CSC3050 Project 4 Report

118010045 Cui Yuncong

March 2019

Contents

1 Background

1.1 MIPS Architecture

MIPS is a simple, streamlined, highly scalable RISC architecture that is available for licensing. Over time, the architecture has evolved, acquired new technologies and developed a robust ecosystem and comprehensive industry support. Its fundamental characteristics – such as the large number of registers, the number and the character of the instructions, and the visible pipeline delay slots – enable the MIPS architecture to deliver the highest performance per square millimeter for licensable IP cores, as well as high levels of power efficiency for today's SoC designs.

The MIPS architecture is one of the most widely supported of all processor architectures, with a broad infrastructure of standard tools, software and services to help ensure rapid, reliable, cost effective development. Microprocessor developers who want maximum flexibility from processor IP have a solution in the MIPS architecture.

1.2 MIPS Instructions

MIPS I has thirty-two 32-bit general-purpose registers (GPR). Register $0 is hardwired to zero and writes to it are discarded. Register $31 is the link register. For integer multiplication and division instructions, which run asynchronously from other instructions, a pair of 32-bit registers, HI and LO, are provided. There is a small set of instructions for copying data between the general-purpose registers and the HI/LO registers.

The program counter has 32 bits. The two low-order bits always contain zero since MIPS

I instructions are 32 bits long and are aligned to their natural word boundaries.

Instructions are divided into three types: R, I and J. Every instruction starts with a 6-bit opcode. In addition to the opcode, R-type instructions specify three registers, a shift amount field, and a function field; I-type instructions specify two registers and a 16-bit immediate value; J-type instructions follow the opcode with a 26-bit jump target.

The following are the three formats used for the core instruction set:

Type	-31- format (bits) -0-					
R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)
I	opcode (6)	rs (5)	rt (5)	immediate (16)		
J	opcode (6)	address (26)				

Figure 1: MIPS Instruction Format

1.3 Verilog

Hardware description languages such as Verilog are similar to software programming languages because they include ways of describing the propagation time and signal strengths (sensitivity). There are two types of assignment operators; a blocking assignment, and a non-blocking assignment. The non-blocking assignment allows designers to describe a state-machine update without needing to declare and use temporary storage variables. Since these concepts are part of Verilog's language semantics, designers could quickly write descriptions of large circuits in a relatively compact and concise form. At the time of Verilog's introduction (1984), Verilog represented a tremendous productivity improvement for circuit designers

who were already using graphical schematic capture software and specially written software programs to document and simulate electronic circuits.

A Verilog design consists of a hierarchy of modules. Modules encapsulate design hierarchy, and communicate with other modules through a set of declared input, output, and bidirectional ports. Internally, a module can contain any combination of the following: net/-variable declarations (wire, reg, integer, etc.), concurrent and sequential statement blocks, and instances of other modules (sub-hierarchies). Sequential statements are placed inside a begin/end block and executed in sequential order within the block. However, the blocks themselves are executed concurrently, making Verilog a dataflow language.

2 Project Description

2.1 Object

This project will deal with a pipeline processor which can execute MIPS instructions with verilog. This project designs a whole CPU module, including the clock, instruction memory, registers, ALU, data memory and control unit. The project defines the MIPS instruction and build the data path in a verilog file and test the MIPS instructions in the test file, then display the result.

2.2 Functions

For each instruction, this processor need five clock cycle to execute it.

In the first clock cycle, the cpu read one instruction in the instruction memory with the new given pc address.

In the second and third clock cycle is dividing the instruction to different parts and decode the MIPS instruction with the registers and control unit.

The operation code and function code in MIPS instruction are sent to the control unit, which decide how to execute in the next part(Cycle 3).

For data transfer instructions, it can read and write data between the data memory and registers(Cycle 4, 5).

For arithmetic or logic instructions, it can use the ALU module to get the answer, and write it to the registers.

For conditional branch instructions, it can do comparison in the ALU module and branch to the address in the MIPS instruction.

For jump instruction, it can directly jump to the target address.

