解释：Reg[rd] = Reg[rs1] AND Reg[rs2]

4.1.1 [5]<4.3>对于上述指令，图4-10中的控制信号各是什么数值？

4.1.2 [5]<4.3>对于上述指令，将用到哪些功能单元？

4.1.3 [10]<4.3>对于上述指令，哪些功能单元不产生任何输出？哪些功能单元的输出不会被用到？

4.2 [10]<4.4>解释图4-18中的每一个“无关项”。

4.3 根据下述指令组合回答问题。

R-type	I-type (non-ld)	Load	Store	Branch	Jump
24%	28%	25%	10%	11%	2%

4.3.1 [5]<4.4>发生数据访存的指令所占比例？

4.3.2 [5]<4.4>发生指令访存的指令所占比例？

4.3.3 [5]<4.4>使用符号扩展的指令所占比例？

4.3.4 [5]<4.4>当不需要符号扩展的结果时，符号扩展单元的行为是什么？

4.4 在制造硅芯片时，材料(例如硅)缺陷和制造错误会导致电路故障。一个非常常见的故障是，信号线发生“断路”，导致总是保持逻辑“0”。这被称为“固定为0”故障。

4.4.1 [5]<4.4>如果MemToReg信号发生以上故障，哪些指令会执行错误？

4.4.2 [5]<4.4>如果ALUSrc信号发生以上故障，哪些指令会执行错误？

4.5 本题中，我们将仔细讨论单周期数据通路中执行指令的细节。假设本周期处理器取来指令：0x00c6ba32。

4.5.1 [10]<4.4>此时ALU控制单元的输入值是多少？

4.5.2 [5]<4.4>该指令执行结束后新的PC地址是多少？标注出计算该PC值的通路。

4.5.3 [10]<4.4>对于每一个多选器，给出执行该指令时的各个输入和输出值，列出寄存器Reg[xn]中的数值。

4.5.4 [10]<4.4>此时ALU和另两个加法单元的输入数值是多少？

4.5.5 [10]<4.4>寄存器堆的所有的输入数值是多少？

4.6 4.4节中没有讨论I型指令，如addi或者andi。

4.6.1 [5]<4.4>如果要基于图4-21添加I型指令，需要额外的什么逻辑单元？

4.6.2 [10]<4.4>为addi指令列出控制单元产生的信号值，如果有控制信号为无关项，给出理由。

4.7 假设用来实现处理器数据通路的各功能模块延迟如下所示：

I-Mem/ D-Mem	Register File	Mux	ALU	Adder	Single gate	Register Read	Register Setup	Sign extend	Control
250ps	150ps	25ps	200ps	150ps	5ps	30ps	20ps	50ps	50ps

其中，寄存器读延迟指的是，时钟上升沿到寄存器输出端稳定输出新值所需的时间。该延迟仅针对PC寄存器。寄存器建立时间指的是，寄存器的输入数据稳定到时钟上升沿所需的时间。该数值针对PC寄存器和寄存器堆。

4.7.1 [5]<4.4>R型指令的延迟是多少？比如，如果想让这类指令工作正确，时钟周期最少为多少？

4.7.2 [10]<4.4>ld指令的延迟是多少？仔细检查你的答案，许多学生会在关键路径上添加额外的寄存器。

4.7.3 [10]<4.4>sd指令的延迟是多少？仔细检查你的答案，许多学生会在关键路径上添加额外的

寄存器。

4.7.4 [5] <4.4> beq 指令的延迟是多少?

4.7.5 [5] <4.4> I 型指令的延迟是多少?

4.7.6 [5] <4.4> 该 CPU 的最小时钟周期是多少?

4.8 [10] <4.4> 假设你能设计一款处理器并让每条指令执行不同的周期数。给定指令比例如下表所示，相比图 4-21 中的处理器，这款新处理器的加速比是多少?

R-type/I-type (non-ld)	ld	sd	beq
52%	25%	11%	12%

4.9 如果在图 4-21 中的 CPU 中添加一个乘法器，这将给 ALU 增添 300ps 的延迟。但是，由于不再需要对乘法指令进行模拟，指令数量将会减少 5%。

4.9.1 [5] <4.4> 改动前后，时钟周期分别是多少?

4.9.2 [10] <4.4> 通过这个改动，将获得多少加速比?

4.9.3 [10] <4.4> 在保证提升性能的条件下，新 ALU 最低频率是多少?

4.10 设计者对处理器数据通路进行改造，通常会根据性价比做出方案折中。在下面三个问题中，若以图 4-21 为数据通路的改造基础，各单元延迟参考习题 4.7，成本如下：

I-Mem	Register File	Mux	ALU	Adder	D-Mem	Single Register	Sign extend	Single gate	Control
1000	200	10	100	30	2000	5	100	1	500

假设将通用寄存器的数量加倍，由 32 个扩展为 64 个，这将让 ld 和 sd 指令的数量减少 12%，但会增加寄存器堆的延迟，由 150ps 增至 160ps，同时还会增加成本，由 200 变为 400。(使用习题 4.8 中的指令组合，忽略习题 2.18 中讨论的其他 ISA 的影响。)

4.10.1 [5] <4.4> 增加这样的改进后，获得的加速比为多少?

4.10.2 [10] <4.4> 比较性能上的变化和成本上的变化。

4.10.3 [10] <4.4> 根据上述计算出来的性价比，考虑增加寄存器数量的优化方案。试给出两个场景，一个说明该优化的有效性，另一个则相反。

4.11 尝试添加 RISC-V 中的指令：`lwi.d rd, rs1, rs2` (地址自增的 load 指令)。

指令释义：`Reg[rd] = Mem [Reg[rs1] +Reg[rs2]]`

4.11.1 [5] <4.4> 对于这条指令，需要添加的新功能部件是什么?

4.11.2 [5] <4.4> 现有的哪些功能部件需要改造?

4.11.3 [5] <4.4> 对于这条指令，需要新添加的数据通路是什么?

4.11.4 [5] <4.4> 为支持这条指令，为控制单元新添加的控制信号有哪些?

4.12 尝试添加 RISC-V 中的指令：`swap rs1, rs2`。

指令释义：`Reg[rs2] = Reg[rs1]; Reg[rs1] = Reg[rs2]`

4.12.1 [5] <4.4> 对于这条指令，需要添加的新功能部件是什么?

4.12.2 [10] <4.4> 现有的哪些功能部件需要改造?

4.12.3 [5] <4.4> 对于这条指令，需要新添加的数据通路是什么?

4.12.4 [5] <4.4> 为支持这条指令，为控制单元新添加的控制信号有哪些?

