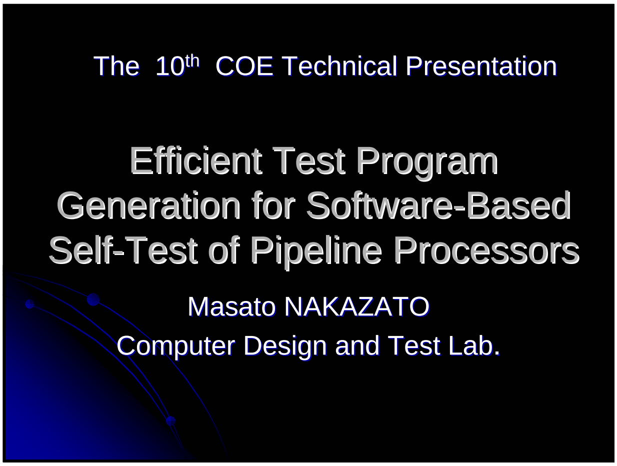
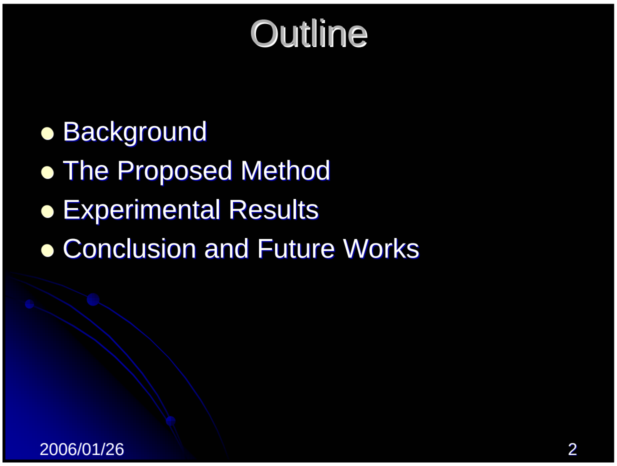


The 10th COE Technical Presentation

Efficient Test Program Generation for Software-Based Self-Test of Pipeline Processors

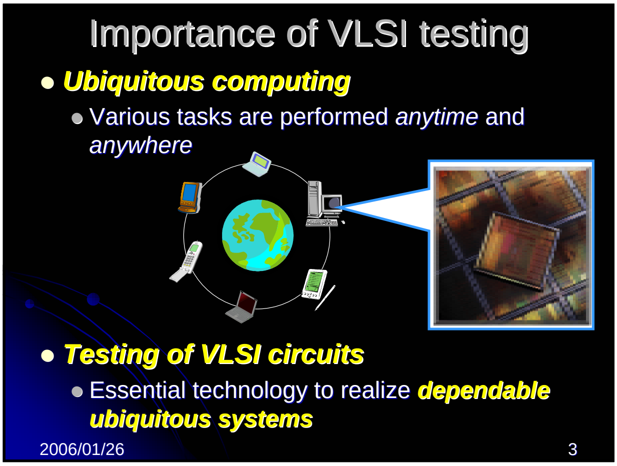
Masato NAKAZATO
Computer Design and Test Lab.



Outline

- Background
- The Proposed Method
- Experimental Results
- Conclusion and Future Works

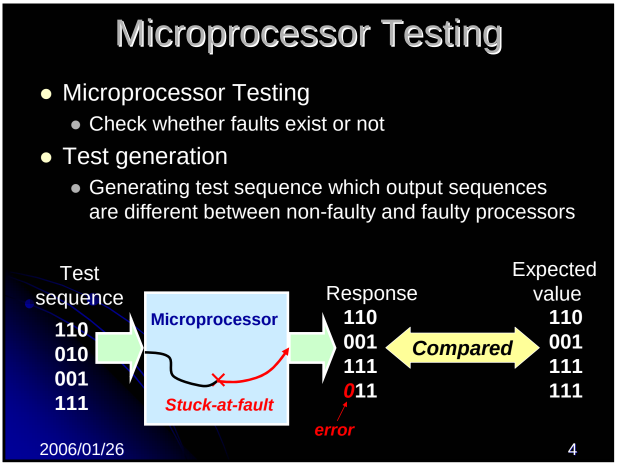
2006/01/26 2



Importance of VLSI testing

- ***Ubiquitous computing***
  - Various tasks are performed *anytime* and *anywhere*
- ***Testing of VLSI circuits***
  - Essential technology to realize ***dependable ubiquitous systems***

