
Instruction Set Principles

M. Sonza Reorda

Politecnico di Torino
Dipartimento di Automatica e Informatica

1

M. Sonza Reorda – Politecnico di Torino

Instruction Set Architecture

The *Instruction Set Architecture (ISA)* is how the computer is seen by the programmer or the compiler.

There are many alternatives possible for the ISA designer.

The different alternatives may be evaluated in terms of

- Processor performance
- Processor complexity
- Compiler complexity
- Code size
- Power consumption
- ...

Different product areas may assign different weights to the above parameters.

2

M. Sonza Reorda – Politecnico di Torino

Taxonomy

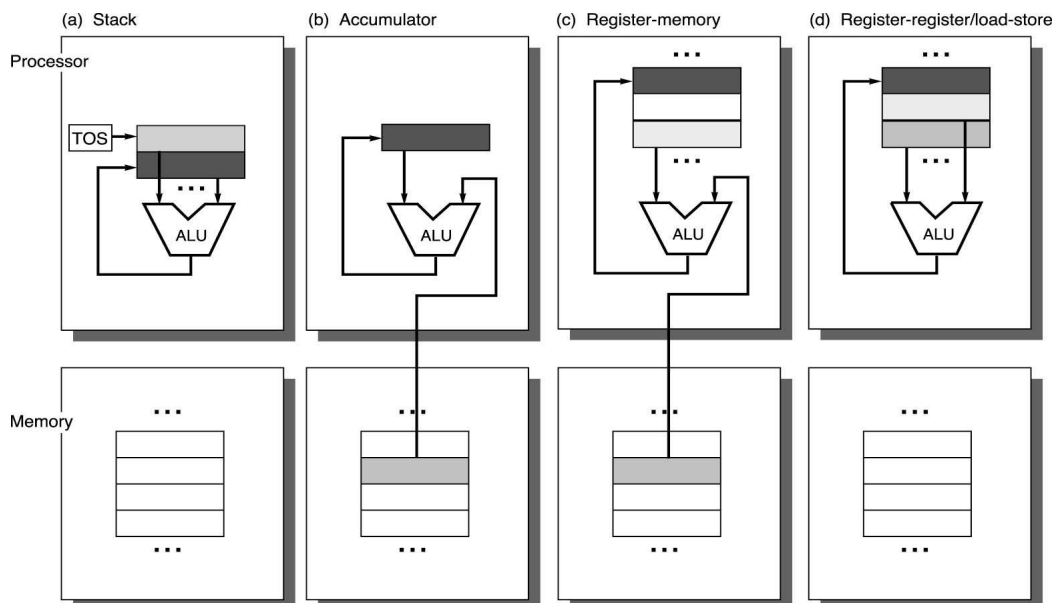
CPU's are often classified according to the type of their internal storage:

- stack
- accumulator
- registers:
 - register-memory
 - register-register (load-store)
 - memory-memory (no real cases).

3

M. Sonza Reorda – Politecnico di Torino

Architectures



Code example

Stack	Accumulator	Register (register-memory)	Register (load-store)
Push A	Load A	Load R1,A	Load R1,A
Push B	Add B	Add R1,B	Load R2,B
Add	Store C	Store C,R1	Add R3,R1,R2
Pop C			Store C,R3

The code sequence for $C = A + B$.

5

M. Sonza Reorda – Politecnico di Torino

GPR machines

Currently, all processors are *General-Purpose Register (GPR)* machines.

The reason for this is:

- registers are faster than memory
- registers are easier for a compiler to use.

6

M. Sonza Reorda – Politecnico di Torino

Operands per ALU instruction

CPUs can be classified according to

- typical number of operands per ALU instruction (2 or 3)
- typical number of memory operands per ALU instruction (from 0 to 3).