4.12.5 [5] <4.4> 修改图 4-21 并实现该条指令。

4.13 尝试添加 RISC-V 中的指令：`ss rs1, rs2, imm` (存储两数之和)。

指令释义：Mem[Reg[rs1]] = Reg[rs2] + immediate

4.13.1 [10]<4.4>对于这条指令，需要添加的新功能部件是什么？

4.13.2 [10]<4.4>现有的哪些功能部件需要改造？

4.13.3 [5]<4.4>对于这条指令，需要新添加的数据通路是什么？

4.13.4 [5]<4.4>为支持这条指令，为控制单元新添加的控制信号有哪些？

4.13.5 [5]<4.4>修改图 4-21 并实现该条指令。

4.14 [5]<4.4>对于哪些指令，立即数产生单元（Imm Gen block）处于关键路径上？

4.15 从 4.4 节可知，ld 是 CPU 中延迟最长的一条指令。如果修改 ld 和 sd 的功能，去掉地址偏移，比如 ld 或 sd 指令的访存地址只能使用计算后存放于 rs1 中的数值，那么就不会有指令同时使用 ALU 和数据存储，这将缩短时钟周期。但是这也会增多指令数目，因为很多 ld 和 sd 指令将会被 ld/add 或者 sd/add 组合取代。

4.15.1 [5]<4.4>修改后的时钟周期是多少？

4.15.2 [10]<4.4>在这款新 CPU 上运行如习题 4.7 中的指令组合，将会变快还是变慢？大概变化多少？为简化起见，假设每条 ld 和 sd 指令将会被两条指令的组合取代。

4.15.3 [5]<4.4>在新款 CPU 上影响程序运行速度变快或者变慢的主要因素是什么？

4.15.4 [5]<4.4>参考图 4-21，你认为原先的 CPU 是一款更好的设计，还是新款 CPU 是一款更好的设计？为什么？

4.16 在本题中将讨论流水线如何影响处理器的时钟周期。假设数据通路的各个流水级的延迟如下：

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

同时，假设处理器执行的指令分布如下：

ALU/Logic	Jump/Branch	Load	Store
45%	20%	20%	15%

4.16.1 [5]<4.5>在流水化和非流水的处理器中，时钟周期分别是多少？

4.16.2 [10]<4.5>在流水化和非流水的处理器中，对于 ld 指令的延迟分别是多少？

4.16.3 [10]<4.5>如果我们将数据通路中的一个流水级拆成两个新流水级，每一个新流水级的延迟是原来的一半，那么我们将拆分哪一级？新处理器的时钟周期是多少？

4.16.4 [10]<4.5>假设没有停顿或冒险，数据存储的利用率如何？

4.16.5 [10]<4.5>假设没有停顿或冒险，寄存器堆的写口利用率如何？

4.17 [10]<4.5>在一个 k 级流水的 CPU 上执行 n 条指令，最少需要多少周期？证明你的公式。

4.18 [5]<4.5>假设初始化寄存器 x11 为 11，x12 为 22，如果在 4.5 节中的流水线结构上执行下述代码，寄存器 x13 和 x14 中最终为何值？注：硬件不处理数据冒险，编程者需要在必要处插入 NOP 指令来解决数据冒险。

```
addi   x11, x12, 5
add    x13, x11, x12
addi   x14, x11, 15
```

4.19 [10]<4.5>假设 x11 初始化为 11，x12 初始化为 22，如果在 4.5 节中的流水线结构上执行下述代码，寄存器 x15 中最终为何值？注：硬件不处理数据冒险，编程者需要在必要处插入 NOP 指令来解决数据冒险。假设寄存器堆先写后读，因此 ID 阶段的指令可以在同一周期内得

到 WB 阶段的数据，具体见 4.7 节和图 4-51。

```
addi   x11, x12, 5
add    x13, x11, x12
addi   x14, x11, 15
add    x15, x11, x11
```

4.20 [5]<4.5>在下述代码中添加 NOP 指令，让它无须处理数据冒险也能正确运行在流水线处理器上。

```
addi   x11, x12, 5
add    x13, x11, x12
addi   x14, x11, 15
add    x15, x13, x12
```

4.21 考虑如 4.5 节中的流水线结构，硬件不处理数据冒险（比如，由编程人员负责发现数据冒险并在必要处插入 NOP 指令）。假设优化后，n 条指令的典型程序需要额外添加 $4n$ 条 NOP 指令才能正确处理数据冒险。

4.21.1 [5]<4.5>假设这条没有前递通路的流水线结构的时钟周期为 250ps，假设添加前递硬件将 NOP 指令的数量从 $4n$ 减少到 $0.05n$，但是时钟周期要增加到 300ps。相比之下，这条新流水线的加速比是多少？

4.21.2 [10]<4.5>不同的程序所需的 NOP 数量是不同的。在不影响程序性能的条件下，在带有前递硬件的流水线结构中执行程序，程序中的 NOP 指令比例为多少？

4.21.3 [10]<4.5>重复上题。使用 x 表示相对于指令总数 n 来说 NOP 指令的数量。比如，在 4.21.1 中，x 为 4。答案使用 x 来表示。

4.21.4 [10]<4.5>在具有前递硬件的流水线结构中，仅有 $0.075n$ 条 NOP 指令的程序会运行更快吗？

4.21.5 [10]<4.5>如果想要在具有前递硬件的流水线结构上更快地运行，最少需要插入多少 NOP 指令（占代码数量的比例）？

4.22 [5]<4.5>对于如下的 RISC-V 的汇编片段：

```
sd    x29, 12(x16)
ld    x29, 8(x16)
sub   x17, x15, x14
beqz  x17, label
add   x15, x11, x14
sub   x15, x30, x14
```

4.22.1 [5]<4.5>请画出流水线图，说明以上代码会在何处停顿。

4.22.2 [5]<4.5>是否可通过重排代码来减少因结构冒险而导致停顿的次数？

4.22.3 [5]<4.5>该结构冒险必须用硬件来解决吗？我们可以通过在代码中插入 NOP 指令来消除数据冒险，对于结构冒险是否可以相同处理？如果可以，请解释原因。否则，也请解释原因。

4.22.4 [5]<4.5>在典型程序中，大约需要为该结构冒险产生多少个周期的停顿？（使用习题 4.8 中的指令分布）。

4.23 如果我们改变 load/store 指令格式，使用寄存器（不需要立即数偏移）作为访存地址，这些指令就不再需要使用 ALU（具体见习题 4.15）。这样的话，MEM 阶段和 EX 阶段就可以重叠，流水级数变为四级。

4.23.1 [10]<4.5>流水线级数的减少会影响时钟周期吗？

4.23.2 [10]<4.5>这样的变化会提高流水线的性能吗？

4.23.3 [10]<4.5>这样的变化会降低流水线的性能吗？

4.24 [10]<4.7>下面两个流水线图中，哪一个更好地描述了流水线冒险检测单元的操作？为什么？

选择1：

```
ld x11, 0(x12):     IF ID EX ME WB
add x13, x11, x14:    IF ID EX..ME WB
or x15, x16, x17:      IF ID..EX ME WB
```

选择2：

```
ld x11, 0(x12):    IF ID EX ME WB
add x13, x11, x14:  IF ID..EX ME WB
or x15, x16, x17:     IF..ID EX ME WB
```

4.25 考虑如下循环：

```
LOOP: ld    x10, 0(x13)
      ld    x11, 8(x13)
      add   x12, x10, x11
      subi  x13, x13, 16
      bnez  x12, LOOP
```

