北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

Computer Organization and Architecture

Chapter 11

Instruction Sets: Addressing Modes and Formats

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 central processing unit 中央处理器
  - Instruction sets: characteristics and function 指令集的特征和功能
    - ✓ 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

Preface

We will focus the following contents today:

- The addressing mode and formats of instruction sets 指令集的寻址模式和格式
  - What are the addressing methods for operands in an instruction? How does each work? 指令中的操作数的寻址方式有哪些？如何进行寻址的
  - How does the current mainstream computer address? 主流计算机有哪些寻址方式
  - How to design instruction format? 如何设计指令格式

Outline

- Addressing 寻址
- x86 and ARM addressing modes x86和ARM的寻址模式
- Instruction Formats 指令格式
- x86 and ARM instruction formats x86和ARM的指令格式

What is addressing mode? 什么是寻址模式?

- Elements in the instruction include: opcode, source operand, destination operand, and next instruction address 指令中元素包括：操作码，源操作数，目的操作数，下一指令地址
- Possible positions of operands 操作数可能的位置
  - Memory
  - Register
  - Immediate
  - I/O
- Addressing mode specifies how to obtain an operand of an instruction 寻址模式确定怎么去获得指令中的操作数

Why need addressing mode?

- Addressing is relatively simple when the operand is in a register or immediate 对于操作数在寄存器或立即数的情况，寻址相对比较简单
- If the operand is in memory 如果操作数在内存中
  - The address field of an operand in an instruction cannot be too long 指令中操作数的地址字段不能太长
  - Want to access a large memory space 希望能够访问大的内存空间
- Memory addressing adopts multiple addressing modes 存储器寻址采用多种寻址方式
  - **Balance** the addressable address range, addressing flexibility, addressing complexity and the number of storage units occupied 在可寻址的地址范围、寻址的灵活性、寻址的复杂度以及占用的存储单元数量之间进行平衡

Memory addressing

- Absolute 绝对寻址
- Displacement 偏移寻址
- Indexed 变址寻址
- register indirect 寄存器间接寻址
- memory indirect 存储器间接寻址
- Autoincrement 自动递增寻址
- Autodecrement 自动递减寻址

Advantage 优点

- Expanding addressable address space 扩充了可寻址的地址空间
- Improved addressing flexibility 提高了寻址的灵活性
- Provide better program architecture to help programmers design more flexible programs 提供更好的程序架构，帮助程序员设计更灵活的程序
  - For example, array, pointer based access, etc 比如说数组，基于指针的访问等

Common addressing mode 常用寻址模式

- Immediate 立即数
- Direct 直接寻址
- Indirect 间接寻址
- Register 寄存器寻址
- Register Indirect 寄存器间接寻址
- Displacement (Indexed) 偏移寻址
- Stack 堆栈

Immediate addressing 立即数寻址

- Operand is part of instruction 操作数是指令的一部分
  - Operand = address field 操作数就是地址域
- e.g.
  - ADD 5
  - Add 5 to contents of accumulator 将5加到累加器中
  - 5 is operand 5就是操作数
- No memory reference to fetch data 不需要去内存取数
  - Fast 快
  - Limited range：length of the address field in the instruction is limited
    值域有限：指令中的地址域长度有限
  - Inflexible 不灵活

Immediate addressing diagram 立即数寻址图示

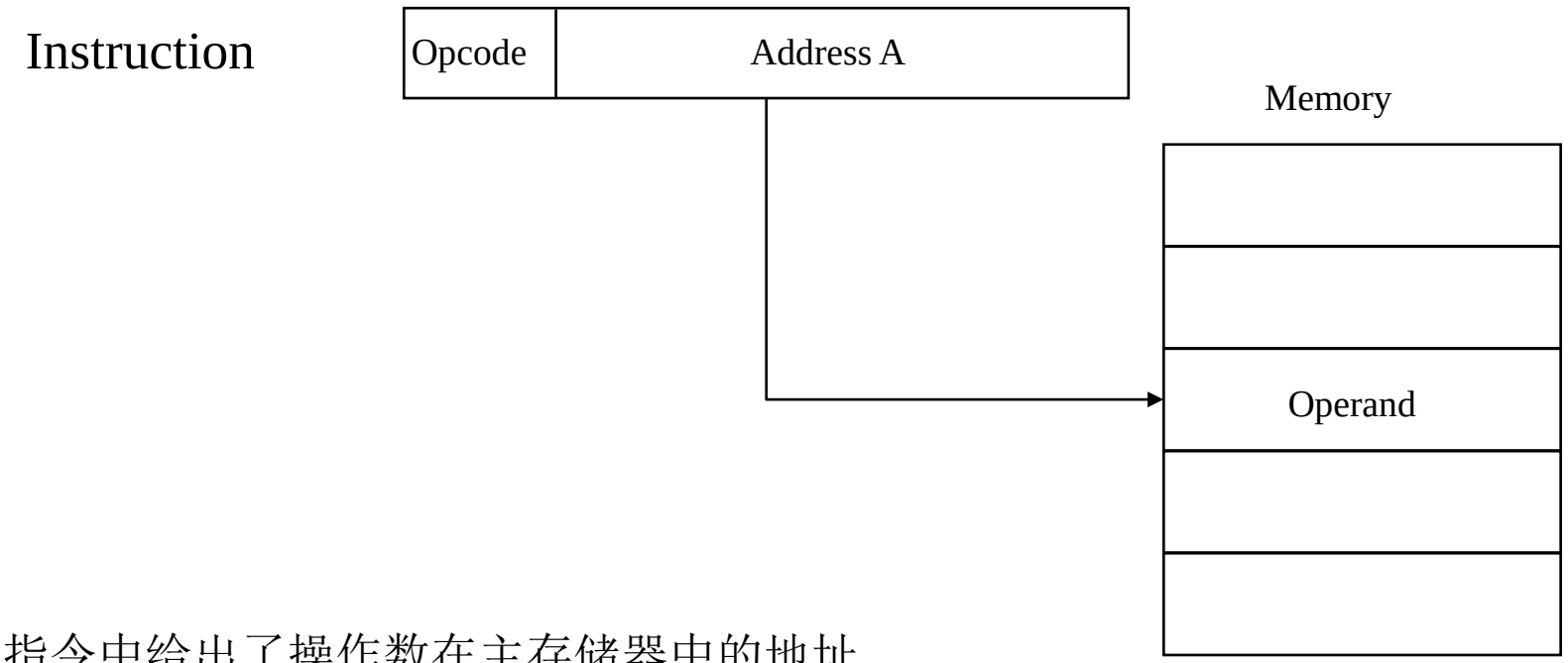
MOV BL,10

- 指令中包含了操作码和立即数
- 复杂一点的指令中，操作数包括立即数，以及其他寻址方式
  - 例如，MOV BL,10。
  - 这个指令把10这个立即数送到BL寄存器中
- 立即数寻址在很多指令中都会用到，但是受到的限制比较大

Direct addressing 直接寻址

- Address field contains address of operand 地址域包含操作数地址
- Effective address (EA) = address field (A) 有效地址=地址域
  - Single memory reference to access data 一次内存引用取数
  - No additional calculations to work out effective address 不需要额外的有效地址的计算
  - Limited address space 有限的地址空间
- e.g. ADD A
  - Add contents of cell A to accumulator 将单元A中的值加到累加器中
  - Look in memory at address A for operand 按地址A到内存中取数

Direct addressing diagram 直接寻址图示



- 指令中给出了操作数在主存储器中的地址
- 通过一次存储器访问，就可以得到操作数
- 操作数的地址直接在指令中。指令的长度有限，能留给直接寻址的地址域的长度有限，导致寻址空间有限

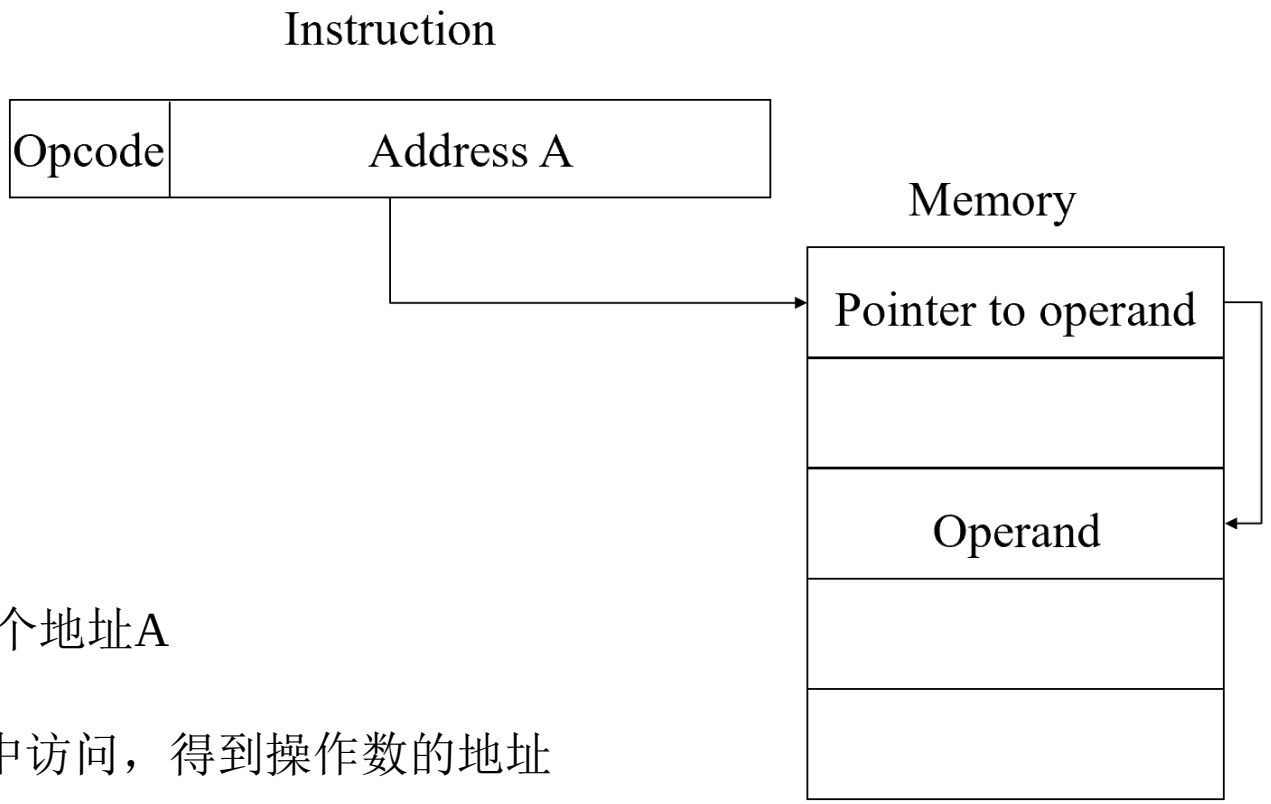
Indirect addressing – 1 间接寻址1

- Memory cell pointed to by address field contains the address of (pointer to) the operand 地址域指向的内存单元包含操作数的地址
- EA = (A)
  - Access the storage unit with address A to obtain the actual address of the operand 访问地址为A的存储单元，得到操作数的实际地址
  - Access the memory according to this address to get the operand 根据这个地址访问存储器得到操作数
  - Memory needs to be accessed twice 需要访问2次存储器
- e.g. ADD (A)
  - Add contents of cell pointed to by contents of A to accumulator 从A地址中得到地址，然后再去取操作数，并和累加器相加

Indirect addressing – 2 间接寻址2

- Large address space 大的地址空间
  - 2^n where n = word length 存储器的字长为n，地址空间为2^n
- May be nested, multilevel, cascaded 可以嵌套，多级和级联
  - e.g. EA = ((A))
  - Effective address is the value of the storage unit pointed to by (A)
    有效地址为（A）指向的存储单元的值
- Multiple memory accesses to find operand 多次内存访问得到操作数
- Hence slower 比较慢

