COA 要点整理

Chapter 4 Cache Memory

Computer Memory System overview

Key Characteristics of Computer Memory Systems

location: refers to whether memory is internal or external to the computer.

The processor requires its own local memory, in the form of registers

“Cache is another form of internal memory.”

“External memory consists of peripheral storage devices, such as disk and tape”

capacity

For internal memory, this is typically expressed in terms of bytes (1 byte = 8 bits) or words. Common word lengths are 8, 16, and 32 bits. External memory capacity is typically expressed in terms of bytes.

unit of transfer: For internal memory, the unit of transfer is equal to the number of electrical lines into and out of the memory module. This may be equal to the word length, but is often larger. There are related concepts:

Word: The “natural” unit of organization of memory.The size of a word is typically equal to the number of bits used to represent an integer and to the instruction length. BUT there are many exceptions.

Addressable units: In some systems, the addressable unit is the word. Others allow addressing at the byte level. In any case, the relationship between the length in bits A of an address and the number N of addressable units is $2^A = N$.

Unit of transfer: For main memory, this is the number of bits read out of or written into memory at a time. Sometimes read and write have different unit of transfer (see Problem 4.23). The unit of transfer need not equal a word or an addressable unit. For external memory, data are often transferred in much larger units than a word, and these are referred to as blocks.

method of accessing

sequential access:

- Memory is organized into units of data, called records. Access must be made in a specific linear sequence.
- A shared read–write mechanism is used
- highly variable access time
- e.g. tape

direct access:

- shared read-write mechanism
- Access is accomplished by direct access to reach a general vicinity plus sequential searching, counting, or waiting to reach the final location.
- highly variable access time

- e.g. disk units

random access:

- Each addressable location in memory has a unique, physically wired-in addressing mechanism.
- constant access time
- e.g. main memory and some cache systems

associative:

- This is a random access type of memory that enables one to make a comparison of desired bit locations within a word for a specified match, and to do this for all words simultaneously.
- a word is retrieved based on a portion of its contents rather than its address.
- each location has its own addressing mechanism
- constant access time
- e.g. cache

performance

access time (latency)

For random-access memory, this is the time it takes to perform a read or write operation, that is, the time from the instant that an address is presented to the memory to the instant that data have been stored or made available for use. For non-random-access memory, access time is the time it takes to position the read–write mechanism at the desired location.

memory cycle time

consists of the access time plus any additional time required before a second access can commence. It's a concept about bus, not processor.

transfer rate

This is the rate at which data can be transferred into or out of a memory unit. For random-access memory, it is equal to 1/(cycle time). For non-random-access memory, the following relationship holds:

$$T_n = T_A + \frac{n}{R}$$

physical type

semiconductor memory, magnetic surface memory, used for disk and tape, and optical and magneto-optical.

physical characteristics

volatile: In a volatile memory, information decays naturally or is lost when electrical power is switched off. e.g. some semiconductor memory

nonvolatile: In a nonvolatile memory, information once recorded remains without deterioration until deliberately changed; no electrical power is needed to retain information. e.g. magnetic-surface memories, some semiconductor memory

nonerasable: Nonerasable memory cannot be altered, except by destroying the storage unit. Must be nonvolatile. e.g. ROM (semiconductor)

remark:

1. External, nonvolatile memory is also referred to as secondary memory or auxiliary memory.
2. A portion of main memory can be used as a buffer to hold data temporarily that is to be read out to disk. Such a technique, sometimes referred to as a disk cache

oraganization: refers to the physical arrangement of bits to form words.

The Memory Hierarchy

The relationships between capacity, access time and cost per bit:

- Faster access time, greater cost per bit
- Greater capacity, smaller cost per bit
- Greater capacity, slower access time.

The way out of this dilemma is not to rely on a single memory component or technology, but to employ a memory hierarchy.(考过翻译)

As one goes down the hierarchy, the following occur:

- Decreasing cost per bit
- Increasing capacity
- Increasing access time
- Decreasing frequency of access of the memory by the processor.

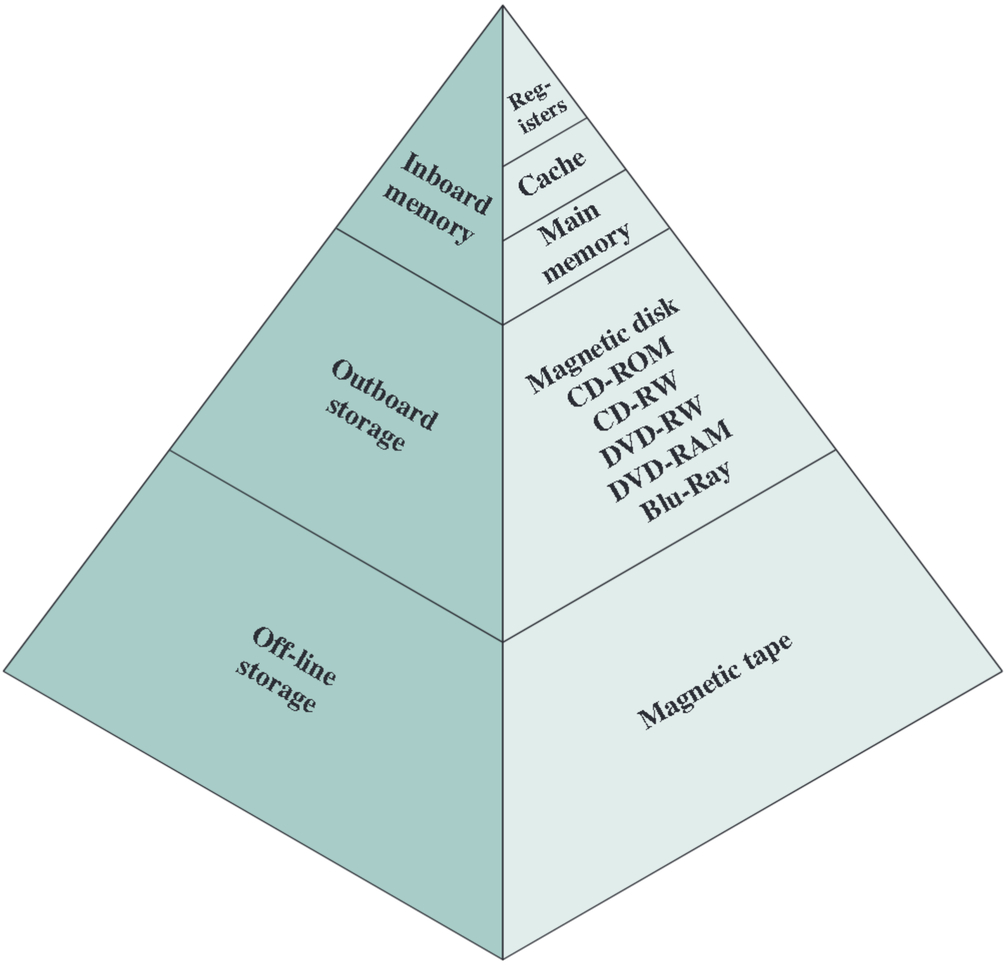


Figure 4.1 The Memory Hierarchy

The basis for the validity of condition (d) is a principle known as **locality of reference** locality of reference： During the course of execution of a program, memory references by the processor, for both instructions and data, tend to cluster.

remark:

another definition on p147: principle of locality, which states that data in the vicinity of a referenced word are likely to be referenced in the near future.

and on p158: This principle states that memory references tend to cluster. Over a long period of time, the clusters in use change, but over a short period of time, the processor is primarily working with fixed clusters of memory references.

Consider the following line of reasoning:

- Except for branch and call instructions, which constitute only a small fraction of all program instructions, program execution is sequential. Hence, in most cases, the next instruction to be fetched immediately follows the last instruction fetched.
- It is rare to have a long uninterrupted sequence of procedure calls followed by the corresponding sequence of returns. Rather, a program remains confined to a rather narrow window of procedure-invocation depth. Thus, over a short period of time references to instructions tend to be localized to a few procedures.
- Most iterative constructs consist of a relatively small number of instructions repeated many times. For the duration of the iteration, computation is therefore confined to a small contiguous portion of a program.
- In many programs, much of the computation involves processing data structures, such as arrays or sequences of records. In many cases, success

Spatial locality refers to the tendency of execution to involve a number of memory locations that are clustered.这在指令上来说，反映了处理器倾向于访问顺序储存的指令；从数据上来说，反映了程序在访问数据时一般也是顺序的。

Temporal locality refers to the tendency for a processor to access memory locations that have been used recently.

如何利用这两条性质：Traditionally, temporal locality is exploited by keeping recently used instruction and data values in cache memory and by exploiting a cache hierarchy. Spatial locality is generally exploited by using larger cache blocks and by incorporating prefetching mechanisms (fetching items of anticipated use) into the cache control logic.

remark: The cache is not usually visible to the programmer or, indeed, to the processor. 缓存对处理器也不可见

Cache Memory Principles

line size: The length of a line, not including tag and control bits.

tag: Because there are more blocks than lines, an individual line cannot be uniquely and permanently dedicated to a particular block. Thus, each line includes a tag that identifies which particular block is currently being stored.

read operation: (书上这么写的，但是比较简略，答题时可能需要结合具体情景和数字作答)

1. The processor generates the read address (RA) of a word to be read.
2. If the word is contained in the cache, it is delivered to the processor.

3. Otherwise, the block containing that word is loaded into the cache, and the word is delivered to the processor. 需要注意的是，另一种经典的架构是，先从mm读到cache，再从cache读到CPU

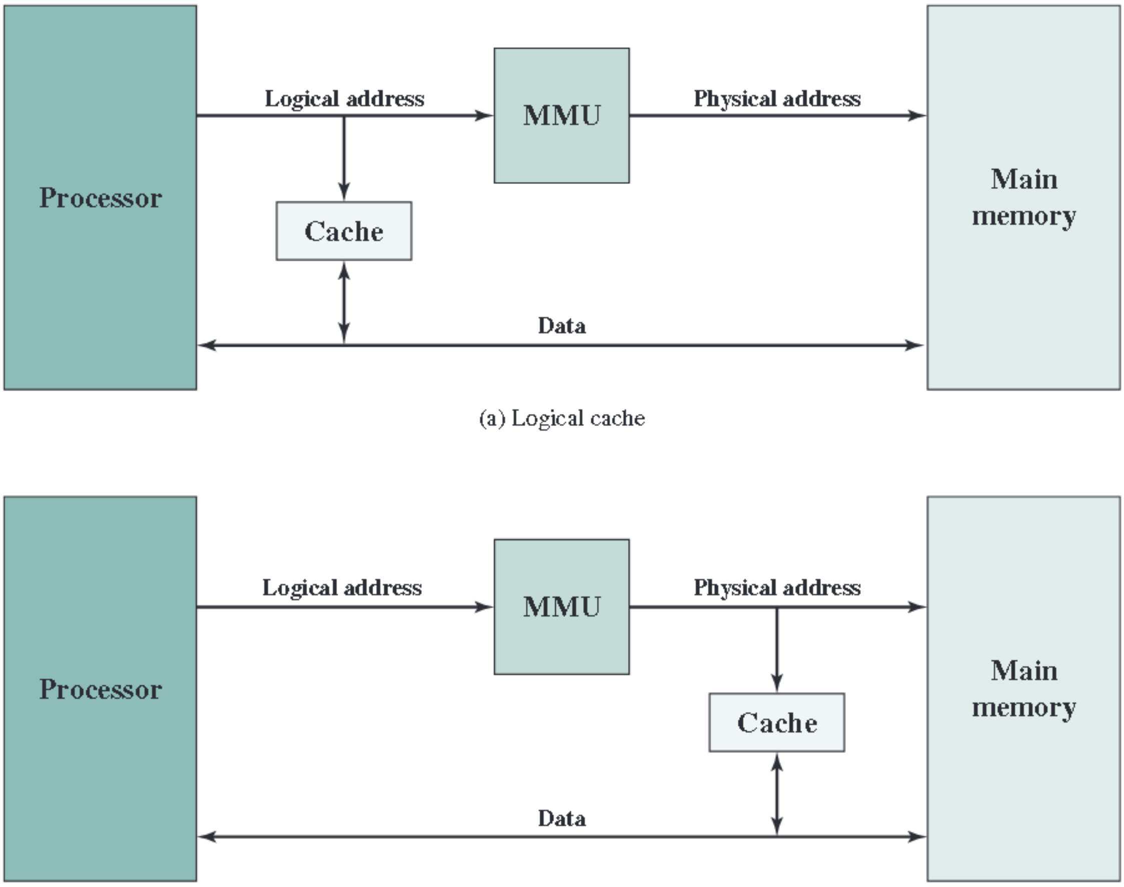
Elements of Cache Design

cache addresses

virtual memory: In essence, virtual memory is a facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available.

logical cache: A logical cache, also known as a virtual cache, stores data using virtual addresses. advantage: faster access. The processor accesses the cache directly, without going through the MMU. disadvantage: most virtual memory systems supply each application with the same virtual memory address space. That is, each application sees a virtual memory that starts at address 0. Thus, the same virtual address in two different applications refers to two different physical addresses. The cache memory must therefore be completely flushed with each application context switch, or extra bits must be added to each line of the cache to identify which virtual address space this address refers to. 就是不同应用使用的虚拟地址都一样（从0开始），那么cache中同一个虚拟地址可能实际对应两个物理地址。解决办法是，要么在切换应用的时候清空cache，要么就增加区分比特。

A physical cache stores data using main memory physical addresses.



(a) Logical cache

(b) Physical cache

Figure 4.7 Logical and Physical Caches

cache size

希望cache尽量小，这样单价就接近mm；又希望cache尽量大，这样访问速度就接近cache。cache越大，访问速度越慢。

mapping function

direct mapping

- simplest
- each block of main memory map into only one possible cache line
- cache line number = main memory block number modulo number of lines in the cache
- For purposes of cache access, each main memory address can be viewed as consisting of three fields: tag, line, word
- disadvatage: if a program happens to reference words repeatedly from two different blocks that map into the same line, then the blocks will be continually swapped in the cache, and the hit ratio will be low (thrashing)

associative mapping

- permit each main memory block to be loaded into any line of the cache
- tag + word field
- 注意，从associative mapping的地址中看不出cache中的line数
- disadvatage: the complex circuitry required to examine the tags of all cache lines in parallel.

set-associative mapping

- As with associative mapping, each word maps into multiple cache lines in a specific set.
- a k way set-associative cache can be physically implemented as v associative caches or k direct mapping caches (small value of k)
- Tag, Set, and Word.
- k = 1 reduce to direct mapping, v = 1 reduce to associative mapping

replacement algorithm

To achieve high speed, such an algorithm must be implemented in hardware.

least recently used (LRU): Replace that block in the set that has been in the cache longest with no reference to it. 对于2路组映射，只需要设置一个USE bit，对于全关联映射，可以维护一个表，maintains a separate list of indexes to all the lines in the cache. When a line is referenced, it moves to the front of the list. For replacement, the line at the back of the list is used.

first-in-first-out (FIFO): Replace that block in the set that has been in the cache longest. FIFO is easily implemented as a round-robin or circular buffer technique.

least frequently used (LFU): Replace that block in the set that has experienced the fewest references. LFU could be implemented by associating a counter with each line.

random replacement

write policy

两个问题:

1. 在多设备共享访问主存时，cache改了内存中的一个内容但没更新，可能导致其他访问同一块内存的设备得到无效数据；反过来，一个设备修改mm可能导致cache中的内容失效
2. 多处理器多cache情况下，一个cache中的内存更改可能导致其他cache中的内容失效

解决第一个问题

write through

- all write operations are made to main memory as well as to the cache, ensuring that main memory is always valid.

