Self-Repairing Statically Scheduled Superscalar Processors

Mario Schölzel
Computer Engineering Group
at
Brandenburg University of Technology
Cottbus, Germany

Outline

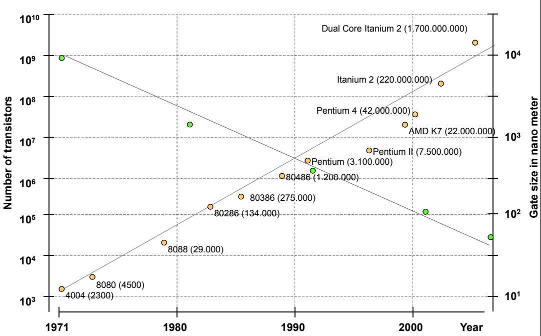
- Introduction
- Hardware-Based Self-Repair
- Software-Based Self-Repair
  - Rebinding
  - Rescheduling
- Hybrid Off-Line Self-Repair
- Lowering the Self-Repair Granularity
- Hybrid On-Line Self-Repair

Introduction

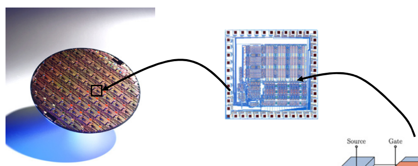
Moore's Law

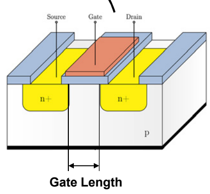
- Gordon Moore is co-founder of Intel (1968)
- Moore is widely known for "Moore's Law," in which in 1965 he predicted that the number of components the industry would be able to place on a computer chip would double every year.
- In 1975, he updated his prediction to once every two years. It has become the guiding principle for the semiconductor industry to deliver ever-more-powerful chips.

Evolution of Transistor Count



Nano-Technology in Computers





We talk about Nano-Technology if the gate length is less than 100 Nano Meters.

1 Nano Meter = 0,000001 mm is approximately the size of 4 neighbored metal atoms.

Nano-Structure Problems

- Transistors in integrated circuits become smaller and smaller.
- Smaller Transistors are
  - more susceptible to process variations,
  - more susceptible to voltage drops,
  - have higher stress in the field due to higher current density.
- Consequence:
  - Some transistors will be „out of specification":
    - after manufacturing,
    - very soon after manufacturing (early-life-failures),
    - after some years of heavy usage (wear-out).
  - **Faults** occur due to unreliable hardware, i.e. wrong outputs inside the system are produced.
- Goal:
  - Wrong outputs inside the system should not appear outside:
    - **What** kind of faults?
    - **How** can they be handled?
    - **When** and **where**?

Types of Faults (what)

Temporary Faults:
Appear only for a very short time period and disappears without any external intervention.

Transient Faults:
Triggered by an external event.

Intermittent Faults:
Triggered by a certain state of the system.

Permanent Faults:
Do not disappear without a repair (deterministic reproducible).

Fault Handling Techniques (how)

- Used Method may depend on the circuit type:
  - regular logic (e.g. memory),
  - irregular logic,
  - bus,
  - reconfigurable logic,
  - **programmable processors.**
- Handling of Temporary Faults by redundancy in:
  - Time,
  - Resources,
  - Data.
- Handling of **Permanent Faults** by
  - Hiding the fault by redundancy in resources or data,
  - Avoiding the usage of defect components (localization needed):
    - Reconfiguration of the hardware (suitable for any circuit type):
      - Statically,
      - Dynamically.
    - Adaptation of the software (only suitable for processors):
      - By hardware methods (=dynamically),
      - By **software-based** methods.

Scenarios for the Handling of Faults (when and where)

Off-Chip

On-Chip

Handling of permanent faults by methods of invasive configuration.

Transient faults: Handling by redundancy in time, resources, or data (Fault Tolerant Computing, well established)

Handling of transient and permanent faults by methods of fault tolerant computing.

Handling of transient faults by fault tolerant computing.

On-Line

Handle permanent faults by methods of self-repair.

Handling of permanent faults by self-repair.

Off-Line

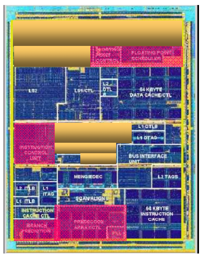
Off-Chip localization and repair by external equipment.

On-Chip localization and on-chip reconfiguration/repair.

Dynamic vs. Static Scheduled Processors

- More performance is obtained by parallelism.
- Parallelism can be discovered at instruction level:
  - Dynamically: Ctrl/DP: 40/60
    - Binary Compatibility
    - Performance
    - Compiler Support
  - Statically: Ctrl/DP: 1/99
    - Easy adaptable to a specific application.
    - Good ratio between performance/power/area consumption.
    - Real-time behavior is well predictable.
    - Compiler knows which resources are used by which operations.

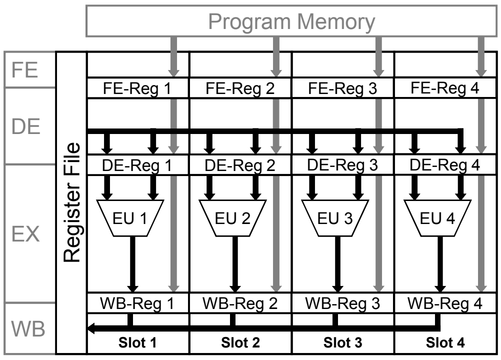
AMD K7 (32 Bit)



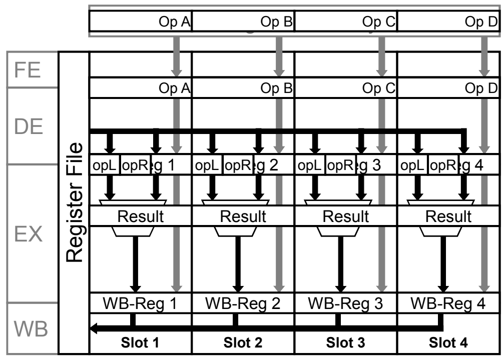
Source: http://www.chip-architect.org/news/AMD_family_pic.jpg