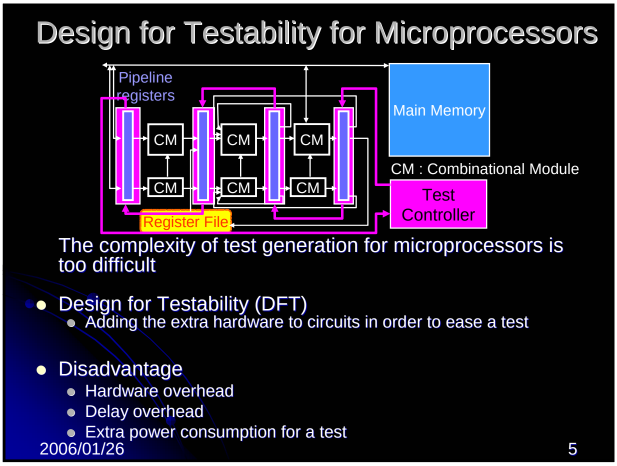
2006/01/26 3



Microprocessor Testing

- Microprocessor Testing
  - Check whether faults exist or not
- Test generation
  - Generating test sequence which output sequences are different between non-faulty and faulty processors

2006/01/26 4

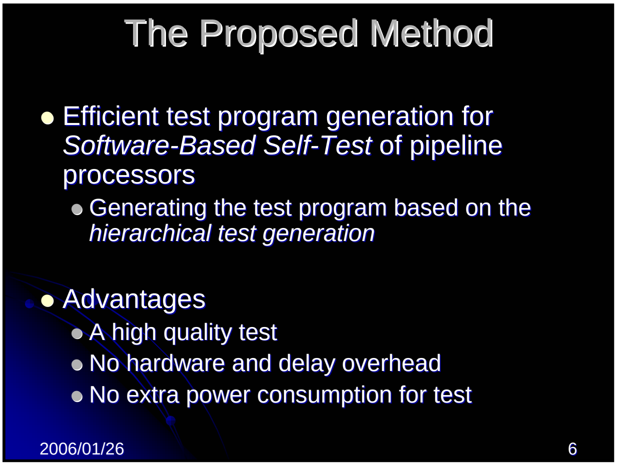


Design for Testability for Microprocessors

The complexity of test generation for microprocessors is too difficult

- Design for Testability (DFT)
  - Adding the extra hardware to circuits in order to ease a test
- Disadvantage
  - Hardware overhead
  - Delay overhead
  - Extra power consumption for a test

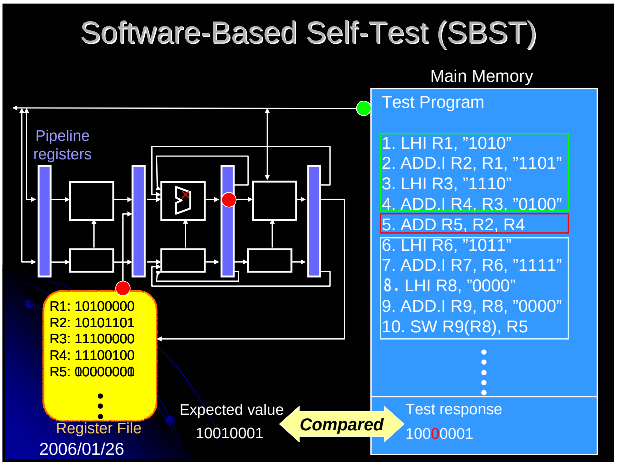
2006/01/26 5



The Proposed Method

- Efficient test program generation for *Software-Based Self-Test* of pipeline processors
  - Generating the test program based on the *hierarchical test generation*
- Advantages
  - A high quality test
  - No hardware and delay overhead
  - No extra power consumption for test

