15

LIFE–1 : A single chip integer VLIW processor core

Gerrit A. Slavenburg & Junien Labrousse

contents

Introduction to VLIW

LIFE - 1 single chip VLIW
- Block diagram & principles of operation
- External interface
- Operation set

Architectural highlights
- Guarded operations

Run-time statistics
- Performance = f(branchdelay)
- Functional unit utilization
- Parallelism and speedup

Conclusions

A challenge for the 90's

Introduction to VLIW

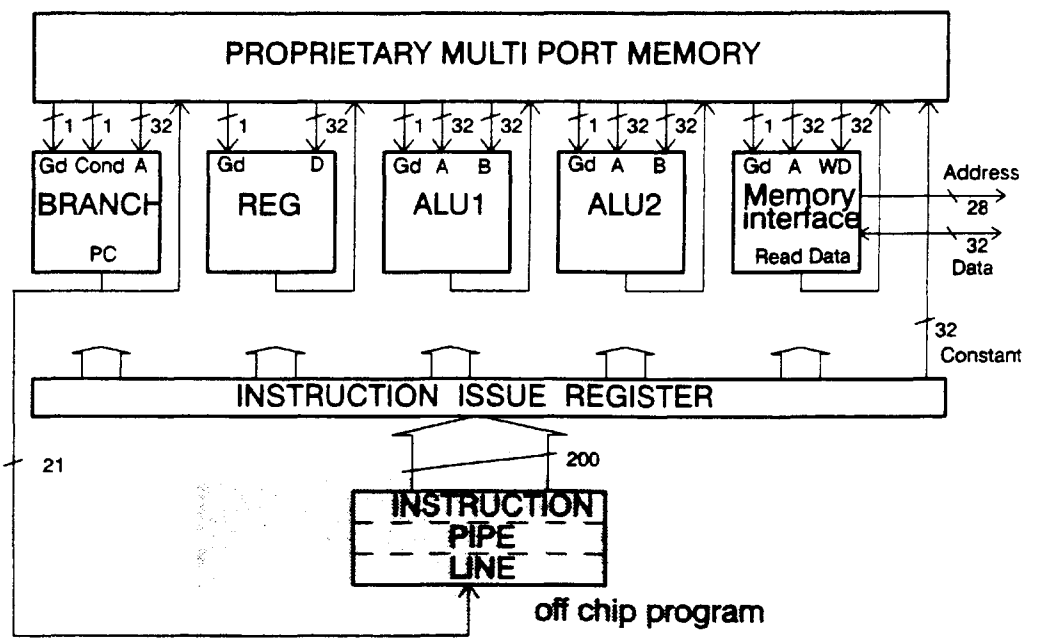
VLIW (Very Long Instruction Word) :

- Fine grain parallelism (RISC instruction level)
- Compiler decides which operations can issue in parallel
- VLIW instruction = multiple operations
- Some programs have inherently little fine-grain parallelism

Design issues :

- The *instruction issue rate* should be comparable to a RISC (for the same technology)
- A solution is needed to cope with the *branchdelay* problem

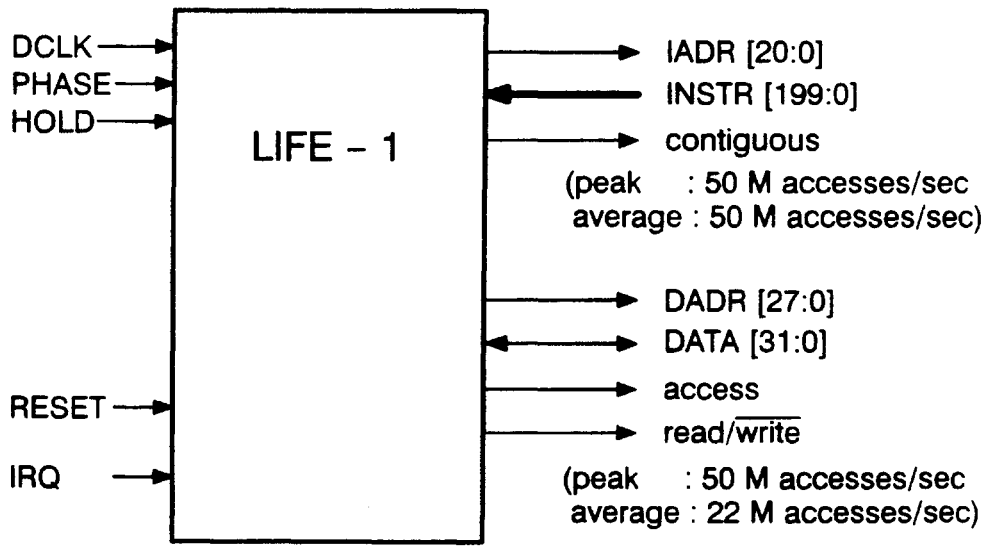
LIFE-1 block diagram



LIFE-1 principles of operation

- ☆ multiple, pipelined *functional units*
- ☆ all functional units receive their arguments/store their result in the central *multiport memory*
- ☆ an *operation* is started on each unit each clock cycle
- ☆ the *latency* time of a unit is the same for all operations on that unit and is determined by the unit's logic complexity
- ☆ a single, *very long instruction* controls all operation codes, argument addresses and destination address of the operations started in a particular clock cycle
- ☆ external data memory is treated as a functional unit with the operations *load* and *store*

