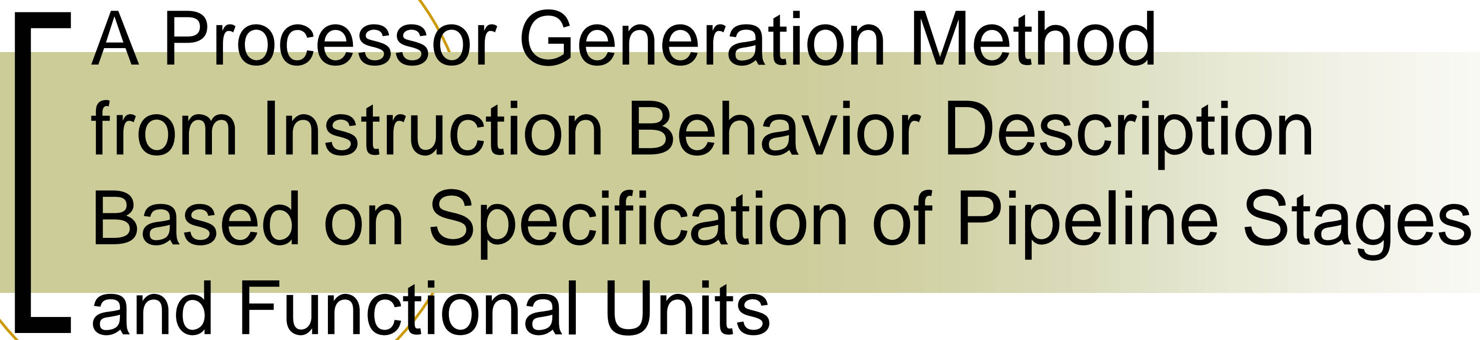


A Processor Generation Method from Instruction Behavior Description Based on Specification of Pipeline Stages and Functional Units

Takeshi Shiro, Masaaki Abe,
Keishi Sakanushi, Yoshinori Takeuchi, and Masaharu Imai

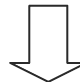
Graduate School of Information Science and Technology,
Osaka University, Japan

