Design and Analysis of 32-bit RISC Processor Based on MIPS

Rama Krishna V, Venu Gopal B

***Abstract*— In this paper, we have studied Microcomputer with out interlocked pipeline stages instruction format instruction data path decoder module function and design theory basend on RISC CPUT instruction set. We have also designed instruction fetch(IF) module of 32-bit CPU based on RISC CPU instruction set. Function of IF module mainly includes fetch instruction and latch module address arithmetic module check validity of instruction module synchronous control module. Function of IF modules are implemented by pipeline and simulated successfully on Xilinx Spartan 3E fpga device Xc3s200..**

Keywords- MIPS, Data Flow, Data Path, Pipeline

I. INTRODUCTION

Because memory was expensive in old days, designer of instruction enhanced complication of instruction to reduce program length. Tendency of complication instruction design brought up one traditional instruction design style, which is named “Complex Instruction Set Computer-CISC” structure. But great disparity among instructions and low universal property result in instruction realization difficulty and long running-time cost. Comparing to CISC, RISC CPU have more advantages, such as faster speed simplified structure easier implementation. RISC CPU is extensive use in embedded system. Developing CPU with RISC structure is necessary choice.

II. INSTRUCTION SET OF MIPS

A. MIPS Processor.

Microcomputer without interlocked pipeline stages was abbreviated as MIPS. It was also informally called as Millions of instructions per second. MIPS was already been pronoun of MIPS instruction set and MIPS instruction set architecture

B. 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. MIPS instruction has been developed from 32-bit MIPSI to 64-bit MIPSIII and MIPSIV since it 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 4-bit instruction set can execute 32-bit instruction. All MIPS instructions are all 32-bit specified nstruction and instruction address is word justification. MIPS divides instructions into three formats: immediate format(I-Format) register format(R-Format) and jump format(J-Format)[2]. Three instruction format shows as Figure.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 conditional jump;
- rd: 5-bit destination register number;
- shamt: 5-bit shift offset;
- funct: 6-bit function field;

R-type Format:

Opcode	rs	rt	rd	Shift amt	Function
6-bits	5-bits	5-bits	5-bits	5-bits	6-bits

I-type Format:

Opcode	rs	rt	Address
6-bits	5-bits	5-bits	16-bits

J-type Format:

Opcode	Destination
5-bits	26-bits

Figure 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 [3].

III. 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 are 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.

A. 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 shows R-Format data path.

Manuscript received on October, 2012.

Rama Krishna, ECE Department,JNTUK University/ Kaushik College of Engineering/ Visakhapatnam, India.

B.Venu Gopal, Assoc .Professor, Dept. of ECE, Kaushik College of Engineering /visakhapatnam, India.

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,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. And then, data flow is in the end.

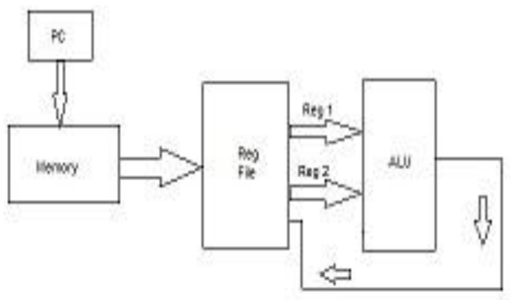

Figure. 2 R-Format Instruction Data Path

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. After arithmetic is over, ALU write back answer to R1 register, And then, data flow is in the end.

B. RI-Format Data Path

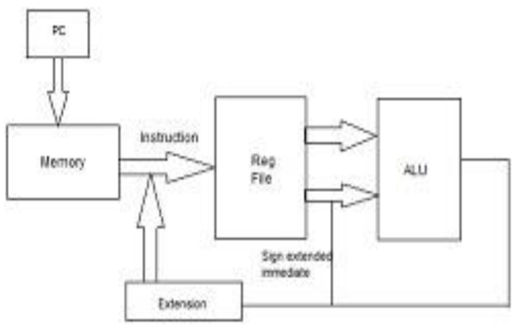

Figure. 3 RI-Format Instruction Data Path

RI-Format instruction is similar to R-Format instruction[4]. The difference between them is that the second read register of R-format instruction is replaced by immediate of RI-Format instruction. 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. RI-Format data path shows as Figure. 3.

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 do subtraction.

C.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.

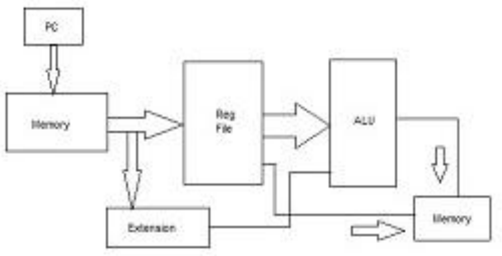

Figure. 4 Load Word Data Path

D. Memory Word Data Path

Memory word data path is similar to load word data path, but target which register is to write is memory but not register file. There is only SW R1, R2, data6 instruction in load word instruction. PC fetches SW R1, R2, 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. 5.

