

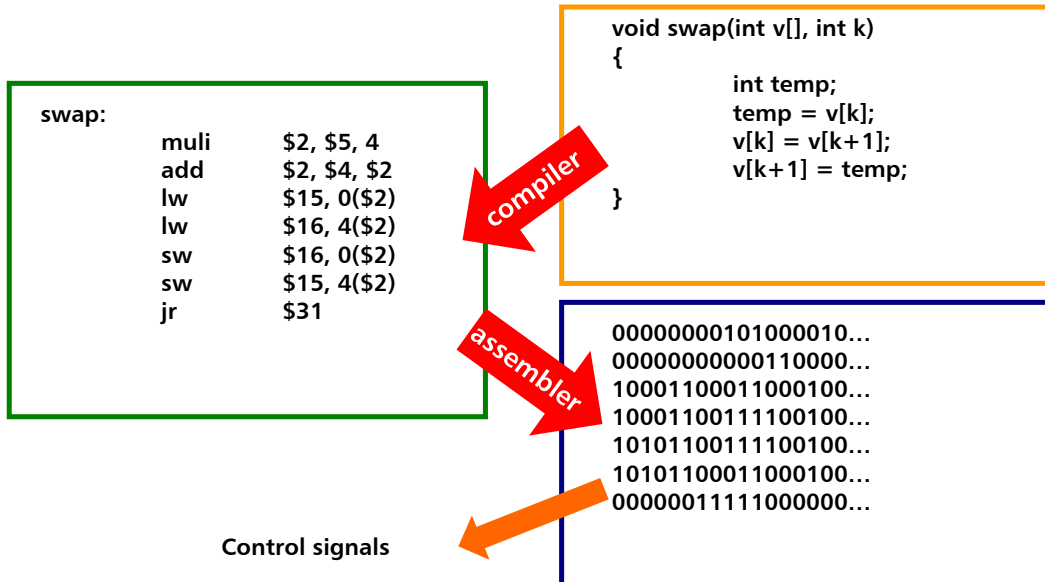
# CS2410: Computer Architecture

## 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

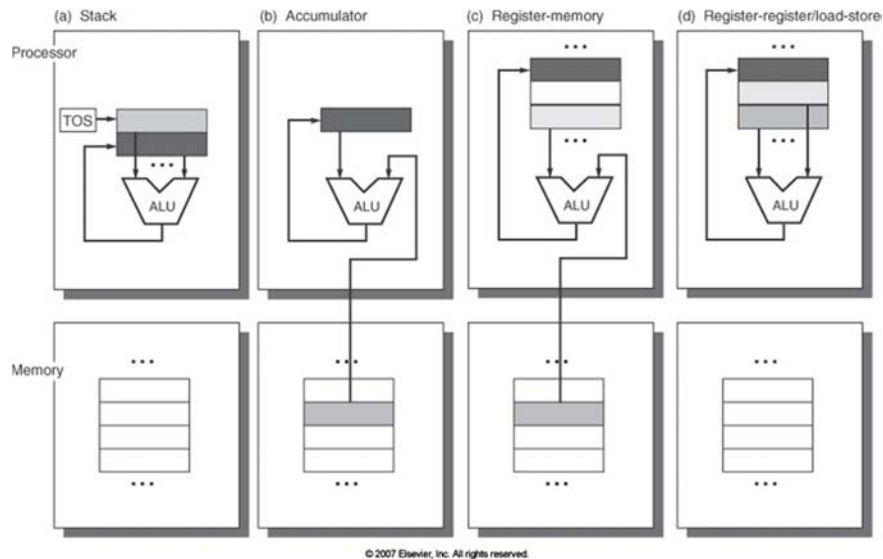
# Machine instructions

- 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



CS2410: Computer Architecture

University of Pittsburgh

# Number of operands

- **Stack**
  - 0 address                      add                       $\text{tos} \leftarrow \text{tos} + \text{next}$
- **Accumulator**
  - 1 address                      add A                       $\text{acc} \leftarrow \text{acc} + \text{mem}[A]$
- **Register/memory**
  - 2 addresses                      add A B                       $A \leftarrow A + B$
  - 3 addresses                      add A B C                       $A \leftarrow B + C$
- Load store architectures disallow operation and memory access in a single instruction