Outline

- Background
- Proposed Processor Generation Method
- Experiments
- Conclusion

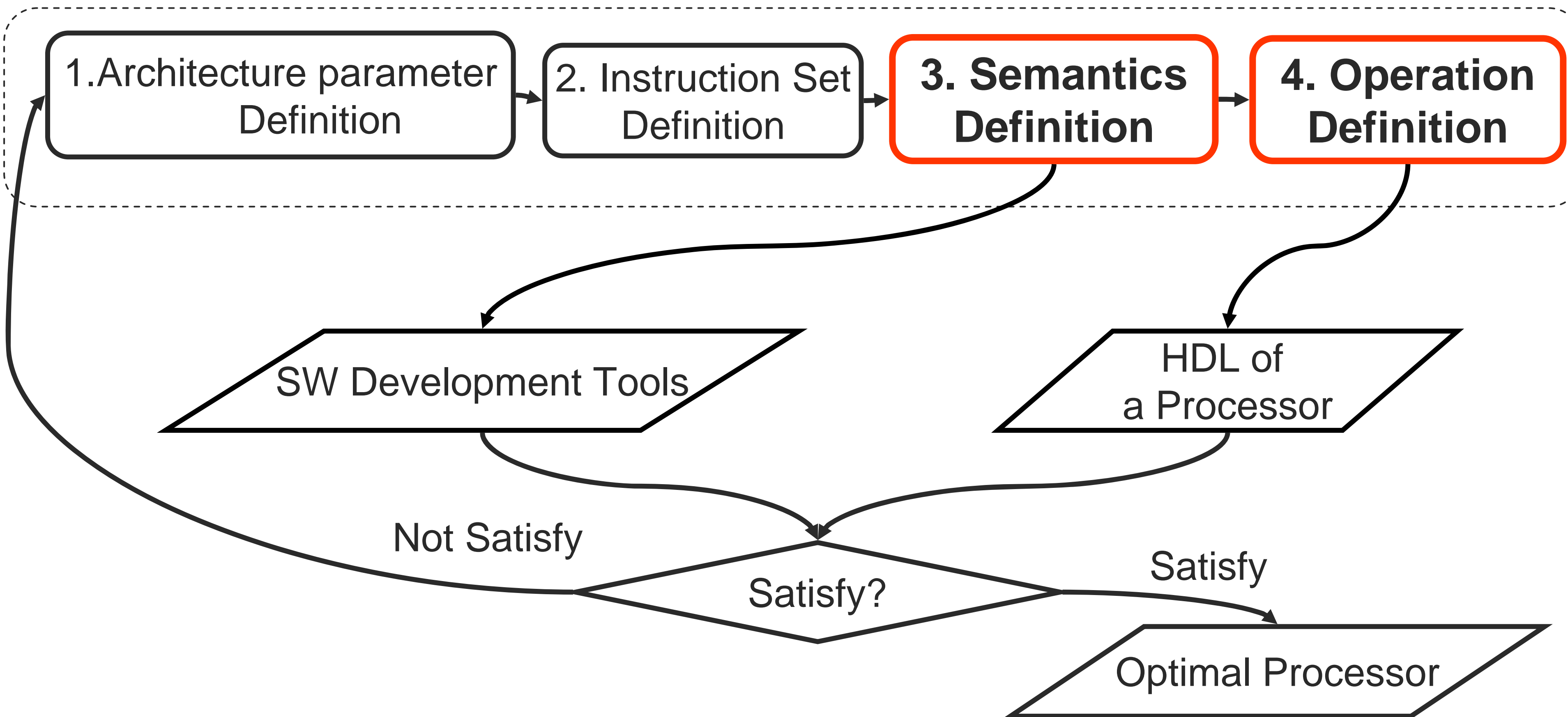
Background

- Application-Specific Instruction set Processors (ASIPs)
  - More flexible than ASICs
  - Higher performance than general purpose processors
- Design Space Exploration (DSE)
  - Explore and evaluate various architectures
  - Requirements
    - Design and modify various processors within a limited time
    - Compilers, assemblers, and simulators are demanded



Processor Design Environment is proposed

Design Flow of Processor Design Environment



Semantics Definition and Operation Definition

- Semantics Definition
  - For software development tools
  - Defined by behavior description
    - Not specify pipeline stages and functional units
- Operation Definition
  - For HDL description
  - Defined by micro-operation description
    - Specify pipeline stages and functional units

Behavior Description of Instruction ADD

```
GPR[rd] = GPR[rs0] + GPR[rs1];
```

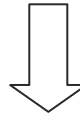
Micro-Operation Description of Instruction ADD

```
Stage1:  current_pc = PC.read();
         inst = IMEM.read(current_pc);
         IR.write(inst);
         PC.inc();
Stage2:  source0 = GPR.read0(rs0);
         source1 = GPR.read1(rs1);
Stage3:  result
            = ALU.add(source0,source1);
Stage4:
Stage5:  GPR.write0(rd,result);
```

Problems and Solutions

- Problems
  - Describing micro-operation description takes up half of processor design time
    - Code size of micro-operation description is more than that of behavior description
  - Consistency between two descriptions is required
    - Human error may be occur

→Instructions should be defined by only one description



Generate micro-operation description from behavior description

Outline

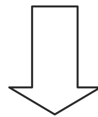
- Background
- Proposed Processor Generation Method
- Experiments
- Conclusion

Generation Flow of the Proposed Method

1. Construct Abstract Syntax Trees (ASTs) from the behavior description
2. Generate micro-operation fragments
   - Micro-operation descriptions without specification of pipeline stages and functional units
3. Allocate the micro-operation fragments to the pipeline stages.
4. Define a functional unit
   for each micro-operation fragment

Assumption

- The behavior description must be complemented with the following information
  - Allocating micro-operation fragments to pipeline stages
  - Binding functional units to micro-operation fragments



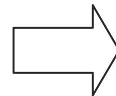
- Give attribute to each pipeline stage
  - execution, memory read, etc.
- Define only one functional unit for a certain function

