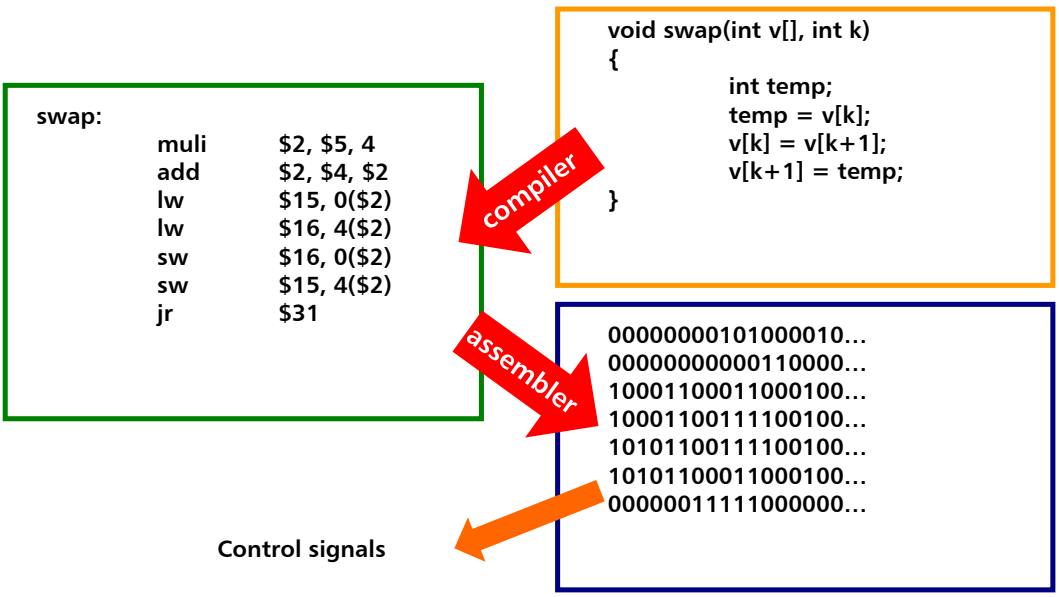
CS2410: Computer Architecture

Instruction set architecture
principles and examples

Sangyeun Cho

Computer Science Department
University of Pittsburgh

C to binary to electrical signals



ISA?

- ISA components:
  - Data types supported (e.g., bytes, half words, words–signed, unsigned)
  - Registers as a storage of data or information
    - General registers: e.g., R0~R7
    - Special registers: e.g., PSR (processor status register), return address from exception, …
  - Processor modes
    - User mode, privileged mode, …
  - Register view in different modes
  - Instruction definitions
    - Basic semantics: e.g., add, multiply
    - Exception behaviors: e.g., load – misaligned access, TLB miss, …
    - Some instructions may behave differently in different modes
    - Some instructions are not available in user mode (e.g., privileged instructions)
    - …
- ABI (application binary interface)
  - ≠ ISA
  - Defines low-level binary interface between an application and the OS
    - Calling convention
    - System call mechanism
    - Binary format, …

ISA design considerations

- Target application
  - General-purpose processor
  - Application-specific processor
- Roadmap
- Properties generally considered desirable
  - Completeness
  - Orthogonality
  - Regularity and simplicity
  - Compactness–code size
  - Ease of programming
  - Ease of implementation
  - Ease of extension