7

M. Sonza Reorda – Politecnico di Torino

Examples

Number of memory addresses	Maximum number of operands allowed	Examples
0	3	SPARC, MIPS, Precision Architecture, PowerPC, ALPHA
1	2	Intel 80x86, Motorola 68000
2	2	VAX (also has three-operand formats)
3	3	VAX (also has two-operand formats)

8

M. Sonza Reorda – Politecnico di Torino

Example: register-register architectures

mem. addr. \Rightarrow 0

operands \Rightarrow 3

**Alpha, ARM, MIPS, PowerPC, SPARC, SuperH, Trimedia
TM5200**

9

M. Sonza Reorda – Politecnico di Torino

Example: register-memory architectures

mem. addr. \Rightarrow 1

operands \Rightarrow 2

**IBM 360/370, Intel 80x86, Motorola 68000, TI
TMS320C54x**

10

M. Sonza Reorda – Politecnico di Torino

Example: memory-memory architectures

mem. addr. \Rightarrow 2/3

operands \Rightarrow 2/3

DEC \Rightarrow VAX

11

M. Sonza Reorda – Politecnico di Torino

Classification criteria

- **Memory addressing**
- Operations in the Instruction Set
- Type and Size of Operands
- Instruction Encoding.

12

M. Sonza Reorda – Politecnico di Torino

Memory Addresses

Alternatives:

- Little Endian vs. Big Endian
- Aligned vs. misaligned accesses.

13

M. Sonza Reorda – Politecnico di Torino

Memory Addresses

Alternatives:

- Little Endian vs. Big Endian
- Aligned vs. misaligned accesses.



The address of the data is that of the least significant byte.

14

M. Sonza Reorda – Politecnico di Torino

Memory Addresses

Alternatives:

- Little Endian vs. Big Endian
- Aligned vs. misaligned accesses.



The address of the data is that of the most significant byte.

15

M. Sonza Reorda – Politecnico di Torino

Memory Addresses

Alternatives:

- Little Endian vs. Big Endian
- Aligned vs. misaligned accesses.



Allowing only aligned accesses to memory is a limitation.

16

M. Sonza Reorda – Politecnico di Torino

Memory Addresses

Alternatives:

- Little Endian vs. Big Endian
- Aligned vs. misaligned accesses.



Allowing misaligned accesses to memory requires:

- hardware overhead
- performance overhead.

17

M. Sonza Reorda – Politecnico di Torino

Addressing modes

In GPR machines an addressing mode specifies a constant, a register, or a memory location (through its *effective address*).

18

M. Sonza Reorda – Politecnico di Torino

Register Mode

Addressing mode	Example instruction	Meaning	When used
Register	Add R4, R3	$\text{Regs}[R4] \leftarrow \text{Regs}[R4] + \text{Regs}[R3]$	When a value is in a register.

19

M. Sonza Reorda – Politecnico di Torino

Immediate Mode

Immediate	Add R4, #3	$\text{Regs}[R4] \leftarrow \text{Regs}[R4] + 3$	For constants.
-----------	------------	--	----------------

20

M. Sonza Reorda – Politecnico di Torino

Displacement Mode

Displacement	Add R4, 100(R1)	Regs[R4] ← Regs[R4] + Mem[100 + Regs[R1]]	Accessing local variables.
--------------	-----------------	---	----------------------------

21

M. Sonza Reorda – Politecnico di Torino

Register Deferred (or Indirect) Mode

Register deferred or indirect	Add R4, (R1)	Regs[R4] ← Regs[R4] + Mem[Regs[R1]]	Accessing using a pointer or a computed address.
-------------------------------	--------------	-------------------------------------	--

22

M. Sonza Reorda – Politecnico di Torino

Indexed Mode

Indexed	Add R3, (R1 + R2)	Regs[R3] ← Regs[R3] + Mem[Regs[R1] + Regs[R2]]	Sometimes useful in array addressing; R1 = base of array; R2 = index amount.
---------	-------------------	---	--

23

M. Sonza Reorda – Politecnico di Torino

Direct (or Absolute) Mode

Direct or absolute	Add R1, (1001)	Regs[R1] ← Regs[R1] + Mem[1001]	Sometimes useful for accessing static data; address constant may need to be large.
-----------------------	----------------	------------------------------------	--

24

M. Sonza Reorda – Politecnico di Torino

Memory Indirect (or Deferred) Mode

Memory indirect or memory deferred	Add R1, @(R3)	Regs[R1] ← Regs[R1] + Mem[Mem[Regs[R3]]]	If R3 is the address of a pointer p , then mode yields $*p$.
------------------------------------	---------------	--	---

25

M. Sonza Reorda – Politecnico di Torino

Autoincrement Mode

Autoincrement	Add R1, (R2)+	Regs[R1] ← Regs[R1] + Mem[Regs[R2]] Regs[R2] ← Regs[R2] + d	Useful for stepping through arrays within a loop. R2 points to start of array; each reference increments R2 by size of an element, d .
---------------	---------------	--	--