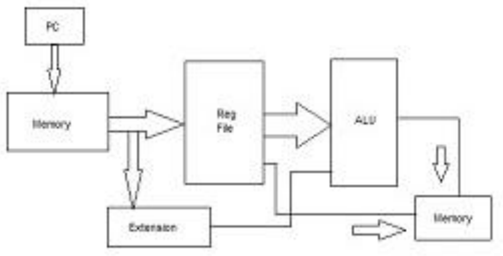

Figure. 5 Memory Instruction Data Path

E. 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 value 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. 6 shows jump instruction data path.

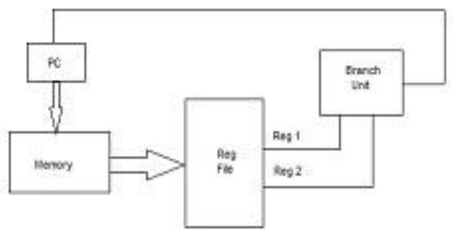

Figure. 6 Jump Instruction Data Path

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. 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 BNZ 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. If R1 value is not equal 0, send value of register R2 to program counter. If R1 value is equal 0, PC adds 1 and

Program continues executing in sequence.

IV. PIPELINE DESIGN

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

A. Instruction Fetch (IF) Instruction fetch (IF)

stage is request for instruction which is fetched from memory. Main component of IF stage shows as Figure. 7. Instruction and PC is memorized in IF/ID pipeline register as temporary memory for next clock cycle. Helpful Hints

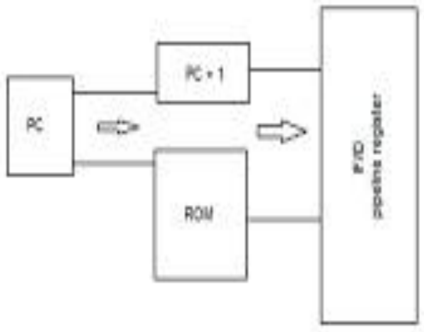

Figure. 7 IF Stage

IF stage mainly depends on program counter(PC) current value. CPU fetches instruction from ROM based on PC value and PC adds 1 automatically. Finally, send all these information to IF/ID pipeline register to decoder.

B. Instruction Decoder(ID)

ID stage sends control command to other units of processor based on decode of instruction. Figure. 8 shows ID stage structure. Instruction is sent to control unit and decoded here. Read register fetches data from register file. Branch unit is also included in ID stage. Input of ID stage is from IF stage. ID stage decodes instruction to control signals and prepared operand. For example, if instruction is I-Format instruction, extend immediate to 32-bit data and access register file. If instruction is J-Format instruction, EXE stage comes after branch unit process completes.

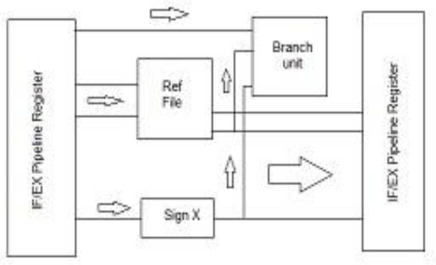

Figure. 8 ID Stage

C. Execution (EXE)

EXE stage executes arithmetic. Main component of EXE stage is ALU. Arithmetic logic unit and shift-register compose of ALU. Figure. 9 shows EXE stage structure. 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.

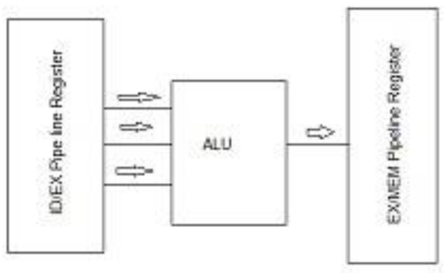

Figure. 9 EXE Stage

D. 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. 10.

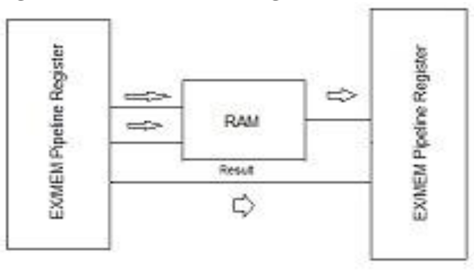

Figure. 10 MEM Stage

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.

Memory data in RAM or register depending on demands in MEM stage. There is a data copy in MEM/WB pipeline register.