CS2410: Computer Architecture

University of Pittsburgh

# Sample code sequence

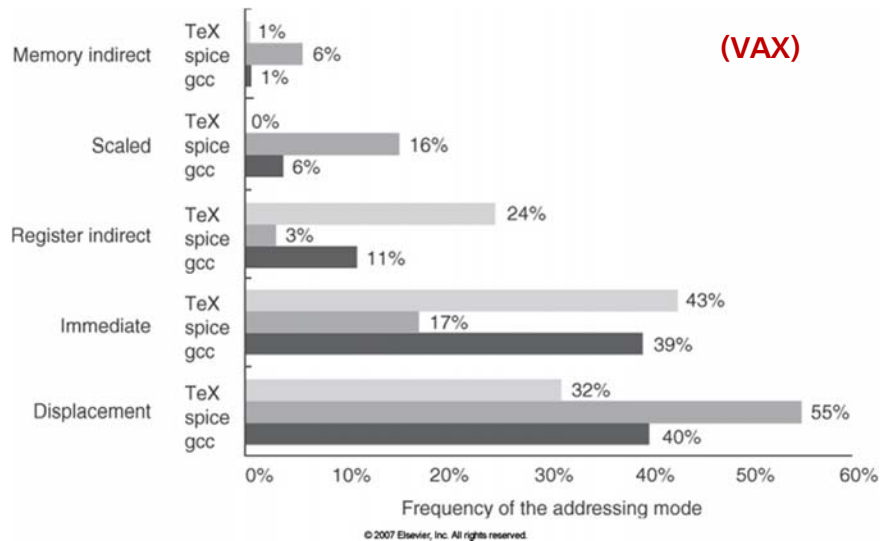
$$C = A + B$$

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

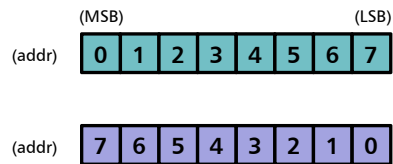
- "Addressing mode"
- Register direct                      add R1, R2, R3
- Immediate                              add R1, R1, #1
- Memory
  - Direct (absolute)                      load R1, @(10000)
  - Register indirect                      load R1, @(R2)
  - Memory indirect                      load 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



