北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

Computer Organization and Architecture

Chapter 10

Instruction Sets: Characteristics and Functions

School of Computer Science（National Pilot Software Engineering School）
AO XIONG (熊翱)
xiongao@bupt.edu.cn

Preface

We have learned:

- Overview
  - Basic Concepts and Computer Evolution 基本概念和计算机发展历史
  - Performance Issues 性能问题
- The computer system
  - Top level view of computer function and interconnection 计算机功能和互联结构顶层视图
  - Cache Memory cache存储器
  - Internal Memory 内部存储器
  - External Memory 外部存储器
  - Input& Output 输入输出
  - Operating System Support 操作系统支持

Preface

We have learned:

- Arithmetic and Logic 算术与逻辑
  - Computer arithmetic 计算机算术
    - ✓ The Arithmetic(算术) and Logic Unit 算术逻辑单元
    - ✓ Integer Representation 整数表示
    - ✓ Integer Arithmetic 整数运算
    - ✓ Floating-Point Representation 浮点数表示
    - ✓ Floating-Point Arithmetic 浮点数运算

Review

- 无符号整数的加法中，如何判断溢出？
- 补码的加法中，如何判断溢出？
- 浮点数在数轴上的密度特征如何？

Preface

Next

- Central Processing Unit 中央处理单元（CPU）
  - Instruction Sets: Characteristics and Function 指令集：特征与功能
  - Instruction Sets: Addressing Modes and Formats 计算机算术指令集：寻址模式和格式
  - Processor Structure and Function 处理器结构与功能
  - Reduced Instruction Set Computer 精简指令集计算机
  - Instruction-Level Parallelism and Superscalar Processors 指令级并行与超标量处理器

Preface

We will focus the following contents today:

- The characteristics and function of instruction sets 指令集的特征和功能
  - What is the characteristics of instruction? 指令的特征是什么?
  - How many types is there about operands? 操作数类型有哪些?
  - What are the date types about Intel x86 and ARM? X86和ARM的数据类型有哪些
  - What are the main types of computer operation from the perspective of instruction? 从指令的角度来看，计算机有哪些操作类型

Outline

- Machine Instruction Characteristics 机器指令特征
- Types of Operands 操作数类型
- Intel x86 and ARM Data Types x86和ARM数据类型
- Types of Operations 操作类型
- Endian (端序) Support 端序支持

Programming language

- Classification of programming language 编程语言分类
  - Machine language 机器语言
  - Assembly language 汇编语言
  - High-level language 高级语言
- Compiler 编译器
  - Computers can only recognize machine language 计算机只能识别机器语言
  - Translation program that converts high-level/assembly language programs into machine language 将高级语言/汇编语言翻译成机器语言的翻译程序

Machine language 机器语言

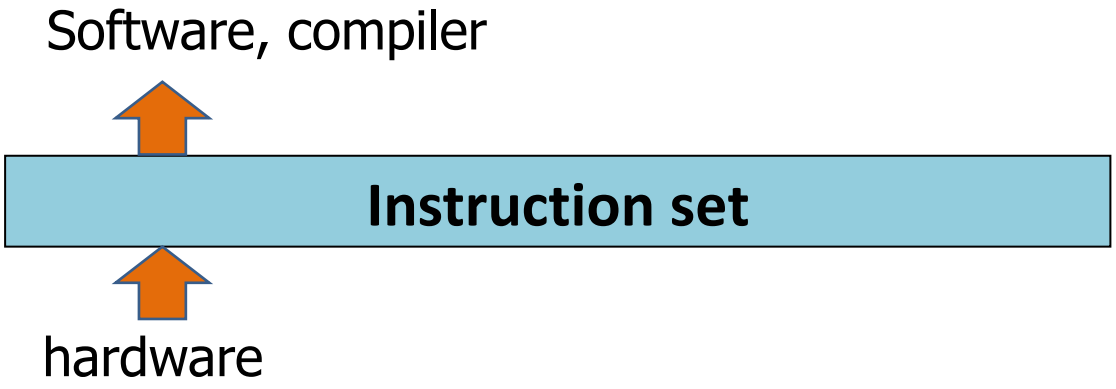
- Machine language 机器语言
  - Defined by the computer’s hardware design 计算机硬件设计定义
  - Consists of streams of numbers (1s and 0s) 由0和1的串构成
  - Instruct the computer to perform the most basic operations 指示计算机去执行最基本的一些操作
  - A computer can understand only its own machine language 计算机只能理解它自己的机器语言
  - It is difficult to remember, and generally will not be used directly 很难记，一般不会直接使用

Assembly language 汇编语言

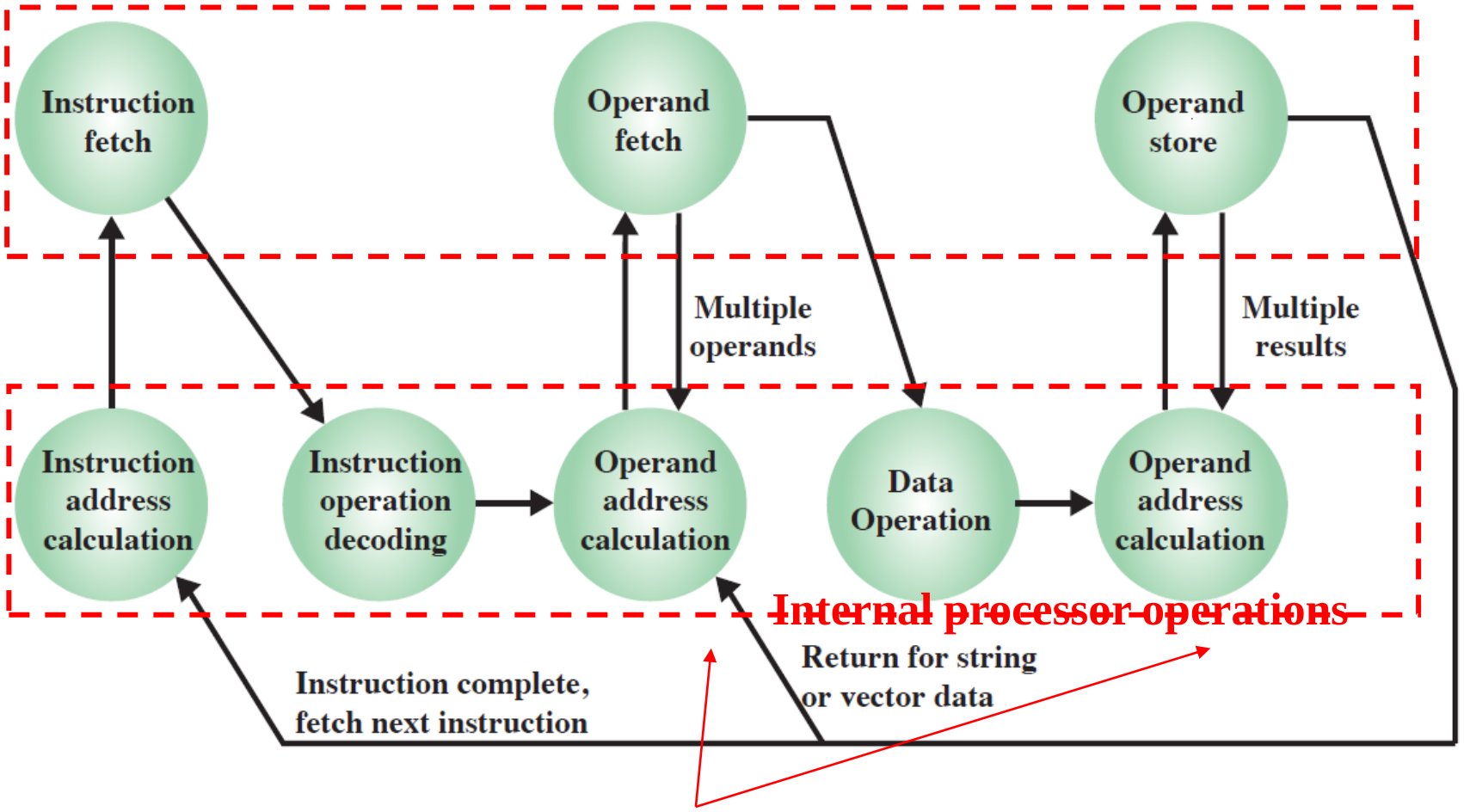
- Assembly language 汇编语言
  - Represents machine-language instructions using English-like abbreviations 使用类似英语的缩略语来表示机器语言指令
  - Replace the address of an instruction or operand with an address symbol or label 用地址符号或标号代替指令或操作数的地址
  - Assemblers convert assembly language to machine language 汇编器将汇编语言转换成机器语言
  - Specific assembly language and specific machine language instruction set are one-to-one, and cannot be directly transplanted between different computer 特定的汇编语言和特定的机器语言指令集是一一对应的，不同计算机之间不可直接移植

What is an instruction set? 什么是指令集?

