Computer Architecture

Dynamic scheduling with Scoreboard Algorithm

14348008, Mingxin Chen, Network Engineering - December 26, 2016

Abstraction

This report is about a program for simulating the dynamic scheduling of instruction in the processor with the algorithm Scoreboard which is usually implied by the hardware in the processor of computers. The program is able to check the dependence between the instructions and try to dispatch it, issue it, execute it and write the result back out of order. I will summarise the algorithm supporting first and then introduce the design briefly, the last part is the usage of the program.

Algorithm

In order to promote the level of instruction-level parallelism (ILP), there are serval ways of pipeline design and instruction scheduling. This first theory is to divide the execution of instruction into serval stages, which can keep all components in the processor busy in an ideal condition with pipeline technology. However, the execution depending may not be as useful as the designers' expectation even if there are bypasses in the pipeline. There are other hazards from which the processor cannot escape, for example, structure hazard(which dose not occur in the five stages pipeline) and WAR hazard which will limit the performance. Designers naturally considered the possibility of Out of Order Scheduling in order to utilise instructions executable to full up the bubbles which generate because of stalling.

In this project, the algorithm for Out of Order Scheduling is named Scoreboard. Scoreboard has no bypass or renaming function, which is just a simply implement for the dynamic scheduling inside of the superscalar processor. However, we are already able to learn the concept of Out of Order from it.

There are four kinds of hazards which needed to be discovered by the processor as the table below. If the hazard was occurred before the instruction entered the stage, the instruction would be being stalled until the hazard disappears.

Stage	D	S	X	W
Hazard	WAW Structure hazard	RAW	-	WAR

The fetch stage had not been put in the table because scoreboard algorithm does not involve the fetch stage which is in order. Although it cannot be deny that structure hazard for instruction buffer may occur if the buffer ran out of the space, we will not consider that

in the program as the limitation in the number of function unit have more negative influence on the performance than the lack of instruction buffer space. So we assume that the instruction buffer space is infinite.

As I have mentioned above, scoreboard will still take the action of stalling when it came to hazard. The method it increases the instruction level parallelism(ILP) is to execute the instructions with no dependence first to burst the bubbles. According to the design pattern, only the structure hazard and RAW hazard can really limit the order handled.

The approach is not complex so it will not be explained in detail.

Design

The design of the simulator can be divided into three parts such as the interaction part that communicates the user and the program, hardware part that copies the elements in the processor and data structure that represents some of the register structures for data transmission.

Interaction Part

Window. The window displaces the status and data inside of the processor and listens to the commands from users. After the user loaded a set of instructions or entered them on the table, the window would try to create a new task if the “START” button was clicked. Then, the user can choose to take one step forward or finish the execution.

Task. The task represents one mission about scheduling a set of instructions. This class is the intermediary between the window which is to interact with users and the processor which is the kernel of the program. After one object of task was created, the user can try to move the processor ahead by one clock or just finish this task. The purpose of setting the class Task is to separate the algorithm logic and the interface design.

Hardware Part

Processor. Processor class contained the objects of the following classes. It can schedule the instruction set in a better order which is the only ambition of the scoreboard algorithm.

Instruction Buffer. Instruction buffer stores the instructions and the information about scheduling such as whether the instruction was finished, the current stage of the instruction, the time of dispatching, issuing, executing and writing back. Limited by the interface design, the maximum of the instructions is seventeen but it can be guaranteed

that the correctness is unbeatable no matter how many the instructions are needed to be executed.

Function Unit Table. Function unit table contains the information of the function units. There are five function units in this case, two for floating point number and one for scalar number, one for loading and one for storing.

Register Status Table. Register status table records the willing to write back for each register according to the information in function unit table. The status is the name of one of function units.

Data Structure

Instruction. Instruction is a class utilised to store the option, source registers, destination register and some execution information of the instruction. It is designed to assist the programming.

Note. Note class contains a static String which records every movement of the processor, which provide an entrance for the user to know the process of dynamic scheduling.

