ELE 655
Microprocessor System Design

Class 3 – ISA

Instruct Set Architecture

- 3 applications areas
  - Desktop
    - Performance is key
    - Code size is no longer a concern
    - → both integer and floating point instructions
  - Servers
    - Focus is on database, file server and WEB applications
    - Integers and characters are primary data types
    - → little emphasis on floating point
  - Mobile / Embedded
    - Power and code size are critical
    - Floating point is optional

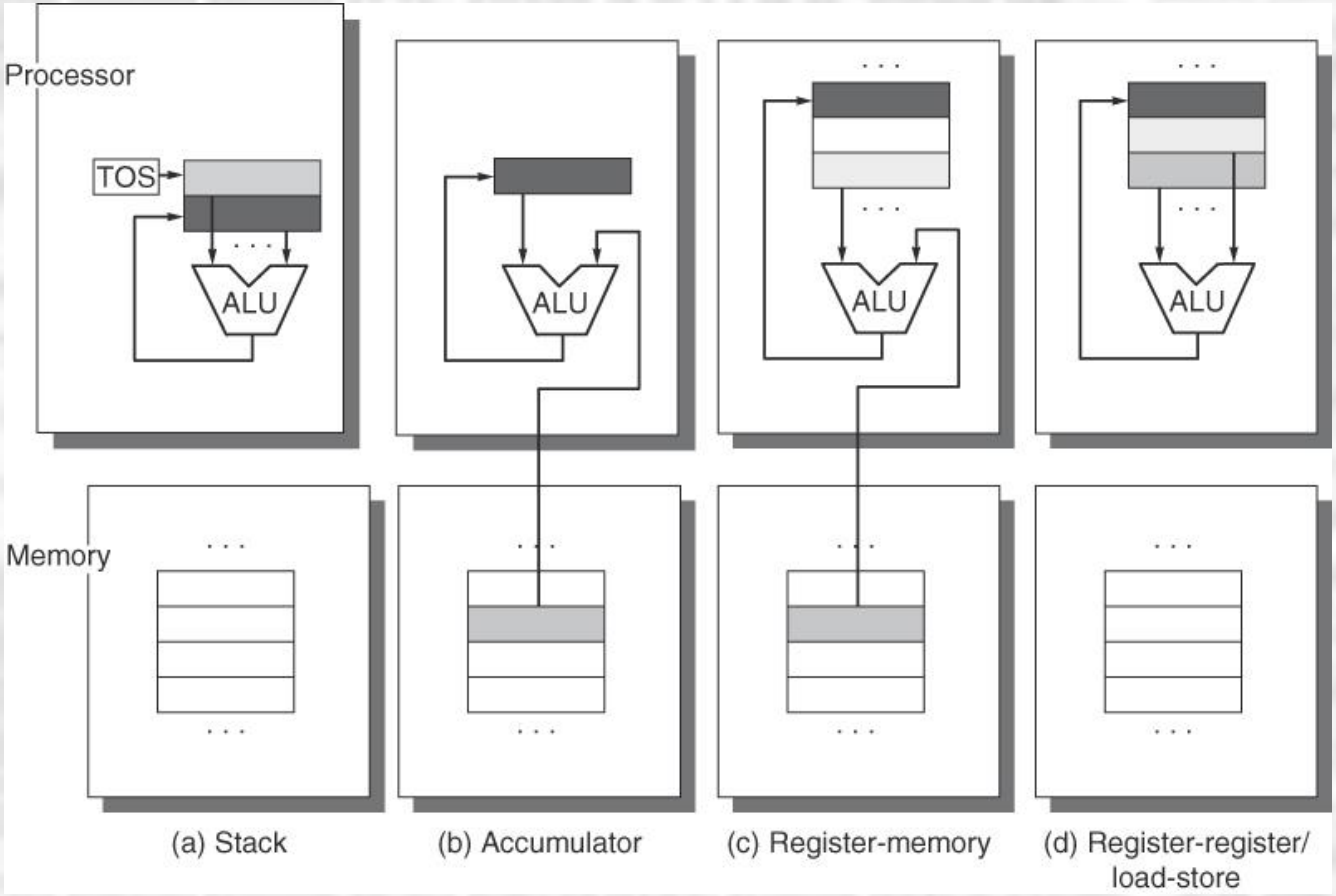


Figure A.1 Ope operand is an input or the result of t inputs, and the dark shade indicates combined with the operand below. d TOS is updated to point to the resu ult. In (c), one input operand is a re and, like the stack architecture, car r (d).

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

Instruct Set Architecture

- GPR architectures dominate
  - Registers are fast
    - Local to processor
    - Logic based instead of memory based
  - Registers enhance compiler options
    - Reorder calculations
      - Not possible on a stack based machine

Instruct Set Architecture

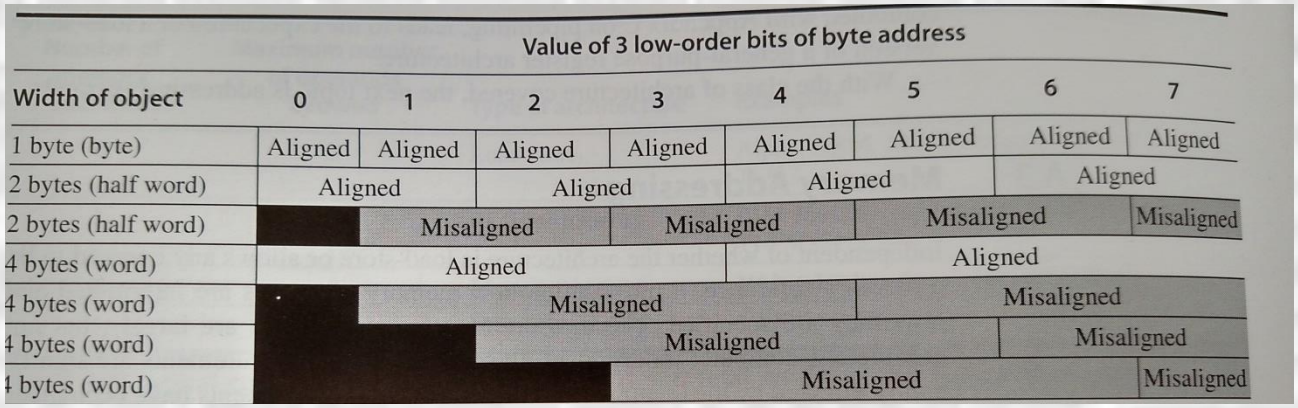
- Memory as an operand

Number of memory addresses	Maximum number of operands allowed	Type of architecture	Examples
0	3	Load-store	Alpha, ARM, MIPS, PowerPC, SPARC, SuperH, TM32
1	2	Register-memory	IBM 360/370, Intel 80x86, Motorola 68000, TI TMS320C54x
2	2	Memory-memory	VAX (also has three-operand formats)
3	3	Memory-memory	VAX (also has two-operand formats)