- The complete collection of instructions that are understood by a CPU CPU能理解的完整的指令集合
  - Machine Code 机器代码
  - Binary 二进制
  - Usually represented by assembly codes 通常用汇编代码表示



Instruction cycle state diagram 指令执行状态



可能有多个操作或多个结果

Elements of an instruction 指令中的元素

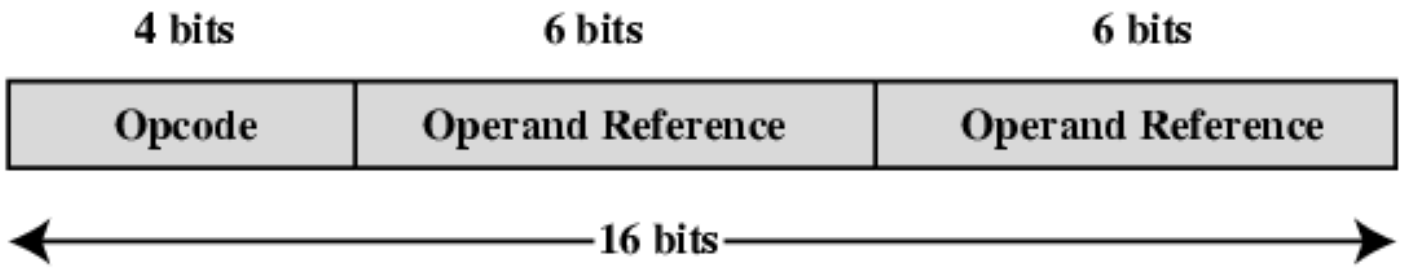
- Operation code (Op code) 操作码
  - Do what 做什么
- Source Operand reference 源操作数引用
  - From this 从这里来
- Result Operand reference 结果操作数引用
  - Put the answer here 结果放这里去
- Next Instruction Reference 下一个指令的引用
  - When you have done that, do this... 做完这个之后做什么
  - Generally, it defaults to the next storage unit 一般默认在下一个存储单元

Instruction representation 指令表示

- In machine code，each opcode has a unique bit string 机器码中，每个操作码都有唯一的bit串
- For programmers a symbolic representation is used 给程序员提供了助记符
  - e.g. ADD, SUB, LOAD
- The operand follows the opcode in the instruction 操作数在指令中接在操作码的后面
- If there are multiple operands, separate them with “,” 如果有多个操作数，用，分隔
  - ADD A,B

Instruction representation 指令表示

简单的指令格式



- 指令总共16个bit长，分为三个部分：
  - 第一部分是操作码，4个bit。最多有16种操作。
  - 第二部分是操作数1的引用，6个bit。
  - 第三部分是操作数2的引用，6个bit。
- 在指令中，源操作数和目的操作数可以在内存、CPU寄存器中或者I/O中，也可能是一个数，称之为立即数

Where are all the operands? 1

- Main memory
  - Memory address must be supplied 需要提供存储器地址
  - If virtual address is supplied，address translation required 如果是虚拟地址，需要进行地址转换
  - It may be in the cache. 可能在cache中
- I/O device
  - The instruction must specify the I/O module and device for the operation 指令中需要指定I/O模块和设备以便操作

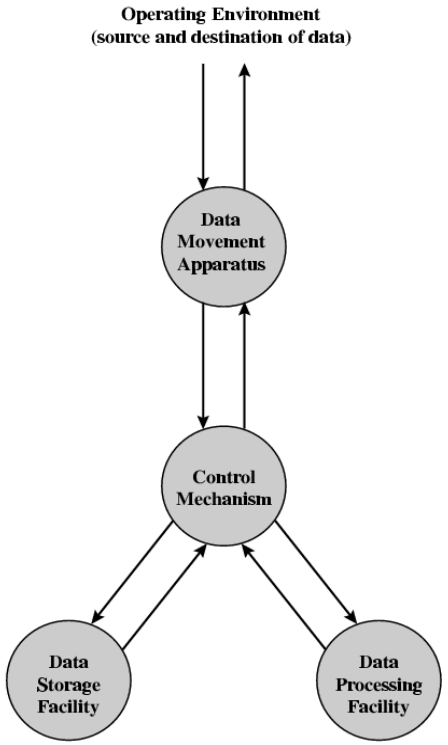
Where are all the operands? 2

- CPU register
  - If only one register exists, reference to it may be implicit. 如果CPU中只有一个寄存器的话，那么引用它是隐含的
  - If more than one register exists, then each register is assigned a unique name or number. CPU中有多个寄存器，每个寄存器一个名字
  - Instruction must contain the number of the desired register. 指令中引用这个寄存器，需要给出寄存器的名字
- Immediate
  - The value of the operand is contained in a field in the instruction being executed. 操作数的值也可以放在指令中

Instruction types 指令类型

The instructions can be categorized into four types 指令分为四种类型

- Data processing 数据处理
- Data storage 数据存储
- Data movement 数据传送
- Program flow control 程序流控制



Instruction types 指令类型

- Data processing instruction 数据处理指令
  - Processing data
  - Including arithmetic and logic instructions 算术和逻辑指令
- Data storage 数据存储指令
  - Storing data 数据存储
  - Mainly refers to the transfer of data between memory and CPU registers 主要指的是数据在内存和CPU的寄存器之间的传送

Instruction types 指令类型

- Data movement 数据移动
  - mainly refers to the data transmission between CPU and I/O 主要指的是CPU和I/O之间的数据传送
  - I/O instructions I/O指令
- Program flow control 程序流控制
  - Some instructions of CPU execution control CPU执行控制的指令
  - Test instructions 测试指令
  - Branch instructions 分支指令

Instruction types 指令类型

- *Arithmetic* instructions 算术指令
- *Logic* (Boolean) instructions 逻辑指令
- *Memory* instructions: moving data between memory and the registers. 存储指令：在内存和寄存器之间移动数据
- *I/O* instructions I/O指令
- *Test* instructions: used to test the value of a data word or the status of a computation. 测试指令：用于测试测试字的值或者计算的状态
- *Branch* instructions: used to branch to a different set of instructions. 分支指令：用于分支到不同的指令集合

Number of addresses

- The address in the instruction is used to address the operand
  指令中的地址用于进行操作数的寻址
- The number of addresses in instructions varies with different instruction types 不同的指令类型，在指令中的地址数量也不相同
- Different computers support different numbers of operands 不同的计算机，支持的操作数的数量也不同
- The number of operands in an instruction may be 3, 2, 1, or none 指令中操作数的数量可能有3个，2个，1个或没有

3 addresses

- 3 addresses 3个地址
  - Operand 1, Operand 2, Result 操作数1，操作数2，结果
  - a = b + c;
  - May be a forth - next instruction (usually implicit) 可能有第四个，也就是下一个指令地址，但一般是隐含的
  - Not common 不常用
  - Needs very long words to hold everything 需要很长的字来容纳这些字段

2 addresses

- 2 addresses 2个地址
  - Two operand 2个操作数
  - One address doubles as operand and result 1个地址为源操作数和结果的复用
  - a = a + b
  - Reduces length of instruction 减少了指令长度
  - Requires some extra work 需要额外的工作
    - Temporary storage to hold some results 临时存储结果

1 addresses

- 1 address 1个地址
  - Implicit second address 隐含了第二个操作数地址
  - Usually a register (accumulator) 通常使用寄存器，比如累加器
  - Common on early machines 在早期的机器上常用
    - LOAD A AC ← A
    - SUB B AC ← AC – B
    - STORE Y Y ← AC

0 addresses

- 0 (zero) addresses: All addresses implicit 没有地址，所有的地址都是隐含的
  - Usually use stack to imply the operands 通常使用栈来处理操作数
  - e.g.
    - push a
    - push b
    - add
    - pop c

弹出a和b，相加的结果再压栈

    - c = a + b

How many addresses? 多少地址合适呢?

- More addresses 地址多一些
  - More complex instructions 指令更复杂
  - Fewer instructions per program 程序中的指令少
  - Inter-register operations are quicker 寄存器间的操作会更快
  - More registers 更多的寄存器
- Fewer addresses 地址少一些
  - Less complex instructions 指令复杂度降低
  - More instructions per program 程序中的指令多
  - Faster fetch/execution of instructions 执行取指和执行更快

Instruction set design 指令集设计

- Design of computer instruction set is the most important and concerned aspect in computer design 计算机指令集的设计是计算机设计中最重要、同时也是最受关注的方面
  - First，the instruction set specifies the functions that the processor needs to complete 指令集规定了处理器需要完成的功能
  - Second，instruction set also needs to meet the requirements of the programmer 指令集还需要满足程序员的要求
- So, it is important to computer system 它对计算机系统的影响非常大

Design issues of instruction set -1 指令集设计要素1

- The most important of the fundamental design issues include the following 指令集设计中最重要的因素包括
  - Operation repertoire: how many and which operations to provides, and how complex operations should be 操作指令表，主要确定需要提供哪些操作给用户，以及操作的复杂度是怎样的
  - Data types: the operands types 数据类型：操作数类型

Design issues of instruction set -2 指令集设计要素2

- Instruction format: instruction length, number of addresses… 指令格式：指令长度，地址数量
- Registers: how many registers can be used by the instructions 寄存器：指令中可以用多少寄存器
- Addressing: how to access a memory location, how many modes can be used 寻址方式：如何访问内存地址，有多少种寻址模式可以用？

Design decision – 1 设计决策1

- Operation repertoire 操作指令表
  - How many ops? 多少种操作？
  - What can they do? 能做什么？
  - How complex are they? 多复杂？
- Data types
  - Integer 整型
  - Float 浮点型
  - Characters 字符型

Design decision – 2 设计决策2

- Instruction formats 指令格式
  - Length of op code field 操作码的长度
  - Number of addresses 地址数量
- Registers 寄存器
  - Number of CPU registers available 多少个CPU寄存器可用
  - Which operations can be performed on which registers? 哪些操作可以用哪些寄存器？
- Addressing modes 寻址模式
  - Direct addressing 直接寻址
  - Indirect addressing 间接寻址
- RISC v CISC 精简指令还是复杂指令