- disadvantage: it generates substantial memory traffic and may create a bottleneck.
- simplest

write back

- minimize memory writes
- With write back, updates are made only in the cache. When an update occurs, a dirty bit, or use bit, associated with the line is set. Then, when a block is replaced, it is written back to main memory if and only if the dirty bit is set.
- disadvantage: portions of main memory are invalid, and hence accesses by I/O modules can be allowed only through the cache. This makes for complex circuitry and a potential bottleneck. 因为如果IO直接访问mm则有可能这块mm已经在cache中被修改了只不过没写回，是无效的。

解决第二个问题

bus watching with write through: Each cache controller monitors the address lines to detect write operations to memory by other bus masters. If another master writes to a location in shared memory that also resides in the cache memory, the cache controller invalidates that cache entry. only for write through.

Hardware transparency: if one processor modifies a word in its cache, this update is written to main memory. In addition, any matching words in other caches are similarly updated.

Noncacheable memory: Only a portion of main memory is shared by more than one processor, and this is designated as noncacheable. In such a system, all accesses to shared memory are cache misses, because the shared memory is never copied into the cache.

line size

As the block size increases from very small to larger sizes, the hit ratio will at first increase because of the principle of locality. The hit ratio will begin to decrease, however, as the block becomes even bigger and the probability of using the newly fetched information becomes less than the probability of reusing the information that has to be replaced. Two specific effects come into play

- Larger blocks reduce the number of blocks that fit into a cache. Because each block fetch overwrites older cache contents, a small number of blocks results in data being overwritten shortly after they are fetched.
- As a block becomes larger, each additional word is farther from the requested word and therefore less likely to be needed in the near future.

number of caches

includes: multilevel cache and split cache

multilevel caches

background: on-chip cache is faster than external bus(这里的external是相对于processor而言的，其external指的是片外cache和mm等，而不是普通意义上的外设), and saves external bus for other transfer.

那么片外cache是不是没用了呢？不是。由此引入经典的两级cache，L1在片上，L2在片外

Two features of contemporary cache design for multilevel caches are noteworthy

- First, for an off-chip L2 cache, many designs do not use the system bus as the path for transfer between the L2 cache and the processor, but use a separate data path, so as to reduce the burden on the system bus.

- Second, with the continued shrinkage of processor components, a number of processors now incorporate the L2 cache on the processor chip, improving performance.

L2一般在L1两倍大的时候，对命中率的提升最显著

The need for the L2 cache to be larger than the L1 cache to affect performance makes sense. If the L2 cache has the same line size and capacity as the L1 cache, its contents will more or less mirror those of the L1 cache.

unified versus split caches

advantages of a unified cache:

- For a given cache size, a unified cache has a higher hit rate than split caches because it balances the load between instruction and data fetches automatically. That is, if an execution pattern involves many more instruction fetches than data fetches, then the cache will tend to fill up with instructions, and if an execution pattern involves relatively more data fetches, the opposite will occur.
- Only one cache needs to be designed and implemented.

More recently, it has become common to split the cache into two: one dedicated to instructions and one dedicated to data. These two caches both exist at the same level, typically as two L1 caches.

The key advantage of the split cache design is that it eliminates contention for the cache between the instruction fetch/decode unit and the execution unit. important for pipelining

The trend is toward split caches at the L1 and unified caches for higher levels, particularly for superscalar machines

appendix 4A

access time for 2-level cache

$$\begin{aligned} T_s &= H \times T_1 + (1 - H) \times (T_1 + T_2) \\ &= T_1 + (1 - H) \times T_2 \end{aligned} \qquad \textbf{(4.2)}$$

where

$$\begin{aligned} T_s &= \text{average (system) access time} \\ T_1 &= \text{access time of M1 (e.g., } \textit{cache, disk cache}) \\ T_2 &= \text{access time of M2 (e.g., } \textit{main memory, disk}) \\ H &= \text{hit ratio (fraction of time reference is found in M1)} \end{aligned}$$

cost

$$C_s = \frac{C_1 S_1 + C_2 S_2}{S_1 + S_2} \qquad \textbf{(4.3)}$$

where

$$\begin{aligned} C_s &= \text{average cost per bit for the combined two-level memory} \\ C_1 &= \text{average cost per bit of upper-level memory M1} \\ C_2 &= \text{average cost per bit of lower-level memory M2} \\ S_1 &= \text{size of M1} \\ S_2 &= \text{size of M2} \end{aligned}$$

To get at this, consider the quantity T1/Ts, which is referred to as the **access efficiency**. It is a measure of how close average access time (Ts) is to M1 access time (T1).

$$\frac{T_1}{T_s} = \frac{1}{1 + (1 - H)\frac{T_2}{T_1}}$$

So if there is strong locality, it is possible to achieve high values of hit ratio even with relatively small upper-level memory size (small Relative memory size (S1/S2)).

Chapter 12 Instruction Sets: Characteristics and Functions

Machine Instruction Characteristics

machine instructions or computer instructions: The operation of the processor is determined by the instructions it executes.

instruction set: The collection of different instructions that the processor can execute

elements of a machine instruction

- Operation code
- Source operand reference
- Result operand reference
- Next instruction reference

Source and result operands can be in one of four areas

- Main or virtual memory
- Processor register
- Immediate
- I/O device: If memory-mapped I/O is used, this is just another main or virtual memory address.

Instruction Representation

Opcodes are represented by abbreviations, called **mnemonics**, that indicate the operation.

Instruction types

A high-level language expresses operations in a concise algebraic form, using variables. A machine language expresses operations in a basic form involving the movement of data to or from registers.

The set of machine instructions must be sufficient to express any of the instructions from a high-level language. With this in mind we can categorize instruction types as follows:

- Data processing: Arithmetic and logic instructions.
- Data storage: Movement of data into or out of register and or memory locations.
- Data movement: I/O instructions.
- Control: Test and branch instructions.

Number of Addresses

The reasoning for the number of addresses:Virtually all arithmetic and logic operations are either unary (one source operand) or binary (two source operands). Thus, we would need a maximum of two addresses to reference source operands. The result of an operation must be stored, suggesting a third address, which defines a destination operand. Finally, after completion of an instruction, the next instruction must be fetched, and its address is needed.

note that zero-address instructions are applicable to a special memory organization called a stack.

tradeoff for number of addresses:“The number of addresses per instruction is a basic design decision. Fewer addresses per instruction result in instructions that are more primitive, requiring a less complex processor. It also results in instructions of shorter length. On the other hand, programs contain more total instructions, which in general results in longer execution times and longer, more complex programs.”

更重要的是一个门限值：Also, there is an important threshold between one-address and multiple-address instructions. 多地址指令才能允许只依赖于寄存器的操作，这更快。

Instruction Set Design

The most important of these fundamental design issues include the following:

- Operation repertoire: How many and which operations to provide, and how complex operations should be.
- Data types: The various types of data upon which operations are performed.
- Instruction format: Instruction length (in bits), number of addresses, size of various fields, and so on.
- Registers: Number of processor registers that can be referenced by instructions, and their use.
- Addressing: The mode or modes by which the address of an operand is specified.

Types of Operands

The most important general categories of data are

- Addressses, in this context, they can be considered to be unsigned integers
- Numbers
- Characters
- Logical data

Numbers

“An important distinction between numbers used in ordinary mathematics and numbers stored in a computer is that the latter are limited.” 精度和范围都是有限的。因此，舍入、溢出和下溢出

常见的三种数值数据

■ Binary integer or binary fixed point

■ Binary floating point

■ Decimal

Packed decimal:压缩BCD码。用四位二进制数表示一个十进制数，而且一般8个8个的使用。比如246=0000 0010 0100 0110

Characters

IRA:International Reference Alphabet,referred to in the United States as the American Standard Code for Information Interchange (ASCII)

用7位二进制数表示一个字符，但是一般都用8位字符，多出来的一位要么是0，要么用于奇偶校验。多出来的部分用来表示控制字符，比如控制打印，通信流程等。

还有一种编码字符集Extended Binary Coded Decimal Interchange Code (EBCDIC)，8位，和IRA一样，与packed decimal兼容

Logical data

当一个n位单元被视为n个1比特数据（要么是1要么是0）时，这个数据就叫做逻辑数据

优点：

- 存布尔值或二进制数据时，更高效
- 有时候我们希望按位操作数据，这时logical数据更方便

Types of Operations

■ Data transfer :most common

■ Arithmetic

■ Logical

■ Conversion

■ I/O

■ System control

■ Transfer of control

Data transfer

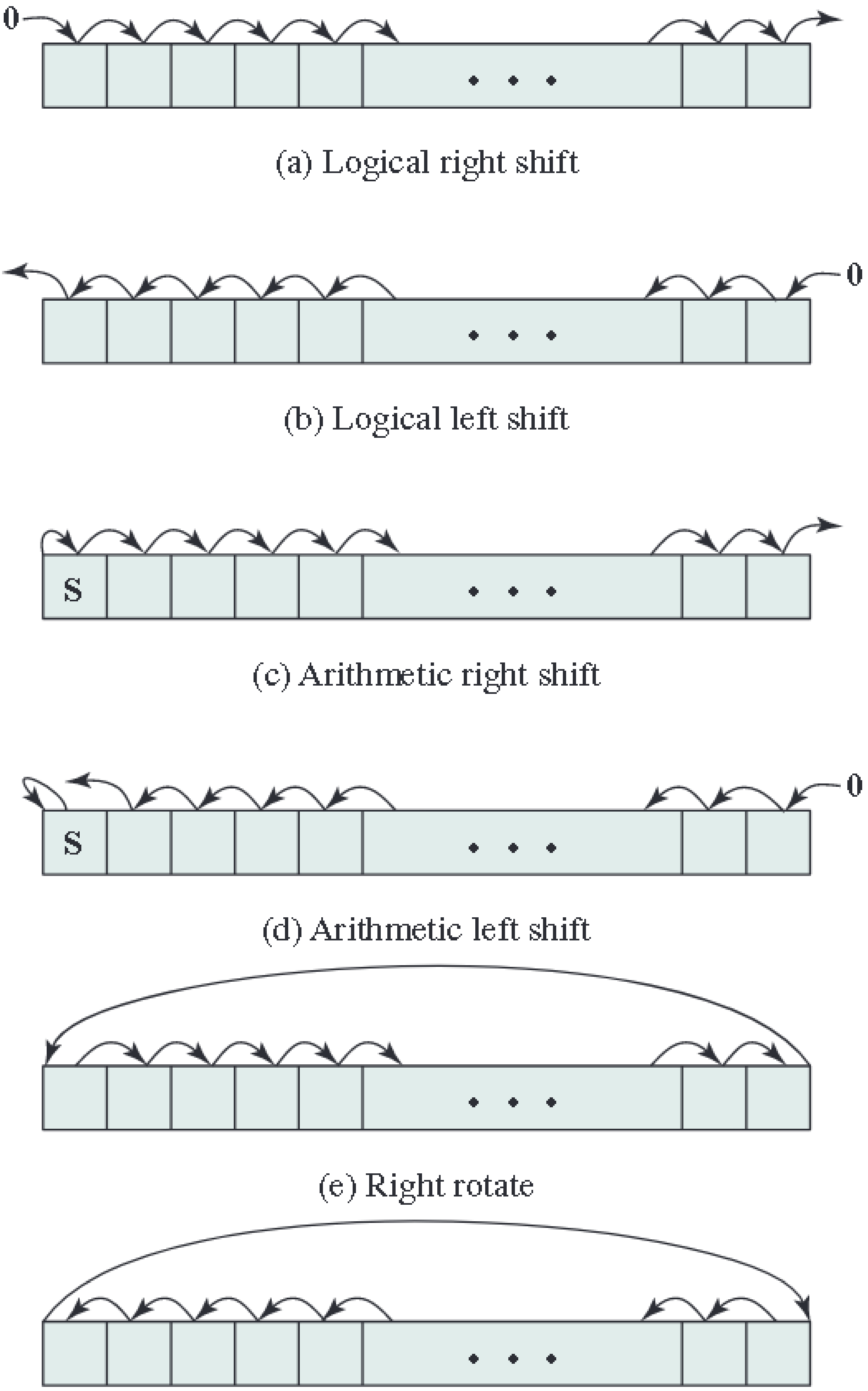
需要明确：

1. 源和目的操作数地址：内存、寄存器或栈
2. 数据长度
3. 寻址方式

“If one or both operands are in memory, then the processor must perform some or all of the following actions: 1. Calculate the memory address, based on the address mode (discussed in Chapter 13). 2. If the address refers to virtual memory, translate from virtual to real memory address. 3. Determine whether the addressed item is in cache. 4. If not, issue a command to the memory module.”

Arithmetic

Most machines also provide a variety of operations for manipulating individual bits of a word or other addressable units, often referred to as “**bit twiddling**.” They are based upon Boolean operations



(a) Logical right shift

(b) Logical left shift

(c) Arithmetic right shift

(d) Arithmetic left shift

(e) Right rotate

(f) Left rotate

Figure 12.6 Shift and Rotate Operations

Transfer of Controls

为什么控制转移是必须的？因为下述情况/需求存在：

- 循环
- 决策/条件转移
- 模块化编程

“most common transfer-of-control operations found in instruction sets: branch, skip, and procedure call.”

“The two principal reasons for the use of procedures are economy and modularity.” 使用程序的两个主要原因是经济性和模块性。经济性使得编程更轻松、存储空间利用率提高：不然重复写相同的程序很占空间。模块性使得编程更轻松

“Three points are worth noting:

1. A procedure can be called from more than one location.
2. A procedure call can appear in a procedure. This allows the nesting of procedures to an arbitrary depth.
3. Each procedure call is matched by a return in the called program.”