如果使用完美的分支预测（即没有控制冒险带来的流水线停顿），流水线中没有使用延迟槽，采用硬件前递解决数据冒险，分支指令在EX阶段判断是否跳转。

4.25.1 [10]<4.7>给出该循环中前两次循环的流水线执行图。

4.25.2 [10]<4.7>标注出没有进行有用操作的流水级。当流水线全负荷工作时，所有五个流水级都在进行有用操作的情况多久会出现一次？（从subi指令进入IF阶段开始计算，到bnez指令进入IF阶段结束。）

4.26 本题尝试帮助大家了解，对于带前递的流水线处理器，如何在成本/复杂度/性能间进行折中。关于本题可参考图4-53中的流水线数据通路。假设在该处理器上执行的指令片段存在一种RAW数据相关。这种RAW数据相关存在于产生结果的流水级（例如EX或MEM）和使用该结果的后续指令之间（例如第一条指令紧跟产生结果的指令，第二条也是如此，或者跟前两条都相关）。假设在时钟的前半周期完成寄存器的写操作，在后半周期完成寄存器的读操作。因此“EX阶段到第三条指令”以及“MEM阶段到第三条指令”之间的数据相关都不会引起数据冒险。假设分支指令是在EX阶段判断是否发生跳转。如果流水线不存在数据冒险，则CPI为1。

EX to 1st Only	MEM to 1st Only	EX to 2nd Only	MEM to 2nd Only	EX to 1st and EX to 2nd
5%	20%	5%	10%	10%

假设单个流水级的延迟如下。对于EX阶段，分为没有前递支持和带有各种类型的前递支持。

IF	ID	EX (no FW)	EX (full FW)	EX (FW from EX/MEM only)	EX (FW from MEM/WB only)	MEM	WB
120ps	100ps	110ps	130ps	120ps	120ps	120ps	100ps

4.26.1 [5]<4.7>对于以上的每种RAW相关，写出一段包含至少三条汇编指令的代码。

4.26.2 [5]<4.7>对于上述每种RAW相关，给出的代码中需要插入多少条NOP指令，才能保证在没有前递支持或冒险检测的流水线数据通路上执行正确？说明NOP指令插入的位置。

4.26.3 [10]<4.7>独立分析单条指令可能会让插入的NOP指令数过量。写一段只有三条汇编指令的

程序，满足：独立分析单条指令时得到的流水线停顿次数大于流水线为避免数据冒险而真实停顿的次数。

4.26.4 [10] <4.7> 假设没有其他冒险，上表中描述的程序运行在没有前递支持的流水线中，CPI 是多少？停顿周期占比多少？（为了简化起见，假设所有必须考虑的情况都被列在上表中。）

4.26.5 [5] <4.7> 如果使用支持完全前递的流水线（前递所有需要前递的结果），CPI 是多少？停顿周期占比多少？

4.26.6 [10] <4.7> 假设无法提供三输入的多选器来实现完全前递，因此需要判断是否只前递 EX/MEM 寄存器（一周期前递）或只前递 MEM/WB 寄存器（两周期前递）。对每种方案计算 CPI。

4.26.7 [5] <4.7> 给定冒险出现的概率以及流水线的各级延迟，相比没有前递支持的流水线，添加了每种前递类型（EX/MEM，MEM/WB，完全前递）的流水线获得的加速比是多少？

4.26.8 [5] <4.7> 如果我们能够添加‘时空穿越’前递逻辑消除所有的数据冒险，相比 4.26.7 中的最快的处理器，提高的加速比是多少？假设相对于完全前递的 EX 阶段，这种还未被发明的“时空穿越”前递电路需要增加 100ps 的延迟。

4.26.9 [5] <4.7> 在冒险类型表中，将“EX 到第一条指令”和“EX 到第一条指令以及 EX 到第二条指令”分成两类，为何表中没有“MEM 到第一条指令和 MEM 到第二条指令”这个类型呢？

4.27 讨论下述指令序列，假设在一个五级流水的数据通路中执行：

```
add  x15, x12, x11
ld   x13, 4(x15)
ld   x12, 0(x2)
or   x13, x15, x13
sd   x13, 0(x15)
```

4.27.1 [5] <4.7> 如果没有前递逻辑或者冒险检测支持，请插入 NOP 指令保证程序正确执行。

4.27.2 [10] <4.7> 对代码进行重排，插入最少的 NOP 指令。假设寄存器 x17 可用来做临时寄存器。

4.27.3 [10] <4.7> 如果处理器中支持前递，但未实现冒险检测单元，上述代码段的执行将会发生什么？

4.27.4 [20] <4.7> 以图 4-59 中的冒险检测和前递单元为例，如果执行上述代码，在前 7 个时钟周期中，每周期哪些信号会被它们置为有效？

4.27.5 [10] <4.7> 如果没有前递单元，以图 4-59 中的冒险检测逻辑为例，需要为其增加哪些输入和输出信号？

4.27.6 [20] <4.7> 如果使用习题 4.26.5 中的冒险检测单元，执行上述代码，在前 5 个时钟周期中，每个周期哪些输出信号会有效？

4.28 分支预测器的重要性与条件分支指令执行频率有关，它与分支预测正确率共同决定了分支预测错误时带来的性能损失。本题中，假设在动态执行指令中，各类型指令比例如下表：

R-type	beqz/bnez	jal	ld	sd
40%	25%	5%	25%	5%

同时，假设各种分支预测器的正确率如下：

Always-Taken	Always-Not-Taken	2-Bit
45%	55%	85%

4.28.1 [10] <4.8> 分支预测错误带来的流水线停顿将会增大 CPI。如果使用静态预测且总是预测跳转，请计算由于分支预测错误带来的额外 CPI 增加。假设：是否跳转在 ID 阶段进行判断，但

在 EX 阶段被使用，因此不会产生数据冒险，同时不使用延迟槽。

4.28.2 [10] <4.8> 如果使用静态预测且总是预测不跳转，上题如何？

4.28.3 [10] <4.8> 如果使用 2 位预测器，上题如何？

4.28.4 [10] <4.8> 假设使用 2 位预测器，如果将一半的分支指令转换为 ALU 指令，将会得到多少性能提升？假设无论是否预测正确，所有的分支指令替换成 ALU 指令的机会均等。

4.28.5 [10] <4.8> 假设使用 2 位预测器，如果将一半的分支指令进行替换，每条分支指令替换为两条 ALU 指令，将会得到多少性能提升？假设无论是否预测正确，所有的分支指令替换成 ALU 指令的机会均等。

4.28.6 [10] <4.8> 相比之下，某些分支指令具备更高的可预测性。如果我们知道 80% 的分支指令是用于向后跳转的循环，且总能被正确预测。对于另外 20% 的分支指令，使用 2 位预测器的预测正确率为多少？

4.29 本题中将讨论不同的分支预测器的正确率，重复执行的分支模式（例如循环）为：T，NT，T，T，NT。

4.29.1 [5] <4.8> 如果采用静态预测，对于总是预测跳转或总是预测不跳转，分支预测正确率分别为多少？

4.29.2 [5] <4.8> 对于前 4 条分支指令采用 2 位预测器，分支预测正确率为多少？假设预测器的初始状态为图 4-61 中的左下状态（预测不跳转）。