Summary

- 指令集： CPU能理解的完整的指令集合
- 指令内容
  - 操作码
  - 操作数：源操作数，目的操作数
  - 下一指令地址
- 指令设计的要素
  - 操作指令表
  - 操作数数量及类型
  - 指令格式
  - 寄存器
  - 寻址方式

Outline

- Machine Instruction Characteristics 机器指令特征
- Types of Operands 操作数类型
- Intel x86 and ARM Data Types x86和ARM数据类型
- Types of Operations 操作类型
- Endian (端序) Support 端序支持

Types of operand 操作数类型

- Addresses 地址
- Numbers 数值
  - Integer/floating point 整型/浮点型
- Characters 字符型
  - ASCII etc. ASCII 码
- Logical Data 逻辑数据
  - Bits or flags 位数据，标志

Address 地址

- The data operated by the instruction may be in memory 指令操作数可能在内存中
  - Address is used for addressing operands 用于进行实际操作数的寻址
  - Treat as an unsigned integer 作为无符号整数
- In many cases, need to process the address to get the actual address of the data 有很多时候，需要对地址进行处理后，才能得到数据实际的地址
- Addressing mode will be discussed later 寻址方式后面讨论

Numbers 数值型

- Three types of numerical data are common in computers 有三种类型的数值型数据
  - Binary integer or binary fixed point 二进制整数，或者二进制定点数
  - Binary floating point 二进制浮点数
  - Decimal 十进制数
    - Packed decimal 压缩的十进制数
    - Use 4-bit binary number to represent a decimal number用4位二进制数表示一个十进制数
    - In the packed decimal representation, only the previous 10 codes are used, that is, from 0000 to 1001 在压缩的十进制表示法中，只用前面的10个代码，也就是从0000~1001

Characters 字符型

- A common form of data is text or character strings 数据的一种常见形式是文本或字符串
- The most commonly used 常用的形式
  - International Reference Alphabet (IRA) 国际参考字母表
  - United States as the American Standard Code for Information Interchange (ASCII) 在美国称为ASCII码
- Extended Binary Coded Decimal Interchange Code (EBCDIC) 扩展的二进制表示的十进制交换码
  - Used on IBM mainframes 用在IBM大型机

ASCII

L \ H	0000	0001	0010	0011	0100	0101	0110	0111
0000	NUL	DLE	SP	0	@	P	‘	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	“	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	,	7	G	W	g	w
1000	BS	CAN	)	8	H	X	h	x
1001	HT	EM	(	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	’	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Logical data 逻辑数据

- Boolean or binary data items 布尔或二进制数据
  - Each item can take on only the values 1 (true) and 0 (false) 每个数据值只能采用值1（真）和0（假）
- There are occasions when we wish to manipulate the bits of a data item 有时希望按位对数据项进行操作

数据都是以二进制串的形式保存的。因此数据是什么类型的主要取决于指令的类型。指令中确定了它所操作的数据的类型

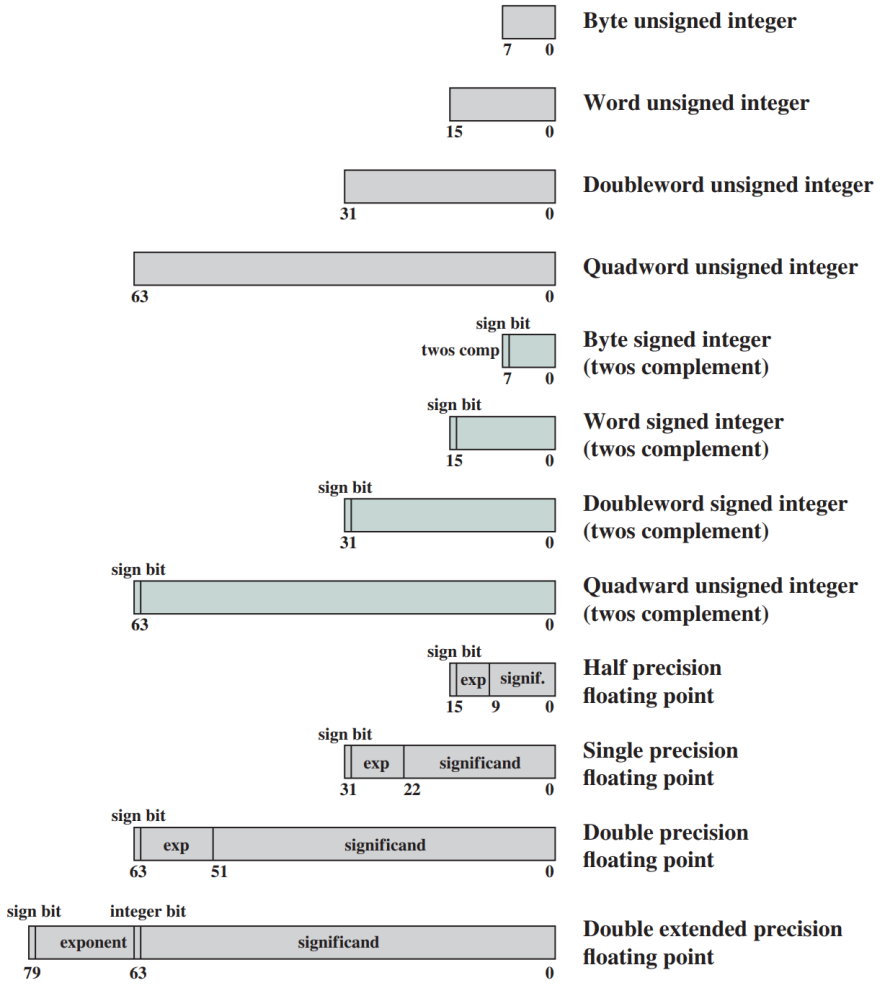
Outline

- Machine Instruction Characteristics 机器指令特征
- Types of Operands 操作数类型
- Intel x86 and ARM Data Types x86和ARM数据类型
- Types of Operations 操作类型
- Endian (端序) Support 端序支持

X86 data types x86的数据类型

- 8 bit Byte 8位字节
- 16 bit word 16位字
- 32 bit double word 32位双字
- 64 bit quad word 64位四字
- 128 bit double quadword 128位双四字
- Addressing is by 8 bit unit 按字节寻址
- Words do not need to align at even-numbered address 字不需要对齐
- Data accessed across 32 bit bus in units of double word read at addresses divisible by 4 数据通过32位总线访问，地址除以4
- Little endian 小端端序

x86 numeric data formats x86数值型数据的格式



- 无符号整数4种格式，分别是8位，16位，32位和64位
- 有符号数采用补码形式，也有4种格式，位数是8位，16位，32位和64位
- 浮点数，包括单精度浮点数，32位；双精度浮点数，64位，以及扩展的双精度浮点数，80位
- 浮点数表示符合IEEE 754标准的要求

SIMD data types -1 SIMD数据类型-1

- In the X86 architecture, MMX （Multi Media eXtension）related instructions are added to improve the processing efficiency of multimedia data X86架构中，为了提高对多媒体数据的处理效率，增加了MMX（多媒体扩展）相关的指令
  - MMX technology adds 57 instructions specially designed for video signal, audio signal and graphic manipulation to the CPU. MMX技术是在CPU中加入了特地为视频信号、音频信号以及图像处理而设计的57条指令
  - Therefore, MMX CPU greatly improves the computer ‘s multimedia (such as stereo, video, 3D animation, etc.) processing function MMX CPU极大地提高了电脑的多媒体(如立体声、视频、三维动画等)处理功能
  - In MMX instructions, one instruction can process multiple data at the same time, which is called single instruction multiple data（SIMD） 在MMX指令中，一个指令可以同时处理多个数据，这个称为单指令多数据

SIMD data types -2 SIMD数据类型-2

- Basic idea of SIMD is to package multiple operands into one memory addressable data, that is, the data obtained by one addressing is the result of multiple data packages SIMD的基本思想是，把多个操作数打包成一个内存寻址的数据，也就是一次寻址得到的数据，是多个数据打包得到的
  - one instruction can obtain multiple operands and process them at the same time 这样一个指令就可以获取到多个操作数，并且同时对这些操作数进行处理
- Five packaging methods for compressed data 数据有五种打包方式
  - Packed byte and packed byte integer
  - Packed word and packed word integer
  - Packed doubleword and packed doubleword integer
  - Packed quadword and packed quadword integer
  - Packed single-precision floating-point and packed double-precision floating-point

SIMD data types -3 SIMD数据类型3

- Packed byte and packed byte integer 打包的字节和打包的字节整数
  - Bytes packed into 64-bit quadword 字节打包为64位四字
  - or 128-bit double quadword 或128位双四字
- Packed word and packed word integer打包的字和打包的字整数
  - 16-bit words packed into 64-bit quadword 16位字打包成64位四字
  - or 128-bit double quadword 或128位双四字
- Packed doubleword and packed doubleword integer打包的双字和打包的双字整数
  - 32-bit doublewords packed into 64-bit quadword 32位双字打包成64位四字
  - or 128-bit double quadword 或128位双四字