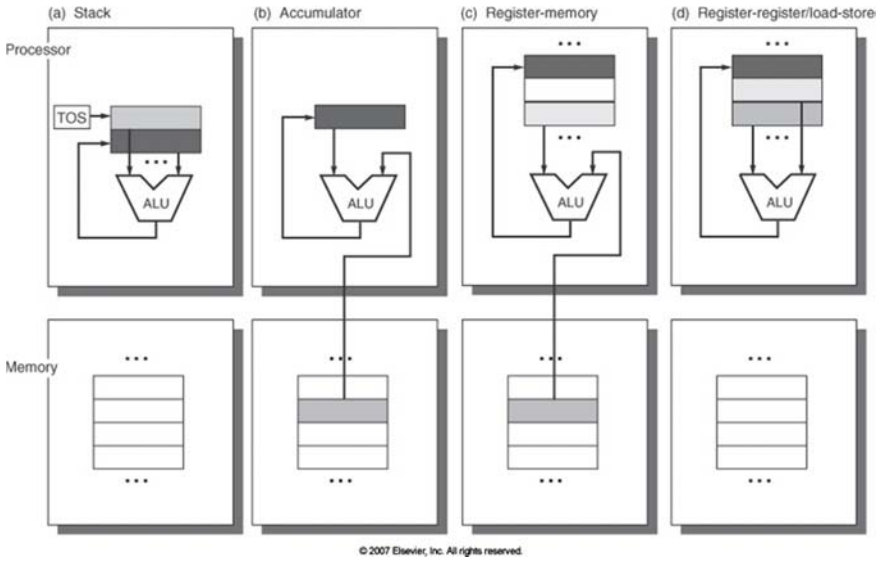
Machine instructions

- Operation – "what" to do?
  - How many operations?
  - What kind of operations?
    - Early processors omitted "multiply"
    - Floating-point operations were also absent
- Operands – what do we operate on?
  - How many operands?
  - Where are they stored?
  - How to specify?
- Format
  - Fixed N byte or variable size
  - What are necessary subfields?
  - How are subfields laid out?

Operand location

- Stack
  - Implicit – use data on the top of the stack
- Accumulator
  - Implicit – there is one accumulator
- Register
- Memory

Operand location



Number of operands

- Stack
  - 0 address — add — tos ← tos + next
- Accumulator
  - 1 address — add A — acc ← acc + mem[A]
- Register/memory
  - 2 addresses — add A B — A ← A + B
  - 3 addresses — add A B C — A ← B + C
- Load store architectures disallow operation and memory access in a single instruction

Sample code sequence

C = A + B

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

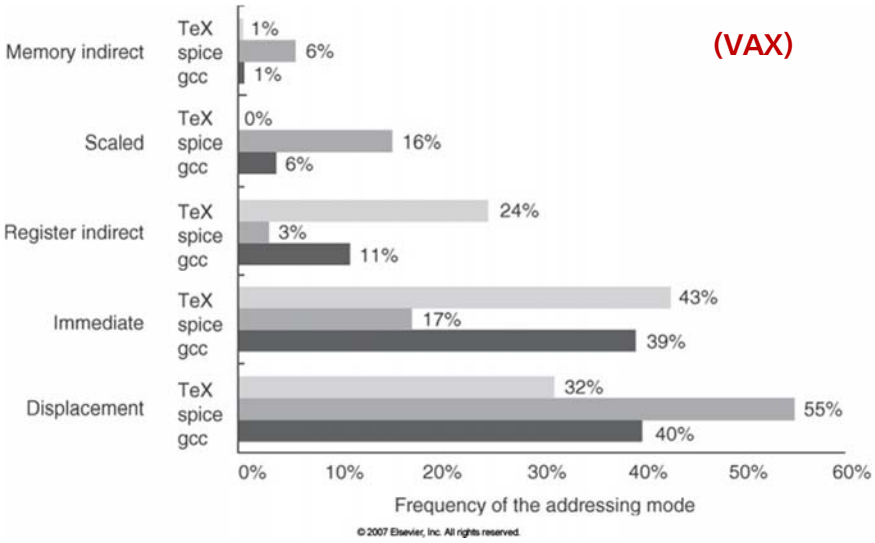
Specifying operands

- "Addressing mode"

- Register direct — add R1, R2, R3
- Immediate — add R1, R1, #1

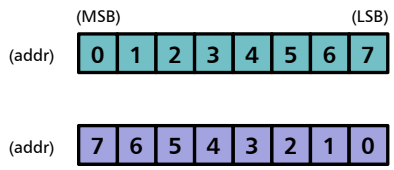
- Memory
  - Direct (absolute) — load R1, @(10000)
  - Register indirect — load R1, @(R2)
  - Memory indirect — load R1, @((R2))
  - Displacement — load R1, @(R2+100)
  - Indexed — load R1, @(R2+R3)
  - Scaled — load R1, @(R2+R3×d+100)
  - Auto-increment/decrement — load R1, @(R2+/–)