4.29.3 [10] <4.8> 如果给定模式无限循环下去，2 位预测器的正确率是多少？

4.29.4 [30] <4.8> 如果给定模式无限循环下去，请设计一个完美的预测器。该预测器应有带一位输出的时序电路以提供预测结果（1 为跳转，0 为不跳转），输入信号只有时钟和一个指示是否为分支指令的控制信号。

4.29.5 [10] <4.8> 如果分支执行结果正好与给定例子相反，使用 4.29.4 中的预测器，分支预测正确率为多少？

4.29.6 [20] <4.8> 重复 4.29.4，对于以上两种分支执行结果，新设计的预测器从完美预测开始（即允许先进行一段时间的暖身，允许做出错误预测）。预测器应具有一个输入，用来提供真实的分支执行结果。提示：这个输入用来让预测器判断当前的分支模式是上述两种中的哪一种。

4.30 本题讨论例外处理程序对流水线设计的影响。本题中的前三个问题与下面两条指令有关：

指令1	指令2
beqz x11, LABEL	ld x11, 0(x12)

4.30.1 [5] <4.9> 这些指令可能会触发哪些例外？对每一种例外，说明分别在哪个流水级被检测到。

4.30.2 [5] <4.9> 如果对于每种例外都有单独的地址存放对应的处理程序，解释如何改造流水线结构使其能够处理这些例外。假设这些例外处理程序对应的地址都是已知的。

4.30.3 [10] <4.9> 如果第二条指令紧接在第一条指令之后取指，按照 4.30.1 中所列出的例外类型，假设第一条指令引发了第一种例外，流水线中会如何处理？给出流水线执行图，从第一条指令取指开始，到第一条指令引发的例外处理程序结束为止。

4.30.4 [20] <4.9> 假设对例外处理程序地址进行向量化，将其以表的形式存放在固定地址开始的数据存储中。为实现例外处理机制如何改造流水线？重复习题 4.30.3，结果如何？

4.30.5 [15] <4.9> 如果在一台只有一个固定例外处理程序入口的机器上模拟向量化例外处理（如 4.30.4 所示），写出位于固定入口处的代码段。提示：该代码段需要完成例外类型的判断，从例外向量表中获得处理程序的正确入口地址，并转向执行例外处理程序。

4.31 本题中对单发射和双发射处理器进行性能比较，并考虑针对双发射执行如何优化程序。代码如下所示（使用 C 语言）：

```
for(i=0;i!=j;i+=2)
    b[i]=a[i]-a[i+1];
```

不做任何优化的编译器产生下述 RISC-V 的汇编代码：

```
      li    x12, 0
      jal   ENT
TOP:  slli  x5, x12, 3
      add   x6, x10, x5
      ld    x7, 0(x6)
      ld    x29, 8(x6)
      sub   x30, x7, x29
      add   x31, x11, x5
      sd    x30, 0(x31)
      addi  x12, x12, 2
ENT:  bne   x12, x13, TOP
```

上述代码中使用了以下寄存器：

i	j	a	b	临时值
x12	x13	x10	x11	x5～x7，x29～x31

假设本题中的双发射、静态调度处理器具有以下特性：

1. 同时发射的指令中必须一条是访存指令，另一条是算术 / 逻辑指令或者分支指令。
2. 处理器各级之间支持所有的前递（包括处理分支指令的 ID 阶段的前递）。
3. 处理器有完美的分支预测。
4. 如果两条指令有依赖关系，它们不能同时被发射（具体见 4.10.2 节）。
5. 如果必须停顿流水线，则同一个发射包中的两条指令需要同时停顿（具体见 4.10.2 节）。

完成以下这些练习，并且所写代码能够获得接近最优的加速比，并统计所花时间。

4.31.1 [30] <4.10> 画出上述 RISC-V 代码运行在双发射处理器上的流水线图。假设该循环执行两遍后退出。

4.31.2 [10] <4.10> 相比单发射处理器，双发射处理器的加速比是多少？假设该循环执行多遍。

4.31.3 [10] <4.10> 重写上述 RISC-V 代码以在单发射处理器上获得性能提升。提示：使用指令“beqz x13, DONE”，在 j = 0 时可以避免进入循环。

4.31.4 [20] <4.10> 重写上述 RISC-V 代码以在双发射处理器上获得性能提升。但是，请不要将循环展开。

4.31.5 [30] <4.10> 使用 4.31.4 中的优化代码重复习题 4.31.1。

4.31.6 [10] <4.10> 运行习题 4.31.3 和 4.31.4 中优化后的代码，从单发射到双发射处理器，获得的加速比是多少？

4.31.7 [10] <4.10> 将 4.31.3 中的 RISC-V 代码进行循环展开，展开后的一次循环处理原始循环中的两遍。面向单发射处理器重写这段代码，以得到更高的性能。假设变量 j 是 4 的倍数。

4.31.8 [20] <4.10> 将 4.31.4 中的 RISC-V 代码循环展开，展开后的一次循环处理原始循环中的两遍。面向双发射处理器重写这段代码，以得到更高的性能。假设变量 j 是 4 的倍数。提示：对循环体进行重组，让某些计算在循环外或者在循环结束时执行。假设临时寄存器中的数值只在循环体内有效。

4.31.9 [10] <4.10> 运行习题 4.31.7 和 4.31.8 中循环展开后、优化后的代码，从单发射到双发射处理器，获得的加速比是多少?

4.31.10 [30] <4.10> 假设双发射处理器上只能同时运行两条算术 / 逻辑指令，重复习题 4.31.8 和 4.31.9。换句话说，指令包中的第一条指令可以是任意类型的指令，但是第二条指令必须是算术或者逻辑指令。两条访存指令是不能同时被调度执行的。

4.32 本题中讨论能耗有效性与性能之间的关系。假设指令存储、寄存器和数据存储的动态能耗如下表所示。假设数据通路其他部件消耗的能量可以忽略不计。其中，“寄存器读”和“寄存器写”仅针对寄存器堆而言。

I-Mem	Register Read	Register Write	D-Mem Read	D-Mem Write
140pJ	70pJ	60pJ	140pJ	120pJ

假设数据通路部件的延迟如下表所示，并假设数据通路的其他部件的延迟可以忽略不计。

I-Mem	Control	Register Read or Write	ALU	D-Mem Read or Write
200ps	150ps	90ps	90ps	250ps

4.32.1 [5] <4.3, 4.6, 4.14> 在单周期和五级流水线的处理器上执行一条 add 指令花费的能量分别是多少?

4.32.2 [10] <4.6, 4.14> 哪类 RISC-V 指令消耗能量最多？消耗的能量是多少?

4.32.3 [10] <4.6, 4.14> 以减少能量消耗为重要目标，如何更改流水线设计？更改后，在流水线上运行 ld 指令，能耗下降的比例是多少?

4.32.4 [10] <4.6, 4.14> 对于上题的流水线更改，如果运行其他指令，能耗下降的比例是多少?

4.32.5 [10] <4.6, 4.14> 习题 4.32.3 中的修改对流水线 CPU 的性能有什么影响?

