Instruction Decoder Module Design of 32-bit RISC CPU Based on MIPS

XiangYunZhu, Ding YueHua
Department of Computer Science and Information Engineer, WuHan Polytechnic University
Wuhan, HuBei Province 430023,China
Email:xyz@whpu.edu.cn

Abstract

This paper introduces architecture and feature of 32-bit micro-processor, and describes internal data path in processor. Through analysis of function and theory of RISC CPU instruction decoder module, we design instruction decoder(ID) module of 32-bit CPU by pipeline theory. The instruction decoder includes register file、write back data to register file、sign bit extend、relativity check, and it is simulated on QuartusII successfully. Static time sequence shows the instruction decode module completing required unction.

1. Introduction

With change of micro-electric, FPGA/CPLD has been replaced of ASCI more and more. FPGA/CPLD has so many advantages, such as repeatable use、low cost、high performance、high density、short development cycle, that more and more IC designer are willing to adopt it. Designer can describe self-designed function module by some method, such as HDL, and verify correctness of design by way of function simulation, such as HDL simulation[1]. Open-source processor is based on design thinking of FPGA/CPLD. Designer can design prospective function on platform and verify function. Finally, download program to FPGA/CPLD unit. Cheap and flexible ID development is achieved.

In our project, we adopt MIPS32 instruction set and implement 32-bit RISC micro-processor. The micro-processor includes five parts: instruction fetch(IF) stage、instruction decode(ID) stage、execution stage (EXE)、memory access(MEM) stage and write-back(WB) stage[2]. Because of trim instruction and easy decoder, RISC instruction set is popular adopted. We analyses MIPS32 instruction set and complete instruction decoder stage design.

2. Function and control of processor

Different stage detached from pipeline can enhance instruction throughput rate. Clock cycle time is determined by the longest stage working time[3]. Instruction decoder(ID) stage send control signals to other stage by decoder of instruction. Figure 1 shows ID stage structure. Instruction is put to control unit and decoded. Read register fetches data from register and branch jump is included in ID stage.

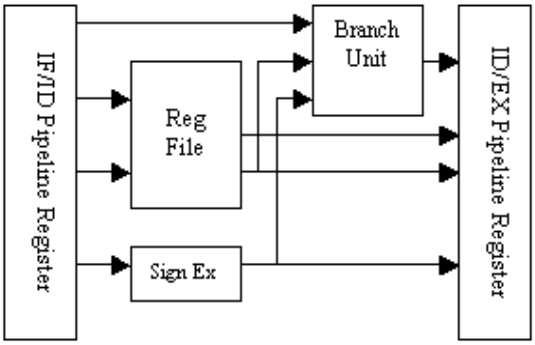

Figure 1 . Instruction decoder(*ID*) component

3. Instruction Set Design

Before start design, the first thing that is to do processor is that decide what kinds instruction is to adopt. For universality of processor design, we choose the most common instruction[4].

3.1 R-Format instruction

The most common instruction is R-Format instruction. R-Format instruction have two read registers and one write register[5]. Figure2 shows typical R-Format ADD R1,R2,R3 instruction. R-Format instruction has 6-bit operation code(opcode), which decide what kind of instruction is to be executed, such as R-Format, J-Format. All registers in R-Type are all 5-bit, so it means that register file includes 32 registers. In R-Format instruction, the first 5-bit register is write register, and the second and the third 5-bitregister are two read registers. The last 6-bit in R-Format instruction is used as function bit. Function bit(func) determines what kind of operation is to be executed, such as add、subtraction, so R-Format includes 32 kinds of instruction at most.

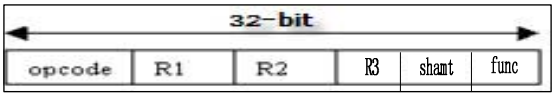

Figure 2. R-Format *ADD R1,R2,R3* instruction

978-0-7695-3334-6/08 $25.00 © 2008 IEEE
DOI 10.1109/WGEC.2008.127

3.2 RI-Format instruction

RI-Format instruction is similar to R-Format instruction. The second read register and function bit in R-Format instruction is replaced by 16-bit immediate in RI-Format. Typical RI-Format shows as Figure 3. Each RI-Format operation code corresponds to one instruction because RI-Format instruction has no function bit.