Hardware-Based Self-Repair

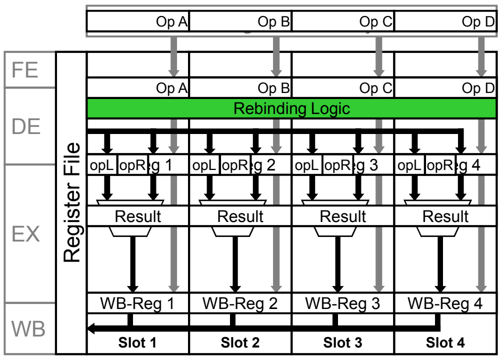
Processor Architecture



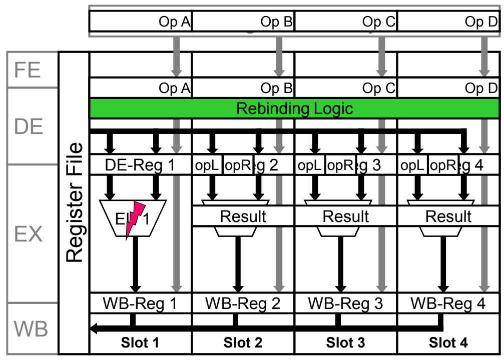
Program Execution



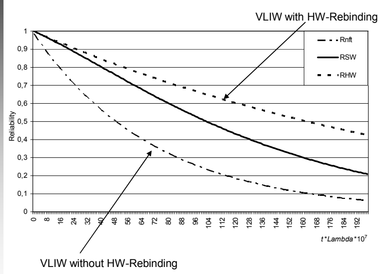
Program Execution (faultless)



Program Execution (faulty)



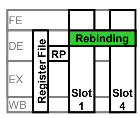
Reliability



System Type	Transistor Count
VLIW	139500
VLIW + HW Rebinding	139500 + 7200
Slot	25400
Single EU	7000
Ctrl	800

Position of the Rebinding Logic

Early Rebinding



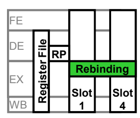
- Faults in the Read-Ports can be handled
- True-dependencies within the same instruction must be avoided:

add r1,r2 -> r3	add r1,r3 -> r4

Sequential execution creates True-Dependency

add r1,r2 -> r3
add r1,r3 -> r4

Late Rebinding



- Faults in the Read-Ports can not be handled.
- Reliability is reduced.

Summary – Hardware-Based Self-Repair

- Used in many approaches concerning VLIW architectures (late rebinding).
- Advantages:
  - Not visible for the software.
  - Handling of multiple Faults.
  - On-Line Handling of permanent faults, if permanent faults can be detected and localized on-line.
- Disadvantages:
  - Contradicts the VLIW concept.
  - Strong graceful performance degradation.
  - Administrative hardware overhead.
  - Rebinding of operations, each time the operation is executed.

Computer Engineering Group

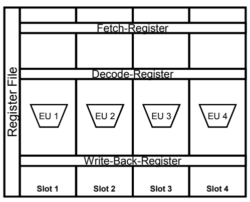
Software-Based Self-Repair

Computer Engineering Group
Introduction · HW Self-Repair · SW Self-Repair · Rebinding · Rescheduling · Hybrid Off-Line · Lower Granularity · Hybrid On-Line · Summary

Idea of the Software-Based Self-Repair

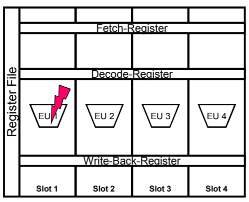
Faultless Data Path

add	sub	mul	add
add	nop	shl	cmp
add	mul	shl	mul
xor	nop	mul	add
jmp	nop	nop	inc



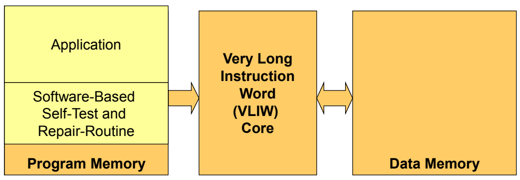
Faulty Data Path

add	sub	mul	add
add	nop	shl	cmp
add	mul	shl	mul
xor	nop	mul	add
jmp	nop	nop	inc



Computer Engineering Group
Introduction · HW Self-Repair · SW Self-Repair · Rebinding · Rescheduling · Hybrid Off-Line · Lower Granularity · Hybrid On-Line · Summary

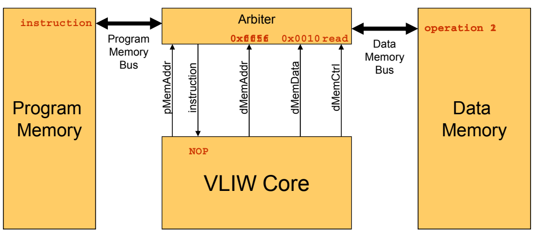
Basic System Architecture



- A permanent fault is detected and repaired during the startup of the system (on-chip but off-line):
  - At Start-up a diagnostic self-test is done by a self-test-routine.
  - A Repair-Routine is executed in order to reorder the operations of the application.
- Problem: Access to the program memory.

Computer Engineering Group
Introduction · HW Self-Repair · SW Self-Repair · Rebinding · Rescheduling · Hybrid Off-Line · Lower Granularity · Hybrid On-Line · Summary

Infrastructure for Moving Instructions



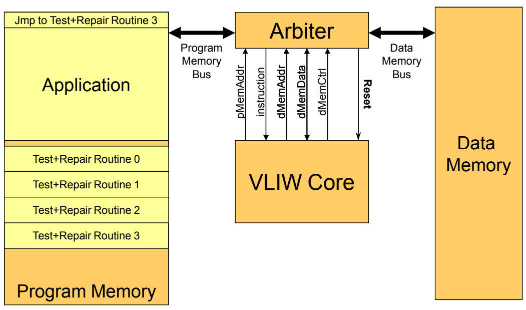
Arbiter makes up about 3% of the transistor count of the VLIW-core.

```
Ldc 0xFFFF -> R0
Ldc 0x0056 -> R2
Ldc 0x0010 -> R3
Load [R0]  -> R1     // Initialize Read-Mode in arbiter
Store R3   -> [R2]   // Read instruction from 0x10 and
                     // save it in dmem to address 0x56
Nop                  // Keep the VLIW-core synchronous
Nop                  // with the program in the program memory
```


Computer Engineering Group
Introduction · HW Self-Repair · SW Self-Repair · Rebinding · Rescheduling · Hybrid Off-Line · Lower Granularity · Hybrid On-Line · Summary

Setting Up the System

Test+Repair Routine *i* uses only slot *i*.



