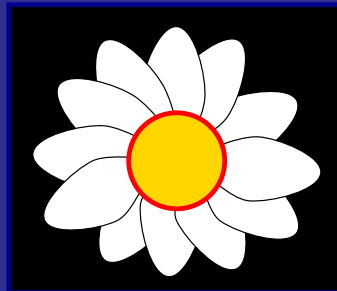


DAISY: Dynamic Compilation for 100% Architectural Compatibility

Kemal Ebcioglu
Erik R. Altman

IBM T.J. Watson Research Center



Presenter: Erik R. Altman

Background

- **Problem:** Many previous novel *ILP Machines* were quite different from *x86*, *PowerPC*, and *S/390* ⇒ Compatibility difficult.
- **Observation:** Acceptance of novel *ILP Machines* would be helped by compatibility with existing architectures.
- **Solution:** **DAISY** — Dynamically Architected Instruction Set from Yorktown.

DAISY Principles

- **First time** a fragment of code is executed, it is rapidly translated to ILP code for a simple underlying *ILP Machine* and saved in main memory.
- **Subsequent executions** of same fragment do not require a translation (unless cast out).
- All software (including kernel code) is translated to ILP code.

DAISY Features

- Achieves **100% architectural compatibility** with complex architectures.
- All existing software runs without changes on new simple *ILP Machine*.
- **Simple hardware** does not need *decode, dispatch, out of order execution* capability.
- Potential for **high frequency** and relatively **small die size**.
- **Scalable**: Same translation software runs on 2, ..., 16 ALU variants of *ILP Machine*.
- **Dynamic compilation** \Rightarrow unprecedented amount of runtime info at compile time.

What Is DAISY's Status?

- A DAISY dynamic compiler/simulator has been implemented.
- Implementation decodes *PowerPC* instructions from the binary.
- It schedules into ILP code using aggressive new, fast ILP compiler techniques, e.g. code is moved past branches, loop iteration boundaries.
- It assembles ILP code to *PowerPC* simulation code directly in binary.
- It executes the simulation code and generates same results as original program.

Outline of Talk

- Basic Concepts in DAISY
- Some Difficult Issues
- Machine Organization
- Results
- Conclusion

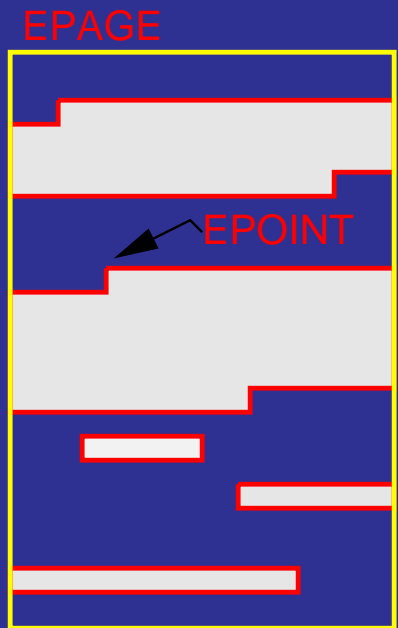
Outline of Talk

- Basic Concepts in DAISY
- Some Difficult Issues
- Results
- Conclusion

Basic Idea

1. Let **EPOINT** be the entry point of a program.
2. Let **EPAGE** be the code page on which **EPOINT** resides.
3. Translate all code on **EPAGE** that is directly reachable from **EPOINT**.
4. Begin execution of translated code for **EPOINT**.
5. When reach a previously untranslated entry point, set **EPOINT** to it, and **EPAGE** to the page and goto *Step 3*.

Translated Areas of Page



Subtleties

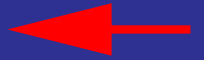
- Translation is done under control of the **Virtual Machine Monitor (VMM)**.
- The **VMM** is invisible to the user and may reside in ROM.
- The **VMM** actually works continuously — not on a per program basis as the previous slides suggested.
- The entry point of a program is jumped to like any other offpage branch.

Original PowerPC Code

DAISY

ISCA - 24

```
1)      add    r1,r2,r3
2)      bc     L1
3)      sli   r12,r1,3
4)      xor   r4,r5,r6
5)      and   r8,r4,r7
6)      bc     L2
7)      b     OFFPAGE
8)  L1:  sub   r9,r10,r11
9)      b     OFFPAGE
10) L2:  cntlz r11,r4
11)     b     OFFPAGE
```



Translated VLIW Code

VLIW1:

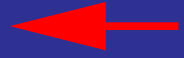
```
+ add r1,r2,r3
```

Original PowerPC Code

DAISY

ISCA - 24

```
1)      add    r1,r2,r3
2)      bc     L1
3)      sli   r12,r1,3
4)      xor   r4,r5,r6
5)      and   r8,r4,r7
6)      bc     L2
7)      b     OFFPAGE
8)  L1:  sub   r9,r10,r11
9)      b     OFFPAGE
10) L2:  cntlz r11,r4
11)     b     OFFPAGE
```



Translated VLIW Code

VLIW1:

```
      |
      | add r1,r2,r3
      | bc     L1
      |
      / \
```



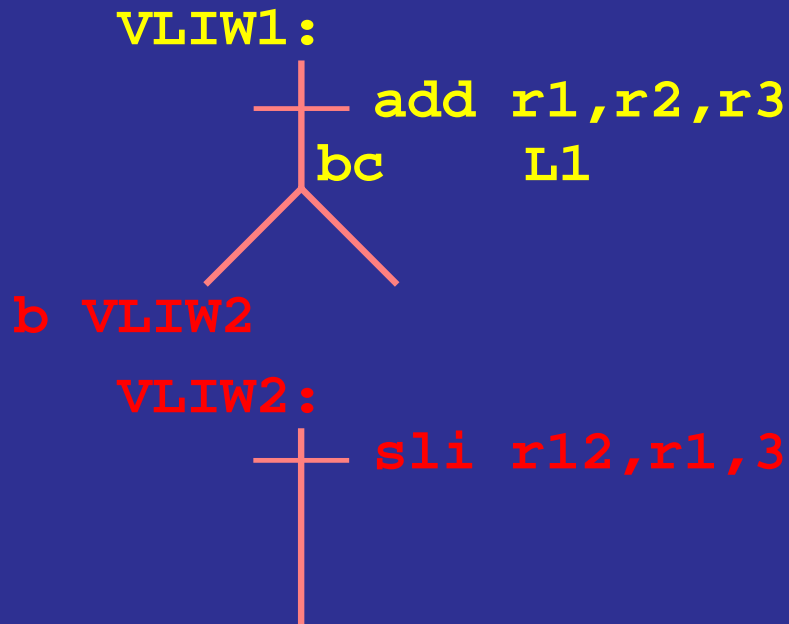
Original PowerPC Code

DAISY 1) **add** **r1,r2,r3**
2) **bc** **L1**
3) **sli** **r12,r1,3**
4) **xor** **r4,r5,r6**
5) **and** **r8,r4,r7**
6) **bc** **L2**
7) **b** **OFFPAGE**
8) L1: **sub** **r9,r10,r11**
9) **b** **OFFPAGE**
10) L2: **cntlz** **r11,r4**
11) **b** **OFFPAGE**