2006/01/26 6



Software-Based Self-Test (SBST)

Test Program

1. LHI R1, ”1010”
2. ADD.I R2, R1, ”1101”
3. LHI R3, ”1110”
4. ADD.I R4, R3, ”0100”
5. ADD R5, R2, R4
6. LHI R6, ”1011”
7. ADD.I R7, R6, ”1111”
8. LHI R8, ”0000”
9. ADD.I R9, R8, ”0000”
10. SW R9(R8), R5

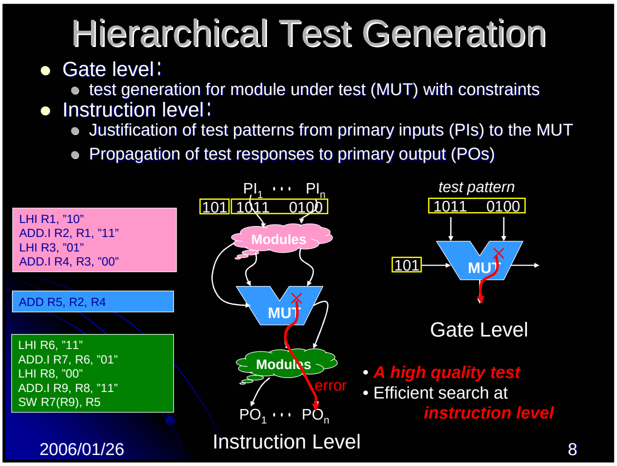
⋮

R1: 10100000
R2: 10101101
R3: 11100000
R4: 11100100
R5: 00000000
⋮

Expected value 10010001

Compared

Test response 10000001



Hierarchical Test Generation

- Gate level：
  - test generation for module under test (MUT) with constraints
- Instruction level：
  - Justification of test patterns from primary inputs (PIs) to the MUT
  - Propagation of test responses to primary output (POs)

LHI R1, ”10”
ADD.I R2, R1, ”11”
LHI R3, ”01”
ADD.I R4, R3, ”00”

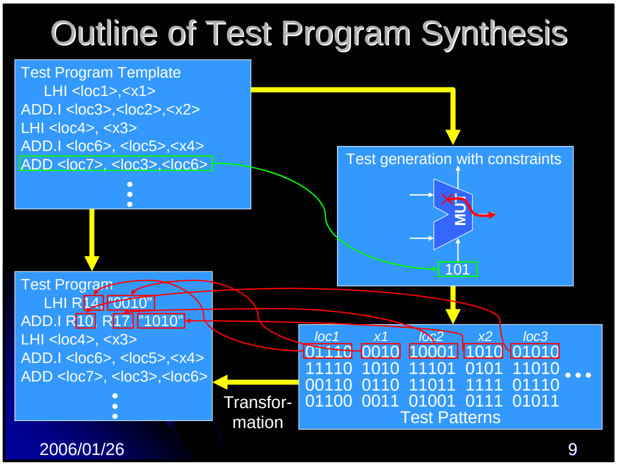
ADD R5, R2, R4

LHI R6, ”11”
ADD.I R7, R6, ”01”
LHI R8, ”00”
ADD.I R9, R8, ”11”
SW R7(R9), R5

Instruction Level

Gate Level

- ***A high quality test***
- Efficient search at ***instruction level***



Outline of Test Program Synthesis

Test Program Template
LHI <loc1>,<x1>
ADD.I <loc3>,<loc2>,<x2>
LHI <loc4>, <x3>
ADD.I <loc6>, <loc5>,<x4>
ADD <loc7>, <loc3>,<loc6>
⋮