Usage of addressing modes



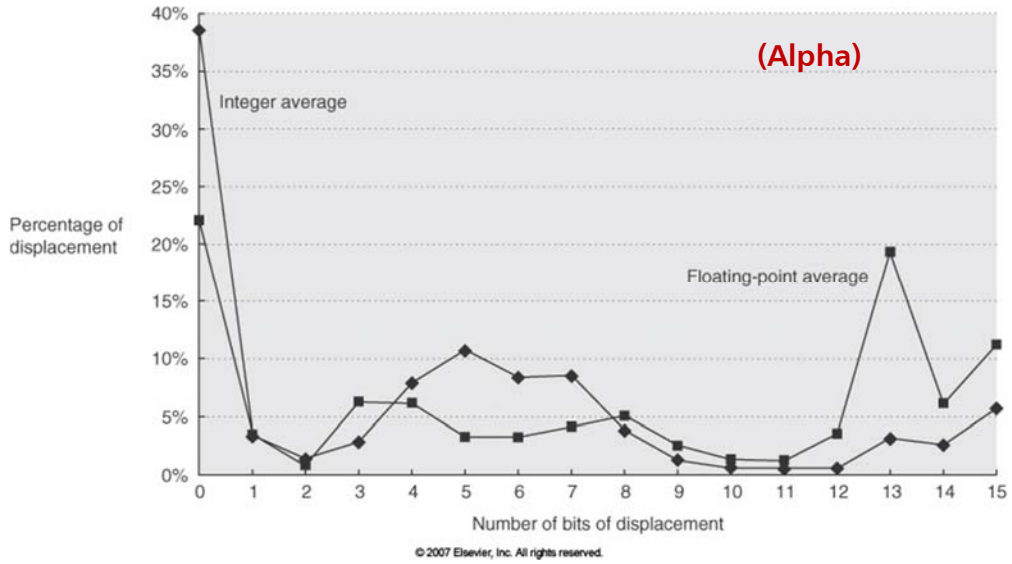
Endian-ness and data alignment

- Big endian vs. little endian
  - Defines byte ordering inside a larger data type stored in memory

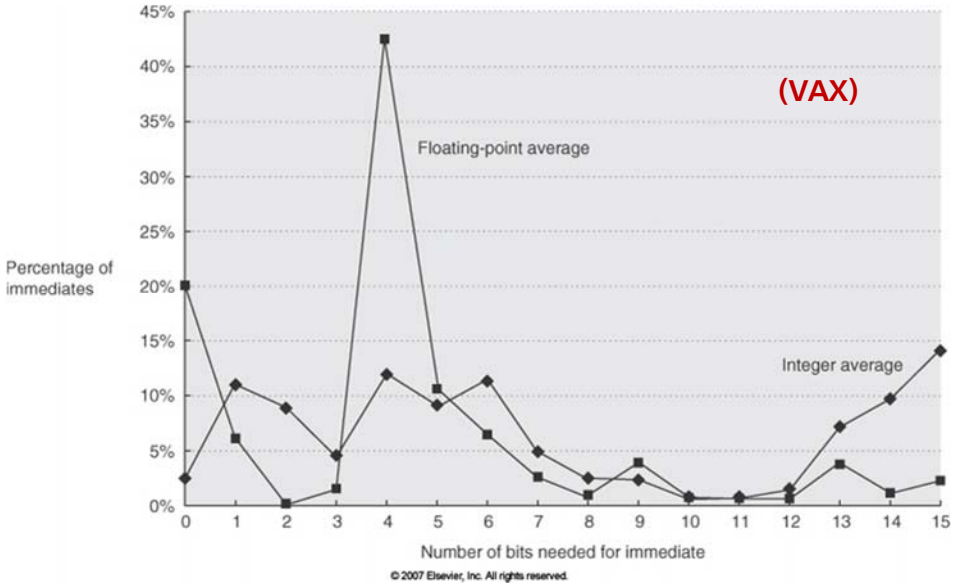


- Alignment
  - Data A is aligned if (addr(A) % sizeof(A) == 0) is true
  - What is the implication on hardware implementation (esp. memory system)?