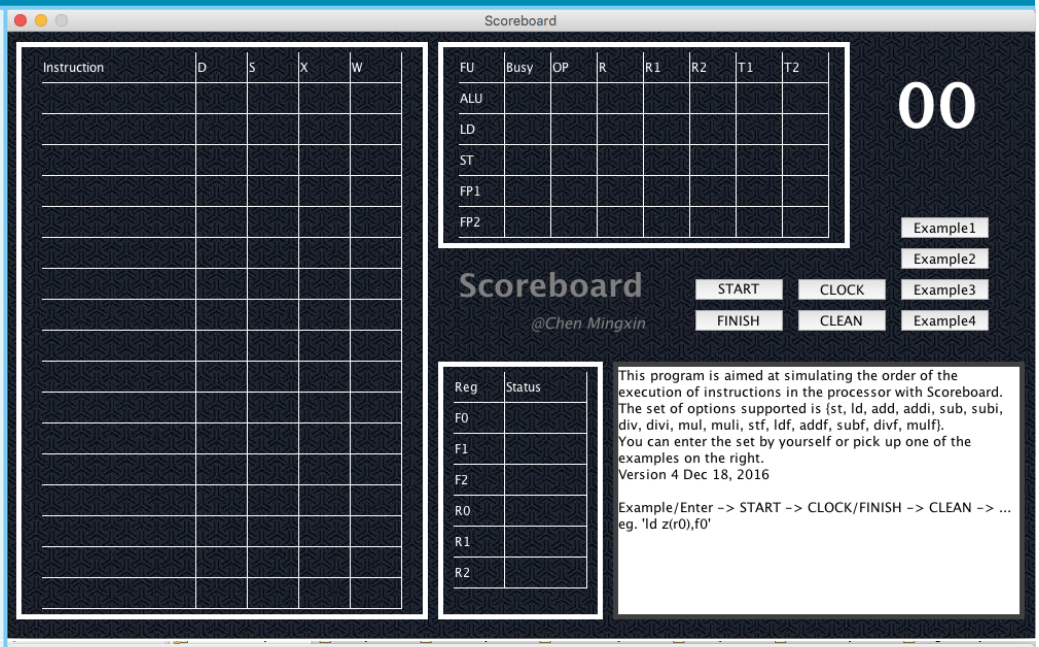
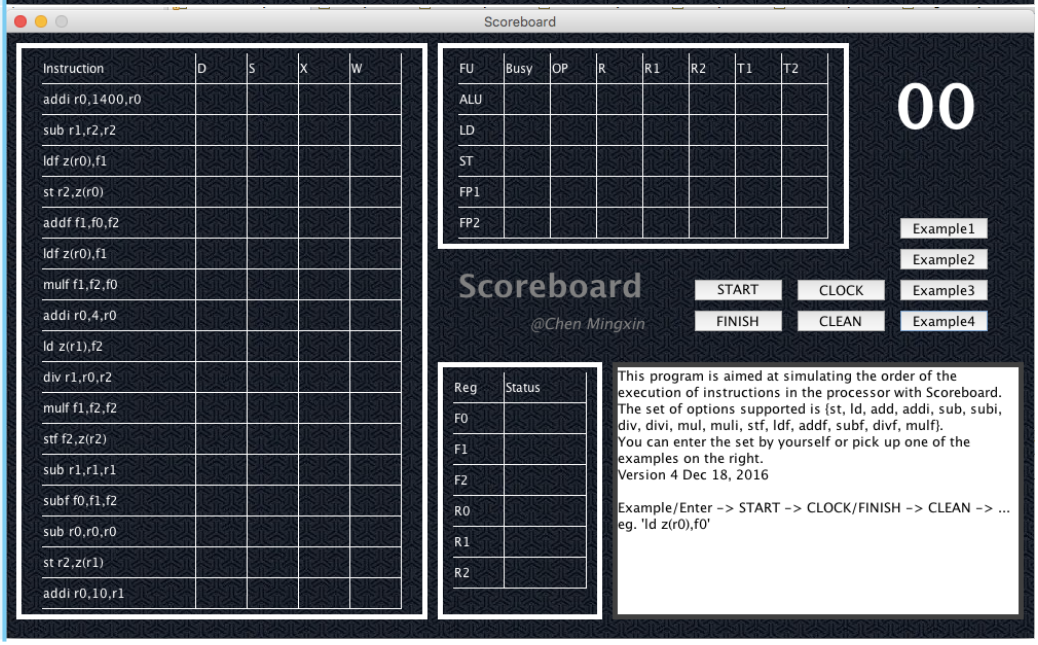
Class	Features	Main Function
Window	Task Screen	
Task	Processor	
Process	Instruction Buffer Set Instruction Set Size Function Unit Table Register Status Current Clock	D Check(is it able to be dispatched) S Check(is it able to be issued) X Check(is it able to be executed) W Check(is it able to be written back) Finish Check(is it able to take one more step)
Instruction Buffer Unit	Instruction D Cycle S Cycle X Cycle W Cycle Finish	
Instruction	Instruction Option R R1 R2 Clock Needed in X stage Finish	Function Unit Needed Check(find out the function unit that which is needed by the certain instruction)

