DESIGN OF RISC MICROPROCESSOR ON SOC

ANAGIRE VISHAL VISHWANATH
[1]*Research Scholar, ECE Dept, St.Mary's Group of Institutions, Hyderabad, AP-India*
E-mail:- vanagire@gmail.com

ABSTRACT: *- In this thesis, we analyze MIPS instruction format, decoder module function, design theory based on RISC CPU instruction set, and instruction data path. Furthermore, we design instruction fetch (IF) module of 32-bit CPU based on RISC CPU instruction set. IF module mainly includes address arithmetic module check validity of instruction module synchronous control module, fetch instruction and latch module. Function of IF modules are implemented by pipeline and simulated successfully on ISE simulator.*

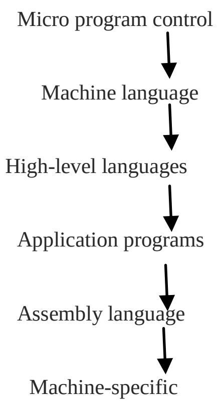
Keywords: *microprocessor, Ram, Rom, vhdl, fpga, Spartan 3e, Risc, Cisc*

1. INTRODUCTION TO RISC

Popular processor designs can be broadly divided into two categories: Complex Instruction Set Computers (CISC) and Reduced Instruction Set Computers (RISC). The dominant processors PC market, the Pentium, belong to the CISC. The recent trend is to use the RISC designs. Intel has moved from CISC to RISC design for their 64-bit processor. To explore RISC assembly language, we select the MIPS processor, which is pedagogically appealing as it closely adheres to the RISC principles. Further, the availability of the SPIM simulator allows us to use a PC to learn the MIPS assembly language.

1.1 Processor Architecture

At the lowest level, we have the machine language that is the native language of the machine. This language understood by the machine hardware. Because digital computers use 0 and 1 as their alphabet, machine language naturally uses 1s and 0s to encode the instructions. As shown in figure 1.1 one level up, there is the assembly language. Assembly language does not use 1s and 0s; instead, it uses mnemonics to express the instructions. Assembly language is related to the machine language.As programmers, we use the instruction set architecture (ISA) as a useful abstraction to understand the processor's internal details. ISA essentially describes the processor at a logical level. This abstraction suits us very well as we are interested in the details. This abstraction suits us very well as we are interested in the logical details of the RISC processor without getting bogged down by the myriad implementation details.



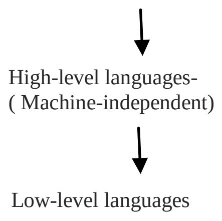


Fig.1.1: A programmer's view of a computer system

1.2 RISC Processor

The dominant architecture in the the Intel IA-32, PC market, belongs to the CISC design. The reason for this

classification is the "complex" nature of its Instruction Set Architecture (ISA). The motivation for designing such complex instruction sets is to provide an instruction set that closely supports the operations and data structures used by Higher-Level Languages (HLLs). However, the side effects of this design effort are far too serious to ignore.

The decision of CISC processor designers to provide a variety of addressing modes leads to variable-length instructions. Instruction length increases if an operand is in memory as opposed to a register. Because we have to specify the memory address as part of instruction encoding, this takes many more bits. This complicates instruction decoding and scheduling. The effect of providing a wide range of instruction types is that the number of clocks required to execute varies instructions widely. This leads to problems in instruction scheduling and pipelining.

Because these ISAs tend to produce instruction sets with far fewer instructions, they coined Reduced Instruction Set Computer (RISC). Even though the main goal was not to reduce the number of instructions, but the complexity, has stuck. We identify these RISC design principles after looking at why the designers took the route of CISC in the first place. Because CISC and RISC have their advantages and disadvantages, modern processors take features from both classes. Example, the PowerPC, which follows the RISC philosophy, has few complex instructions.

The following are the principles of RISC

- Simple Instructions
- Few Data Types
- Simple Addressing Modes
- Large Register Set
- Register-to-Register Operations
- Fixed-Length, Simple Instruction Format

2. ARCHITECTURE OF RISC MICROPROCESSOR

2.1 MIPS Processor

Full name of MIPS is microcomputer without interlocked pipeline stages. Informal full name is Millions of instructions per second. MIPS have already been pronoun of MIPS instruction set and MIPS instruction set architecture.