E. Write-Back (WB)

WB stage charges of writing result、store data and input data to register file. The purpose of WB stage is to write data to destination register. For example, ADD R1, R2, R3 instruction memories result in R1 register to make program run faster. Figure. 11 shows WB unit instruction.

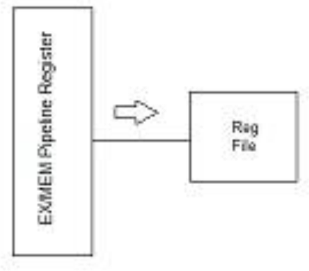

Figure. 11 WB Stage

V. INSTRUCTION FETCH STAGE DESIGN

A. Function Statement.

Function of instruction fetch(IF) stage shows as below:

1) Fetch instruction and latch. Fetch instruction from instruction register depending on PC value and send the instruction to IF/ID pipeline register to latch.
2) Address arithmetic. Based on value of sel[3..0] in pcselector, select next value of PC from four address jump sources. These address jump sources are incPC branchPC retiPC and retPC .
   - If instruction in WB stage of pipeline is jump instruction or successful branch instruction, select branchPC value and destination address of program jump acts as address arithmetic result;
   - If instruction is not jump instruction or fail branch instruction, PC adds 1 automatically and points to next instruction in instruction register;
   - If instruction is interrupt-return instruction, select retiPC value;
   - If instruction is subprogram return instruction, select retPC value.
3) Check validity of instruction. Check operation code and function code validity based on definition of instruction set. If instruction is wrong, an exception is thrown.
4) Synchronous control. Use CLK to control synchronous of external sign..

B. Module and Implementation

IF stage includes five modules: incPC、lpm_rom0 progc pcselector and ifid. Figure. 12 shows connection of each module.

Their function shows as below:

- incPC: PC adds 1 automatically. PC points to address of next instruction;
- lpm_rom0: application store program;
- progc: program counter;
- pcselector: control next instruction selection;
- ifid: pipeline latch.

Every module uses VHDL to describe. Input signal of IF stage includes branchPC , retPC, retiPC, sel clk ifid_flush, ifid_enable and pc_enable. Their function shows as below:

- branchPC: jump address of branch signal
- retPC: subprogram return address signal
- retiPC: interrupt return address signal
- sel : selection signal from pcselector in EXE stage
- clk: clock signal
- ifid_flush: data signal
- ifid_enable、pc_enable: control signal
- Output signal of IF stage includes ins[31..0]
- pcvelue[31..0],insOut[31..0]and pcout[31..0]. Their function shows as below:
- ins[31..0]: instruction code fetch from instruction register;
- pcvelue[31..0]: PC value in IF stage;
- insOut[31..0]: instruction code which is to sent to next stage and comes from pipeline register ifid;
- pcout[31..0]: program counter value.

Module Implementation shows as below:

1) pcselector module. Input port includes nextpc[31..0] branchpc[31..0] 、 retpc[31..0] 、 retipc[31..0] and sel[3..0]. Output port includes newpc[31..0]. Select data from four source data as next instruction address determined by sel[3..0]. The four source data are nextpc[31..0] 、 branchpc[31..0] 、 retpc[31..0] and retipc[31..0].

Input signal are nextPC, branchPC, retPC, retiPC and sel. Output signal are newPC. Function of input signal shows as below:

nextPC: next instruction address;
branchPC: address of branch jump signal;
retPC: subprogram return address signal;
retiPC: interrupt return address signal;
sel: selector signal.

Time sequence simulation waveform of pcselector Input different address sign into nextpc、branchpc、retpc retipc ports, and newpc selects one of the four input signal to output depending on value in sel[3..0].

Figure. 13 pcselector Stage Simulation Waveform

2) progc module. Input port includes pcin[31..0]、clk and enable. Output port includes pcout[31..0]. The function of the module is to communicate with instruction memory. When positive clock edge comes, send value of address bus pcin[31..0] to instruction memory and fetch next instruction from ins[31..0]. ins[31..0] is output of instruction memory. Send instruction out when negative clock edge comes.
3) incPC module. Input port includes pcin[31..0] and output port includes pcout[31..0]. The function of incPC module is to PC add 1 and the new PC cat as one optional value.

When negative clock sign comes, PC value is sent to pcselector module. Figure. 15 shows incPC module entity structure and RTL structure. Figure. 16 shows simulation waveform of incPC module. We can know pcIn value adds 1 and send result to pcVal from waveform.