26

M. Sonza Reorda – Politecnico di Torino

Autodecrement Mode

Auto-decrement	Add R1, -(R2)	Regs[R2] ← Regs[R2] - d Regs[R1] ← Regs[R1] + Mem[Regs[R2]]	Same use as autoincrement. Autodecrement/increment can also act as push/ pop to implement a stack.
----------------	---------------	--	--

27

M. Sonza Reorda – Politecnico di Torino

Scaled Mode

Scaled	Add R1, 100(R2) [R3]	Regs[R1] ← Regs[R1] + Mem[100 + Regs[R2] + Regs[R3] * d]	Used to index arrays. May be applied to any indexed addressing mode in some machines.
--------	----------------------	--	---

28

M. Sonza Reorda – Politecnico di Torino

Choosing the addressing modes

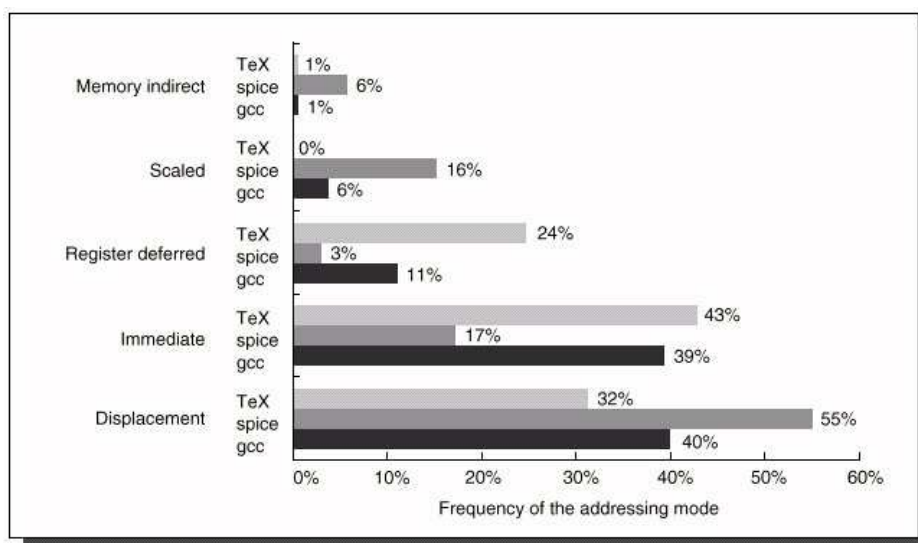
By carefully choosing the addressing modes, one can obtain some important consequences:

- Reducing the number of instructions
- Increasing the CPU architecture complexity
- Increasing the average CPI.

29

M. Sonza Reorda – Politecnico di Torino

Usage of addressing modes



30

M. Sonza Reorda – Politecnico di Torino

Open issue

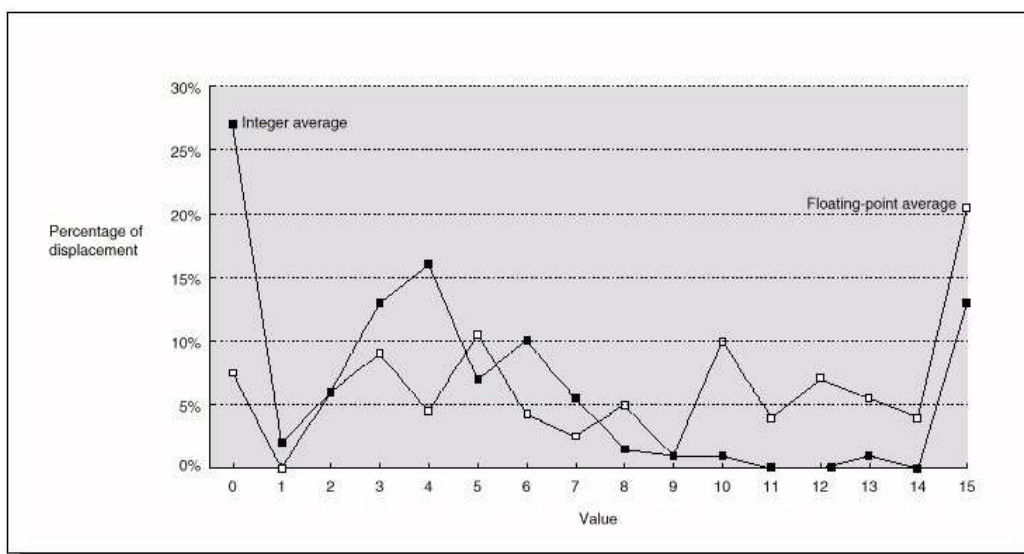
When the selected addressing requires a displacement, how many bits should be devoted to it in the instruction code?

