Restructuring of Visual MIPS Datapath Simulator with Object and Aspect Oriented Approaches

Hidayet Aksu
Özgür Bağlıoğlu
Mümin Cebe

Bilkent University


Agenda

- Introduction
- Problem Case: EZ_MIPS
- Reengineering of Design Patterns
- Analysis of crosscutting concerns
- Discussion and conclusion

Aim of paper

- **to explain the process of reengineering the EZ_MIPS according to**
  - **object oriented**
  - **aspect oriented principles.**

Introduction

- OOP is commonly used during the last decade
- providing many advantages
  - by breaking systems into unit parts to manage the whole system easily,
- **however;**
  - inadequate for encapsulating the concerns such as concurrency, failure handling and logging
  - concerns break down the integrity and cause the coupling of different components.

So what is the solution?

- AOPD (Aspect oriented programing and development)
  - powerful solutions to this difficulty by identifying these concerns and encapsulating them efficiently.
  - separates the cross-cutting concerns, by providing explicit concepts to modularize and integrates with the system components.

What is MIPS(Microprocessor without Interlocked Pipeline Stages)?

- developed a team led by John L. Hennessy at Stanford University in 1981
- based on RISC architecture
- enhance performance through the implementation of instruction pipelines
- focused on entirely on the pipeline making sure it could be run fully integrated.

MIPS

- has a wide variety of usage area in today's computer technology
  - DVD players,
  - Networking devices such as WLAN Access points,
  - televisions,
  - portable devices such as digital cameras,
  - video game consolers such as play station

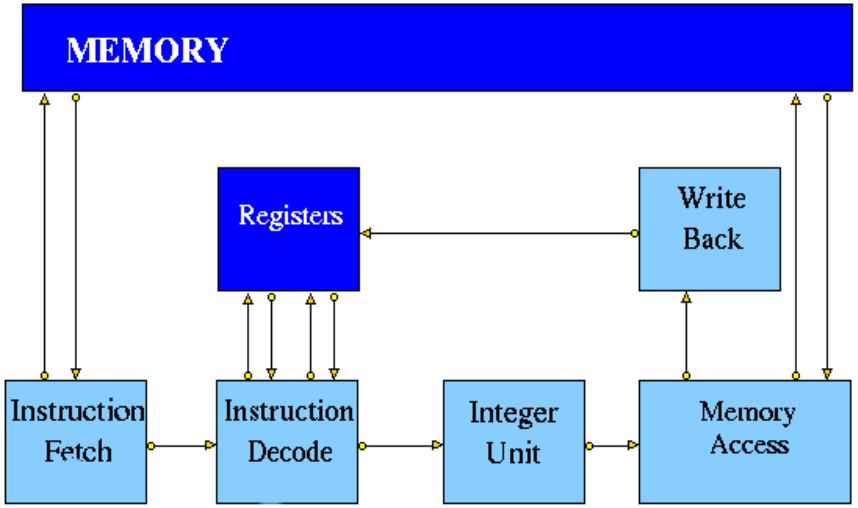
Why MIPS?

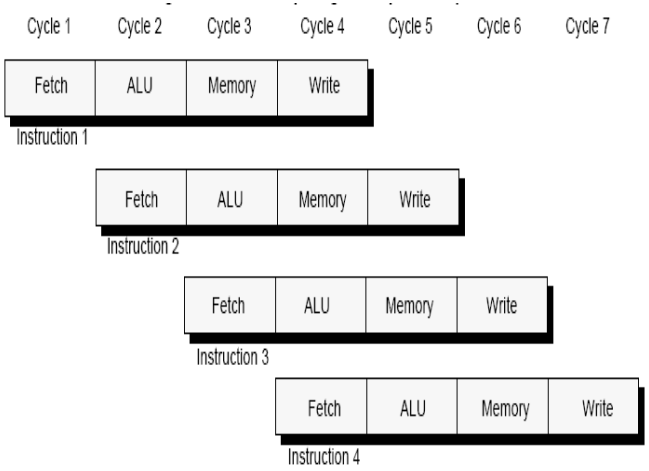
- important to develop a visual MIPS for students to grasp the concepts of MIPS architecture fully.
- EZ_MIPS
  - aims to make his task easier with a visual datapath simulator
  - will be helpful in understanding the MIPS architecture
  - widely discussed in most of Computer Architecture courses