Indirect addressing diagram 间接寻址图示



- 指令中包含了一个地址A
- 根据A去存储器中访问，得到操作数的地址
- 根据这个地址去获得操作数
- 要2次访问存储器，访问速度相对比较慢。

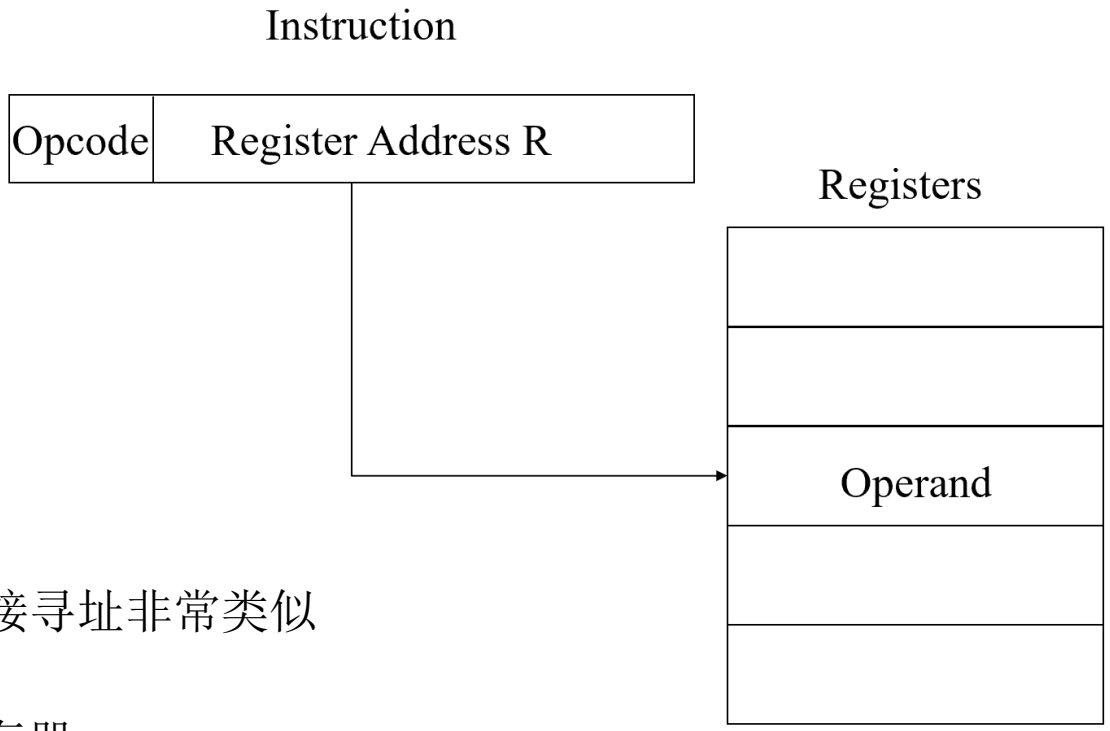
Register addressing – 1 寄存器寻址1

- Operand is held in register named in address field 操作数保存在地址域指定的寄存器中
- EA = R
- Limited number of registers 寄存器数量有限
  - Register address field is 3-5 bits, and the number of accessible registers ranges from 8 to 32 一般寄存器地址字段为3~5位，能访问的寄存器数量从8个到32个
- Very small address field needed 只需要很小的地址域
  - Shorter instructions 指令短
  - Faster instruction fetch 指令取指快

Register addressing – 2 寄存器寻址2

- Similar to direct addressing 和直接寻址类似
  - No memory access 不需要访问内存
  - Very fast execution 速度快
- Very limited address space 很有限的地址空间
- Multiple registers helps performance 多个寄存器能提高性能
  - Requires good assembly programming or compiler writing 要求很好的汇编器和编译器
  - Multiple used operands are placed in registers 经常使用的操作数放在寄存器中

Register addressing diagram 寄存器寻址图示

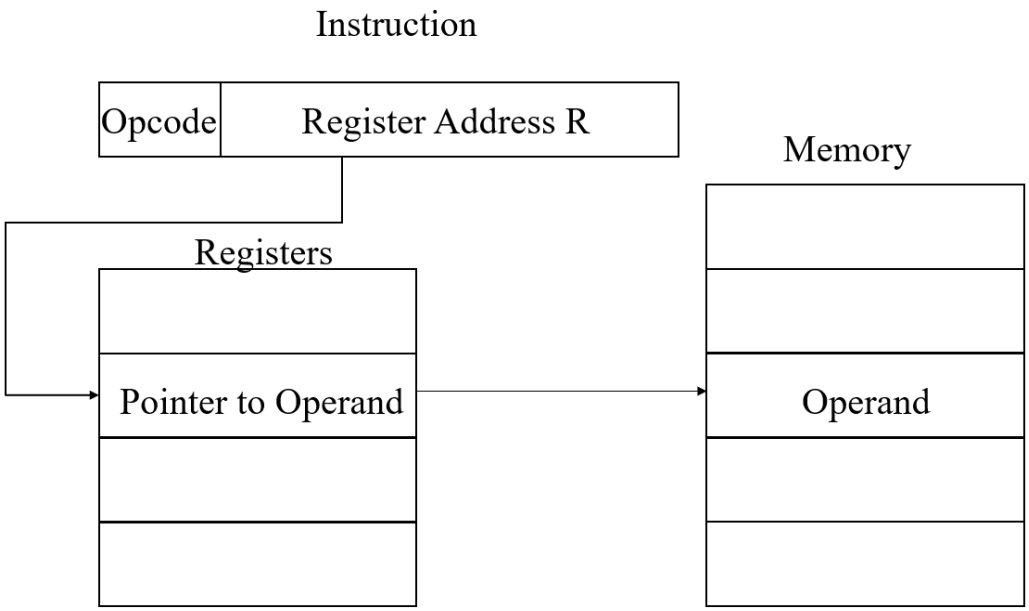


- 寄存器寻址和存储器直接寻址非常类似
- 访问的是CPU内部的寄存器
- 访问寄存器的速度比访问存储器快很多，并且寄存器的数量少，充分利用好寄存器寻址，可以提高处理速度

Register indirect addressing 寄存器间接寻址

- Similar to indirect addressing 类似于间接寻址
- EA = (R) 有效地址=R中内容
  - Operand is in memory cell pointed to by contents of register R 操作数在内存中的位置由寄存器R中的内容决定
- Large address space (2^n) 最大寻址空间2^n
  - n is the word length of register n为寄存器的字长
- Much faster than indirect addressing 比间接寻址快很多
  - One memory access + one register access 1次内存访问+1次寄存器访问

Register indirect addressing diagram 寄存器间接寻址图示

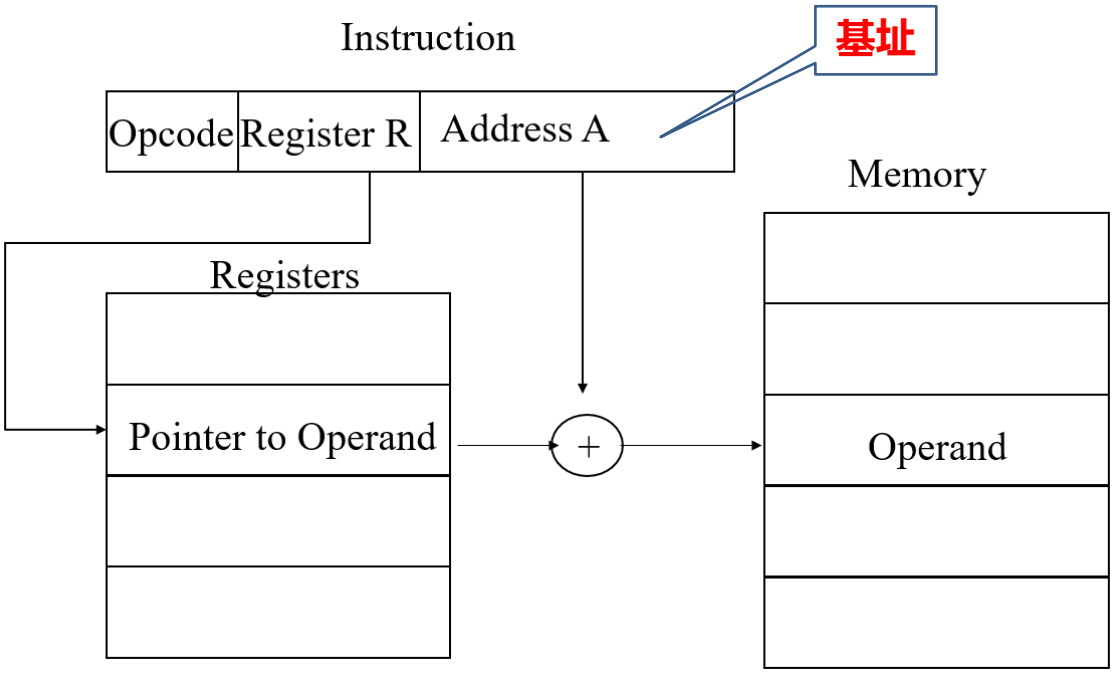


- 指令中的地址域中是寄存器R，而寄存器R中的值是操作数在主存中的地址
- 经过两次访问，才能得到操作数。第一次是寄存器访问，第二次是存储器访问
- 由于寄存器的访问时间很短，所以寄存器间接寻址的时间，基本上和访问存储器的时间相当

Displacement addressing 偏移寻址

- Add a displacement to the base address to obtain the actual address of the operand 基址加一个偏移量，通过计算得到操作数的实际地址
  - EA = A + (R) 有效地址=基址+偏移量
- Address field hold two values 地址域包括两个部分
  - A = base value 基址
  - R = register that holds displacement 存储偏移量的寄存器
  - or vice versa 或者反过来也可以
- The operand address is the relative address of the base address, which is often used in virtual addresses 操作数地址是基址的相对地址，在虚拟地址中经常用到

Displacement addressing diagram 偏移寻址图示

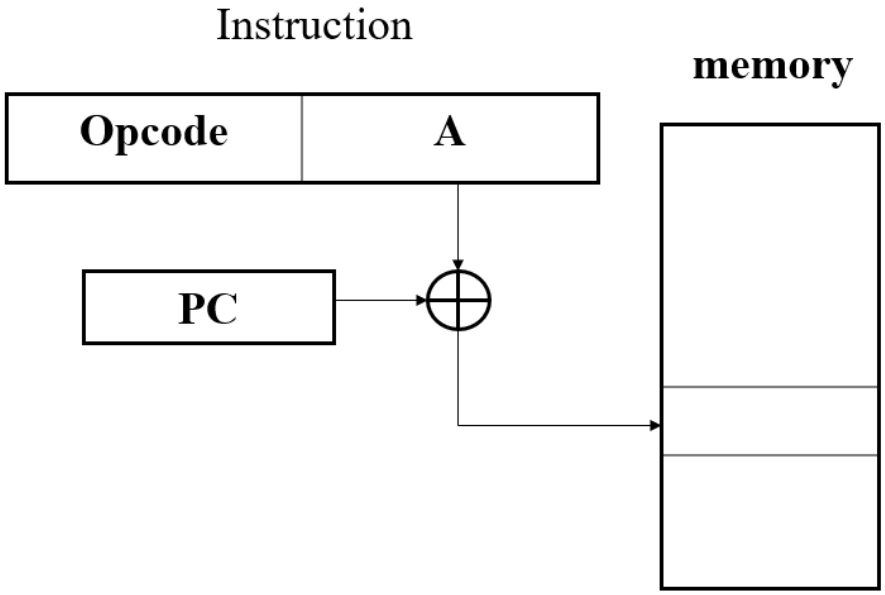


- 指令中包含了2个地址字段，寄存器R和基址A。
- 寻址时，根据R的值，去寄存器中读取操作数的地址偏移量，加上基址A，得到操作数在主存中的地址，访问存储器，得到操作数
- 偏移寻址有三种方式：第一种是相对寻址，第二种是基址寄存器寻址，第三种是变址寻址

Relative addressing 相对寻址

- A version of displacement addressing 偏移寻址的一种方式
  - R = Program counter, PC R为程序计数器
  - EA = A + (PC) 有效地址=偏移量+PC的值
  - obtain the operand from the memory, and the address of the operand comes from PC and A 从内存中获得的操作数的地址来源于PC和偏移量A
- Locality of reference & cache usage 局部性引用和cache使用
  - Program counter is instruction address 程序计数器为指令地址
  - Based on the principle of locality, the probability of data in cache is very high, and data access is fast 基于局部性原理，数据在cache中的概率很大，存取数据快