ISCA - 24



Translated VLIW Code



Original PowerPC Code

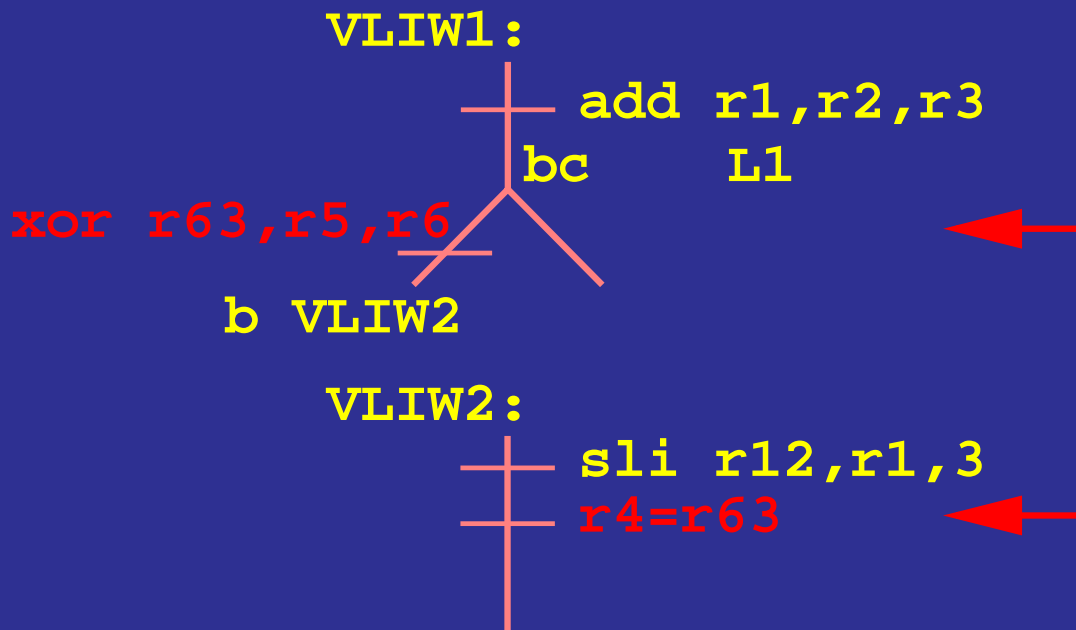
DAISY

```
1)      add    r1,r2,r3
2)      bc     L1
3)      sli   r12,r1,3
4)      xor   r4,r5,r6
5)      and   r8,r4,r7
6)      bc     L2
7)      b     OFFPAGE
8)  L1:  sub   r9,r10,r11
9)      b     OFFPAGE
10) L2:  cntlz r11,r4
11)     b     OFFPAGE
```

ISCA - 24



Translated VLIW Code



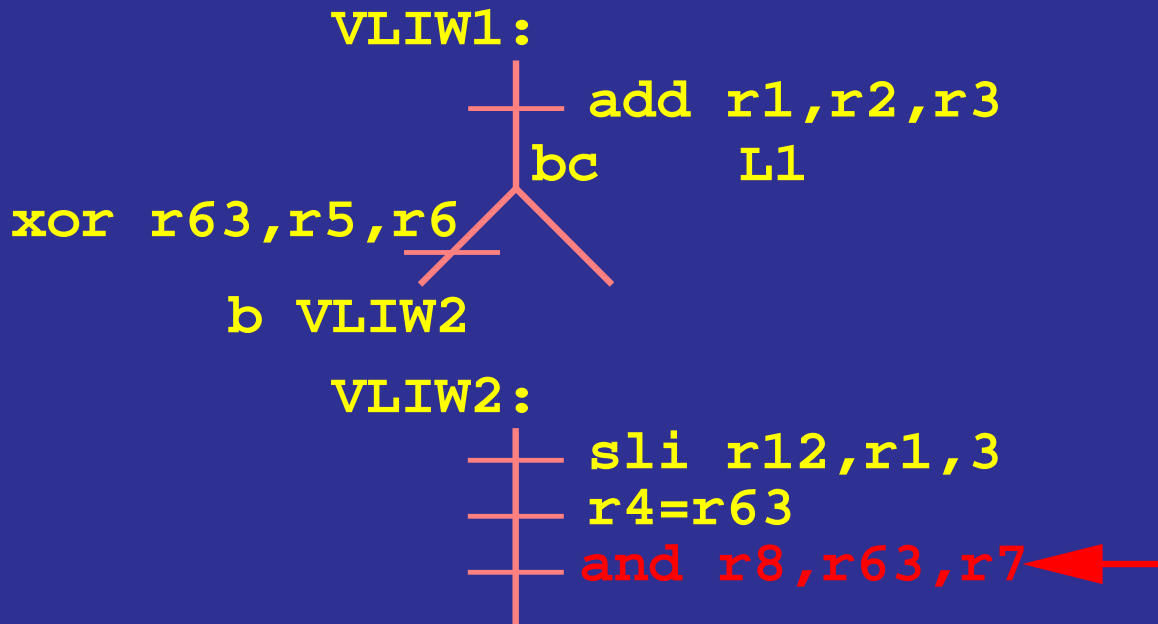
Original PowerPC Code

DAISY 1) **add** **r1,r2,r3**
2) **bc** **L1**
3) **sli** **r12,r1,3**
4) **xor** **r4,r5,r6**
5) **and** **r8,r4,r7**
6) **bc** **L2**
7) **b** **OFFPAGE**
8) L1: **sub** **r9,r10,r11**
9) **b** **OFFPAGE**
10) L2: **cntlz** **r11,r4**
11) **b** **OFFPAGE**

ISCA - 24



Translated VLIW Code



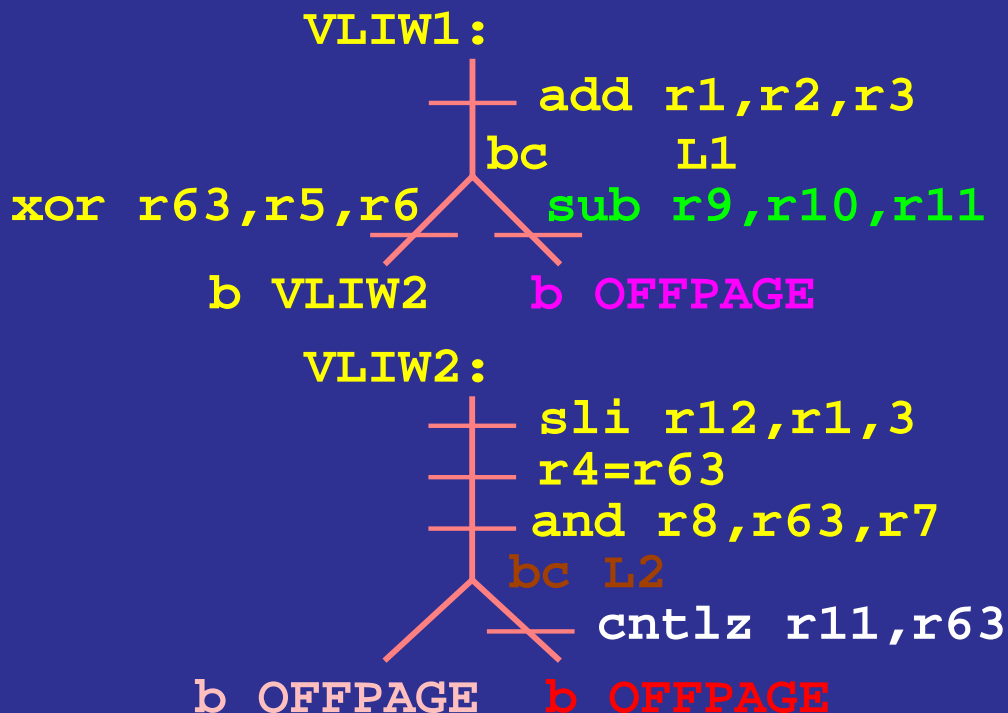
Original PowerPC Code

DAISY

```
1)      add    r1,r2,r3
2)      bc     L1
3)      sli   r12,r1,3
4)      xor   r4,r5,r6
5)      and   r8,r4,r7
6)      bc     L2
7)      b     OFFPAGE
8) L1:  sub   r9,r10,r11
9)      b     OFFPAGE
10) L2: cntlz r11,r4
11)     b     OFFPAGE
```