EZ_MIPS

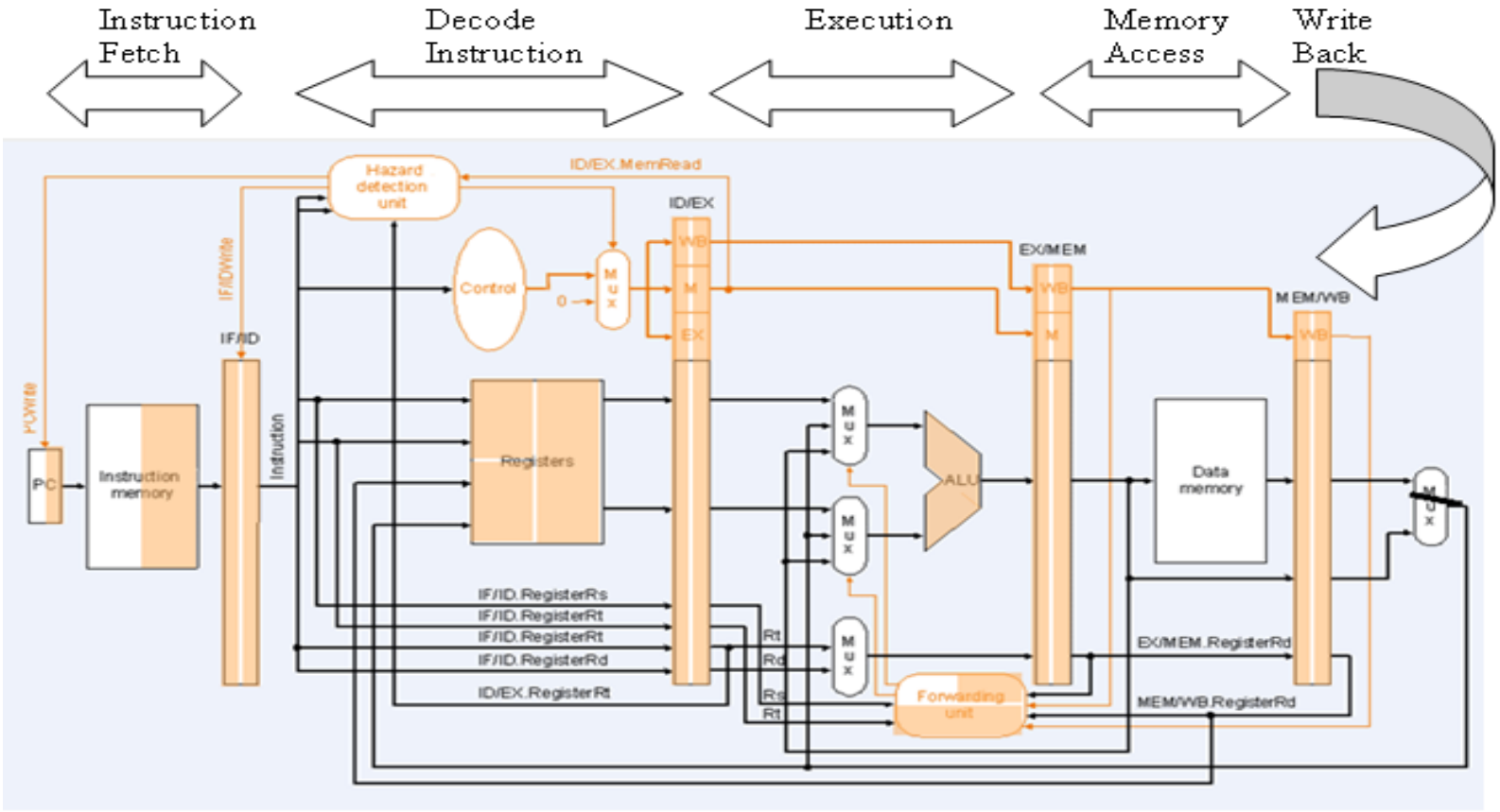
- is first implemented in 2003 with agile programming
- there are some design defects and bugs
- most of the design of modules is improved by using OOP and AOP principles.
- The purpose is to make the program
  - more stable by using de-facto patterns and
  - cleaning code from crosscutting concerns,
  - scattering code segments finally purifying design by means of aspect oriented approaches.