Displacement size



Immediate size



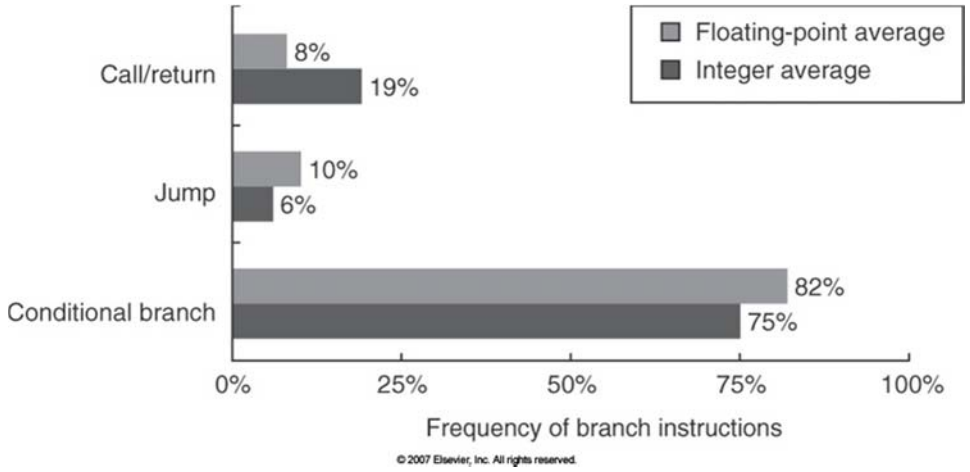
Operations

- Arithmetic
  - add, sub, mul, div, …
- Logical
  - and, or, xor, not, …
- Shift
  - shift-left, shift-right, shift-right-arithmetic, …
- Memory access
  - load, store, prefetch, …
- Control
  - branches, jumps, procedure call/return, …
- System-related
  - trap, break, …