LIFE-1 external interface

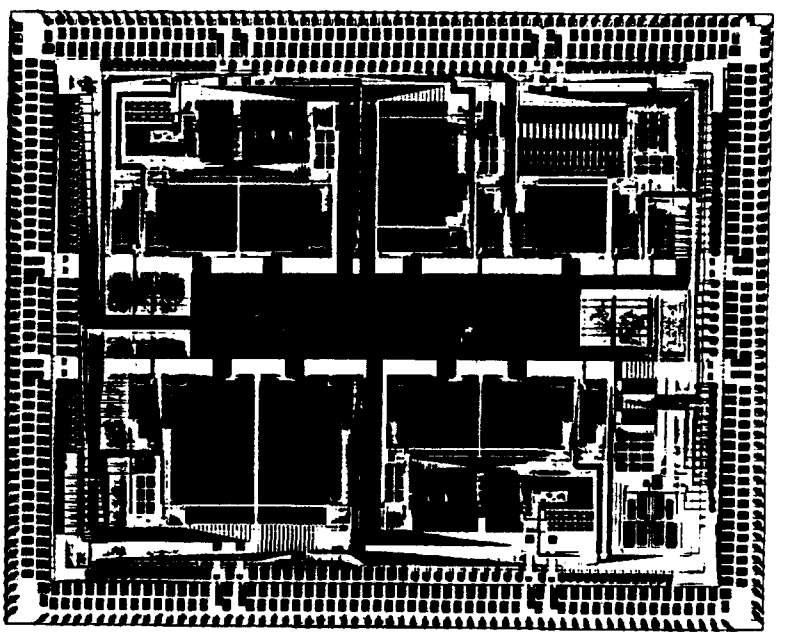


LIFE-1 operation set

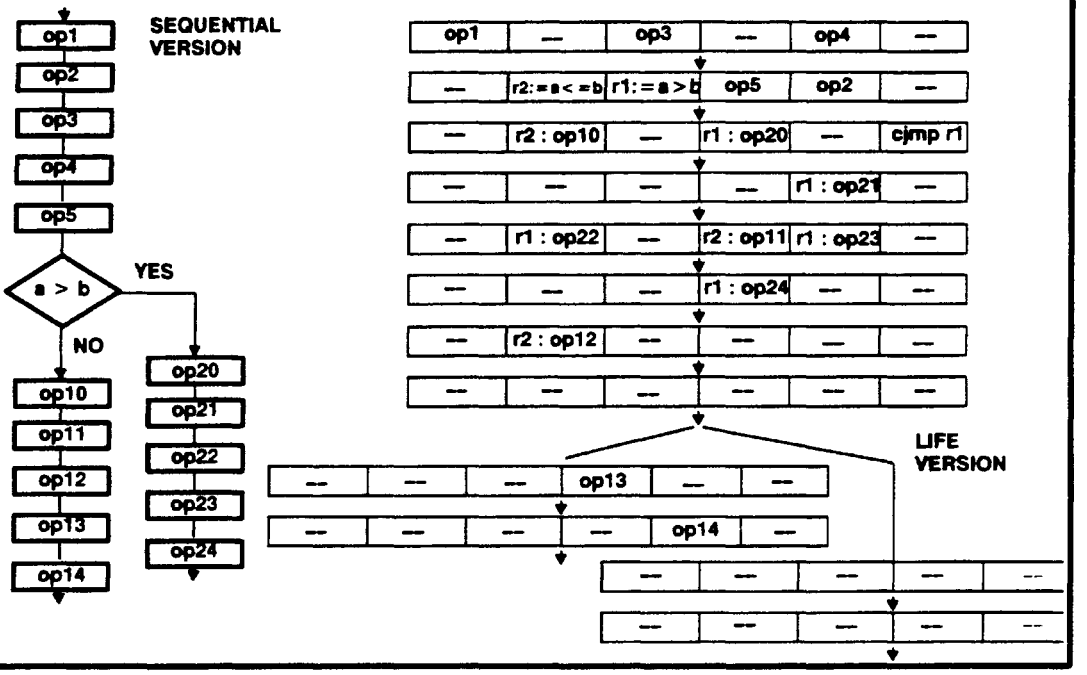
CONST	$-2^{31} .. 2^{31}-1$	
ALU	Arithmetic :	ADD*, SUBA*, SUBB* NEG*, ASR, ASL*
	Logical :	A, B, AND, OR, XOR NOT, ANOTB, BNOTA ROL, LSR
	Integer Comparison :	EQL, NEQ, LES, LEQ GTR, GEQ
	Boolean Comparison :	EQL, NEQ
	Boolean Translation :	LBOOL, CBOOL
REGISTER	NOP, READREG(i), WRITEREG(i)	
MEMORY	NOP, READ, WRITE	
BRANCH	NOP, JUMP, JTRUE, JFALSE	