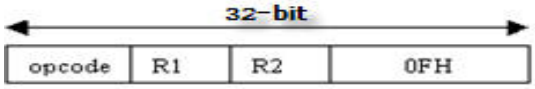

Figure 3. RI-Format *ADDI R1,R2,0FH* instruction

3.3 I-Format instruction

I-Format instruction includes two shift immediate instructions and two PC relative jump instructions. I-Format is composed of 6-bit operation code、5-bit register value、20-bit immediate field and 1-bit function bit. I-Format instruction MVIL R1,FFH shows as Figure 4.

Figure 4. I-Format instruction *MVIL R1,FFH*

3.4 SI-Format instruction

SI-Format is shift instruction. It is composed of 6-bit operation code、5-bit destination register、5-bit source register、15-bit immediate field and 1-bit function bit.

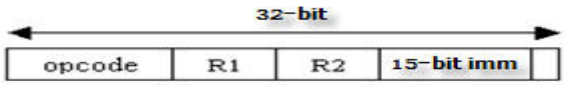

Figure 5. SI-Format instruction *RORI R1,R2,2H*

4. Data path design

We drew data path after we completed instruction set design. Combine all kinds of instruction data path to integrate processor data path. There are usually several data paths, such as R-Format data path、RI-Format data path、load word data path、store word data path、register jump data path.

4.1 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, write the result to register file after ALU outputs result. R-Format data path shows as Figure 6.

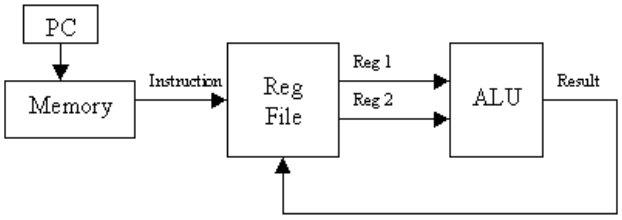

Figure 6. R-Format instruction data path

4.2 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. The immediate is 32-bit signed number which is extend by 20-bit number, and put to ALU as the second operand. Finally, result is write-back to register file. RI-Format data path shows as Figure 7.

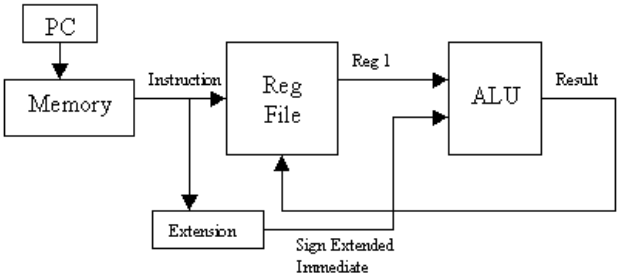

Figure 7. RI-Format ALU instruction data path

4.3 Load word data path

Load word data path is similar to RI-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 RI-Format. In load word data path, fetch data from memory and load it to register file. Load word data path shows as Figure 8.

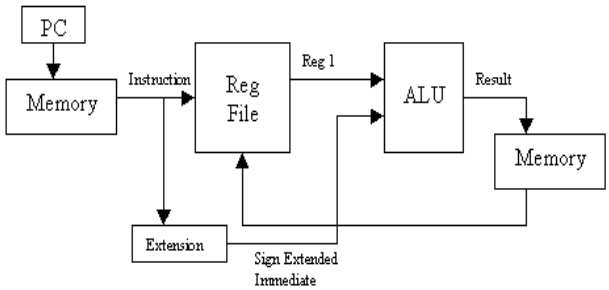

Figure 8. Load word data path

4.4 Store word data path

Store word data path is similar to load word data path, but in store word data path the target that register is to write is memory but not register file. Memory instruction data path shows as Figure 9.

