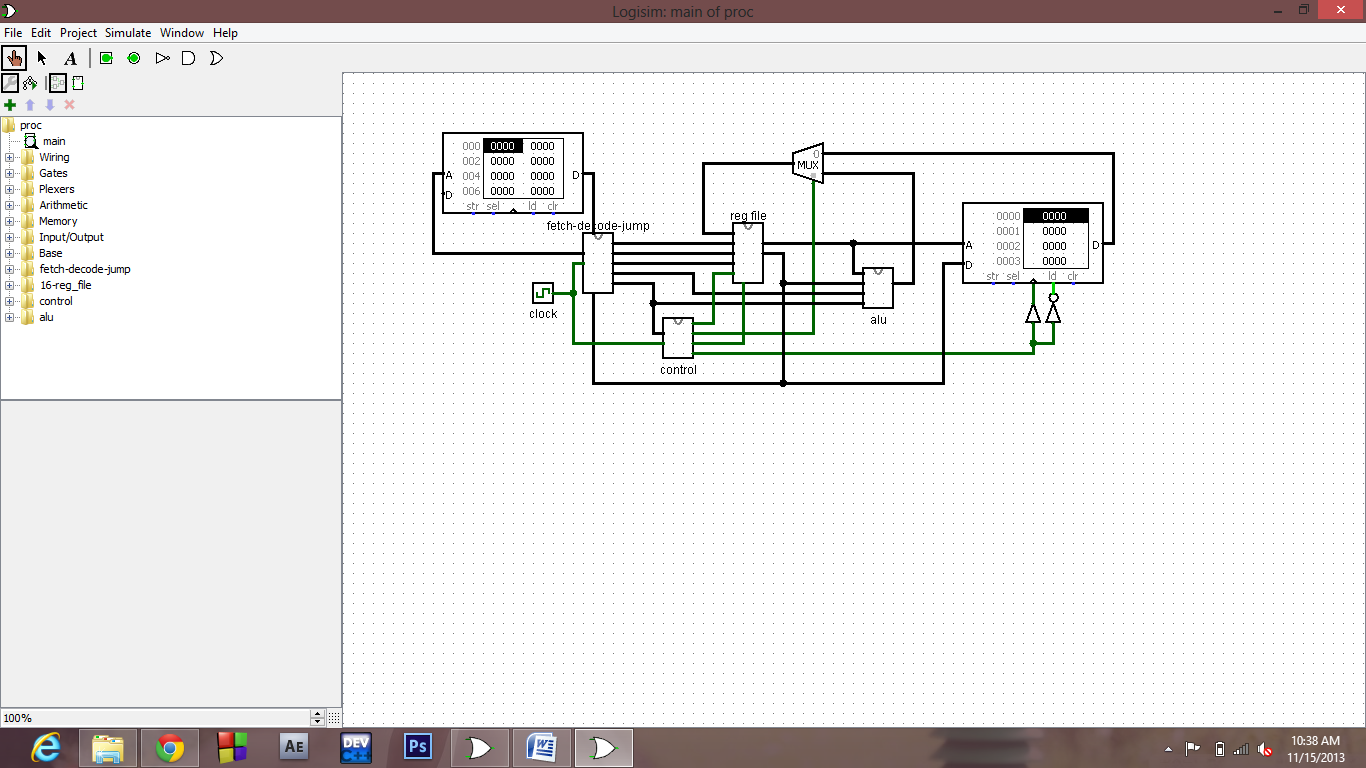
1. Write Back

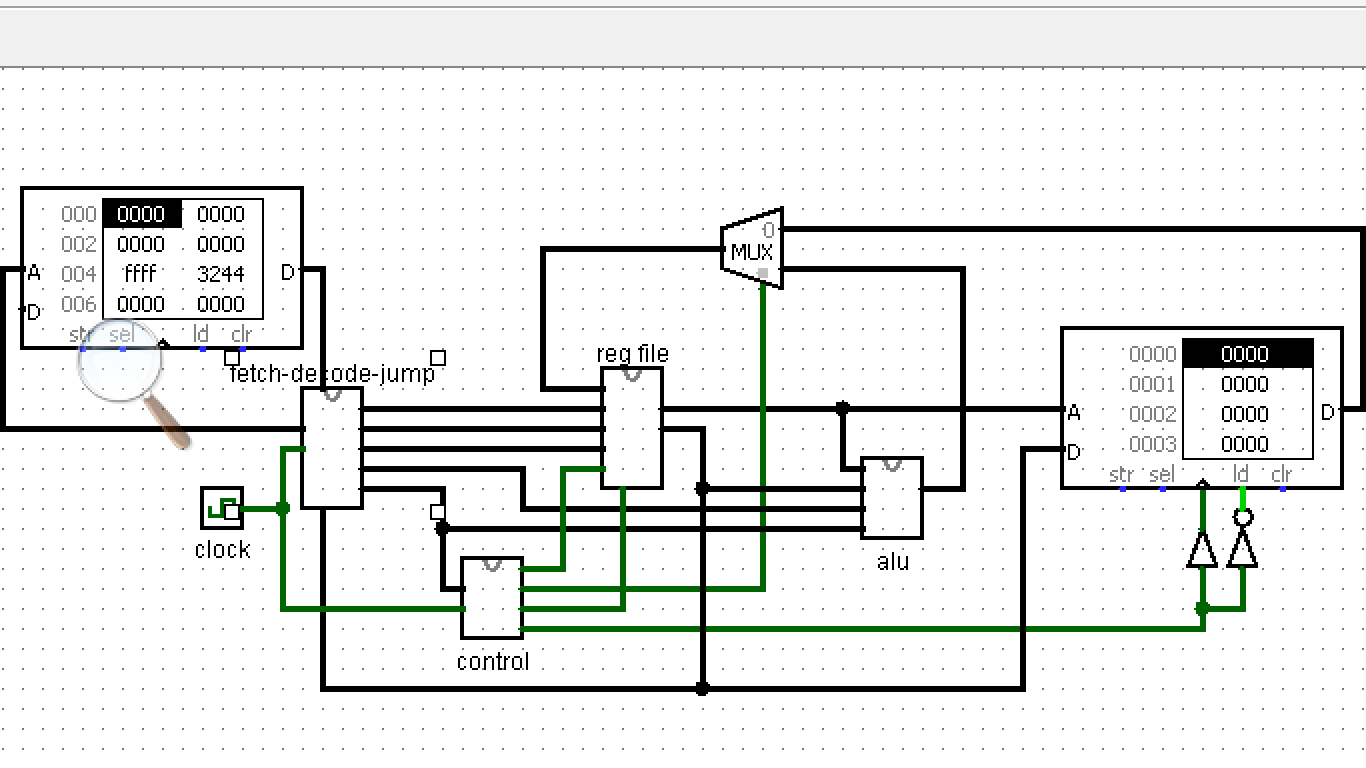
During this stage, both single cycle and two cycle instructions write their results into the register file.