ISCA - 24

Translated VLIW Code



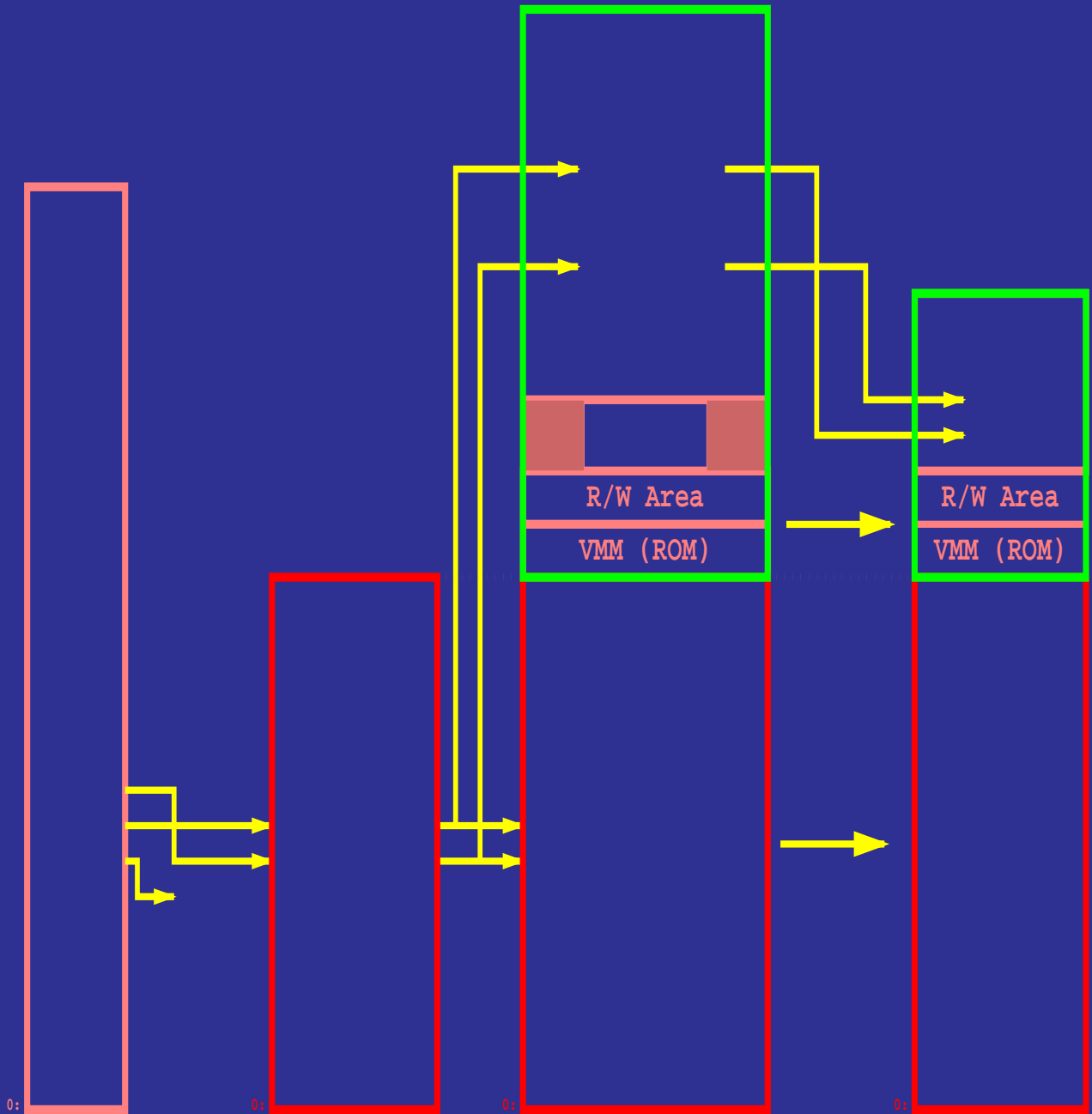
Memory Map

Base Arch Virtual

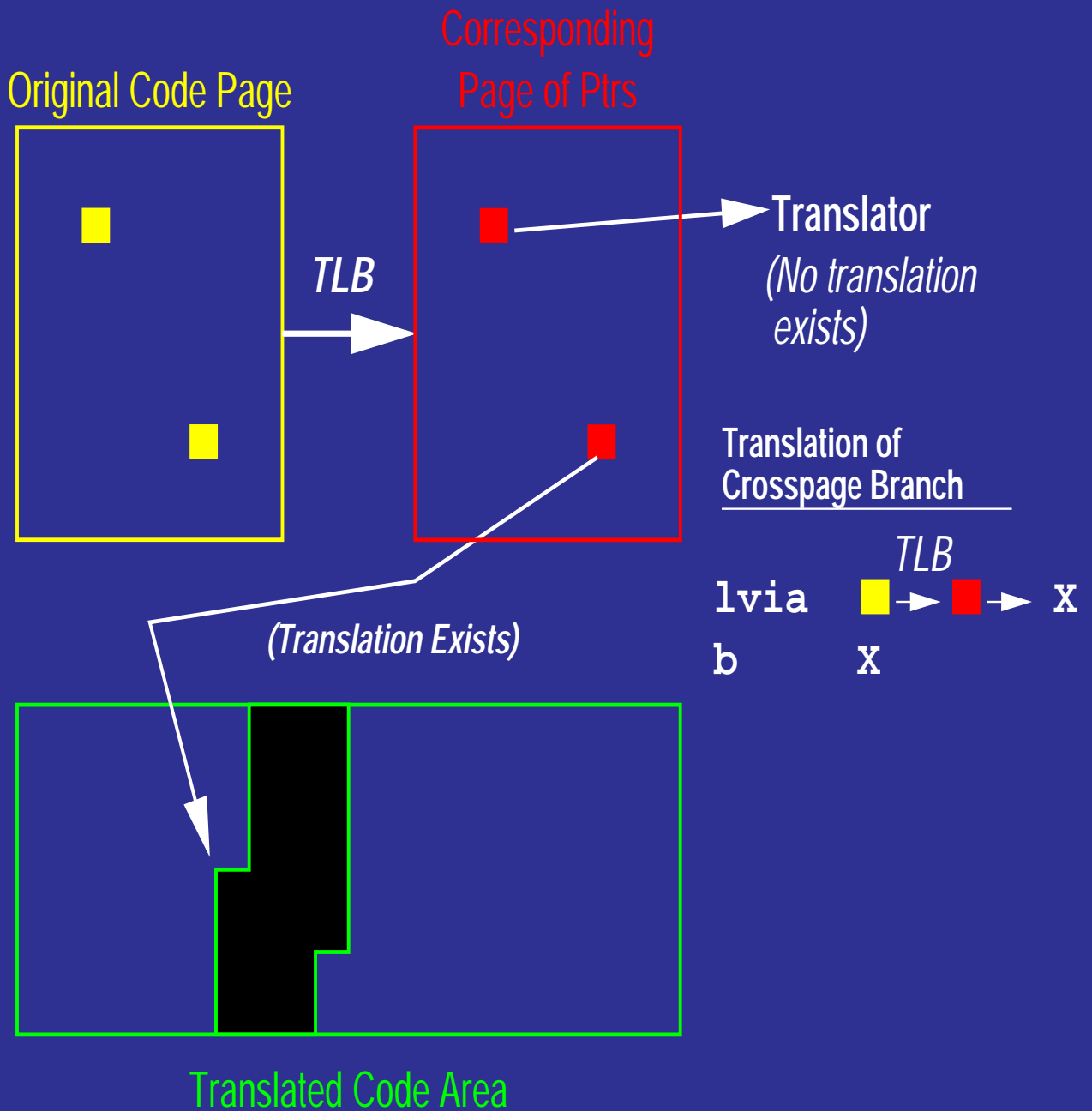
Base Arch Real

ILP Virtual

ILP Real



Crosspage Branches



Old Ins Addr from DAISY Addr

- DAISY has VPA register (Virtual Page Address)
- VPA is set to *ILP Machine* address on crosspage and indirect jumps.
- Each DAISY instruction has a 10-bit field to indicate offset in a 4K page.
- ⇒ Can determine address of *Base Arch* op corresponding to any DAISY op modifying an *Base Arch* resource.