An experimental evaluation can be performed.

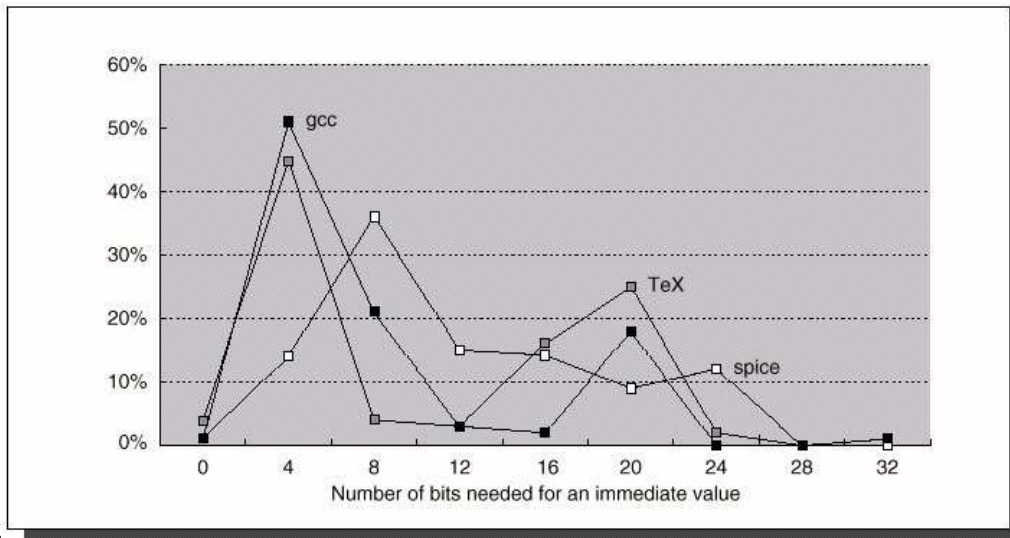
31

M. Sonza Reorda – Politecnico di Torino

Displacement values



Immediate values



Summary

- Displacement, immediate and register indirect modes represent from 75% to 99% of the addressing modes
- The address size for displacement mode should be from 12 to 16 bits (75% to 99% of the displacements)
- The size of the immediate field should be at least 8 or 16 bits (50% and 80% of the cases, respectively).

Classification criteria

- Memory addressing
- **Operations in the Instruction Set**
- Type and Size of Operands
- Instruction Encoding.

35

M. Sonza Reorda – Politecnico di Torino

Operations in the Instruction Set

Operator type	Examples
Arithmetic and logical	Integer arithmetic and logical operations: add, and, subtract, or
Data transfer	Loads-stores (move instructions on machines with memory addressing)
Control	Branch, jump, procedure call and return, traps
System	Operating system call, virtual memory management instructions
Floating point	Floating-point operations: add, multiply
Decimal	Decimal add, decimal multiply, decimal-to-character conversions
String	String move, string compare, string search
Graphics	Pixel operations, compression/decompression operations

36

M. Sonza Reorda – Politecnico di Torino

Making the Common Case Fast

Not all the instructions are executed with the same frequency!

When designing and implementing an instruction set, the most commonly executed instructions should be made faster.

37

M. Sonza Reorda – Politecnico di Torino

80x86 instruction frequency

Rank	80x86 instruction	Integer average (% total executed)
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
	Total	96%

38

Control Flow Instructions

They can be distinguished in four categories:

- conditional branches
- jumps
- procedure calls
- procedure returns.

Conditional branches are by far the most frequently executed control flow instructions.