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	1118	4656	24%
Number of Slice Flip Flops	514	9312	5%
Number of 4 input LUTs	1951	9312	20%
Number of bonded IOBs	256	190	134%
Number of GCLKs	1	24	4%

Figure. 15 Device Utilization Summary

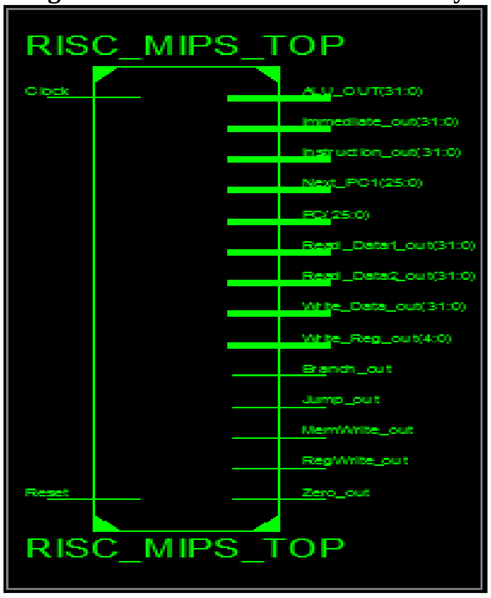

Figure. 16 incPC Module Simulation Waveform

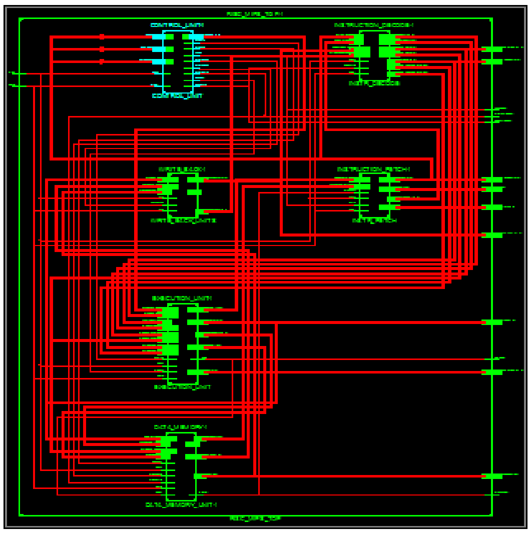

Figure.17 RTL Schematic

4) Lpm_rom0 module. Input port includes address[5..0] and inclock. Output port includes q[31..0]. Function of the module is to memory program machine code. Access memory location which is specified by address bus address[5..0], moreover, fetch next instruction from memory and send out the instruction by instruction bus q[31..0].

lpm_rom0 module can be implemented EAB of FPGA by calling macro function module. Adopt lpm_rom structure in macro function library to realize the module. Parameter configuration is that address bus address is 6-bit and output bus q is 32-bit. Process of lpm_rom0 is described as following: when positive inclock edge comes, latch address[5..0] and ouput the data pointed by value of address[5..0] to output port q[31..0]. Set up data in lpm_rom0 by memory initialization file (.mif), or edit、update and reload data on debugging by system memory editor tool.

5) ifid module. Input port includes pcin[31..0] 、insin[31..0]、clkid_flush and ifid_enable. Output port includes pcout[31..0] and insout[31..0]. Function of ifid is to latch PC and instr of Statge1 and send them to next stage. Time sequence simulation waveform of Ifid module shows as Figure. 17. We can see fact that when ifid_enalbe is high level and id_flush is low level, data are not relative in pipeline. When positive edge of clk comes, values of insOutand pcOut are same to insIn and pcIn respectively; When ifid_enable and id_flush are all high level, data is relative in pipeline. When positive edge of clk comes, insOut changes to “0000H”, but pcOut maintains its original value ; After pipeline conflicts, insOut and pcOut returns to normal working state; if ifid_enable is low level, pipeline stops working and insOut and pcOut maintain its original state.

VI. CONCLUSION

In this research, we adopt top-down design method and use VHDL to describe system. At first, we design the system from the top, and do in-depth design gradually. The structure and hierarchical of design is very clear. It is easy to edit and debug. Design of instruction fetch (IF) stage simulates integrate and routes on Spartan 3E fpga. The result indicates IF stage completes prospective function.

REFERENCES

1. Bai-ZhongYing, Computer Organization, Science Press, 2000.11.
2. Wang-AiYing, Organization and Structure of Computer, 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. Pan-Song, Huang-JiYe, SOPC Technology Utility Tutorial , Tsinghua University Press,2006.
7. MIPS32 4KTMProcessor Core Family Software User's Manual , MIPS Technologies Inc.
8. Mo-JianKun, Gao-JianSheng,Computer Organization, Huazhong University of Science and Technology Press, 1996.
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.
11. Yi-Kui, Ding-YueHua, Application of AMCCS5933 Controller in PCI BUS, DCABES2007, 2007.7.