“There are three common places for storing the return address:

■ Register

$$RN \leftarrow PC + \Delta$$
$$PC \leftarrow X$$

■ Start of called procedure

$$X \leftarrow PC + \Delta$$
$$PC \leftarrow X + 1$$

■ Top of stack:最常见的方式。而且是唯——种便于重入程序的设计

“A **reentrant procedure** is one in which it is possible to have several calls open to it at the same time. A recursive procedure (one that calls itself) is an example of the use of this feature”

调用过程时如何传参：使用寄存器或者存在内存中紧跟call指令的位置。仍然，还是使用栈最灵活

Chapter 13 Instruction Sets (2)

13.1 Addressing Modes and Formats

The most common addressing techniques, or modes: ■ Immediate

■ Direct

■ Indirect

■ Register

■ Register indirect

■ Displacement

■ Stack

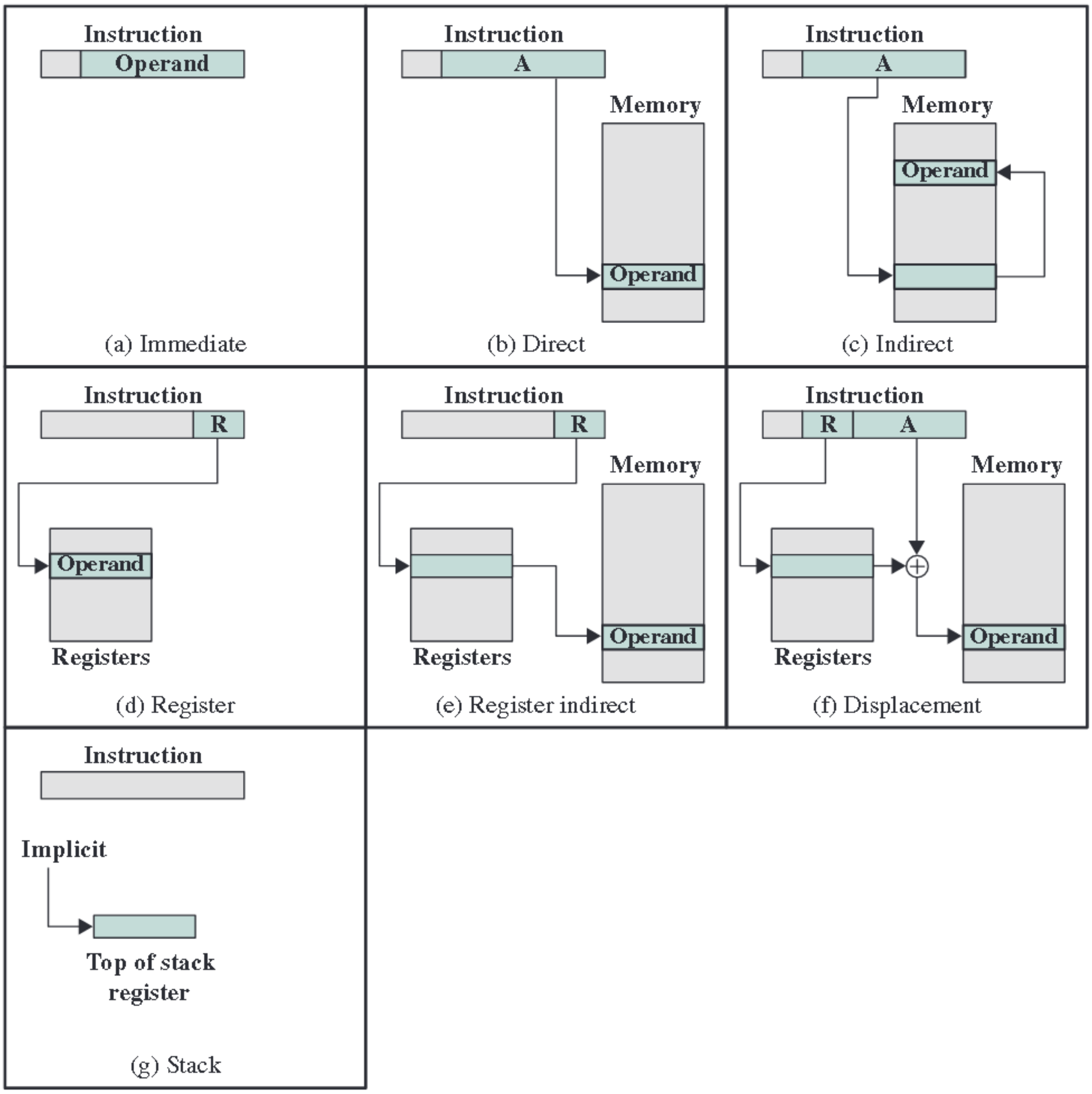


Figure 13.1 Addressing Modes

Table 13.1 Basic Addressing Modes

Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	Operand = A	No memory reference	Limited operand magnitude
Direct	EA = A	Simple	Limited address space
Indirect	EA = (A)	Large address space	Multiple memory references
Register	EA = R	No memory reference	Limited address space
Register indirect	EA = (R)	Large address space	Extra memory reference
Displacement	EA = A + (R)	Flexibility	Complexity
Stack	EA = top of stack	No memory reference	Limited applicability

很多计算机都提供多种寻址方式，那么，处理器如何判断一条特定的指令中用到的是什么指令呢？有几种方法：

- 通过操作码判断：different opcodes will use different addressing modes
- 通过添加几个比特来区分：mode field

接下来分不同的寻址模式来介绍

Immediate Addressing

operand value is present in the instruction

$$Operand = A$$

- PROS:no memory reference other than the instruction fetch
- CONS:the size of the number is restricted to the size of the address field

Direct Addressing

the address field contains the effective address of the operand:

$$EA = A$$

- requires only one memory reference and no special calculation
- provides only a limited address space.

Indirect Addressing

have the address field refer to the address of a word in memory, which in turn contains a full-length address of the operand.

$$EA = (A)$$

- PROS:for a word length of N, an address space of 2N is now available
- an indirect memory reference will involve, at most, one page fault rather than two.
- CONS:instruction execution requires two memory references to fetch the operand

multilevel or cascaded indirect addressing:

$$EA = (\dots(A)\dots)$$

need an indirect flag. If the I bit is 0, then the word contains the EA. If the I bit is 1, then another level of indirection is invoked

Register Addressing

Register addressing is similar to direct addressing. The only difference is that the address field refers to a register rather than a main memory address

$$EA = R$$

- only a small address field is needed in the instruction (because registers are less tahn mm units)
- no time-consuming memory references are required
- CONS:the address space is very limited.

既然寄存器寻址要快得多，那么经常被用到的数就应该多放在reg中，给编程者提出挑战

Register Indirect Addressing

register indirect address

$$EA = (R)$$

The advantages and limitations of register indirect addressing are basically the same as for indirect addressing.

In addition, register indirect addressing uses one less memory reference than indirect addressing.

Displacement Addressing

A very powerful mode of addressing combines the capabilities of direct addressing and register indirect addressing. It is known by a variety of names depending on the context of its use, but the basic mechanism is the same. We will refer to this as displacement addressing

$$EA = A + (R)$$

requires that the instruction have two address fields, **at least one of which is explicit**. The value contained in one address field (value = A) is used **directly**. The other address field, or an implicit reference based on opcode, refers to a register whose contents are added to A to produce the effective address.

进一步可以分为三类：Relative addressing，Base-register addressing，Indexing。

Relative addressing： also called PC-relative addressing, the implicitly referenced register is the program counter (PC).That is, the next instruction address is added to the address field to produce the EA. Typically, the address field is treated as a twos complement number for this operation（以补码形式运算）

Relative addressing exploits the concept of locality. relative addressing **saves address bits**

base-register addressing: The referenced register contains a main memory address, and the address field contains a displacement (usually an unsigned integer representation) from that address. The register reference may be explicit or implicit

Base-register addressing also exploits the locality of memory references. It is a convenient means of implementing segmentation 基址寻址作用不在于扩充寻址空间而在于方便地分段。基址寄存器可以默认也可以程序员指定

“if the length of the address field is K and the number of possible registers is N, then one instruction can reference any one of N areas of 2K words.”

Indexing： The address field references a main memory address, and the referenced register contains a positive displacement from that address (register reference is sometimes explicit and sometimes implicit). Note that this usage is just the opposite of the interpretation for base-register addressing.

与基址寻址的区别在于：“Because the address field is considered to be a memory address in indexing, it generally contains more bits than an address field in a comparable base-register instruction.”

An important use of indexing is to provide an efficient mechanism for **performing iterative operations.** 这种方式非常常见，所以有自动变址机制

This is known as autoindexing. If certain registers are devoted exclusively to indexing, then autoindexing can be invoked implicitly and automatically.

$$EA = A + (R)$$
$$(R) \leftarrow (R) + 1$$

间址寻址和变址寻址可以结合使用。(Typically, an instruction set will **not** include both preindexing and postindexing.)

If indexing is performed after the indirection, it is termed postindexing:

$$EA = (A) + (R)$$

This technique is useful for accessing one of a number of blocks of data of a fixed format. For example, a process control block

With preindexing, the indexing is performed before the indirection:

$$EA = (A + (R))$$

An example of the use of this technique is to construct a multiway branch table.

Stack Addressing

The stack pointer is maintained in a register. Thus, references to stack locations in memory are in fact **register indirect addresses.** The stack mode of addressing is a form of implied addressing. The machine instructions need not include a memory reference but implicitly operate on the top of the stack.

13.3 Instruction Formats

Instruction Length

“This decision affects, and is affected by, memory size, memory organization, bus structure, processor complexity, and processor speed.”

The most obvious trade-off here is between the desire for a powerful instruction repertoire and a need to save space. Programmers want more opcodes, more operands, more addressing modes, and greater address range

other consideration: “Either the instruction length should be equal to the memory-transfer length (in a bus system, databus length) or one should be a multiple of the other.”

A related consideration is the memory transfer rate. This rate has not kept up with increases in processor speed. Accordingly, memory can become a bottleneck if the processor can execute instructions faster than it can fetch them. One solution to this problem is to use cache memory, another is to use shorter instructions. 因为指令越短，对硬件要求越低，传输越快

the instruction length should be a multiple of the character length, which is usually 8 bits, and of the length of fixed-point numbers. The word length of memory is, in some sense, the “natural” unit of organization. The size of a word usually determines the size of fixed-point numbers (usually the two are equal). Word size is also typically equal to, or at least integrally related to, the memory transfer size.

Allocation of Bits

For a given instruction length, there is clearly a trade-off between the number of opcodes and the power of the addressing capability. 操作码长度和地址长度的矛盾、

The following interrelated factors go into determining the use of the addressing bits.

- Number of addressing modes
- Number of operands
- Register versus memory：一般寄存器数目比较少，寻址所需比特少
- Number of register sets：如果reg还分了组，某些reg特定用于某些功能，那又可以省下一些比特
- Address range

- Address granularity 寻址粒度

Chapter 14 Processor Structure and Function

14.1 Processor Organization

To understand the organization of the processor, let us consider the requirements placed on the processor, the things that it must do:

■ Fetch instruction: The processor reads an instruction from memory (register, cache, main memory).

■ Interpret instruction: The instruction is decoded to determine what action is required.

■ Fetch data: The execution of an instruction may require reading data from memory or an I/O module.

■ Process data: The execution of an instruction may require performing some arithmetic or logical operation on data.

■ Write data: The results of an execution may require writing data to memory or an I/O module.

14.2 Register Organization

The registers in the processor perform two roles:

■ User-visible registers: Enable the machine- or assembly language programmer to minimize main memory references by optimizing use of registers.

■ Control and status registers: Used by the control unit to control the operation of the processor and by privileged, operating system programs to control the execution of programs.

User-Visible Registers

A user-visible register is one that may be referenced by means of the machine language that the processor executes.We can characterize these in the following categories:

■ General purpose： Sometimes their use within the instruction set is orthogonal to the operation. That is, any general-purpose register can contain the operand for any opcode.

■ Data： only to hold data and cannot be employed in the calculation of an operand address.

■ Address： may be devoted to a particular addressing mode. Segment pointers， index registers, stack pointers(This allows implicit addressing; that is, push, pop, and other stack instructions need not contain an explicit stack operand.).

关于是否应该用通用寄存器还是分成数据和地址寄存器，存在争论。专用寄存器可以省比特，但是灵活性下降

另一个争论是寄存器数目应该设置多少（8-32）。寄存器多了耗费比特，少了意味着更多的内存访问。RISC中用了很多寄存器

最后一个问题是寄存器位宽。Registers that must hold addresses obviously must be at least long enough to hold the largest address. Data registers should be able to hold values of most data types. Some machines allow two contiguous registers to be used as one for holding double-length values.

■ Condition codes (flags): “Generally, machine instructions allow these bits to be read by implicit reference, but the programmer cannot alter them.”

Control and Status Registers

Four registers are essential to instruction execution:

■ Program counter (PC): Contains the address of an instruction to be fetched.

■ Instruction register (IR): Contains the instruction most recently fetched.

■ Memory address register (MAR): Contains the address of a location in memory.

■ Memory buffer register (MBR): Contains a word of data to be written to memory or the word most recently read.

The four registers just mentioned are used for the movement of data between the processor and memory. Within the processor, data must be presented to the ALU for processing. The ALU may have direct access to the MBR and user-visible registers.

Many processor designs include a register or set of registers, often known as the program status word (PSW), that contain status information. Common fields or flags include the following:

- Sign: Contains the sign bit of the result of the last arithmetic operation.
- Zero: Set when the result is 0.
- Carry: Set if an operation resulted in a carry (addition) into or borrow (subtraction) out of a high-order bit. Used for multiword arithmetic operations.
- Equal: Set if a logical compare result is equality.
- Overflow: Used to indicate arithmetic overflow.
- Interrupt Enable/Disable: Used to enable or disable interrupts.
- Supervisor: Indicates whether the processor is executing in supervisor or user mode. Certain privileged instructions can be executed only in supervisor mode, and certain areas of memory can be accessed only in supervisor mode.

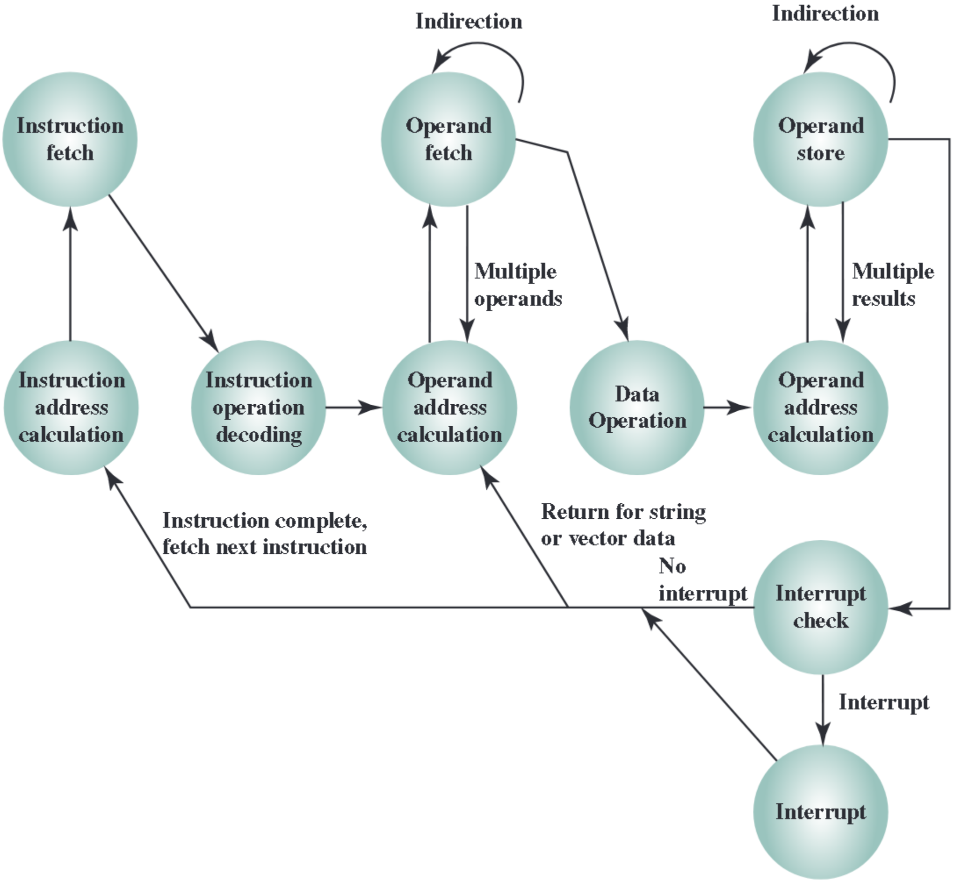
There may be a pointer to a block of memory containing additional status information (e.g., process control blocks)

“One key issue is operating system support.”“Another key design decision is the allocation of control information between registers and memory.”