Type	Advantages	Disadvantages
Register-register (0, 3)	Simple, fixed-length instruction encoding. Simple code generation model. Instructions take similar numbers of clocks to execute (see Appendix C).	Higher instruction count than architectures with memory references in instructions. More instructions and lower instruction density lead to larger programs.
Register-memory (1, 2)	Data can be accessed without a separate load instruction first. Instruction format tends to be easy to encode and yields good density.	Operands are not equivalent since a source operand in a binary operation is destroyed. Encoding a register number and a memory address in each instruction may restrict the number of registers. Clocks per instruction vary by operand location.
Memory-memory (2, 2) or (3, 3)	Most compact. Doesn't waste registers for temporaries.	Large variation in instruction size, especially for three-operand instructions. In addition, large variation in work per instruction. Memory accesses create memory bottleneck. (Not used today.)

Instruct Set Architecture

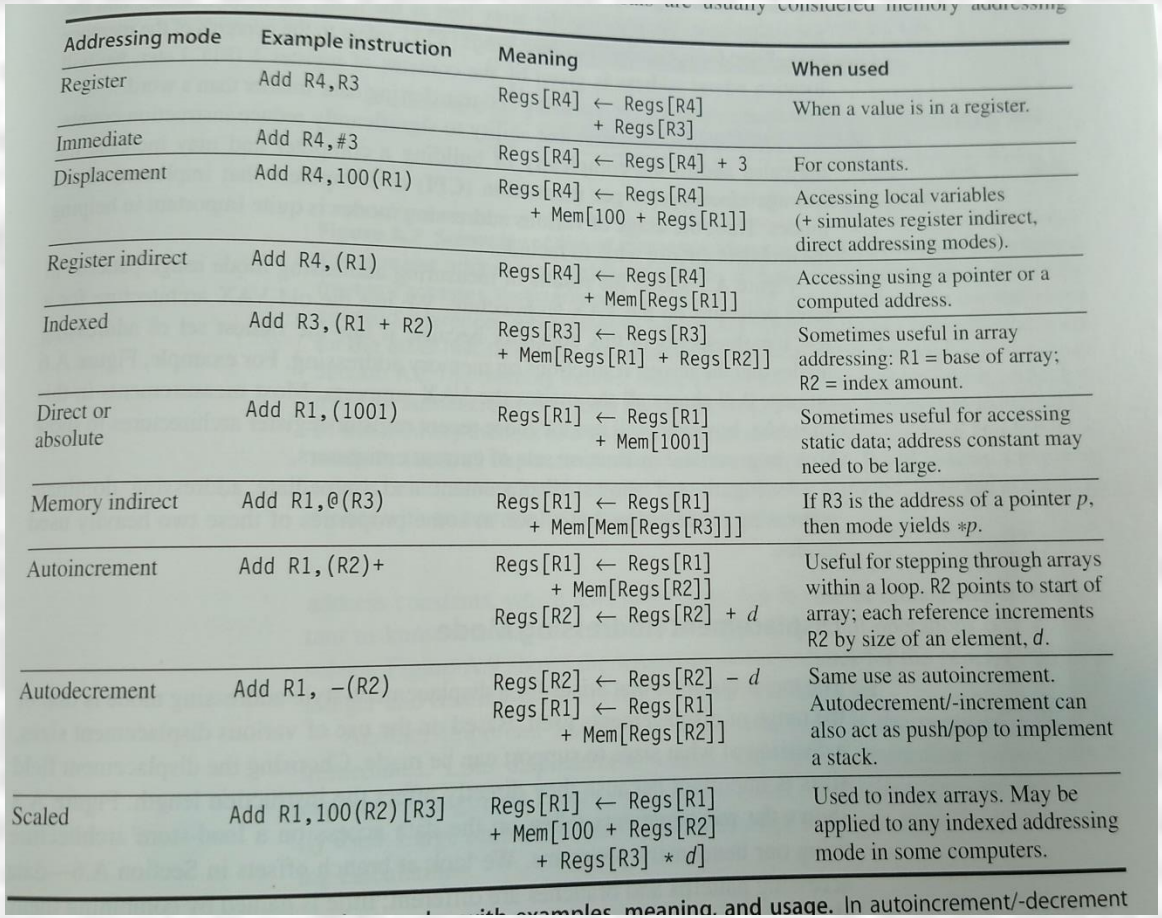
- Interpreting Memory Addresses
  - Big vs Little Endian
    - SDRAEKCAB
  - Byte Addressed
  - Byte / word aligned



	Value of 3 low-order bits of byte address							
Width of object	0	1	2	3	4	5	6	7
1 byte (byte)	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned
2 bytes (half word)	Aligned		Aligned		Aligned		Aligned	
2 bytes (half word)		Misaligned		Misaligned		Misaligned		Misaligned
4 bytes (word)	Aligned				Aligned			
4 bytes (word)		Misaligned				Misaligned		
4 bytes (word)			Misaligned				Misaligned	
4 bytes (word)				Misaligned				Misaligned