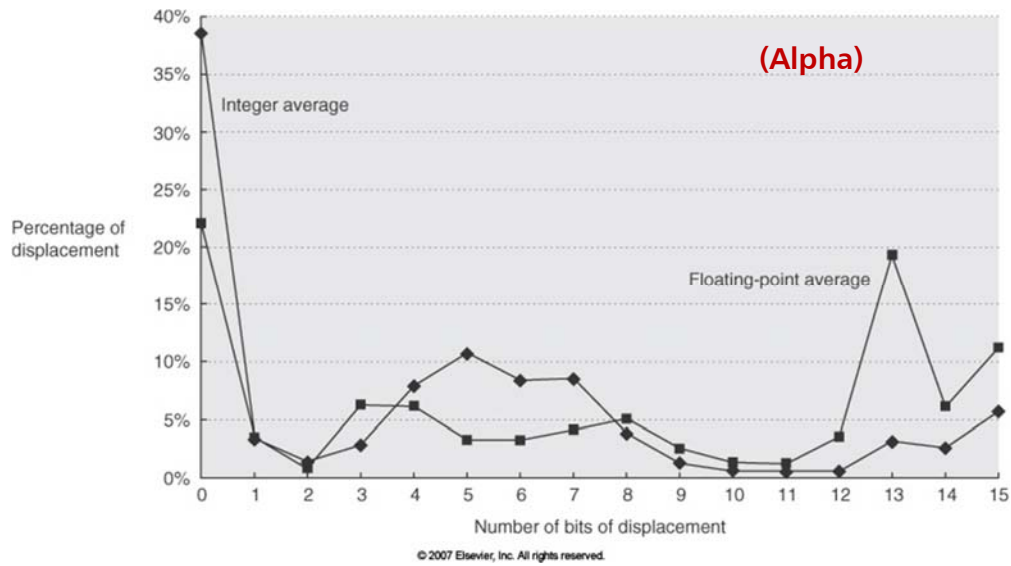
# Endian-ness and data alignment

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

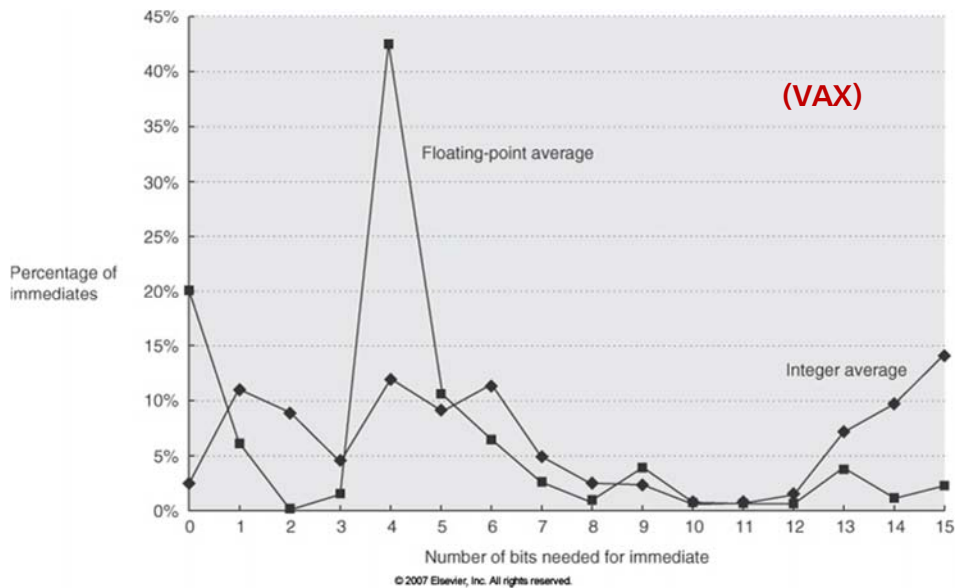


- Alignment
  - Data A is aligned if  $(\text{addr}(A) \% \text{sizeof}(A) == 0)$  is true
  - What is the implication on hardware implementation (esp. memory system)?

# Displacement size



# Immediate size



# Operations

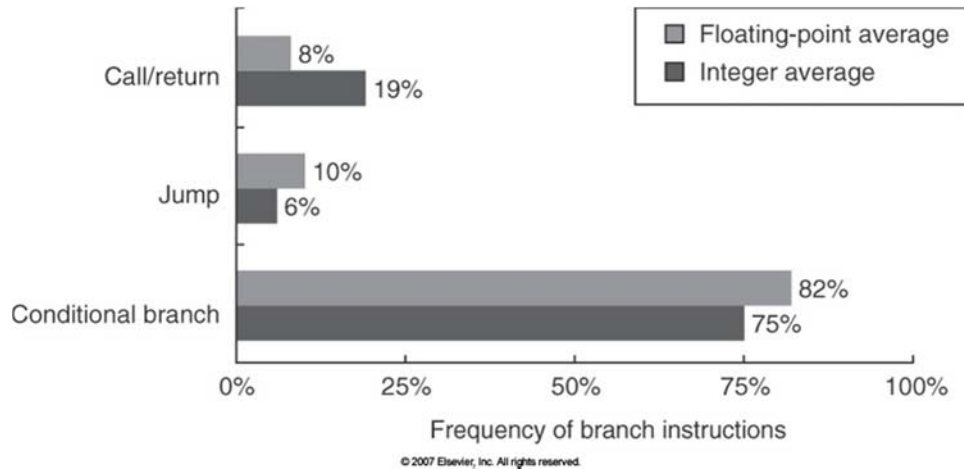
- Arithmetic
  - add, sub, mul, div, ...
- Logical
  - and, or, xor, not, ...
- Shift
  - shift-left, shift-right, shift-right-arithmetic, ...
- Memory access
  - load, store, prefetch, ...
- Control
  - branches, jumps, procedure call/return, ...
- System-related
  - trap, break, ...