Frequent 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%

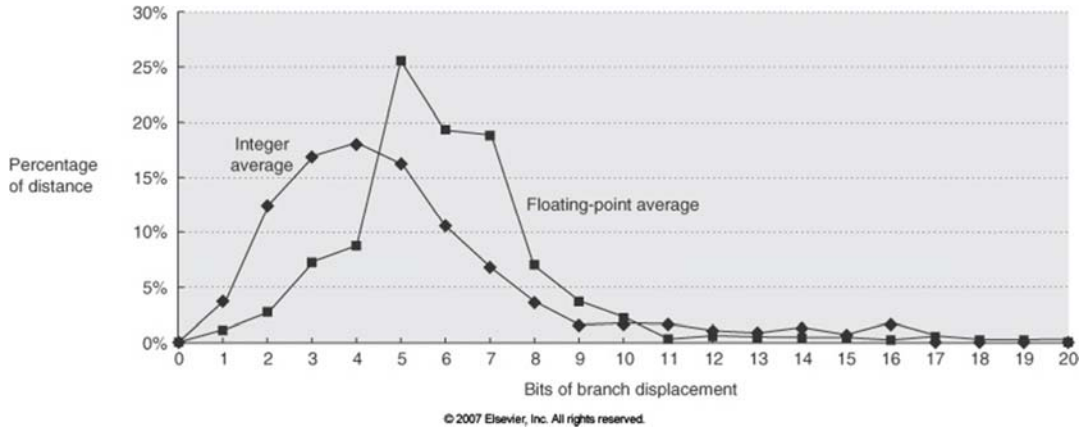
Control flow instructions



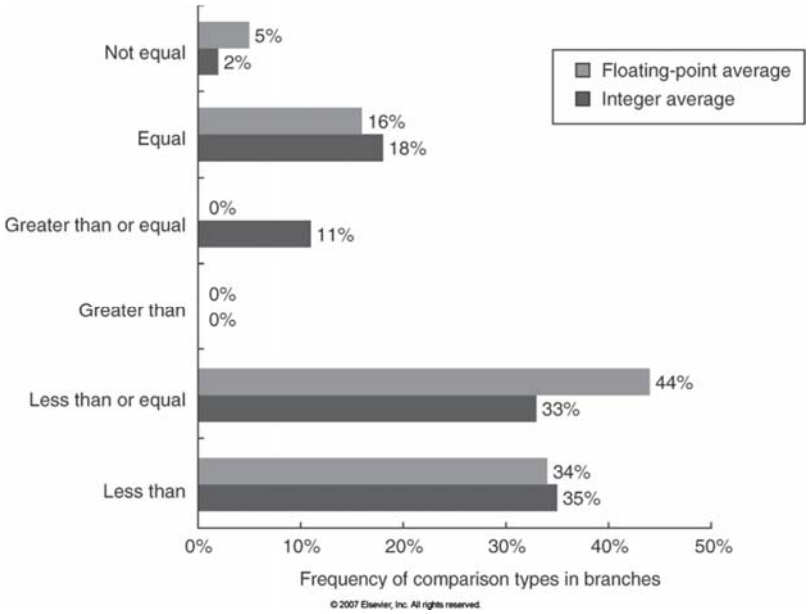
Operands of control flow inst.

- Target description
  - Instruction with an immediate value
    - PC-relative or absolute
  - In register
    - Typically absolute
- Condition description
  - Condition bit (zero, carry, overflow, …)
    - sub R1, R2, R3; bz LABEL
  - Condition register
    - cmp R1, R2, R3; bgtz R1, LABEL
  - Compare-and-branch
    - bgt R1, R2, LABEL