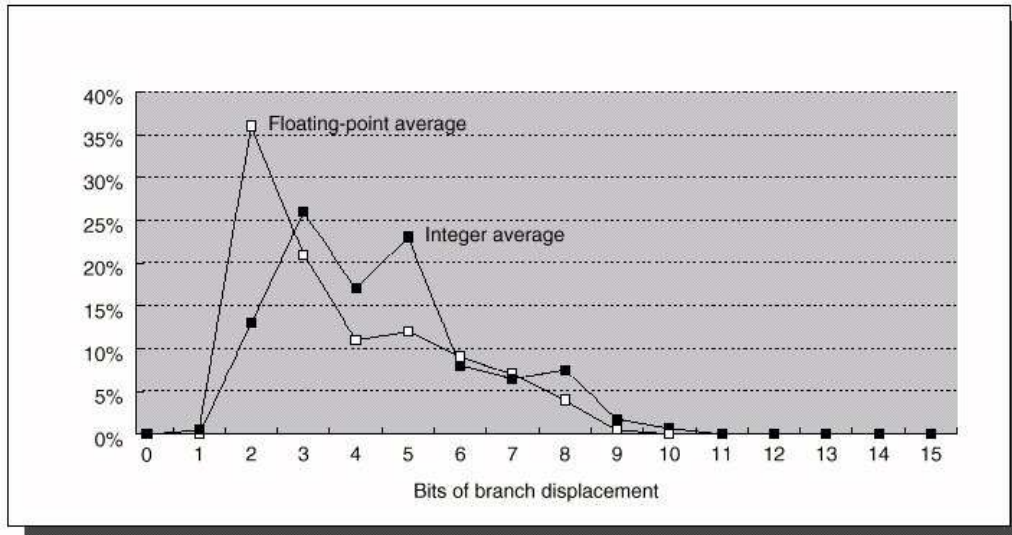
The destination address

It is normally specified as a displacement with respect to the current value of the Program Counter.

In this way:

- we save bits, since the target instruction is often close to the source one
- the code is position-independent.

Branch distances



Register Indirect Jumps and Procedure Calls

They allow:

- to write code including jumps whose target is not known at compile time
- to implement *case* or *switch* statements
- to support *dynamically shared libraries* (i.e., libraries which are loaded only when called)
- to support *virtual functions* (i.e., calling different functions depending on the data type).

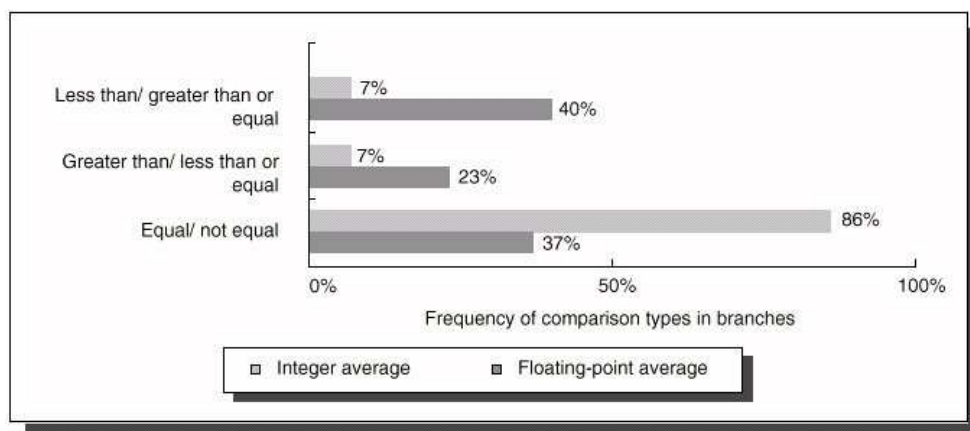
Evaluating Branch Conditions

Name	How condition is tested	Advantages	Disadvantages
Condition code (CC)	Special bits are set by ALU operations, possibly under program control.	Sometimes condition is set for free.	CC is extra state. Condition codes constrain the ordering of instructions since they pass information from one instruction to a branch.
Condition register	Test arbitrary register with the result of a comparison.	Simple.	Uses up a register.
Compare and branch	Compare is part of the branch. Often compare is limited to subset.	One instruction rather than two for a branch.	May be too much work per instruction.

43

M. Sonza Reorda – Politecnico di Torino

Conditional branches



44

M. Sonza Reorda – Politecnico di Torino

Procedures

Some information need to be saved:

- the return address
- the accessed registers:
 - caller saving
 - callee saving.