# 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%
<b>Total</b>		<b>96%</b>



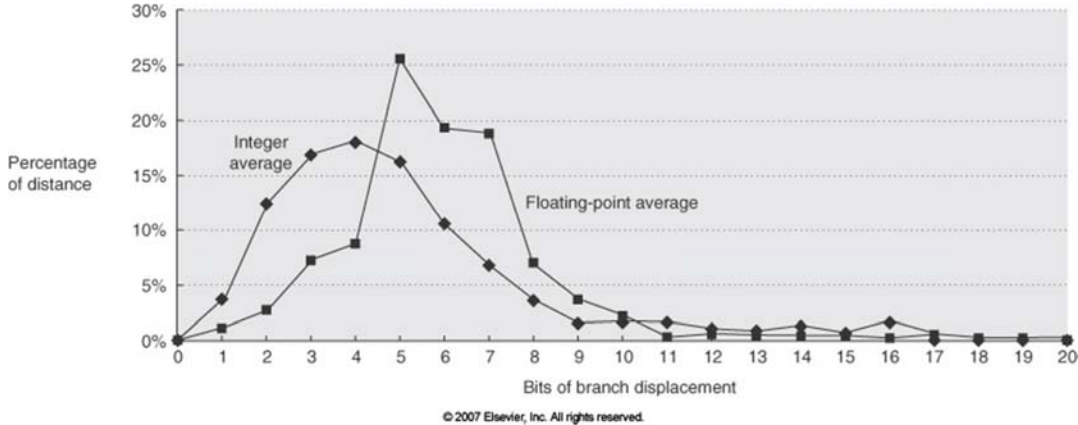
# Control flow instructions



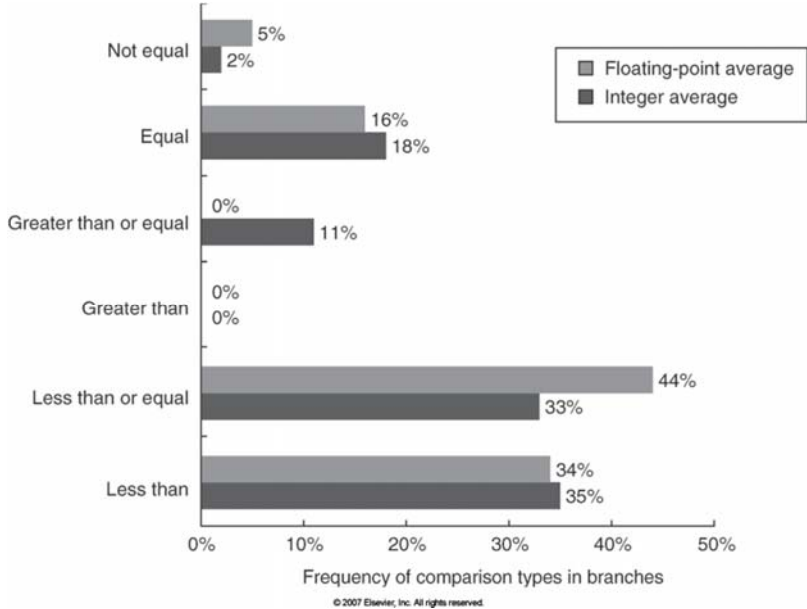
## Operands of control flow inst.

- 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



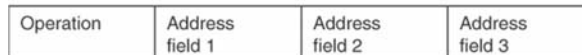
# Comparison types



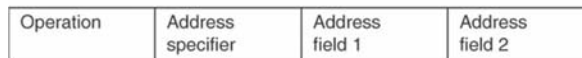
# Instruction format



(a) Variable (e.g., Intel 80x86, VAX)



(b) Fixed (e.g., Alpha, ARM, MIPS, PowerPC, SPARC, SuperH)



(c) Hybrid (e.g., IBM 360/370, MIPS16, Thumb, TI TMS320C54x)

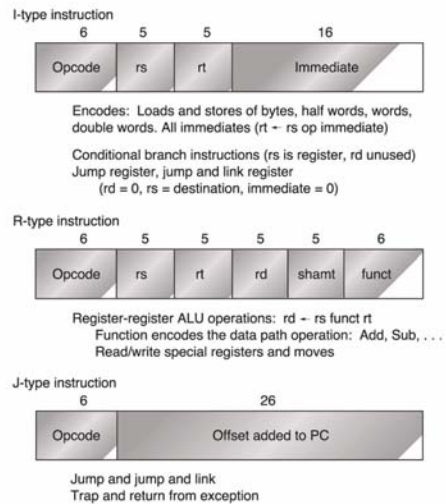
© 2007 Elsevier, Inc. All rights reserved.

## 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

Modes						
Privileged modes						
Exception modes						
User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8, Iq
R9	R9	R9	R9	R9	R9	R9, Iq
R10	R10	R10	R10	R10	R10	R10, Iq
R11	R11	R11	R11	R11	R11	R11, Iq
R12	R12	R12	R12	R12	R12	R12, Iq
R13	R13	R13, Iq	R13, Iq	R13, Iq	R13, Iq	R13, Iq
R14	R14	R14, Iq	R14, Iq	R14, Iq	R14, Iq	R14, Iq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR, Iq	SPSR, Iq	SPSR, Iq	SPSR, Iq	SPSR, Iq