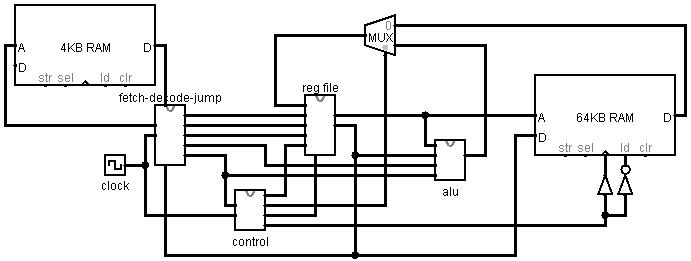
# IMPLEMENTATION IN LOGISIM

SCREENSHOTS FROM LOGISIM

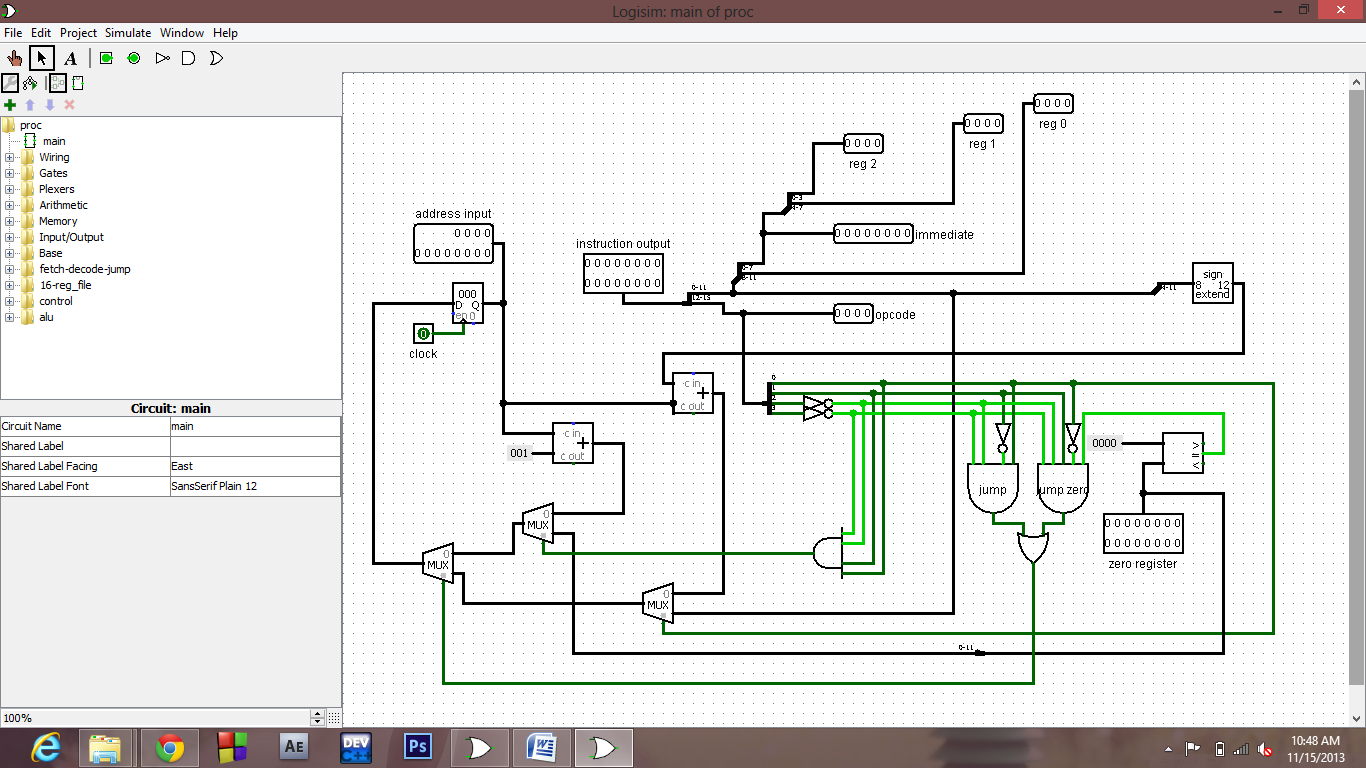
1. MAIN CIRCUIT



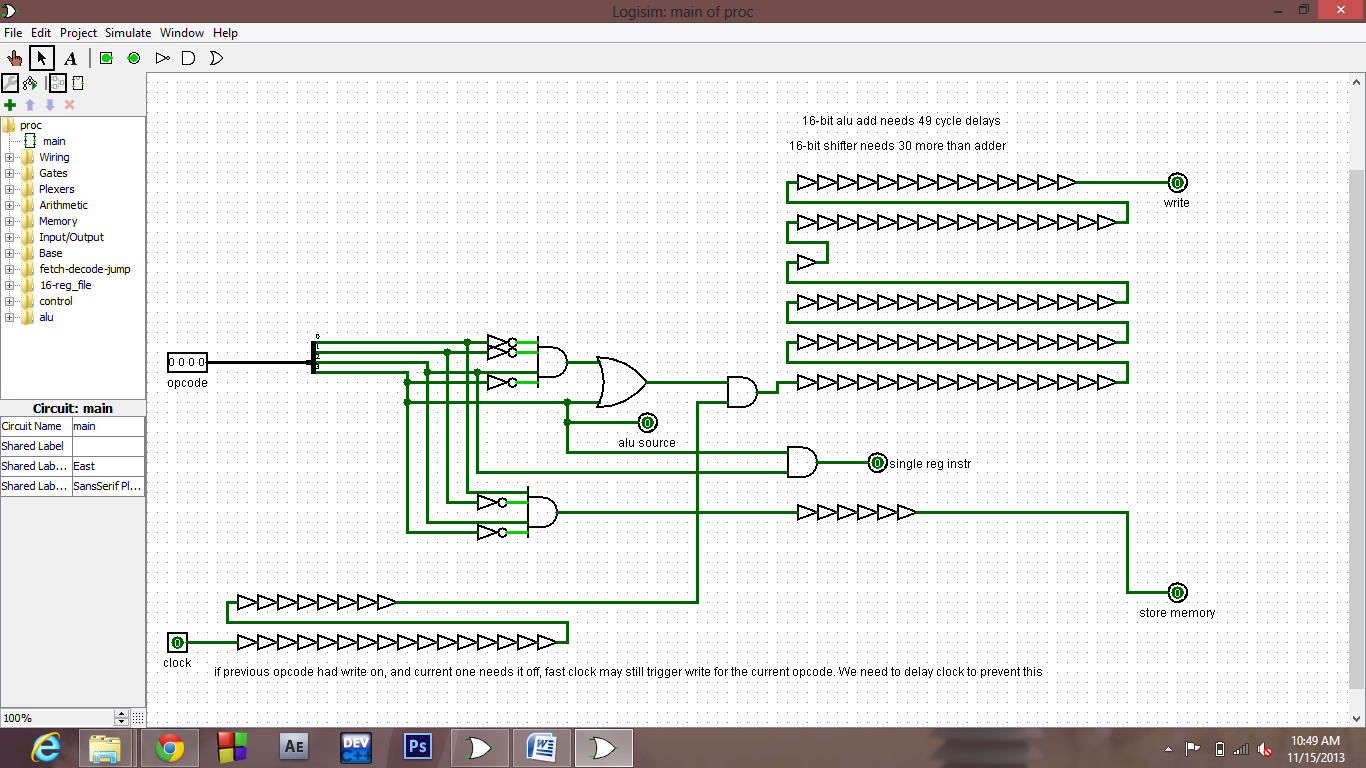