Relative addressing diagram 相对寻址图示

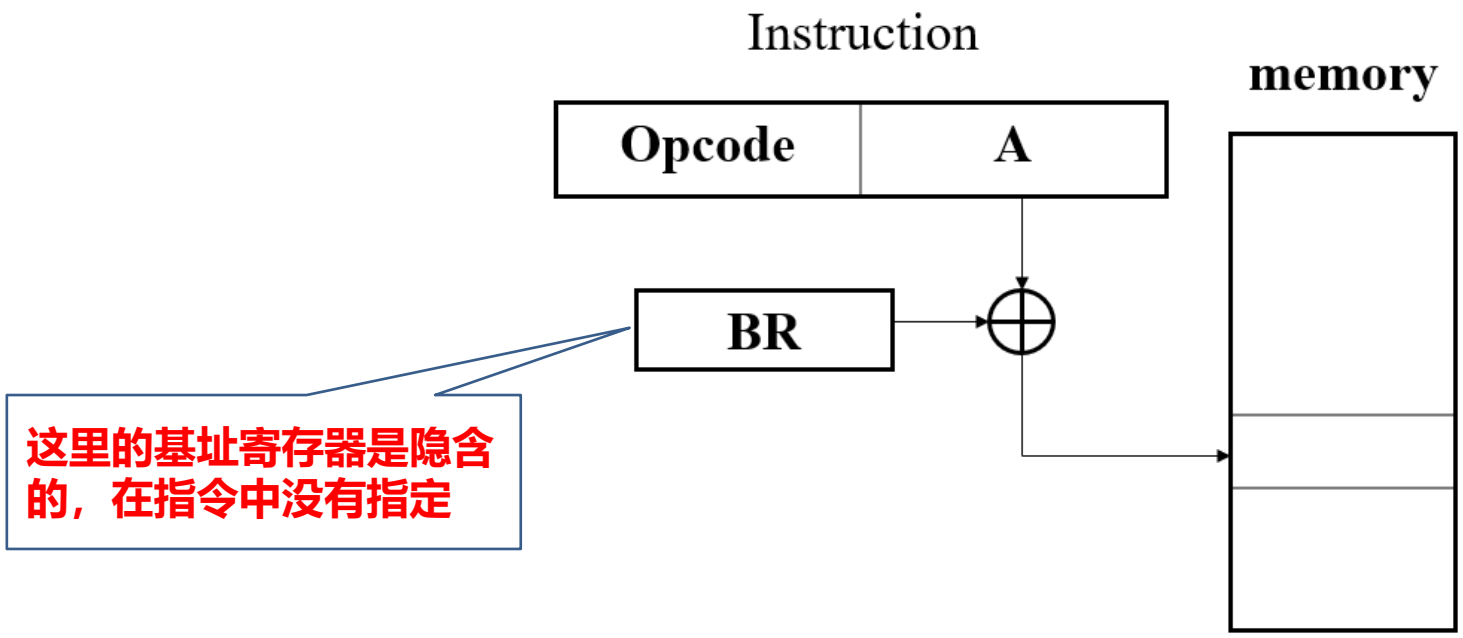


- 相对寻址中，隐含使用了PC作为基址，用指令中地址域中的A作为偏移量
- 通过两个的计算，得到操作数在主存中的地址
- 根据这个地址，访问存储器，得到实际的操作数

Base-register addressing 基址寄存器寻址

- Use a register R as the base register 使用寄存器R作为基址寄存器
  - R holds pointer to base address R包含基址地址
  - R may be explicit or implicit R可能是显示或默认
  - e.g. segment registers in 80x86 is implicit X86中的段寄存器隐含
- The address field in the instruction gives displacement A 指令中的地址字段给的地址是偏移量
- The operation of R and A can obtain the actual address of the operand 两者的运算，可以得到操作数的实际地址

Base-register addressing diagram 基址寄存器寻址图示

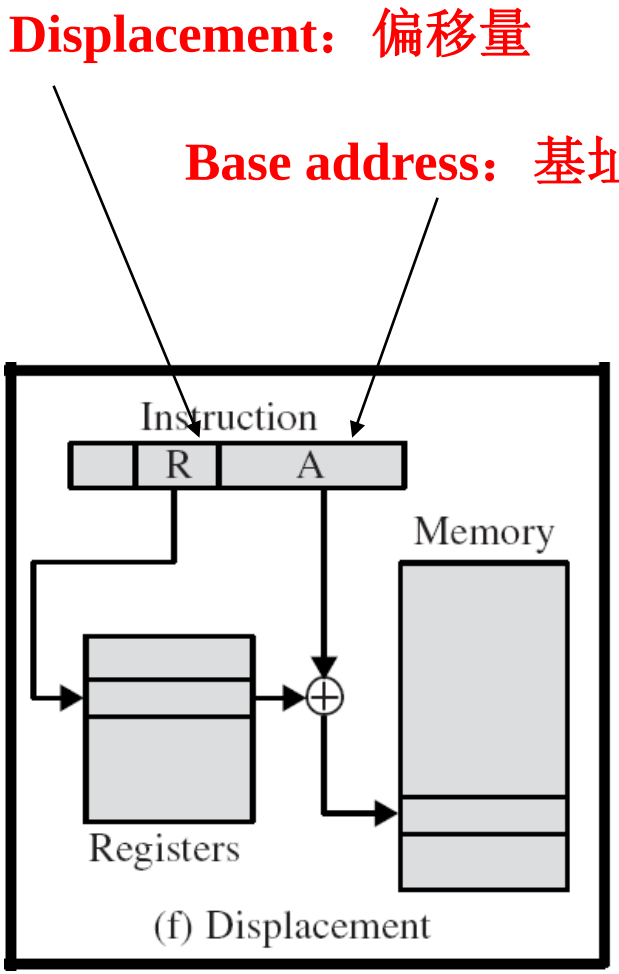


- 基址寄存器BR中包含了寻址的基址，而指令中的地址字段中包含了偏移量
- 这两个相加，得到操作数地址，访问存储器，获取操作数
- 基址寄存器寻址的寻址过程包括：1. 访问1次寄存器；2. 进行一次加法运算；3. 访问一次主存。

Indexed addressing 变址寻址

- One type of displacement addressing mode 偏移寻址的一种类型
- The base address is in the address field, and the offset is in the register 基址在地址域中，偏移量在寄存器中
  - A = base A为基址
  - R = displacement R为偏移量
  - EA = A + R 有效地址=基址+偏移量
- Good for accessing arrays 访问数组比较有效
  - EA = A + R
  - R++

Indexed addressing 变址寻址图示

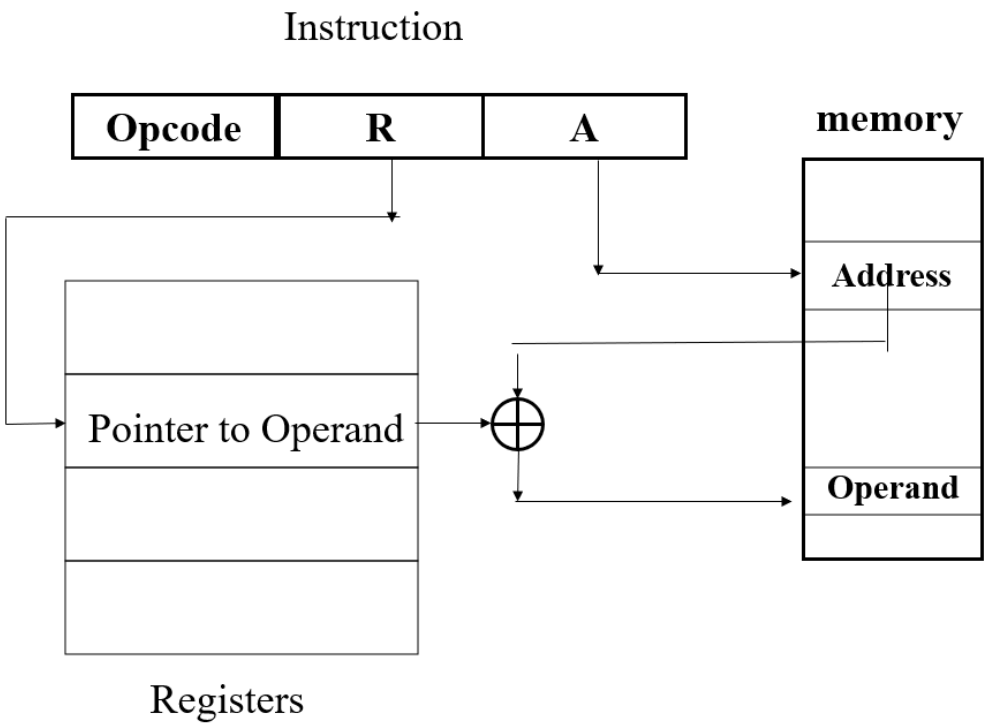


- 变址寻址中，寄存器中的值是偏移量，基址为指令中给出的地址。
- 寻址时，将基址和寄存器中的偏移量进行相加，得到存储器地址
- 根据这个地址访问内存，得到操作数

Combinations 混合寻址

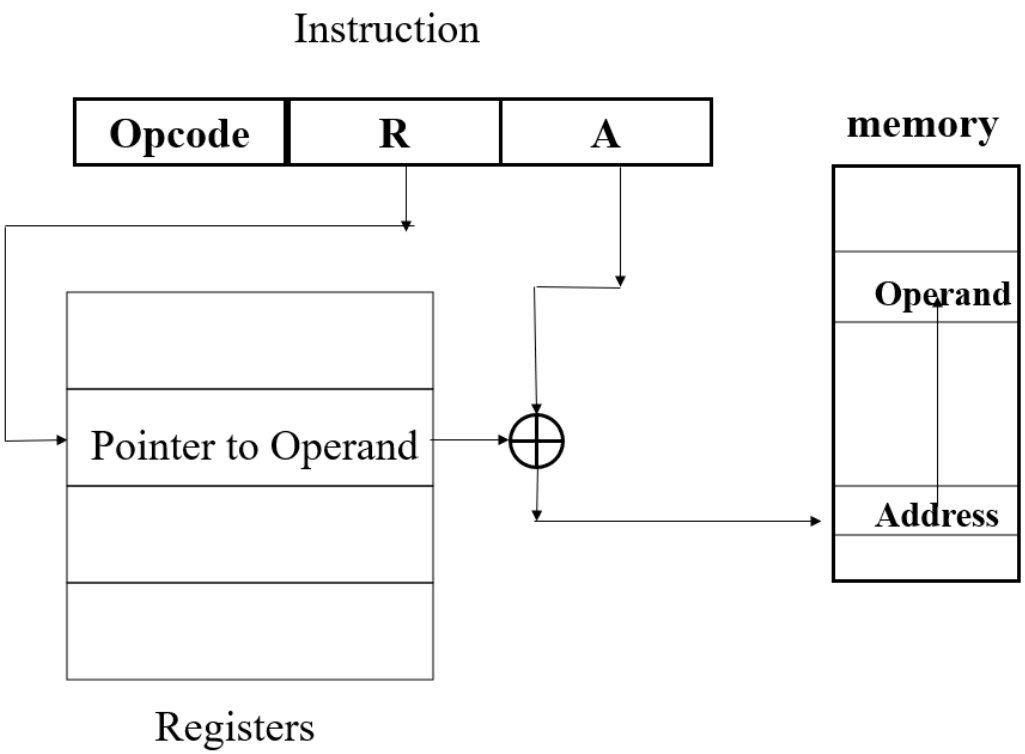
- Post-index：Indexing after indirect addressing 后变址：间接寻址之后再进行变址
  - First get address from memory, then indexing address 先从内存中获取地址，然后再变址
  - EA=(A) + (R) 有效地址=A中的内容+寄存器的值
- Pre-index：Indirect addressing after indexing 前变址：变址后间接寻址
  - Index first, read the memory after getting the address 先变址，得到地址后再间接寻址
  - EA = (A+(R)) 有效地址=存储单元的值，这个存储单元的地址为A+R的值

Post-indexed addressing 后变址寻址



- 指令中地址字段的内容用来访问存储器，获得操作数的直接地址
- 直接地址被寄存器值变址，得到操作数的实际地址，然后访问这个地址，得到操作数。

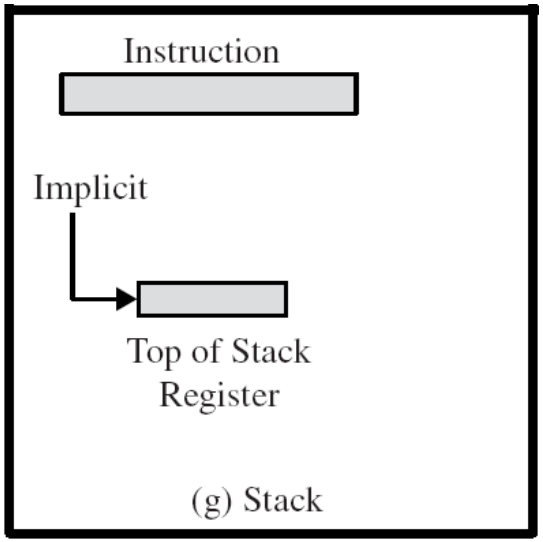
Pre-indexed addressing 前变址寻址



- 指令中地址字段和寄存器先进行变址，得到操作数的间接地址
- 访问存储器，得到操作数的实际地址
- 再一次访问存储器，得到操作数

Stack addressing 栈寻址

- Operand is implicitly on top of stack
  操作数隐含在栈顶
- e.g. ADD
  - Pop top two number from stack 弹出栈顶的2个数
  - Add the two numbers 两个数相加
  - Push the sum 和压栈



Outline

- Addressing 寻址
- x86 and ARM addressing modes x86和ARM的寻址模式
- Instruction Formats 指令格式
- x86 and ARM instruction formats x86和ARM的指令格式

Swapping 交换

- Problem: I/O is so slow compared with CPU that even in multi-programming system, CPU can be idle most of the time
  问题：I/O太慢了，所以即使是多个程序系统，CPU在大多数时间里还是空闲的
- Solutions 解决方法
  - Increase main memory 增加主存容量
    - Expensive 价格昂贵
    - Leads to larger programs 程序会越来越大
  - Swapping 采用交换的方法

Partitioning 分区

- Splitting memory into sections to allocate to processes (including Operating System) 将内存分为多个区域，分配给进程，包括操作系统
- Fixed-sized partitions 固定大小分区
  - May not be equal size 不一定是等大小
  - Process is fitted into smallest hole that will take it (best fit) 进程放到能容纳它的最小分区
  - Some wasted memory 有浪费的内存
  - Leads to variable sized partitions 导致使用可变大小分区