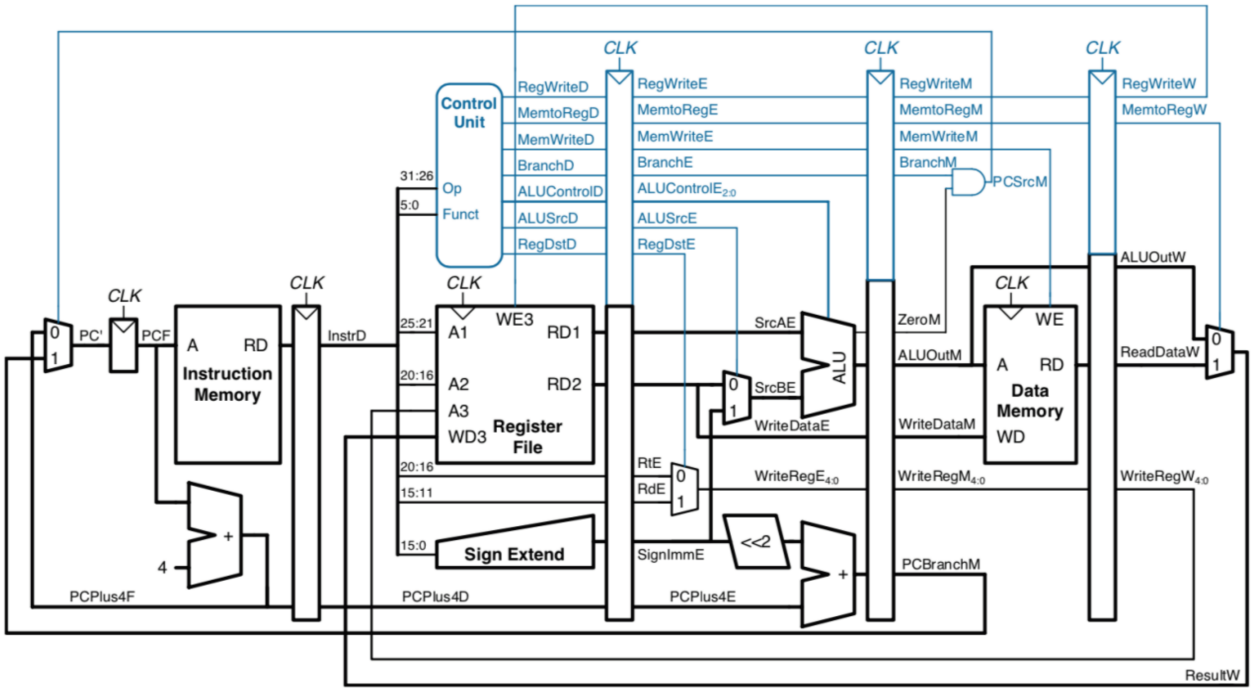

Figure 2: Block Diagram

3 Project Analysis

3.1 Basic Components

According to the characteristic of Verilog Language, some units are omitted and some units are added as well. In this design part, the content is divided into the followings: basic components, five main modules, four link registers and control signals.

The pipeline CPU has three inputs. The clock is used to control the running of the whole CPU. The CPU will execute when the clock is up. The start is used to initialize the CPU. The instr is the instruction transported to CPU from the outside. Besides, there are 32 registers representing the general registers, and a register represents the instruction memory with another register represents the data memory. The same PC controller as the single-cycled CPU. There are other small registers work for the five main modules to store contemporary data.

For each initialization, the first general register is set to zero, and the instruction memory will store the instruction from the instr input.

3.2 Main Modules

3.2.1 Instruction Fetch

The instruction fetch module is responsible for fetching the instruction from the instruction memory according to the PC controller. Besides, based on the Verilog language characteristic, I also let it to calculate the next address earlier to escape the branch hazard.