Instruct Set Architecture

- Addressing Modes

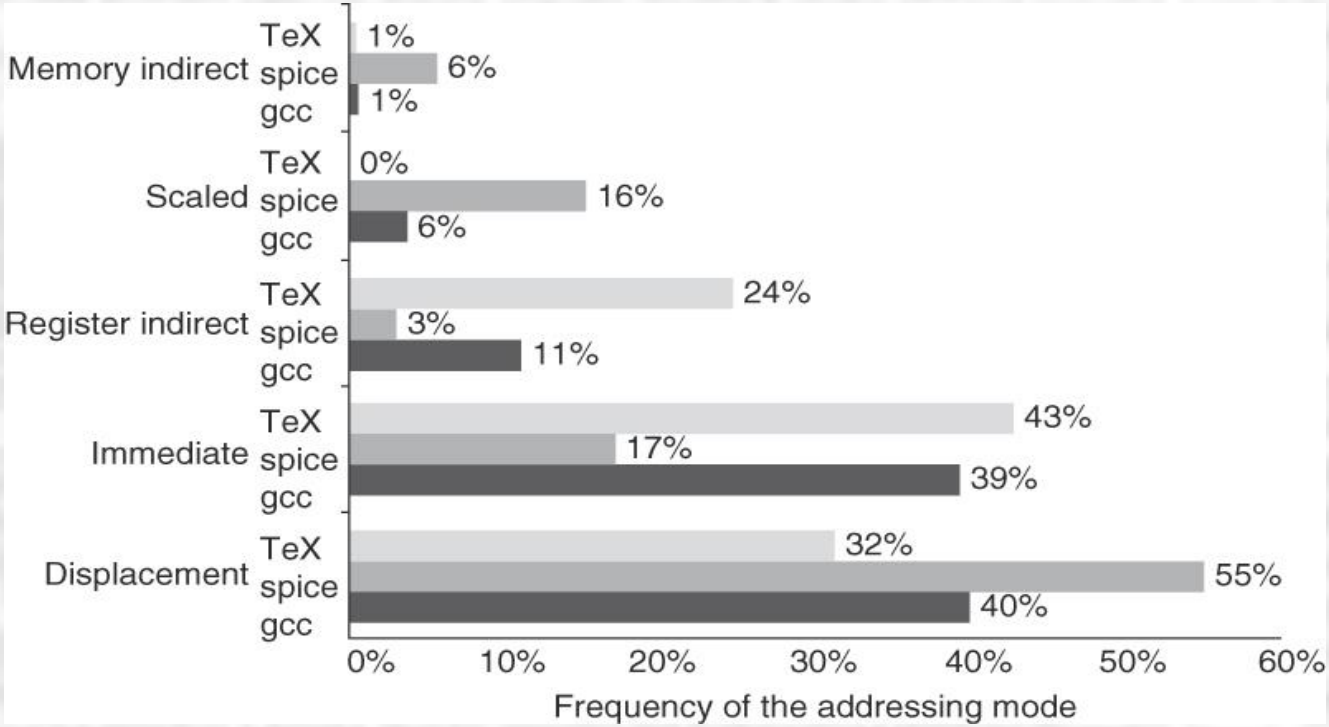


Addressing mode	Example instruction	Meaning	When used
Register	Add R4,R3	Regs[R4] ← Regs[R4] + Regs[R3]	When a value is in a register.
Immediate	Add R4,#3	Regs[R4] ← Regs[R4] + 3	For constants.
Displacement	Add R4,100(R1)	Regs[R4] ← Regs[R4] + Mem[100 + Regs[R1]]	Accessing local variables (+ simulates register indirect, direct addressing modes).
Register indirect	Add R4,(R1)	Regs[R4] ← Regs[R4] + Mem[Regs[R1]]	Accessing using a pointer or a computed address.
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.
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.
Memory indirect	Add R1,@(R3)	Regs[R1] ← Regs[R1] + Mem[Mem[Regs[R3]]]	If R3 is the address of a pointer p, then mode yields $*p$.
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.
Autodecrement	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.
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 computers.