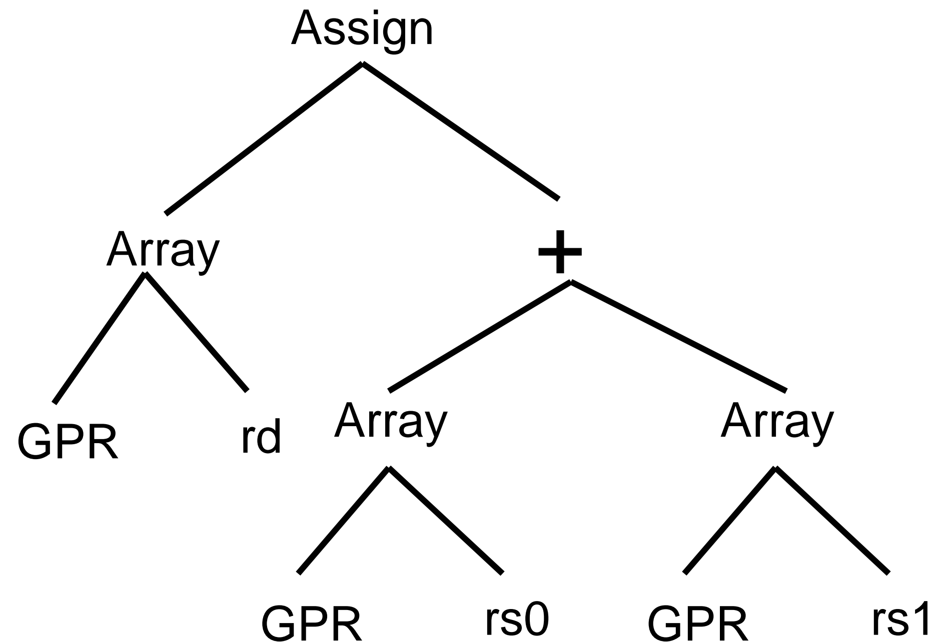
Construction of Abstract Syntax Trees (ASTs)

- Parse behavior description, and construct AST

Instruction ADD

GPR[rd]
  = GPR[rs0] + GPR[rs1]