SIMD data types -4 SIMD数据类型4

- Packed quadword and packed quadword integer 打包的四字和打包的四字整数
  - Two 64-bit quadwords packed into 128-bit double quadword 两个64位四字打包成128位双四字
- Packed single-precision floating-point and packed double-precision floating-point 打包的单精度浮点数和打包的双精度浮点数
  - Four 32-bit floating-point or two 64-bit floating-point values packed into a 128-bit double quadword 4个32位的单精度浮点数或2个64位的双精度浮点数打包到1个128位的双四字中

ARM data types ARM的数据类型

- 8 (byte), 16 (halfword), 32 (word) bits 8位字节，16位半字，32位字
- Halfword and word accesses should be word aligned 半字和字都需要对齐
- Nonaligned access alternatives 不对齐的处理
  - Default 默认方式
    - Treated as truncated 截断
    - Load single word instructions rotate right word aligned data transferred by non word-aligned address one, two or three bytes Alignment checking 处理器在载入单个字的时候，如果地址没有对齐，会以这个地址为基础，进行右移1个、2个或3个字节
  - Data abort signal indicates alignment fault for attempting unaligned access 进行对齐检查，如果没有对齐，就报错
  - Unaligned access：Processor uses one or more memory accesses to generate transfer of adjacent bytes transparently to the programmer 不对齐的访问：处理器将进行多次存储器访问，然后进行拼装，并返回给程序

ARM data types ARM的数据类型

- Unsigned integer interpretation supported for all types 无符号整数支持各种类型
- Twos-complement signed integer interpretation supported for all types 补码数也能支持各种类型
- Majority of implementations do not provide floating-point hardware 大部分不支持浮点数
  - Saves power and area 节能
  - Floating-point arithmetic implemented in software 用软件实现
  - Optional floating-point coprocessor 可选的浮点数协处理器
  - Single- and double-precision IEEE 754 floating point data types 支持单精度和双精度浮点数

Outline

- Machine Instruction Characteristics 机器指令特征
- Types of Operands 操作数类型
- Intel x86 and ARM Data Types x86和ARM数据类型
- Types of Operations 操作类型
- Endian (端序) Support 端序支持

Types of operation 操作类型

- Arithmetic 算术运算
- Logical 逻辑运算
- Data Transfer 数据传送
- Conversion 转换
- I/O 输入/输出
- System Control 系统控制
- Transfer of Control 控制转移

1. Data transfer 数据传送指令

- Location of source and destination must be specified 需要指定源数据和目的数据的位置
  - Memory 内存
  - Register 寄存器
  - Top of the stack 栈顶
- For the memory access, addressing mode must be specified 对于内存，寻址模式需要指定
  - Memory has multiple addressing modes, such as direct addressing and indirect addressing 内存有多种寻址方式，比如直接寻址、间接寻址等
- The length of the operands must be specified 操作数的长度需要确定

Data transfer 数据传送

- Data transmission instructions need to specify 数据传送指令需要明确：
  - Source address 源数据地址
  - Destination address 目的数据地址
  - Amount of data 数据数量
- Which data transfer instructions are included is one of the important issues to be considered in instruction set design 指令集里包括哪些数据传送指令，是设计中需要考虑的重要问题之一
- For example, whether the location of the operand is determined by the opcode or by the operand needs to be designed 比如说，操作数所在的位置，是用操作码来确定，还是说由操作数本身来确定，就是需要设计

Data transfer 数据传送

- IBM 390
  - Use different instructions for different movements 不同的指令完成不同的移动
  - Operation code determines the direction of data movement 操作码决定了数据的移动方向
- VAX
  - Data transmission between different data sources with the same用同一个指令实现不同数据源之间的数据传送
  - The position of each operand must be specified separately in the instruction 每个操作数的位置必须在指令中分别指定

IBM S/390 data transfer operation IBM390的数据传送

Operation Mnemonic	Name	Number of Bits Transferred	Description
L	Load	32	Transfer from memory to register
LH	Load Halfword	16	Transfer from memory to register
LR	Load	32	Transfer from register to register
LER	Load (Short)	32	Transfer from floating-point register to floating-point register
LE	Load (Short)	32	Transfer from memory to floating-point register
LDR	Load (Long)	64	Transfer from floating-point register to floating-point register
LD	Load (Long)	64	Transfer from memory to floating-point register
ST	Store	32	Transfer from register to memory
STH	Store Halfword	16	Transfer from register to memory
STC	Store Character	8	Transfer from register to memory
STE	Store (Short)	32	Transfer from floating-point register to memory
STD	Store (Long)	64	Transfer from floating-point register to memory

- 不同的数据源，不同的数据类型，都使用不同的指令
- 程序员根据数据传送的要求，选用不同的指令
- 指令类型多
- 指令结构比较紧凑

Common Data Transfer Instructions 通用数据传输指令

Type	Operation Name	Description
Data transfer	Move (transfer)	Transfer word or block from source to destination
	Store	Transfer word from processor to memory
	Load (fetch)	Transfer word from memory to processor
	Exchange	Swap contents of source and destination
	Clear (reset)	Transfer word of 0s to destination
	Set	Transfer word of 1s to destination
	Push	Transfer word from source to top of stack
	Pop	Transfer word from top of stack to destination

- 常见的数据传送指令包括：移动、存储、加载、交换、清除、设置、进栈、出栈等
- 数据传送对于处理器来说是最基本、最简单的操作，实现了数据从一个位置到另一个位置的移动

Common Data Transfer Instructions 通用数据传输指令

Data transfer	Transfer data from one location to another
	If memory is involved: Determine memory address Perform virtual-to-actual-memory address transformation Check cache Initiate memory read/write

- 数据传送指令将数据从一个位置移动到另一个位置
- 如果数据传送涉及到内存的话，还有一点复杂：
  - 需要根据寻址方式来计算存储器的地址
  - 如果给的是虚拟地址的话，还需要进行虚拟地址到实际地址的转换
  - 所以得到实存地址后，需要检查数据是否在cache中。如果在cache中，就对cache进行读取操作；如果没有命中，需要进行存储器的读或者写操作

2. Arithmetic 算术指令

Arithmetic	Add	Compute sum of two operands
	Subtract	Compute difference of two operands
	Multiply	Compute product of two operands
	Divide	Compute quotient of two operands
	Absolute	Replace operand by its absolute value
	Negate	Change sign of operand
	Increment	Add 1 to operand
	Decrement	Subtract 1 from operand

- Single operand instruction 单操作数指令
  - absolute, negate, increment, decrement 绝对值，取反，加1，减1
- Two operands instruction 双操作数指令
  - Add, subtract, multiply, divide 加，减，乘，除
- Mainly completed by CPU 主要由CPU来完成

2. Arithmetic 算术指令

Arithmetic	May involve data transfer, before and/or after
	Perform function in ALU
	Set condition codes and flags

- The operands are 操作数包括
  - Signed integer (fixed point) numbers 有符号数或定点数
  - Floating-point numbers 浮点数
  - Packed decimal numbers 压缩的十进制数
- 算术运算的操作可能会包括数据传输传送操作。数据传送操作的目的是在运算前给ALU提供操作的数据，或者在运算后将结果输出
- 算术操作的实际执行是在ALU中
- 计算完成之后，还会设置状态码或标志位，用以表示计算的结果，比如是否溢出，是否出错等

3. Logical 逻辑指令

Logical	AND	Perform logical AND
	OR	Perform logical OR
	NOT	(complement) Perform logical NOT
	Exclusive-OR	Perform logical XOR
	Test	Test specified condition; set flag(s) based on outcome
	Compare	Make logical or arithmetic comparison of two or more operands; set flag(s) based on outcome
	Set Control Variables	Class of instructions to set controls for protection purposes, interrupt handling, timer control, etc.
	Shift	Left (right) shift operand, introducing constants at end
	Rotate	Left (right) shift operand, with wraparound end

- Most processors can operate on a single bit of a word or addressable unit 大部分的处理器能够对一个字或可寻址单元的单个位进行操作
  - Called "bit twiddling" 称为位操纵
  - It's actually a bit wise Boolean operation 实际上就是按位进行布尔运算

3. Logical 逻辑指令

- Basic logic operation 基本的逻辑操作
  - AND, OR, NOT, Exclusive-OR 与，或，非，异或
- Extended logical operations 扩展的逻辑操作
  - Test，compare，set control variables，shift，rotate 测试，比较，设置控制变量，移位，旋转
- Test：测试指令，进行特定条件的测试并设置标志位
- Compare：比较指令，对两个或多个操作数进行比较，并设置标志位
- Set control variables：一组用于设置控制位的指令，以进行保护，中断处理，定时控制的用途
- Shift：左移或右移数据
- Rotate：循环移位