4.32.6 [10] <4.6, 4.14> 如果去掉 MemRead 控制信号，每个周期都可以读数据存储，即 MemRead 恒为 1。解释如此修改后为何处理器功能仍然正确。如果 25% 的指令类型是 load 指令，这个修改在时钟频率和能耗方面会产生什么影响?

4.33 在制造硅芯片时，材料（例如硅）的缺陷和制造错误会导致电路失效。一个非常普遍的问题是一根线上的信号会对相邻线上的信号产生影响，这被称为串扰。有一类串扰问题是这样的，某些线上的信号为常值（如电源线），该线附近的线也被固定为 0（stuck-at-0）或 1（stuck-at-1）。下面习题中的缺陷发生在图 4-21 中寄存器堆的输入端“寄存器写”的第 0 位。

4.33.1 [10] <4.3, 4.4> 假设测试处理器缺陷的方式如下：先给 PC、寄存器堆、数据和指令存储器中预设数值（可以自己选择）；任意执行一条指令，然后读出 PC、寄存器堆和存储器中的值；最后检查这些值以判断处理器中是否存在缺陷。能否设计一个方案，用来检查该信号上是否有“固定为 0”缺陷吗?

4.33.2 [10] <4.3, 4.4> 重复习题 4.33.1，本次检查“固定为 1”缺陷。能否设计一个测试方案，同时检查这两个缺陷？如果可以，请解释如何实现；如果不能，请说明理由。

4.33.3 [10] <4.3, 4.4> 如果我们知道处理器的某个信号具有“固定为 1”缺陷，该处理器还能用吗？如果可用，需要将在正常 RISC-V 处理器上运行的程序转换成在本处理器上可以运行的程序。假设有足够的空闲存储，可以存放更长的程序和额外的数据。

4.33.4 [10] <4.3, 4.4> 重做习题 4.33.1，本次检测控制信号 MemRead 是否存在如下缺陷：如果 branch 为 0 时，MemRead 信号为 0，则有缺陷，否则无缺陷。

4.33.5 [10] <4.3, 4.4> 重做习题 4.33.1，本次检测控制信号 MemRead 是否存在如下缺陷：如果

RegRd 为 1 时，MemRead 信号为 1，则有缺陷，否则无缺陷。提示：本问题需要操作系统知识，需考虑是否会引发段错误（segmentation fault）。

自我检测答案

4.1 节 控制，数据通路，存储；缺少输入和输出。

4.2 节 错。边沿触发的寄存器让同时的读和写变为可能和清晰的。

4.3 节 Ⅰ a；Ⅱ c。

4.4 节 是的，信号 Branch 和 ALUOp0 是相同的。除此之外，可以通过对无关位赋值来灵活地组合其他信号。例如，将 MemtoReg 信号的两个无关位置为 1 和 0，信号 MemtoReg 和 ALUSrc 的赋值是相同的。将 MemtoReg 的无关位置为 1，则信号 ALUOp1 和 MemtoReg 的赋值正好相反。这样，就可以不需要额外的反相器，只需要简单地使用一个其他信号，并翻转一下 MemtoReg 多选器的输入顺序即可。

4.5 节 1. 由于 ld 指令存在 load-use 数据冒险，因此停顿流水线。2. 由于 x11 寄存器存在 RAW 相关造成的数据冒险，可前递 add 指令的结果，以避免在第三条指令发生停顿 3. 不需要停顿，即使没有前递支持。

4.6 节 第 2 句和第 4 句是正确的，剩下的是不正确的。

4.8 节 1. 预测不发生；2. 预测发生；3. 动态预测。

4.9 节 第一条语句，因为逻辑上它会在其他指令之前执行。

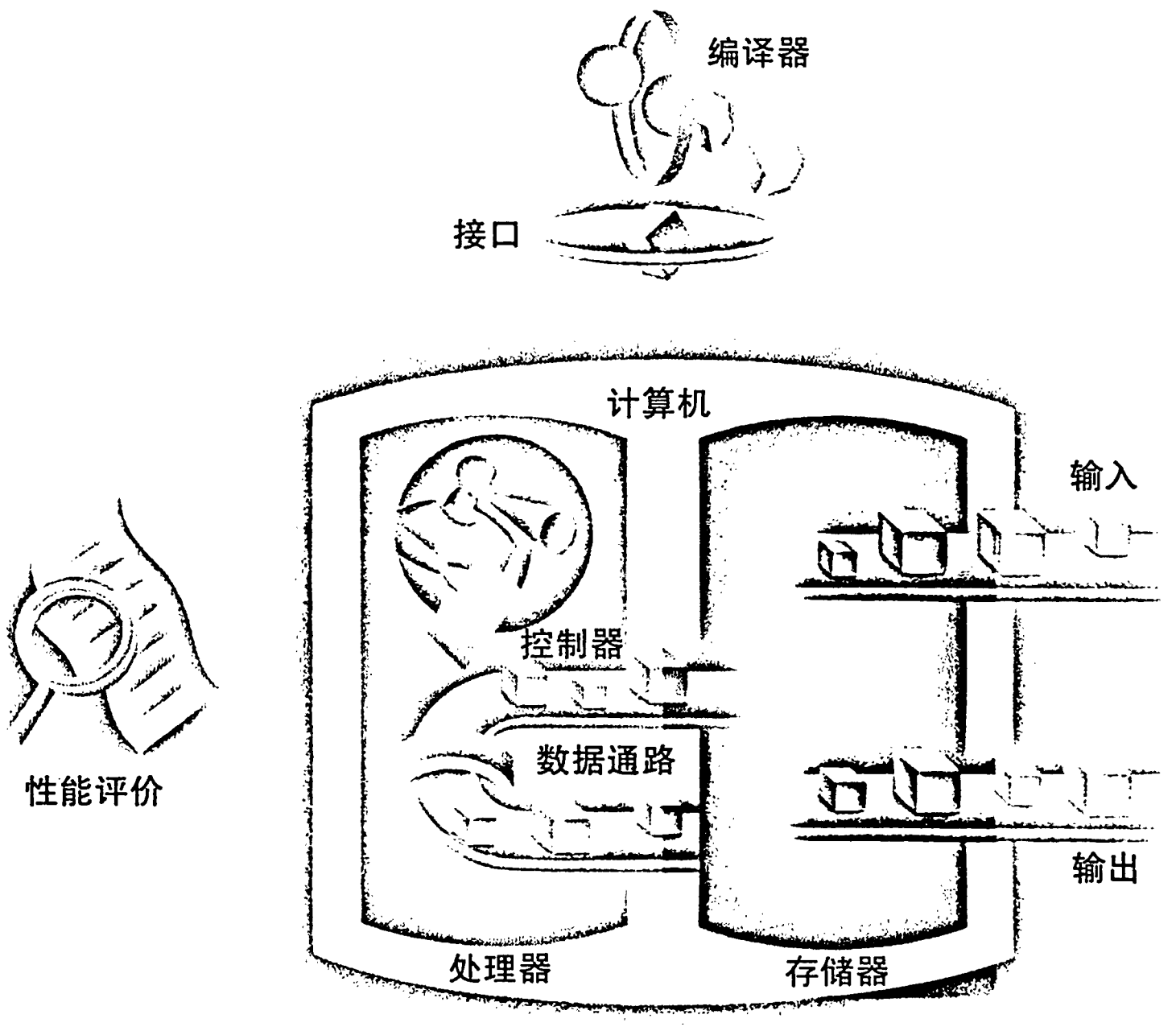
4.10 节 1. 都有，2. 都有，3. 软件，4. 硬件，5. 硬件，6. 硬件，7. 都有，8. 硬件，9. 都有。