...with examples, meaning, and usage. In autoincrement/-decrement

Instruct Set Architecture

- Addressing Modes

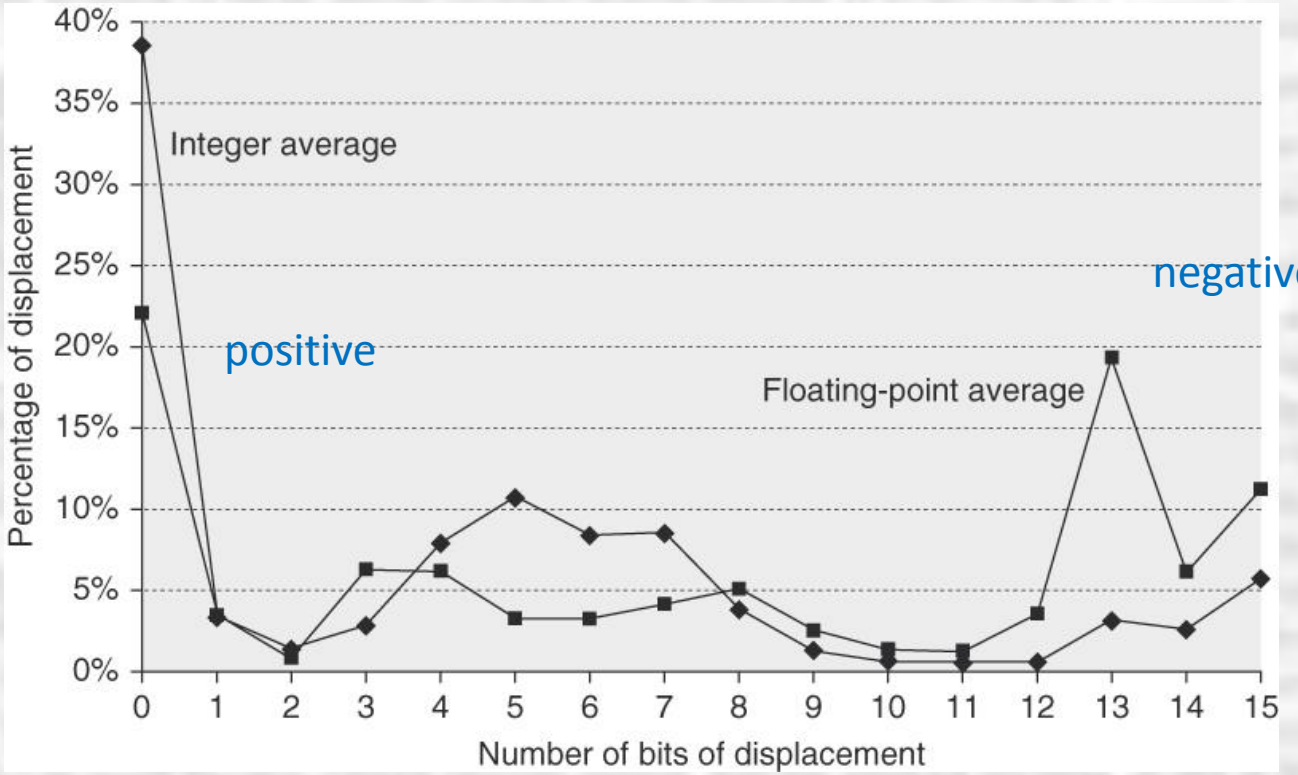


Instruct Set Architecture

- Addressing Modes

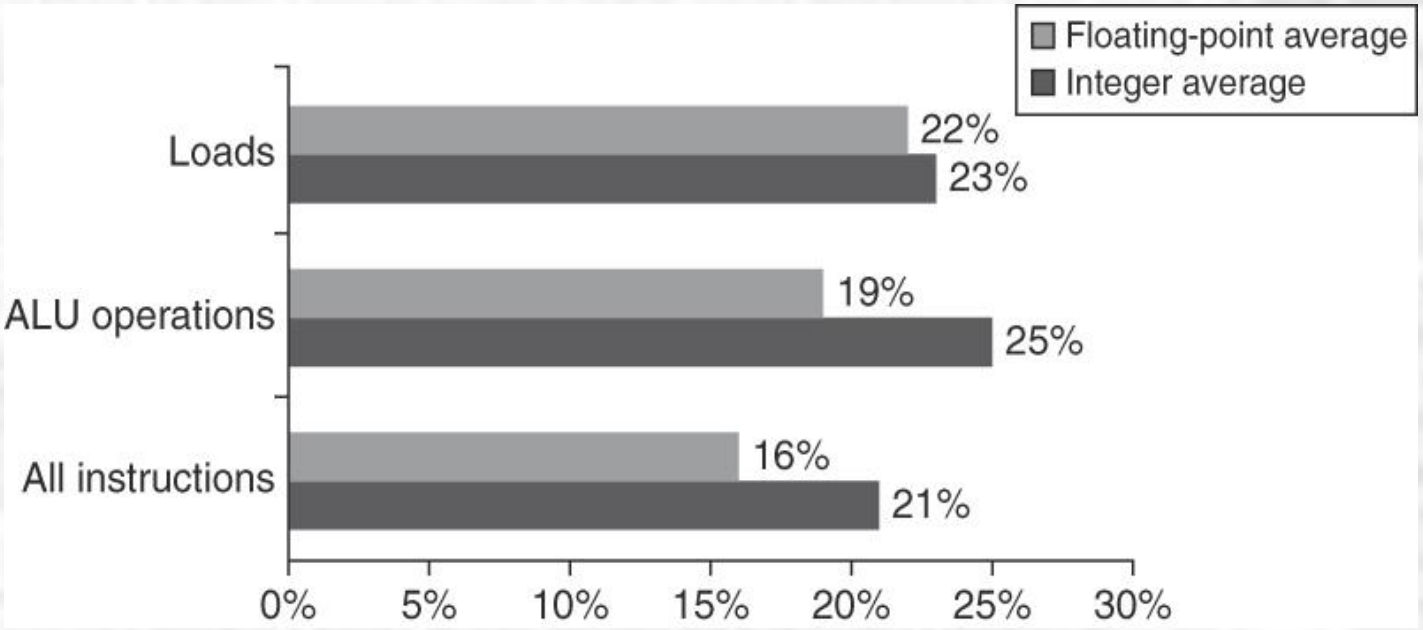
What does a displacement of 0 mean?

  - Displacement – How big?



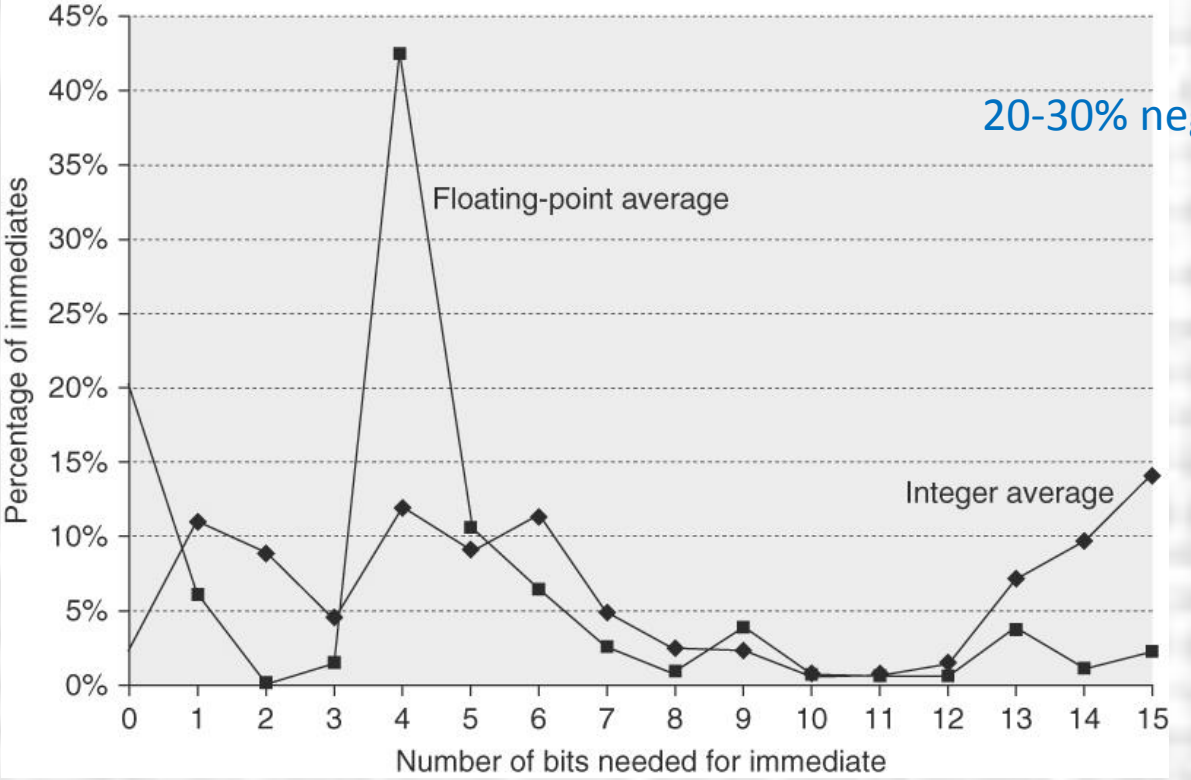
Instruct Set Architecture

- Addressing Modes
  - Immediates – how often?