Test generation with constraints

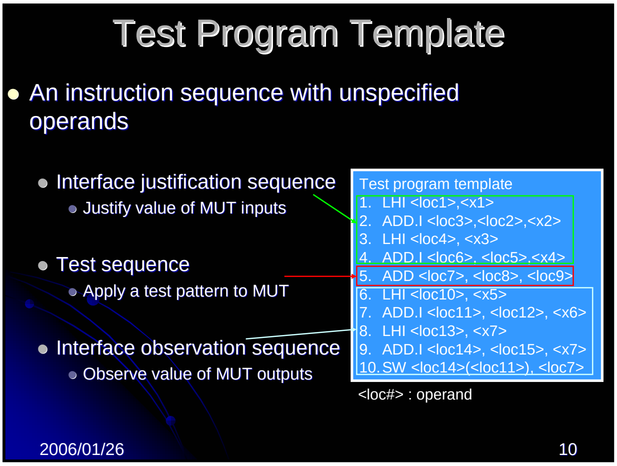
Test Program
LHI R14, ”0010”
ADD.I R10, R17, ”1010”
LHI <loc4>, <x3>
ADD.I <loc6>, <loc5>,<x4>
ADD <loc7>, <loc3>,<loc6>
⋮

Transfor-mation

loc1	x1	loc2	x2	loc3
01110	0010	10001	1010	01010
11110	1010	11101	0101	11010
00110	0110	11011	1111	01110
01100	0011	01001	0111	01011

• • •

Test Patterns

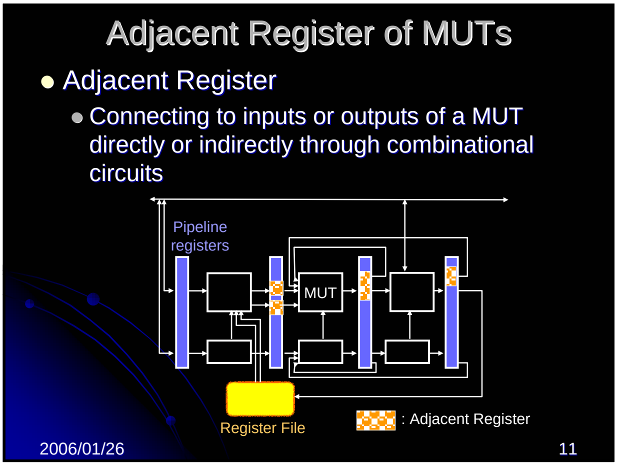


Test Program Template

- An instruction sequence with unspecified operands
  - Interface justification sequence
    - Justify value of MUT inputs
  - Test sequence
    - Apply a test pattern to MUT
  - Interface observation sequence
    - Observe value of MUT outputs

Test program template
1. LHI <loc1>,<x1>
2. ADD.I <loc3>,<loc2>,<x2>
3. LHI <loc4>, <x3>
4. ADD.I <loc6>, <loc5>,<x4>
5. ADD <loc7>, <loc8>, <loc9>
6. LHI <loc10>, <x5>
7. ADD.I <loc11>, <loc12>, <x6>
8. LHI <loc13>, <x7>
9. ADD.I <loc14>, <loc15>, <x7>
10. SW <loc14>(<loc11>), <loc7>

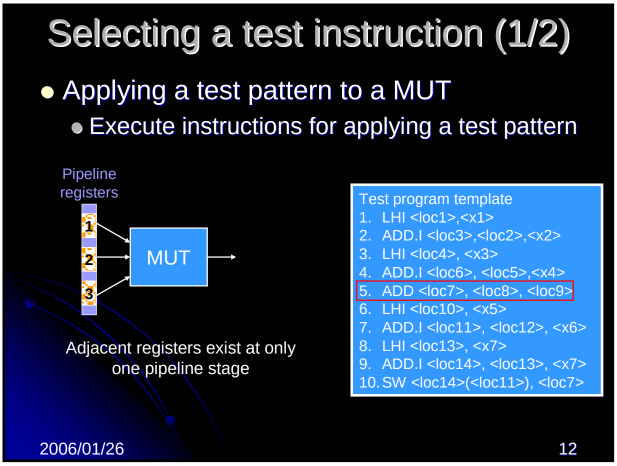
<loc#> : operand



Adjacent Register of MUTs

- Adjacent Register
  - Connecting to inputs or outputs of a MUT directly or indirectly through combinational circuits