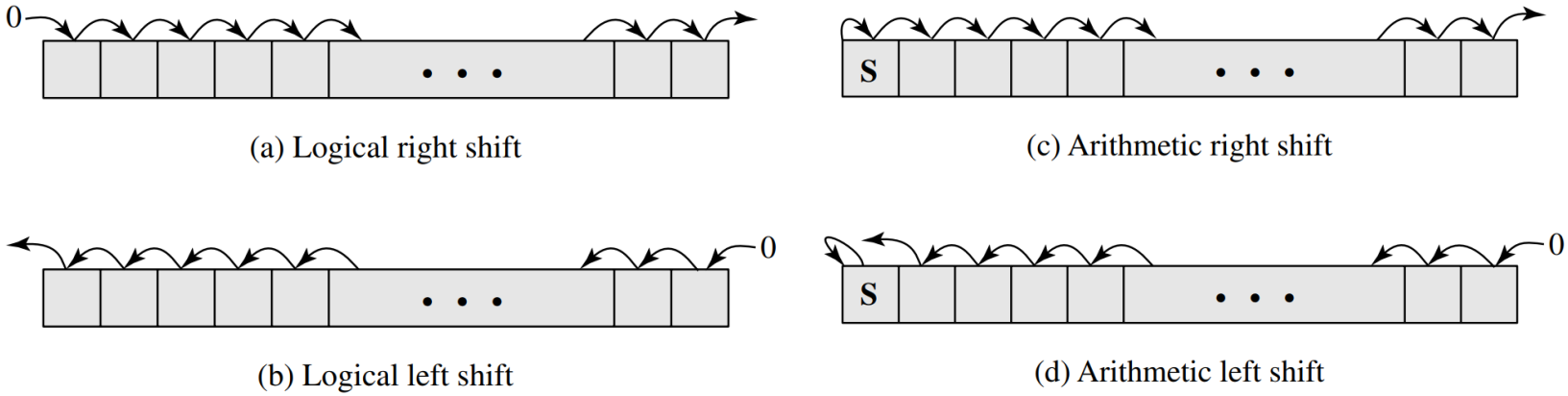
Shift operations 移位操作

- Logical shift：without considering the highest sign bit 逻辑移位：不考虑最高符号位
  - Logical right：Move the operand to the right by n bits, and fill in 0 at the left position 逻辑右移：操作数向右移动n位，左边空出的位置补0
  - Logical left：Move the operand to the left by n, and fill in 0 for the n bits vacated on the right 操作数向左移动n为，右边空出的n位补0

Shift operations 移位操作

- Arithmetic shift：consider the highest sign bit 算术移位：考虑最高符号位
  - Arithmetic right：Shift n bits to the right as a whole, and fill the empty n bits on the left with the highest sign bit 算术右移：整体右移n位，左边空出的n位用最高位的符号位来填充
  - Arithmetic left：Retain the sign bit of the highest bit, and then shift the other bits by n bits to the left 算术左移：保留最高位的符号位，然后其他位左移n位

Example



(a) Logical right shift

(c) Arithmetic right shift

(b) Logical left shift

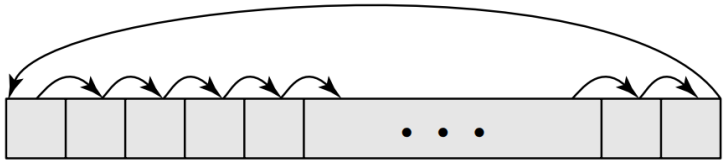
(d) Arithmetic left shift

- 逻辑移位：不考虑最高位的符号位
- 算术移位：考虑并保留最高位的符号位
- 10100110：移动3位
  - 逻辑右移：00010100，逻辑左移：00110000
  - 算术右移：11110100，算术左移：10110000

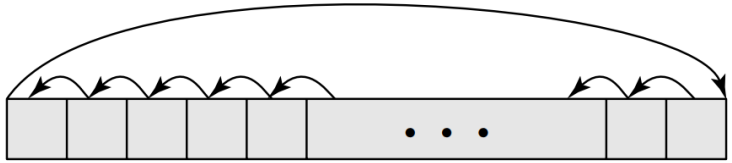
Rotate/Cyclic shift 旋转/循环移位

- Rotate right：Each number moves one digit to the right, and the rightmost digit moves to the leftmost digit 循环右移：每个数字都向右移动一位，最右边的那一位移到最左边的位上
- Rotate left：Each number moves one digit to the left, and the leftmost digit moves to the rightmost digit 循环左移：每个数字都向左移动一位，最左边的那一位移到最右边的位上
- One possible use of the loop is to move left circularly, place each bit at the highest bit in turn, and then test the sign bit to determine the value of each bit 循环的一个可能用途是：通过循环左移，将每一位轮流放在最高位，然后进行符号位测试，确定每一位的值

Example



(e) Right rotate



(f) Left rotate

- 循环右移：整体向右移动一位，移出的最右边一位到最左边的位置
- 循环左移：整体向左移动一位，移出的最左边一位到最右边的位置
- 10100110：移动3位
  - 循环右移：11010110
  - 循环左移：00110101

Role of shift 移位的作用

- Right shift: 右移
  - Logical right shift：which is equivalent to dividing an unsigned integer by 2 逻辑右移，相当于对无符号整数除以2
  - Arithmetic right shift: for complement representation, it is equivalent to dividing by 2 算术右移：对于补码表示法，相当于除以2
- Left shift: overflow needs to be considered 左移：需要考虑溢出的情况
  - When there is no overflow, it is equivalent to multiplying by 2 没有溢出时，相当于乘以2
  - When there is overflow, it has different effects on logical shift left and arithmetic shift left 有溢出时，对于逻辑左移和算术左移产生不同的效果

Example

- 48（0011 0000）的移位
  - 逻辑右移：0001 1000（24）
  - 算术右移：0001 1000（24）
  - 逻辑左移：0110 0000（96）
  - 算术左移：0110 0000（96）
- -48（1101 0000）的移位
  - 逻辑右移：0110 1000（没意义）
  - 算术右移：1110 1000（-24）
  - 逻辑左移：1010 0000（没意义）
  - 算术左移：1010 0000（-96）
- 96（0110 0000）的移位
  - 逻辑右移：0011 0000（48）
  - 算术右移：0011 0000（48）
  - 逻辑左移：1100 0000（溢出）
  - 算术左移：0100 0000（溢出）
- -96（1010 0000）的移位
  - 逻辑右移：0101 1000（没意义）
  - 算术右移：1101 0000（-48）
  - 逻辑左移：010 00000（没意义）
  - 算术左移：1100 0000（溢出）

4. Conversion 转换指令

Conversion	Translate	Translate values in a section of memory based on a table of correspondences
	Convert	Convert the contents of a word from one form to another (e.g., packed decimal to binary)

- 转换指令主要目的是改变数据格式，或者对数据格式进行操作
- 比如说，对二进制的数据格式进行转换，转换成十进制，或者从压缩的10进制转换为二进制
- 还有一种是翻译，根据一个表的相应位的值，将内存块中的一些数据翻译成另一些数

5. Input/output 输入/输出指令

- May be specific instructions 可能是特殊指令
- May be done using data movement instructions (memory mapped)
  可能用数据移动指令
- May be done by a separate controller (DMA) 可能是DMA

Input/output	Input (read)	Transfer data from specified I/O port or device to destination (e.g., main memory or processor register)
	Output (write)	Transfer data from specified source to I/O port or device
	Start I/O	Transfer instructions to I/O processor to initiate I/O operation
	Test I/O	Transfer status information from I/O system to specified destination

6. System control 系统控制指令

- Privileged instructions 特权指令
- CPU needs to be in specific state CPU需要处于特定状态
  - Kernel mode 内核状态
- For operating systems use 操作系统使用
- Some control instructions 一些控制指令
  - Read or write a control register 读写控制寄存器
  - Read or write a storage protection key 读写存储保护锁
  - Access to process control blocks in a multiprogramming system 在多程序系统中访问进程控制块

7. Transfer of control 控制转移

Why need transfer 为什么需要转移?

- The first scenario is that we need to repeat some instructions

  第一个场景是，我们需要重复执行某些指令

  - Multiplication of vector or matrix is easy to implement if circular statements are used 向量或矩阵的乘法，如果用循环语句，就很容易实现
  - Need use the transfer instruction, starting from the end of the loop body 需要用到转移指令，从循环体的最后回到开始
  - It is almost impossible without a transfer instruction 需要用到转移指令

7. Transfer of control 控制转移

- The second scenario is that when we write a program, we often need to judge which operation to do next according to a calculation result 第二个场景是，我们在编写程序的时候，经常需要根据一个计算结果，来判断下一步需要做哪个操作
  - When calculating division, you can first verify whether the divisor is 0. If it is 0, you can directly report an error 计算除法的时候，先可以验证除数是否为0，如果为0，直接报错就可以
  - Transfer instructions are required, and the instructions to be executed in the next step are determined according to the judgment results 需要有转移指令，根据判断结果确定下一步需要执行的指令

7. Transfer of control 控制转移

- The third scenario is that when we write programs, we often use procedures or functions 第三个场景是，我们编写程序，经常会用到过程或函数
  - Break a large program into several small parts, and then process them separately 将一个大的程序分解为若干个小的部分，然后分别处理
  - Procedures or functions can be called multiple times 过程或函数可以多次调用
  - Transfer instructions must be used when calling procedures or functions 过程调用的时候，必须要用到转移指令

Role of control transfer 控制转移的作用

- Normal execution of instructions 一般的指令执行
  - PC（program counter ）stores the address of the next instruction to be executed 程序计数器中保存的是下一个要执行的指令地址
  - After the instruction retrieval is completed, PC automatically adds 1 to point to the next instruction address 取指完成后，程序计数器自动加1，指向下一个指令地址
- Control transfer instructions 控制转移指令
  - Determine the next instruction to be executed according to the execution result of the current instruction 根据当前指令的执行结果，去决定下一步需要执行的指令
  - Change the original instruction execution order 改变原有指令执行顺序