14.3 The Instruction Cycle

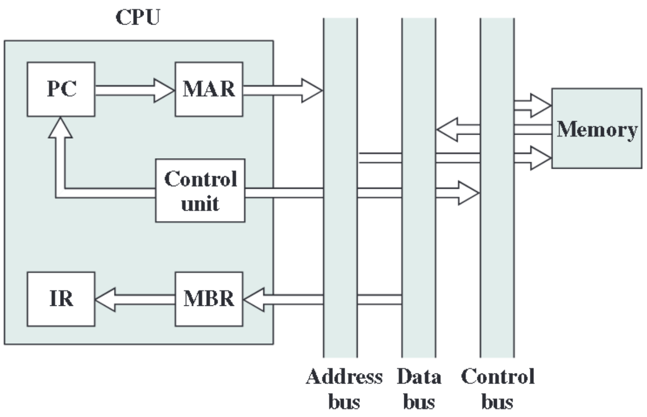
一般分为：Fetch,Execute and interupt. 其实还有间址周期

Data Flow


Indirection
Indirection
Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Instruction complete, fetch next instruction
Return for string or vector data
No interrupt
Interrupt check
Interrupt
Interrupt

fetch

During the fetch cycle, an instruction is read from memory. Figure 14.6 shows the flow of data during this cycle. The PC contains the address of the next instruction to be fetched. This address is moved to the MAR and placed on the address bus. The control unit requests a memory read, and the result is placed on the data bus and copied into the MBR and then moved to the IR. Meanwhile, the PC is incremented by 1, preparatory for the next fetch.



indirect

The rightmost N bits of the MBR, which contain the address reference, are transferred to the MAR. Then the control unit requests a memory read, to get the desired address of the operand into the MBR.

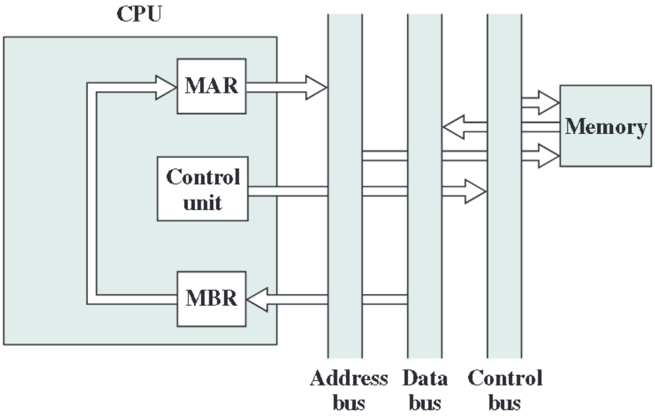


Figure 14.7 Data Flow, Indirect Cycle

interrupt

The current contents of the PC must be saved so that the processor can resume normal activity after the interrupt. Thus, the contents of the PC are transferred to the MBR to be written into memory. The special memory location reserved for this purpose is loaded into the MAR from the control unit. It might, for example, be a stack pointer. The PC is loaded with the address of the interrupt routine. As a result, the next instruction cycle will begin by fetching the appropriate instruction.

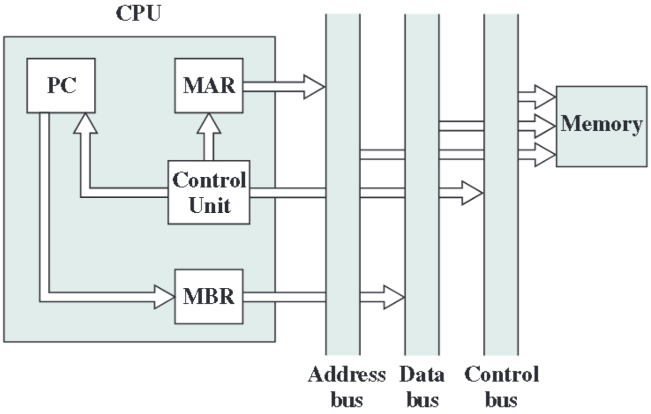


Figure 14.8 Data Flow, Interrupt Cycle

execute cycle is unpredictable.

14.4 Instruction Pipelining

将指令分为取指和执指作流水线，处理速度翻倍不可能达到的原因是：耗时不同+条件跳转

- The execution time will generally be longer than the fetch time. Execution will involve reading and storing operands and the performance of some operation. Thus, the fetch stage may have to wait for some time before it can empty its buffer.
- A conditional branch instruction makes the address of the next instruction to be fetched unknown. Thus, the fetch stage must wait until it receives the next instruction address from the execute stage. The execute stage may then have to wait while the next instruction is fetched.

指令可以更细分为6步

- Fetch instruction (FI): Read the next expected instruction into a buffer.
- Decode instruction (DI): Determine the opcode and the operand specifiers.
- Calculate operands (CO): Calculate the effective address of each source operand. This may involve displacement, register indirect, indirect, or other forms of address calculation.

- Fetch operands (FO): Fetch each operand from memory. Operands in registers need not be fetched.
- Execute instruction (EI): Perform the indicated operation and store the result, if any, in the specified destination operand location.
- Write operand (WO): Store the result in memory.

同样地，也会有一些问题阻碍：可能耗时不同造成等待，可能有条件分支，可能有中断，可能有数据依赖性

不能无限细分阶段来提速的原因：分阶段是有开销的（比如buffering）；用来处理内存/寄存器依赖性的控制逻辑可能会变得非常复杂；锁存也要用时

Pipeline Performance

The cycle time τ of an instruction pipeline is the time needed to advance a set of instructions one stage through the pipeline

$$\tau = \max_i[\tau_i] + d = \tau_m + d, 1 \leq i \leq k$$

$$\begin{aligned}
\tau_i &= \text{time delay of the circuitry in the } i\text{th stage of the pipeline} \\
\tau_m &= \text{maximum stage delay (delay through stage which experiences the largest} \\
&\quad\ \text{delay)} \\
k &= \text{number of stages in the instruction pipeline} \\
d &= \text{time delay of a latch, needed to advance signals and data from one stage} \\
&\quad\ \text{to the next}
\end{aligned}$$

the total time required for a pipeline with k stages to execute n instructions

$$T = [k + (n-1)]\tau$$

The speedup factor for the instruction pipeline compared to execution without the pipeline is defined as

$$S_k = \frac{T_{1,n}}{T_{k,n}} = \frac{nk}{k + (n-1)}$$

Pipeline Hazards

A pipeline hazard occurs when the pipeline, or some portion of the pipeline, must stall because conditions do not permit continued execution. Such a pipeline stall is also referred to as a pipeline bubble. There are three types of hazards: resource, data, and control.

Resource hazards

“resource hazards A resource hazard occurs when two (or more) instructions that are already in the pipeline need the same resource, like ALU or an I/O port.” A resource hazard is sometime referred to as a structural hazard. One solutions to such resource hazards is to increase available resources, such as having multiple ports into main memory and multiple ALU units.

Data hazards

“data hazards A data hazard occurs when there is a conflict in the access of an operand location.” Two instructions in a program are to be executed in sequence and both access a particular memory or register operand. If the two instructions are executed in strict sequence, no problem occurs. However, if the instructions are executed in a pipeline, then it is possible for the operand value to be updated in such a way as to produce a different result than would occur with strict sequential execution.

There are three types of data hazards:

- Read after write (RAW), or true dependency: A hazard occurs if the read takes place before the write operation is complete.
- Write after read (WAR), or antidependency: A hazard occurs if the write operation completes before the read operation takes place.
- Write after write (WAW), or output dependency: A hazard occurs if the write operations take place in the reverse order of the intended sequence.

Control hazards

“A control hazard, also known as a **branch hazard**, occurs when the pipeline makes the wrong decision on a branch prediction and therefore brings instructions into the pipeline that must subsequently be discarded.”

专门开一个小节来讲

Dealing with Branches

A variety of approaches have been taken for dealing with conditional branches:

Multiple streams: replicate the initial portions of the pipeline and allow the pipeline to fetch both instructions, making use of two streams. There are two problems with this approach:1. With multiple pipelines there are contention delays for access to the registers and to memory.2. Additional branch instructions may enter the pipeline (either stream) before the original branch decision is resolved. Each such instruction needs an additional stream.

Prefetch branch target: When a conditional branch is recognized, the target of the branch is prefetched, in addition to the instruction following the branch. This target is then saved until the branch instruction is executed. If the branch is taken, the target has already been prefetched.

Loop buffer: loop buffer is a small, very-high-speed memory maintained by the instruction fetch stage of the pipeline and containing the n most recently fetched instructions, in sequence. If a branch is to be taken, the hardware first checks whether the branch target is within the buffer.

1. With the use of prefetching, the loop buffer will contain some instruction sequentially ahead of the current instruction fetch address. Thus, instructions fetched in sequence will be available without the usual memory access time.
2. If a branch occurs to a target just a few locations ahead of the address of the branch instruction, the target will already be in the buffer. This is useful for the rather common occurrence of IF–THEN and IF–THEN–ELSE sequences.
3. This strategy is particularly well suited to dealing with loops, or iterations; hence the name loop buffer. If the loop buffer is large enough to contain all the instructions in a loop, then those instructions need to be fetched from memory only once, for the first iteration. For subsequent iterations, all the needed instructions are already in the buffer.

Branch prediction

static:■ Predict never taken ■ Predict always taken ■ Predict by opcode

dynamic:

■ Taken/not taken switch: one or more bits can be associated with each conditional branch instruction that reflect the recent history of the instruction. they are kept in temporary highspeed storage (like a cache), rather than mm.

the state machine of two-bit case

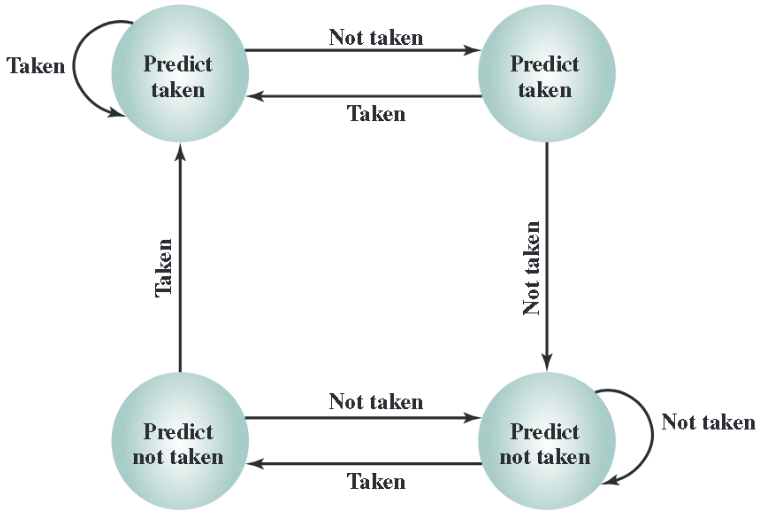


Figure 14.19 Branch Prediction State Diagram

Drawback: If the decision is made to take the branch, the target instruction cannot be fetched until the target address, which is an operand in the conditional branch instruction, is decoded. 这催生出了分支历史表的诞生，不仅可以用来预测一个分支是否要跳转，还可以暂存分支的目标

■ Branch history table: a small cache memory associated with the instruction fetch stage of the pipeline. Each entry in the table consists of three elements: the address of a branch instruction, some number of history bits that record the state of use of that instruction, and information about the target instruction (address). Each prefetch triggers a lookup in the branch history table. If no match is found, the next sequential address is used for the fetch. If a match is found, a prediction is made based on the state of the instruction: Either the next sequential address or the branch target address is fed to the select logic.

Delayed branch: It is possible to improve pipeline performance by automatically rearranging instructions within a program, so that branch instructions occur later than actually desired.

Chapter 15 Reduced Instruction Set Computers

15.1 Instruction Exection Characteristics

some of the major advances since the birth of the computer:

- The family concept: The family concept decouples the architecture of a machine from its implementation. A set of computers is offered, with different price/performance characteristics, that presents the same architecture to the user. The differences in price and performance are due to different implementations of the same architecture.
- Microprogrammed control unit: Microprogramming eases the task of designing and implementing the control unit and provides support for the family concept.
- Cache memory
- Pipelining: A means of introducing parallelism into the essentially sequential nature of a machine-instruction program.
- Multiple processors
- Reduced instruction set computer (RISC) architecture

Key elements shared by most designs:

- A large number of general-purpose registers, and/or the use of compiler technology to optimize register usage.
- A limited and simple instruction set.
- An emphasis on optimizing the instruction pipeline.

High-level languages (HLLs)

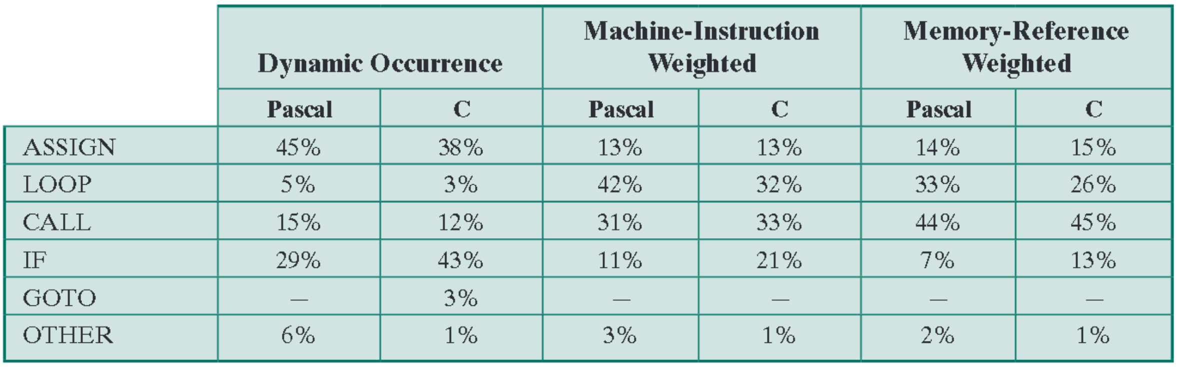
- allow the programmer to express algorithms more concisely
- allow the compiler to take care of details that are not important in the programmer's expression of algorithms
- often support naturally the use of structured programming and/or object-oriented design.

产生了semantic gap: the difference between the operations provided in HLLs and those provided in computer architecture

由此产生了越来越复杂的指令集，这导致一些人开始逆向思考，想构造一些简单的支持HLL的架构，即RISC。要明白RISC的合理性，要从三个方面来分析：Operation performed, operand used, execution sequencing. 检测的方法有动态和静态两种：Dynamic: measurements are collected by executing the program and counting the number of times some feature has appeared or a particular property has held true. static measurements merely perform these counts on the source text of a program.

Operation

Table 15.2 Weighted Relative Dynamic Frequency of HLL Operations [PATT82a]



	Dynamic Occurrence		Machine-Instruction Weighted		Memory-Reference Weighted	
	Pascal	C	Pascal	C	Pascal	C
ASSIGN	45%	38%	13%	13%	14%	15%
LOOP	5%	3%	42%	32%	33%	26%
CALL	15%	12%	31%	33%	44%	45%
IF	29%	43%	11%	21%	7%	13%
GOTO	—	3%	—	—	—	—
OTHER	6%	1%	3%	1%	2%	1%

Assign 最多，Call 最耗时

Operand

There is a preponderance of references to scalars, and these are highly localized. So "prime candidate for optimization is the mechanism for storing and accessing local scalar variables."

Procedure calls

"Two aspects are significant: the number of parameters and variables that a procedure deals with, and the depth of nesting."

Tanenbaum's study [TANE78] found that 98% of dynamically called procedures were passed **fewer than six arguments** and that 92% of them used **fewer than six local scalar** variables.