▨ : Adjacent Register

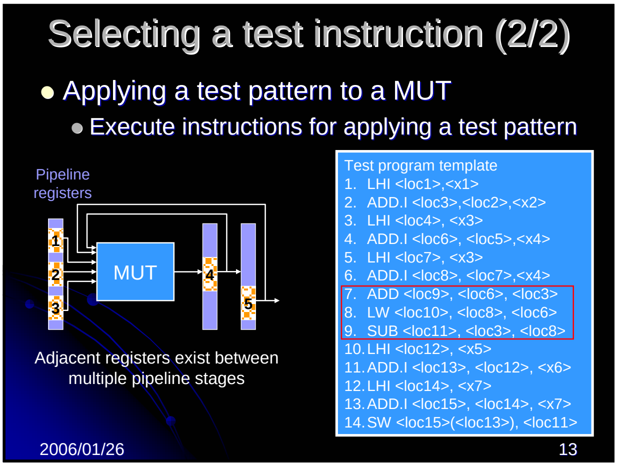


Selecting a test instruction (1/2)

- Applying a test pattern to a MUT
  - Execute instructions for applying a test pattern

Adjacent registers exist at only one pipeline stage

Test program template
1. LHI <loc1>,<x1>
2. ADD.I <loc3>,<loc2>,<x2>
3. LHI <loc4>, <x3>
4. ADD.I <loc6>, <loc5>,<x4>
5. ADD <loc7>, <loc8>, <loc9>
6. LHI <loc10>, <x5>
7. ADD.I <loc11>, <loc12>, <x6>
8. LHI <loc13>, <x7>
9. ADD.I <loc14>, <loc13>, <x7>
10. SW <loc14>(<loc11>), <loc7>

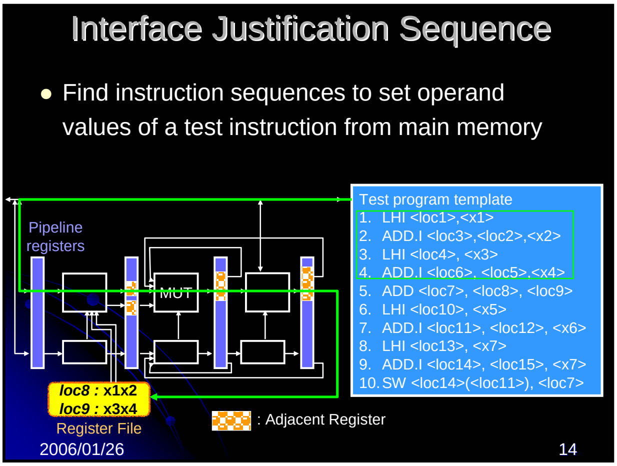


Selecting a test instruction (2/2)

- Applying a test pattern to a MUT
  - Execute instructions for applying a test pattern

Adjacent registers exist between multiple pipeline stages

Test program template
1. LHI <loc1>,<x1>
2. ADD.I <loc3>,<loc2>,<x2>
3. LHI <loc4>, <x3>
4. ADD.I <loc6>, <loc5>,<x4>
5. LHI <loc7>, <x3>
6. ADD.I <loc8>, <loc7>,<x4>
7. ADD <loc9>, <loc6>, <loc3>
8. LW <loc10>, <loc8>, <loc6>
9. SUB <loc11>, <loc3>, <loc8>
10. LHI <loc12>, <x5>
11. ADD.I <loc13>, <loc12>, <x6>
12. LHI <loc14>, <x7>
13. ADD.I <loc15>, <loc14>, <x7>
14. SW <loc15>(<loc13>), <loc11>

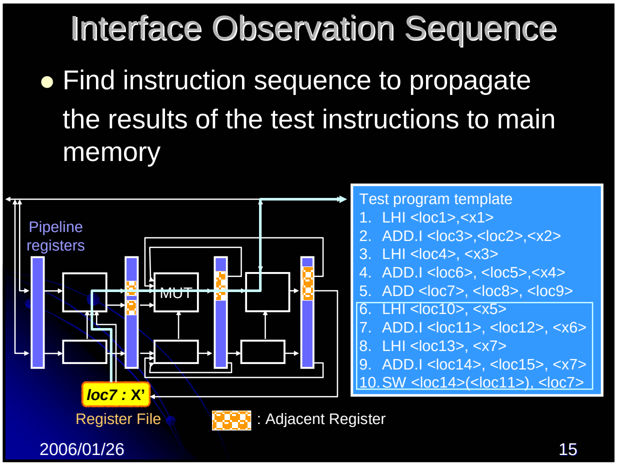


Interface Justification Sequence

- Find instruction sequences to set operand values of a test instruction from main memory