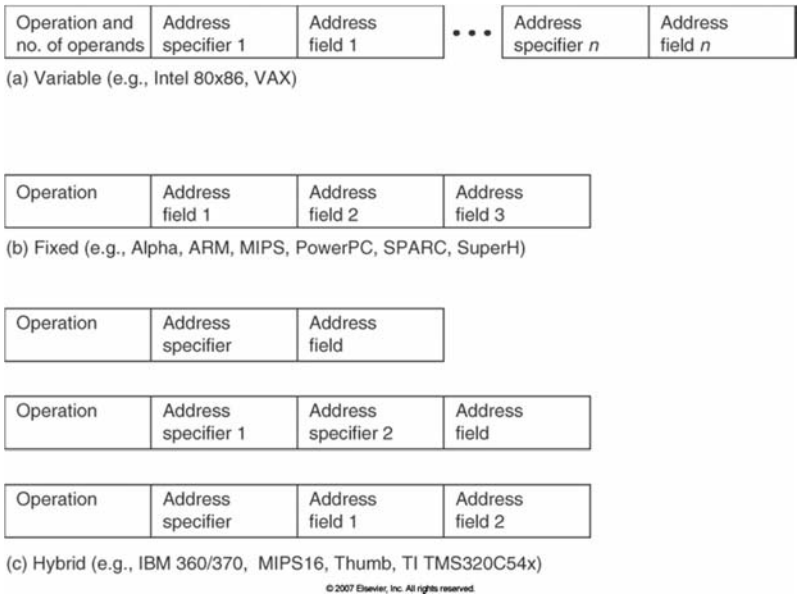
Branch distance



Comparison types



Instruction format

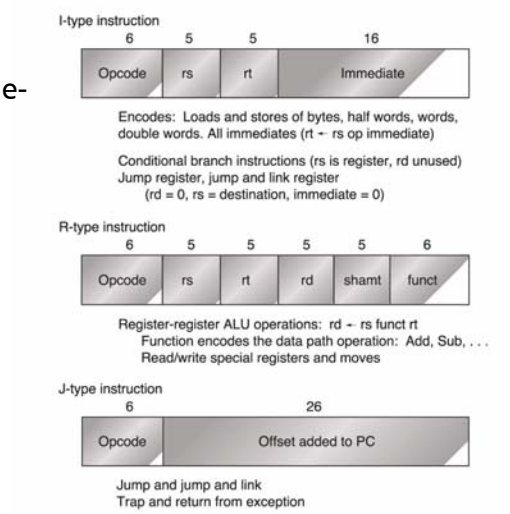


How to help compiler writers

- At least 16 GPRs
- Provide regularity
  - Orthogonality
  - No restrictions on register usage
- Provide primitives, not solutions
  - e.g., HLL-like operations
- Simplify trade-offs in alternatives
- Provide instructions that bind constants

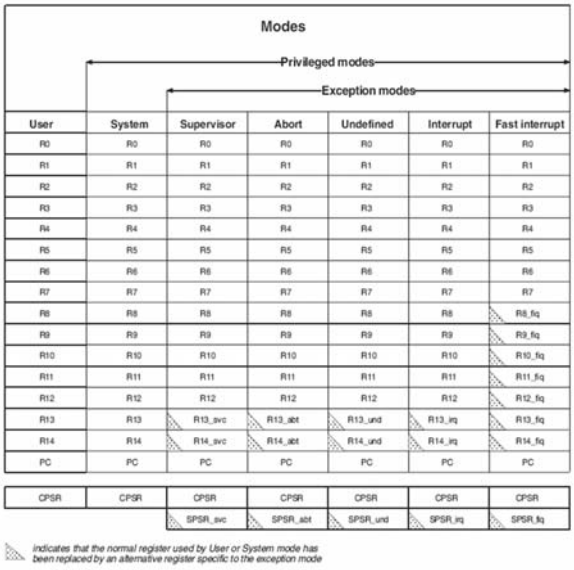
MIPS example

- Thirty-two 32-bit GPR
  - R0 wired to 0
  - Separate 32/16 SP/DP (single-/double-precision) FP registers
- Byte/half/word/dword, SP/DP FP data types
- Immediate and displacement addressing modes
- 32-bit fixed instruction encoding

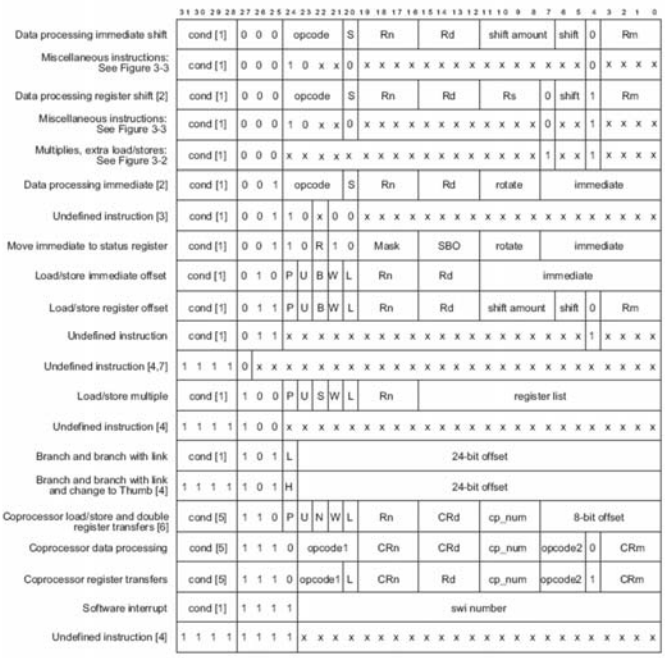


ARM example

- Sixteen 32-bit GPR
  - R14 is link register
  - R15 is PC
- Different CPU modes have different register view
- Immediate and displacement addressing modes
- "Thumb" mode supports 16-bit instruction encoding