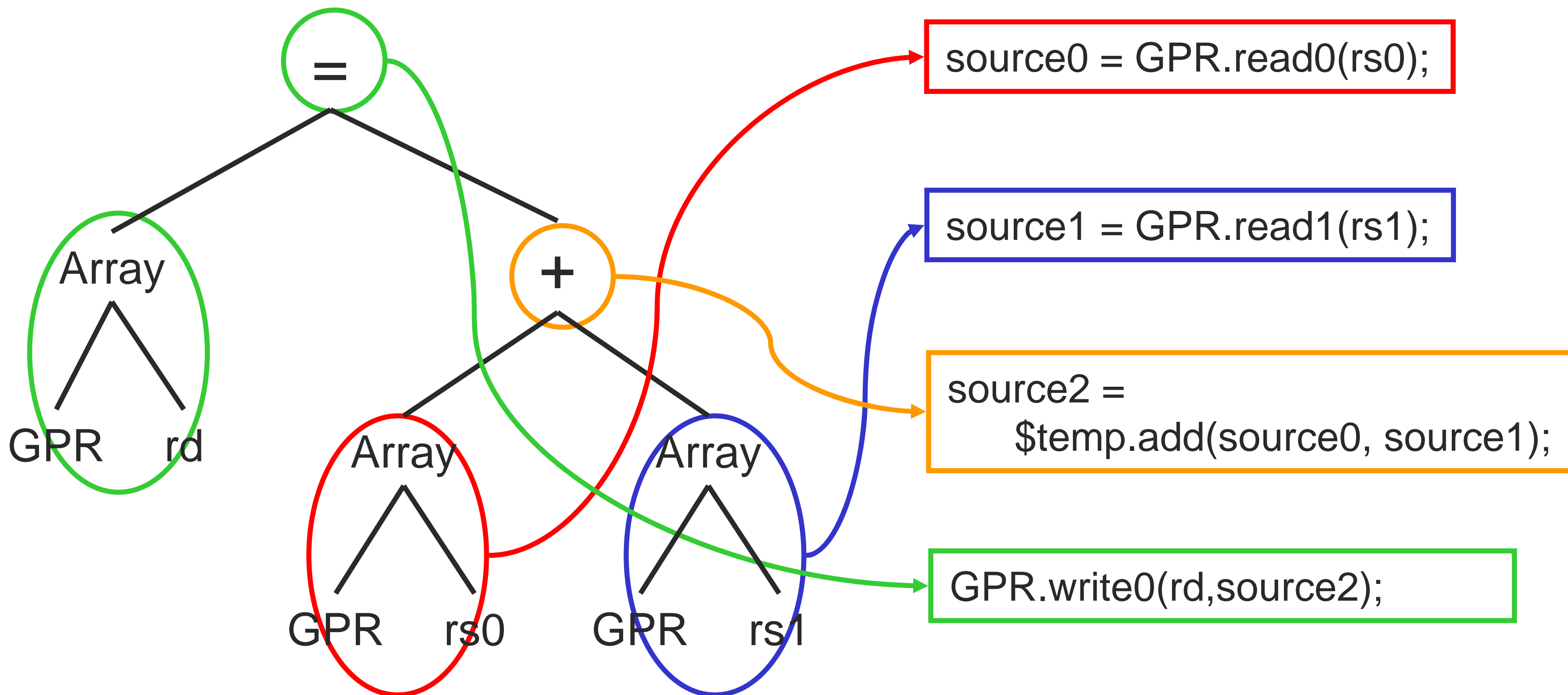
Generation of Micro-Operation fragments

- Micro-operation fragments
  - Micro-operation description without specification of pipeline stages and functional units
  - result = $temp.add(source0,source1);

- Generate micro-operation fragments by scanning generated ASTs
- Generate micro-operation fragments of operator nodes
  - Functional units are not decided yet
  - Only functions are decided
    - add, addu, mul, etc…

Example of Generating Micro-Operation fragments

Instruction ADD
GPR[rd] = GPR[rs0] + GPR[rs1];



Allocation to Pipeline Stages

- Allocate each micro-operation fragment to pipeline stage
- Attribute of each stage
  - Given in architecture definition step by a designer