Software-Based Self-Repair Approaches

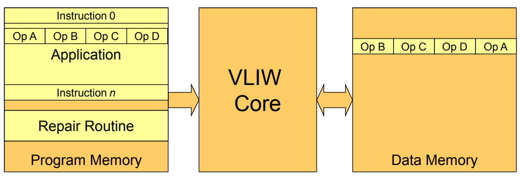
- Rebinding:
  - Operations are rebound within the same instruction.
  - Dependencies are not violated.
  - Restricted to single-cycle operations.
- Rescheduling:
  - Operations are rescheduled within a whole basic block.
  - Dependencies must be taken into account.
  - Multi-cycle Operations can be handled.

Software-Based Self-Repair

Rebinding

Organization of the Self-Repair using Rebinding

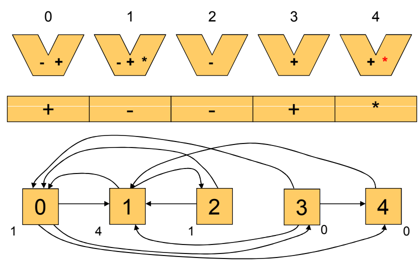
- Save the fault state of the processor; e.g. faulty operators and execution units in a fault memory after the software-based self-test.
- Move each instruction of the application into the data memory.
- Compute a permutation there.
- Move the permutated instruction back into the program memory.



How to Find a Permutation

- Model the problem as a directed graph (*N*,*E*):
  - Nodes represent FUs.
  - There is an edge (*u*,*v*) (with label *op*), iff
    - *u* executes an operation of type *op* and
    - *v* can execute an operation of type *op* and
    - the operator of type *op* is not faulty in v.
- *G* is the set of FUs that consists of the faulty FU and all FU that execute a NOP in the current instruction.
- Goal: Find in the graph any path from the faulty FU *a* to a FU in set *G*.
- I.e.: Compute the transitive closure E^+ and check whether for any $b \in G$: $(a,b) \in E^+$ or not.
- Shift all operations along the path by one edge.

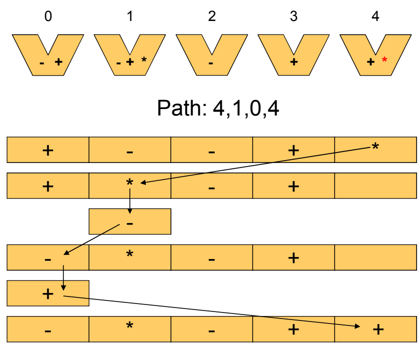
Computing the Permutation



G = {4}

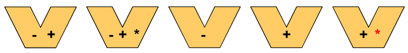
Path: 4,1,0,4

Computing the Permutation



Problem of Computing a Permutation

Functional Units in the Data Path



Instruction 1 to be executed

+	-	-	+	*

Executed instructions

-	*	-	+	+

Permutation exists.

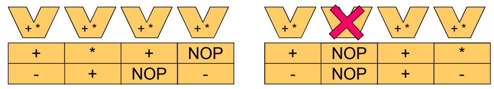
Instruction 2 to be executed

-	-	-	+	*

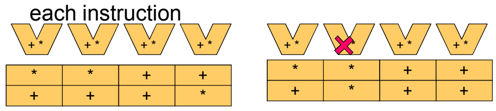
No permutation exists.

Fault-Tolerant Scheduling

- Handle *m*-execution unit faults:
  - schedule *m* NOPs in each instruction



- Handle *n*-operator faults:
  - leave at least *n* operators of each type unused in each instruction



Fault Tolerant Scheduling

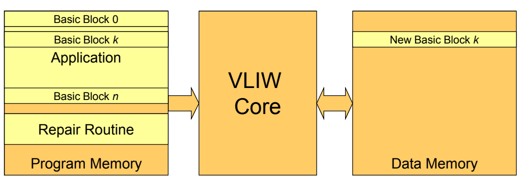
```
(N,E) .. DAG of the basic block
 m    .. number of tolerable execution unit faults
 n    .. number of tolerable operator faults

i = 1 // current instruction
while not all nodes in N are scheduled do
  Compute Ready-List r
  for each node in r in priority order do
    schedule operation r into instruction i if
    -there is a free resource for r and
    -at least m EUs are unused after scheduling r and
    -at least n EUs that contain an operator for r
     are unused after scheduling r
  od
  i = i +1
od
```


Software-Based Self-Repair

Rescheduling

Organization of the Self-Repair using Rescheduling

- Save the fault state of the processor; e.g. faulty operators and execution units in a fault memory after the software-based self-test.
- Move the schedule of each basic block of the application into the data memory.
- Compute a new schedule there.
- Move the new schedule back into the program memory.

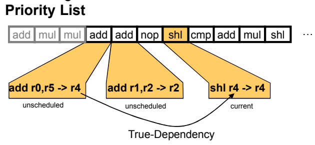


Rescheduling Algorithm

1. Starting with the **Original Basic Block**

add	mul	mul	add
add	nop	shl	cmp
add	mul	shl	mul
xor	nop	mul	add
jmp	nop	nop	inc

2. Building a **Priority List**



3. Building the **New Basic Block**

-	mul	mul	add
-	4	5	6
-	7	8	9
-	10	11	12

Defined Registers

r0	r1	r2	r3	r4	r5	r6	r7
0	0	1	0	1	0	0	0

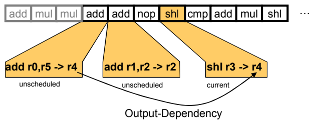
Check if one of the *used* registers of the **current** operation is *defined* by a preceding **unscheduled** operation.