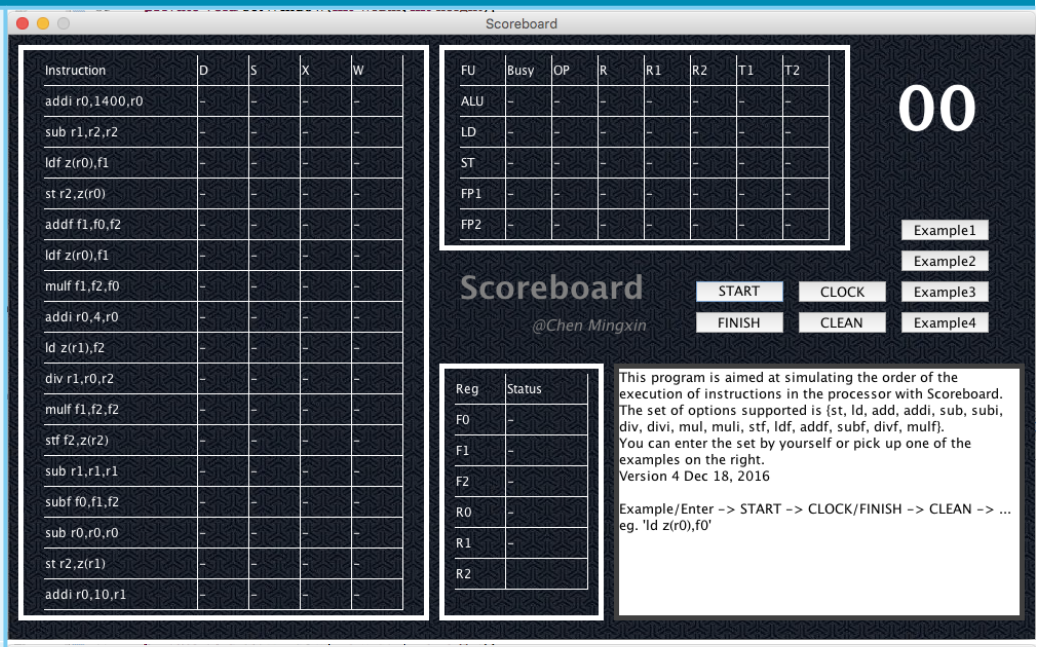
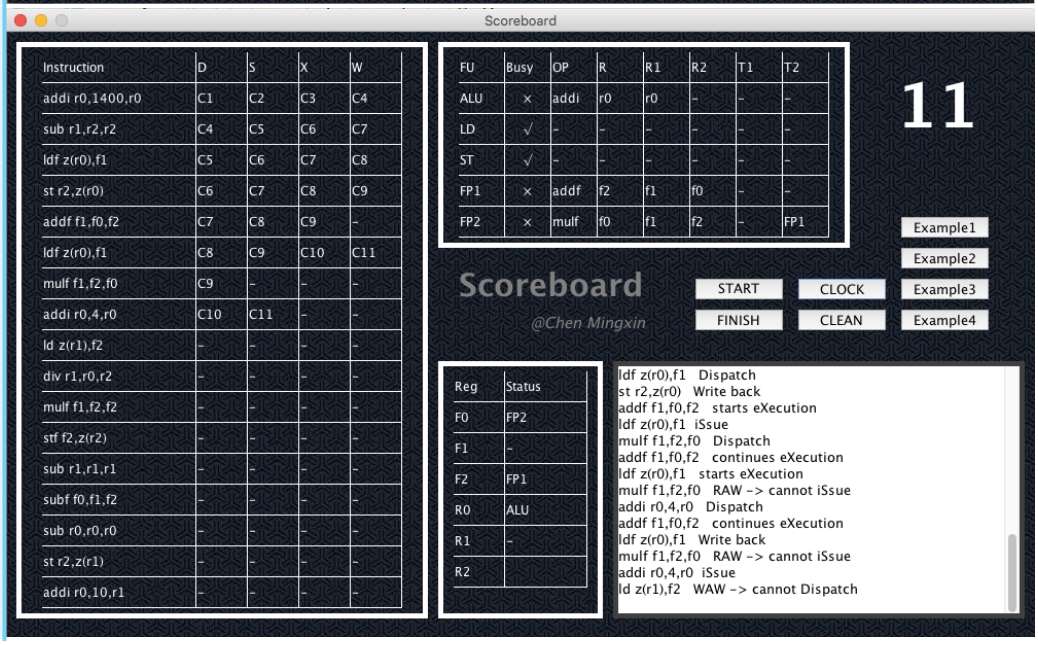
Class	Features	Main Function
Function Unit Table	Function Unit Set Function Unit Set Size	Structure Hazard Check WAR Check RAW Check
Function Unit Table Unit	Function Unit Name Option R R1 R2 T1 T2 Available	
Register Status Table	Register Status Unit Set Register Status Unit Set Sum	WAW Check
Register Status Table Unit	Register Name Status	
Note	Note String	