4.12 节 前两个是错的，后两个是对的。

第 5 章

Computer Organization and Design: The Hardware/Software Interface, RISC-V Edition

大而快：层次化存储



计算机的五个经典部件

5.1 引言

从最早的计算开始，编程者就希望能有无限大的高速存储。本章主要探讨如何帮助程序员构造一个容量无限大的存储器。开始之前，让我们做一个简单的类比，来说明使用的基本概念和关键技术。

假设你是一个学生，正在写关于计算机硬件重要发展历史的毕业论文。你坐在图书馆的书桌前，桌上堆满了从书架上抽出来的各种书籍。你发现许多你要写到的重要计算机都能从书中找到，但是唯独没有 EDSAC。因此，你回到书架前开始寻找。你找到了一本记录英国早期计算机的书籍，里面涉及 EDSAC。如果之前摆在你书桌上的那堆书是经过精心挑选的，那么很多你所需要的资料都能从中找到，这样你会把大量的时间花费在阅读上，而不是返回到书架前寻找。相比于书桌上只放一本书然后频繁地在书架和书桌之间往返，在书桌上放置更多的书籍显然会节省时间。

> 理想中，我们都希望有无限大的存储容量，任何特别的数据都能够立即获得……我们不得不认识到构建层次化存储的可能性。在这样的结构中，相比于上一级存储，（每级存储）具有更大的容量，但访问速度却会更慢。
> *A. W. Burks、H. H. Goldstine 和 J. von Neumann，浅谈电子计算仪器的逻辑设计，1946*

按照这样的原理，我们也可以建立一个大容量的存储，并且其访问速度和小容量存储一样快。就像你不需要同时等概率地阅读图书馆中的每一本书一样，程序也不会同时等概率地访问每一段代码或数据。否则，就不可能建立一个容量又大、访问速度又快的存储，就像不

可能把图书馆中的所有书搬到你的书桌上还想快速检索到任何你想要的资料一样。

不管是在图书馆中工作，还是执行计算机程序，都存在*局部性原理*（principle of locality）。局部性原理表明，在任意一段时间内程序都只会访问地址空间中相对较小的一部分内容，就如你只会查阅图书馆的一部分藏书一样。局部性有两种：

- **时间局部性**（temporal locality）：如果某个数据项被访问，那么在不久的将来它可能再次被访问。就如你最近刚借阅了某本书，那么很可能你很快需要再次借阅它。
- **空间局部性**（spatial locality）：如果某个数据项被访问，与它地址相邻的数据项可能很快也将被访问。例如，当你借阅了关于英国早期计算机的书籍来学习 EDSAC，你可能还会注意到书架上挨着它的还有另一本关于早期工业计算机的书籍，因此你也会将这本书借回来并发现其中有不少有用的资料。图书馆将相同主题的书籍放在相同的书架上，这增加了空间局部性。本章稍后将会讨论层次化存储如何利用程序的空间局部性。

时间局部性：该原理表明如果某个数据项被访问，那么在不久的将来它可能再次被访问。

空间局部性：该原理表明如果某个数据项被访问，与它地址相邻的数据项可能很快也将被访问。

正如查阅书桌上的书籍具有天然的局部性，程序的局部性就体现在简单、自然的程序结构中。例如，大多数程序都有循环结构，因而指令和数据很可能会被重复访问，这显示出很明显的时间局部性。指令一般都是顺序访问，因而也显示出很明显的空间局部性。数据访问本质上具有很强的空间局部性。例如，顺序访问数组元素或者结构体，这本身就具有很明显的空间局部性。

我们可以利用局部性原理来构建计算机的存储系统，也称为**存储层次结构**（memory hierarchy）。存储层次结构包括不同速度和容量的多级存储。存储速度越快，价格越昂贵，但容量越小。