2.1.1 MIPS Instruction Set

ISA (Instruction Set Architecture) of processor is composed of instruction set and corresponding registers. Program based on same ISA can run on the same instruction set. It has been developed from 32-bit MIPSI to 64-bit MIPSIII and MIPSIV was created. To assure downward compatibility, every generation production of MIPS instruction directly extends new instruction based on old instruction but not abnegates any old instruction, so MIPS processor of 64-bit instruction set can execute 32-bit instruction. All MIPS instructions are all 32-bit specified instruction and instruction address is word justification. MIPS instructions are in three formats: immediate format (IFormat), register format(R-Format) and jump format (JFormat). Three instruction format shows as Figure. 2.1. Meaning of every instruction field as following: OP: 6-bit operation code; rs: 5-bit source register; rt: 5-bit temporary (source/destination) register number or branch condition; immediate: 16-bit immediate, branch instruction offset or address offset; destination: 26-bit destination address of unconditional jump; rd: 5-bit destination register number; shamt: 5-bit shift offset; funct: 6-bit function field;

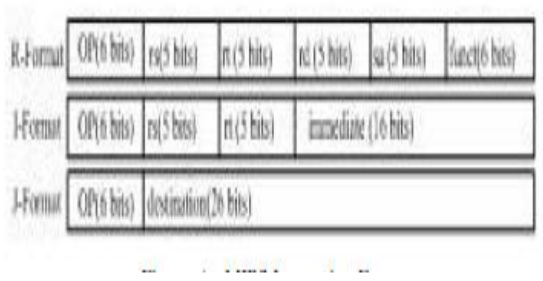


Fig.2.1: MIPS Instruction Format

MIPS instruction decoder or MIPS instruction execution is very high performance because of three type format with given length. Several simple MIPS instructions can accomplish complicated operation by complier.

2.2 Data Flow

Data flow is determined by hardware data path, which express data flow process. There is no clear difference between data and control. Operation code, operand, memory address and value, register address and value, jump destination address and content usually included in data, but control composes of control signal of unit, time sequence control signal and interrupt control signal, and these signals are not always defined clearly and strictly.

2.2.1 R-Format Data Path

In R-Format data path, fetch instruction from memory and analyze instruction into different parts. Two register specified by instruction fetch data from register file and ALU execute instruction command. Finally, after ALU outputs answer write the answer to register file. Figure. 2.2 shows RFormat data path. For example, ADD R1, R2, R3 instruction, which is add signed word instruction (R1= R2+ R3). Data flow of this instruction shows as following: PC fetches ADD R1, R2, and R3 instruction from memory. At first, the instruction access two registers R2 and R3 and value of the two register is put to ALU. After arithmetic is over, ALU write back result to R1 register. Then, data flow is in the end.

For another example, SRL R1, R2, R3 instruction, which is shift word right logical instruction. Data flow of this instruction shows as below: PC fetches SRL R1, R2, R3 instruction from memory. At first, the instruction access two register R2 and R3 and value of the two register is put to ALU.

2.2.2 RI-Format Data Path

RI-Format instruction is similar to R-Format instruction. The difference between them is
that the second read register of R-format instruction is replaced by immediate of RI-Format instruction.

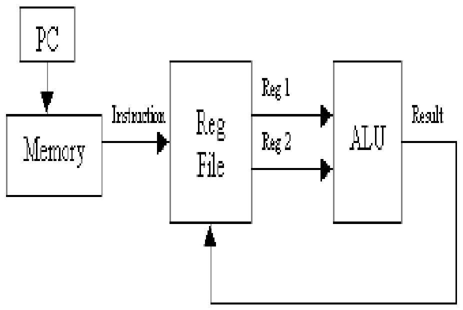


Fig.2.2: R-Format Instruction Data Path

The immediate is 32-bit signed number which is extend by 20-bit number, and put to ALU as the second operand. Finally, write-back result to register file. Figure 2.3 shows RI-Format data path.