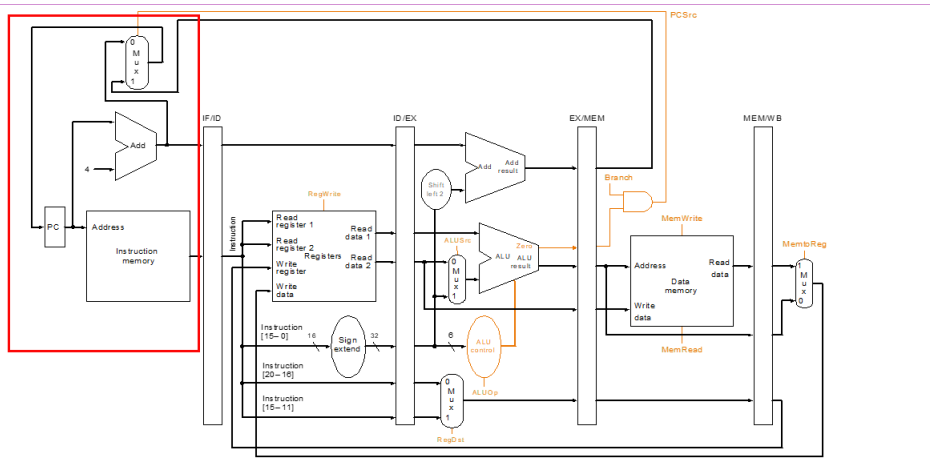

Figure 3: Instruction Fetch

3.2.2 Instruction Decode

The instruction decode module is responsible for decoding the instruction to get opcode, function code and the address of register which will be used for ALU calculation later. The content of the used register or immediate number is prepared for transporting to ALU in this module. Besides, I also let it to produce and update the control signals because this module is the earliest one which can view the whole part of the instruction. What's more, a hazard detect unit is also designed here, which will be introduced later in details.

3.2.3 ALU Execution

This module uses ALU to do the ALU calculation based on the two input registers' content and the control signals produced in the Instruction Decode module. The result of the calculation will be transported to the next module for further processing.

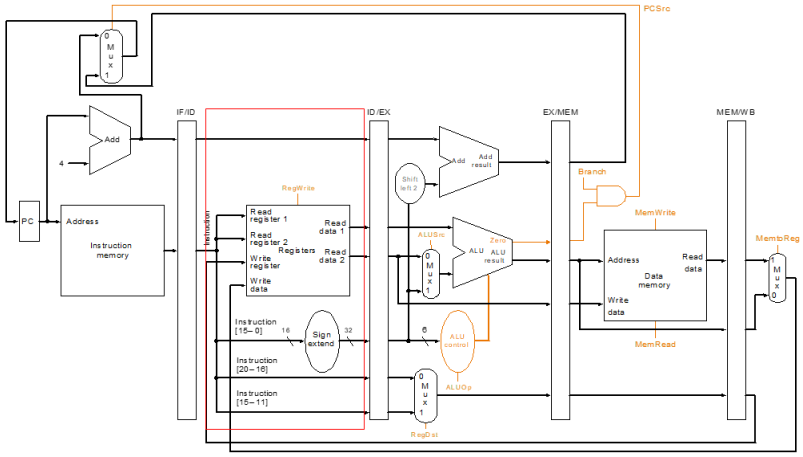

Figure 4: Instruction Decode

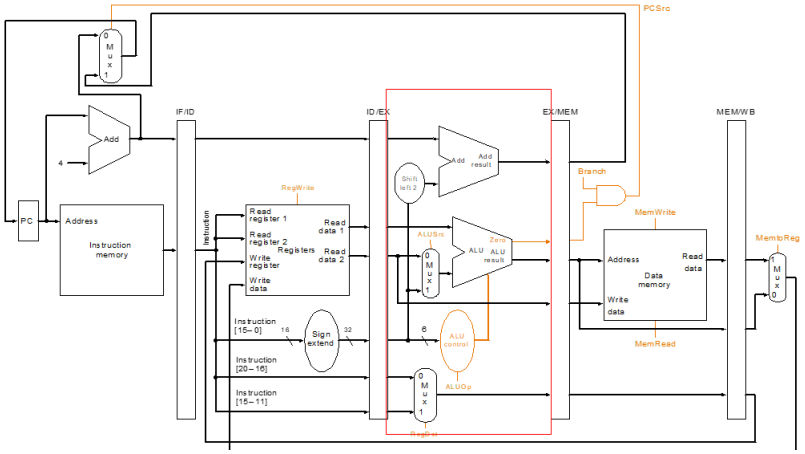

Figure 5: ALU Execution

3.2.4 Memory Operation

Memory Operation module will do the operation related to the memory (load and write) according to the result of ALU calculation (get address) and the control signals. The memory result and the ALU result will be both transported to the next module.