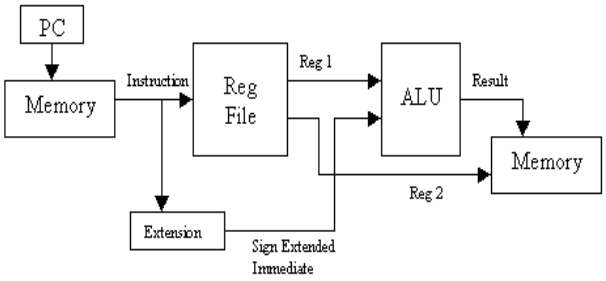

Figure 9. Store instruction data path

4.5 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 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. Jump instruction data path shows as Figure 10.

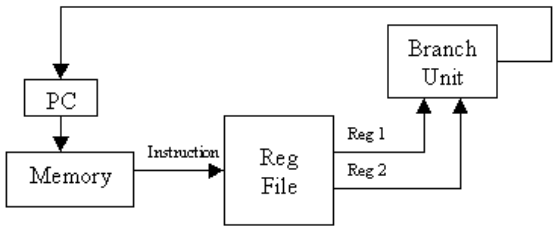

Figure 10. Jump instruction data path

5. Module implementation

The structure of *ID* stage shows as Figure 11 and chip graph shows as Figure 12. *ID* stage includes five modules: *bus_mux_4*、*signext*、*regFile*、*branch* and *idex*. They are all described by VHDL.

ID stage function shows as below:

- Fetch instruction from memory and send it to control unit;
- Control unit decodes instruction and sends decoded control instruction to other unit;
- Fetch data from register file by read register instruction;
- Branch unit judges branch direction.