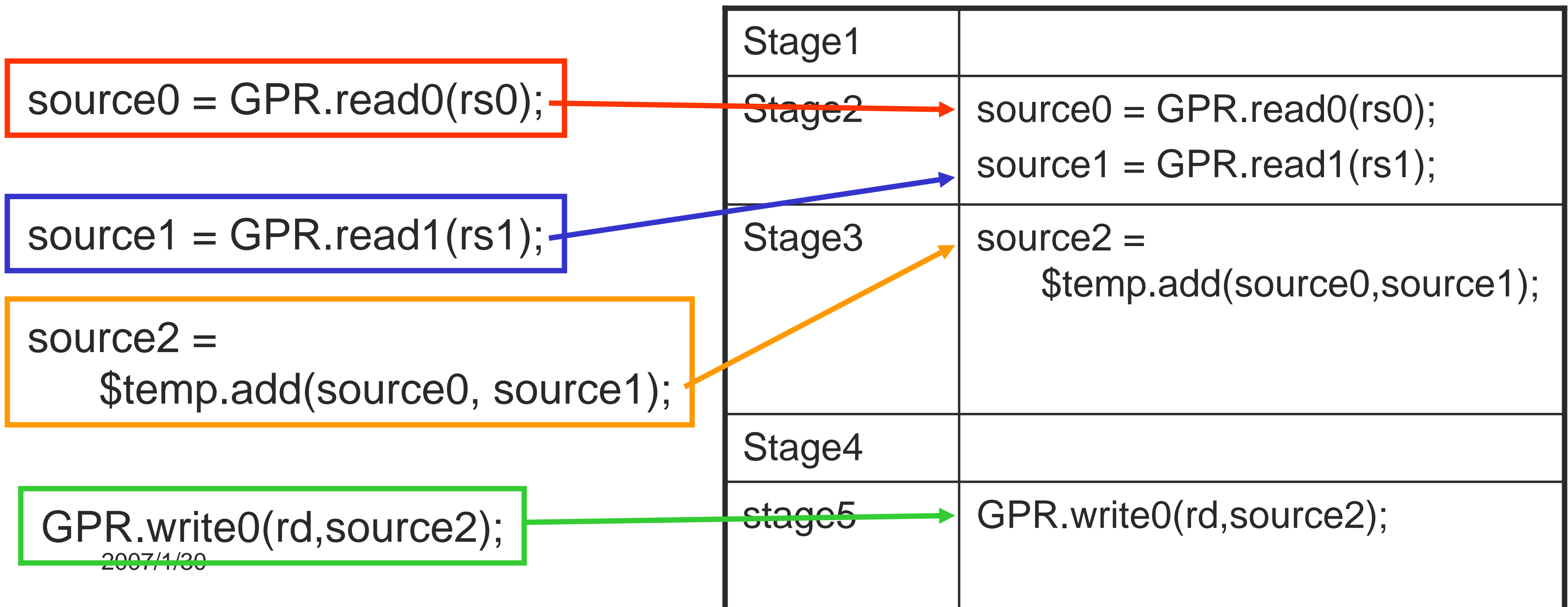
Attribute of Pipeline Stages

- Attribute
  - Instruction fetch, execution, memory read, etc…

Attribute of Pipeline Stages	
Stage Name	Attribute
Stage1	Instruction Fetch
Stage2	Operand Fetch & Sign-Extension
Stage3	Execution & Jump
Stage4	Memory Read & Memory Write
Stage5	Write Back

Example of Allocating to Pipeline Stages

- Attribute of each pipeline stage
  - Operand Fetch (read0, read1, ..)→ stage2
  - Execution (add, sub, …) → stage3
  - Memory Access (load, store, …)→ stage4
  - Write Back (write0, …) → stage5



Bind Functional Units

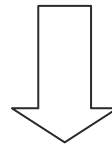
- Bind Functional Units to Allocated Micro-operation fragments
  1. List all functions in generated micro-operation fragments
  2. Decide a functional unit for each function
  3. Bind decided functional unit to each micro-operation fragment

Example of Binding Functional Units

instruction	operator	function		functional unit
ADD	+	add		ALU
SUB	-	sub		ALU
MUL	*	mul		MUL
LSFT	<<	lsft		SFT

```
source2 = $temp.add(source0,source1);
```

```
$temp ← ALU
```



```
source2 = ALU.add(source0,source1);
```

Outline

- Background
- Proposed Processor Generation Method
- Experiments
- Conclusion

Experiments

- Experimental setup
  - Design MIPS R3000 and DLX, and compare design time and design quality between conventional method and proposed method
    - Confirm the reduction of design time without degradation of design quality
  - Modify DLX by changing the pipeline architecture and implementing extra specific instructions, and compare design time and design quality
    - Confirm fast modification
- Conventional method
  - A designer manually describe micro-operation description
- Environment
  - Use ASIP Meister* as a processor design environment

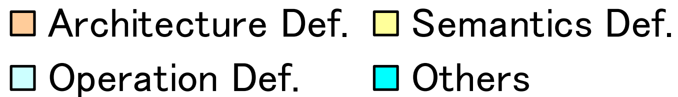
* M. Imai, ''ASIP Meister : A Configurable Processor Core Development System,'' Proc. of ICICT 2005, Cairo, Egypt, Dec. 2005.