45

M. Sonza Reorda – Politecnico di Torino

Summary

- Few instructions are responsible for most of the execution time: load, store, add, subtract, move register-register, and, shift, compare equal, compare not equal, branch, jump, call, and return.
- PC-relative branch displacements of at least 8 bits are the best choice.
- Register-indirect and PC-relative addressing can also be used in procedure call and return.

46

M. Sonza Reorda – Politecnico di Torino

Classification criteria

- Memory addressing
- Operations in the Instruction Set
- **Type and Size of Operands**
- Instruction Encoding.

47

M. Sonza Reorda – Politecnico di Torino

Type and Size of Operands

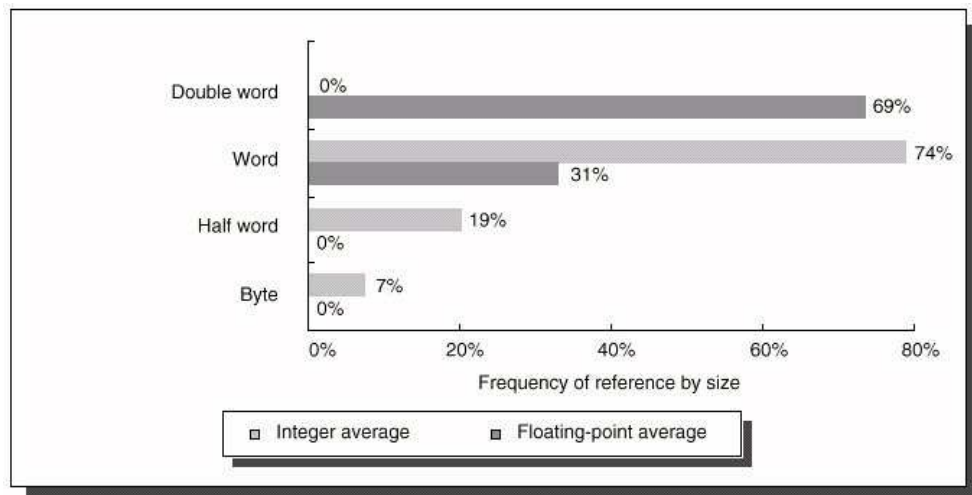
Most frequently supported data types:

- char (1 byte)
- half word (2 bytes)
- word (4 bytes)
- single-precision floating-point (4 bytes)
- double-precision floating-point (8 bytes).

48

M. Sonza Reorda – Politecnico di Torino

Distribution of Data Accesses by Size



49

M. Sonza Reorda – Politecnico di Torino

Classification criteria

- Memory addressing
- Operations in the Instruction Set
- Type and Size of Operands
- **Instruction Encoding.**

50

M. Sonza Reorda – Politecnico di Torino

Instruction Set Encoding

Instruction Set Encoding depends on

- which instructions compose the Instruction Set
- which addressing modes are supported.

When a high number of addressing modes is supported, an *address specifier* field is used to specify the addressing mode and the registers which are possibly involved.

When the number of addressing modes is low, they can be encoded together with the opcode.

51

M. Sonza Reorda – Politecnico di Torino

Conflicting Issues

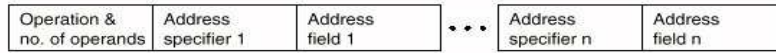
The designer should address several conflicting issues:

- the code size
- the size of the Instruction Set, the number of addressing modes, and the number of registers
- the complexity of the fetch and decoding hardware.

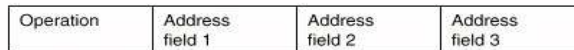
52

M. Sonza Reorda – Politecnico di Torino

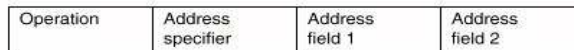
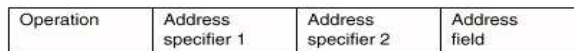
Instruction Set Encoding



(a) Variable (e.g., VAX)



(b) Fixed (e.g., DLX, MIPS, Power PC, Precision Architecture, SPARC)



(c) Hybrid (e.g., IBM 360/70, Intel 80x86)

53

Instruction Set Encoding



(a) Variable (e.g., VAX)



(b) Fixed (e.g., DLX, MIPS, Power PC, Precision Architecture, SPARC)

(c) Hybrid (e.g., IBM 360/70, Intel 80x86)