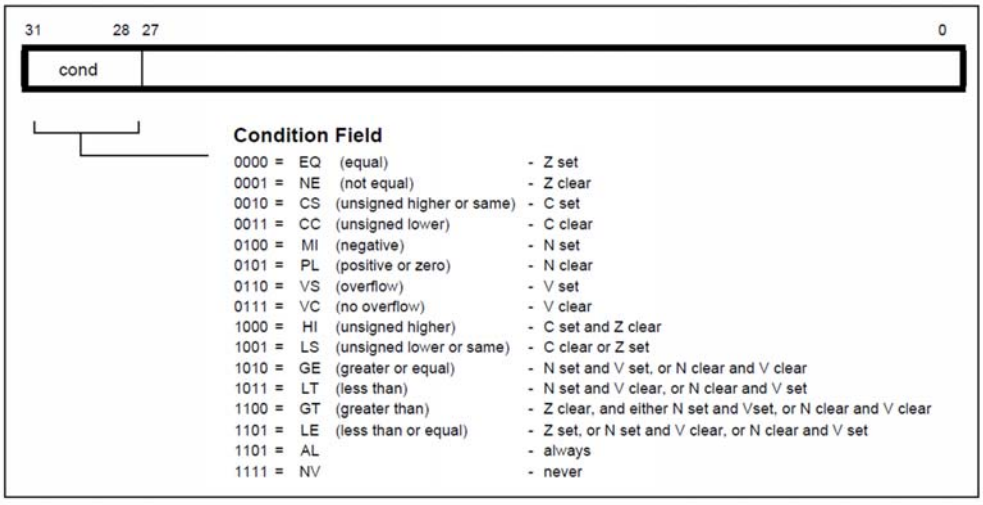


ARM example



ARM example, cont'd

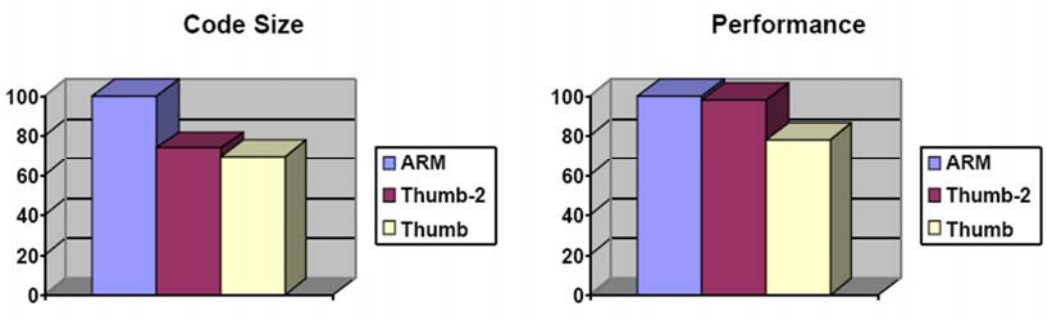
The Condition Field



Condition Field

0000 =	EQ	(equal)	- Z set
0001 =	NE	(not equal)	- Z clear
0010 =	CS	(unsigned higher or same)	- C set
0011 =	CC	(unsigned lower)	- C clear
0100 =	MI	(negative)	- N set
0101 =	PL	(positive or zero)	- N clear
0110 =	VS	(overflow)	- V set
0111 =	VC	(no overflow)	- V clear
1000 =	HI	(unsigned higher)	- C set and Z clear
1001 =	LS	(unsigned lower or same)	- C clear or Z set
1010 =	GE	(greater or equal)	- N set and V set, or N clear and V clear
1011 =	LT	(less than)	- N set and V clear, or N clear and V set
1100 =	GT	(greater than)	- Z clear, and either N set and Vset, or N clear and V clear
1101 =	LE	(less than or equal)	- Z set, or N set and V clear, or N clear and V set
1101 =	AL		- always
1111 =	NV		- never

ARM example



(Phelan, '03)

Summary

- Instruction set design requires understanding of
  - Application
  - Roadmap
  - Properties affecting hardware implementation
  - Properties affecting software design (e.g., application, compiler, …)
- Binary compatibility has been a key market driver
  - Legacy binaries
- Dynamic binary translation technology may weaken the dependency