Pipelined Structure of MIPS

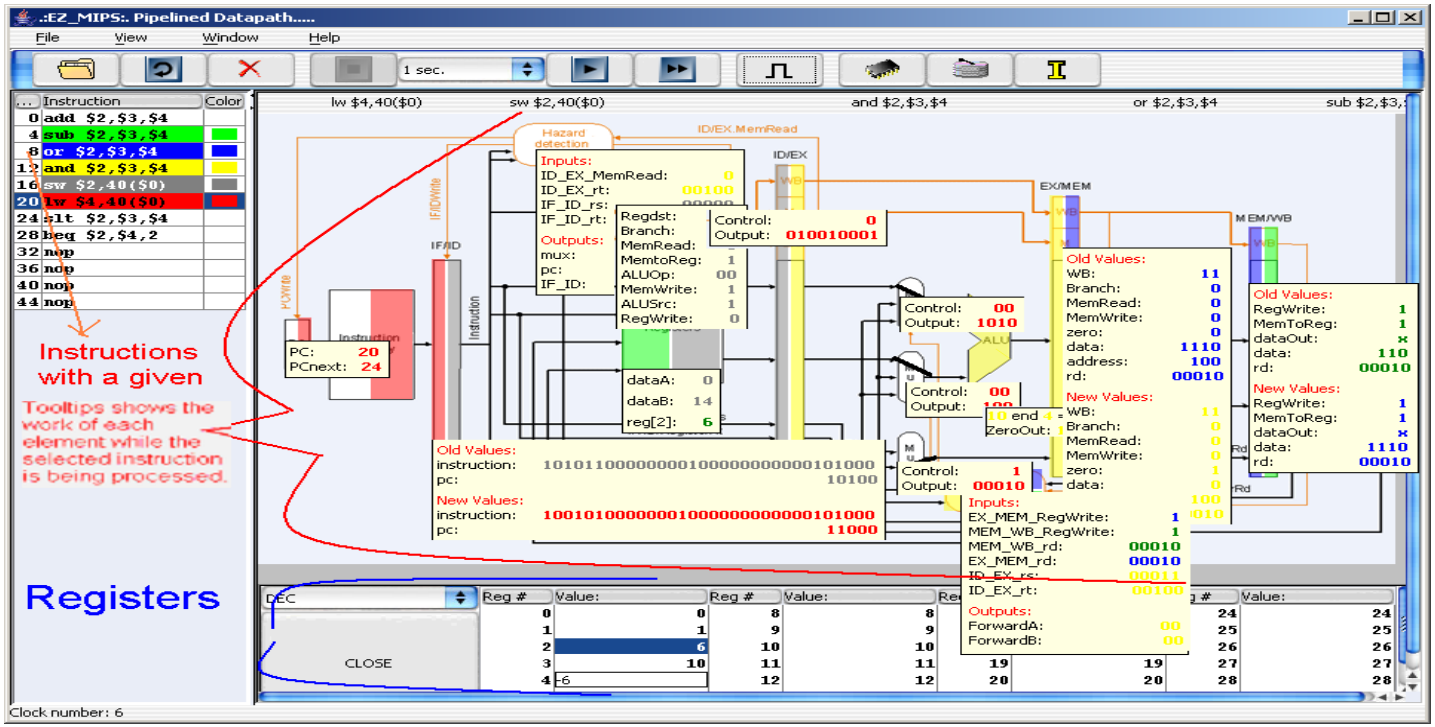




Execution of MIPS(picture from EZ_MIPS)



EZ_MIPS


.:EZ_MIPS:. Pipelined Datapath.....
File
View
Window
Help
1 sec.
Instruction
Color
0 add $2,$3,$4
4 sub $2,$3,$4
8 or $2,$3,$4
12 and $2,$3,$4
16 sw $2,40($0)
20 lw $4,40($0)
24 slt $2,$3,$4
28 beq $2,$4,2
32 nop
36 nop
40 nop
44 nop
lw $4,40($0)
sw $2,40($0)
and $2,$3,$4
or $2,$3,$4
sub $2,$3,
Instructions with a given
Tooltips shows the work of each element while the selected instruction is being processed.
Registers
Hazard detection
Inputs:
ID_EX_MemRead: 0
ID_EX_rt: 00100
IF_ID_rs:
IF_ID_rt:
Outputs:
mux:
pc:
IF_ID:
Regdst:
Branch:
MemRead:
MemtoReg: 1
ALUOp: 00
MemWrite: 1
ALUSrc: 1
RegWrite: 0
Control: 0
Output: 010010001
ID/EX.MemRead
ID/EX
EX/MEM
MEM/WB
IF/ID
PC: 20
PCnext: 24
dataA: 0
dataB: 14
reg[2]: 6
Control: 00
Output: 1010
Control: 00
Output:
Control: 1
Output: 00010
Old Values:
instruction: 10101100000000100000000000101000
pc: 10100
New Values:
instruction: 10010100000001000000000000101000
pc: 11000
Old Values:
WB: 11
Branch: 0
MemRead: 0
MemWrite: 0
zero: 0
data: 1110
address: 100
rd: 00010
New Values:
WB: 11
Branch: 0
MemRead: 0
MemWrite: 0
zero: 1
data: 0
Inputs:
EX_MEM_RegWrite: 1
MEM_WB_RegWrite: 1
MEM_WB_rd: 00010
EX_MEM_rd: 00010
ID_EX_rs: 00011
ID_EX_rt: 00100
Outputs:
ForwardA: 00
ForwardB: 00
Old Values:
RegWrite: 1
MemToReg: 1
dataOut: x
data: 110
rd: 00010
New Values:
RegWrite: 1
MemToReg: 1
dataOut: x
data: 1110
rd: 00010
DEC
CLOSE
Reg #
Value:
Clock number: 6