Types of control transfer 类型

Transfer of control	Jump (branch)	Unconditional transfer; load PC with specified address
	Jump Conditional	Test specified condition; either load PC with specified address or do nothing, based on condition
	Jump to Subroutine	Place current program control information in known location; jump to specified address
	Return	Replace contents of PC and other register from known location
	Execute	Fetch operand from specified location and execute as instruction; do not modify PC
	Skip	Increment PC to skip next instruction
	Skip Conditional	Test specified condition; either skip or do nothing based on condition
	Halt	Stop program execution
	Wait (hold)	Stop program execution; test specified condition repeatedly; resume execution when condition is satisfied
	No operation	No operation is performed, but program execution is continued

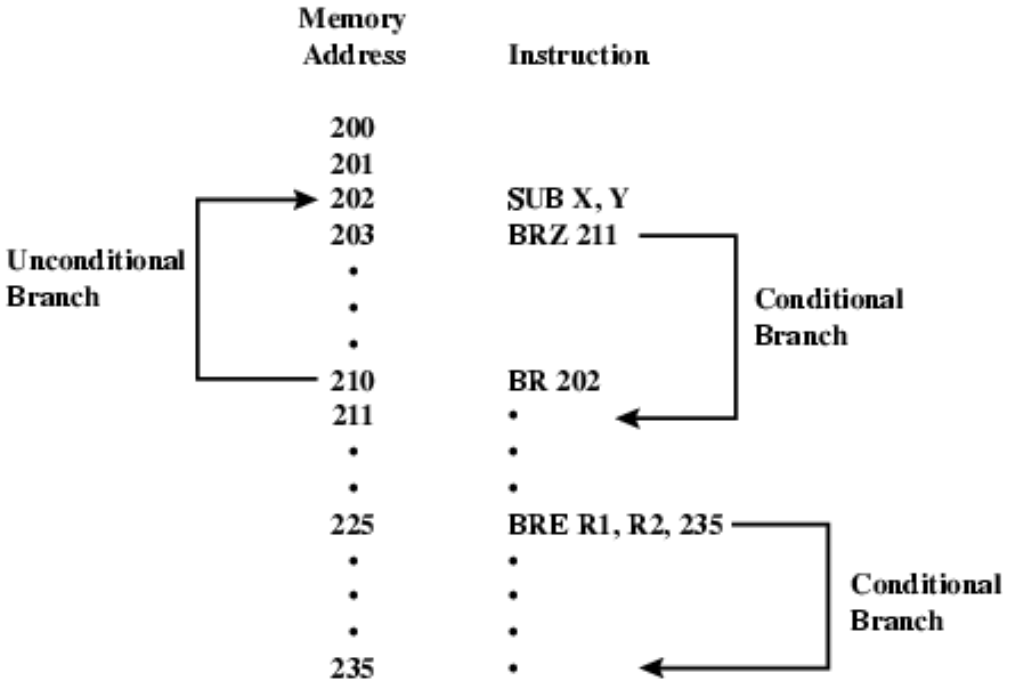
Some control transfer instruction 控制转移指令

- Branch 分支指令
  - Also called jump 也称为jump，跳转
  - Take the address of next instruction to be executed as an operand of current instruction 把下一个要执行的指令的地址，作为当前这个指令的的一个操作数
- Skip 跳步指令
  - Skip execution of the next instruction 跳过下一个指令的执行
  - It is not necessary to specify the address of the next instruction in the instruction 不需要在指令中指明下一个指令的地址
- Subroutine call 过程调用指令
  - Call the procedure code to execute the procedure调用过程代码，执行这个过程
  - After execution, return to the point where the call occurred and continue to execute next instruction 执行完成之后，返回到调用的发生点，继续执行下一个指令

Branch instruction 分支指令

- Also called jump instructions 也称为跳转指令
  - The operands involve the address of the next instruction to be executed 操作数中包含下一步要执行的指令的地址
- For conditional branch instructions, the branch is made only if a certain condition is met 对于条件跳转指令，只有当特定条件满足后，分支才会发生
  - Otherwise, executes next instruction in sequence 否则顺序执行
- Usually the condition is taken as a result of an operation (arithmetic or logic) 通常特定条件就是算术或逻辑运算的结果

Example of branch instruction 分支指令举例



- 包括无条件分支和条件分支
- 202号指令，对x和y执行减法运算，然后判断结果是否为0。如果为0，就转到执行211号指令。如果不为0，继续执行204~209号指令
- 210号指令是一个无条件转移指令，转到202号指令
- 225号指令进行r1和r2的对比，如果相同，就执行235号指令。否则继续执行226号指令

Skip instruction 跳步指令

- The skip instruction includes an implied address 跳步指令包含一个隐含的地址
- The skip implies that one instruction be skipped 一个指令被跳过
  - The implied address equals the address of next instruction plus one instruction-length 隐含地址等于下一条指令的地址加上一条指令长度

```
301
…
309   ISZ   R1
310   BR    301
311 …
```

ISZ: increment and skip if zero

- 309指令，对R1进行加1，判断R1是否为0，如果为0，就跳过310，直接执行311。如果R1不为0，执行310
- 而310就是进行跳转，直接跳转到301
- 实现了一个循环。循环的终止条件是R1为0

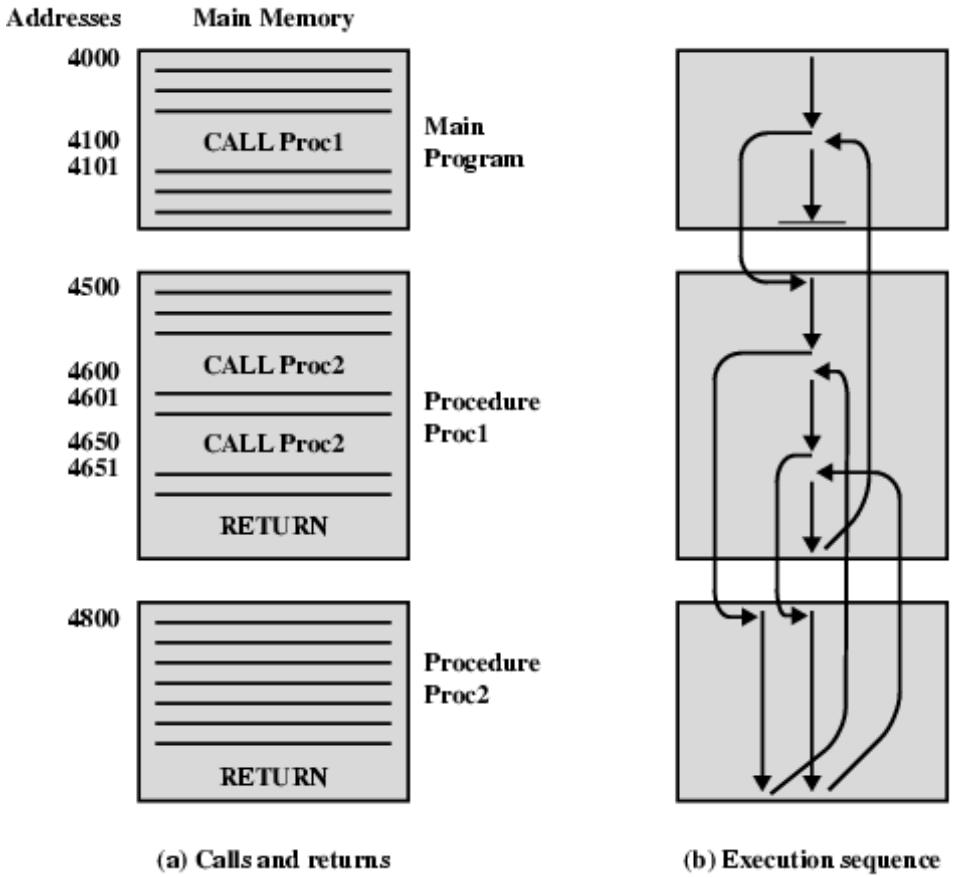
Procedure call instruction 过程调用指令

- A procedure is a subroutine, that is, a computer program, which can perform certain functions 过程是一个子例程，一段计算机程序，可以完成一定的功能
- Write the general function as a procedure, which can be called many times in the program 将通用的功能写成一个过程，在程序中可以多次调用
  - Save the workload of programming 节省编程的工作量
  - Memory space occupied by programs is also reduced 程序的存储空间也减小
- Through procedure writing, modular programming is carried out to improve the efficiency of programming 通过过程的编写，进行模块化的编程，提高了编程的效率