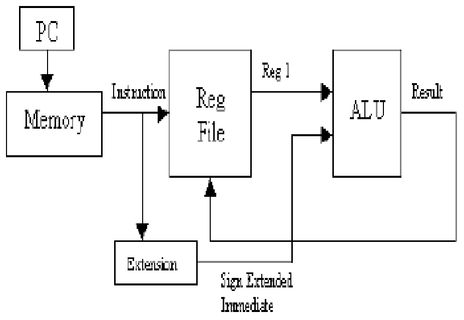


Fig. 2.3: RI-Format Instruction Data Path

Format includes ADDI R1, R2, data6 instruction，SUBI R1, R2, data6 instruction etc. When ADDI R1, R2, data6 instruction executes, PC fetches ADDI R1, R2 data6 instruction from memory and register R2 value is put to ALU. At the same time, immediate data6 is extended to 32-bit signed number and put to ALU Finally, after ALU completes add of the two operands, ALU writes back answer to R1 register. The difference data flow between SUB R1, R2 data6 instruction and ADD R1, R2, data6 instruction is that the former instruction does subtraction.

2.2.3 Load Word Data Path

Load word data path is similar to I-Format data path. The difference between the two data path is that result is written to memory in load word data but result is written to register in I-Format. In load word data path, fetch data from memory and load it to register file. Load word data path shows as Figure. 4. LW R1, R2, data6 instruction is the only one instruction in load word data path. It works shows as below: PC fetch LW R1, R2, data6 instruction from memory. R1 register is to load data. Firstly send R2 register value to ALU, at the same time, extend data6 immediate to 32-bit and send it to ALU. The answer of adding the two numbers is memory address, AND then, copy content of the memory address to R1 register.

2.2.4 Memory Word Data Path

It is similar to load word data path, but target which register is to write to memory but not to register file.

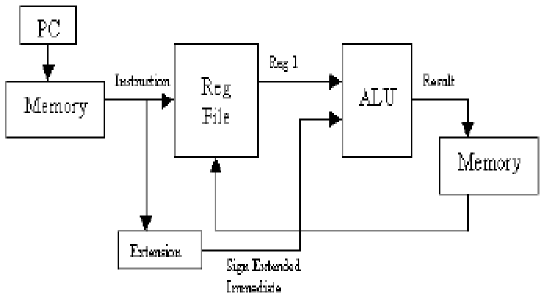


Fig. 2.4: Load Word Data Path

Only SW R1, R2, data6 instruction in load word instruction. PC fetches SW R1, R2 and data6 instruction from memory. R1 register stores data which is to be stored. Firstly, send R2 register value to ALU, at the same time, extend data6 immediate to 32-bit and send it to ALU. The result of adding the two numbers is memory address. Memory instruction data path shows as Figure. 2.5.

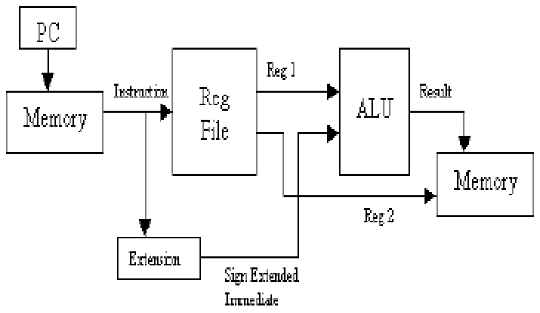


Fig 2.5: Memory instruction data path

2.2.5 Register Jump Data Path

In register jump data path, one register compares to 0. When jump instruction is jump if zero instruction and register value is 0, the second register loads to program counter. When jump instruction is jump if zero instruction and value in register is not 0, the next program counter value is loaded and instruction execution continues. Jump if not zero instruction is similar. Figure.2.6 shows jump instruction data path.

Register jump instruction includes two instructions: BZ R1, R2 instruction and BNZ R1, R2 instruction. BZ R1, R2 instruction expresses if it

is equal to constant 0 jump. Program counter fetches BZ R1, R2 instruction from memory, and instruction accesses R1 register and R2 register. And then, send value of the two registers to branch unit. Branch unit judges whether R1 value is equal to 0.