Instruct Set Architecture

- Addressing Modes
  - Immediates – how big?



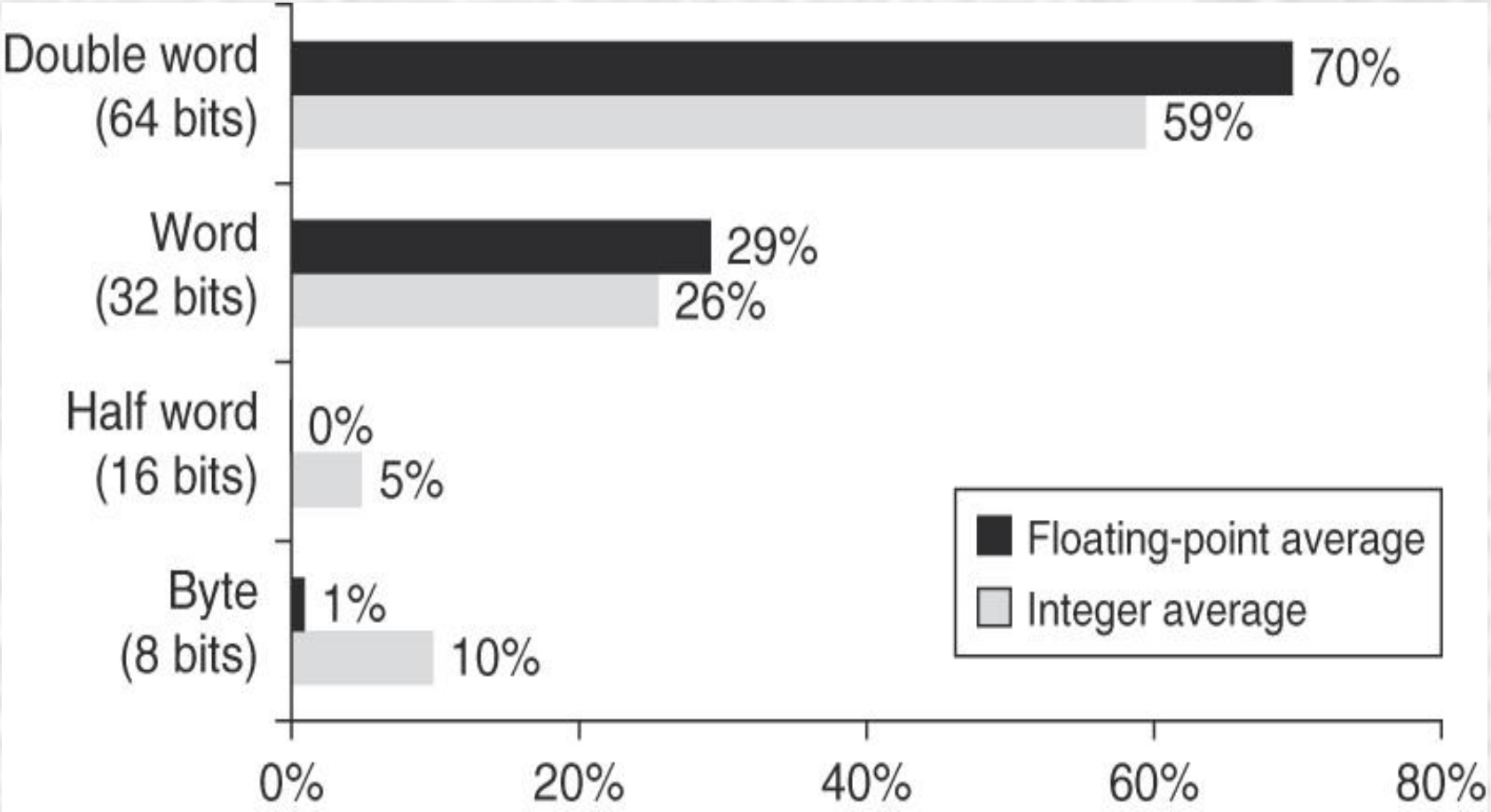
20-30% negative

Instruct Set Architecture

- Operands
  - Integers
    - 2's compliment
    - Byte(8), half word(16), word(32), doubleword(64)
  - Floating Point
    - IEEE 754 standard
    - Single precision (32), double precision (64)
  - Characters
    - ASCII – 8 bits
    - Unicode – 16 bits
  - BCD

Instruct Set Architecture

- Operands



Instruct Set Architecture

- Operations

Operator type	Examples
Arithmetic and logical	Integer arithmetic and logical operations: add, subtract, and, or, multiply, divide
Data transfer	Loads-stores (move instructions on computers 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, divide, compare
Decimal	Decimal add, decimal multiply, decimal-to-character conversions
String	String move, string compare, string search
Graphics	Pixel and vertex operations, compression/decompression operations