Variable sized partitions -1 可变大小分区1

- Allocate exactly the required memory to a process 给进程分配正好大小的内存
  - This leads to a hole at the end of memory, too small to use 内存的最后会有一个空块，太小不能用
  - Only one small hole - less waste 只有一个小的空块，浪费不多
- When all processes are blocked, swap out a process and bring in another 当进程都阻塞了，把一个进程交换出去，换一个新的进程进来
  - New process may be smaller than swapped out process 新的进程可能比交换出去的进程小
  - Another hole 带来另一个空块

Variable sized partitions -2 可变大小分区2

- Eventually have lots of holes，called fragmentation 最后，会形成很多的空块，称为碎片
- Solutions 解决方法
  - Coalesce - Join adjacent holes into one large hole 合并法，将相邻的空块合并成一个大的空块
  - Compaction - From time to time go through memory and move all hole into one free block 紧缩法，时不时地对内存进行整理，将所有的空块整合成个大的空块

Another problem：Relocation 重定位问题

- Instructions contain addresses 指令中包含地址
  - Locations of data 数据的位置
  - Addresses for instructions (branching) 指令地址，比如分支
- No guarantee that process will load into the same place in memory 没有机制保证进程在内存中的同一个位置
- Solution：Use logical addresses in instructions 解决方法：指令中使用逻辑地址
  - Logical address - relative to beginning of program 逻辑地址：相对于程序开始的位置
  - Physical address - actual location in memory (this time) 物理地址：内存中的当前实际位置
  - Automatic conversion using base address 通过基址进行自动转换

Paging 分页

- Use paging to solve the problem of memory waste 用分页解决内存浪费的问题
  - Split memory into equal sized, small chunks -page frames 把内存分为等大的小块，称为页帧
  - Split programs (processes) into equal sized small chunks – pages 把进程也分为等大的小块，称为页
  - Allocate the required number page frames to a process 分配需要数量的页帧给进程使用
- Operating System is responsible for the management of page tables 操作系统负责页表的管理
  - A process does not require contiguous page frames 进程不需要连续的页帧
  - Each process uses a page table to record which page frames in memory it uses 每个进程使用一个页表来记录页帧的使用

Paging 分页

- Each process has its own page table 每个进程都有一个自己的页表
- Each page table entry contains the frame number of the corresponding page in main memory 每个页表行中包含进程在内存中对应的页帧号
- Two extra bits are needed to indicate 需要2个特别的位来标注
  - whether the page is in main memory or not 页是否在内存中
  - Whether the contents of the page has been altered since it was last loaded 最近一次加载之后是否发生了改变

Real and virtual memory 实存和虚存

- Real memory 实存储器
  - Main memory, the actual RAM 主存储器，实际的RAM
- Virtual memory 虚拟内存
  - Memory on disk 存储在磁盘上的内存
  - Allows for effective multiprogramming and relieves the user of tight constraints of main memory 多道程序设计的运行更加有效，并且避免了主存大小对程序的限制
- Advantage of virtual memory 虚拟内存的优点
  - You do not need to load all processes into memory 不需要将进程都加载到内存中
  - Running multiple processes simultaneously 同时运行多个进程
  - Improved operational efficiency 运行效率提高

Segmentation 分段

- Paging is not (usually) visible to the programmer 分页对程序员不可见
- Segmentation is visible to the programmer 分段对程序员可见
- Usually different segments allocated to program and data 通常分配不同的段给程序和数据
- May be a number of program and data segments 一个程序可能有多个程序段和数据段

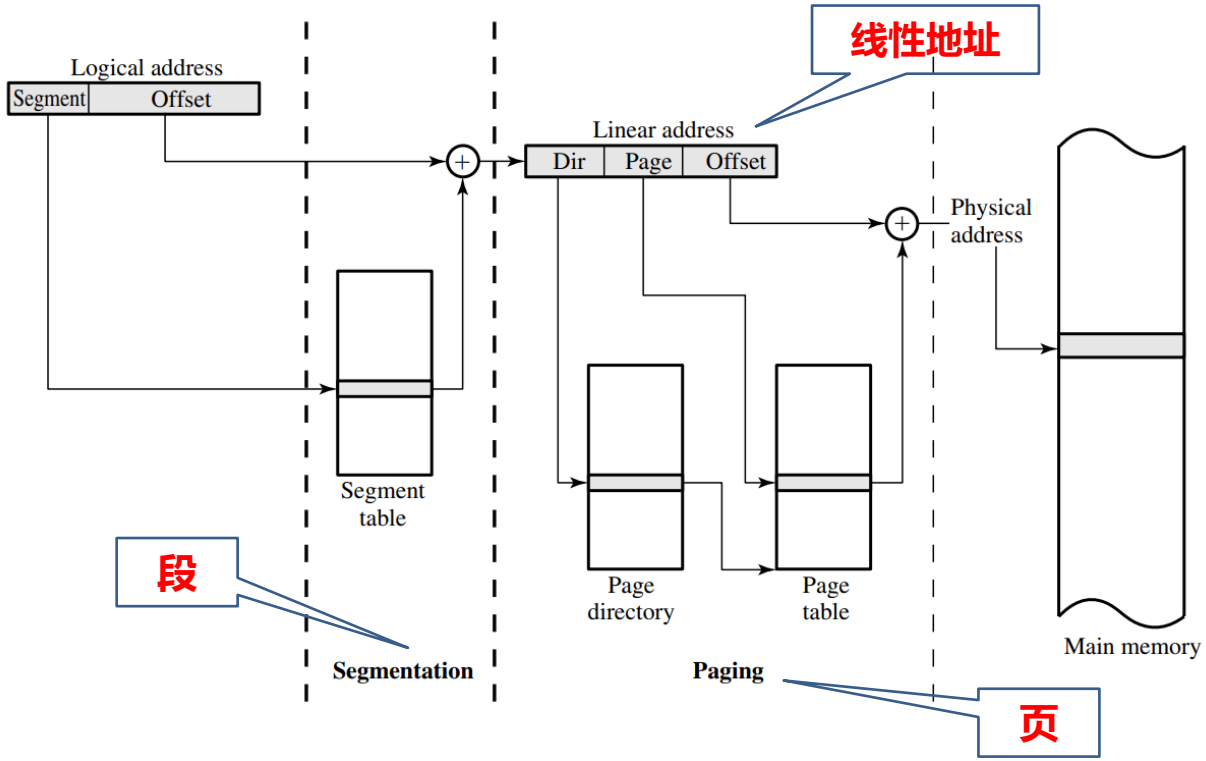
X86 addressing modes x86寻址模式

- X86 adopts a memory management mechanism combining segments and pages X86采用段和页相结合的存储器管理机制
- Virtual or effective address is offset into segment 虚拟地址或有效地址是对段的偏移量
  - Starting address plus offset gives linear address 段起始地址（段基址）加上偏移量，得到线性地址
  - This goes through page translation if paging enabled 如果用了分页机制，还需要增加页地址转换机制

X86 addressing modes x86寻址模式

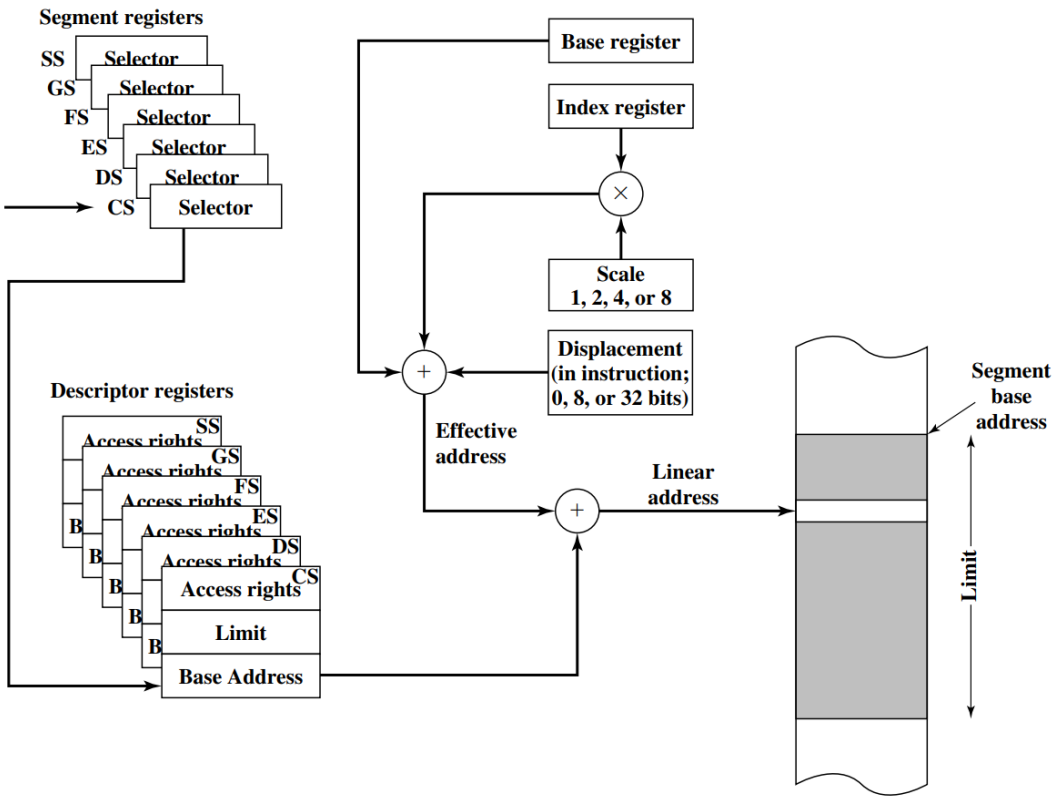
- 9 addressing modes available 9种寻址模式
  - Immediate 立即数
  - Register operand 寄存器
  - Displacement 偏移寻址
  - Base 基址寻址
  - Base with displacement 带偏移量的基址寻址
  - Scaled index with displacement 比例变址带偏移量
  - Base with index and displacement 基址变址带偏移量
  - Base scaled index with displacement 基址带比例变址和偏移量
  - Relative 相对寻址

X86 addressing modes diagram x86寻址模式示意



- 指令中给的逻辑地址包含两个部分：段和段内偏移量
- 查找段表，可以得到段起始地址，加上段内偏移量，得到操作数的线性地址
- 线性地址采用了分页的方式，所以还需要通过页转换机制，得到物理地址，最后通过物理地址查询得到这个操作数。页表采用两级页表的形式。页地址由操作系统管理

X86 addressing mode calculation x86地址模式计算



- 6个段寄存器，每个进程使用哪个段寄存器由指令和执行的上下文来确定。每个段寄存器对应一个段描述符表，记录了段的访问权限，段的起始地址和段的长度
- 基址寄存器和变址寄存器，用于构造复杂的寻址方式
- 基址、变址以及指令中的偏移量计算得到有效地址，加上段地址得到操作数的线性地址，然后再根据分页的规则，得到物理地址

Terms 术语

Effective address 有效地址

Physical address 物理地址

LA: linear address 线性地址

SR : segment register 段寄存器

B: base register 基址寄存器

I : index register 变址寄存器

S: scale factor 变址比例因子

X86 addressing modes x86寻址模式1

Mode	Algorithm
Immediate 立即数	Operand=A
Register Operand 寄存器 32-bit general registers： 8 16-bit general registers： 8 8-bit general registers： 8	LA=R

- 8个32位通用寄存器，分别是EAX、EBX、ECX、EDX、ESI、EDI、ESP、EBP
- 8个16位通用寄存器，AX、BX、CX、DX、SI、DI、SP、BP
- 8个8位通用寄存器，AH、BH、CH、DH、AL、BL、CL、DL

X86 addressing modes x86寻址模式2

Mode	Algorithm
Displacement 偏移寻址	LA=(SR)+A
Base 基址	LA=(SR)+(B)
Base with Displacement 基址带偏移寻址	LA=(SR)+(B)+A
Scaled Index with Displacement 比例变址带偏移寻址	LA=(SR)+(I)*S+A

- 通过段寄存器来确定段的起始地址，然后计算得到线性地址
- 比例变址寻址带偏移量寻址模式中，变址比例因子为1、2、4、8，这个是因为x86是按字节寻址，设置比例因子可以按16位或32位进行变址