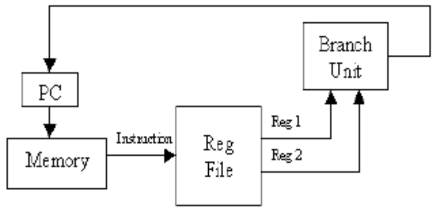


Fig.2.6: Jump Instruction Data Path

If R1 value is equal 0, send value of register R2 to program counter. If R1 value is not equal 0, PC adds 1 and program continues executing orderly. BNZ R1, R2 instruction expresses if it is not equal to constant 0 then jump. Program counter fetches instructions: BNZ R1, R2 from memory, and accesses R1 register and R2 register. And then, send value of the two registers to branch unit. Branch unit judges whether R1 value is 0. If R1 value is not 0, send value of register R2 to program counter. If R1 value is equal 0, PC adds 1 and program continues executing in sequence.

2.3 Pipeline Design

Pipeline decomposition enhances throughput rate of instruction. Clock cycle is decided by the slowest stage running time. Pipeline includes five stages: instruction fetch(IF) instruction decoder(ID), execution(EXE), memory/ IO(MEM)、write-back(WB).

2.3.1 Instruction Fetch (IF)

Instruction fetch (IF) stage is request for instruction which is fetched from memory, IF stage is shows in Figure. 7. Instruction and PC is memorized in IF/ID pipeline register as temporary memory for next clock cycle.

2.3.2 Execution (EXE)

EXE stage executes arithmetic. Main component is ALU. Arithmetic logic unit and shift- register compose of ALU. EXE stage structure is shown in figure 2.7.

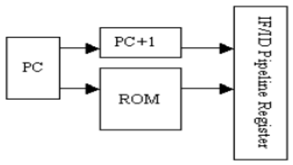


Fig.2.7 IF Stage

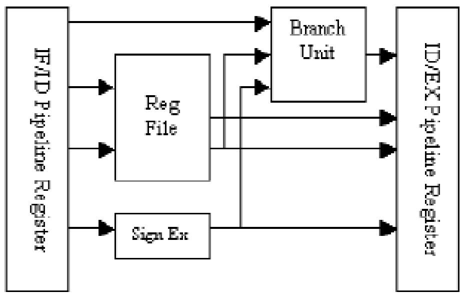


Fig.2.8: ID Stage

Function of EXE stage is to do operation of instruction, such as add and subtraction. ALU sends result to EX/MEM pipeline register before entering MEM stage

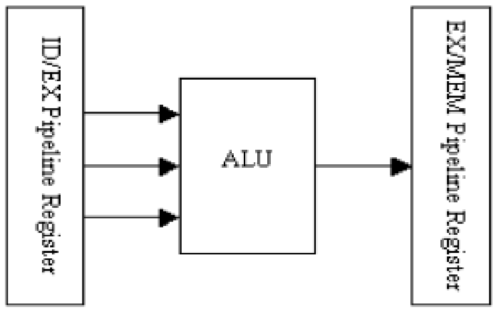


Fig.2.9: EXE Stage

2.3.3 Memory and IO (MEM)

Function of MEM stage is to fetch data from memory and store data to memory. Another function is to input data to processor and output data. If instruction is not memory instruction or IO instruction, result is sent to WB stage.MEM stage structure shows as Figure. 2.10.

Storing data in register is main function after result is calculated. Some result may be not stored in RAM definitely, and some result can be written to register directly. Give an example; some temporary variable is not memorized in RAM because of low execution efficiency. However, some data must be stored in RAM. Data in RAM or register depending on demands in MEM stage.

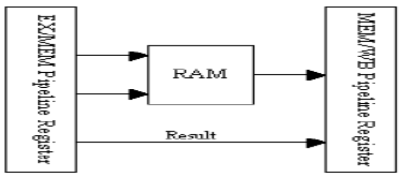


Fig 2.10: MEM Stage

There is a data copy in MEM/WB pipeline register.

2.3.4 Write-Back (WB)

WB stage functionality is input data to register file, store data, and writing result. The functionality of WB stage is to write data to destination register. For instance, ADD R1, R2, R3 instruction, result is stored in R1 registers to make program run faster.