(Interpretation of ops within a DAISY instruction may be required.)

Can DAISY Handle Self-Modifying Code?

- **Answer:** Yes.
- Each *unit* of *base architecture* memory has a new **read-only** bit, not known to the *base architecture*.
- Whenever the **VMM** translates any code in a memory *unit*, it sets its **read-only** bit to a 1.
- Whenever a store occurs to a memory unit marked **read-only**, an interrupt occurs to the **VMM**, which invalidates the translation of the page containing the *unit*.

Does DAISY Work with Self-Referential Code?

- **Answer:** Yes.
- **Example:** Programs that checksum themselves.
- All *base arch* regs, including **IAR** also in *ILP Machine*.
- All *base arch* regs contain the same values as the program would running on the *base arch*.
- Only means for code to refer to itself is through these regs.
- \Rightarrow Self-referential code is trivially handled, i.e. the reference looks at *base arch* code.

How Does DAISY Handle External Interrupts?

1. The **VMM** determines the *base arch* instruction that was executing when the exception occurred.
2. The **VMM** performs interrupt actions required by the *base arch*.
3. **Example:** Put *base arch* address of **current instruc** in **reg** appropriate for the *base arch*.
4. The **VMM** branches to the translation of the *base OS* code that handles the exception.

How Does DAISY Handle Real Time Programs?

- Translations can be pinned in memory.
- Areas to pin can be determined:
 - By heuristics (e.g. low memory interrupt handlers).
 - By passing information to VMM.

Result Extenders in DAISY

Integer, Float, and Condition registers get extender bits.

- Extenders have bit(s) indicating exceptions associated with result.
- Extenders allow unique serializing resources to be renamed.
- E.g. *PowerPC addic* sets Integer Register **Rx** and **CA** bit \Rightarrow If **Rx** renamed **Rx'**, then **CA** renamed in **Rx'** extender bit.

Speculative loads to I/O space are squashed by DAISY machine, and exception tag set on destination register.

Support for *S/390* in DAISY

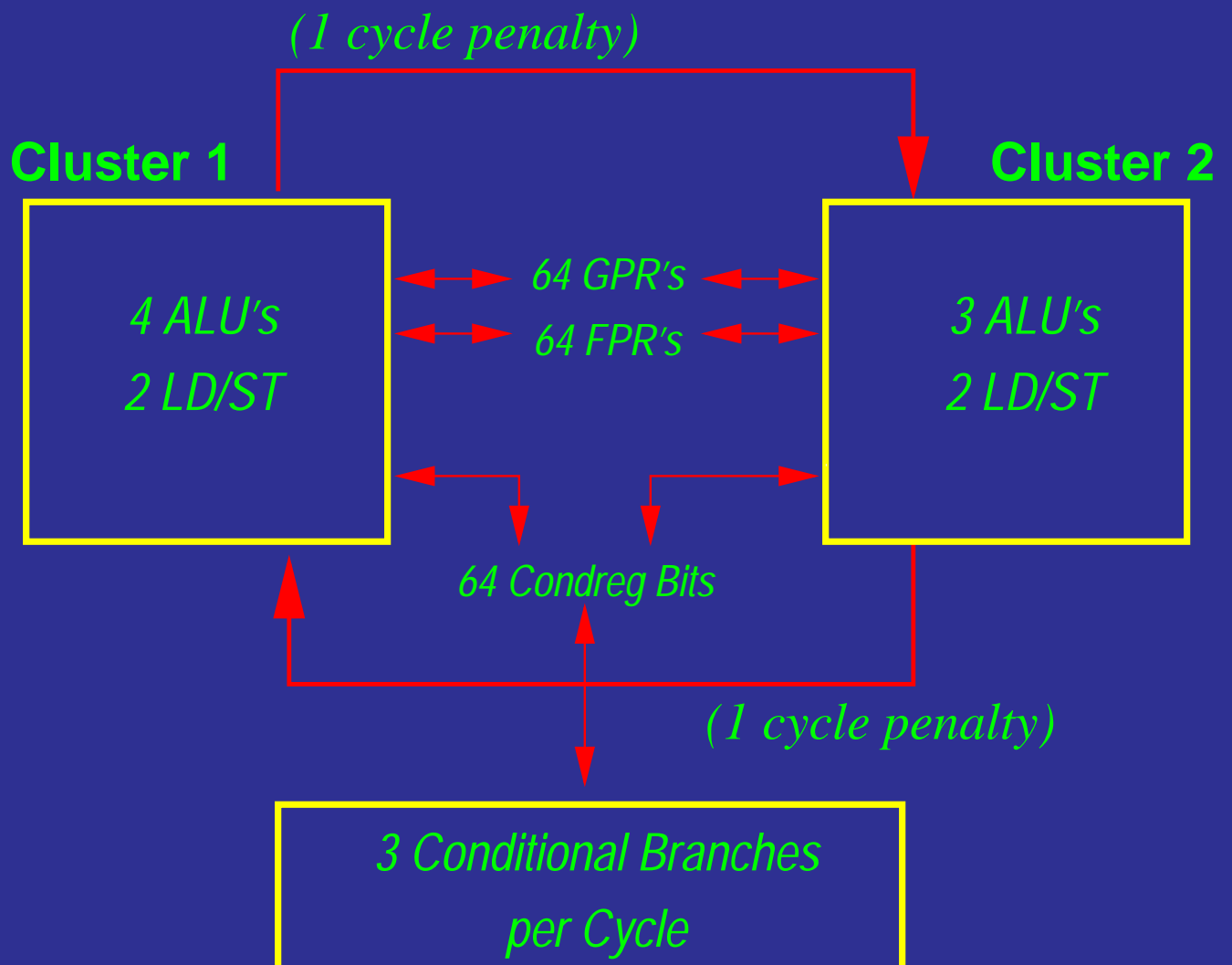
To avoid long, inefficient software emulation sequences, DAISY requires some hardware support specific to *S/390*:

- **BCD Operations:** \Rightarrow *Excess 6* Arithmetic on 64-bit registers.
- **24 and 31 Bit Addressing:** \Rightarrow **Effective Address Mask Register.**
- **DAT/key Protection:** **Address prefix register** (containing e.g. **PSW** key) is prepended to *S/390* effective address before accessing DTLB. Software handles DTLB miss.
- **Address Calculations:** \Rightarrow 3-input **add** op.