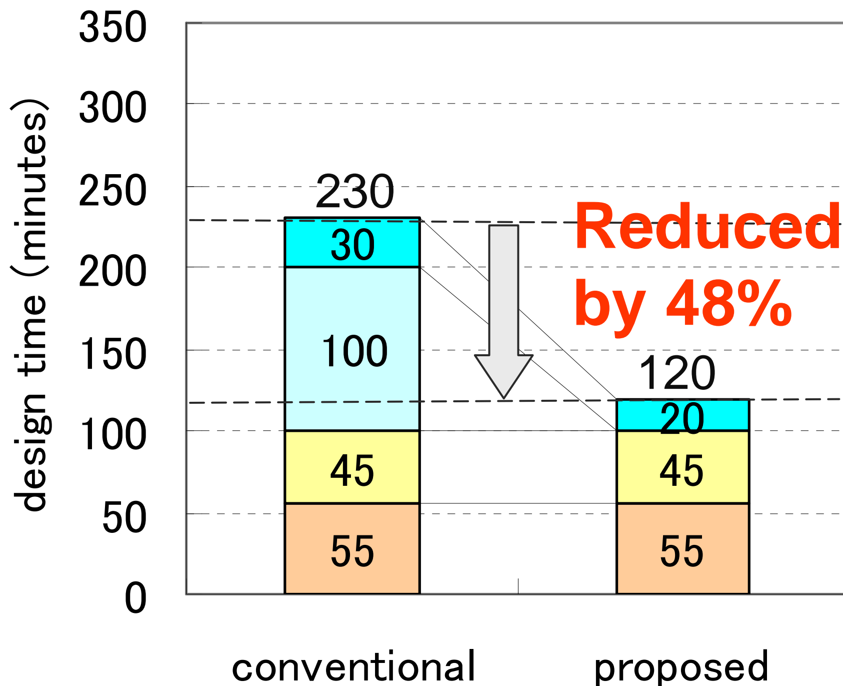
Implemented Processors

- MIPS R3000 subset
  - 5 pipeline stages
  - 42 instructions
    - 13 ALU operation ,4 mult/div, 11 immediate operation, 8 memory access , and 6 jump/branch
- DLX subset with 3 pipeline stages
  - 3 pipeline stages
  - 51 instructions
    - 16 ALU operation, 4 mult/div, 17 immediate operation, 8 memory access , and 6 jump/branch
- Modified DLX subset
  - 5 pipeline stages
  - 8 extra instructions
    - 2 multiply and accumulate (MAC), 4 memory access with post-increment/decrement, ABS(calculation of absolute), and CEX(compare and exchange)