Procedure call instruction 过程调用指令

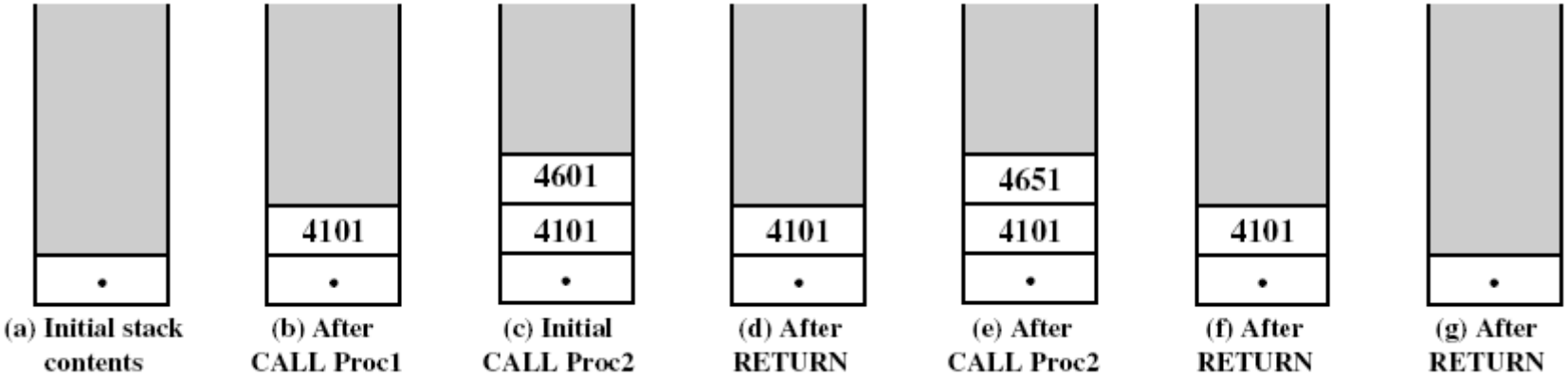
- It is invoked by a calling instruction and returned by a return instruction 它由调用指令调用，并由返回指令返回
  - Procedure call can be nested 过程调用能够嵌套
  - Each procedure call is matched by a return in the called program 每个过程调用和返回匹配
- The CPU must save the return address，it also need to pass parameters to the procedure in one of the following ways CPU需要保存返回地址，还需要通过某种方式返回参数
  - Register 寄存器
  - Start of called procedure 被调用过程的开始
  - Top of stack 栈顶

Nested procedure calls 嵌套过程调用



- 主程序中调用过程1
- 过程1中有2个调用过程2的步骤
- 每次调用完必须有返回的指令，返回到调用的地方
- 返回地址的保存和使用方式，一般采用栈来完成

Use of stacks 栈的使用



- 初始化时，栈是空栈
- 调用过程1时，需要把返回地址，也就是4101压栈
- 过程1中调用过程2时，需要把返回地址4601压栈
- 第一个过程2执行完成，把4601弹出，这样就返回到过程调用前的状态
- 第二次调用过程2，同样把返回地址压栈
- 过程执行结束，弹出4651。过程1执行结束，弹出4101，继续执行主程序

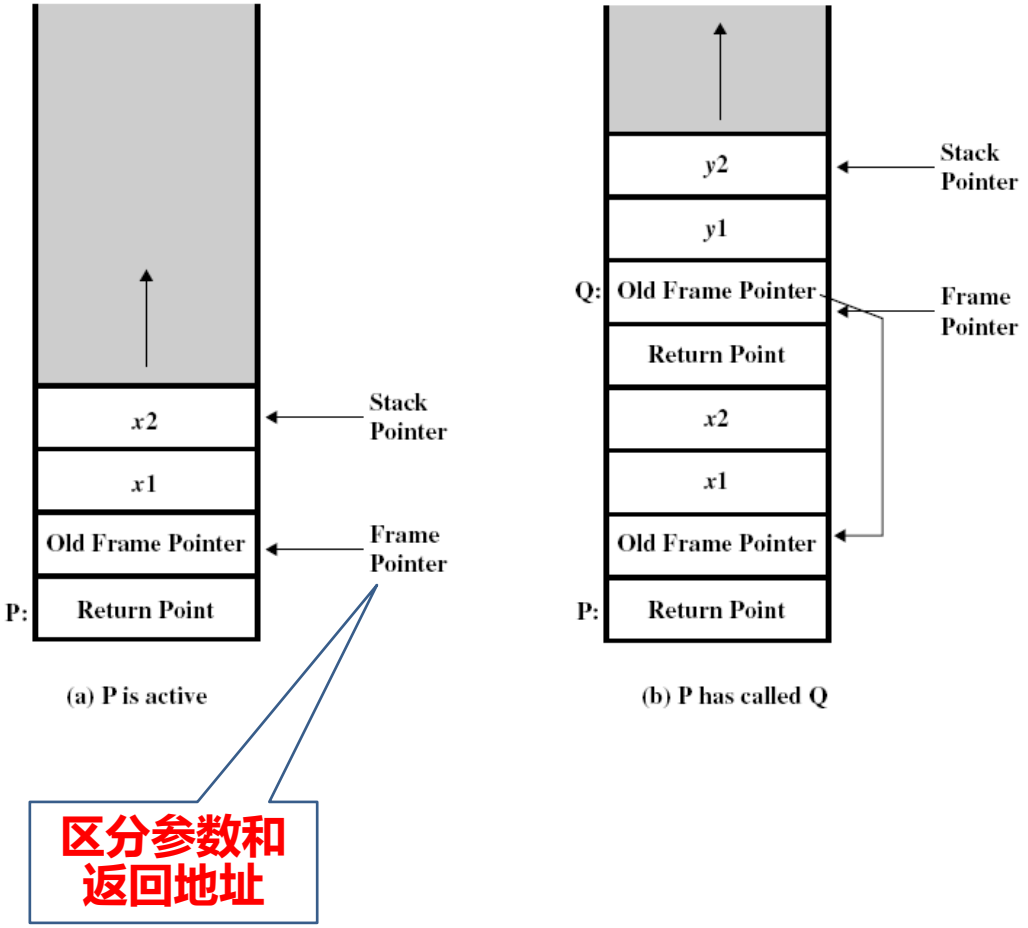
Passing parameters 参数传递

- Pass parameters is important to the procedure call 参数传递对过程调用很重要
- Using registers 使用寄存器
  - Must assure that the registers are used properly 必须确保寄存器正确地使用
- Using memory cells 使用内存单元
  - It is difficult to exchange the variables 交换变量比较困难
- Using stack 使用栈
  - more flexible 用栈更灵活

Stack as parameter passing 栈用作参数传递

- When a procedure is called 当过程调用时
  - Stack the return address 返回地址压栈
  - Stack parameters to be passed to the called procedure 需要传递给被调用过程的参数压到栈
- When return 过程返回时
  - Return parameters can also be placed on the stack 返回时，返回参数也可以放在栈里
- All above stacked info for the procedure is called a stack frame 为过程调动而存储的栈信息称为栈帧

Passing parameters using stack 用栈传递参数

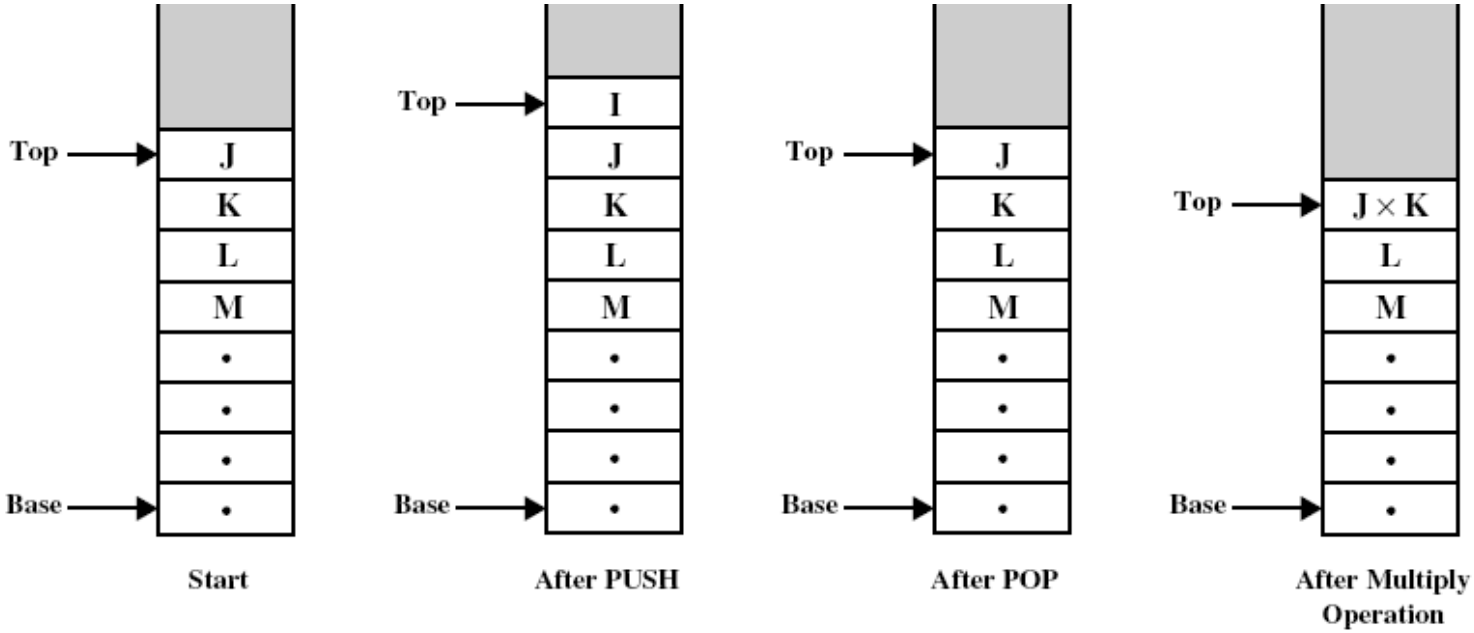


- 主程序调用P的时候，先把返回地址压栈，然后将之前的帧指针地址保存，之后再将x1和x2这两个参数压到栈里面
- P调用Q的时候，先把返回地址压栈，然后将老的帧地址保存，再将P要传给Q的y1和y2地址压栈
- 通过栈，完成了参数、返回地址的传递