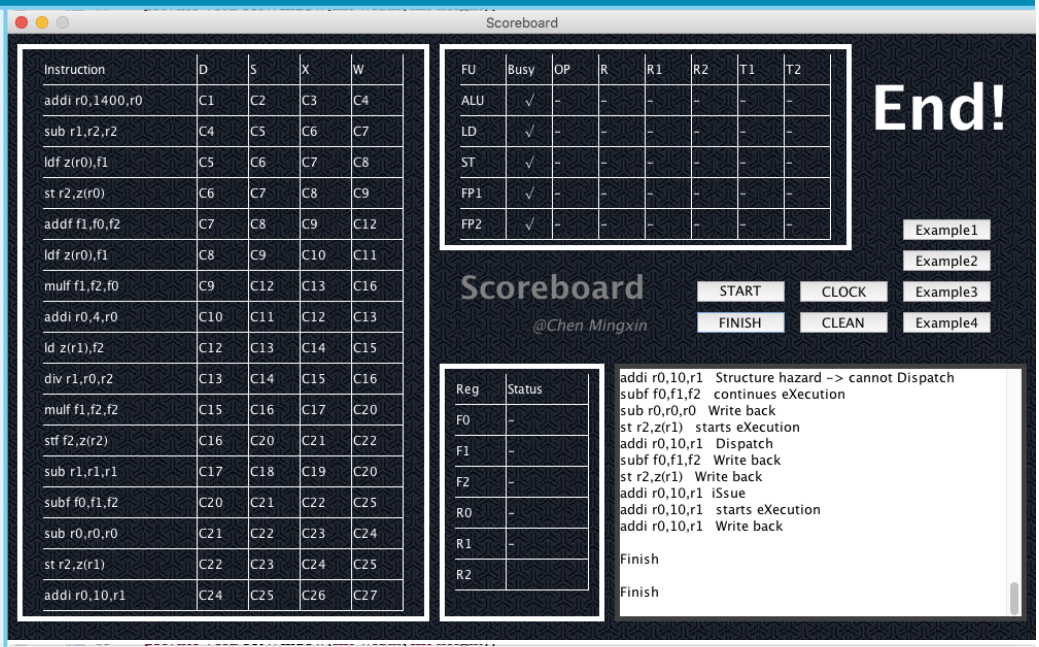
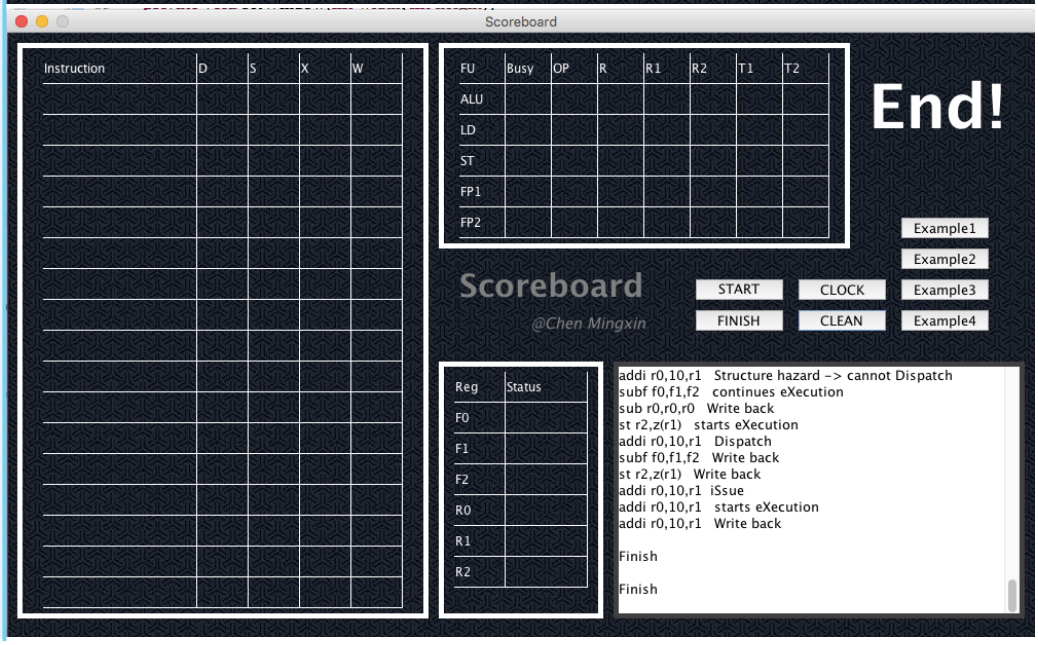
Usage