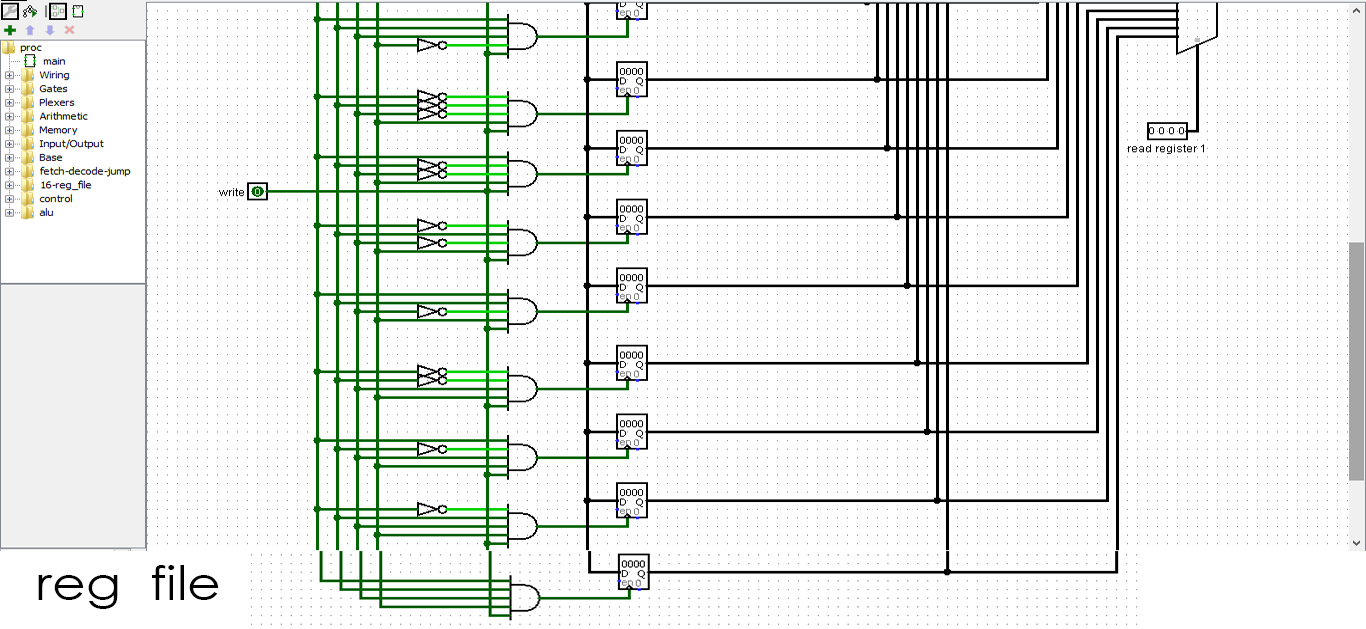
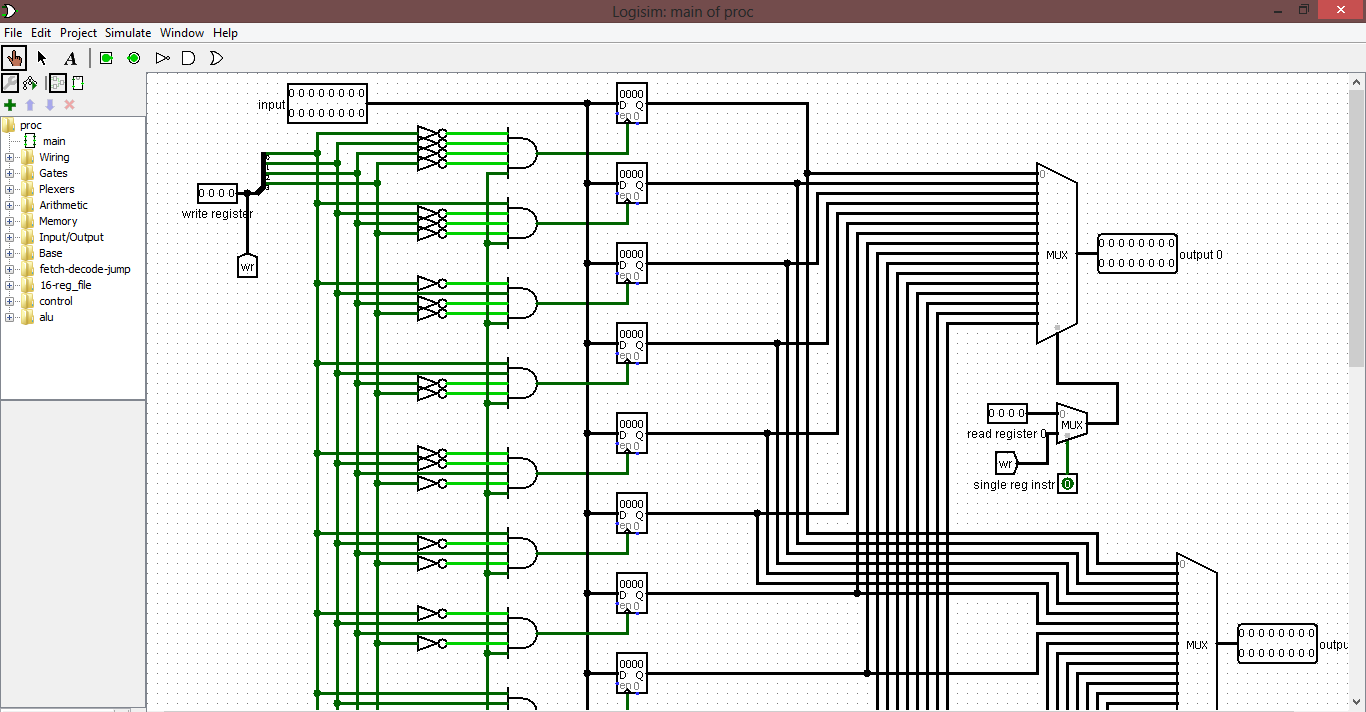
2. FETCH - DECODE - JUMP