Rescheduling Algorithm

1. Starting with the **Original Basic Block**

add	mul	mul	add
add	nop	shl	cmp
add	mul	shl	mul
xor	nop	mul	add
jmp	nop	nop	inc

2. Building a **Priority List**



3. Building the **New Basic Block**

-	mul	mul	add
-	4	5	6
-	7	8	9
-	10	11	12

Defined Registers

r0	r1	r2	r3	r4	r5	r6	r7
0	0	1	0	1	0	0	0

Check if one of the *defined* registers of the **current** operation is *defined* by a preceding **unscheduled** operation.

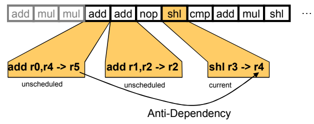
b-tu Brandenburg University of Technology Cottbus
Computer Engineering Group
Introduction
HW Self-Repair
SW Self-Repair
Rebinding
Rescheduling
Hybrid Off-Line
Lower Granularity
Hybrid On-Line
Summary

Rescheduling Algorithm

1. Starting with the **Original Basic Block**

add	mul	mul	add
add	nop	shl	cmp
add	mul	shl	mul
xor	nop	mul	add
jmp	nop	nop	inc

2. Building a **Priority List**



3. Building the **New Basic Block**

-	mul	mul	add
-	4	5	6
-	7	8	9
-	10	11	12

Used Registers

r0	r1	r2	r3	r4	r5	r6	r7
1	1	1	0	1	0	0	0

Check if one of the *defined* registers of the **current** operation is *used* by a preceding **unscheduled** operation.

b-tu Brandenburg University of Technology Cottbus
Computer Engineering Group
Introduction
HW Self-Repair
SW Self-Repair
Rebinding
Rescheduling
Hybrid Off-Line
Lower Granularity
Hybrid On-Line
Summary

Rescheduling Algorithm

```
while not all operations are scheduled do
  Create a new empty instruction currInstr
  for slot := 1 to 4 do
    if EU[slot] = 0 then
      for reg := 0 to 64 do
        Def[reg] := 0; Use[reg] := 0;
      od
      for each op in priority order do
        if op.time = currInstr then
          Def[op.dst] := 1;
        fi

        if op.state != scheduled then
          if CanBeScheduled(slot,op) &&
             Def[op.Src1]=0 && Def[op.Src2]=0 &&
             Use[op.Dst]=0  && Def[op.Dst]=0
          then
              EU[i] := lat(op);
              instr[slot] := op;
              op.time := currInstr;
              lastDef[op.dst] := currInstr;
              op.state := scheduled;
              break;
          else
            Use[op.Src1] := 1; Use[op.Src2] := 1;
            Def[op.Dst] := 1
          fi
        fi
      od
    fi
  od
  if EU[1] > 0 then EU[1]--;   if EU[2] > 0 then EU[2]--;
  if EU[3] > 0 then EU[3]--;   if EU[4] > 0 then EU[4]--;
od
```

-	mul	mul	add
-	4	5	6

Process each slot of the new instruction.

Handle multi-cycle operations

-	mul	mul	add
-			6

Search an operation for the current slot of the current instruction.

Process only unscheduled operations from the priority list.

Check for available resources.

Check for True-Dependencies.

Check for Anti- and Output-Dependencies.

Schedule the unscheduled operation.

Operation remain unscheduled. Update Use-/Def-information.

b-tu Brandenburg University of Technology Cottbus
Computer Engineering Group
Introduction
HW Self-Repair
SW Self-Repair
Rebinding
Rescheduling
Hybrid Off-Line
Lower Granularity
Hybrid On-Line
Summary

Repair Time

Runtime of the SW-based Self-Repair routines in clock cycles

	SW-Rebinding		Scoreboarding	
Benchmark	**1-operator fault**	**1-execution unit fault**	**1-operator fault**	**1-execution unit fault**
ARF	2103	1950	9246	15592
DIF	3246	2730	9607	26872
FFT	3190	2535	11145	23464
EWF	2956	3120	45582	47862
DIT	3650	3510	19643	36616
LEE	4240	3315	15929	37651

Estimated static runtime overhead in seconds for three different application scenarios

Instructions in application	*static overhead for SW-rebinding*	*static overhead for Scoreboarding*
16000	0,07s	0,66s
64000	0,31s	2,64s
256000	1,23s	10,57s

Assumptions:

- 10% of the code is time critical.
- Average size of a time critical basic block is 14 instructions.
- Average size of a non-time critical basic block is 7 instructions.
- SW-rebinding on average: 240 clock cycles per instruction.
- Rescheduling on average: 35200 clock cycles for time-critical and 14100 clock cycles for non-time critical basic blocks.
- Clock rate of about 50 MHz.

b-tu Brandenburg University of Technology Cottbus
Computer Engineering Group
Introduction
HW Self-Repair
SW Self-Repair
Rebinding
Rescheduling
Hybrid Off-Line
Lower Granularity
Hybrid On-Line
Summary

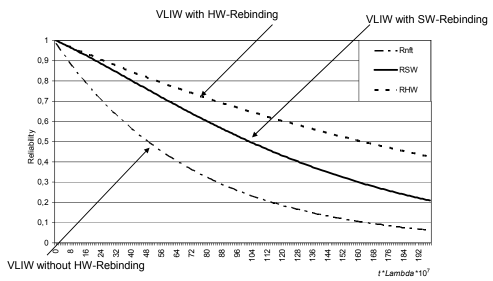
Runtime of the Application

Worst-case execution time overhead for several operator- and EU-fault classes

class	1-operator fault			1-execution unit fault			2-operator fault			2-execution unit fault		
Method	**HR**	**Sc**	**SR**	**HR**	**Sc**	**SR**	**HR**	**Sc**	**SR**	**HR**	**Sc**	**SR**
ARF	63%	25%	25%	100%	25%	25%	125%	25%	38%	200%	75%	88%
DIF	82%	27%	37%	100%	37%	64%	136%	64%	91%	200%	118%	91%
FFT	70%	20%	10%	100%	40%	30%	130%	90%	50%	200%	130%	80%
EWF	93%	7%	0%	93%	43%	14%	150%	36%	21%	171%	36%	36%
DIT	57%	7%	7%	100%	21%	29%	107%	36%	50%	200%	71%	79%
LEE	79%	43%	0%	100%	43%	21%	157%	43%	29%	200%	86%	79%
Average:	**74%**	**22%**	**13%**	**99%**	**35%**	**31%**	**134%**	**49%**	**47%**	**195%**	**86%**	**76%**

b-tu Brandenburg University of Technology Cottbus
Computer Engineering Group
Introduction
HW Self-Repair
SW Self-Repair
Rebinding
Rescheduling
Hybrid Off-Line
Lower Granularity
Hybrid On-Line
Summary