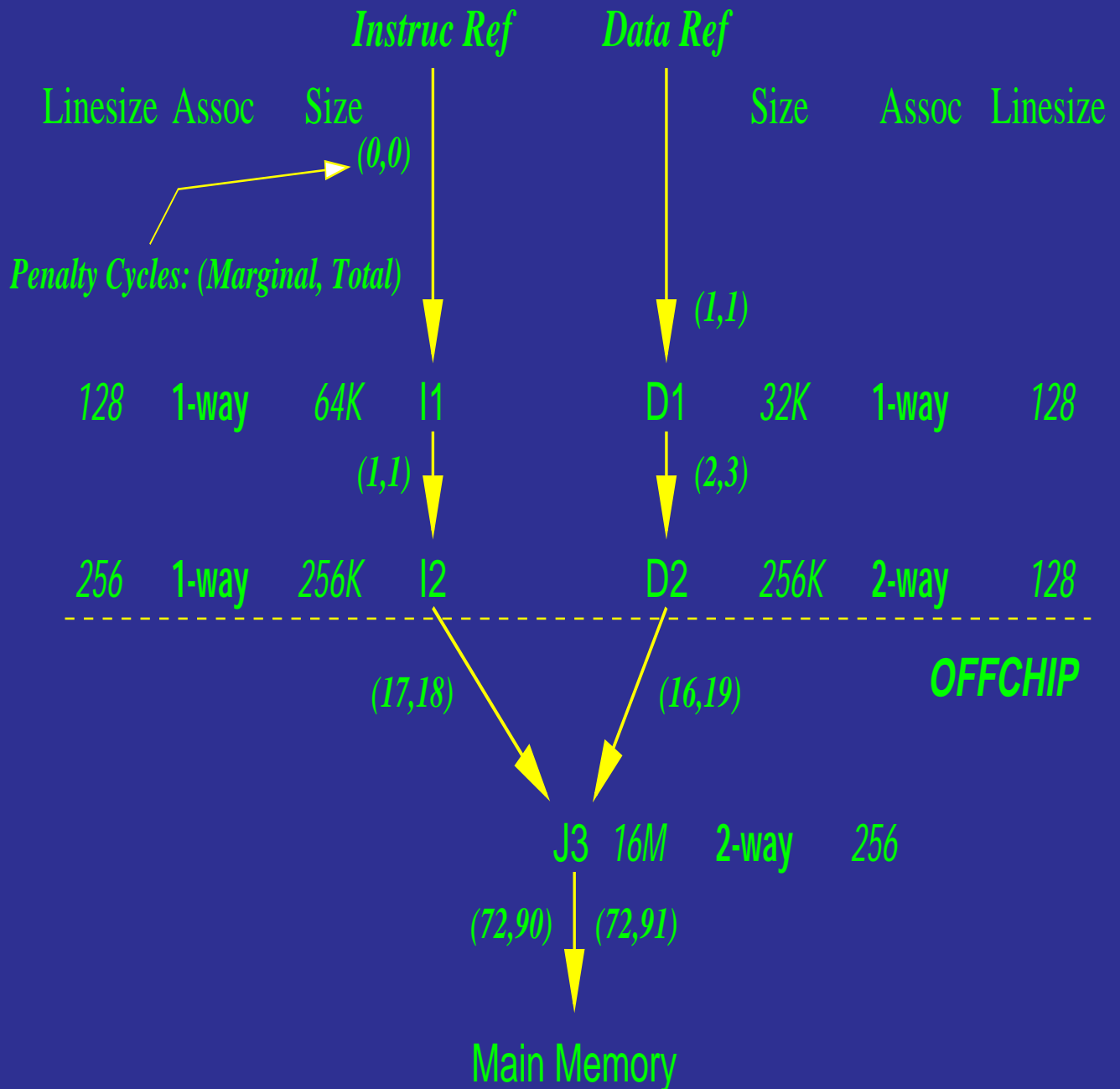
DAISY Essentials

- VMM controlled TLB.
- **LVIA** instruction.
- **VPA** register.
- **Read-only** bits for each chunk of native architecture memory.
- Superset of base architecture registers.
- Extra registers and extenders for renaming.
- Instructions for committing extender results.
- Able to reserve part of real mem for **DAISY**.