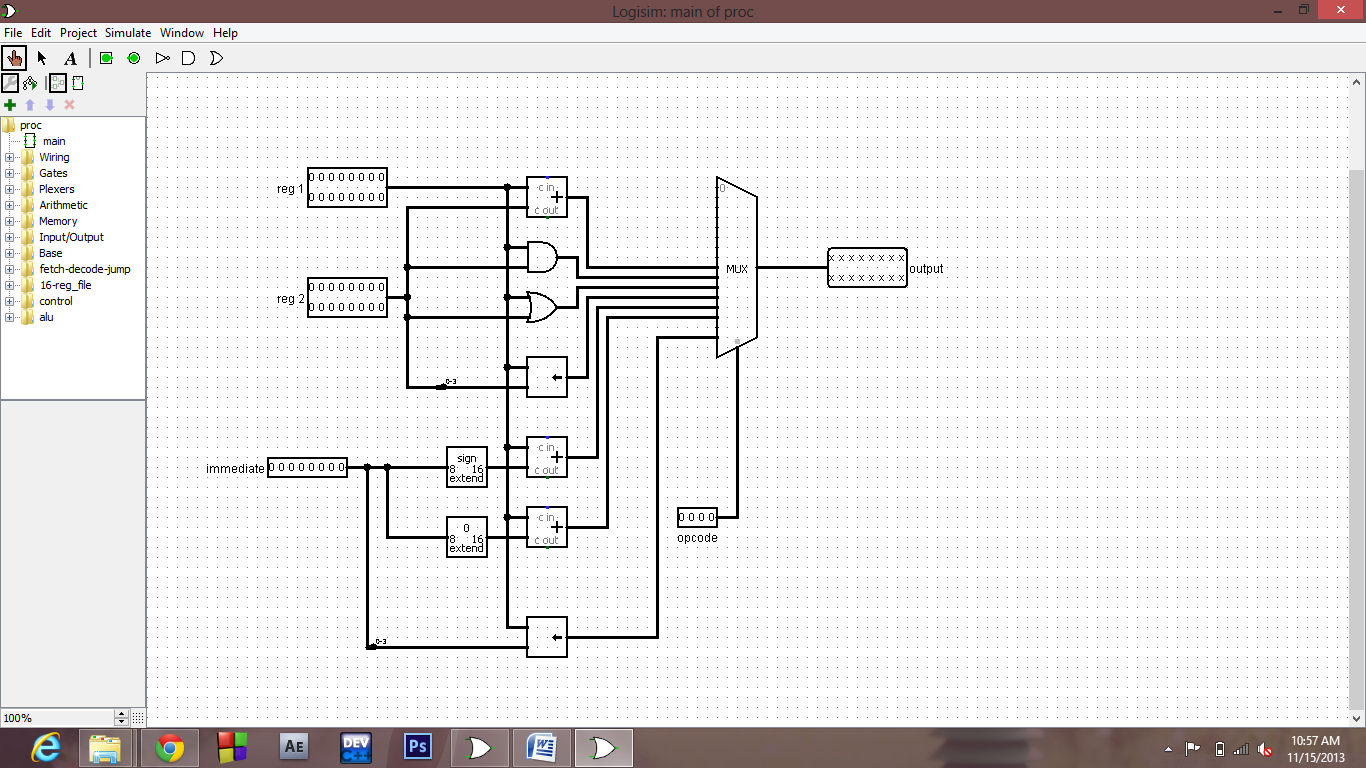
3. CONTROL



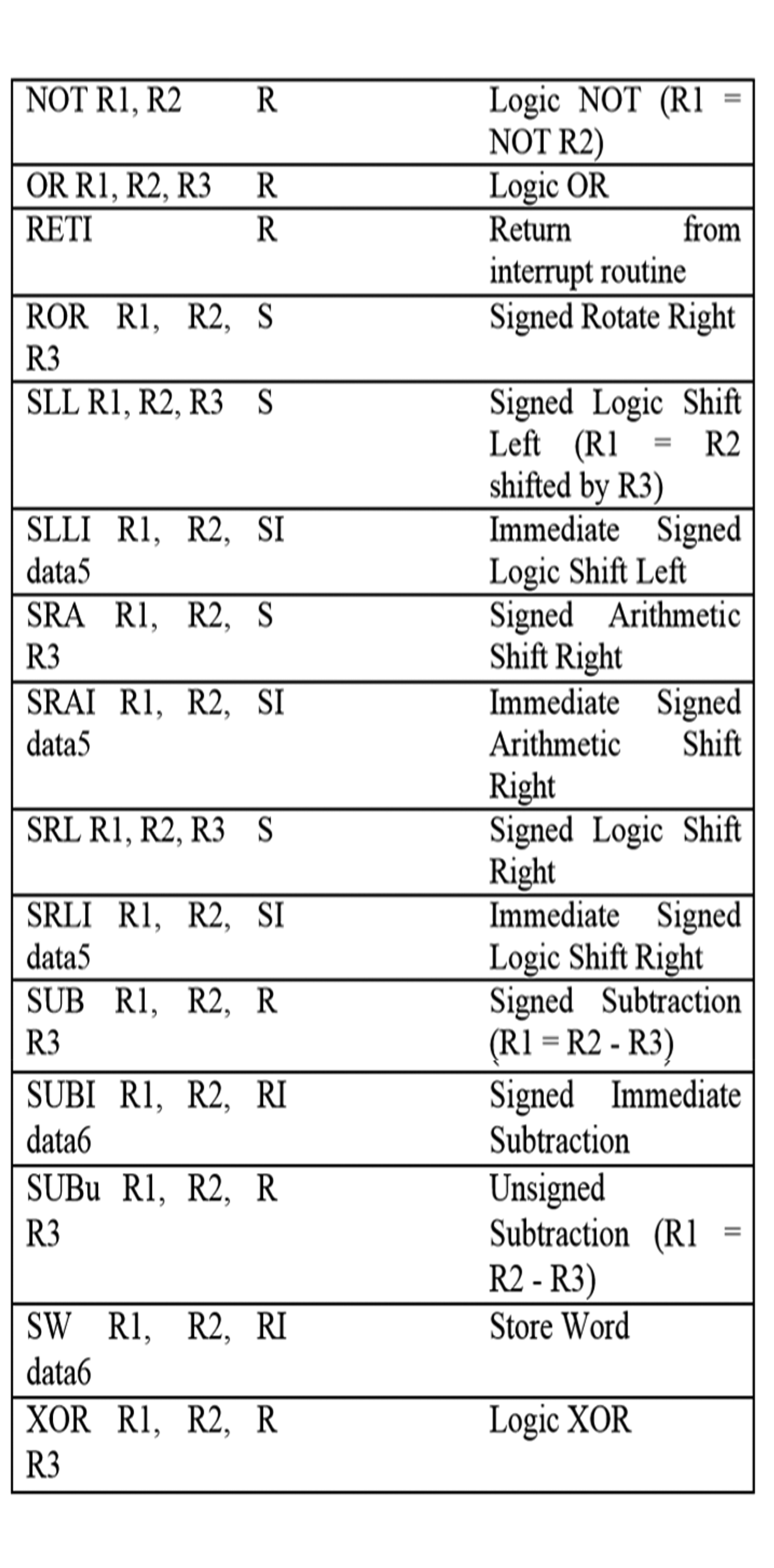
4. REGISTER FILE



5. ALU ( Arithmetic and logic unit )



ALGORITHM

The table below shows the actual instructions that have been implemented. A complex instruction can be implemented by a combination of two or more instructions in the instruction set: 

REFERENCE :-

1. Computer organization and architecture, J.P. Hayes.

2. Computer organization and architecture a quantitative approach, Patterson.

3. Wikipedia, Google

Links :-

1. http://en.wikipedia.org/wiki/Reduced\_instruction\_set\_computing

2. http://www-cs-faculty.stanford.edu/~eroberts/courses/soco/projects/risc/