Reliability



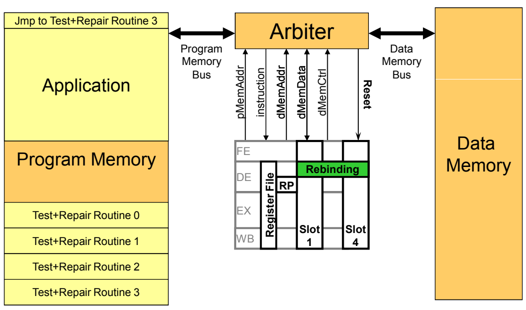
Summary – Software-Based Self-Repair

- Pure software-based self-repair scheme:
  - No extensions within the core itself.
  - The repair routine is executed on the faulty core itself.
  - Administration of spare resources by software.
- Cores that were not designed to be fault tolerant can be made fault tolerant.
- Scaling the reliability in software by specifying the number of tolerable faults and then compiling the application.
- Static runtime overhead in the range from a fraction of a second up to a few seconds.

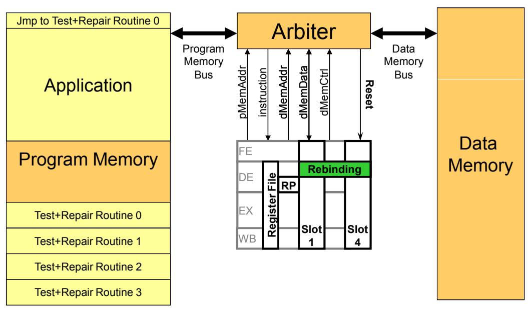
Hybrid Self-Repair

Off-Line

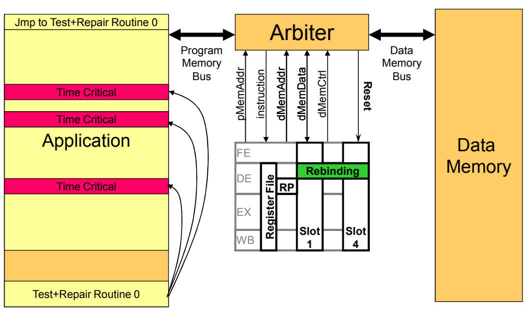
Architecture for hybrid method



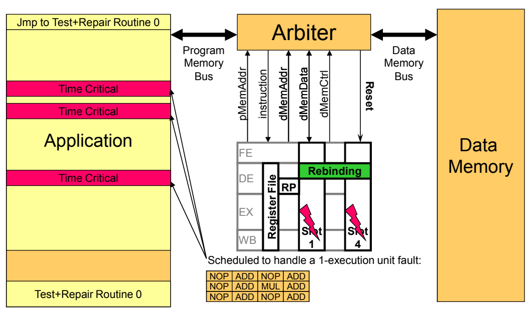
Simplified Start-up



Partial Repair



Extended Reliability



Summary – Hybrid Methods

- Simplified Start-Up:
  - One self-repair routine is sufficient.
- Partial repair reduces repair-time:

Instructions in application	static overhead for SW-rebinding	static overhead for Scoreboarding	static overhead for HSW	static overhead for HSC
16000	0,07s	0,66s	0,007s	0,08s
64000	0,31s	2,64s	0,03s	0,32s
256000	1,23s	10,57s	0,12s	1,29s

- Extended Reliability:
  - Handling of fault situations, which can not be handled by pure SW-based methods, is possible.
  - Certain amount of faults can be handled without graceful degradation by software-based methods; after that graceful degradation.
- Hybrid methods provide a good compromise between repair-time and performance degradation.

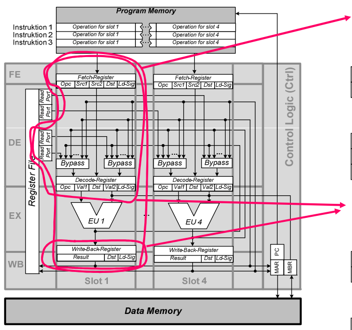
Summary – Off-Line Self-Repair

Method	Dynamic runtime overhead	Static runtime overhead	Reliability	HW-overhead	Allowed operation latency
HR	high	**none**	**high**	low	1
SR	**low**	medium	scalable	**none**	1
SC	**low**	high	scalable	**none**	**arbitrary**
HSR	low to medium	**low**	**high**	low	1
HSC	low to medium	medium	**high**	low	1

Message:
Move as much of the administrative work as possible into software!

Lowering the Self-Repair Granularity

Fault-Handling at Slot/Operator-Level



Slot-Level:
All components of a slot are critical.

Op1	Op2	Op3	Op4
Op5	NOP	Op6	Op7

self-repair

NOP	Op2	Op3	Op4
NOP	Op1	Op5	NOP
NOP	NOP	Op6	Op7

Operator-Level:
All components of a slot (except EUs) are critical.