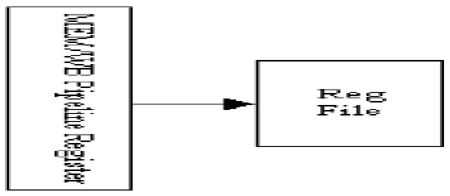


Fig.2.11: WB Stage

3. ARCHITECTURE OF RISC MICROPROCESSOR

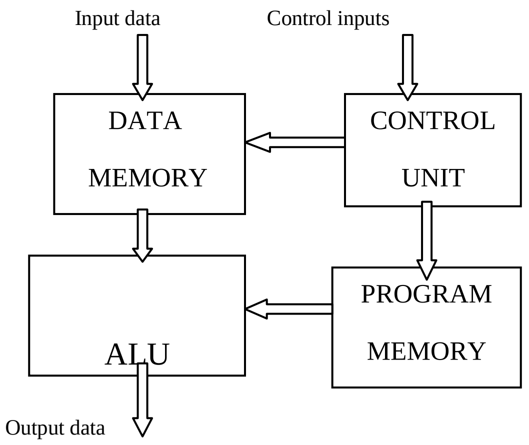


Fig 3.1: Block diagram of RISC microprocessor

The fig 3.1 shows 32 bit RISC microprocessor. It contain four blocks which are Data memory block, program memory block, Control unit and Arithmetic unit.

3.1 Data memory

Data memory is a region of memory where arrays, the temporary variables, and information used by a program can be stored without using the hard disk or long term memory. Memory allocations come from when more memory for data structures is needed in the course of the program. The data memory shall have an address width of four bits and capable of storing up to one 4 byte (eight bits) of data at each address – In total 4X16 bytes of data can be stored.

3.2 Program Memory

Program (CODE) memory is read only; it cannot be written to. The code program is a sequence of instructions values in memory. It consists generally
1. Processor registers (for direct and fast access).
2. The stack.
3. A data segment (static allocation region).
4. The heap (dynamic allocation region).

Only the stack and the dynamic allocation region can change in size during the execution of a program. Depending on programming language, some control over these classes of memory can be exercised. Whereas the program instructions (code) usually reside in static memory, dynamic linking makes use of dynamic memory. Program code, including all functions and library routines, are stored in memory. Constant variables are stored in programming memory.

3.2.1 Control Unit

The control unit maintains order within the computer system and directs data and the flow of operations. The control unit selects each program statement every time from storage area, interprets the statement, and sends the appropriate electronic impulses to the arithmetic-logic unit and storage section to cause them to carry out the instruction.

The control unit selects one program statement at a time from the program storage area, interprets the statement, and sends the appropriate electronic impulses to the arithmetic and logic unit and storage section to carry out the instruction. It controls the flow of all data entering and leaving the computer which is by communicating or interfacing with the arithmetic and logic unit, memory unit, and I/O areas. It provides the computer with the ability to function under program control.

3.2.2 Arithmetic Logic Unit

The first entity described is the ALU. This entity performs a number of arithmetic or logical operations on one or more input busses. A symbol for the ALU is shown in Fig.3.2 The arithmetic-logic unit (ALU) performs logic operations and all arithmetic operations (addition, subtraction, multiplication, division). Logic operations test various conditions encountered during processing and allow for different actions to be taken based on the results. The data to perform the arithmetic and logical functions are input from the designated CPU registers and operands.

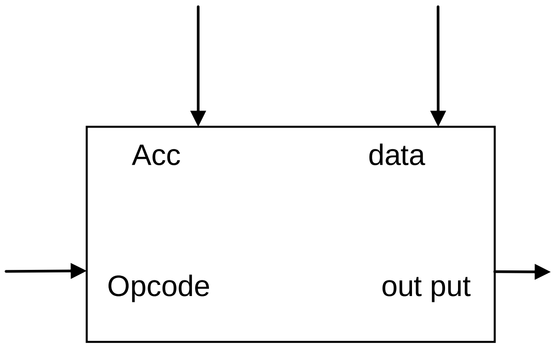


Fig.3.2: ALU

ALU relies on basic items to perform its operations. It include number systems, operands, instructions, data routing circuits (adders/subtracters) timing and registers. The Logic Unit is a sequential unit with a single Accumulator, in which to store intermediate data.

The Logic Unit has an single 8-bit data input port, and a single 8-bit data output port. The Logic Unit should have four function pins (S0 – S3) to select the operation as defined below. The Logic Unit should flag for accumulator zero, and for overflow.