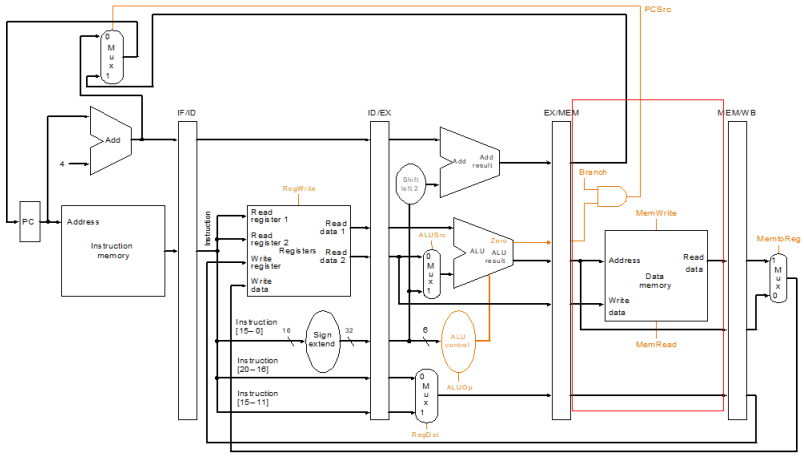

Figure 6: Memory Operation

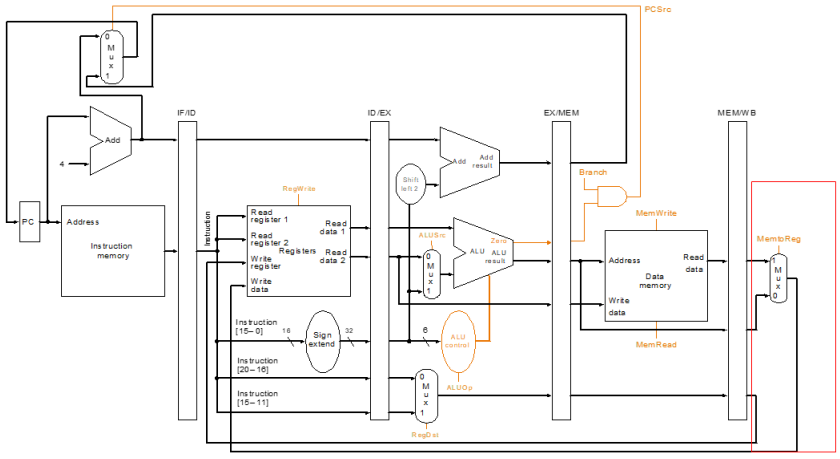

Figure 7: Write Back

3.2.5 Write Back

The write back module will write the result from ALU or memory back to the general register according to the control signals. The control signals decide what data to write and where (which register) to write.

3.3 Four link Registers

Pipeline CPU has four link registers totally, which are IF/ID, ID/EX, EX/MEM, MEM/WB. They are all responsible for storing the information from the previous module and transporting the information to the next modules.

IF/ID is used to store the instruction from Instruction Fetch module and transport it to the Instruction Decode module.

ID/EX is used to store the control signals, two registers' content and address and immediate number from Instruction Decode Module and transport it to ALU Execution module.

EX/MEM is used to store the control signals, ALU calculation result and the register address from the ALU Execution module and transport it to the Memory Operation Module.

MEM/WB is used to store the control signals, ALU calculation result, memory result and the register address from the Memory Operation module and transport it to Write Back Module.

3.4 Control Signals

To control the whole CPU and make it run stably, totally eight control signals are designed here, and they are represented as registers.

For the PC control signal, it is used to control the PC register and the Instruction Fetch Module. When it is 0, Instruction Fetch module is triggered and PC register will be refreshed. When it is 1, Instruction Fetch module is inactive and PC register will stay.

3.5 How to Detect the Hazards

3.5.1 Structure Hazards

There will be no structure hazards in this pipeline CPU because all the instructions should follow the operation in the five main modules and the order is the same. Even though some of the operation is useless for them, but they have to wait there to escape the structure hazards.

3.5.2 Branch Hazards

In this pipeline CPU, the method to solve branch hazards is to make the address calculation earlier. In the traditional pipeline CPU, the calculation of the address is after the ALU calculation, and it's too late and greatly increases the risk of branch hazards. This CPU adds a small ALU in the Instruction Fetch module, and it can calculate the address directly and there's no need for waiting. Therefore, it greatly reduces the risk of branch hazards.