Stack 栈

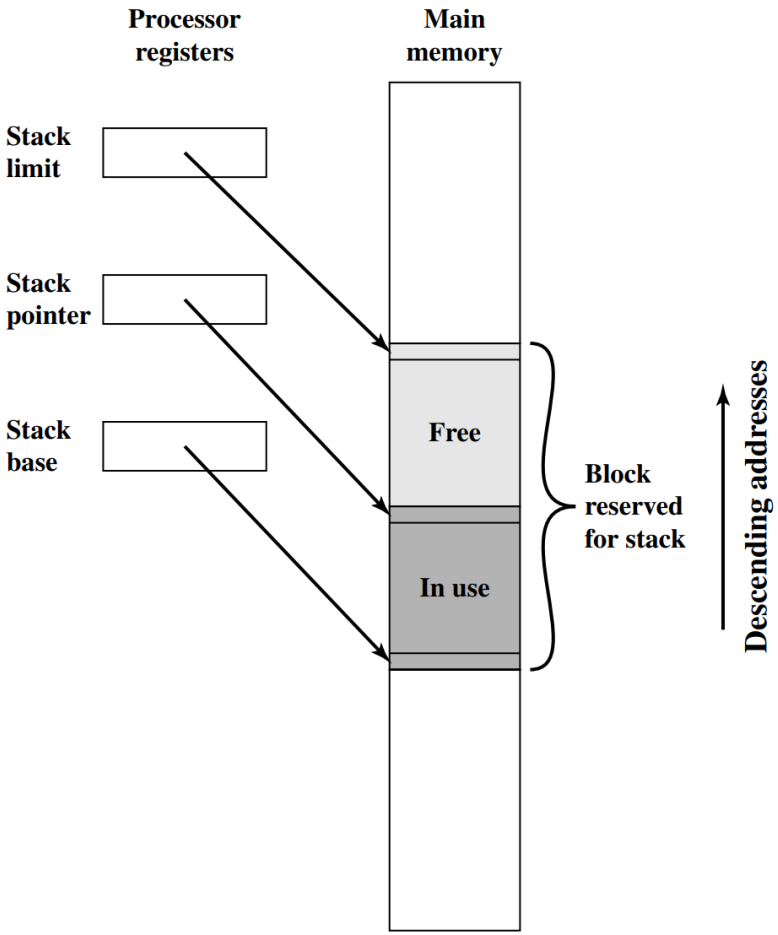
- Queues work in two basic ways 队列有两种工作方式
  - FIFO：first in first out 先进先出队列
  - LIFO： last in first out 后进先出队列
- Stack is a LIFO 栈属于后进先出队列
- A stack is an ordered set of elements, only one of which can be accessed at a time 栈是一个有序元素的集合，一次只能取栈的一个元素
- The point of access is called the top of the stack 栈存取的那个点称为栈顶

Stack operation 栈操作



- 栈里面的最后一个元素称为栈底。
- 栈里面的元素的总数量称为栈的长度，栈的长度是可变的
- 栈里的元素只能从栈顶压到栈里，或从栈顶取出
- 元素放到栈里的过程叫压栈或入栈，从栈里取出元素叫出栈

Typical stack organization 典型栈组织方式



- CPU的寄存器中，定义了三个栈寄存器：栈基址地址寄存器，栈界限地址寄存器，栈指针
- 栈基址是栈所占的内存块的底部地址
- 栈界限是栈所占内存块的顶部地址
- 栈指针指向当前栈顶地址，也就是最后一个在栈里的数据所在的内存地址。当有新元素入栈，或有元素出栈，栈指针都要相应地上移或下移
- 栈地址一般是往低地址方向增长的，也有往高地址方向增长

Outline

- Machine Instruction Characteristics 机器指令特征
- Types of Operands 操作数类型
- Intel x86 and ARM Data Types x86和ARM数据类型
- Types of Operations 操作类型
- Endian Support 端序支持

Example of Endian 端序的例子

- Suppose we want to store a 32-bit hex value 12345678 to address 184 假定我们要存一个12345678的16进制数到地址184

Address	Value
184	12
185	34
186	56
187	78

Address	Value
184	78
185	56
186	34
187	12

184	185	186	187
12	34	56	78

big-endian ordering

存储单元

184	185	186	187
78	56	34	12

little-endian ordering

存储的值

Endian 端序

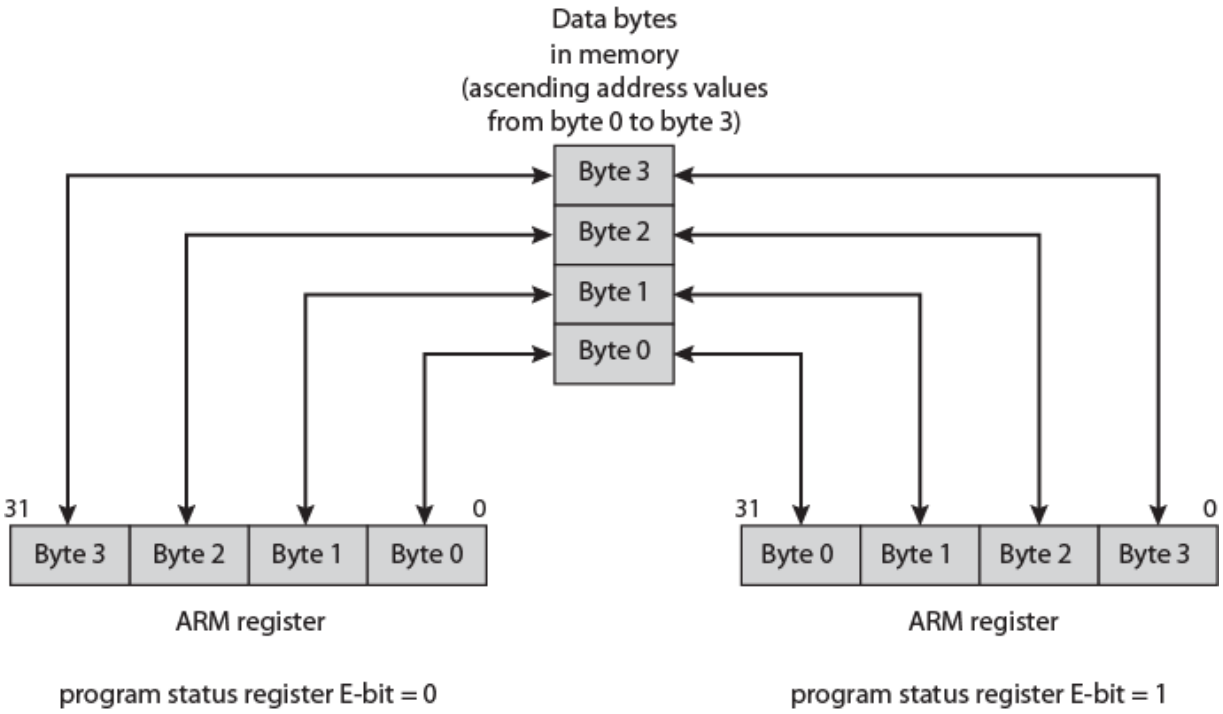
- Big endian 大端端序
  - The most significant byte in the lowest numerical byte address 最高的地址位存放最低的数字位
  - Equivalent to the left-to-right order of writing 相当于从左到右写的顺序
- Little endian 小端端序
  - The least significant byte in the lowest numerical byte address最低地址位存放最低的数字位
  - Reminiscent of the right-to-left order of arithmetic operations in arithmetic units 相当于从右到左进行数学计算
- Machines from different manufacturers may adopt different endian 不同厂家的机器，可能会采用不同的端序

Standard? 标准?

- Pentium (x86), VAX, Alpha are little-endian 奔腾，VAZ，Alpha是小端端序
- IBM 370, Motorola 680x0 (Mac), and most RISC are big-endian IBM370，68000，RISC是大端端序
- Internet is big-endian 互联网是大端端序
  - Makes writing Internet programs on PC more awkward! PC上写互联网程序就很麻烦！
  - WinSock provides HtoI and ItoH (Host to Internet & Internet to Host) functions to convert Winsock接口提供转换函数

ARM endian support ARM双端端序

- ARM supports two endian ARM支持两种端序
- E-bit in system control register 系统寄存器控制
- E-bit=0 is the big endian; if E-bit=1, it is the little endian E-bit=0，就是大端端序，如果E-bit=1，就是小端端序



Key Terms

Address	Conditional branch	Machine instruction	Procedure return	stack
Arithmetic shift	Instruction set	operand	Push	
Bi-endian	Jump	Operation	Reentrant procedure	
Big endian	Little endian	Pop	Rotate	
branch	Logical shift	Procedure call	Skip	

Summary and Question

- 小结
  - 这节课我们对机器指令的特征进行了讨论，并对指令中的操作数类型和操作类型进行了分析。
- 问题
  - 问题1：指令中需要包括哪几个要素？
  - 问题2：指令中的操作数有哪些类型？
  - 问题3：指令有哪些操作类型？

Summary and Question

- 问题
  - 问题1：指令中需要包括哪几个要素？

    操作码，源操作数引用，目的操作数引用，下一指令地址
  - 问题2：指令中的操作数有哪些类型？

    数值，地址，字符，逻辑数据
  - 问题3：指令有哪些操作类型？

    传送，算术运算，逻辑运算转换，系统控制，输入输出，控制转移

Assignments

- Review questions:
  - 12.1 ~ 12.5，12.7 ~12.11
- Problems:
  - 12.6，12.13

谢谢大家！