EZ_MIPS features:

- Control of execution speed, including single step to variable speed
- Thirty-two registers and 1KB memory unit visible at the same time, selectable via tables,
- “spreadsheet” (WYSIWYG) modification of values in registers and memory and instruction sets,
- Selection of data value display in binary, decimal or hexadecimal formats
- toolbar icons for every menu item

EZ_MIPS features (cont.)

- Different look-and-feel options.
- Have opportunity to start execution of instructions from selected instruction step of instruction set.
- Modification of instructions read from file in anytime.
- Showing the status of program in information bar.
- Profiling of environment variables in simulation (new feature comes from AOP approach)

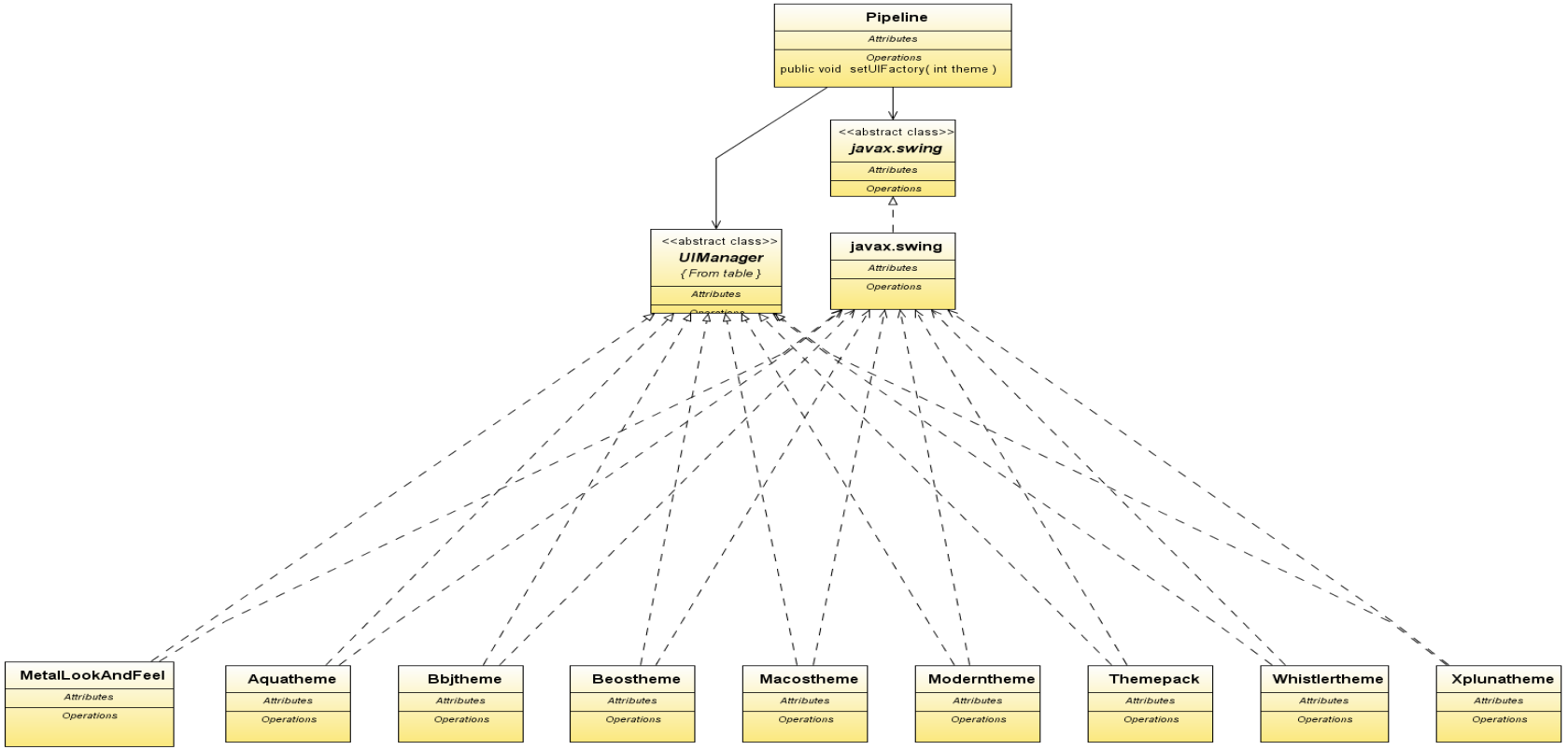