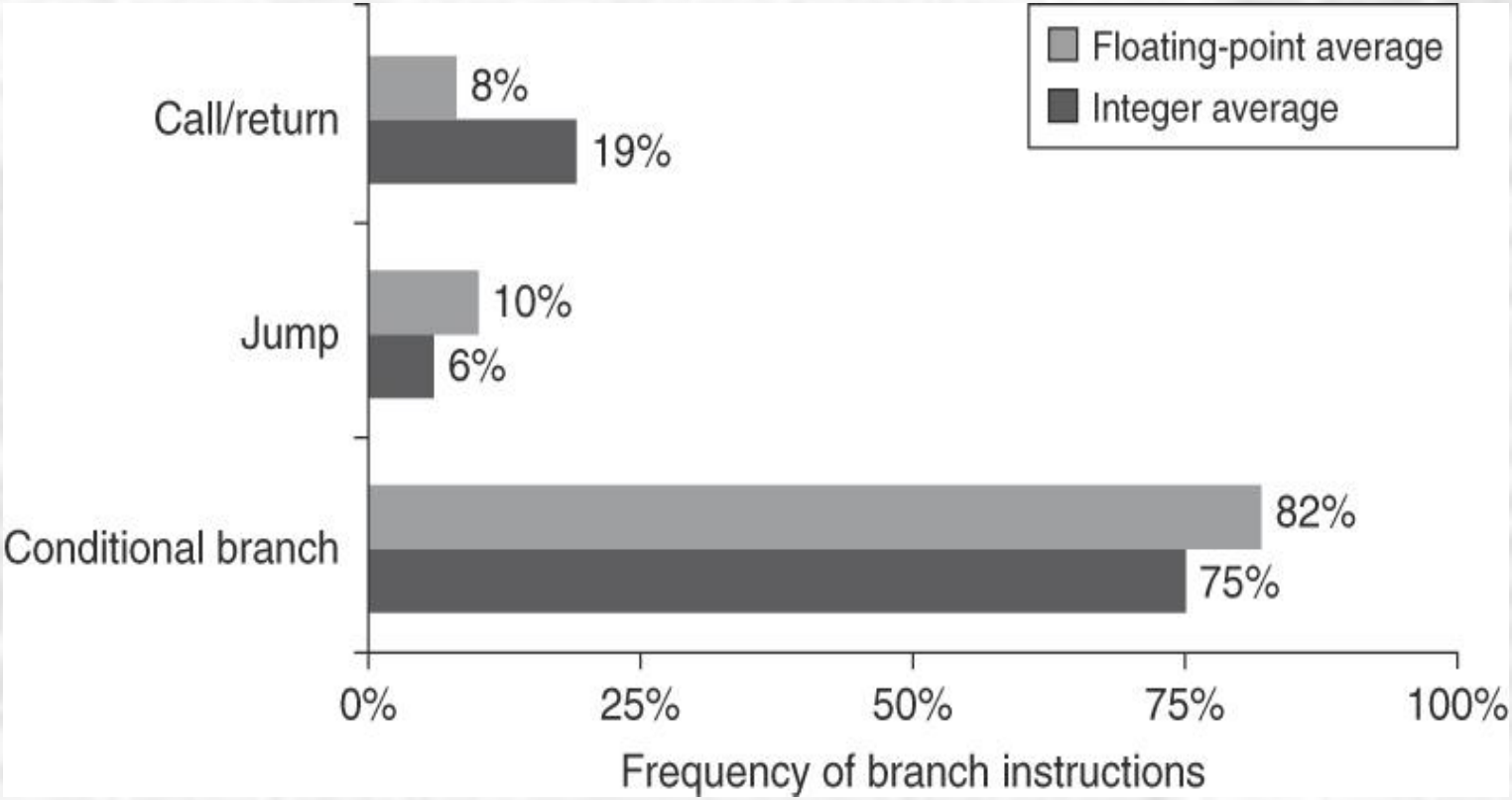
Instruct Set Architecture

- Operations

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%**

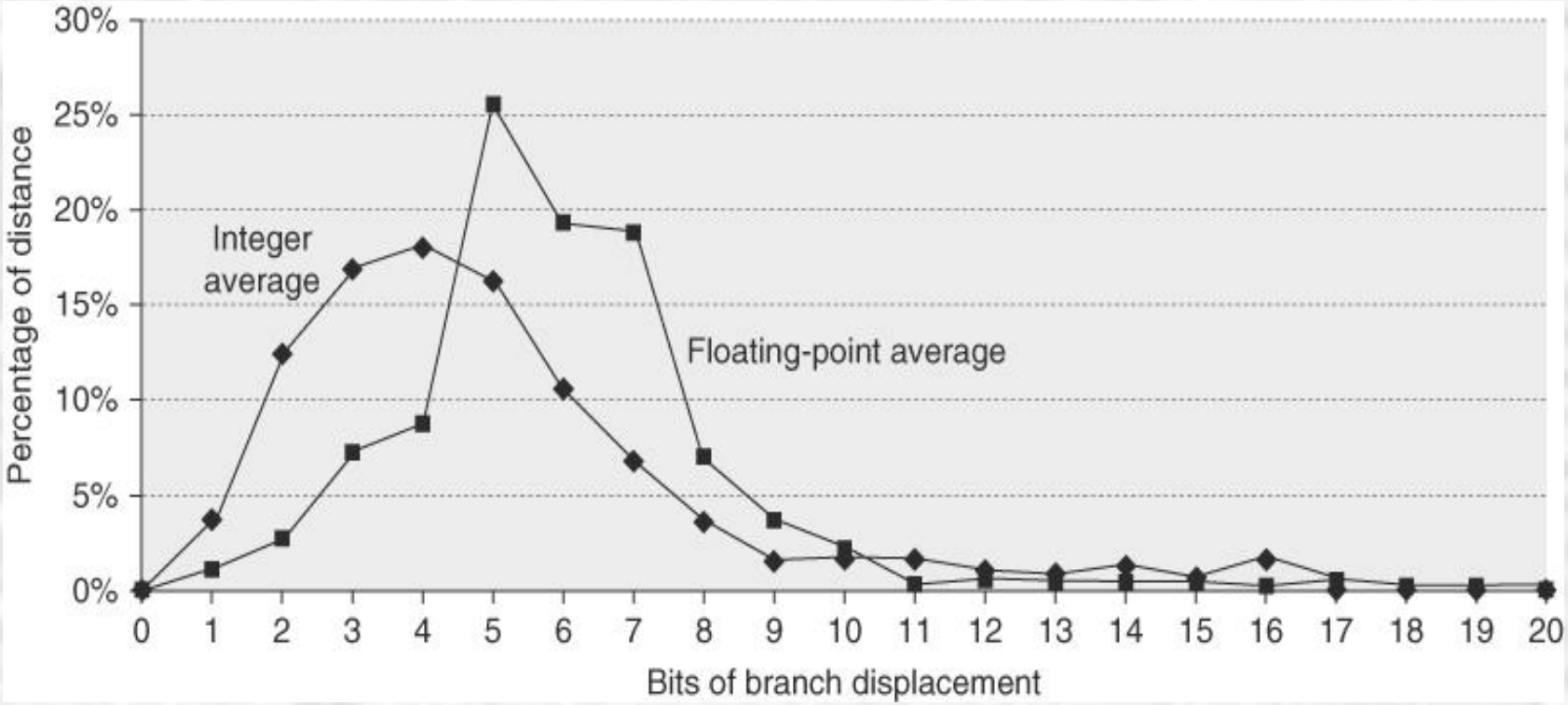
Instruct Set Architecture

- Operations – control flow



Instruct Set Architecture

- Operations – control flow
  - Branches addresses are known at compile time – PC relative