Op1	Op2	Op3	Op4
Op5	NOP	Op6	Op7

self-repair

Op1	Op2	Op3	Op4
NOP	Op5	Op6	Op7

Rescheduling Algorithm

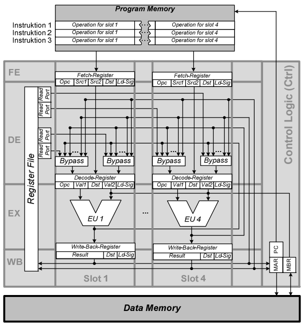
```
while not all operations are scheduled do
  Create a new empty instruction currInstr
  for slot := 1 to 4 do
    if EU[slot] = 0 then
      for reg := 0 to 64 do
        Def[reg] := 0; Use[reg] := 0;
      od
      for each op in priority order do
        if op.time = currInstr then
          Def[op.dst] := 1;
        fi

        if op.state != scheduled then
          if CanBeScheduled(slot,op) &&
             Def[op.Src1]=0 && Def[op.Src2]=0 &&
             Use[op.Dst]=0  && Def[op.Dst]=0
          then
              EU[i] := lat(op); instr[slot] := op;
              op.time := currInstr;
              lastDef[op.dst] := currInstr;
              op.state := scheduled;
              break;
          else
            Use[op.Src1] := 1; Use[op.Src2] := 1;
            Def[op.Dst] := 1
          fi
        fi
      od
    fi
  od
  if EU[1] > 0 then EU[1]--;   if EU[2] > 0 then EU[2]--;
  if EU[3] > 0 then EU[3]--;   if EU[4] > 0 then EU[4]--;
od
```


Modeling Faults for Slot- and Operator-Level

- Fault state of EU k is described by $F_k(i)$.
- Thereby:
  - $F_k(i) = 0$, iff operator i can be used,
  - $F_k(i) = 1$, iff operator i can not be used.
- If for all i holds $F_k(i) = 0$, then the whole slot k can not be used anymore.
- $CanBeScheduled(slot,op) := F_{slot}(op)$.

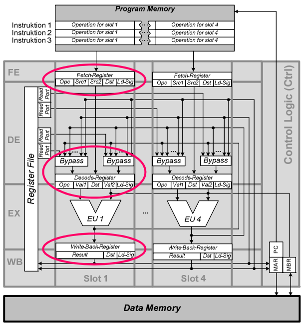
Introduction
HW Self-Repair
SW Self-Repair
Rebinding
Rescheduling
Hybrid Off-Line
Lower Granularity
Hybrid On-Line
Summary

Size of the Components



Component	Transistors	Instances
Ctrl	800	1
FE-Reg	580	4
DE-Reg	1300	4
Bypass	1400	8
EU	7000	4
WB-Reg	500	4
RP	6900	8
REG	890	64

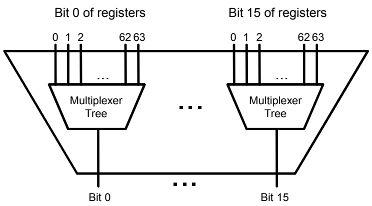
Fault-Handling at Read-Port-Level



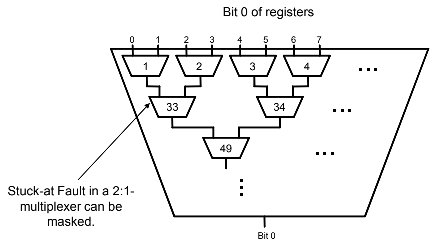
Read-Port-Level:
Pipeline registers and bypasses are critical.

Effect of Self-Repair:
If there is a fault in a read port, then only a certain subset of the registers in the register file can be accessed through this read port.

Structure of a Read Port



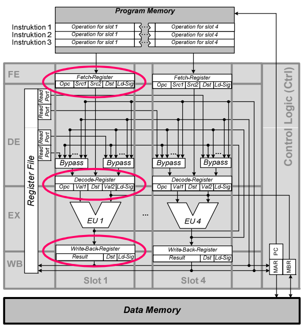
Structure of a Multiplexer Tree



Adapting the Self-Repair Algorithm

- Fault-state of the read port in slot *k*:
  - left: `rpState[2*k]`
  - right: `rpState[2*k+1]`
- Thereby:
  - `rpState[i][r] = 0`, if register *r* can be accessed correctly via read port *i*,
  - `rpState[i][r] = 1`, if register *r* can not be accessed correctly via read port *i*.
- `CanBeScheduled(slot,op) :=` $\mathtt{F}_{slot}(op)$ `&& rpState[2*slot][op.src1] && rpState[2*slot+1][op.src2]`

Fault-Handling at Bypass-Level



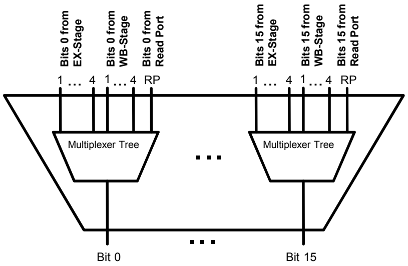
Bypass-Level:
Pipeline register are critical.

Effect of Self-Repair:
Do not use forwarding from EX-stage into DE-stage, if this network is faulty.

Do not use forwarding from WB-stage into DE-stage, if this network is faulty.

If a read port is faulty, then feed the slot with data via the bypass.

Structure of the Bypass

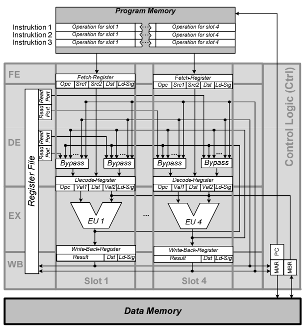


Adapting the Self-Repair Algorithm

- Fault-state of the bypasses in slot *k*:
  - left: `bypassState[2*k]`
  - rigth: `bypassState[2*k+1]`
- Thereby:
  - `bypassState[i] & 1 = True`, iff forwarding from the EX-stage into the DE-stage can be used.
  - `bypassState[i] & 2 = True`, iff forwarding from the WB-stage into the DE-stage can be used.
- `currInstr – lastDef[r]` ist the number of past clock cycles since the latest definition of register *r*.
- `CanBeScheduled(slot,op) :=`

```
F_slot(op) &&
(currInstr - lastDef[op.src1] = 1 && (bypassState[2*slot] & 1) ||
 currInstr - lastDef[op.src1] = 2 && (bypassState[2*slot] & 2) ||
 currInstr - lastDef[op.src1] > 2 && (rpState[2*slot][op.src1]) &&

(currInstr - lastDef[op.src2] = 1 && (bypassState[2*slot+1] & 1) ||
 currInstr - lastDef[op.src2] = 2 && (bypassState[2*slot+1] & 2) ||
 currInstr - lastDef[op.src2] > 2 && (rpState[2*slot+1][op.src2])
```

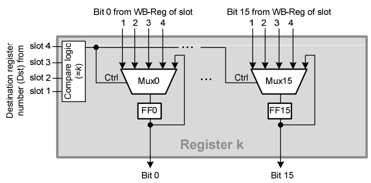

Fault-Handling at Register-Level



Register-Level

The usage of certain registers is avoided by register renaming.