Reengineering of Design Patterns

- we followed a migration strategy
  - an incremental approach widely used in reengineering of current systems.
- The strategy
  - starts with analyzing the system from the object oriented perspective,
  - finding the demands and problems in current design
  - prototyping the target solution for the defective parts
  - incrementally migrating from current to target solution.

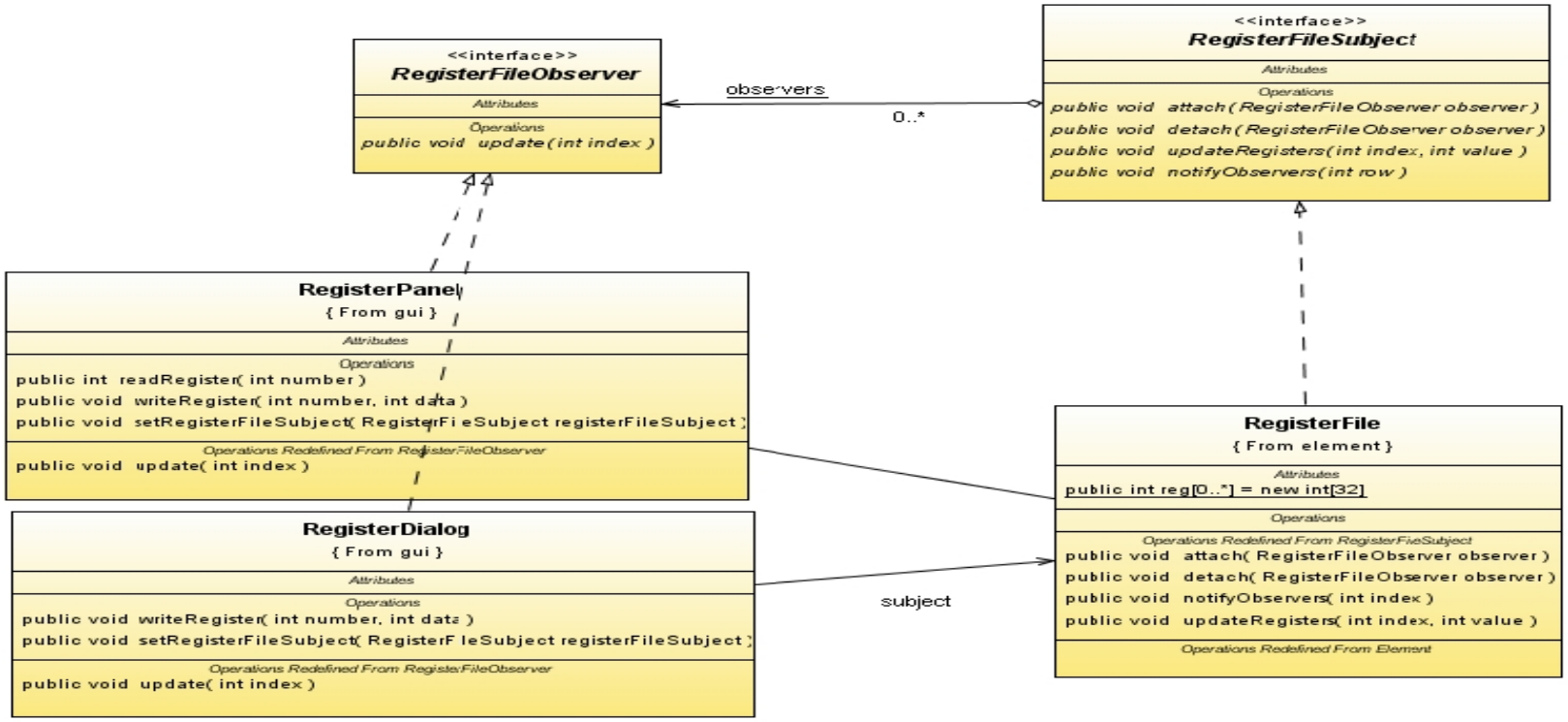
Reengineering of Design Patterns (cont.)

- In this process,
  - always have a running version in all steps of migration
  - with each unit of change in design we tested the system's stability.
- After attaching new patterns
  - Acstract factory, observer and state pattern.
- We have more cohesive and less coupled code.