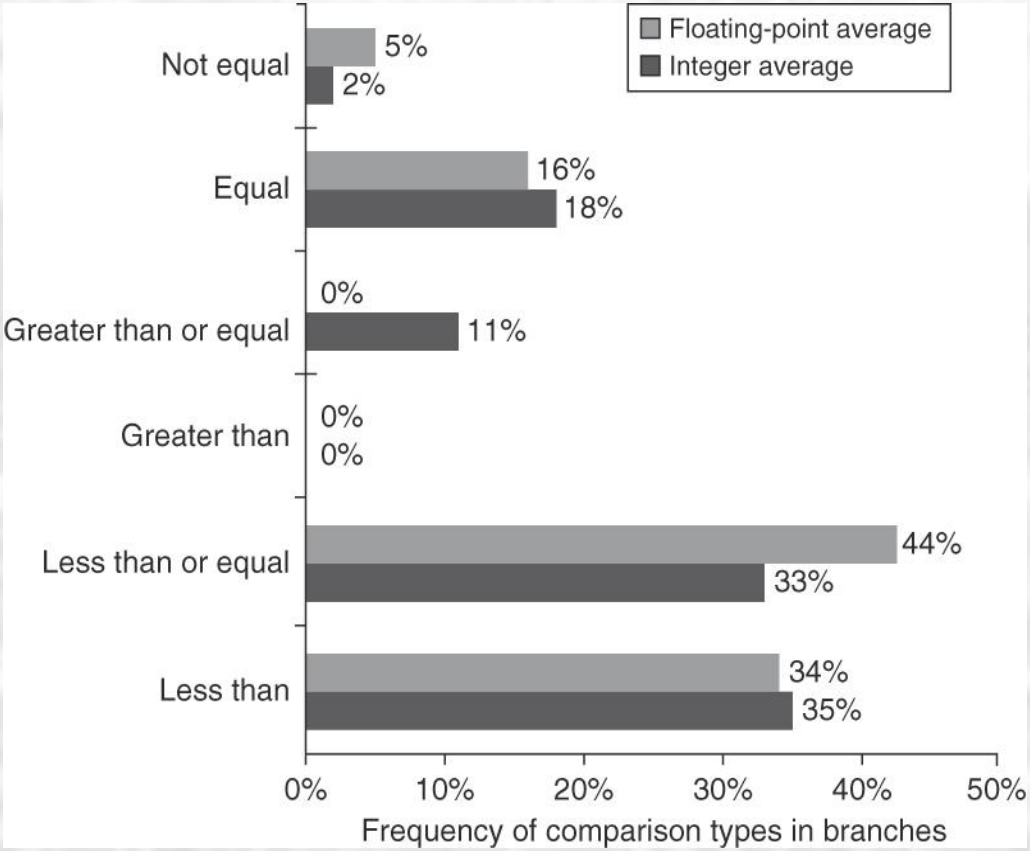
Instruct Set Architecture

- Operations – condition testing

Name	Examples	How condition is tested	Advantages	Disadvantages
Condition code (CC)	80x86, ARM, PowerPC, SPARC, SuperH	Tests special bits 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	Alpha, MIPS	Tests arbitrary register with the result of a comparison.	Simple.	Uses up a register.
Compare and branch	PA-RISC, VAX	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 for pipelined execution.