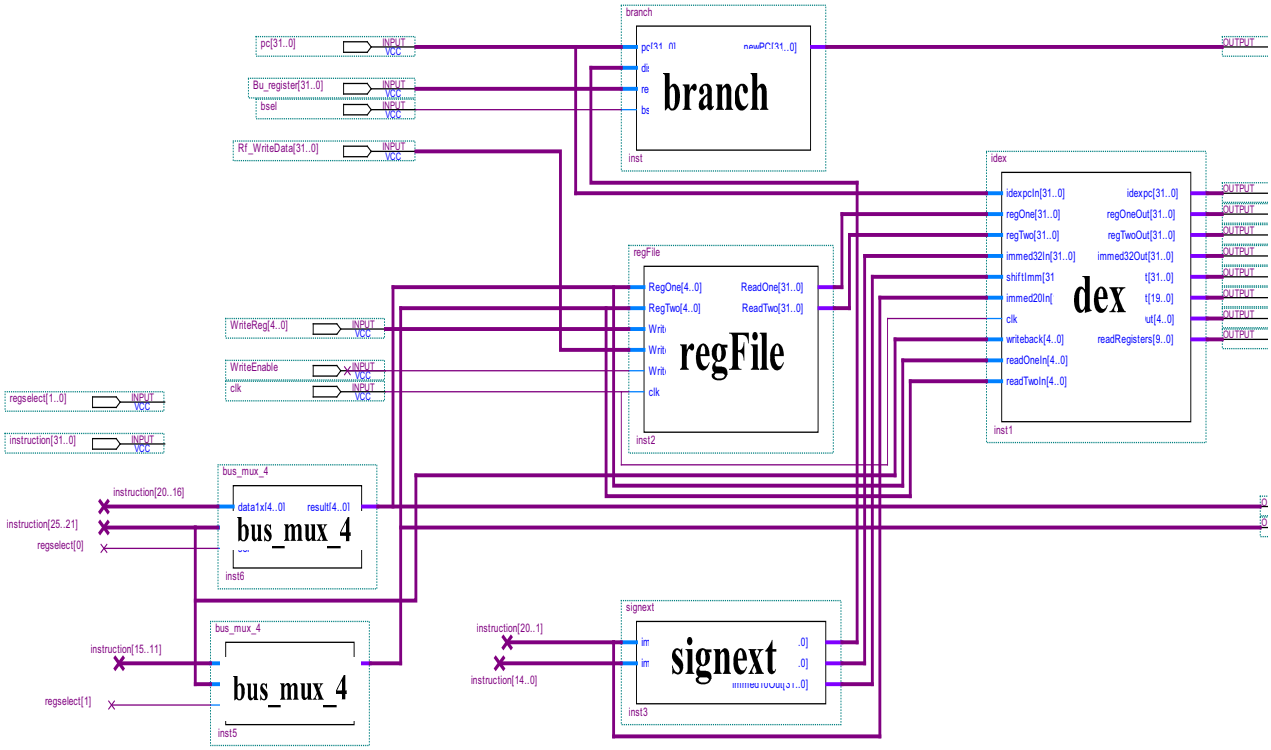

Figure 11. *ID* unit structure

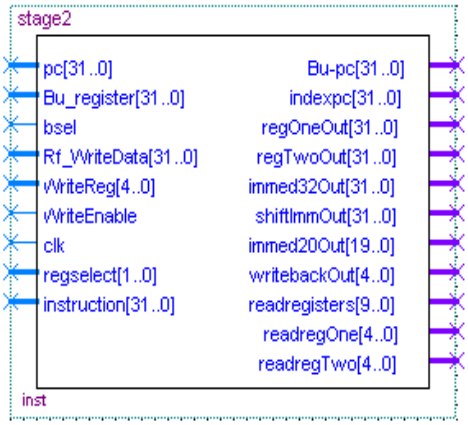

Figure 12. chip graph

5.1 bus_mux_4 module

One of the two input signals data1x[4..0]、data2x[4..0] is sent to output port Result[4..0] depending on select signal sel. The chip graph of bus_mux_4 module shows as Figure 13.

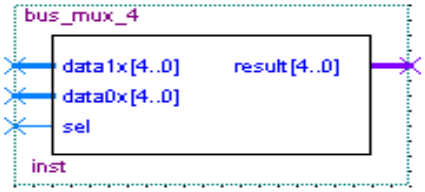

Figure 13. *bus_mux_4* chip graph

5.2 signext module

Depending on sign of operation code(positive or negative), zero-extend or sign-extend immediate in I-Format instruction to generate immediate operand. The chip graph of signext module shows as Figure 14.

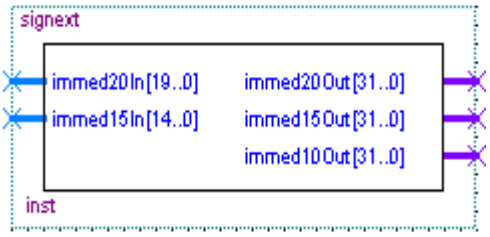

Figure 14. *signext* chip graph

5.3 regFile module

Depending on register request and operation code, fetch data from register file and write data to register file. The chip graph of regFile module shows as Figure 15.

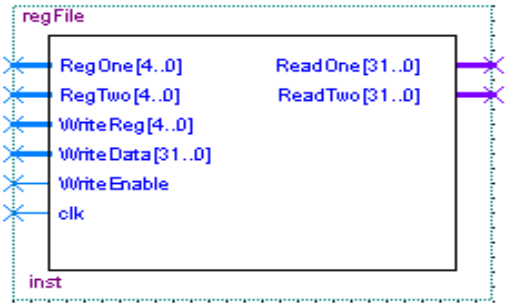

Figure 15. *regFile* chip graph

5.4 branch module

Decide output value by branch unit control signal dsel:
When bsel equals 0, newPC[31..0]=reg[31..0];
When bsel equals 1,
newPC[31..0]=pc[31..0]+displacement[31..0].
The chip graph of regFile module shows as Figure 16. Offset address is usually used in relative jump.

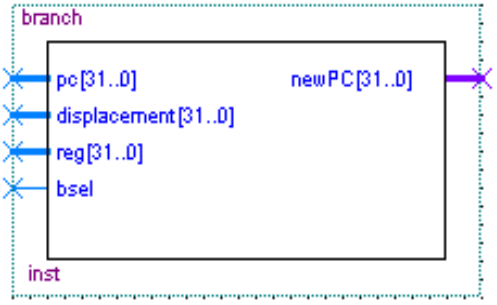

Figure 16. *branch* chip graph

5.5 idex module

idex module is used as signal latch unit in ID stage. The chip graph of idex module shows as Figure 17. Simulation waveform of ID stage shows as Figure 18.