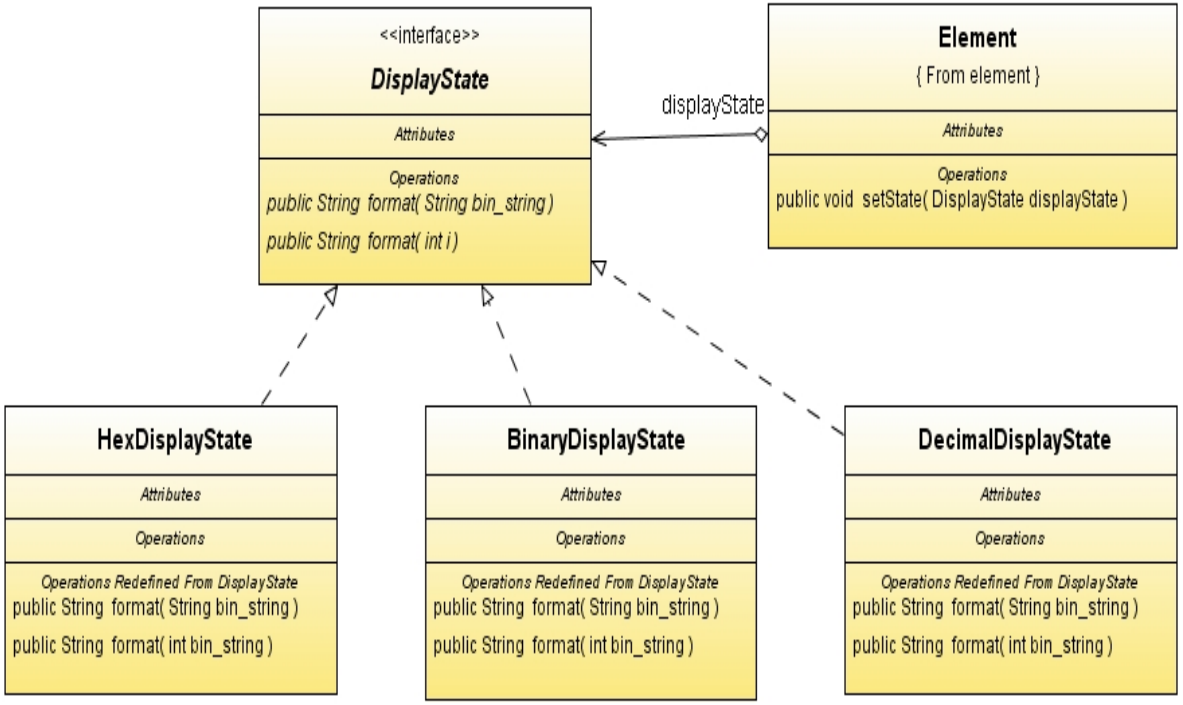
Abstract Factory Pattern



Observer Pattern

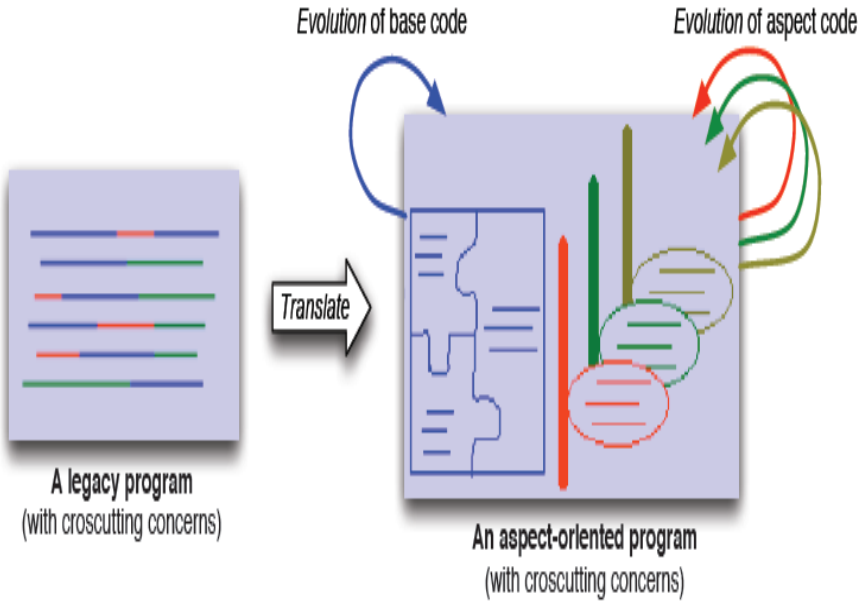


State Pattern



Analysis and solution of Aspects

- translate existing code into their aspect oriented equivalents,re-factor them continuously
- Refactoring process starts with identification of concerns
- redesigning of them using AOP principles and implementing these concerns.