X86 addressing modes x86寻址模式3

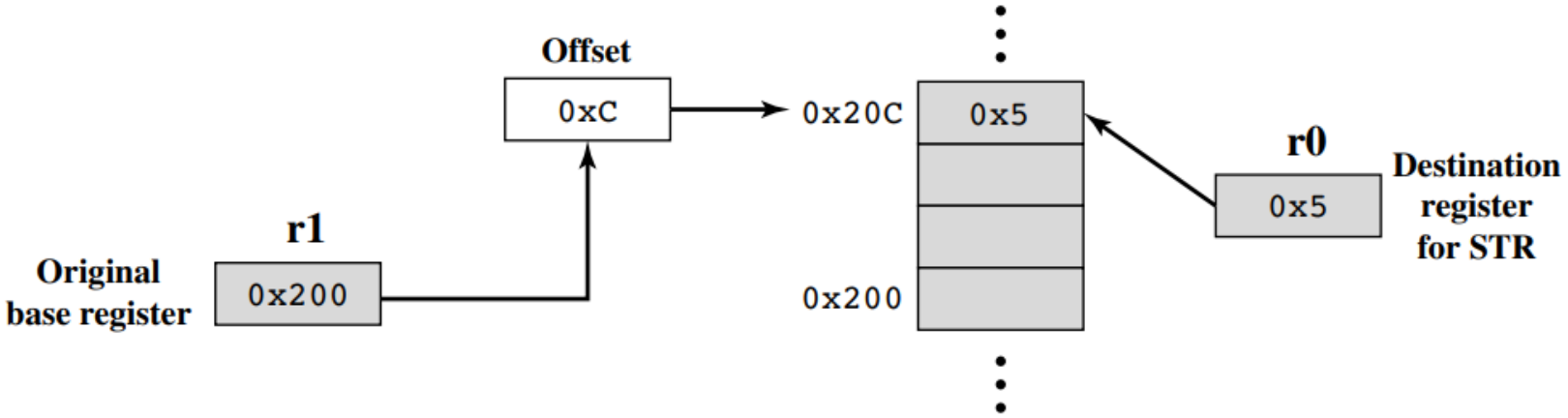
Mode	Algorithm
Base with Index and Displacement 基址变址带偏移	LA=(SR)+(B)+(I)+A
Base with Scaled Index and Displacement 基址比例变址带偏移	LA=(SR)+(I)*S +(B)+A
Relative 相对寻址	LA=(PC)+A

- 相对寻址主要用于控制转移指令
- 将偏移量加到程序计数器中，得到相对于下一个需要执行指令的地址的偏移地址
- 偏移量是一个有符号整数，通过计算，可以增加也可以减少程序计数器中的地址值

ARM addressing modes ARM寻址模式

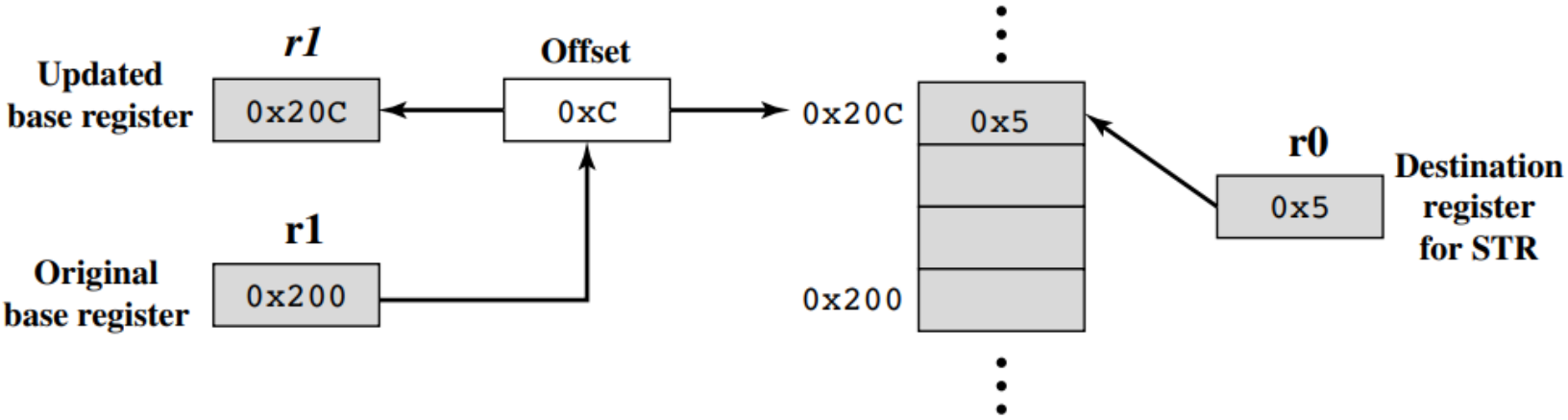
- ARM is a RISC architecture processor ARM是RISC架构处理器
- RISC uses simple addressing modes, but ARM provides more addressing modes 一般的RISC会采用简单寻址模式，但是ARM又提供了比较多的寻址模式
- Only load/store instructions can reference memory 只有加载/保存指令能访问存储器
- Indirectly through base register plus offset 间接寻址，通过基址加偏移
- Base register itself may be updated during addressing 寻址中会可能对基址寄存器本身进行更新
- 3 addressing mode 3种寻址方式

Offset 偏移寻址



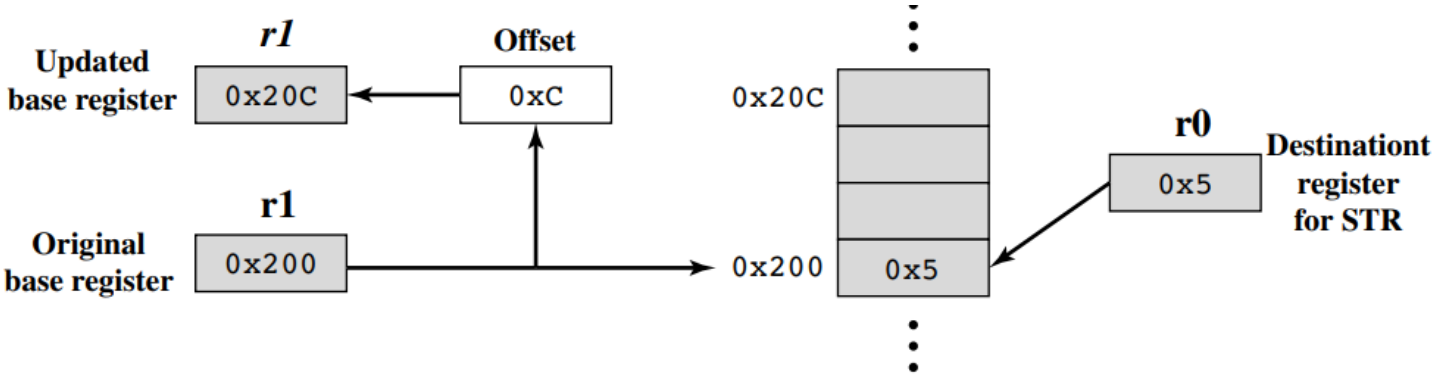
- 偏移寻址：只偏移，不变址。从基址寄存器增加或减少偏移量来形成内存地址
- 例如：STRB r0，[r1,#12]
  - 将r0存放到存储器中，存储器地址为r1的值加上立即数12

Pre-index 前变址



- 内存地址跟偏移寻址一样，基址寄存器增加或减少偏移量来形成内存地址
- 内存地址会写回到基址寄存器，基址寄存器的值会增加或减少一个偏移量
- 例如：STRB r0，[r1,#12]！
  - 这里！就是标识是前变址
  - 寻址完成后，r1寄存器的值变成了r1-12

Post-index 后变址



- 操作数的地址就是在基址寄存器的值
- 寻址完成后，基址寄存器的值会增加或减少一个偏移量，相当于寻址完成后，基址寄存器自身增加或减少了一个偏移量
- 例如：STRBv r0，[r1],#12
  - #表示后变址
  - 寻址用r1地址，同时r1寄存器的值变成了r1-12

ARM addressing modes ARM寻址模式

- Base register acts as index register for pre-index and post-index addressing 在前变址和后变址中，基址寄存器相当于变址寄存器
- Offset either immediate value in instruction or another register 偏移量可以是一个立即数，也可以是另一个寄存器
- If register，scaled register addressing available 如果是寄存器，可以实现比例寻址
  - Offset register value scaled by shift operator 通过移位实现比例变址
  - Instruction specifies shift size 指令中指明移位的数量

Addressing of data process 数据处理寻址

- Data Processing 数据处理
  - Register addressing 寄存器寻址
  - Value in register operands may be scaled using a shift operator 寄存器中的操作数可以通过移位来改变大小
  - Or mixture of register and immediate addressing 或者寄存器和立即数混合寻址

Addressing of branch 分支指令寻址

- Branch 分支
  - Only immediate 只有立即数寻址
  - Instruction contains 24 bit value 指令包括24位立即数
  - When addressing, this immediate value will be shifted two bits to the left, reaching the boundary of a 32-bit word 寻址的时候，这个立即数会左移两位，到达32位字的边界
  - Shifted 2 bits to the left, which is equivalent to an offset of 26 bits. The effective address range is+- 32MB 左移了2位，所以相当于有26位的偏移量，有效的地址范围是+-32MB

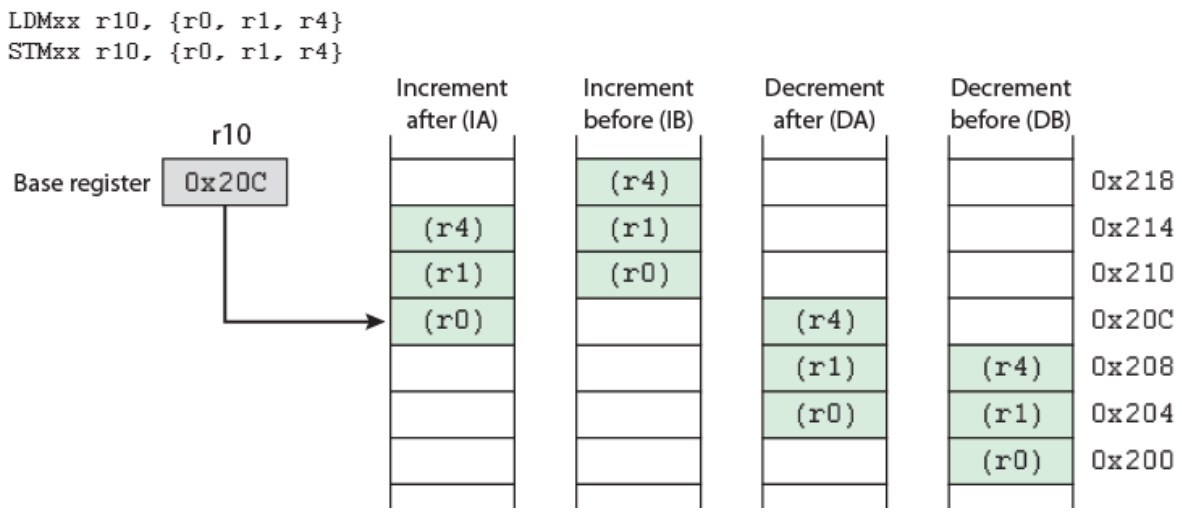
ARM Load/Store Multiple Addressing 多载/多存

- One instruction can load or store multiple data at the same time 一个指令可以同时进行多个数据的加载或存储
  - Load or store a set of general registers 加载或保存一组通用寄存器
- 16-bit instruction field in instruction specifies list of registers 指令中的一个16位字段指定寄存器列表
  - Registers corresponds to a sequential storage unit in memory 寄存器对应到内存中连续的单元
  - Memory unit with the lowest address corresponds to the register with the lowest number 地址最小的存储单元对应编号最小的寄存器

ARM Load/Store Multiple Addressing 多载/多存

- Base register specifies first main memory address 基址寄存器指定内存初始地址
- Four types 四种方式
  - increment after 后递增
  - increment before 前递增
  - decrement after 后递减
  - decrement before 前递减
- Incrementing or decrementing starts before or after first memory access 增加或减少在第一个内存访问前或后开始进行

Multiple addressing diagram 多地址寻址图示



- r10开始的三个单元内容加载到r0，r1，r4这三个寄存器中。R0为低地址，r4为高地址
- 采用后递增，从0x20C开始，连续三个存储单元的内容取出后，分别给r0，r1和r4。采用前递增，第一个存储单元的地址要在基址寄存器中的地址基础上加1，然后取连续三个存储单元的内容取出后，分别给r0，r1和r4。
- 对于后递减，就是从基址寄存器开始，地址递减的连续三个存储单元。对于前递减，就是先在基址寄存器的地址上减1，然后地址递减的连续三个存储单元。

Outline

- Addressing 寻址
- x86 and ARM addressing modes x86和ARM的寻址模式
- Instruction Formats 指令格式
- x86 and ARM instruction formats x86和ARM的指令格式

About instruction set

- Instruction set is the interface provided by the processor to the upper layer 指令集是处理器向上层提供的接口
  - An important symbol of CPU performance CPU性能体现的一个重要标志
  - The rationality of the instruction set has a great impact on the performance of the CPU 指令集的合理与否，对CPU性能的发挥有很大影响
- Therefore, the design of instruction format is the core content of processor design 指令格式的设计是处理器设计中的核心内容

Instruction formats 指令格式

- Instruction include：
  - Opcode 包含操作码
  - Operand(s) (implicit or explicit) and addressing mode 操作数以及寻址模式，操作数可能是显式或隐含
- Instruction formats：How many bits do the parts of the instruction occupy, and in what order 指令格式：包含的每个部分分别占用多少位，先后顺序是怎样
- Layout of bits in an instruction 指令中每一位的编排方式
- Usually more than one instruction format in an instruction set 通常指令集中包含多种指令格式