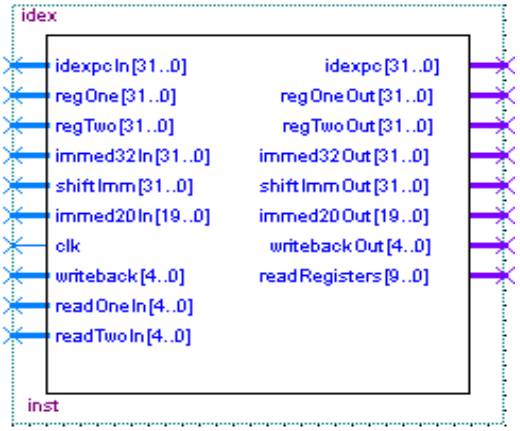

Figure 17. *idex* chip graph

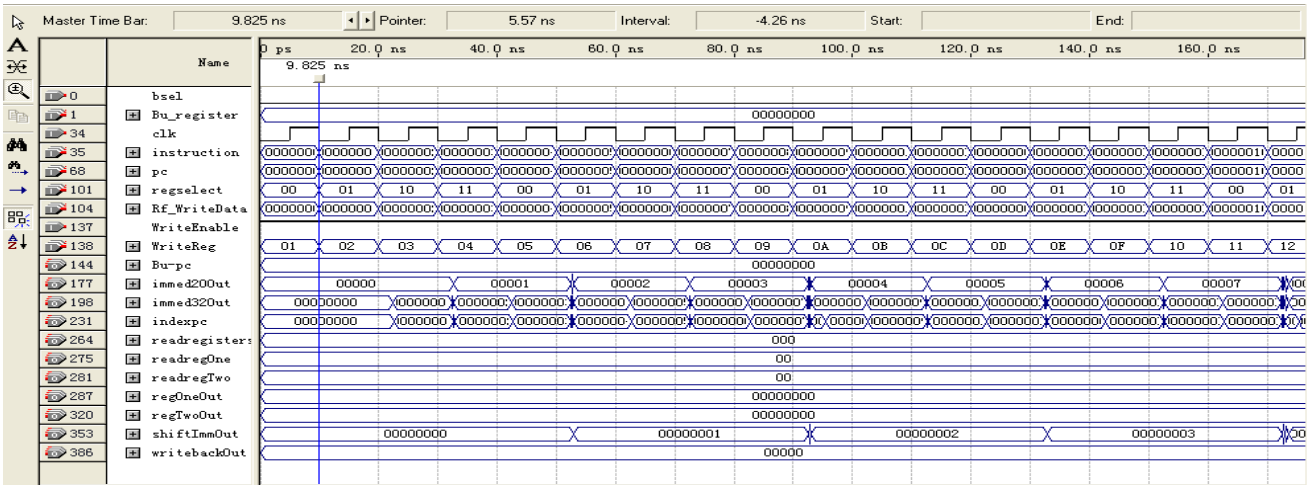

Figure 18. *ID* Stage Simulation Waveform

6. Conclusion

In this paper, we design instruction decode (ID) stage of RISC microprocessor, and emphasize instruction set design principle. Furthermore, we realize ID stage design and analyze module implementation. Design of ID stage simulates、integrates and routes on Quartus II 4.3. The result indicates ID unit completes prospective function, which prepares for next processor design primely.

References

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

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

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

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

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

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

[7] Bai-ZhongYing, Computer Organization, Science Press, 2000.11.

[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.

[12] Yi-Kui, Xiong-Pin, Ding-YueHua , 32 bit Floating-Point Addition and Subtraction ALU Design, DCABES2007, 2007.7.

[13] Ding-YueHua, Yi-Kui, 32 bit Multiplication and Division ALU Design Based on RISC Structure, DCABES2007, 2007.7.