存储层次结构：多级存储采用的结构，即与处理器的距离越远，存储的容量越大，但访问速度越慢。

图 5-1 中，速度越快的存储越靠近处理器；越慢的存储成本越低，离处理器越远。这样做的目的是以最低的价格为用户提供最大容量的存储，同时访问速度与最快的存储相当。

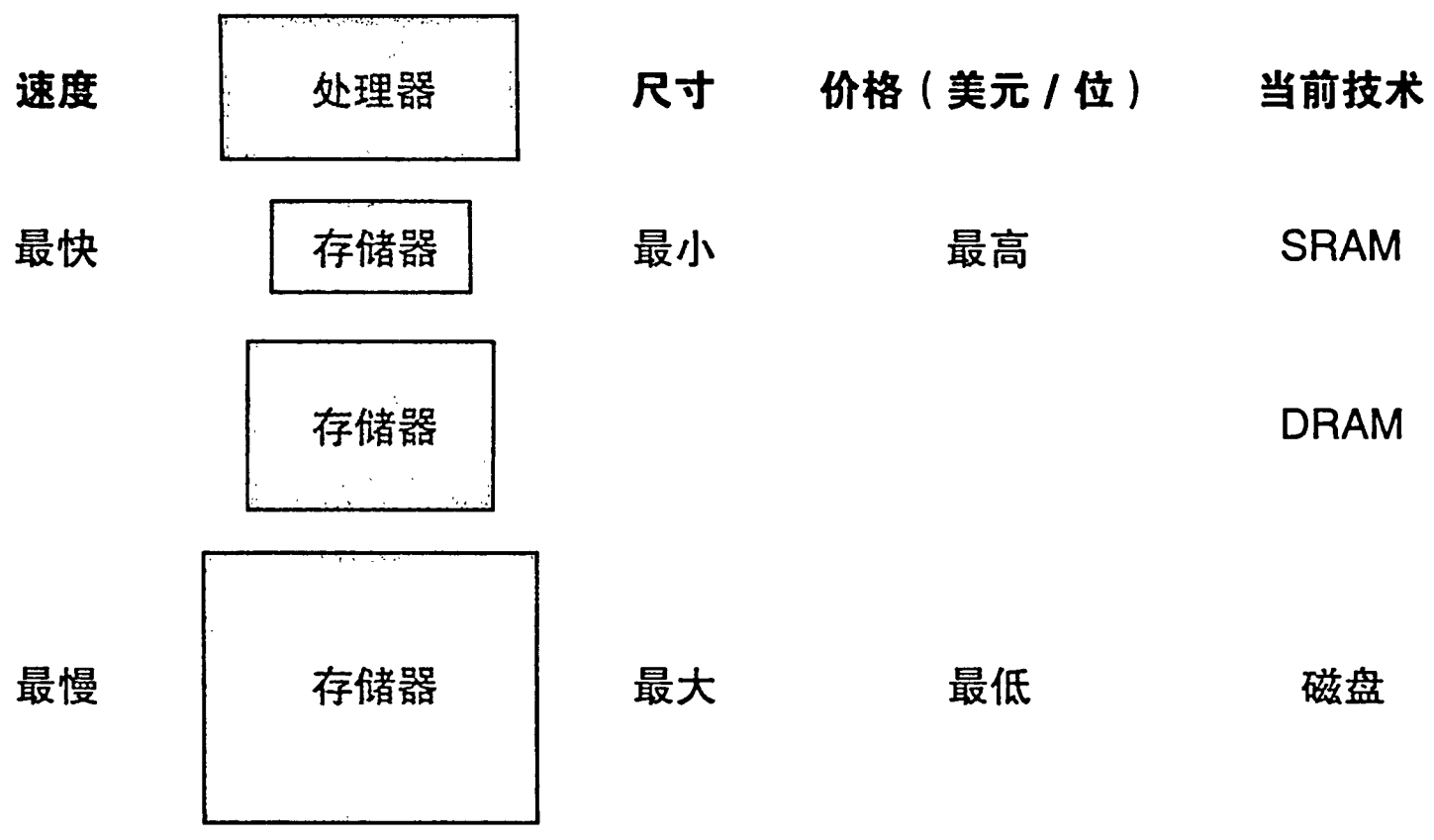


图 5-1　层次化存储的基本结构。通过实现存储系统的层次化，用户对其有了如下认识：它的容量和最下层存储一样大，访问速度和最快的那层存储相当。在许多个人移动设备中闪存（flash memory）替代了磁盘，同时可以作为台式计算机和服务器的一个新的存储层次。具体见 5.2 节

同样，数据也有相似的层次性：靠近处理器那一层中的数据是那些较远层次中数据的子集，所有的数据都被存在最远的那一层。依然使用图书馆的例子进行类比，书桌上的书籍是图书馆藏书的一个子集，也是学校所有图书馆藏书的一个子集。而且，离处理器越远的存储层次访问时间也越长，就像我们在学校图书馆系统中可能遇到的情况一样。

层次化存储可以由不同的层次组成，但是数据只能在两个相邻层次之间进行复制。因此，我们重点关注这两个层次。上一层（靠近处理器的层次）比下一层容量要小，速度要快，因为上一层存储使用了成本更高的工艺。如图 5-2 所示，在相邻两层之间进行信息交换的最小单元称为**块或行**（block 或 line）。在上述图书馆的类比中，最小的信息块就是一本书。

块或行：在缓存中存储信息的最小单位。

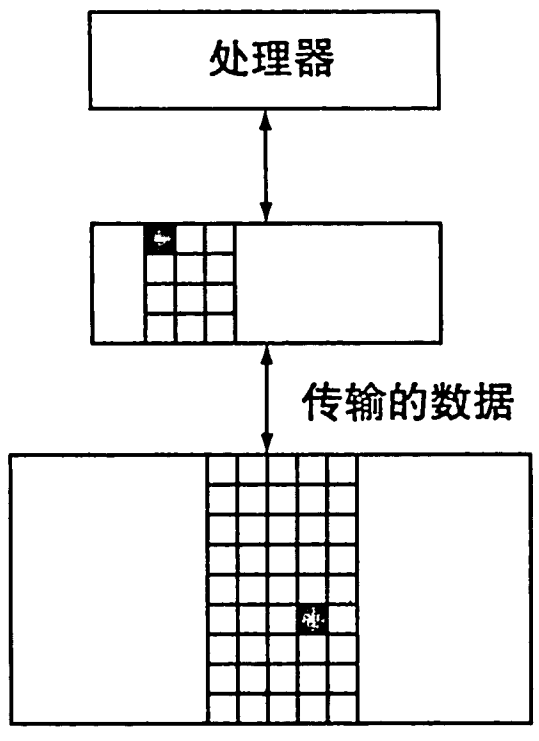


图 5-2　层次化存储中每对层次可被看作上层和下层。在每层中，信息存储的最小单位称为块或行（block 或 line）。通常，当我们在相邻层次之间进行拷贝时，都是传输一个完整的块

如果处理器所需的数据在本层的存储中找到，称为命中（hit），类比于你在桌上的某本书中找到了所需信息。如果没在本层存储中找到所需数据，称为失效（miss），将访问下一级存储。可类比于你从书桌来到书架前，寻找想要的书籍。**命中率**（hit rate 或 hit ratio）指的是在访问本层存储时命中的次数占总次数的比例，通常作为存储层次结构的性能衡量指标之一。**失效率**（miss rate，即 1−hit rate）指的是在访问本层存储时失效的次数占总次数的比例。

命中率：在访问某个存储层次时命中的次数占总访问次数的比例。

失效率：在访问某个存储层次时失效的次数占总访问次数的比例。

由于性能是引入层次化存储的主要原因，数据命中和失效的处理时间是非常重要的。**命中时间**（hit time）是访问本层存储的时间，包括判断访问命中或失效的时间（即查阅桌上所有书籍的时间）。**失效损失**（miss penalty）指的是将相应的块从下层存储替换到上层存储中的时间，加上将该数据块返回给处理器的时间（即从书架上获得另一本书并将它放在书桌上的时间）。由于上层存储容量更小，并由速度更快的存储组成，命中时间也比下层存储的访问时间短，而访问下层存储的时间是失效损失的重要组成部分（即查阅书桌上的书的时间比走到书架前获得一本有用新书的时间要少得多）。

命中时间：访问某个存储层次所需的时间，包括判断命中或失效的时间。

失效损失：将数据块从下层存储复制至某层所需的时间，包括数据块的访问时间、传输时间、写入目标层的时间和将数据块返回给请求者的时间。

正如我们将在本章中所学，用来建立存储系统的很多概念对计算机的其他方面也产生了影响，包括操作系统如何管理存储和输入

输出，编译器如何产生代码，甚至应用程序如何使用计算机等。当然，由于所有的程序都会在访存上花费大量时间，存储系统成为性能的主要决定因素。利用存储系统获得性能上的提升，这意味着在过去程序员可以把存储器看成一个线性的随机访问存储设备，而现在必须充分理解存储器的层次结构才能获得良好的性能。例如在图 5-18 以及 5.14 节中，给出了如何使矩阵乘法的性能加倍。

由于存储系统对于性能非常关键，计算机设计者为此投入了大量的精力，开发了很多复杂的技术来改善存储系统的性能。本章中，我们将讨论主要的概念性观点。为了保证内容的长度和复杂度可控，对其进行了简化和抽象。

重点 程序中同时存在时间局部性和空间局部性。前者指复用最近刚访问过的数据的趋势，后者指访问与最近被访问过的数据项地址空间相近的数据项的趋势。层次化存储将最近刚访问过的数据留在离处理器更近的存储层次中，以此来充分利用程序的时间局部性；将包含了多个连续字的数据块移动至更上层存储，以此来充分利用程序的空间局部性。

图 5-3 说明，层次化存储结构在靠近处理器的位置采用更小更快的存储技术。因此，如果命中最上层存储，访问速度会很快。如果失效，将会访问下一级存储。虽然容量变大，但访问速度变慢。如果命中率足够高，该层次化存储有效的访问时间将非常接近最上层存储（也是速度最快的存储），容量将相当于最下层存储（也是容量最大的存储）。