* these operations can raise the overflow exception.

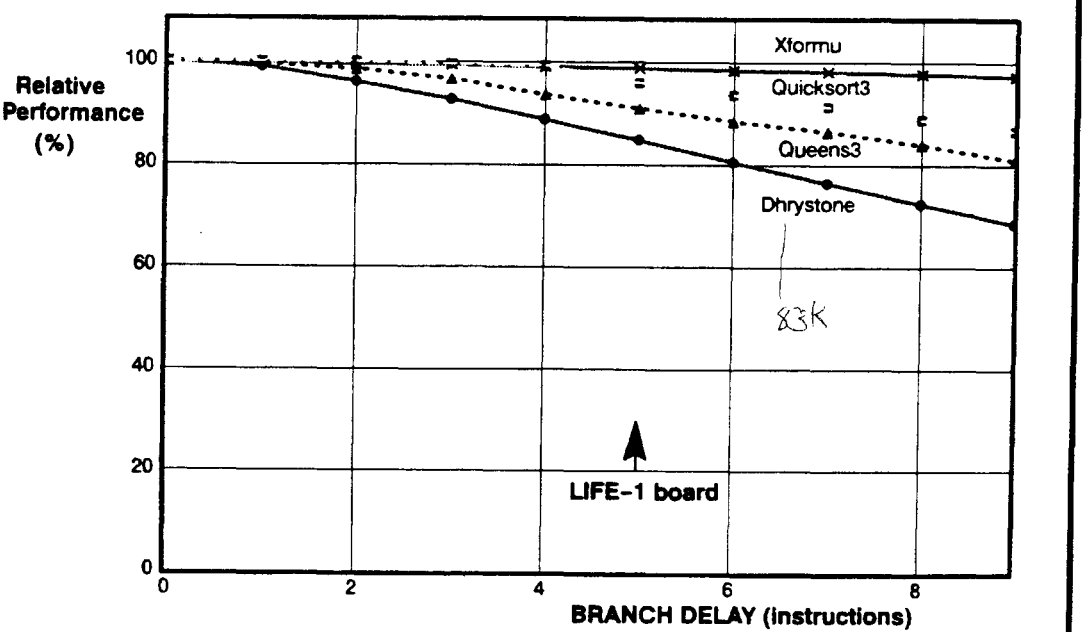
LIFE-1 chip photograph



Guarded operations



Performance = f(branchdelay)



Guarded operations

- ☆ Guarding is a form of *speculative execution*
- ☆ Guarding is a good compiletime/runtime trade-off :
  Compiler can statically schedule resources
  At runtime the scheduling decisions are limited
- ☆ Guarding makes an architecture less sensitive to the pipeline depth of the instruction issue subsystem
- ☆ The cost of guarding :
  4 % of LIFE-1 chip area
  25 % of LIFE-1 instruction bandwidth

LIFE-1 runtime statistics

Functional unit issue utilization :

CONST unit	**63 %**
ALU 1 & 2	**65 resp. 52 %**
REG unit	**49 %**
DMEM unit	**44 %**
BRANCH unit	**21 %**

Average operation issues/cycle : **2.94**

Percentage of issues actually used : **59 %**

Effective operation issues/cycle : **1.74**

Speedup (over single funcunit with 1 cycle latency) **2.63 x**

[Averaged over QUEENS3, QSORT3, QSORT3CA, DHRYREG, XFORMU, LOOPS1
5 cycle branchdelay, 3 cycle data memory load latency]

Conclusions

- LIFE-1 is a 60-100 VAX MIPS integer processor
- LIFE-1 is the prototype of a processor family
  (LIFE-1 is not available as an external product)
- We are currently working on customer specific processors using the LIFE concepts and compiler tools
- Target applications
  - Imaging
  - Graphics
  - Video processing
  - Multi media
  - Advanced Signal Processing
  - Robotics

A Challenge for the 90's

'What CPU architecture can reach a **sustained** performance of 200 MIPS and 500 MFLOPs on non–vectorizable code on a single chip'