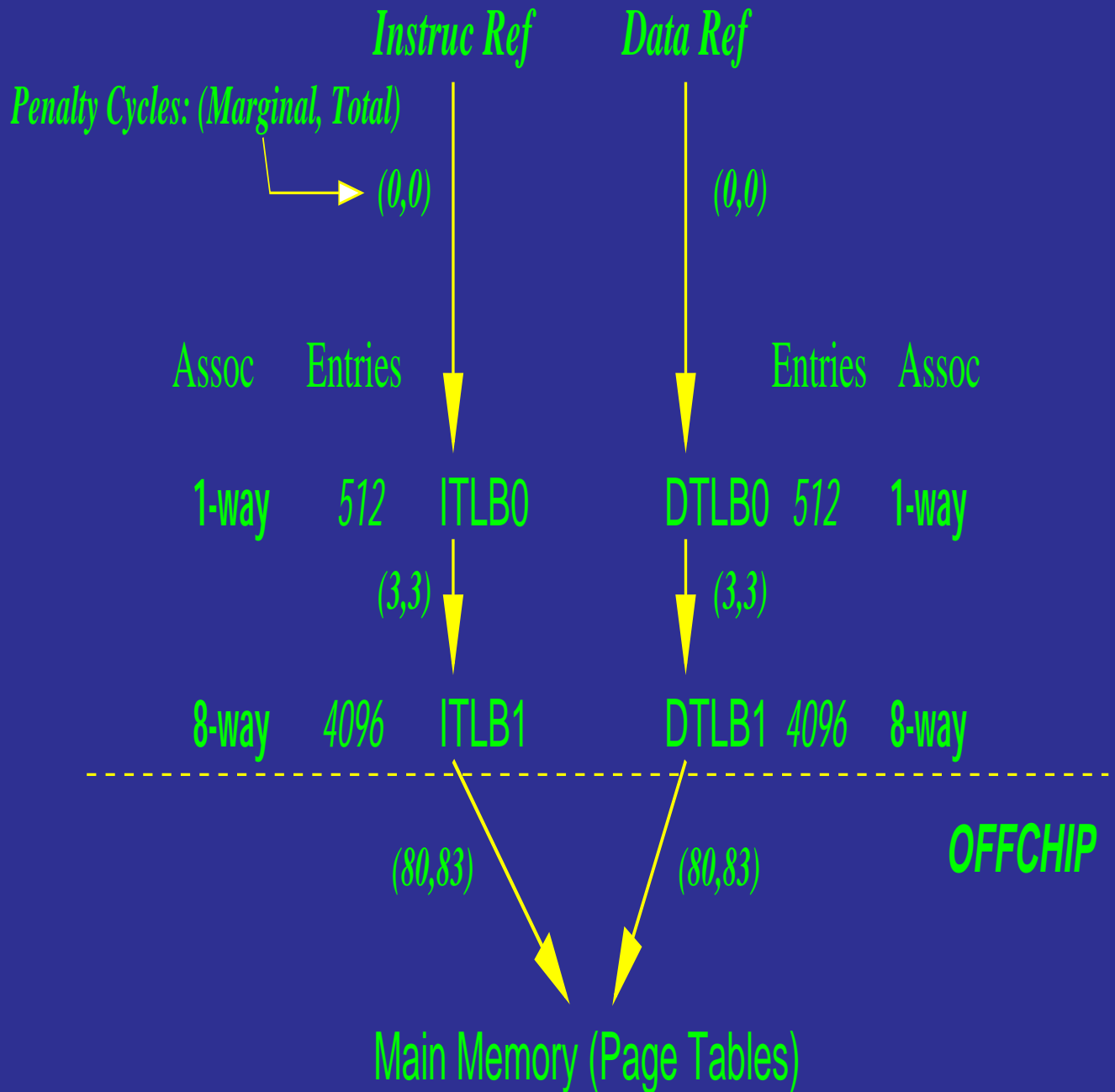
Current Implementation of DAISY



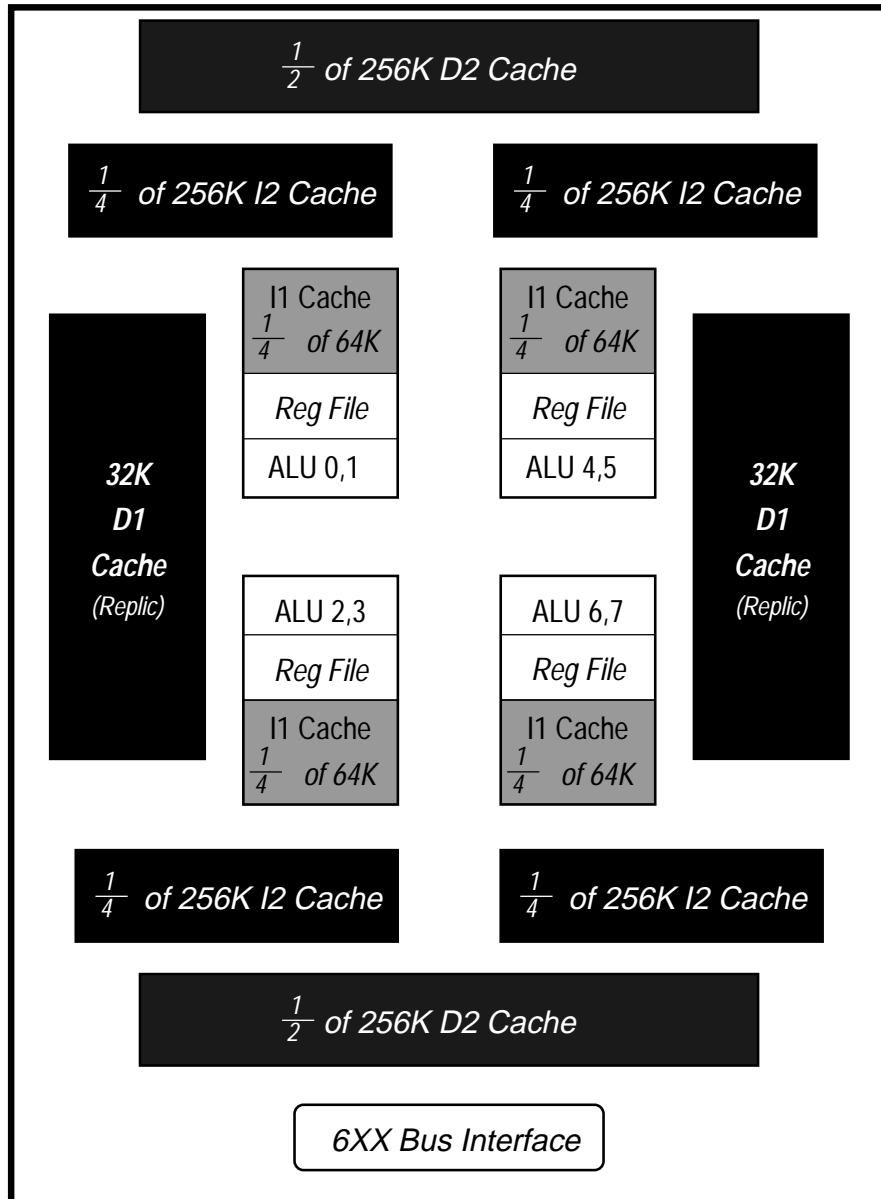
Cache Configuration



TLB Configuration



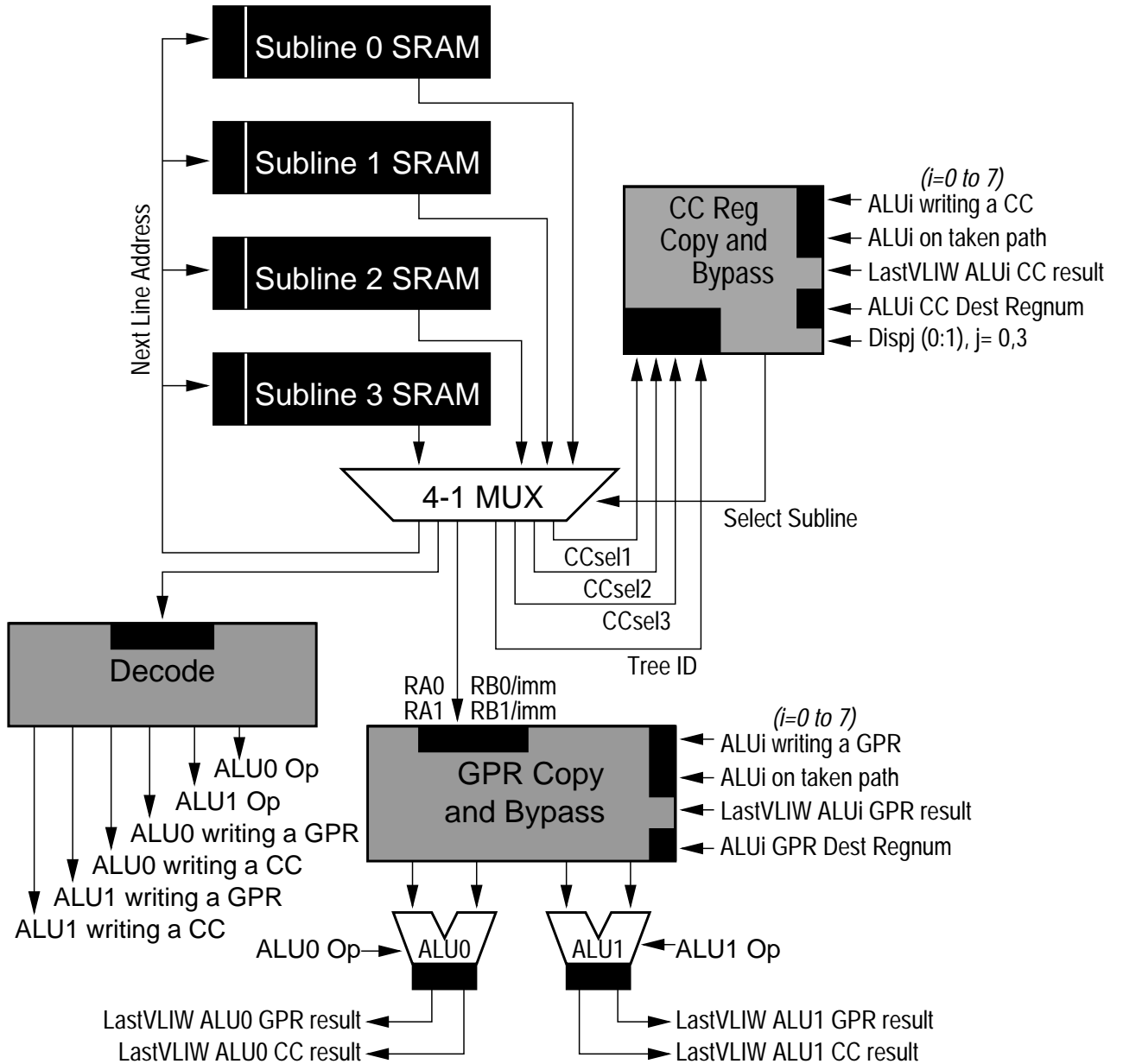
DAISY Layout



↕
6XX Bus

DAISY Cluster

ALU 0,1 Cluster



Current Implementation of DAISY

- DAISY is currently implemented on RS/6000 *PowerPC* Workstations running AIX.
- The *Base Arch* is *PowerPC*.
- The *ILP Machine* is a VLIW executing tree instructions.
- The *ILP Machine* size (e.g. # of ALU's) is configurable.

What Exactly Does DAISY Simulate?

- Does the DAISY implementation use traces or architectural simulation?

Answer: A functional level architectural simulation. Simple cache and TLB simulators have been implemented also.

- Were programs fully simulated to completion?