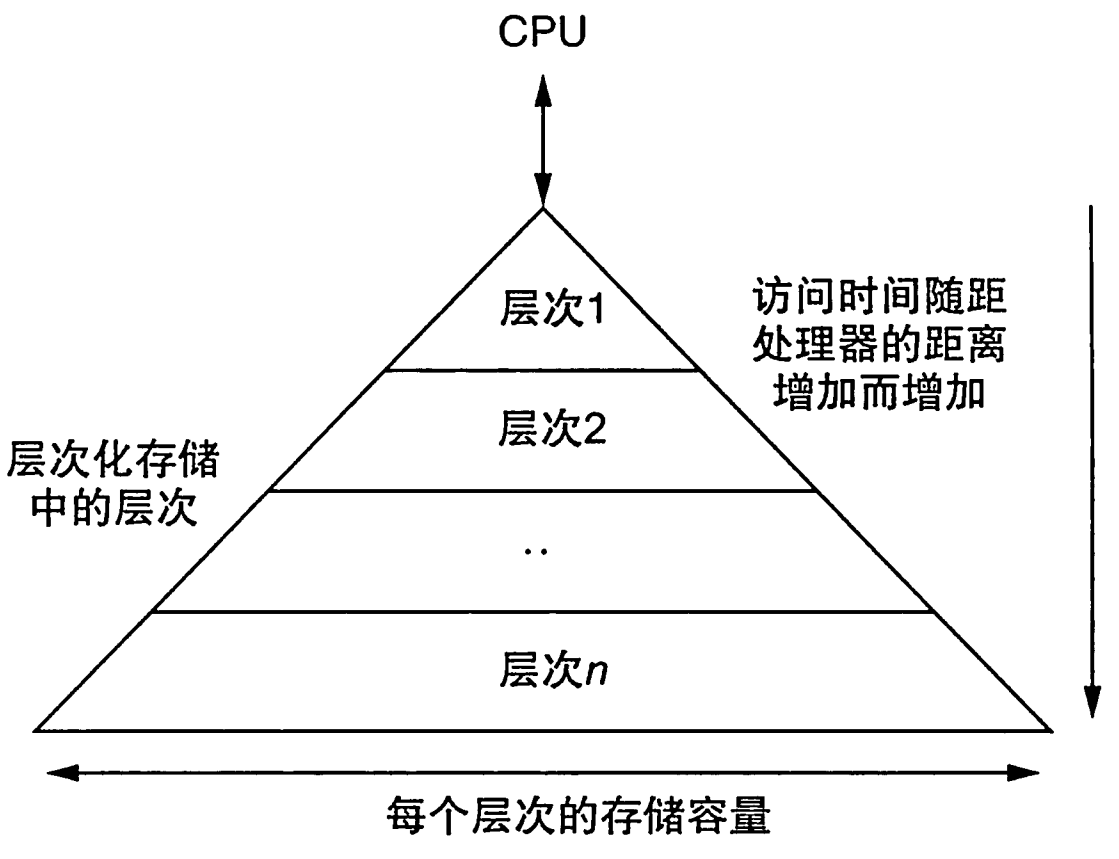


图 5-3 本图显示了层次化存储的结构：与处理器的距离增加，存储容量也随之增加。配以合适的操作机制，该结构可以使得处理器拥有层次 1 的访问速度，同时还拥有层次 n 的存储容量。本章的主题就是如何管理这样的结构。尽管本地磁盘通常位于层次结构的底端，但仍有一些系统使用磁带或者局域网内的文件服务器作为再下一层的存储

在大多数系统中，层次化存储是真实存在的。这意味着数据只有先在第 i+1 层出现，才能在第 i 层出现。

自我检测 下面哪些表述通常是正确的？

1. 存储层次结构利用了时间局部性。
2. 在一次读操作中，返回的数值取决于哪些块在缓存（cache）中。
3. 存储层次结构的大部分开销都在最上层。
4. 存储层次结构的大部分容量都在最下层。

5.2　存储技术

在当今的层次化存储中有四种主要技术。主存采用动态随机访问存储（Dynamic Random Access Memory，DRAM）器件实现，靠近处理器的存储层次使用静态随机访问存储（Static Random Access Memory，SRAM）器件实现。DRAM 的访问速度慢于 SRAM，但它的每比特成本（cost per bit）也要低很多。两者的价格差主要源于 DRAM 的每比特占用面积远远小于 SRAM，因而 DRAM 能在等量的硅上实现更大的存储容量。两者的速度差源于许多因素，主要在附录 A 的 A.9 节进行介绍。第三种技术称为闪存（flash memory）。这种非易失性存储一般作为个人移动设备的二级存储。第四种技术是磁盘（magnetic disk），用来实现服务器上最大也是最慢的存储层次。在这些技术中，每比特的价格和访问时间差别很大，如下表所示，表中使用的是 2012 年的典型数据。下面我们一一介绍这些存储技术。

存储技术	典型访问时间	单位成本（美元/GB，2012年）
SRAM半导体存储	0.5～2.5ns	500～1000
DRAM半导体存储	50～70ns	10～20
闪存半导体存储	5 000～50 000ns	0.75～1.00
磁盘	5 000 000～20 000 000ns	0.05～0.10

5.2.1　SRAM 存储技术

SRAM 存储是一种存储阵列结构的简单集成电路，通常有一个读写端口。虽然读写操作的访问时间不同，但对于任意位置的数据，SRAM 的访问时间是固定的。

SRAM 不需要刷新电路，所以访问时间可以和处理器的时钟周期接近。为防止读操作时信息丢失，典型的 SRAM 每比特采用 6 个或 8 个晶体管来实现。在待机模式下，SRAM 只需要最小的功率来保持电荷。

过去，大多数个人电脑和服务器使用独立的 SRAM 芯片作为一级、二级甚至三级高速缓存（cache）。如今，感谢摩尔定律，所有的高速缓存都被集成到了处理器芯片上，因此独立 SRAM 芯片的市场已经消失。

5.2.2　DRAM 存储技术

在 SRAM 中，只要提供电源，数值会被一直保存。而在 DRAM 中，使用电容保存电荷的方式来存储数据。采用单个晶体管来访问存储的电荷，或者读取它，或者改写它。DRAM 的每个比特仅使用单个晶体管来存储数据，它比 SRAM 的密度更高，每比特价格更低廉。由于 DRAM 在单个晶体管上存储电荷，因此不能长久保持数据，必须进行周期性的刷新。与 SRAM 相比，这也是该结构被称为动态的原因。

为了刷新数据单元，我们只需要读取其中的内容并再次写回，DRAM 中的电荷可以保持几微秒。如果每一比特数据都从 DRAM 中读出再被一一写回，就必须不停进行刷新操作，那么就会没有时间进行正常的数据访问。幸运的是，DRAM 使用两级译码电路，这使我们可以使用一个读周期紧跟一个写周期的方式一次性完成整行刷新（同一行上的数据共享一条字线）。

图 5-4 中给出 DRAM 的内部结构，图 5-5 给出 DRAM 的密度、成本和访问时间的变化趋势。