Structure of a Register



Adapting the Rescheduling-Algorithm

```
...
if CanBeScheduled(slot,op) &&
   Def[op.Src1]=0  && Def[op.Src2]=0  &&
   Use[op.Dst]=0   && Def[op.Dst]=0
then
   EU[i] := lat(op);
   instr[slot] := rename(op);
   op.time := currInstr;
   lastDef[op.dst] := currInstr;
   op.state := scheduled;
   break;
else
   Use[op.Src1] := 1;
   Use[op.Src2] := 1;
   Def[op.Dst] := 1
fi
...
```


Results - Runtime

- Four injected Faults:
  - Adder in slot 2,
  - Left bypass in slot 1 (Forwarding from EX-stage defect),
  - Registers 1 to 4 und 5 to 6 can not be accessed via the left read port in slot 4.

Benchmark	Level of the Repair: Slots	Slots & Operators	Slots & Operators & Bypasses	Slots & Operators & Bypasses & Read ports
ARF (8)	350%	200%	125%	113%
FFT (10)	380%	270%	160%	120%
EWF (14)	243%	193%	129%	107%

Results – Fault Injection

- Fault Injection in 1.000.000 VLIWs.
- 10 injected faults in each VLIW system.
- Surviving VLIWs if self-repair takes place at slot level: 0,3%.
- Surviving VLIWs if self-repair takes place at slot- and register level: 51%.
- Surviving systems if self-repair takes place at all levels of granularity: 82%.

b-tu Brandenburg University of Technology Cottbus
Computer Engineering Group

Summary – Lowering the Granularity

- Fine-grained self-repair method for statically scheduled data paths.
- Refining the granularity does not require additional hardware (already existing switches in the system are employed).
- Multiple Faults did not reduce dramatically the run time of the application.
- Number of surviving systems is increased very much compared to coarse-grained approaches.
- Demands for the granularity for a diagnostic self-test are defined.

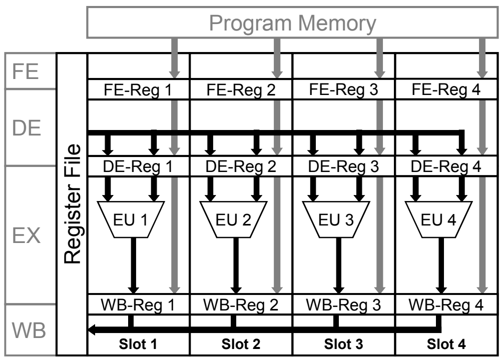
b-tu Brandenburg University of Technology Cottbus
Computer Engineering Group

Hybrid Self-Repair

On-Line

b-tu Brandenburg University of Technology Cottbus
Computer Engineering Group

Non-Fault Tolerant Architecture



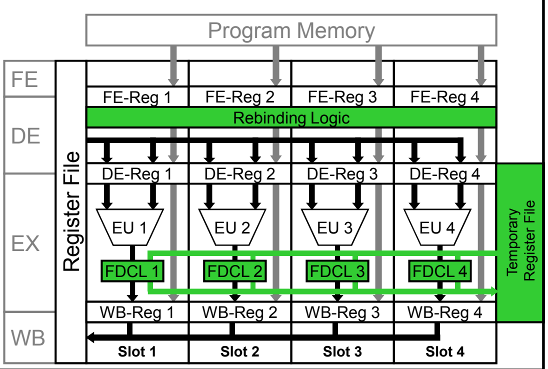
b-tu Brandenburg University of Technology Cottbus
Computer Engineering Group

Idea of the Fault Tolerant Approach

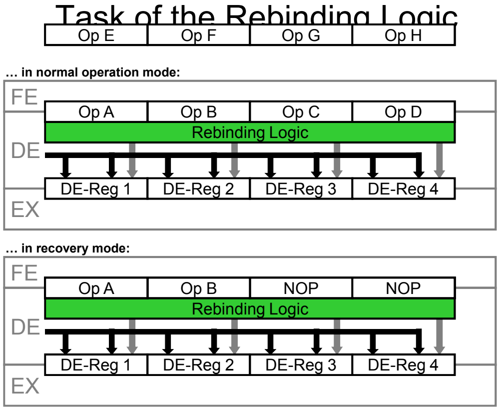
- Compiler Support:
  - Duplication of each operation.
  - Scheduling original and duplicate on different execution units.
  - The result of an original operation is not used before the duplicated operation is executed.
- Hardware Support:
  - Comparing results of original and duplicated operations in hardware.
  - Recovery-Mode: Re-execution of failed operations in order to obtain a third result.

b-tu Brandenburg University of Technology Cottbus
Computer Engineering Group

Fault Tolerant Architecture



Task of the Rebinding Logic

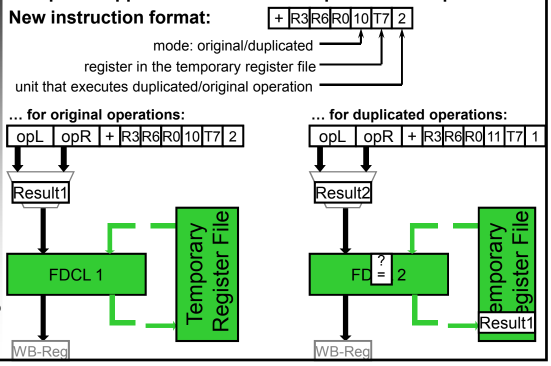


Computer Engineering Group
Introduction
HW Self-Repair
SW Self-Repair
Rebinding
Rescheduling
Hybrid Off-Line
Lower Granularity
Hybrid On-Line
Summary

Task of the FDCL

Compiler Support: **Each operation is duplicated.**

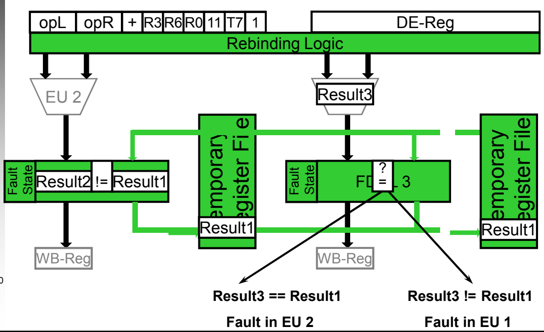
New instruction format:



Computer Engineering Group
Introduction
HW Self-Repair
SW Self-Repair
Rebinding
Rescheduling
Hybrid Off-Line
Lower Granularity
Hybrid On-Line
Summary

Fault Localization and Recovery

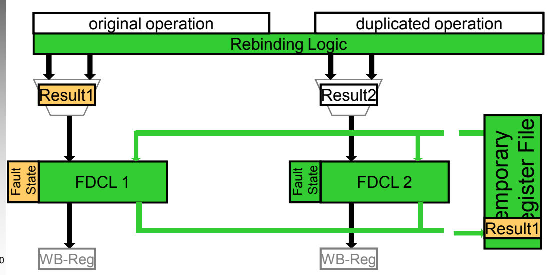
Comparing detects a mismatch:



Computer Engineering Group
Introduction
HW Self-Repair
SW Self-Repair
Rebinding
Rescheduling
Hybrid Off-Line
Lower Granularity
Hybrid On-Line
Summary

Handling of Permanent Faults

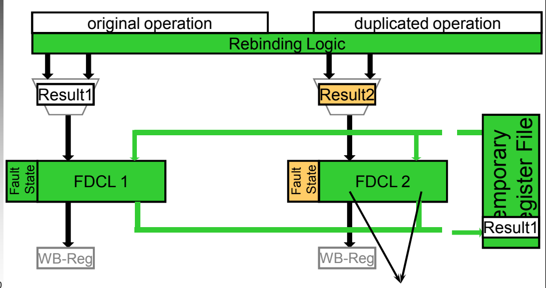
Original operation executes on faulty unit:



Computer Engineering Group
Introduction
HW Self-Repair
SW Self-Repair
Rebinding
Rescheduling
Hybrid Off-Line
Lower Granularity
Hybrid On-Line
Summary

Handling of Permanent Faults

Original operation executes on faulty unit:

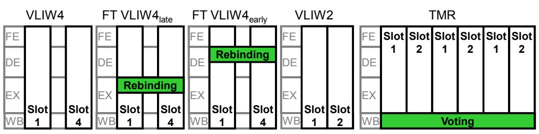


If both results are equal, a transient error has been discovered before.

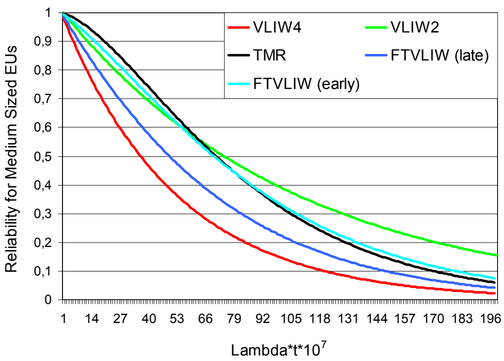
Computer Engineering Group
Introduction
HW Self-Repair
SW Self-Repair
Rebinding
Rescheduling
Hybrid Off-Line
Lower Granularity
Hybrid On-Line
Summary

Synthesis Results

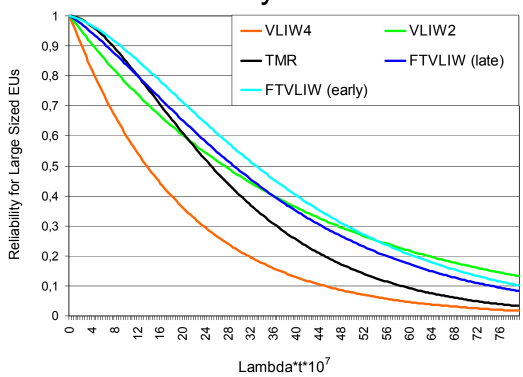
Architecture	Number of transistors	Transistors per slot that can fail
VLIW (4 slots)	**111.500 + EUs**	**None**
FT VLIW$_{late}$ (4 slots)	**142.700 + EUs**	**4000 + EU**
FT VLIW$_{early}$ (4 slots)	**143.400 + EUs**	**22.200 + EU**
VLIW (2 slots)	**54.500 + EUs**	**None**
TMR (6 slots)	**123.200 + EUs**	**17.000 + EU**



Reliability Results



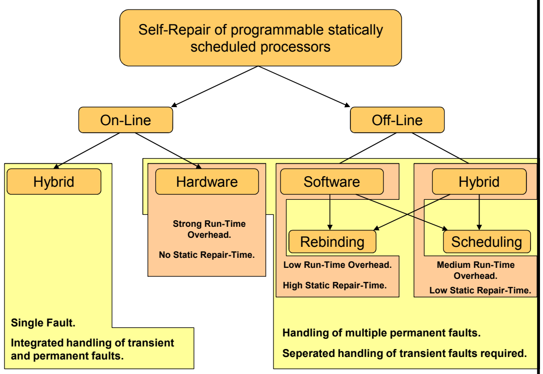
Reliability Results



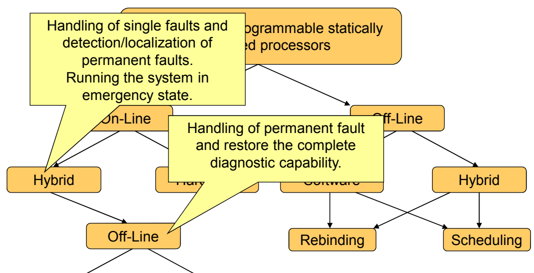
Summary – On-Line Self-Repair

- HW/SW-based method:
  - for the detection of transient faults by concurrent execution and
  - recovery from faults with localization of permanent faults by re-execution and majority vote (only 1 to 2 clock cycles).
- A detected permanent fault is masked without delay.
- Capability of detecting all faults get lost after an occurred fault.
- Simplicity of the processor is maintained; it remains scalable.
- Applicable for statically scheduled data paths with medium to large scaled EUs.
- Reliability can be traded against performance by the compiler.

Summary – Overall



What's Coming Next?



Multi-Core systems with embedded compilers.

Design for self-repair.

Thank You for Your Attention.