"They found that it is rare to have a long uninterrupted sequence of procedure calls followed by the corresponding sequence of returns. Rather, they found that a program remains confined to a rather narrow window of procedure-invocation depth."

Implication

The attempt to make the instruction set architecture close to HLLs is not the most effective design strategy. Rather, the HLLs can best be supported by optimizing performance of the most time-consuming features of typical HLL programs. 不应该开发与HLL相近的指令集(这很复杂且没什么用)，而应该努力优化那些在HLL程序中最耗时的特征的性能。

总的来说，针对上述分析，RISC架构的特征是

- use a large number of registers or use a compiler to optimize register usage. 因为MOVE指令很多，很多又是标量引用，而且还有局部性原理，所以用reg可以比用mm快，因此需要大量reg或者优化reg的使用。
- careful attention needs to be paid to the design of instruction pipelines. Because of the high proportion of conditional branch and procedure call instructions, a straightforward instruction pipeline will be inefficient.
- an instruction set consisting of high-performance primitives(原语) is indicated.

15.2 The Use of Large Register File

The register file is physically small, on the same chip as the ALU and control unit, and employs much shorter addresses than addresses for cache and memory.

Two basic approaches are possible, one based on software and the other on hardware. The **software approach** is to rely on the compiler to maximize register usage. The compiler will attempt to assign registers to those variables that will be used the most in a given time period. The **hardware approach** is simply to use more registers so that more variables can be held in registers for longer periods of time.

Register Windows

前面的讨论说，可利用数据的局部性。但是局部性是针对每一个子程序而言的，每一次call和return都会改变“局部”的定义。解决办法是，利用这两个事实：First, a typical procedure employs only a few passed parameters and local variables (Table 15.4). Second, the depth of procedure activation fluctuates within a relatively narrow range

具体表现为： multiple small sets of registers are used, each assigned to a different procedure. A procedure call automatically switches the processor to use a different fixed-size window of registers, rather than saving registers in memory. Windows for adjacent procedures are overlapped to allow parameter passing.

Parameter registers hold parameters passed down from the procedure that called the current procedure and hold results to be passed back up. Local registers are used for local variables, as assigned by the compiler. Temporary registers are used to exchange parameters and results with the next lower level. The temporary registers at one level are physically the same as the parameter registers at the next lower level.

The actual organization of the register file is as a circular buffer of overlapping windows.

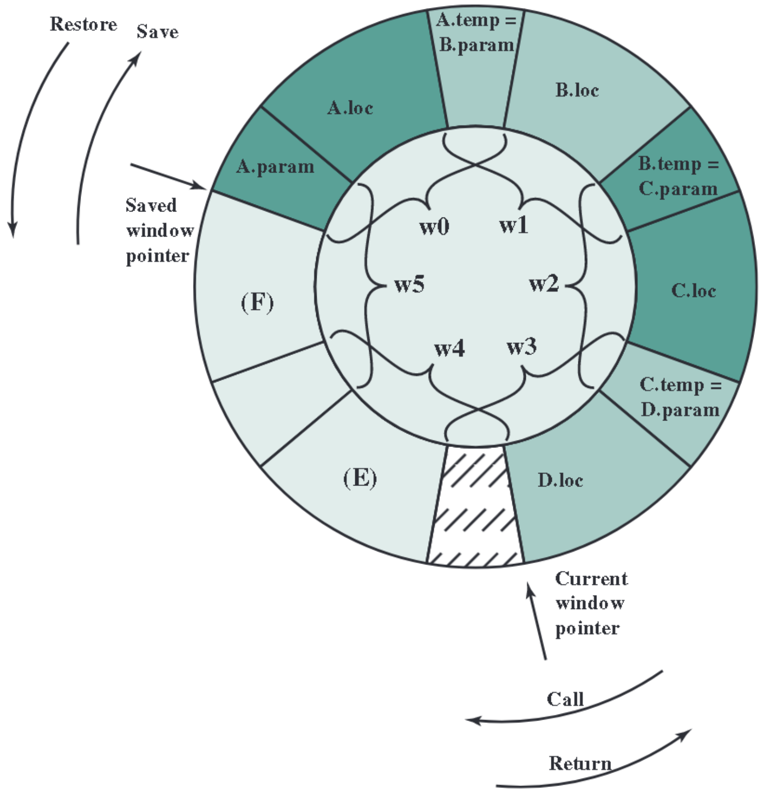


Figure 15.2 Circular-Buffer Organization of Overlapped Windows

The current-window pointer (CWP) points to the window of the currently active procedure.

The saved-window pointer (SWP) identifies the window most recently saved in memory. When CWP is incremented (modulo 6) so that it becomes equal to SWP, an interrupt occurs, and A's window is saved. Only the first two portions (A.in and A.loc) need be saved. Then, the SWP is incremented and the call to F proceeds. Similarly, when CWP is decremented and becomes equal to SWP, an interrupt occurs so that results in the restoration of A's window.

Global variables

全局变量可能被多个子程序调用。既可以被安置在mm中（这样很低效），也可以使用一组固定数量的全局变量寄存器.

Large Register File versus Cache

The register file, organized into windows, acts as a small, fast buffer for holding a subset of all variables that are likely to be used the most heavily. From this point of view, the register file acts much like a cache memory, although a much faster memory. 那要不干脆就用cache,是不是更简单更好?

优缺点对比:

1. 寄存器窗口大小固定,浪费;cache每次读一大块数据进来,浪费: A register file may make inefficient use of space, because not all procedures will need the full window space allotted to them. On the other hand, the cache suffers from another sort of inefficiency: Data are read into the cache in blocks. Whereas the register file contains only those variables in use, the cache reads in a block of data, some or much of which will not be used.
2. 对于全局变量的处理: The cache is capable of handling global as well as local variables. There are usually many global scalars, but only a few of them are heavily used [KATE83]. A cache will dynamically discover these variables and hold them. For reg files, when program modules are separately compiled, it is impossible for the compiler to assign global values to registers; the linker must perform this task.

3. 对内存的访问: For reg files, because this depth usually fluctuates within a narrow range, the use of memory is relatively infrequent. But most cache memories are set associative with a small set size. Thus, there is the danger that other data or instructions will compete for cache residency.

两者的优劣不是很明显.但是有一点是reg file一定更优的: This distinction shows up in the amount of **addressing overhead** experienced by the two approaches. reg只需要一个window编号和一个reg编号就可以定位到一个数据,而cache需要一个满长度的地址,寻址开销很大

15.3 Complier-Based Register Optimization

Each program quantity that is a candidate for residing in a register is assigned to a symbolic or virtual register. The compiler then maps the unlimited number of symbolic registers into a fixed number of real registers.

“The graph coloring problem is this. Given a graph consisting of nodes and edges, assign colors to nodes such that adjacent nodes have different colors, and do this in such a way as to minimize the number of different colors.”

“If two symbolic registers are “live” during the same program fragment, then they are joined by an edge to depict interference.”

15.4 Reduced Instruction Set Arithmetic

Why CISC

“Two principal reasons have motivated this trend: a desire to simplify compilers and a desire to improve performance.” 接下来论述CISC实际上是没有做到这两点的

The first of the reasons cited, compiler simplification, seems obvious, but it is not. “If there are machine instructions that resemble HLL statements, this task is simplified.” But they have found that complex machine instructions are often hard to exploit because the compiler must find those cases that exactly fit the construct. Most of the instructions in a compiled program are the relatively simple ones.也就是说,复杂的指令其实基本上没有被用到

The other major reason cited is the expectation that a CISC will yield smaller, faster programs. Let us examine both aspects of this assertion: that programs will be smaller and that they will execute faster. 简短的程序可以省空间,但是这一点已经不重要了因为存储空间很便宜. 小的程序也可以更快地执行: First, fewer instructions means fewer instruction bytes to be fetched. Second, in a paging environment, smaller programs occupy fewer pages, reducing page faults. Third, more instructions fit in cache(s).

但是编译出来的CISC程序不一定就比RISC的小. In many cases, the CISC program, expressed in symbolic machine language, may be shorter (i.e., fewer instructions), but the number of bits of memory occupied may not be noticeably smaller. 为什么呢?

- We have already noted that compilers on CISCs tend to favor simpler instructions, so that the conciseness of the complex instructions seldom comes into play.
- Also, because there are more instructions on a CISC, longer opcodes are required, producing longer instructions.
- Finally, RISCs tend to emphasize register rather than memory references, and the former require fewer bits. An example of this last effect is discussed presently.

Characteristics of Reduced Instruction Set Architectures

Although a variety of different approaches to reduced instruction set architecture have been taken, certain characteristics are common to all of them:

- One instruction per cycle
- Register-to-register operations
- Simple addressing modes
- Simple instruction formats

A machine cycle is defined to be the time it takes to fetch two operands from registers, perform an ALU operation, and store the result in a register.

One instruction per cycle: Thus, RISC machine instructions should be no more complicated than, and execute about as fast as, microinstructions on CISC machines. the machine instructions can be hardwired. Such instructions should execute faster than comparable machine instructions on other machines, because it is not necessary to access a microprogram control store during instruction execution.

register to register: This design feature simplifies the instruction set and therefore the control unit. Another benefit is that such an architecture encourages the optimization of register use, so that frequently accessed operands remain in high-speed storage.

simple addressing modes: this design feature simplifies the instruction set and the control unit.

simple instruction format: Generally, only one or a few formats are used. Instruction length is fixed and aligned on word boundaries. Field locations, especially the opcode, are fixed. This design feature has a number of benefits.

- opcode decoding and register operand accessing can occur simultaneously.
- simplify the control unit.
- Instruction fetching is optimized because word-length units are fetched.
- Alignment on a word boundary also means that a single instruction does not cross page boundaries.

由上述特征可以评估出RISC架构的可能优势

- First, more effective optimizing compilers can be developed. With more-primitive instructions, there are more opportunities for moving functions out of loops, reorganizing code for efficiency, maximizing register utilization, and so forth.
- A second point, already noted, is that most instructions generated by a compiler are relatively simple anyway.
- A third point relates to the use of instruction pipelining. RISC researchers feel that the instruction pipelining technique can be applied much more effectively with a reduced instruction set.
- RISC processors are more responsive to interrupts because interrupts are checked between rather elementary operations.

CISC versus RISC Characteristics

CISC和RISC实际上在相互借鉴

the following are considered typical of a classic RISC:

1. A single instruction size.
2. That size is typically 4 bytes.
3. A small number of data addressing modes, typically less than five. This parameter is difficult to pin down. In the table, register and literal modes are not counted and different formats with different offset sizes are counted separately.
4. No indirect addressing that requires you to make one memory access to get the address of another operand in memory.
5. No operations that combine load/store with arithmetic (e.g., add from memory, add to memory).
6. No more than one memory-addressed operand per instruction.
7. Does not support arbitrary alignment of data for load/store operations.
8. Maximum number of uses of the memory management unit (MMU) for a data address in an instruction.
9. Number of bits for integer register specifier equal to five or more. This means that at least 32 integer registers can be explicitly referenced at a time.
10. Number of bits for floating-point register specifier equal to four or more. This means that at least 16 floating-point registers can be explicitly referenced at a time.

15.5 RISC Pipelining

Load rA ← M	I	E	D										
Load rB ← M				I	E	D							
Add rC ← rA + rB							I	E					
Store M ← rC									I	E	D		
Branch X												I	E

(a) Sequential execution

Load rA ← M	I	E	D							
Load rB ← M		I		E	D					
Add rC ← rA + rB				I		E				
Store M ← rC						I	E	D		
Branch X							I		E	
NOOP									I	E

(b) Two-stage pipelined timing

Load rA ← M	I	E	D					
Load rB ← M		I	E	D				
NOOP			I	E				
Add rC ← rA + rB				I	E			
Store M ← rC					I	E	D	
Branch X						I	E	
NOOP							I	E

(c) Three-stage pipelined timing

Load rA ← M	I	E_1	E_2	D							
Load rB ← M		I	E_1	E_2	D						
NOOP			I	E_1	E_2						
NOOP				I	E_1	E_2					
Add rC ← rA + rB					I	E_1	E_2				
Store M ← rC						I	E_1	E_2	D		
Branch X							I	E_1	E_2		
NOOP								I	E_1	E_2	
NOOP									I	E_1	E_2

(d) Four-stage pipelined timing

Figure 15.6 The Effects of Pipelining

图a没有流水线，图b有流水线但是由于mm端口只有一个所以E和D不能同时进行，只能将整个流程划分为两个阶段；图c有两个端口，可以ED同时进行，但是由于data dependency需要加NOOP，图d考虑了E实际上要长一些，可以分成两个子阶段，并且考虑了branch

Optimization of Pipelining

通过将流水线划分得更细可以加速，但是data dependency 和branch妨碍了加速。下面是一些应对方法

Delayed branch: Delayed branch, a way of increasing the efficiency of the pipeline, makes use of a branch that does not take effect until after execution of the following instruction (hence the term delayed). The instruction location immediately following the branch is referred to as the delay slot.

最传统的流水线遇到跳转只能用额外的电路清理已经在流水线中的指令；RISC pipeline可以插入NOOP来避免使用额外的清理电路；还可以使用Reversed instructions。但是对于有条件跳转，For conditional branches, this procedure cannot be blindly applied. If the condition that is tested for the branch can be altered by the immediately preceding instruction, then the compiler must refrain from doing the interchange and instead insert a NOOP. Otherwise, the compiler can seek to insert a useful instruction after the branch. 感觉只要不影响条件跳转涉及到的量就行。

Delayed load：传统地，On LOAD instructions, the register that is to be the target of the load is locked by the processor. The processor then continues execution of the instruction stream until it reaches an instruction requiring that register, at which point it idles until the load is complete. If the compiler can rearrange instructions so that useful work can be done while the load is in the pipeline, efficiency is increased. 因为LOAD指令会锁寄存器，所以应就是尽量延后LOAD，让要用到相同寄存器的其他指令在LOAD之前完成，避免被LOAD耽搁

Loop unrolling：Unrolling replicates the body of a loop some number of times called the unrolling factor (u) and iterates by step u instead of step 1. Unrolling can improve the performance by

- reducing loop overhead （简单理解为循环次数）
- increasing instruction parallelism by improving pipeline performance （在一个循环中尽可能多的操作一些数，这样在循环内的并行化程度更高）
- improving register, data cache, or TLB locality （在一个循环中尽可能多的操作一些数，因为这些数是通过递增索引取来的，所以局部性很高，可以充分利用寄存器或cache）

15.7 SPARC

SPARC (Scalable Processor Architecture) refers to an architecture defined by Sun Microsystems.

As with the Berkeley RISC, the SPARC makes use of register windows. Each window gives addressability to 24 registers, and the total number of windows is implementation dependent and ranges from 2 to 32 windows.

The processor maintains a current window pointer (CWP), located in the processor status register (PSR) points to the window of the currently executing procedure. The window invalid mask (WIM), also in the PSR, indicates which windows are invalid.

15.8 The RISC versus CISC Controversy

人们如何比较RISC和CISC

- Quantitative: Attempts to compare program size and execution speed of programs on RISC and CISC machines that use comparable technology.
- Qualitative: Examines issues such as high-level language support and optimum use of VLSI real estate.

关于这两个方面的争论都没有定论

Chapter 16 Instruction-Level Parallelism and Superscalar Processors

A **superscalar** implementation of a processor architecture is one in which common instructions—integer and floating-point arithmetic, loads, stores, and conditional branches—can be initiated simultaneously and executed independently.

16.1 Overview

The term **superscalar** refers to a machine that is designed to improve the performance of the execution of scalar instructions.