Answer: Yes

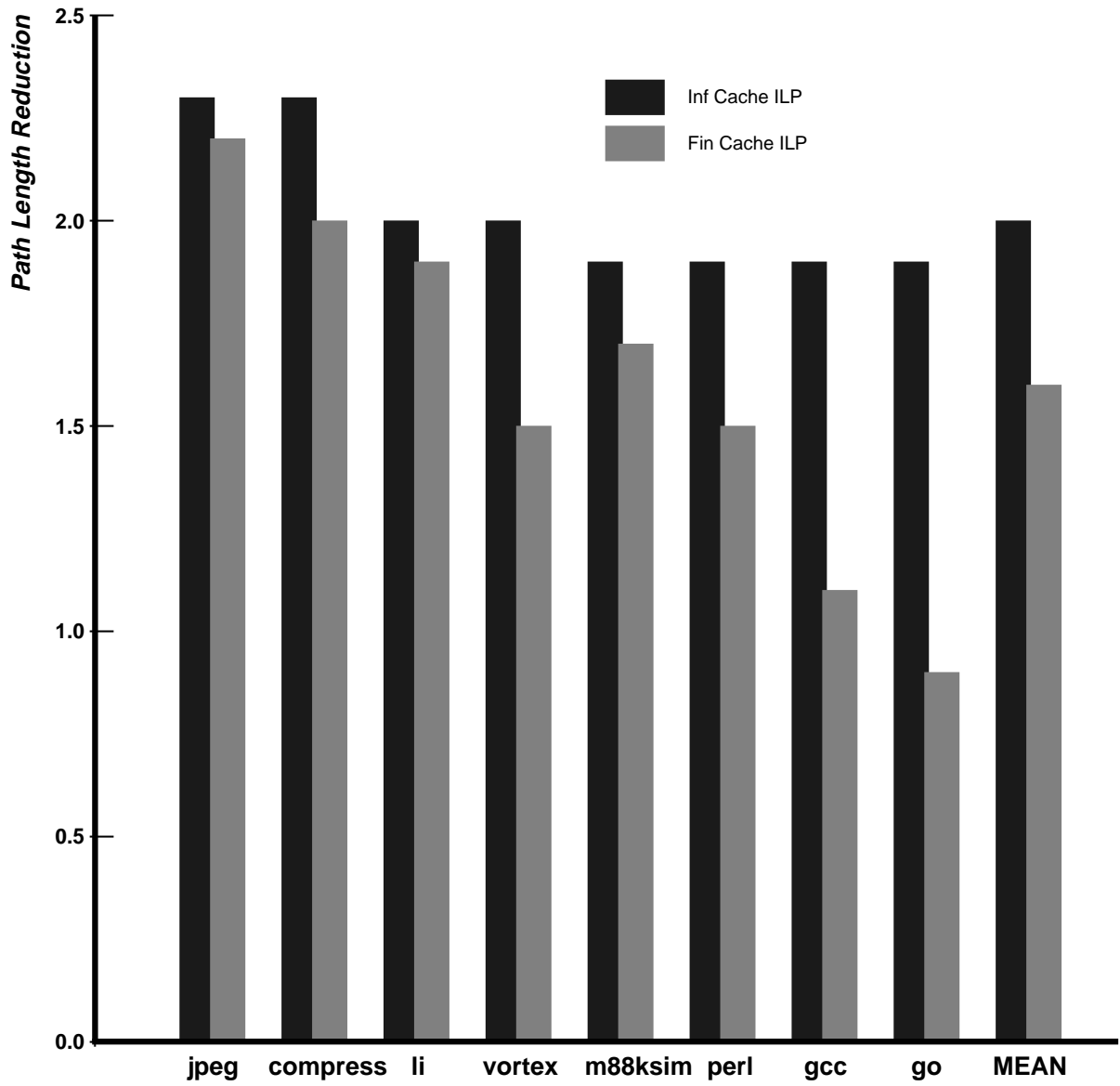
- Are shared libraries simulated?

Answer: Yes

- Is kernel code simulated?

Answer: No.

DAISY's ILP



Ways to Improve ILP

- Schedule across code pages
- Profile directed feedback
- Remove remaining serializers, e.g. **rlwimi**, **fcmpo**
- Software Pipelining
- Unification of ops occurring on multiple paths
- Special hardware for register copies
- Stall-on-use memory system

Crosspage Branch Frequency

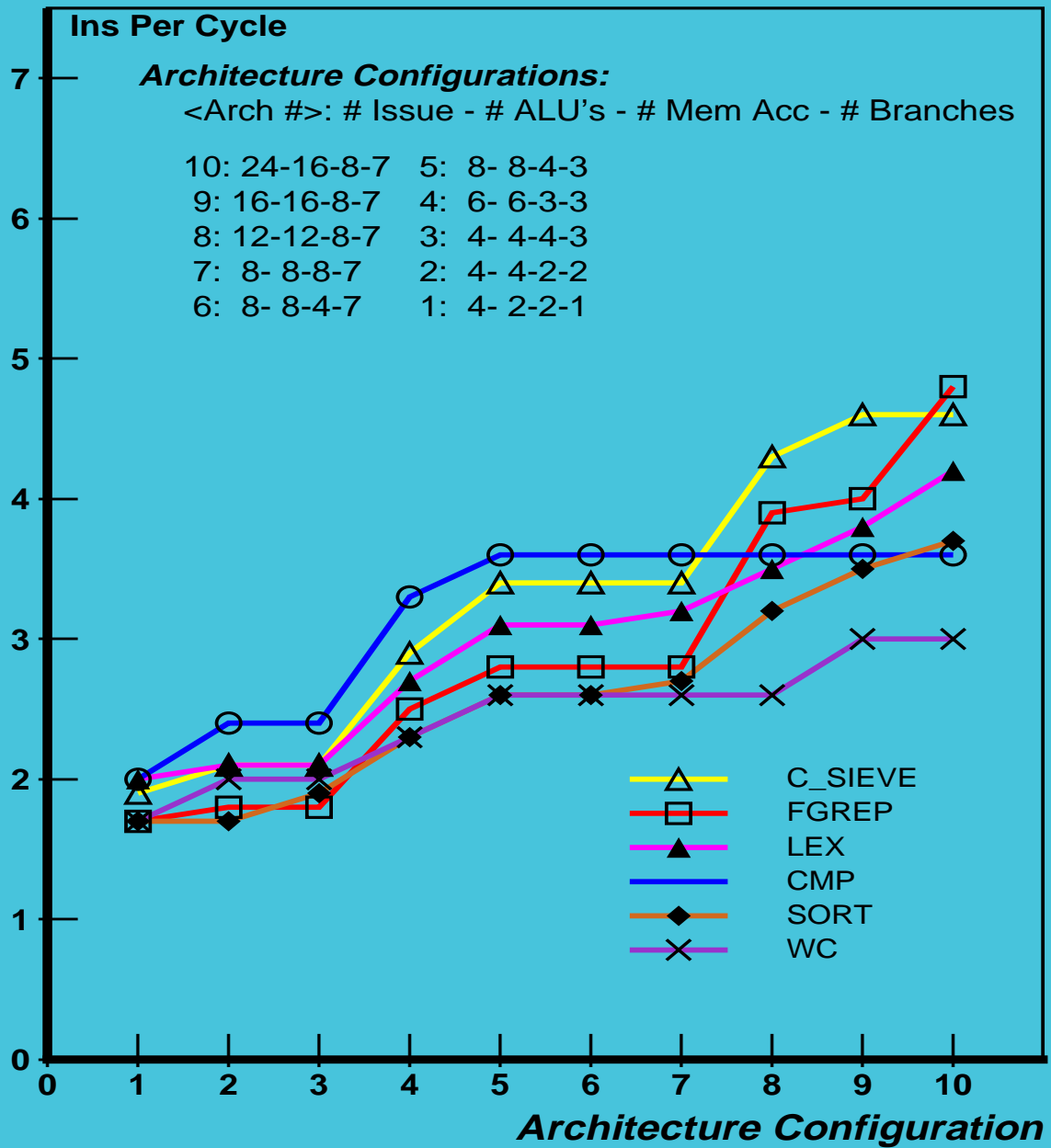
Program	VLIW's Between Crosspage Branches
compress	22
m88k	13
go	12
li	11
vortex	10
jpeg	38
cc1	11
perl	7
MEAN	16