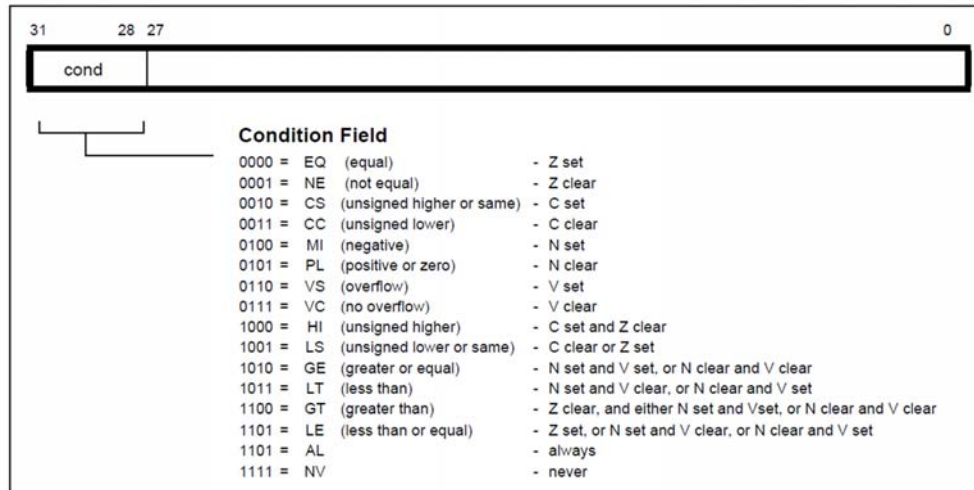
Indicates that the normal register used by User or System mode has been replaced by an alternative register specific to the exception mode

# ARM example

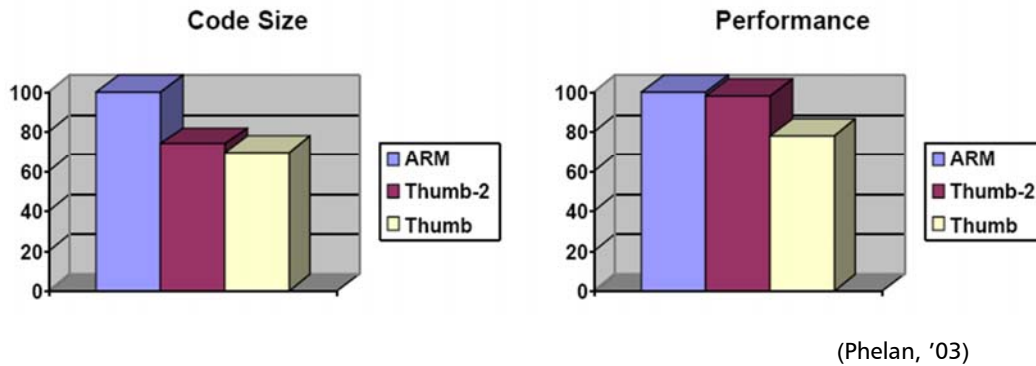
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data processing immediate shift	cond [1]	0	0	0	opcode	S	Rn	Rd	shift amount	shift	0	Rm																				
Miscellaneous instructions: See Figure 3-3	cond [1]	0	0	0	1	0	x	x	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
Data processing register shift [2]	cond [1]	0	0	0	opcode	S	Rn	Rd	Rs	0	shift	1	Rm																			
Miscellaneous instructions: See Figure 3-3	cond [1]	0	0	0	1	0	x	x	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
Multiples, extra load/stores: See Figure 3-2	cond [1]	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	1	x	x	1	x	x	x	x	
Data processing immediate [2]	cond [1]	0	0	1	opcode	S	Rn	Rd	rotate		immediate																					
Undefined instruction [3]	cond [1]	0	0	1	1	0	x	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
Move immediate to status register	cond [1]	0	0	1	1	0	R	1	0	Mask	SBO	rotate	immediate																			
Load/store immediate offset	cond [1]	0	1	0	P	U	B	W	L	Rn	Rd	immediate																				
Load/store register offset	cond [1]	0	1	1	P	U	B	W	L	Rn	Rd	shift amount	shift	0	Rm																	
Undefined instruction	cond [1]	0	1	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
Undefined instruction [4,7]	1	1	1	1	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
Load/store multiple	cond [1]	1	0	0	P	U	S	W	L	Rn	register list																					
Undefined instruction [4]	1	1	1	1	1	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
Branch and branch with link	cond [1]	1	0	1	L																											
Branch and branch with link and change to Thumb [4]	1	1	1	1	1	0	1	H																								
Coprocessor load/store and double register transfers [6]	cond [5]	1	1	0	P	U	N	W	L	Rn	CRd	cp_num	8-bit offset																			
Coprocessor data processing	cond [5]	1	1	1	0	opcode1	CRn	CRd	cp_num	opcode2	0	CRm																				
Coprocessor register transfers	cond [5]	1	1	1	0	opcode1	L	CRn	Rd	cp_num	opcode2	1	CRm																			
Software interrupt	cond [1]	1	1	1	1																											
Undefined instruction [4]	1	1	1	1	1	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	

# ARM example, cont'd

## The Condition Field



# ARM example



## 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