The essence of the superscalar approach is the ability to execute instructions independently and concurrently in different pipelines.

Superscalar versus Superpipelined

An alternative approach to achieving greater performance is referred to as **superpipelining**. Superpipelining exploits the fact that many pipeline stages perform tasks that require less than half a clock cycle. Thus, a doubled internal clock speed allows the performance of two tasks in one external clock cycle.

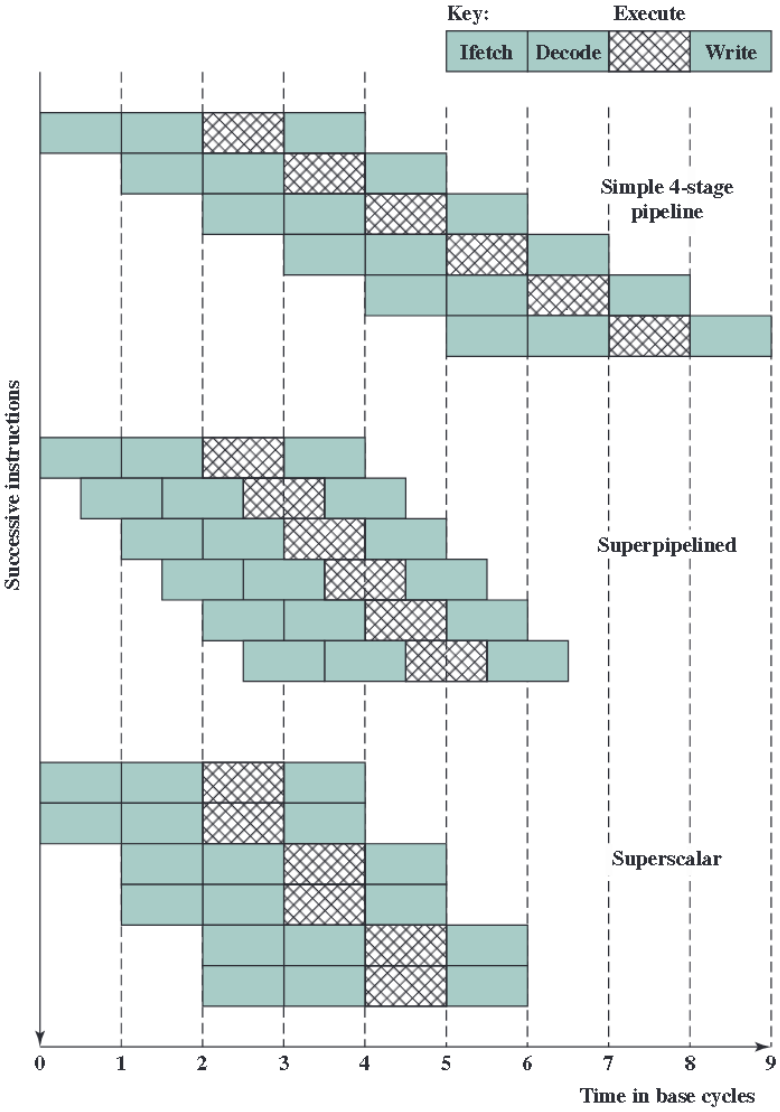


Figure 16.2 Comparison of Superscalar and Superpipeline Approaches

"An alternative way of looking at this is that the functions performed in each stage can be split into **two nonoverlapping parts** and each can execute in half a clock cycle. A superpipeline implementation that behaves in this fashion is said to be of degree 2."

Constraints

使用超标量的关键是并行化执行多条指令。这要看指令间的并行化水平

The term **instruction-level parallelism** (ILP) refers to the degree to which, on average, the instructions of a program can be executed in parallel. 一系列手段可以用来提升ILP，当然，也有一定的限制：

- True data dependency;
- Procedural dependency;
- Resource conflicts;
- Output dependency;
- Antidependency.

True data dependency: the second instruction needs data produced by the first instruction. (also called flow dependency or read after write [RAW] dependency). In general, any instruction must be delayed until all of its input values have been produced.

Procedural dependencies: The instructions following a branch (taken or not taken) have a procedural dependency on the branch and cannot be executed until the branch is executed.

另一种Procedural dependencies来自于可变长度的指令。Because the length of any particular instruction is not known, it must be at least partially decoded before the following instruction can be fetched（这样才可以知道下一条指令有多长，该取几个字）. This prevents the simultaneous fetching required in a superscalar pipeline.

Resource conflict: A resource conflict is a competition of two or more instructions for the same resource at the same time. Examples of resources include memories, caches, buses, register-file ports, and functional units (e.g., ALU adder).

数据依赖性和资源冲突的区别：For one thing, resource conflicts can be overcome by duplication of resources, whereas a true data dependency cannot be eliminated. Also, when an operation takes a long time to complete, resource conflicts can be minimized by pipelining the appropriate functional unit.

16.2 Design Issues

ILP is determined by the frequency of true data dependencies and procedural dependencies in the code. Or it is also determined by what [JOUP89a] refers to as **operation latency**: the time until the result of an instruction is available for use as an operand in a subsequent instruction.

Machine parallelism is a measure of the ability of the processor to take advantage of instruction-level parallelism. 机器并行度是建立在ILP之上的。 Machine parallelism is determined by the number of instructions that can be fetched and executed at the same time (the number of parallel pipelines) and by the speed and sophistication of the mechanisms that the processor uses to find independent instructions.

Instruction Issue Policy

the term **instruction issue** refers to the process of initiating instruction execution in the processor's functional units

the term **instruction issue policy** refers to the protocol used to issue instructions. In general, we can say that instruction issue occurs when instruction moves from the decode stage of the pipeline to the first execute stage of the pipeline. 一般可以分为3类

- In-order issue with in-order completion.
- In-order issue with out-of-order completion.
- Out-of-order issue with out-of-order completion.

in-order issue with in-order completion： issue instructions in the exact order that would be achieved by sequential execution (in-order issue) and to write results in that same order (in-order completion).

in-order issue with out-of-order completion： Instruction issuing is stalled by a resource conflict, a data dependency, or a procedural dependency. In addition to the aforementioned limitations, a new dependency, which we referred to earlier as an **output dependency** (also called write after write [WAW] dependency), arises. The issuing of the third instruction must be stalled if its result might later be overwritten by an older instruction that takes longer to complete.

out-of-order issue with out-of-order completion： To allow out-of-order issue, it is necessary to decouple the decode and execute stages of the pipeline. This is done with a buffer referred to as an instruction window. 只要指令窗口不满，就可以取指并解码，只要执行单元有空，就可以从指令窗口发射。窗口本身不是一个流水线阶段（当然要满足"所发射的指令对应所需的资源空闲"这一条件）As before, the only constraint is that the program execution behaves correctly.

In addition, a new dependency, which we referred to earlier as an antidependency (also called write after read [WAR] dependency), arises. "the second instruction destroys a value that the first instruction uses."

一种常用于支持失序完成的技术： The **reorder buffer** is temporary storage for results completed out of order that are then committed to the register file in program order. A related concept is Tomasulo's algorithm

Register Renaming

When a new register value is created (i.e., when an instruction executes that has a register as a destination operand), a new register is allocated for that value.

When a new allocation is made for a particular logical register, subsequent instruction references to that logical register as a source operand are made to refer to the most recently allocated hardware register (recent in terms of the program sequence of instructions).

Machine Parallelism

提供了一张图，展示了加速因子与资源量、指令窗口大小以及是否renaming之间的关系

The first is that it is probably not worthwhile to add functional units without register renaming. There is some slight improvement in performance, but at the cost of increased hardware complexity. With register renaming, which eliminates antidependencies and output dependencies, noticeable gains are achieved by adding more functional units.

There is a significant difference in the amount of gain achievable between using an instruction window of 8 versus a larger instruction window.

Superscalar Execution

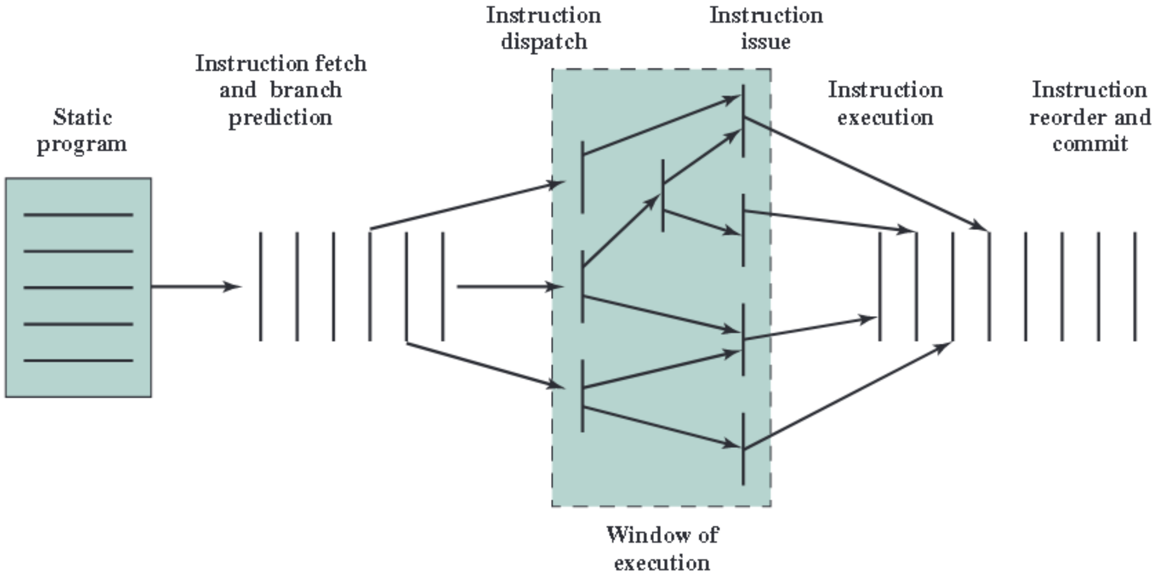


Figure 16.7 Conceptual Depiction of Superscalar Processing

1. The program to be executed consists of a linear sequence of instructions. This is the static program as written by the programmer or generated by the compiler.
2. The instruction fetch stage, which includes branch prediction, is used to form a dynamic stream of instructions. This stream is examined for dependencies, and the processor may remove artificial dependencies.
3. The processor then dispatches the instructions into a window of execution. In this window, instructions no longer form a sequential stream but are structured according to their true data dependencies.
4. The processor executes each instruction in an order determined by the true data dependencies and hardware resource availability.
5. Finally, instructions are conceptually put back into sequential order and their results are recorded.

The final step mentioned in the preceding paragraph is referred to as **committing**, or **retiring**, the instruction.这一步是必要的，因为有时候分支预测导致一些结果被提前算出来，不知道对不对（taken or not taken）这时候就需要先存起来，等分支结果出来后再确认提交

Superscalar Implementation

hardware required for the superscalar approach:

- 同时多取与分支预测逻辑：Instruction fetch strategies that simultaneously fetch multiple instructions, often by predicting the outcomes of, and fetching beyond, conditional branch instructions. These functions require the use of multiple pipeline fetch and decode stages, and branch prediction logic.
- 依赖性检测逻辑：Logic for determining true dependencies involving register values, and mechanisms for communicating these values to where they are needed during execution.
- 并行执行机制：Mechanisms for initiating, or issuing, multiple instructions in parallel.

- 为并行化提供的资源要足够：Resources for parallel execution of multiple instructions, including multiple pipelined functional units and memory hierarchies capable of simultaneously servicing multiple memory references.
- 从无序恢复到有序的提交机制：Mechanisms for committing the process state in correct order.

Chapter 20 Control Unit Operation

要想清晰地了解一个处理器的功能，需要明确以下这些内容

functional requirements for a processor:

- Operations (opcodes)
- Addressing modes
- Registers
- I/O module interface
- Memory module interface
- Interrupts

They determine what a processor must do. 这些内容在前述章节探讨过，这一章主要讨论这些功能如何实现

20.1 Micro-operation

取指，执指等过程可以进一步细分。In fact, we will see that each of the smaller cycles involves a series of steps, each of which involves the processor registers. We will refer to these steps as micro-operations.

To summarize, the execution of a program consists of the sequential execution of **instructions**. Each instruction is executed during an instruction cycle made up of shorter **subcycles** (e.g., fetch, indirect, execute, interrupt). The execution of each subcycle involves one or more shorter operations, that is, **micro-operations**.

The Fetch Cycle

Four registers are involved:

- Memory address register (MAR): Is connected to the address lines of the system bus. It specifies the address in memory for a read or write operation.
- Memory buffer register (MBR): Is connected to the data lines of the system bus. It contains the value to be stored in memory or the last value read from memory.
- Program counter (PC): Holds the address of the next instruction to be fetched.
- Instruction register (IR): Holds the last instruction fetched.

取指可以分为下面几个阶段

```
t1: MAR <- (PC)
t2: MBR <- Memory
    PC <- (PC) + I
t3: IR <- (MBR)
```

I is the instruction length. PC自增这一步也可以放到第三个时隙。

The groupings of micro-operations must follow two simple rules:

- The proper sequence of events must be followed. 防止data dependency
- Conflicts must be avoided. 避免在同一时刻访问同一资源

The Indirect Cycle

```
t1: MAR <- (IR(addr.))
t2: MBR <- Memory
t3: IR(addr.) <- (MBR(addr.))
```

注意，间址微操作的开头就是IR中已经有指令，只不过指令的地址域还需要寻址，所以只把IR(addr.)的内容拿给MAR即可；同理，间址拿回来在MBR中的内容也只需要更新IR的地址域中的内容。

The Interrupt Cycle

```
t1: MBR <- (PC)
t2: MAR <- Save_Address
    PC <- Routine_Address
t3: Memory <- (MBR)
```

The Execute Cycle

The fetch, indirect, and interrupt cycles are simple and predictable. 但是对于执指阶段，由于操作码的多样性，并不能简单的总结其微指令过程。需要解码，看操作具体是什么。

The control unit examines the opcode and generates a sequence of micro-operations based on the value of the opcode. This is referred to as **instruction decoding**.