3.5.3 Data Hazards

There are two units in this pipeline CPU to detect data hazards. The first one is bypassing unit:

When the bypassing units is triggered, it will use the result from ALU directly to replace the data from register.

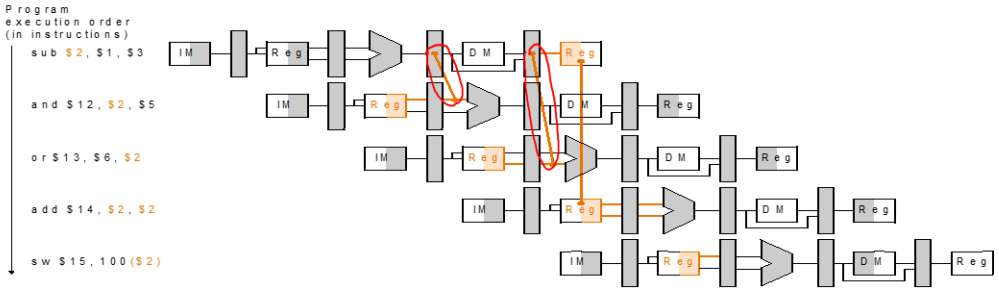

Figure 8: Bypassing Unit

The second unit is stall unit. When the ALU want to use the data from the load word operation, the bypassing operation is not enough because the load word result is from the Memory Operation module instead of ALU Execution Module, and it is later. Therefore, facing such situation, we have to stall the operation by stall units.

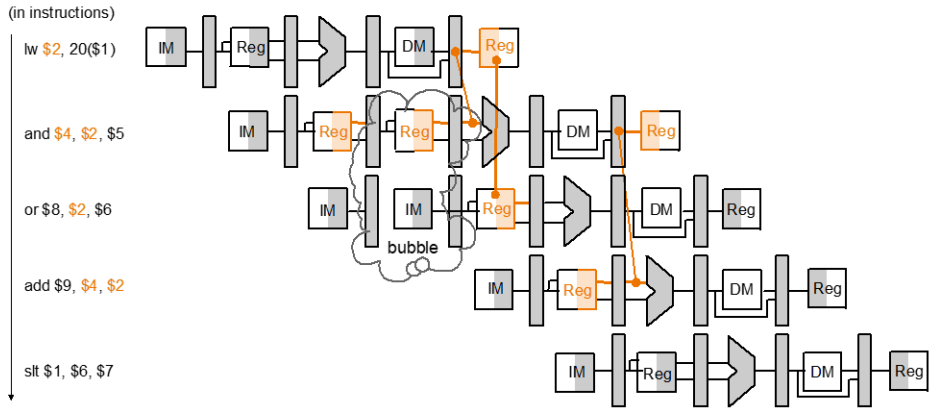

Figure 9: Stall Unit

Stall means delay all the operation one period. Therefore, in this pipeline my method is to set all the control signals to 0, therefore, all the operation including instruction fetching will stop.

4 Test

4.1 File Details

Source Code Filename: CPU.v test_CPU.v

4.2 Virtual System Configuration

VMware Workstation version: VMware Workstation 15.5 Pro

System: Linux version 5.0.0-31-generic (buildd@lcy01-amd64-010) (gcc version 8.3.0 (Ubuntu 8.3.0-6ubuntu1)) #33-Ubuntu SMP Mon Sep 30 18:51:59 UTC 2019

4.3 Compile and Execute

Compile: $ iverilog -o CPU CPU.v test_CPU.v

Demonstrate: $ vvp CPU

4.4 Test Details

4.4.1 Instructions

gr1=gr0+3

gr2=gr1+gr1

gr3=gr2+1

gr4=gr2+gr3

jump 1

gr5=gr1+1

gr5=gr1+3

gr6=slt (gr2¡gr3)