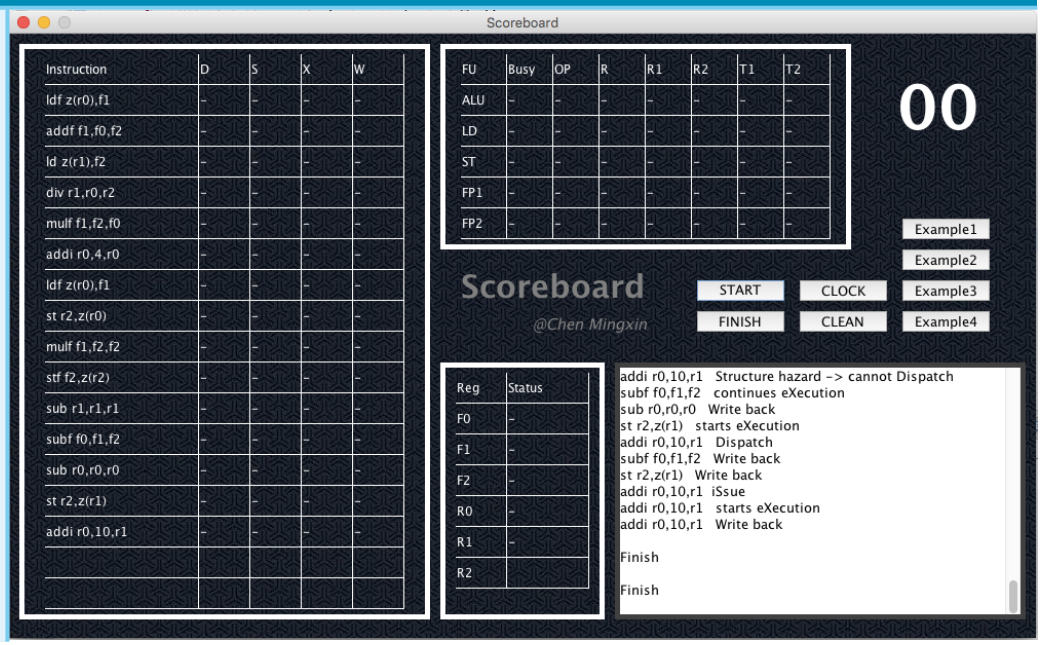
Here are some screen slots taken while the program is running. I would like to use them to explain the usage of my program. There are four examples available for the fundamental tests but the user can input the instruction set randomly.

It is essential to point out that the order of the button to click is unchangeable because the programmer did not consider too many situations for the user's operations. The right order is "ExampleN/Input user-defined instructions -> START -> CLOCK/FINISH -> CLEAN -> ExampleN/Input user-defined instructions -> …".

Operation	Window
Open the jar file	
Click Example 4	

Operation	Window
Click START	
Click CLOCK for 11 times	

Operation	Window
Click FINISH	
Click CLEAN	

Operation	Window
Input the user-defined set of instructions then click START	

Conclusion

It is a pity that scoreboard did not contain bypass, which may be caused by the complexity of superscalar design but, in fact, it is not a fundamental problem. To eliminate the hazard as much as possible, the renaming technology and bypass are indispensable, which is implied in the Tomasulo's Scheduling algorithm. Scoreboard algorithm is the first try of dynamic scheduling by IBM 360 and it proved the possibility of out of order execution with stable ISA semantic.

The history of computer architecture has come a long way. Although it is heard that there was no landmark breakthrough after the golden era in 1960s, it is still worthwhile to be expectant to some subversive concepts, for example, the quantum computer. It is meaningful to learn about the computer architecture especially when we need to measure every slight improvement of the performance of our programs or the outcomes of highly parallel methods. The effort I have made for presenting what I had learned is inappreciable but the men who behind those theories are so respected that their contributions created the life of modern citizens. What I need to do is to understand more valuable ideas about computer and make myself a better programmer.