Key of instruction formats 指令格式中的关键要素

- The width of opcodes: determines number of operation 操作码宽度：决定了多少种操作
  - The more opcodes, the more functions of the instruction set, and the larger the number of bits 操作码数量越多，指令集的功能越多，占的位数也越大
  - If operation code is 4 bits, so there are 2^4=16 operations at most 操作码是4位，那么最多有2^4=16种操作
  - If operation code is 8 bits, so there are 2^8=256 operations at most 操作码是8位，那么最多有2^8=256种操作

Key of instruction formats 指令格式中的关键要素

- The width of operands: effect the instruction length 操作数宽度：影响指令长度
  - The operand takes up a large proportion of the instruction length 操作数占指令的长度比较大
  - Number of operands, addressing mode and size of addressing space have a great impact on the length of instructions 操作数的数量、寻址模式以及寻址空间的大小，对指令的长度影响非常
  - Example：two operands in the instruction，direct addressing ，addressing space is 1G. Each operand needs 30 bits, 4 bytes. Two operands 8 bytes. 指令中有2个操作数，采用直接寻址，寻址空间为1G。那么每个操作数的位长是30位，占4个字节，2个操作数，总共需要8个字节。

Key of instruction formats 指令格式中的关键要素

- Addressing modes: determine the complexity and the length of the instruction 寻址模式：决定了复杂性和指令长度
  - The more complex the addressing mode is, the more operations are required to obtain the physical address of the operand, and the higher the time complexity is 寻址模式越复杂的指令，获得操作数的物理地址所需要的操作就越多，时间复杂度也越高
  - Complex addressing mode can use less address field length to obtain larger addressing space and save instruction length 复杂的寻址模式，可以用较少的地址域长度，获得较大的寻址空间，节省了指令的长度

Design of instruction length 指令长度的设计

- First step in instruction set design is to determine the length of instructions 指令集设计的第一步就是确定指令的长度
  - Instruction length related to opcode number，storage space，memory organization, bus architecture, CPU complexity and speed 指令长度跟操作码种类、存储空间大小。内存组织方式、总线架构、CPU复杂度和速度都有关系
  - These factors together determine whether the instruction set is rich and whether the instruction function and addressing mode are flexible 这些因素共同决定指令集是否丰富，指令功能和寻址方式是否灵活
- Trade off between powerful instruction repertoire and saving space 在强大的指令集和节省空间之间进行权衡

Programmer ‘s perspective 程序员的角度

- More opcode number 操作码种类多
- More operand number 操作数数量多
- More Addressing mode 寻址模式多
- Greater addressing space 更大的寻址空间
- Select the most concise operation code to complete the operation 选择最简洁的操作码去完成操作
- Select the most appropriate addressing mode 选择最合适的寻址方式
- Larger address space available 可以使用的地址空间更大
- Instruction length is required to be longer and longer 指令长度要求越来越长

Hardware design perspective 硬件设计的角度

- The longer the instruction, the more memory space it takes 指令越长，占用的存储空间越大
- Length of instruction should be consistent with the width of the data bus, or an integer multiple 指令的长度应该和数据总线的宽度一致，或者两者之间为整倍数的关系
  - Otherwise, fetched instruction number cannot be integer in the fetching cycle 否则取指周期中不能取到整个数的指令
- Short instructions enable multiple instructions to be fetched at one time, improving processing capacity 较短的指令使得能够一次取多个指令，提高处理能力
  - 32-bit bus can access two 16 bit instructions at a time 32位总线可以一次取到2个16位的指令

Summary

- The operation code and operands should have as many digits as possible 操作码、操作数都希望位数越多越好
- The longer the instruction, the more memory space it takes 指令越长，占用的存储空间越大
- Generally，instruction length is consistent with the bus width，or an integer multiple 一般来说，指令长度和总线宽度一致或整数倍
- In the design of instruction set 在指令集的设计中
  - Every part of the directive needs to be properly planned需要合理规划指令中的每一个部分
  - Seeking the best balance among various design scheme 各种设计方案中寻求一种最佳的平衡

Allocation of bits 位的分配

- After the length of the instruction is determined, each bit in the instruction needs to be allocated reasonably to maximize the use of each bit 确定了指令的长度之后，就需要对指令中的每一位进行合理的分配，最大化利用每一位
- For example, 32 bit long instructions 例如，32位长的指令
  - How to divide 32-bit length into opcodes and operands 如何将32位长分给操作码和操作数
  - If the opcode is long, operands is short 操作码长，操作数的位数就少
  - Variable length opcode, additional bits determine operation 变长操作码，需要额外的位数确定操作
- First, you need to determine the number of operands and opcodes 首先就是需要确定操作码和操作数的位数

Design of operands bits 操作数位的设计

The following factors need to be considered

- Number of operands 操作数的数量
- Number of addressing modes 寻址模式的数量
- Register versus memory 寄存器 vs 存储器
- Number of register sets 寄存器组的数量
- Address range 地址范围
- Address granularity 地址的粒度

Number of addressing modes 寻址模式的数量

- Some opcodes implicitly specify the addressing mode of the operand, which does not need to be specified separately 有些操作码隐含指定了操作数的寻址模式，不需要单独说明
- Sometimes it is necessary to explicitly specify the addressing mode of this operand, and one or more addressing mode bits are required 有时候需要显式指定该操作数的寻址方式，需要一位或多位寻址方式位
- There may be multiple addressing modes in an instruction 一个指令中可能会有多种寻址方式

Number of operands 操作数的数量

- If the instruction only supports one operand, it is troublesome to write the program 如果指令只支持一个操作数，程序写起来很麻烦
- Generally, two operands are supported 一般都支持2个操作数
- Each operand hope an independent addressing mode 每个操作数希望有独立的寻址方式
  - Flexible 比较灵活
  - Need addressing indication bit 需要寻址指示位
- Some processors allow one operand to specify the addressing bit 有的处理器允许1个操作数指定寻址位

Register versus memory 寄存器和存储器

- Data needs to be loaded into CPU through registers for processing 数据需要通过寄存器装入CPU进行处理
- If there is only one register, it does not need to be specified, but it is very troublesome to use 如果只有一个寄存器，不需要指定，但是使用起来非常麻烦
- Several registers are generally provided 一般都提供若干个寄存器
  - Several bits can specify a register, which takes up less instruction bits 几个bit可以指定一个寄存器，占用的指令位数少
- Most processors have more than 32 registers 大多数处理器有32个以上的寄存器

Number of register sets 寄存器组的数量

- Most processors provide only one set of general-purpose registers 大多数处理器只提供一组通用寄存器
  - Store Data 保存数据
  - Store address field in offset addressing mode 保存偏移寻址方式中的地址字段
- Some processors, such as the x86 processor, can provide multiple sets of registers 有些处理器，比如x86处理器，它能提供多组寄存器
  - Divide by function, some store data, some store offset 按照功能进行划分，有些保存数据，有些保存偏移量
  - Opcode implicitly determines which set of registers to use 操作码隐含确定使用哪组寄存器
  - Reduce the number of instructions 减少指令的位数

Address range 地址范围

- In direct addressing, the address range is determined by the length of the address field in the instruction 直接寻址中，地址范围由指令中地址字段的长度确定
  - Instruction length is limited 指令长度受到限制
  - The address range of direct addressing is small 直接寻址的地址范围小
- General use offset addressing 一般用偏移寻址
  - Length of the address register is critical 地址寄存器的长度很关键
  - If the offset is large, the length of the address field in the instruction is also long 如果偏移较大，指令中的地址字段的长度也比较长

Address granularity 地址粒度

- The smaller the addressable address granularity is, the longer the address bits are required 可寻址的地址粒度越小，需要的地址位数越长
- Addressing by byte 按照字节寻址
  - Some operations are more convenient 有些操作会比较方便
  - e.g. character processing 比如对字符进行处理
  - More address bits required 地址位要求更多
- Operate according to words 按照字来进行操作
  - Number of address bits reduced 地址位数减少了
  - Reduced operational flexibility 降低了操作灵活性

Example: PDP-8 instruction format PDP-8指令格式

Memory reference instructions

Opcode			D/I	Z/C	Displacement						
0		2	3	4	5						11

Input/output instructions

1	1	0	Device						Opcode		
0		2	3					8	9		11

Register reference instructions

Group 1 microinstructions

1	1	1	0	CLA	CLL	CMA	CML	RAR	RAL	BSW	IAC
0	1	2	3	4	5	6	7	8	9	10	11

Group 2 microinstructions

1	1	1	0	CLA	SMA	SZA	SNL	RSS	OSR	HLT	0
0	1	2	3	4	5	6	7	8	9	10	11

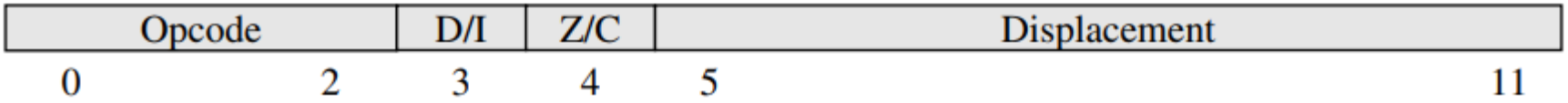
Group 3 microinstructions

1	1	1	0	CLA	MQA	0	MQL	0	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11

- 指令长度12位，字长也是12位
- 只有一个寄存器，累加器
- 3位操作码的格式，最多只有8种指令
- 操作码0~5这6个指令是内存引用指令，6是输入/输出指令，7是寄存器引用指令

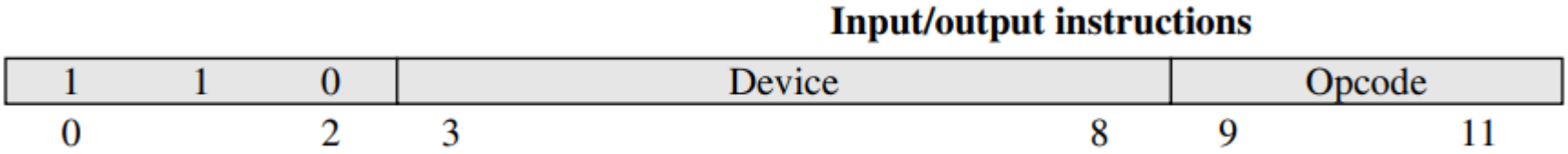
Memory reference instruction 内存引用指令

Memory reference instructions



- 设置了2个修饰符，在指令的3和4位
- 修饰符Z/C是页号指示位，修饰符D/I表示是直接寻址还是间接寻址
- 存储器分为固定长度的页，每页有2^7=128个字
- 页0上有8个专用字是自动变址的“寄存器”，相当于用存储器的空间来完成寄存器的功能。通过这个“寄存器”可以实现前变址

Input/Output instruction 输入输出指令



- 对于输入/输出指令，操作码为110
- 第3-8位，总共6位，用于选择64个设备中的一个
- 后面3位用于指定具体的IO操作
- 相当于把操作码的一部分放到操作数的位置

Register reference instruction 寄存器引用指令

Register reference instructions

Group 1 microinstructions

1	1	1	0	CLA	CLL	CMA	CML	RAR	RAL	BSW	IAC
0	1	2	3	4	5	6	7	8	9	10	11

Group 2 microinstructions

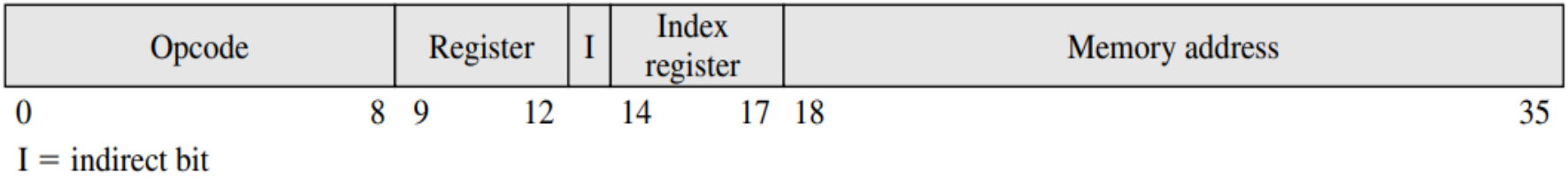
1	1	1	0	CLA	SMA	SZA	SNL	RSS	OSR	HLT	0
0	1	2	3	4	5	6	7	8	9	10	11