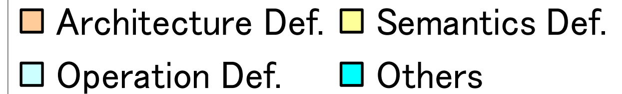
Comparison of Design Time

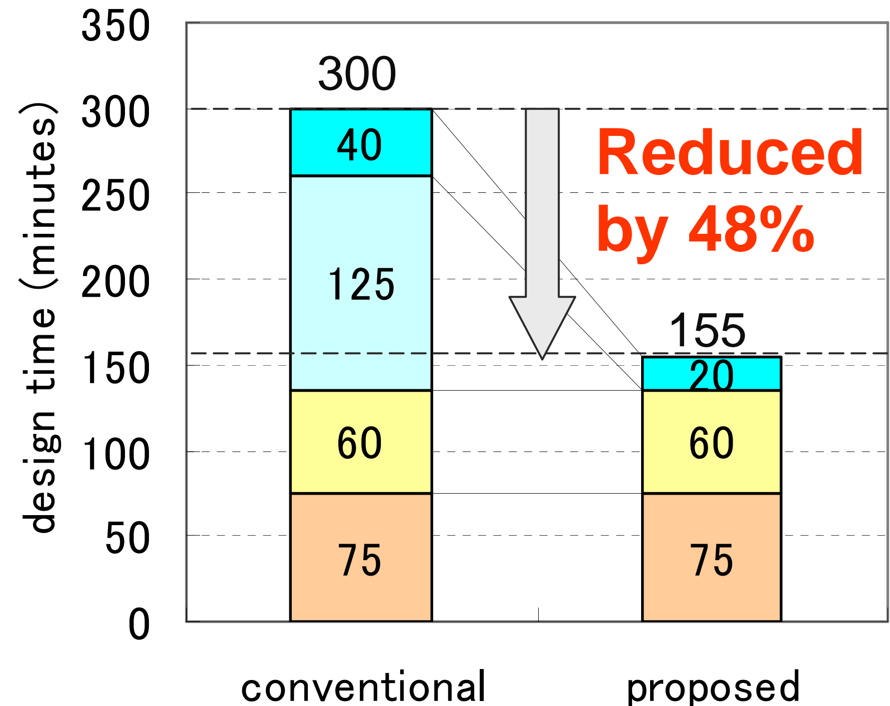
MIPS R3000 subset





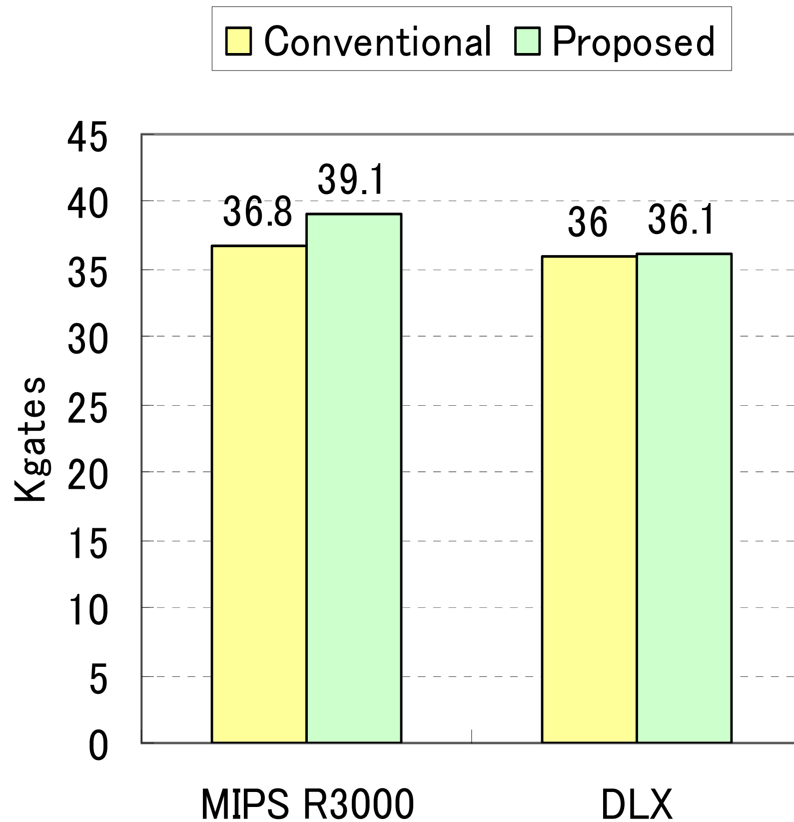
DLX subset

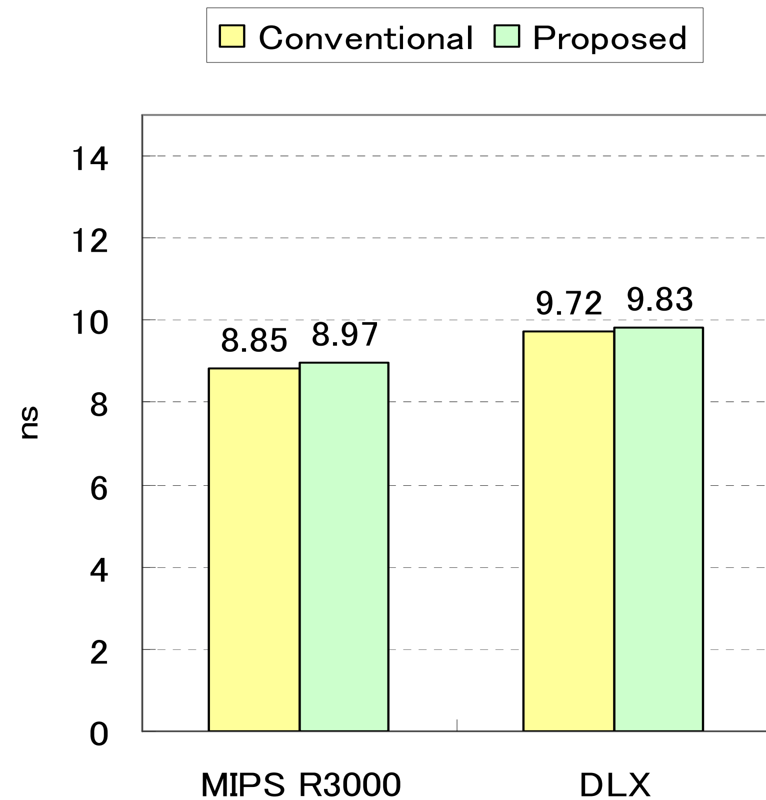




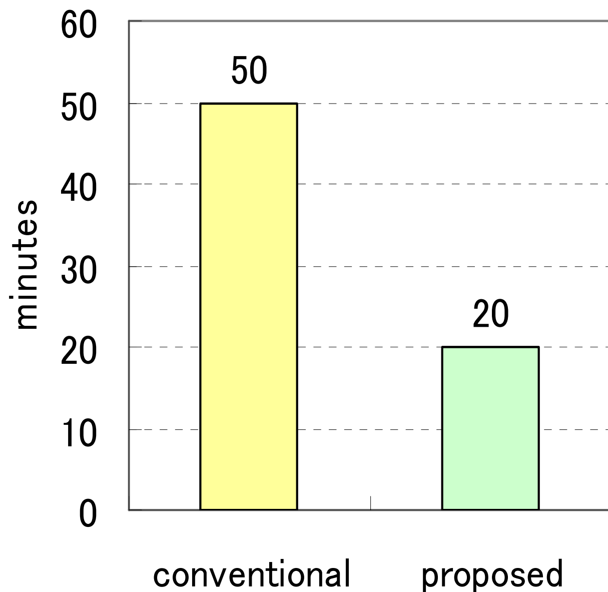
Comparison of Design Quality

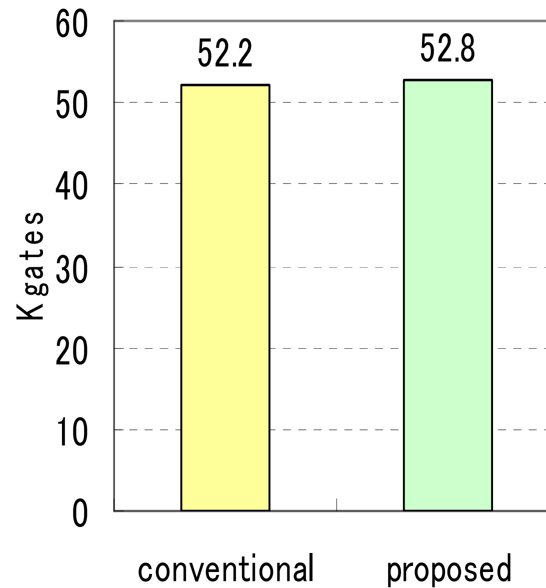
library: 0.18 μ m CMOS

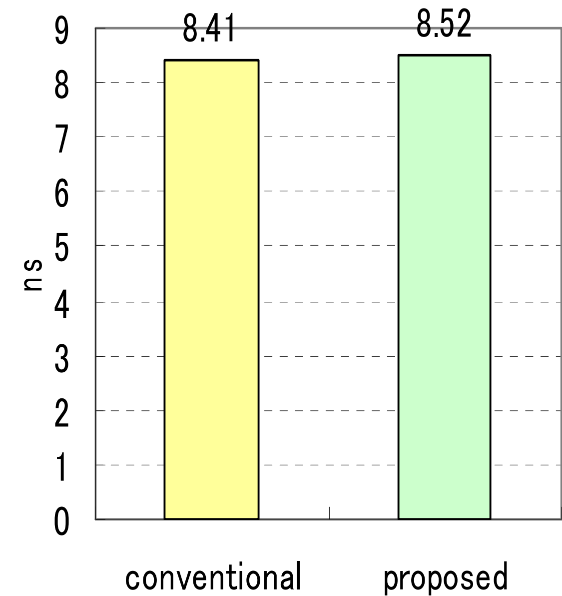




Modification of DLX processor







Outline

- Background
- Proposed Processor Generation Method
- Experiments
- Conclusion

Conclusion

- A method of generating micro-operation description from behavior description
  - Generate micro-operation fragments from abstract syntax trees
  - Automatically allocate fragments, base on attribute of each stage
- Quick design becomes possible
  - Reduce code by about 65 %
  - Reduce design time by about 50%
  - Hardly degrade
- Future Work
  - Optimizing combination of functional units
  - Generate micro-operation description of interrupts