▩ : Adjacent Register

Test program template
1. LHI <loc1>,<x1>
2. ADD.I <loc3>,<loc2>,<x2>
3. LHI <loc4>, <x3>
4. ADD.I <loc6>, <loc5>,<x4>
5. ADD <loc7>, <loc8>, <loc9>
6. LHI <loc10>, <x5>
7. ADD.I <loc11>, <loc12>, <x6>
8. LHI <loc13>, <x7>
9. ADD.I <loc14>, <loc15>, <x7>
10. SW <loc14>(<loc11>), <loc7>

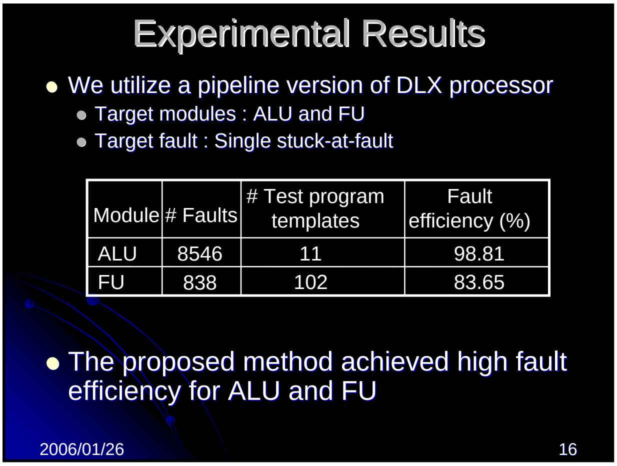


Interface Observation Sequence

- Find instruction sequence to propagate the results of the test instructions to main memory

▩ : Adjacent Register

Test program template
1. LHI <loc1>,<x1>
2. ADD.I <loc3>,<loc2>,<x2>
3. LHI <loc4>, <x3>
4. ADD.I <loc6>, <loc5>,<x4>
5. ADD <loc7>, <loc8>, <loc9>
6. LHI <loc10>, <x5>
7. ADD.I <loc11>, <loc12>, <x6>
8. LHI <loc13>, <x7>
9. ADD.I <loc14>, <loc15>, <x7>
10. SW <loc14>(<loc11>), <loc7>

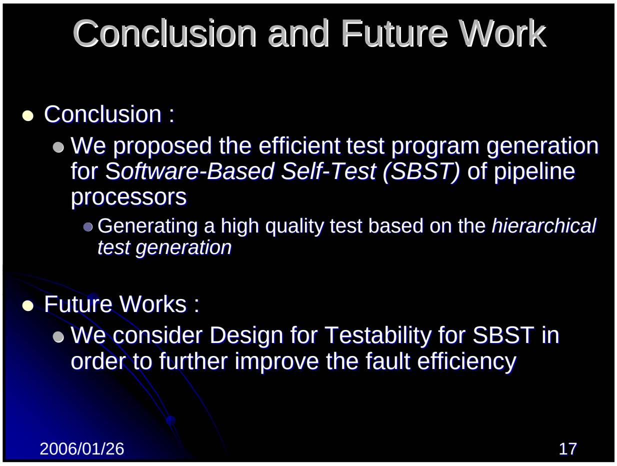


Experimental Results

- We utilize a pipeline version of DLX processor
  - Target modules : ALU and FU
  - Target fault : Single stuck-at-fault

Module	# Faults	# Test program templates	Fault efficiency (%)
ALU	8546	11	98.81
FU	838	102	83.65

- The proposed method achieved high fault efficiency for ALU and FU



Conclusion and Future Work

- Conclusion :
  - We proposed the efficient test program generation for *Software-Based Self-Test (SBST)* of pipeline processors
    - Generating a high quality test based on the *hierarchical test generation*
- Future Works :
  - We consider Design for Testability for SBST in order to further improve the fault efficiency