接下来介绍了几个执指的具体例子，分析其微指令流程。分别对应一般指令、条件跳转指令和子程序调用指令

ADD R1, X

```
t1: MAR <- (IR(address))
t2: MBR <- Memory
t3: R1 <- (R1) + (MBR)
```

Increment and skip if zero: ISZ X

The content of location X is incremented by 1. If the result is 0, the next instruction is skipped.

```
t1: MAR <- (IR(address))
t2: MBR <- Memory
t3: MBR <- (MBR) + 1
t4: Memory <- (MBR) If ((MBR) = 0) then (PC d (PC) + I)
```

执指阶段默认IR已经有待执行的指令了，不用考虑间址。

Branch-and-save-address instruction: BSA X

The address of the instruction that follows the BSA instruction is saved in location X, and execution continues at location X + I. The saved address will later be used for return. This is a straightforward technique for supporting subroutine calls.

```
t1: MAR <- (IR(address)) ; 下一条本应执行的指令要存到指令中给出的地址，也就是X
    MBR <- (PC)
t2: PC <- (IR(address)) ; 用X更新PC
    Memory <- (MBR)
t3: PC <- (PC) + I ;使得下一条要执行的指令为X+I
```

The Instruction Cycle

对于取指，执指等指令subcycles，都有一套微指令流程，可以将其绑定起来，用一套编码来指示，某一个码对应一套微指令流程。这套编码存放在一个新的reg中，called the instruction cycle code (ICC). The ICC designates the state of the processor in terms of which portion of the cycle it is in： 00: Fetch； 01: Indirect； 10: Execute ； 11: Interrupt

每一个subcycle结束后，ICC都会被设置为一个值。比如The indirect cycle is always followed by the execute cycle. The interrupt cycle is always followed by the fetch cycle

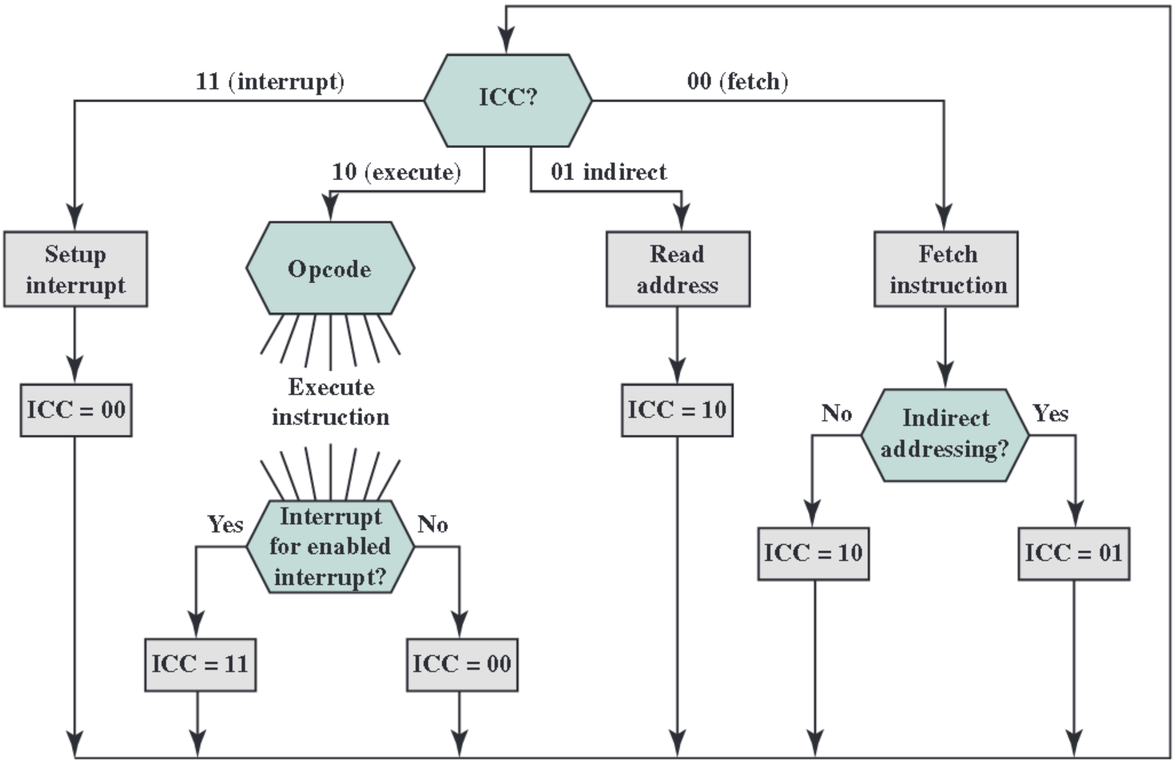


Figure 20.3 Flowchart for Instruction Cycle

以上，这一节完成了对subcycle的拆解，拆解为微操作，下面探讨如何实现微操作。

20.2 Control of the Processor

We can define the **functional requirements** for the control unit: those functions that the control unit must perform. 定义控制单元的基本组成元素、定义CU必须完成的微操作、定义CU必须有的功能，来完成微操作。

首先定义基本组成元素 First, the basic functional elements of the processor are the following:

- ALU

- Registers
- Internal data paths: used to move data between registers and between register and ALU
- External data paths: link registers to memory and I/O modules, often by means of a system bus.
- Control unit

然后定义必须完成的微操作 All microoperations fall into one of the following categories:

- Transfer data from one register to another.
- Transfer data from a register to an external interface (e.g., system bus).
- Transfer data from an external interface to a register.
- Perform an arithmetic or logic operation, using registers for input and output.

The control unit performs two basic tasks:

- Sequencing: The control unit causes the processor to step through a series of micro-operations in the proper sequence, based on the program being executed.
- Execution: The control unit causes each micro-operation to be performed.

CU的常见控制信号(inputs)：

- Clock: This is how the control unit “keeps time.” The control unit causes one micro-operation (or a set of simultaneous micro-operations) to be performed for each clock pulse. This is sometimes referred to as the processor cycle time, or the clock cycle time.
- Instruction register: The opcode and addressing mode of the current instruction are used to determine which micro-operations to perform during the execute cycle.
- Flags
- Control signals from control bus

outputs:

- Control signals within the processor: These are two types: those that cause data to be moved from one register to another, and those that activate specific ALU functions.（向内部发出的控制信号)
- Control signals to control bus: These are also of two types: control signals to memory, and control signals to the I/O modules.（向外部发出的控制信号）

Three types of control signals are used: 1.those that activate an ALU function; 2. those that activate a data path; 3. and those that are signals on the external system bus or other external interface.

给出了一个具体的例子

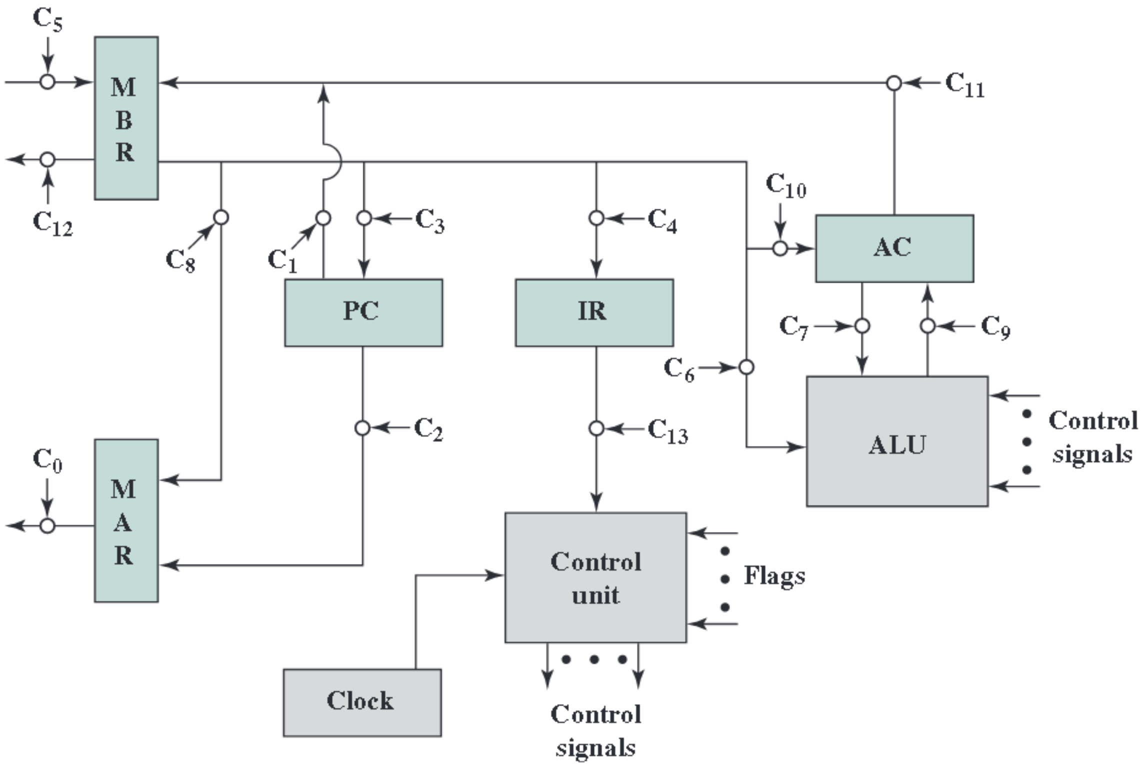


Figure 20.5 Data Paths and Control Signals

Table 20.1 Micro-operations and Control Signals

	Micro-operations	Active Control Signals
Fetch:	t_1: MAR ← (PC)	C_2
	t_2: MBR ← Memory PC ← (PC) + 1	C_5, C_R
	t_3: IR ← (MBR)	C_4
Indirect:	t_1: MAR ← (IR(Address))	C_8
	t_2: MBR ← Memory	C_5, C_R
	t_3: IR(Address) ← (MBR(Address))	C_4
Interrupt:	t_1: MBR ← (PC)	C_1
	t_2: MAR ← Save-address PC ← Routine-address	
	t_3: Memory ← (MBR)	C_{12}, C_W

C_R = Read control signal to system bus.

C_W = Write control signal to system bus.

Internal Processor Organization

内部一般采用总线结构而不用上面例子所展示的两两连接。优点有两个：The use of common data paths simplifies the interconnection layout and the control of the processor. Another practical reason for the use of an internal bus is to save space.

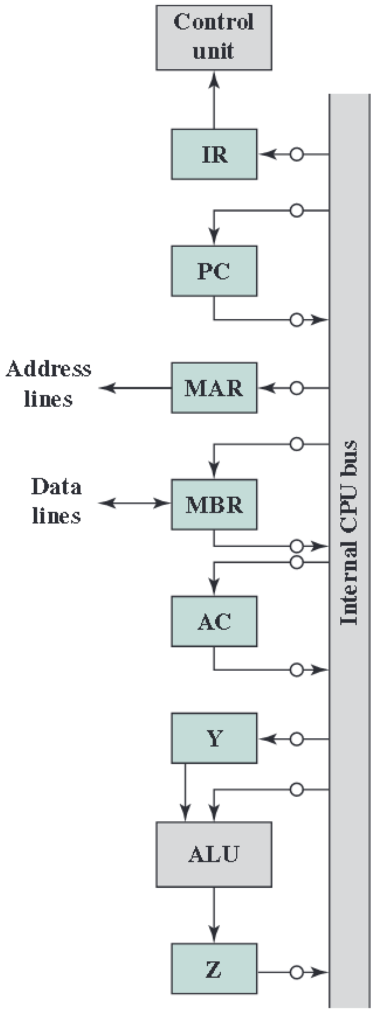


Figure 20.6 CPU with Internal Bus

注意因为AC是组合逻辑电路，并且得到的计算结果会上总线然后反馈到自身，所以需要先输出一个寄存器，不能直接上总线

20.3 Hardwired Implementation

在物理上怎么实现上述的控制功能？有两种实现方法：

- Hardwired implementation: the control unit is essentially a state machine circuit. Its input logic signals are transformed into a set of output logic signals, which are the control signals.
- Microprogrammed implementation

Control Unit Inputs

The key inputs are the IR, the clock, flags, and control bus signals.

IR: To simplify the control unit logic, there should be a unique logic input for each opcode. This function can be performed by a decoder, which takes an encoded input and produces a single output.

Clock: we would like a counter as input to the control unit, with a different control signal being used for T1,T2, and so forth. At the end of an instruction cycle, the control unit must feed back to the counter to reinitialize it at T1. 比如执指永远只发生在取指或间址之后，所以一个控制门的表达式可以写成操作码、状态码和时间的布尔表达式，比如

$$C_5 = \overline{P} \bullet \overline{Q} \bullet T_2 + \overline{P} \bullet Q \bullet T_2 + P \bullet \overline{Q} \bullet (\mathrm{LDA} + \mathrm{ADD} + \mathrm{AND}) \bullet T_2$$

这种硬布线的方式非常复杂

Chapter 21 Micro-programmed Control

21.1 basic Concepts

Microinstructions

In addition to the use of control signals, each microoperation is described in symbolic notation. This notation (is) known as a **microprogramming language**. Each line describes a set of micro-operations occurring at one time and is known as a **microinstruction**. A sequence of instructions is known as a **microprogram**, or **firmware**（固件）.

因为控制信号本质上就是一系列的0和1，所以 we could construct a **control word** in which each bit represents one control line. Then each micro-operation would be represented by a different pattern of 1s and 0s in the control word. ... So let us put our control words in a memory, with each word having a unique address. Now add an address field to each control word, indicating the location of the next control word to be executed if a certain condition is true. 这导致了水平微指令的产生

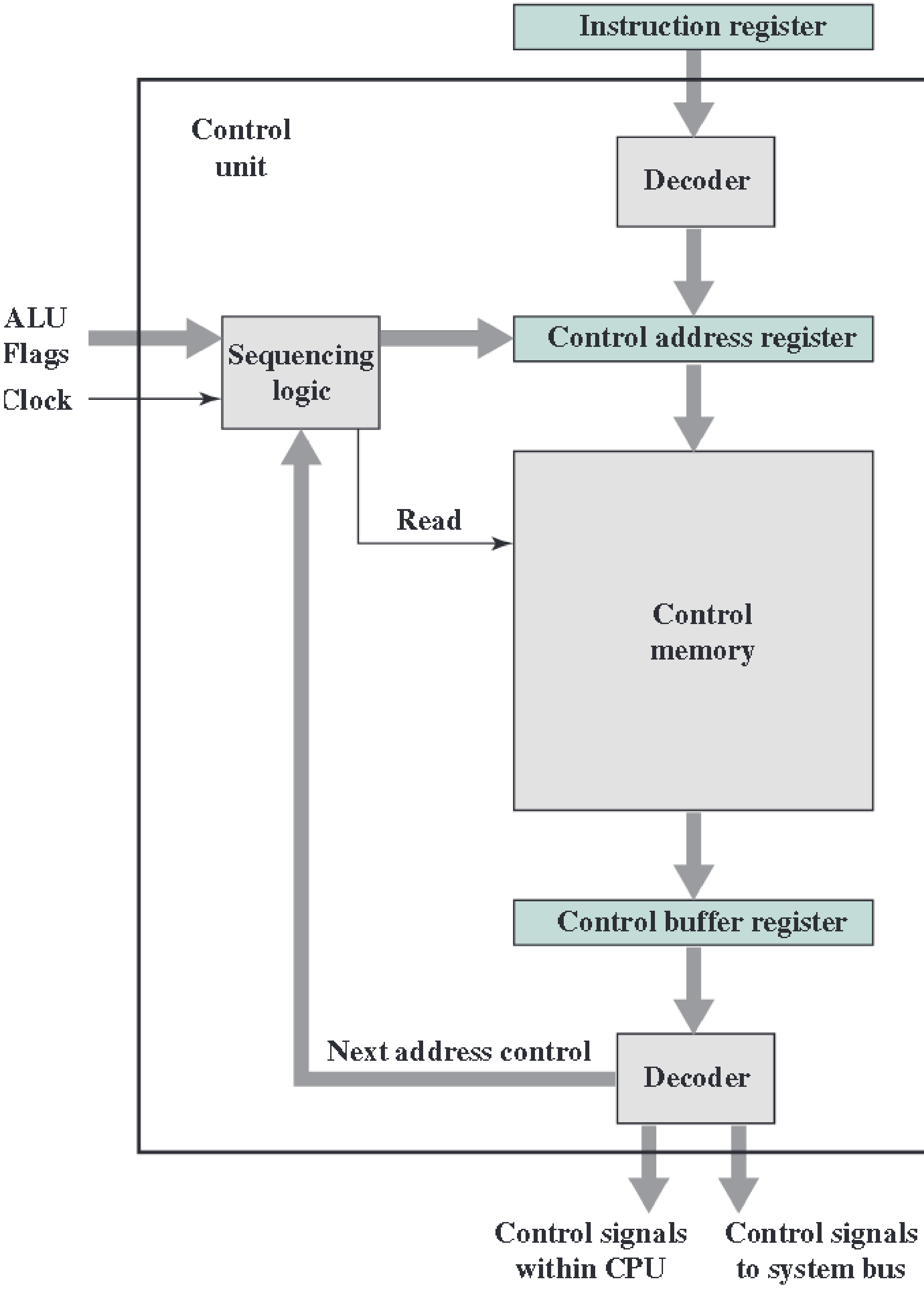
horizontal microinstruction: There is one bit for each internal processor control line and one bit for each system bus control line. There is a condition field indicating the condition under which there should be a branch, and there is a field with the address of the microinstruction to be executed next when a branch is taken.（注意，是当condition bit为1（真）时，地址域所指示的指令才会被执行，否则下一条顺序执行）

Microprogrammed Control Unit

The set of microinstructions is stored in the **control memory**. The **control address register** (CAR) contains the address of the next microinstruction to be read （有点像PC）. When a microinstruction is read from the control memory, it is transferred to a **control buffer register** (CBR) （有点像IR）.