Aspects identified in the System

- ***Development Aspects:*** can be easily removed from production builds
  - ***Policies and Enforcements***
  - ***Consistency Verification***
- ***Production Aspects:*** included in the production builds of an application.
  - ***Tooltip Concern***
  - ***Profiling of Simulation Variants***

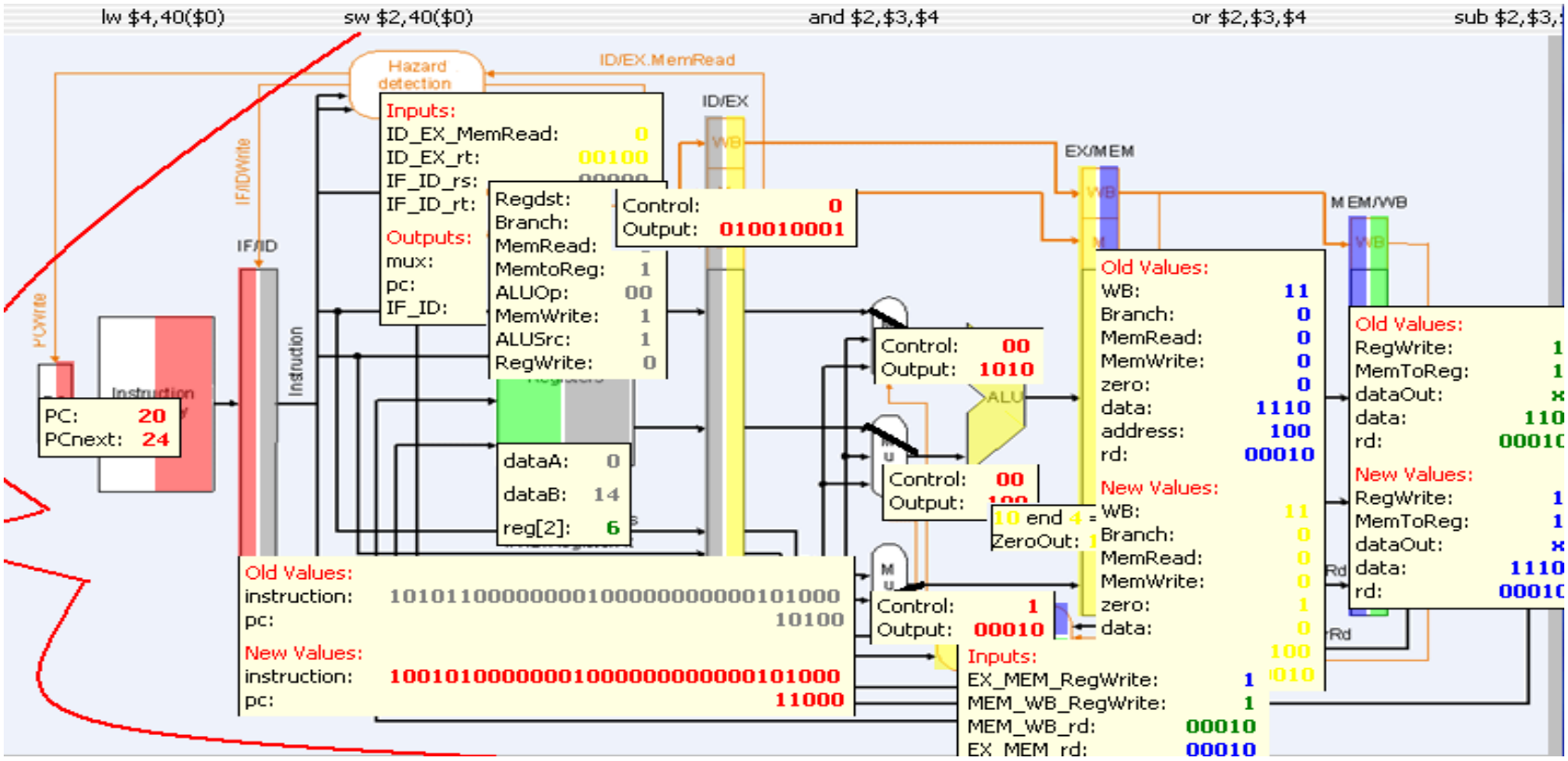
Policies and enforcements:

- are the core components in the project's life cycle.
- make all the development more concrete, organized, and systematic.
- we have defined some rules to improve code quality in a team
- Some of these implementation policies are as follows:
  - There should not be no System.out.{print*(),error*()}
  - The Class, method and variable names should be compliant with Java's coding conventions

Consistency Verification

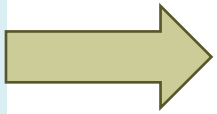
- for providing the correctness of execution of each element.
- control whether the code of element fine or not, we must put some checker into pre and post conditions of each operations.
- Example:
- ALU unit must get 2 objects created by 2 MUX element
- Also the inputs must have some characteristics
- As ALU all other elements have some pre and post conditions (object creation, object format of input and output elements).

Tooltip Concern



Tooltip Concern

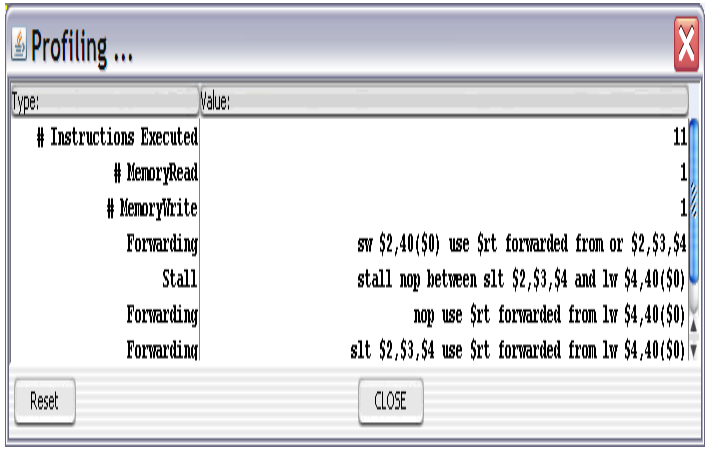
```
if (control == "010") {       // addition
     result = a + b;
     tooltip = "<html>" + colorA + a + End + " + "
     + colorB + b + End + " = " + colorR + result
     + End + ' ';
 } else if (control == "110") {     // substruction
     result = a - b;     //
     tooltip = "<html>" + colorA + a + End + " - "
     + colorB + b + End + " = " + colorR + result
     + End + ' ';
 }
  else if (control == "000") {      // and
     result = a & b;
     tooltip = "<html>" + colorA + a + End + " end
     " + colorB + b + End + " = " + colorR +
     result + End + ' ';
 } else if (control == "001") {     // or
     result = a | b;
     tooltip = "<html>" + colorA + a + End + " or
     " + colorB + b + End + " = " + colorR +
     result + End + ' ';
 }
```



```
public privileged aspect TooltipAspect {
        pointcut adderwork(ALU alu ):this(alu) &&
          (execution(void ALU.work(boolean))|| execution(* Element.setState(..)));

        after(ALU alu ): adderwork (alu)
  {
                if(alu.output == null ){
                        alu.setToolTipText("ALU....");
                        return;
                }
                tooltip = "<html>" + colorA + formatted_a + End + op + colorB
                                + formatted_b + End + " = "
                                + colorR + formatted_out + End
                                + ' ' + "<br>aspect ZeroOut: " + colorR
                                + (wireSet.wireSet[36].getValue())  + "</font>"
                                + "</html>";

                alu.setToolTipText(tooltip);

        };
}
```

Profiling of simulation variants

- To measure the totality of element's behavior from invocation to termination,
- to make statistical summary of the events observed
- to see ongoing interaction



Summary

- Improved the code quality
  - regarded as software refactoring, aims developing the internal structure without changing external.
  - includes adding new design patterns
  - finds existing concerns or tangling code segments scattered over objects such as tooltip concern.
- Add new features using AOP
  - profiling feature of simulation in order user to monitor the MIPS

Results of Restructuring

- The components of software are better modularized.
- better separation of concerns and
- a clearer and cohesive responsibility of the system components,
- The end products are better maintainable and reusable.
- a better modularized and simpler design, resulting in a reduction of costs.

Conclusion

- we had a chance to make our hands dirty with experience on AOSD
- by practical usage of aspects
  - give a PoC that aspect oriented development will ease the implementation of crosscutting concerns
- Lastly we also improved the quality of code by eliminating tangling code segments.

Acknowledgement

- Thanks to Asst. Prof. Dr. Bedir Tekinerdogan, for generously sharing his comments and feedback with us.
- Also thanks to our class mates and all the participants of the TAOSD 2008, with whom we had a most beneficial workshop, sharing experiences related to AOP.

Questions, welcome!