- Supports any number of operands
- Instructions have a variable length
- Lower performance
- Minimum code size

54

Instruction Set Encoding

Operation & no. of operands	Address specifier 1	Address field 1	...	Address specifier n	Address field n
-----------------------------	---------------------	-----------------	-----	---------------------	-----------------

(a) Variable (e.g., VAX)

Operation	Address field 1	Address field 2	Address field 3
-----------	-----------------	-----------------	-----------------

(b) Fixed (e.g., DLX, MIPS, Power PC, Precision Architecture, SPARC)

Operation

Oper

Oper

(c) Hybrid

- Fixed number of operands
- Address specifier included in the opcode
- Fixed instruction length
- Maximum performance
- Larger code size

55

Instruction Set Encoding

Op
no

(a)

- Multiple formats (specified by the opcode)
- Allow trading-off between code size and performance

Operation	Address field 1	Address field 3
-----------	-----------------	-----------------

(b) Fixed (e.g., DLX, MIPS, Power PC, Precision Architecture, SPARC)

Operation	Address specifier	Address field
-----------	-------------------	---------------

Operation	Address specifier 1	Address specifier 2	Address field
-----------	---------------------	---------------------	---------------

Operation	Address specifier	Address field 1	Address field 2
-----------	-------------------	-----------------	-----------------

(c) Hybrid (e.g., IBM 360/70, Intel 80x86)

56

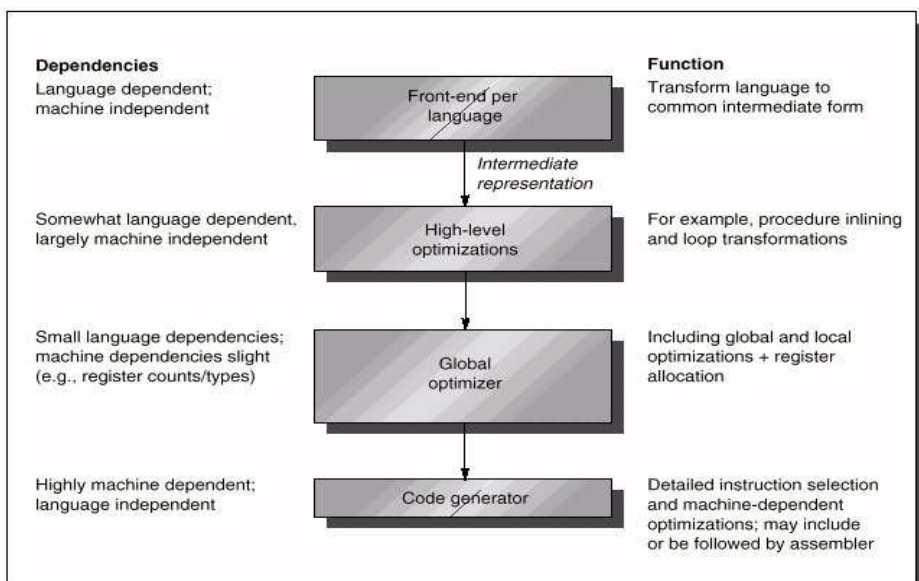
Hardware-Compiler interaction

- Assembly-level programs are now produced by compilers, only
- The CPU designer and the compiler writer must interact and cooperate.

57

M. Sonza Reorda – Politecnico di Torino

Compiler structure



58

Register allocation

Choosing which variables have to be put in which registers and when is one of the crucial optimization phases in a compiler.

This problem is based on *graph* coloring and can be better solved if the number of registers is higher (>16).

59

M. Sonza Reorda – Politecnico di Torino

Variables access

Optimizing variable access time by allocating variables to registers is only possible for those stored in the stack or for global variables in memory.

It is impossible for variables belonging to the heap, due to the *aliasing problem* (i.e., the access to a variable is done through pointers).

60

M. Sonza Reorda – Politecnico di Torino

How the Architect can Help the Compiler Writer

Make the frequent case fast and the rare case correct

- Regularity
- Provide primitives, not solutions
- Simplify trade-offs among alternatives
- Provide instructions that bind the quantities known at compile time as constants.

61

M. Sonza Reorda – Politecnico di Torino

Recommendations

- At least 16 registers
- Orthogonality
- Simplicity.

62

M. Sonza Reorda – Politecnico di Torino