在一个时钟周期里，CU做了四件事

1. sequencing logic unit issues a READ command to the control memory.
2. The word whose address is specified in the control address register is read into the control buffer register 微指令进CBR
3. 微指令发力：输出控制信号与下一指令的信息，后者给到排序单元（由于微指令的最左边就是它要发出的控制信号，所以读微指令相当于执行微指令）
4. 排序单元根据下条指令的信息和ALU flag加载下一条指令的地址到CAR。具体判决情况分为3种：
    1. Get the next microinstruction 比如正常情况下间址之后是执指
    2. Jump to a new routine based on a jump microinstruction 比如取指，中断等等
    3. Jump to a machine instruction routine：Load the control address register based on the opcode in the IR. 比如在执指阶段，根据opcode来决定具体要执行ADD还是AND等（这用到了图中上面的decoder）



The upper decoder translates the opcode of the IR into a control memory address. The lower decoder is not used for horizontal microinstructions but is used for **vertical microinstructions**： In a vertical microinstruction, a code is used for each action to be performed [e.g., MAR <- (PC)], and the decoder translates this code into individual control signals.

垂直微指令的优缺点：The advantage of vertical microinstructions is that they are more compact (fewer bits) than horizontal microinstructions, at the expense of a small additional amount of logic and time delay.

Wilkes Control

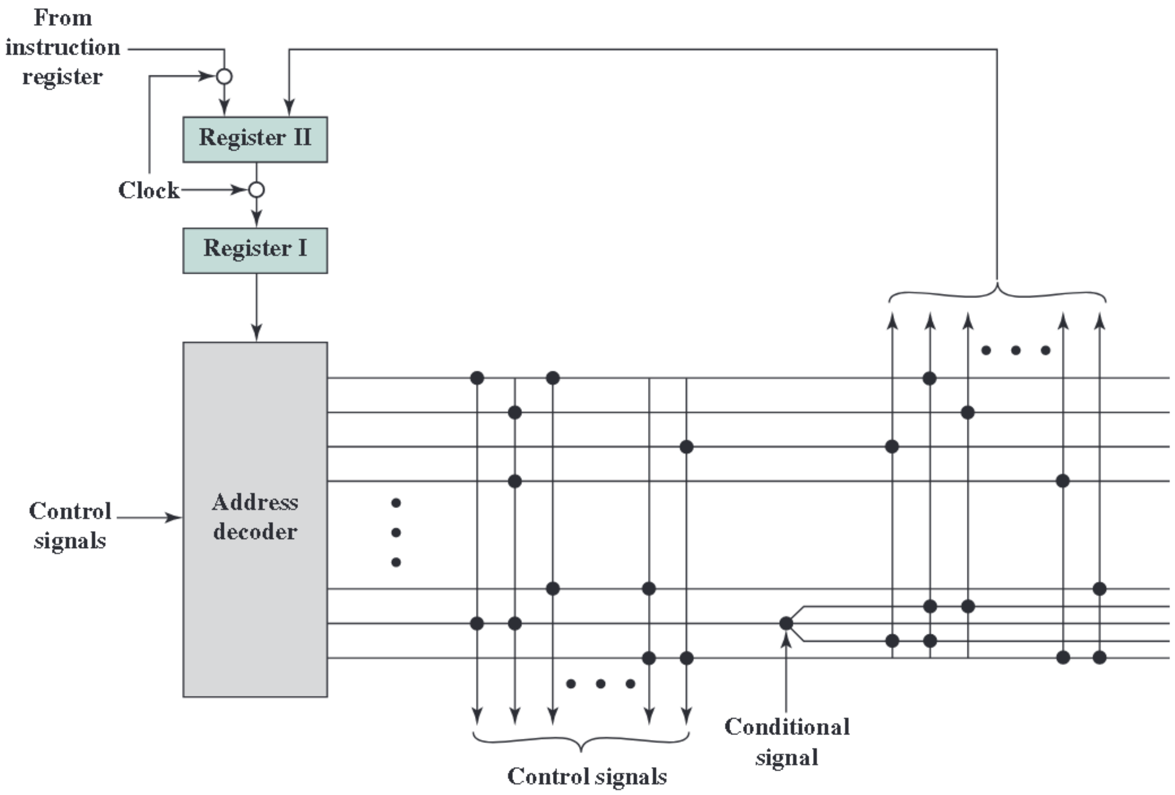


Figure 21.5 Wilkes's Microprogrammed Control Unit

The first part of the row generates the control signals that control the operation of the processor. The second part generates the address of the row to be pulsed in the next machine cycle. Thus, each row of the matrix is one microinstruction, and the layout of the matrix is the control memory.

为什么需要两个寄存器：Because the decoder is simply a combinatorial circuit; with only one register, the output would become the input during a cycle, causing an unstable condition.

需要注意，这种实现下，下一条指令的来源只能是second part，不会是CAR自增 + 1

Advantages and Disadvantages

微编程式的控制单元实现，与硬布线的实现相比，优点：简单，所以便宜且不易出错；缺点：慢一些。

一般纯CISC架构用微编程，因为简单；RISC因为指令集本来就比较简单，所以用硬布线

21.2 Microinstruction Sequencing

The two basic tasks performed by a microprogrammed control unit are as follows:

- Microinstruction sequencing: Get the next microinstruction from the control memory.
- Microinstruction execution: Generate the control signals needed to execute the microinstruction.

Two concerns are involved in the design of a microinstruction sequencing technique: the size of the microinstruction and the address-generation time.

Sequencing Techniques

Sequencing其实就是决定下一条指令的地址的来源。注意到执行一个微程序时，下一条指令的来源只有三种

- Determined by instruction register (only once per instr. cycle, after fetch)
- Next sequential address (most common)
- Branch

有不同的方法来决定指令来源，这些方法可以由微指令中地址域的格式来划分：双地址域，单地址域和变长地址域

双地址域方式：微指令的两个地址域和IR都被送到一个复用器，复用器通过地址选择信号决定将哪一个送到CAR，CAR中的内容再送去解码得到下一条指令的地址。地址选择信号的来源于分支逻辑，其输入是CU flags以及微指令中的一些控制比特。双地址域的方式简单但是需要更长的指令。

更常用的是单地址域：复用器的三个输入是（CAR）+1（即下一顺序地址）、微指令的地址域以及IR。注意到很多时候连单地址域中的信息也很难用到，所以有变长地址域方式。

变长地址域方式用到了一个模式选择位。在一个模式下，微指令除了模式选择bit，其他全是控制信号；下一条指令来自于下一顺序地址或者由IR推导得到；在另一种模式下，除了模式选择比特，还用一些比特驱动分支选择模块，剩下的位用来提供一个地址。在第二种模式下，整个周期都用来决定下一条指令的地址了，没有提供控制信号的输出。所以有点浪费时间

Address Generation

上面一小节从格式角度探讨了逻辑需求，现在从“地址如何产生或被计算/推导出来”这一角度来看排序问题

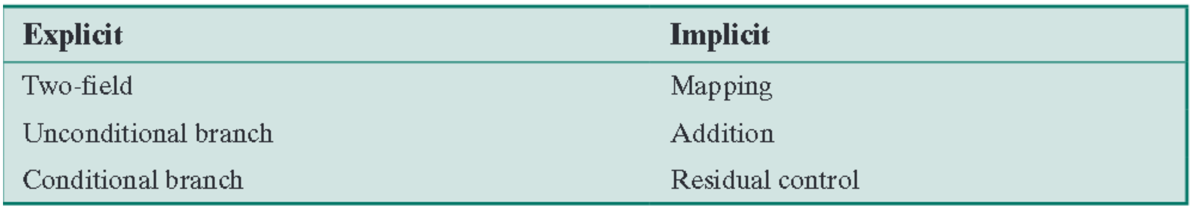
地址产生技术可以分为两部分：显式的和隐式的。显式的包括上面提到的双地址域，以及条件和无条件跳转。隐式的有下面三种

One of these, mapping, is required with virtually all designs. The opcode portion of a machine instruction must be mapped into a microinstruction address. This occurs only once per instruction cycle.

A common implicit technique is one that involves combining or adding two portions of an address to form the complete address.

residual control: This approach involves the use of a microinstruction address that has previously been saved in temporary storage within the control unit.

Table 21.3 Microinstruction Address Generation Techniques



Explicit	Implicit
Two-field	Mapping
Unconditional branch	Addition
Conditional branch	Residual control

21.3Microinstruction Execution

The effect of the execution of a microinstruction is to generate control signals. 产生的控制信号分为两种 Some of these signals control points internal to the processor. The remaining signals go to the external control bus or other external interface. 还有附加作用 As an incidental function, the address of the next microinstruction is determined.

A Taxonomy of Microinstructions

- ertical/horizontal
- Packed/unpacked
- Hard/soft microprogramming
- Direct/indirect encoding

最简单的一种微指令是，每一个控制信号对应一个比特，如果有K个控制信号需要被产生，那么就有2^K种可能的组合，对应2^K个微指令。但是实际上有一些控制信号的组合是永远不会发生的。比如

- 两个数据源不可能同时被门控到同一个目的地
- 一个寄存器不能既做源又做目的地
- ALU和外部控制总线每次都只能接收一个模式的控制信号

所以实际上可能的微指令数应该小于2^K，也就是说，可以用小于K位的微指令来产生K位的控制信号，这就引出了编码问题。按理说，可以列出所有可能的控制信号组合，然后只对这些组合编码，这是最紧凑的模式，但是不太可能实现，因为

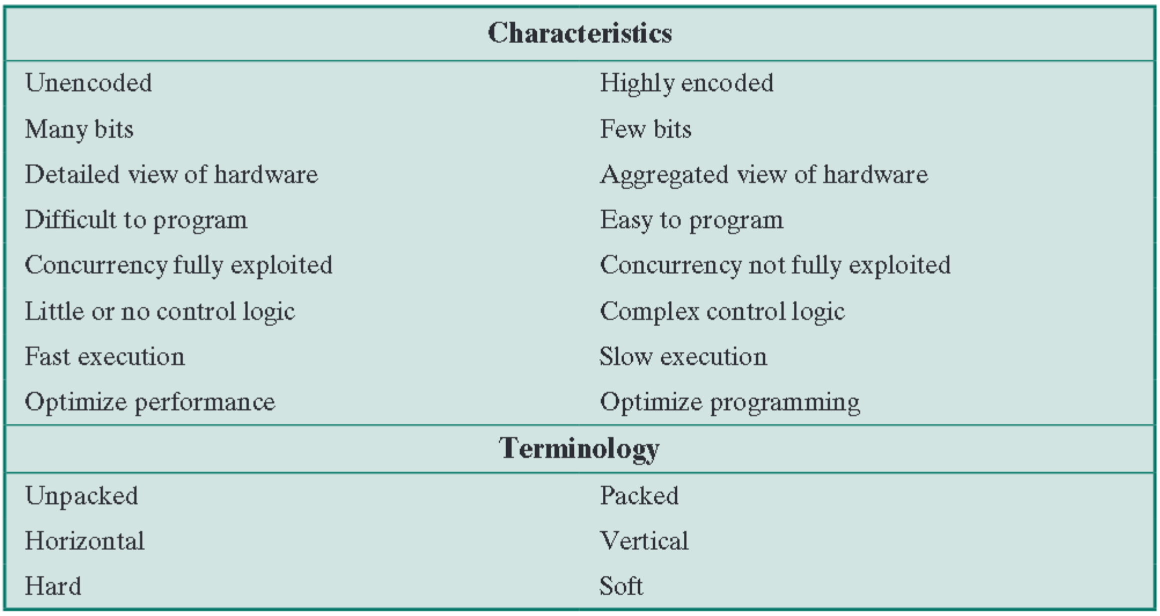
- it is as difficult to program as a pure decoded (Wilkes) scheme. This point is discussed further presently.
- It requires a complex and therefore slow control logic module.

为此我们做出一些妥协

- More bits than are strictly necessary are used to encode the possible combinations. 这需要更多位的微指令
- Some combinations that are physically allowable are not possible to encode. 这会省一些比特

总的效果是，比纯encoded用的比特多，但是比不编码用的比特少

Table 21.4 The Microinstruction Spectrum



Characteristics	
Unencoded	Highly encoded
Many bits	Few bits
Detailed view of hardware	Aggregated view of hardware
Difficult to program	Easy to program
Concurrency fully exploited	Concurrency not fully exploited
Little or no control logic	Complex control logic
Fast execution	Slow execution
Optimize performance	Optimize programming
Terminology	
Unpacked	Packed
Horizontal	Vertical
Hard	Soft

In general, a design that falls toward the left end of the spectrum is intended to optimize the performance of the control unit. Designs toward the right end are more concerned with optimizing the process of microprogramming.

关于上图的Terminology有如下解释

The degree of packing relates to the degree of identification between a given control task and specific microinstruction bits. As the bits become more packed, a given number of bits contains more information. Thus, packing connotes encoding.

The terms horizontal and vertical relate to the relative width of microinstructions. [SIEW82] suggests as a rule of thumb that vertical microinstructions have lengths in the range of 16 to 40 bits and that horizontal microinstructions have lengths in the range of 40 to 100 bits. 即垂直微指令一般比特数更少。

> ⓘ **Note**
>
> 水平微指令不代表没有编码，只是编码度低一些

The terms hard and soft microprogramming are used to suggest the degree of closeness to the underlying control signals and hardware layout. Hard microprograms are generally fixed and committed to read-only memory. Soft microprograms are more changeable and are suggestive of user microprogramming.

Microinstruction Encoding

那么如何进行编码呢？一般微指令的控制信号部分会分为几个域，每一个域产生一些控制信号。域和域之间的控制信号是互斥的，互不影响。有两种方式来将编码后的微指令安排进不同的域中

- functional encoding: The functional encoding method identifies functions within the machine and designates fields by function type. 比如一个域专门用来指示数据转移指令，它指出数据转移指令的源或目的地
- Resource encoding views the machine as consisting of a set of independent resources and devotes one field to each (e.g., I/O, memory, ALU).比如，一个域专门用来管I/O口是输入还是输出

另一个编码要考虑的问题是direct or indirect

With indirect encoding, one field is used to determine the interpretation of another field. 比如一个用来指示ALU操作的域，其码字不能完全决定ALU的行为，比如一个码字只能让ALU做移位，需要另一个域的比特来指示是逻辑移位还是算术移位。This technique generally implies two levels of decoding, increasing propagation delays.