You Can't Always Get What You Want

If you try sometimes you might find you get what you need – *but maybe not*. That's the problem with **caches**. As modern processors have sprouted increasingly complex memory hierarchies, they still have not solved the fundamental problem of ensuring that instructions and data are always ready when they are needed.

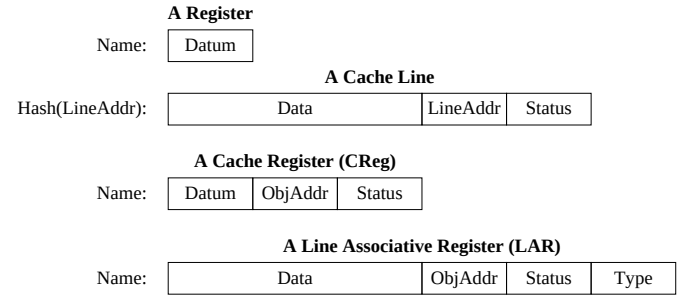
A value kept in a **register** is there when you need it. However, there are lots of memory objects conventional machines do not keep in registers. Instructions are not kept in registers, although there really isn't a good reason why not. There is a widely held belief that registers don't help with "spatial locality" – but that is true only if we assume that a register is just one object wide.

Of course, ambiguously aliased data cannot be kept in registers. For example, if a code refers to `a[i]` and `a[j]` a number of times, there are three possible circumstances:

- If the compiler knows `i==j`, then `a[i]` and `a[j]` are *unambiguously aliased* and can share a single register
- If the compiler knows `i!=j`, it can allocate one register for `a[i]` and another for `a[j]`
- If the compiler doesn't know the relationship between `i` and `j`, any change to one might require update of the other, so the changed value is stored and the value of the *ambiguously aliased* object must be reloaded

In fact, ambiguously aliased object references are the only reason a processor needs cache: without cache, the store and reload would become slow references to main memory. Even so, it is possible that the cache would be ineffective, because references to other objects might have had addresses hashing to the same cache line slot, thus evicting the desired object from cache. However, everything needed to resolve the ambiguity was in the processor – why not simply modify the register file to automatically update aliased objects in other registers?

CRegs & LARs. Our SC'88 paper, *CRegs: A New Kind of Memory for Referencing Arrays and Pointers*, introduced CACHE-REGISTERS to solve the ambiguous alias flushing problem by *associatively updating* registers whose address fields match. Commercial adoption of CREGs was impeded by the need for a CREG ISA, although the Itanium "Advanced Load" mechanism obtains some of the benefits with only minor ISA adjustments. However, data CREGs don't make use of spatial locality – our work on LARs (LINE ASSOCIATIVE REGISTERS) over the past few years combines CREGs with SWAR (SIMD WITHIN A REGISTER) to take full advantage of spatial locality. The basic hardware cell structures are:



An example using Data LARs. In the following C code, suppose `a[i]`, `a[j]`, and `a[k]` are ambiguously aliased. The LAR code not only reduces memory loads, but also provides implicit lazy stores. Because each LAR records the current object position within its data field, each LAR is type tagged, and type information need not be encoded within arithmetic instructions. The LAR code as written is scalar; however, if the data are properly aligned, replacing `Mul` and `Add` with `Mulp` and `Addp` would perform the SWAR parallel operations on a "line" of data at a time. In any case, scalar or parallel, the LAR code makes no more than 3 memory references – fewer if any of the lines have the same base addresses.

C Code	Conventional RISC	LAR RISC
`float *a;`	`Lea r0,a[j]`	`Ldf r1,a[j]`
`...`	`Ldf r1,@r0`	`Ldf r3,a[k]`
`a[i]=a[j]*a[k];`	`Lea r2,a[k]`	`Ldf r5,a[i]`
`a[k]=a[j]+a[k];`	`Ldf r3,@r2`	`Mul r5,r1,r3`
	`Mulf r4,r1,r3`	`Add r3,r1,r3`
	`Lea r5,a[i]`	
	`Stf r4,@r5`	
	`Ldf r1,@r0`	
	`Ldf r3,@r2`	
	`Addf r6,r1,r3`	
	`Stf r6,@r2`	

Instruction LARs. Instruction LARs remove the instruction fetch process from the execution of each instruction, replacing it with separate, explicit, instructions that load of compressed blocks of instructions. If a load requests a block that is already in another instruction LAR, the decoded instruction block is logically copied without any memory activity. Control flow targets are specified using an instruction offset within an instruction LAR, rather than by comparatively lengthy memory addresses. The overall result is a smaller memory footprint, improved utilization of memory bandwidth, and complete freedom from misses during instruction processing.

Status. Krishna Melarkode's 2004 M.S. Thesis, *Line Associative Registers*, is the most complete document describing the new concepts in LARs beyond CREGs. A number of theses are currently in progress; watch **`Aggregate.Org`** for more information.

This document should be cited as:

```
@techreport{sc07lar,
author={Henry Dietz and William Dieter},
title={You Can't Always Get What You Want},
institution={University of Kentucky},
address={http://aggregate.org/WHITE/sc07lar.pdf},
month={Nov}, year={2007}}
```

Aggregate.Org UNBRIDLED COMPUTING

SC07