Instruction Counts

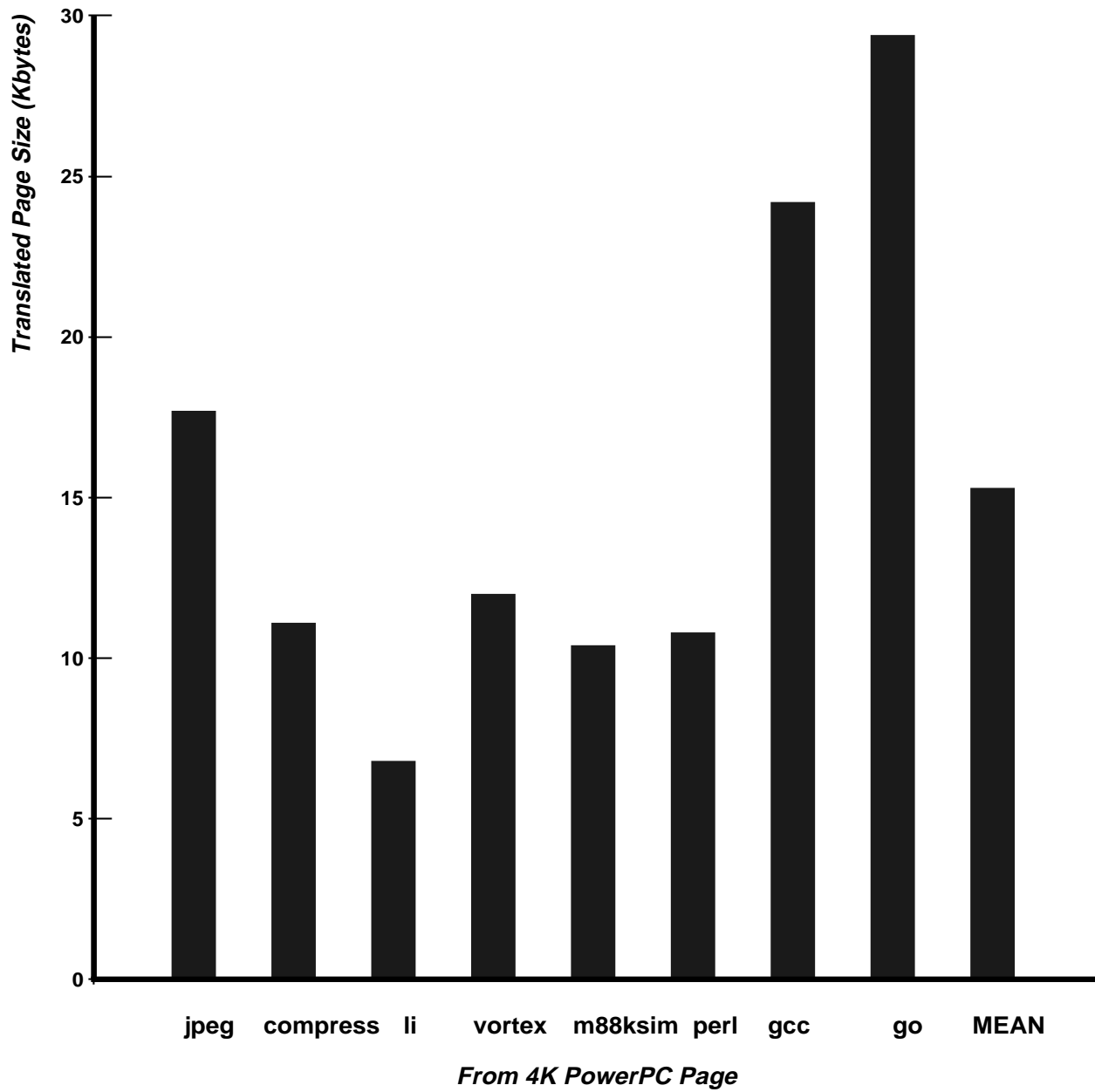
SPECint95 Training Inputs

	<i>PowerPC Ops</i>	VLIW Ins
compress	29M	12M
m88k	111M	58M
go	405M	216M
li	179M	88M
vortex	2334M	1167M
jpeg	1122M	482M
gcc	1131M	535M
perl	2093M	1099M

How Does ILP Scale?



Code Expansion



How Fast Is DAISY?

- About 4315 Instructions to Translate One Instruction. We hope to reduce this to 1000.
- Execution of **/bin/ls** on 60 MIPS machine:
 - **Takes** 8 seconds.
 - **Executes** 122M ops during translation.
 - **Produces** 292K of VLIW code.
- 8 seconds includes generation of *PowerPC* simulation code.

How Much Instruc Reuse Does DAISY Need To Be Viable? – 1

# Ins to Compile an Instruction	Unique Code pages	Reuse Factor	Time Change
4000	200	39000	-47%
4000	1000	7800	14%
4000	10000	780	707%
1000	200	39000	-59%
1000	1000	7800	-43%
1000	10000	780	130%

2 second program on 1 GHz machine.

How Much Instruc Reuse Does DAISY Need To Be Viable – 2?

What about short programs where execution time actually increases because of small reuse?

- Total time (*translation+execution*) is still imperceptible.
- If too many unique pages are executed, base machine would have paging overhead anyway.
- *Caution:* Cache and memory in DAISY must be sized to prevent thrashing in translated code area.

Instructions in SPEC95

SPEC95 Program	Dynamic Ins Executed	Static Code Size in Ins Words	Ins Reuse Factor
<i>INTEGER</i>			
go	81G	136K	595K
m88ksim	74G	85K	878K
cc1	34G	357K	95K
compress95	46G	52K	889K
li	67G	67K	994K
jpeg	69G	89K	771K
perl	49G	139K	351K
vortex	82G	212K	385K
<i>FLOATING POINT</i>			
tomcatv	20G	81K	243K
swim	23G	81K	287K
su2cor	25G	94K	264K
hydro2d	35G	96K	367K
ngrid	52G	83K	627K
applu	36G	100K	364K
turb3d	61G	90K	675K
apsi	21G	120K	177K
fpppp	98G	91K	1077K
wave5	25G	120K	210K
MEAN	50G	116K	514K

Strengths of DAISY – 1

- **Compatible:** No changes are needed in existing code.
- **Reliable:** If a bug is found in a **DAISY** implementation, only a software patch is needed to fix it.
- **Upgradable:** If better compiler algorithms are found, for **DAISY** only a software patch is needed to install them.
- **Upgradable:** Future architectural improvements are transparent to the user.

Strengths of DAISY – 2

- **Testable:** During evaluation, simulation is straightforward and direct for
 - Shared library code (e.g. **malloc**).
 - Existing binaries for which source code is not available.
- **Simple Fast Hardware:** Intelligence is in software, allowing simple in-order implementations.
- **Wide scope for ILP:** DAISY looks at a full page of code, providing good opportunity to find parallelism.

Strengths of DAISY – 3

- **Efficient:** Only pages with executed code are translated.
- **Robust:** No auxiliary information about program entry points and jump targets is needed.
- **Intelligent:** During idle time, more optimized compilation is possible for frequently executed code.

Weaknesses of DAISY

- ILP attained may be slightly lower than that achieved by compiler without “real-time” constraints.
- **Slow Initial Compilation:** Initial compilation time required by DAISY may mean short programs with little reuse do not achieve the speedup of larger programs with significant code reuse.
- More memory is required than in the *Base Machine*.

What's Ahead for DAISY?

- Support a Java Virtual Machine.
- Other architectures, such as *S/390* and *x86*?
- Handle kernel code.
- Additional optimizations, such as software pipelining.

Conclusions

- **DAISY** is a new architectural platform providing:
 - **Full compatibility** with existing architectures.
 - **High levels of ILP.**
 - **Simple hardware** ⇒ Potential for high clock speeds.
 - **Robustness and Flexibility:** Performance improvements and bug fixes to hardware in the field can be made via a simple software patches.

⇒ www.research.ibm.com/daisy