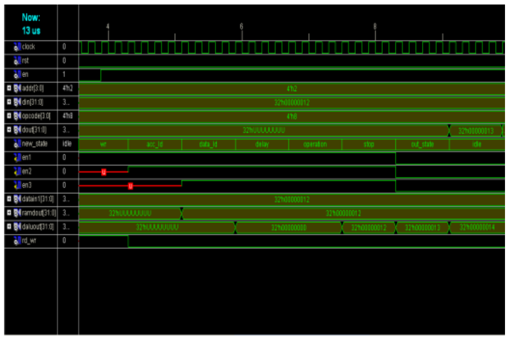
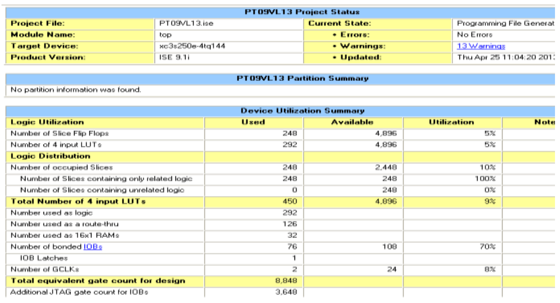


Fig 3.3: simulation results of top level design



PT09VL13 Project Status			
Project File:	PT09VL13.ise	**Current State:**	Programming File Generat
Module Name:	top	**• Errors:**	No Errors
Target Device:	xc3s250e-4tq144	**• Warnings:**	13 Warnings
Product Version:	ISE 9.1i	**• Updated:**	Thu Apr 25 11:04:20 201

PT09VL13 Partition Summary
No partition information was found.

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note**
Number of Slice Flip Flops	248	4,896	5%	
Number of 4 input LUTs	292	4,896	5%	
Logic Distribution				
Number of occupied Slices	248	2,448	10%	
Number of Slices containing only related logic	248	248	100%	
Number of Slices containing unrelated logic	0	248	0%	
Total Number of 4 input LUTs	450	4,896	9%	
Number used as logic	292			
Number used as a route-thru	126			
Number used as 16x1 RAMs	32			
Number of bonded IOBs	76	108	70%	
IOB Latches	1			
Number of GCLKs	2	24	8%	
Total equivalent gate count for design	8,848			
Additional JTAG gate count for IOBs	3,648			

Fig 3.4: Design summary report of top level design

4. CONCLUSION

In this research, we adopt top-down design method and use VHDL to describe system. At first, we design the system from top and in-depth design gradually. The hierarchical structure of design is very clear. It is easy to edit and debug. Design of instruction fetch (IF) stage simulates integrate and routes on Xilinx. The result indicates IF stage completes prospective function.

5. REFERENCES

[1]32-bit RISC CPU Based on MIPS Instruction Fetch Module Design , Kui YI Department of Computer Science and Information Engineer, WuHan Polytechnic University Wuhan, HuBei Province 430023, China,, of Computer Science and Information Engineer, WuHan Polytechnic University Wuhan, HuBei Province 430023, China. IEEE 2009 International Joint Conference on Artificial Intelligence.

[2] Pan-Song, Huang-JiYe, SOPC Technology Utility Tutorial , Tsinghua University Press,2006.

[3] Wang-YuanZhen, IBM-PC Macro Asm Program, Huazhong University of Science and Technology Press, 1996.9.

[4] MIPS Technologies, Inc. MIPS32™ Architecture For Programmers Volume II: The MIPS32™ Instruction Set，June 9, 2003.

[5] Zheng-WeiMin, Tang-ZhiZhong. Computer System Structure (The second edition), Tsinghua University Press,2006.

[6] Wang-AiYing, Organization and Structure of Computer, Tsinghua University Press, 2006.

[7] Mo-JianKun, Gao-JianSheng, Computer Organization, Huazhong University of Science and Technology Press, 1996.

[8] MIPS32 4KTMProcessor Core Family Software User's Manual , MIPS Technologies Inc.

[9] Zhang-XiuJuan, Chen-XinHua, EDA Design and emulation Practice [M]. BeiJing, Engine Industry Press. 2003.

[10] "IEEE Standard of Binary Floating-Point Arithmetic" IEEE Standard754, IEEE Computer Society, 1985