gr7=gr4-gr6

4.4.2 Result

```
cyche@Christian-Computer MINGW64 ~/Desktop/CUHKSZ/CSC3050/Assignment/CSC3050_Project4
$ iverilog -o CPU CPU.v test_CPU.v

cyche@Christian-Computer MINGW64 ~/Desktop/CUHKSZ/CSC3050/Assignment/CSC3050_Project4
$ vvp CPU
instruction=00100000000000010000000000000011 | gr_0=00000000 gr_1=xxxxxxxx gr_2=xxxxxxxx gr_3=xxxxxxxx gr_4=xxxxxxxx gr_5=xxxxxxxx gr_6=xxxxxxxx gr_7=xxxxxxxx pc=00000004
instruction=00000000001000010001000000100000 | gr_0=00000000 gr_1=xxxxxxxx gr_2=xxxxxxxx gr_3=xxxxxxxx gr_4=xxxxxxxx gr_5=xxxxxxxx gr_6=xxxxxxxx gr_7=xxxxxxxx pc=00000008
instruction=00100000010000110000000000000001 | gr_0=00000000 gr_1=xxxxxxxx gr_2=xxxxxxxx gr_3=xxxxxxxx gr_4=xxxxxxxx gr_5=xxxxxxxx gr_6=xxxxxxxx gr_7=xxxxxxxx pc=0000000c
instruction=00000000010000110010000000100000 | gr_0=00000000 gr_1=xxxxxxxx gr_2=xxxxxxxx gr_3=xxxxxxxx gr_4=xxxxxxxx gr_5=xxxxxxxx gr_6=xxxxxxxx gr_7=xxxxxxxx pc=00000010
instruction=00001000000000000000000000000001 | gr_0=00000000 gr_1=00000003 gr_2=xxxxxxxx gr_3=xxxxxxxx gr_4=xxxxxxxx gr_5=xxxxxxxx gr_6=xxxxxxxx gr_7=xxxxxxxx pc=00000018
instruction=00100000001001010000000000000011 | gr_0=00000000 gr_1=00000003 gr_2=00000006 gr_3=xxxxxxxx gr_4=xxxxxxxx gr_5=xxxxxxxx gr_6=xxxxxxxx gr_7=xxxxxxxx pc=0000001c
instruction=00000000010000110011000000101010 | gr_0=00000000 gr_1=00000003 gr_2=00000006 gr_3=00000007 gr_4=xxxxxxxx gr_5=xxxxxxxx gr_6=xxxxxxxx gr_7=xxxxxxxx pc=00000020
instruction=00000000010000110011100000100011 | gr_0=00000000 gr_1=00000003 gr_2=00000006 gr_3=00000007 gr_4=0000000d gr_5=xxxxxxxx gr_6=xxxxxxxx gr_7=xxxxxxxx pc=00000024
instruction=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx | gr_0=00000000 gr_1=00000003 gr_2=00000006 gr_3=00000007 gr_4=0000000d gr_5=xxxxxxxx gr_6=xxxxxxxx gr_7=xxxxxxxx pc=00000028
instruction=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx | gr_0=00000000 gr_1=00000003 gr_2=00000006 gr_3=00000007 gr_4=0000000d gr_5=00000006 gr_6=xxxxxxxx gr_7=xxxxxxxx pc=0000002c
instruction=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx | gr_0=00000000 gr_1=00000003 gr_2=00000006 gr_3=00000007 gr_4=0000000d gr_5=00000006 gr_6=00000001 gr_7=xxxxxxxx pc=00000030
instruction=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx | gr_0=00000000 gr_1=00000003 gr_2=00000006 gr_3=00000007 gr_4=0000000d gr_5=00000006 gr_6=00000001 gr_7=0000000c pc=00000034
instruction=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx | gr_0=00000000 gr_1=00000003 gr_2=00000006 gr_3=00000007 gr_4=0000000d gr_5=00000006 gr_6=00000001 gr_7=0000000c pc=00000038
instruction=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx | gr_0=00000000 gr_1=00000003 gr_2=00000006 gr_3=00000007 gr_4=0000000d gr_5=00000006 gr_6=00000001 gr_7=0000000c pc=0000003c
```

Figure 10: Result

Firstly, all the calculated result is correct and obey the MIPS operation. In this test, the biggest challenge is that data hazard is almost in every instruction. For example, gr2 needs to use gr1, gr3 needs to use gr2, gr4 needs to use gr3 one by one and so on. The correct result indicates that the bypassing unit is designed successfully to detect data hazards. What's more, the data is arranged in a ladder shape, which indicates the greatest feature of pipeline CPU.