Group 3 microinstructions

1	1	1	0	CLA	MQA	0	MQL	0	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11

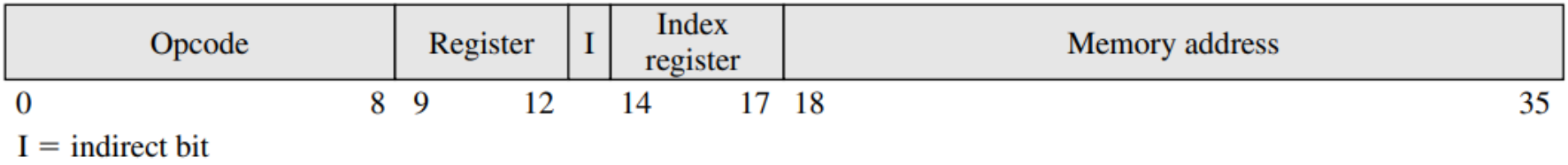
- 操作码为111的指令是寄存器引用指令，也称为微指令。共分为三组微指令
- 用于对寄存器进行操作。每一位可以定义一个操作，并且可以多位进行组合，以完成多个操作
- 有限的12位长指令中，实现了多种寻址方式，并且支持的指令总数有35种

Example: PDP-10 instruction format PDP-10指令格式



- 采用了36位的指令长度和36位的字长
- 为大型分时系统设计，强调编程的便利性，牺牲空间，用空间的代价换取了便利性
- 指令具有如下特点
  - 正交性：地址和操作码无关
  - 完整性：每种数据类型都有完整的操作
  - 直接寻址：都是直接寻址模式

Example: PDP-10 instruction format PDP-10指令格式



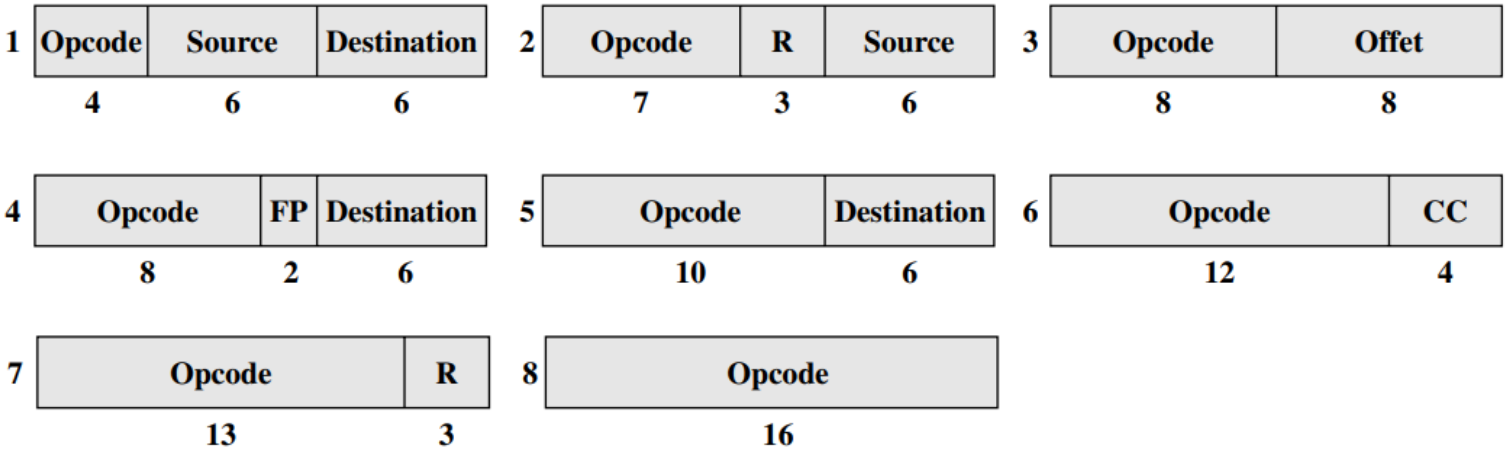
I = indirect bit

- 一般的指令有2个操作数
  - 一个是寄存器，占了4位，最多能有16个寄存器可以引用
  - 另一个在18位的地址域字段中，可以是立即数或存储器地址
- 允许变址寻址。有一位I，标识是否采用变址寻址。如果是变址寻址，I位后面的4位为变址寄存器
- 如果采用直接寻址，最多有2^{18}=256K个存储单元可以寻址
- 提供了变址寻址

Example: PDP-11 instruction format PDP-11指令格式

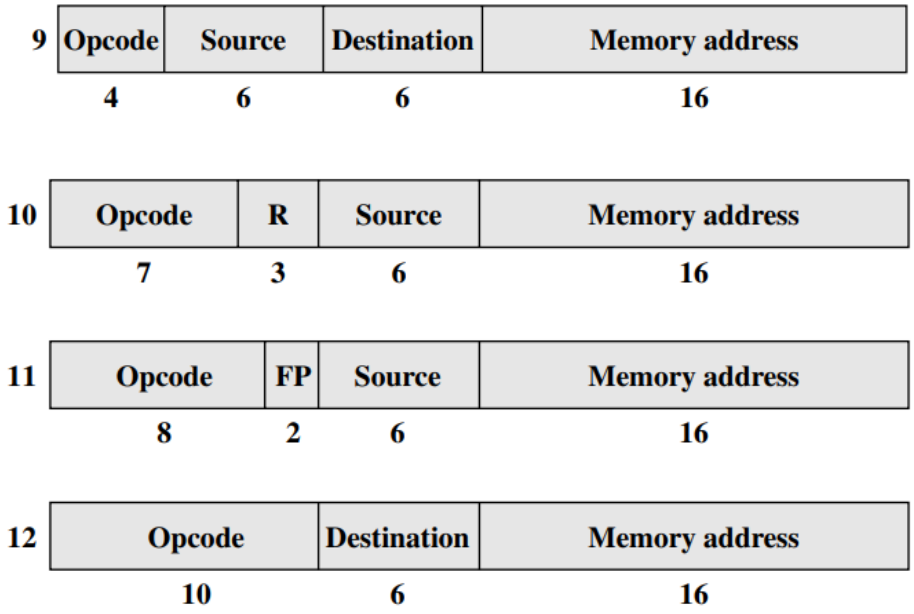
- Variable-Length Instructions：instruction length may be different for different opcodes 变长指令：不同的操作码，指令长度可能不一样
  - More flexible addressing mode 寻址方式更加灵活
  - increase the complexity of the CPU 增加了CPU的复杂度
- PDP-11 adopts three instruction lengths PDP-11采用了三种指令长度
  - 16 bits 16位
  - 32 bits 32位
  - 48 bits 48位
- 13 instruction formats 总共13种指令格式

16 bits instruction format 16位指令格式



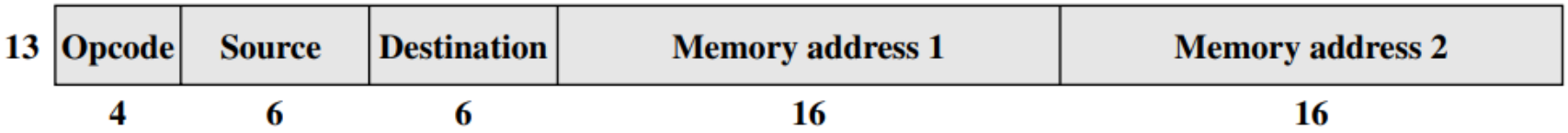
- 指令有8种，包括零地址、单地址、双地址。操作码的长度从4位到16位不等。
- 操作数中，寄存器引用用了6位，其中3位用于指定具体是哪个寄存器，另外3位是指定寻址方式
- 寻址方式和操作码不挂钩，对于任意指令都可以自由选择操作数的寻址方式，增加了操作的灵活性，称为正交性

32 bits instruction format 32位指令格式



- 4种指令格式
- 存储器地址占用了16位，寻址空间为2^{16}=64k
- 操作码从4为到10位不等
- 寄存器引用用了6位，其中3位用于指定具体是哪个寄存器，另外3位是指定寻址方式

48 bits instruction format 48位指令格式



- 只有1种指令格式
- 操作码占用了4位
- 2个存储器地址，占用了32位的长度
- 2个寄存器引用，占12位。每个寄存器引用中，3位指定寄存器，还有3位指定寻找模式

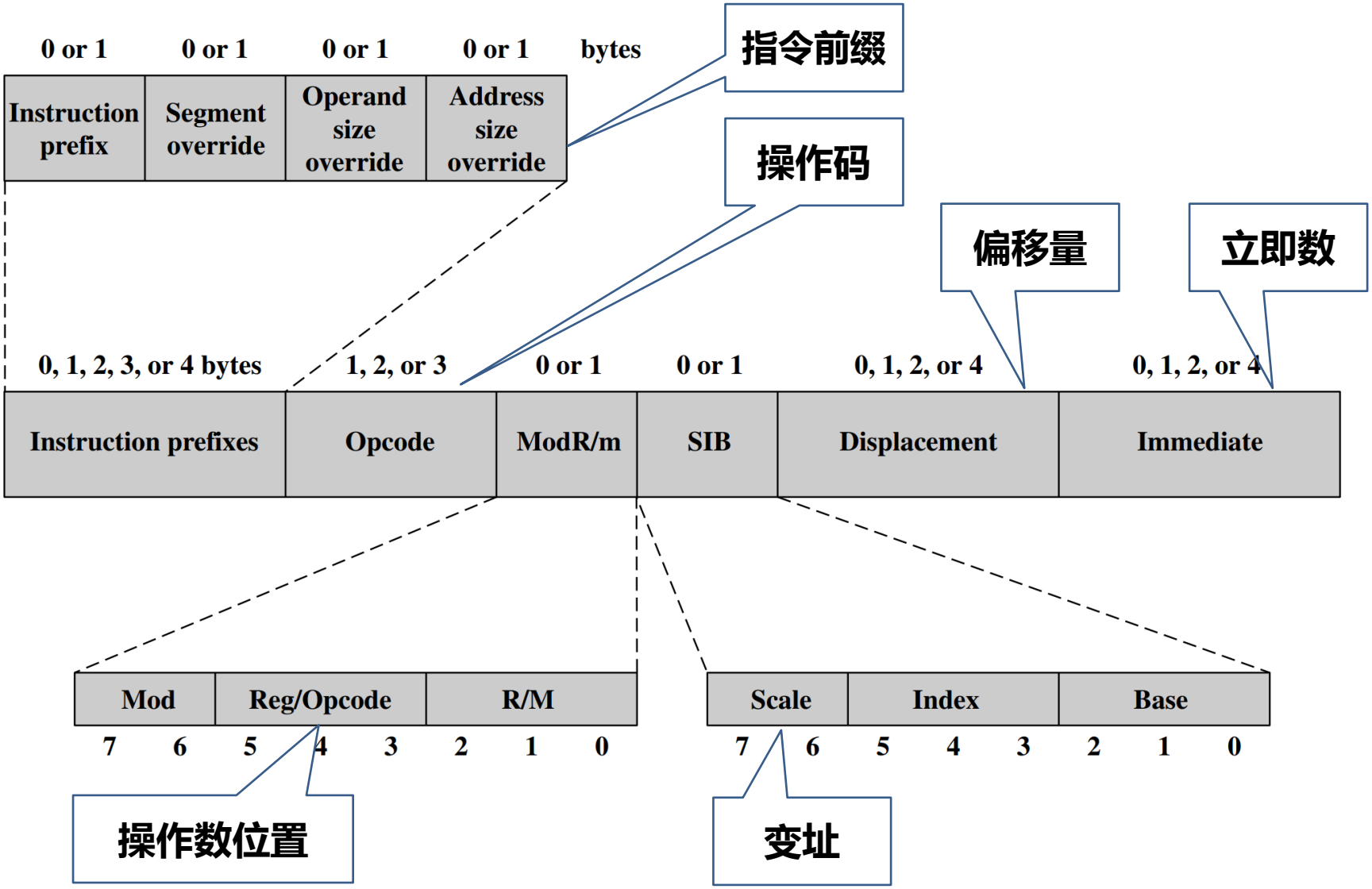
Summary of PDP-11 小结

- Provides rich instruction sets and addressing capabilities 提供了丰富的指令集和寻址能力
  - Variable length instructions increase programming flexibility 变长指令增加了编程的灵活性
  - The complexity of the processor has been greatly improved 处理器的复杂度大大提高
  - Hardware costs also increase 硬件成本也增加
- Many RISC and superscalar machines use fixed length instructions 很多RISC和超标量机器采用了固定长度的指令
  - Reduce processor complexity 减少处理器的复杂性
  - Improve processor performance 提高处理器的性能

Outline

- Addressing 寻址
- x86 and ARM addressing modes x86和ARM的寻址模式
- Instruction Formats 指令格式
- x86 and ARM instruction formats x86和ARM的指令格式

x86 instruction format x86指令格式



Characteristic 特点

- Addressing mode is associated with the instruction opcode 寻址模式和指令操作码关联
- An instruction has only one addressing mode 一个指令只有一种寻址模式
- Only one memory operand can be referenced in an instruction 指令中只能引用一个存储器操作数
- Typical CISC architecture，use complex instruction format 典型的CISC架构，采用复杂的指令格式
  - X86 needs to consider downward compatibility 需要考虑向下兼容
  - Hope to provide richer instructions for compiler developers 希望给编译器开发者提供更丰富的指令