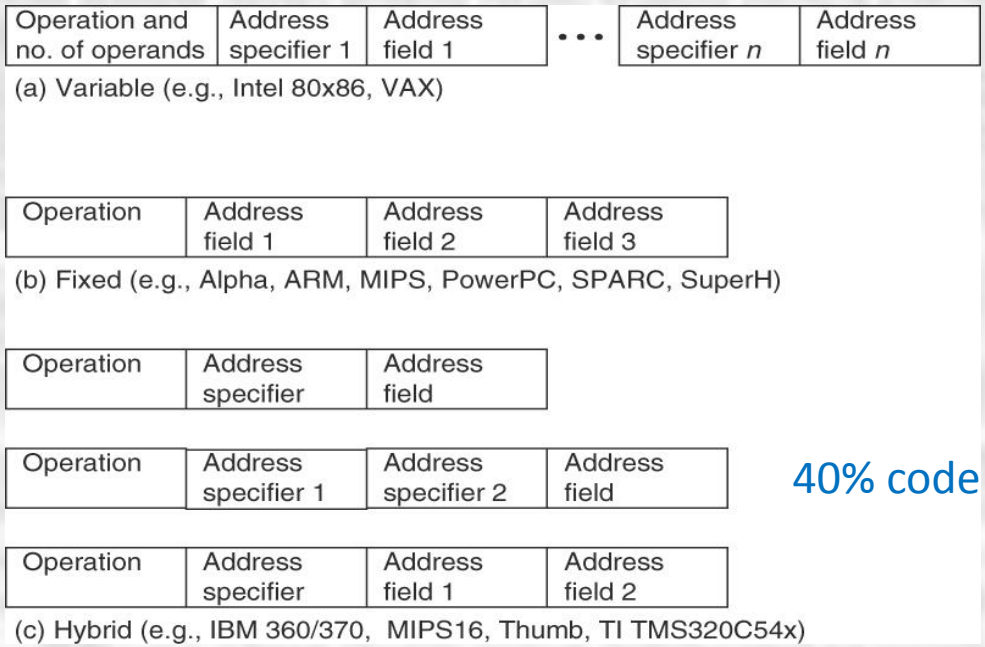
Instruct Set Architecture

- Operations – condition testing



Instruct Set Architecture

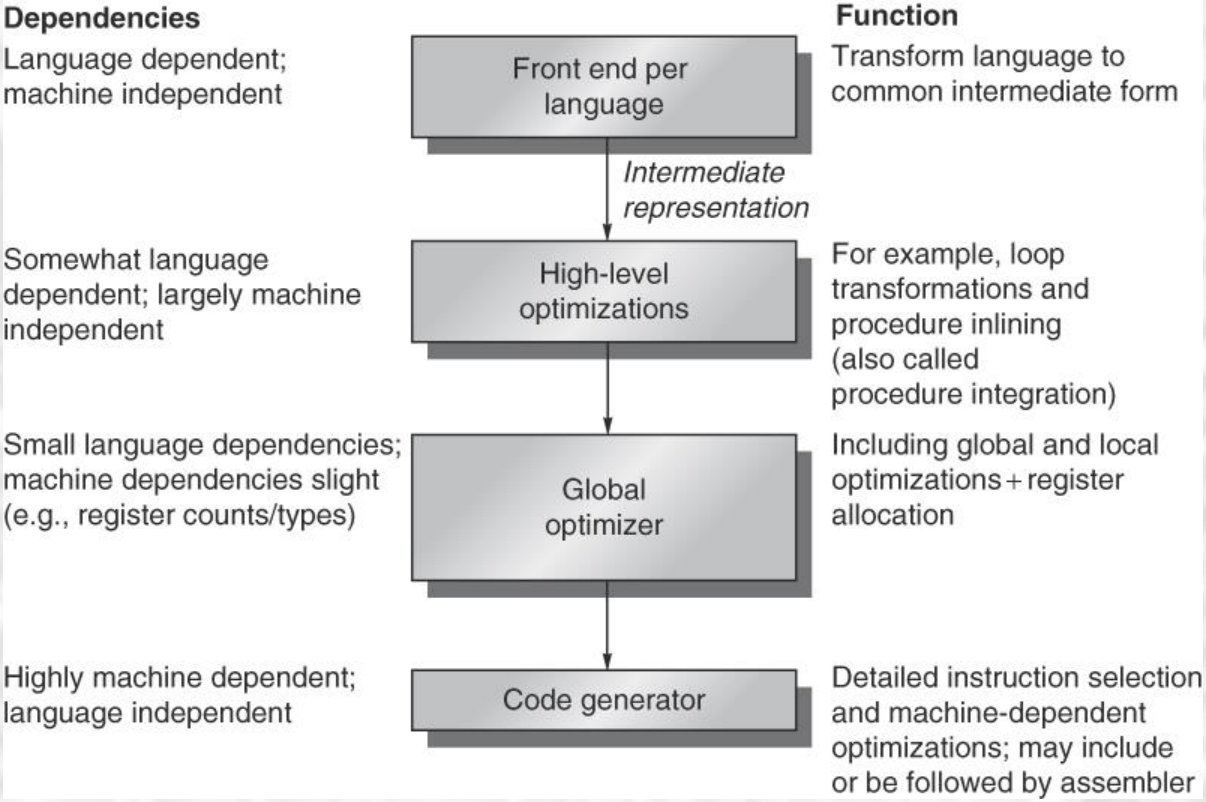
- Instruction Encoding
  - Flexibility
  - Code Size
  - Implementation of decode



40% code size reduction

Instruct Set Architecture

- Compilers



Instruct Set Architecture

- A.5
- A.8
- A.10
- A.11
- A.22

Instruct Set Architecture

- A.5

- A = B + C ;The operands here are given, not computed
- by the code, so copy propagation will not
- transform this statement.
- 2. B = A + C ;Here A is a computed value, so transform
- the code by substituting
- A = B + C to get
- = B + C + C ;Now no operand is computed
- 3. D = A – B ;Both operands are computed so substitute
- for both to get
- = (B + C) – (B + C + C) ;Simplify algebraically to get
- = – C ;This is a given, not computed, operand

Instruct Set Architecture

- A.8

	addr[11:10]	addr[9:5]	addr[4:0]
3 two-address instr.	'00', '01', '10'	'00000' to '11111'	'00000' to '11111'
Other instructions	'11'	'00000' to '11111'	'00000' to '11111'

	addr[11:10]	addr[9:5]	addr[4:0]
3 two-address instr.	'00', '01', '10'	'00000' to '11111'	'00000' to '11111'
30 one-address instr.	'11'	'00000' to '11101'	'00000' to '11111'
45 zero-address instr.	'11'	'11110'	'00000' to '11111'
	'11'	'11111'	'00000' to '01100'

	addr[11:10]	addr[9:5]	addr[4:0]
3 two-address instr.	'00', '01', '10'	'00000' to '11111'	'00000' to '11111'
24 zero-address instr.	'11'	'00000'	'00000' to '10111'
X one-address instr.	'11'	'00001' to '11111'	'00000' to '11111'

Instruct Set Architecture

- A.10

Reasons to increase the number of registers include:
1. Greater freedom to employ compilation techniques that consume registers, such as loop unrolling, common subexpression elimination, and avoiding name dependences.
2. More locations that can hold values to pass to subroutines.
3. Reduced need to store and re-load values.

Reasons not to increase the number of registers include:
1. More bits needed to represent a register name, thus increasing the overall size of an instruction or reducing the size of some other field(s) in the instruction.
2. More CPU state to save in the event of an exception.
3. Increased chip area and increased power consumption.

Instruct Set Architecture

- A.11
- 44 → 40
- 56 → 48

Data type	Data size on 32-bit machine (bytes)	Data size on 64-bit machine (bytes)
char	1	1
bool	1	1
int	4	4
long	4	8
double	8	8
short	2	2
float	4	4
pointer	4	8

Instruct Set Architecture

- A.22

a.

43	4F	4D	50	55	54	45	52
C	O	M	P	U	T	E	R

b.

45	52	55	54	4D	50	43	4F
E	R	U	T	M	P	C	O

c. 4F4D, 5055, and 5455. Other misaligned 2-byte words would contain data from outside the given 64 bits.

d. 52 55 54 4D, 55 54 4D 50, and 54 4D 50 43. Other misaligned 4-byte words would contain data from outside the given 64 bits.