ARM instruction formats ARM指令格式

- Typical RISC architecture 典型的RISC架构
- All the instructions are 32 bits, and the format is very neat 所有指令都是32位，格式规整
- ARM instructions are divided into four categories ARM指令分为四类
  - data processing instructions 数据处理
  - load / save instructions 加载/保存
  - overload / save instructions 多载/多存
  - branch instructions 分支
- All instructions are conditionally executed 所有指令都是条件执行

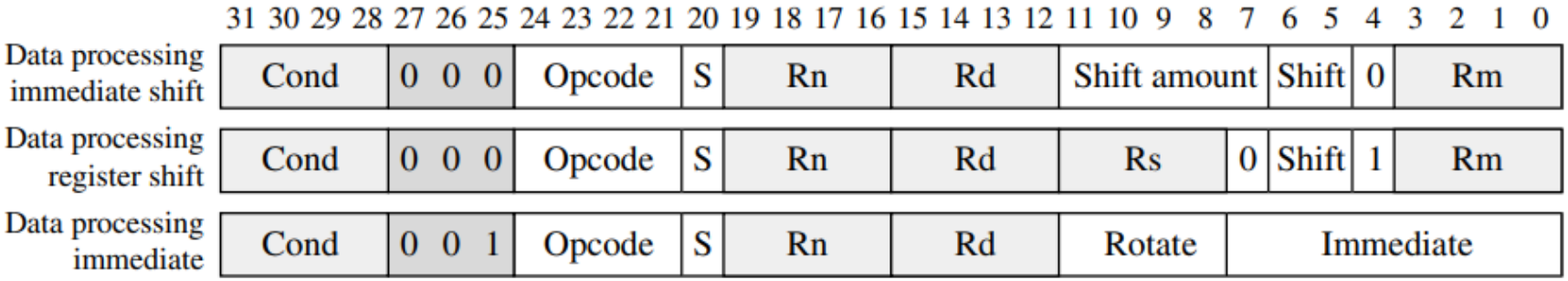
Condition code 条件码

- All instructions are conditionally executed 所有指令都是条件执行
- The instruction contains a 4-bit condition code, which is in the highest 4-bit of the instruction 指令包含一个4位的条件码，在指令的最高4位
- Except for the condition flags 1110 and 1111, all other instructions must meet the conditions before they can be executed 除了条件标志为1110和1111之外，其他的所有指令，都需要满足条件，才能执行

Condition code 条件码

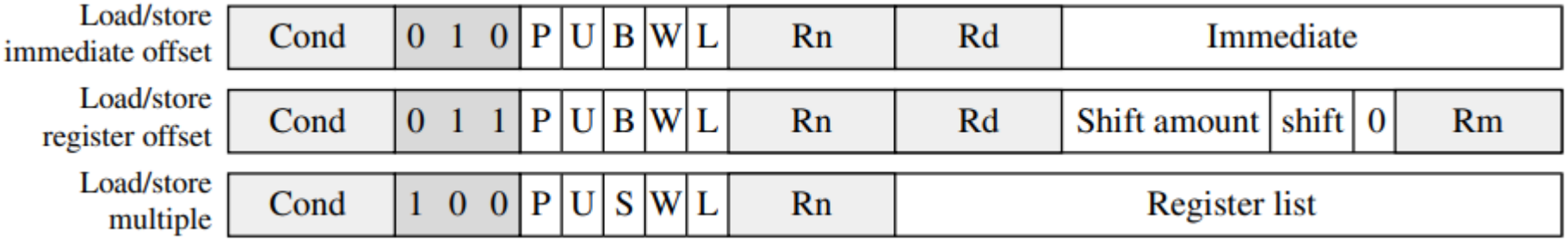
- The condition code includes four condition flags, which are stored in the program status register 条件码包括4个条件标志，保存在程序状态寄存器中
- The four condition flags are N negative flag, Z zero flag, C carry flag, V overflow flag 四个条件标志分别是N负标志、Z零标志，C进位标志，V溢出标志
- For all arithmetic or logic instructions, an S bit is given to indicate whether the instruction modifies the condition flag bit 对所有的算术或逻辑指令，都给了一个S位，指示这个指令是否修改条件标志位

Data processing 数据处理



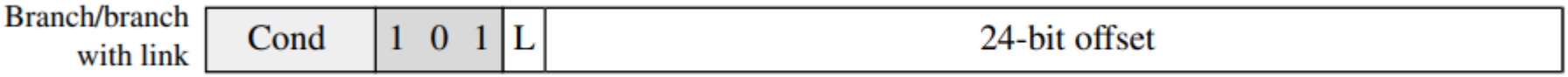
- 数据处理指令类型为000或001。操作码都是4位，s表示是否修改条件标志位。指令中都有三个操作数
- 第一种格式中，目的寄存器Rd，第一个操作数寄存器Rn和第二个操作数寄存器Rm，操作数可以根据shift的标志进行移位，shift amount指明移动多少位
- 第二种格式跟第一种类似，只是移位的位数不是立即数，而是由寄存器Rs来确定
- 第三种格式中，第二操作数是一个立即数，并且可以针对立即数进行循环右移，循环右移的次数由rotate域中的值决定

Load / Store 装载/保存



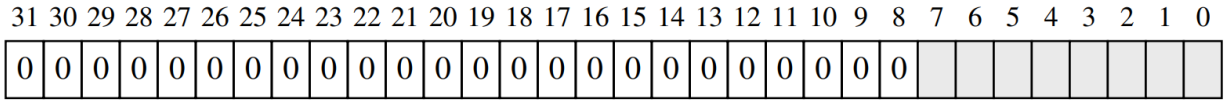
- 加载/保存指令中，指令一般类型为010和011。后面5位标识了寻址模式、数据类型，是字节还是字，以及加载和保存标志。
- 第一种加载/保存指令是立即数偏移指令，指令中给出了12位的偏移量。内存地址就是基址寄存器Rn加上或减去立即数偏移量。
- 第二种指令是寄存器偏移。偏移量在Rm寄存器中，通过shift确定移位操作，移动shift amount位之后得到，然后再和基址寄存器Rn计算，得到内存地址。
- 多载/多存指令中，指令一般类型为100。指令中给了16位的寄存器列表，内存地址在Rn中，是先递增，先递减，还是后递增，后递减，由寻址模式来决定。

Branch 分支

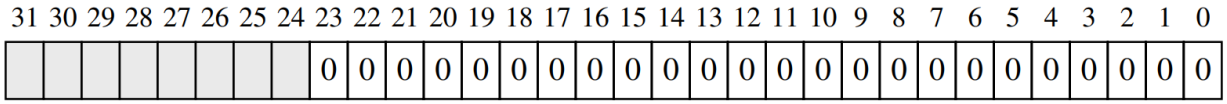


- 分支指令的指令一般类型为101，提供了一个24位的立即数
- 还有一个标志位L，这个标志位决定返回地址是否保存在连接寄存器，也就是link register中。

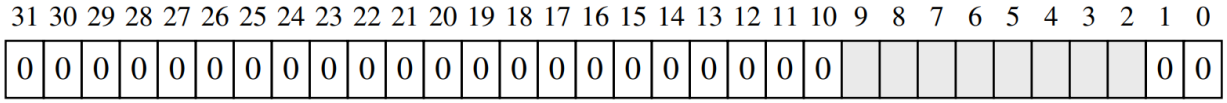
ARM immediate constants 立即数



ror #0—range 0 through 0x000000FF—step 0x00000001



ror #8—range 0 through 0xFF000000—step 0x01000000



ror #30—range 0 through 0x000003FC—step 0x00000004

- 数据处理指令中，立即数占了8位，同时还规定了一个循环移位的值。这样设计的目的是为了获得取值范围较大的数
- 通过循环移位，可以将立即数的范围从8位最多扩展到32位。

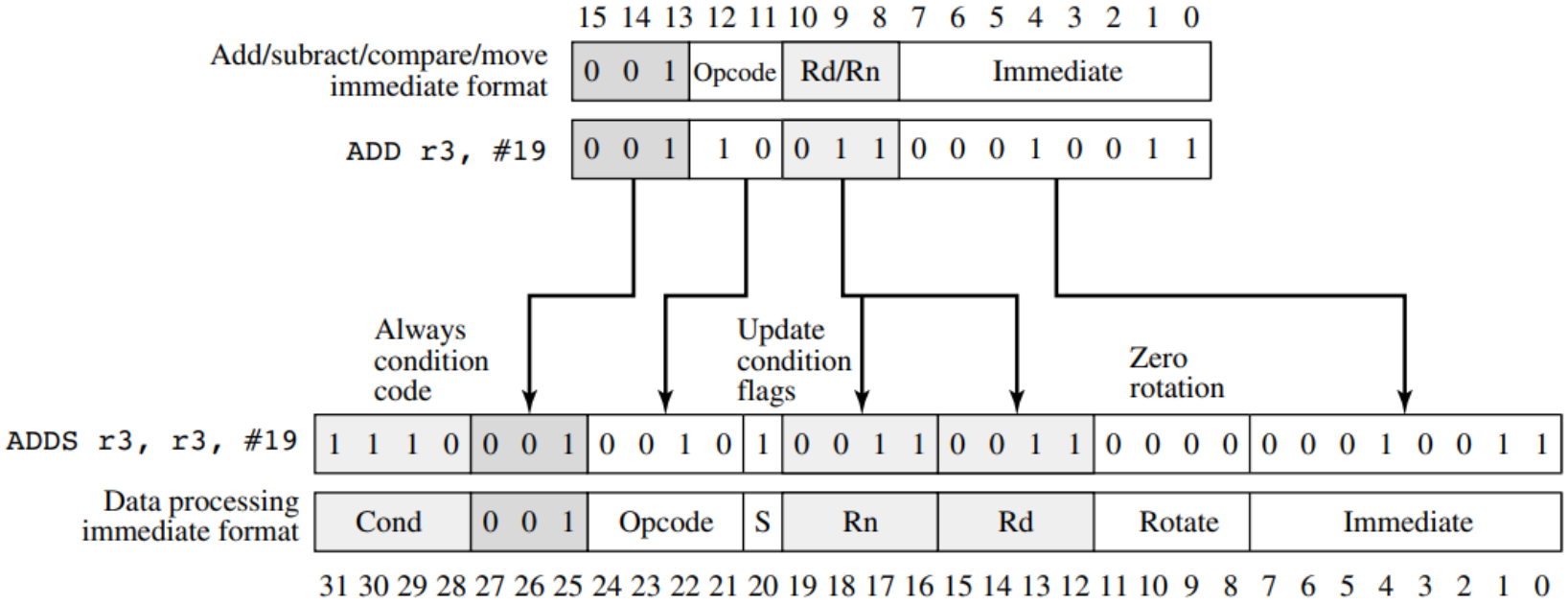
Thumb instruction set 压缩指令集

- Special Usage：use 16 bit instructions to implement most of 32-bit instructions 特殊用法：用16位长度的指令实现32位指令中的大部分指令
- In an embedded system, there may only be a 16 bit bus 在嵌入式系统中，可能只有16位总线
- Thumb instruction set：Re-encoded subset of ARM instruction set 压缩指令集：ARM指令集的重编码的子集
- Increases performance in 16-bit or less data bus 16位总线时能够提高性能

Thumb instruction set 压缩指令集

- Need to reduce 16 bits in the instruction 需要减少指令中的16位
- Unconditional (4 bits saved) 没有条件码，节省4位
- Always update conditional flags 总是更新条件标志
  - Update flag not used (1 bit saved) 节省1位
- Subset of instructions 指令的子集
  - 2 bit opcode, 3 bit type field (2 bit saved) 2位操作码，3位类型域
  - Reduced operand specifications (9 bits saved) 减少操作数，省了9位

Thumb instruction set 压缩指令集



- 压缩指令集的16位指令可以扩展到32位的标准指令
  - 压缩的指令集只有16位，可以在配置较低的硬件上执行
  - 如果在标准的ARM处理器上执行，可以按照这个图上的方法，扩充到32位之后进行执行
- ARM处理器能够执行16位和32位的指令，并且能够两种格式混合执行
- 处理器中的控制寄存器中的1位用来确定当前的执行是16位的指令还是32位的指令

Key Terms

Autoindexing	Effective address	Instruction format	Register indirect addressing
Base-register addressing	Immediate addressing	Postindexing	Relative address
Direct addressing	Indexing	Preindexing	word
Displacement addressing	Indirect addressing	Register addressing	

Summary and Question

- 小结
  - 这节课我们对指令的寻址模式和指令格式进行了讨论。
- 问题
  - 问题1：什么是寻址模式？为什么要采用多种寻址模式？
  - 问题2：影响指令格式的三个关键因素是哪些？

Summary and Question

- 问题
  - 问题1：什么是寻址模式？为什么要采用多种寻址模式？
    - 寻址模式：确定怎么去获得指令中的操作数
    - 可寻址的地址范围、寻址的灵活性、寻址的复杂度以及占用的存储单元数量之间进行平衡
  - 问题2：影响指令格式的三个关键因素是哪些？
    - 操作码，操作数，寻址模式

Assignments

- Review Questions
  - 11.1~11.7，11.11
- Problem
  - 11.2 11.3 11.7 11.13

谢谢大家！