**姓名： 胡云飞 学号： 2021150047 得分：**

1. 一个程序在一台计算机上运行时需要100秒，其中80秒的时间用于乘法操作，通过将乘法操作的速度改进到只需16秒，从而把程序的运行速度提高到5倍。这里改进性能所使用到的是哪个伟大设计思想 D 。

A.通过预测提高性能 B.通过流水线提高性能

C.加速大概率事件 D.通过并行提高性能

1. 对某一芯片只提高工作电压，则其功耗 A ：

A.提高； B.下降； C.不确定； D.保持不变；

1. 下面的图表代表的是哪条MIPS指令? **（ C ）**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| op | Rs | rt | rd | shamt | funct |
| 0 | 8 | 9 | 10 | 0 | 34 |

A. sub $t0, $t1, $t2 B. add $t2, $t0, $t1

C. sub $t2, $t1, $t0 D. sub $t2, $t0, $t1

1. 假设$s1中的值是0xD0000000,给定下列$s0的值,执行下列指令是否会产生溢出？

**（ A ）**（1）$s0=0x70000000, 执行add $s0,$s0,$s1

A.会溢出 B.不会溢出

**（ B ）**（2）$s0=0x80000000, 执行sub $s0,$s0,$s1

A.会溢出 B.不会溢出

**（ B ）**（3）$s0=0x7FFFFFFF, 执行sub $s0,$s0,$s1

A.会溢出 B.不会溢出

1. 假设$t0=0xBEADFEED, $t1= 0xDEADFADE. 求执行下面指令后寄存器$t2的值。
2. sll $t2, $t0, 4

or $t2, $t2, $t1 则$t2= **（ 0xFEFFFEDF ）**

（2） sll $t2, $t0, 4

andi $t2, $t2, -1 则$t2= **（ 0xEADFEED0 ）**

（3） srl $t2, $t0, 3

andi $t2, $t2, 0xFFEF 则$t2= **（ 0x15D5BDDA ）**

1. 如果要将乘法指令结果的高32位保存在$t1，低32位保存在$t2中，需要使用两条指令来完成，它们是  **mfhi $t1;** **mflo $t2;**
2. 为了调用函数myfunc1，应该使用指令 **jal myfunc1 ，**函数返回时应该使用指令 **jr $ra** 返回到调用函数处的下一跳指令。
3. 如果指令“beq $t0,$t1, 32”指令位于0x1000地址，执行该指令时PC= **0x1000 ，**若$t0=16,$t1=16，则下一条被执行的指令位于 **0x1020** ：
4. 把下面的C代码翻译成MIPS代码。假设变量f,g,h,i和j分别赋值给寄存器$s0,$s1,$s2,$s3和$s4。假设数组A和B的基地址分别存放在$s6和$s7中。假设数组A和B中的元素均为4字节的字：

1）B[8]=A[i]+A[j];

2）f=g-A[B[4]];

(1) # 将i的值加载到$t0中

add $t0, $zero, $s3

# 将j的值加载到$t1中

add $t1, $zero, $s4

# 计算A[i]的地址并将结果存储在$t2中

sll $t2, $t0, 2 # A中每个元素占4字节，因此需要乘以4

add $t2, $t2, $s6 # 加上A的基地址

# 计算A[j]的地址并将结果存储在$t3中

sll $t3, $t1, 2 # A中每个元素占4字节，因此需要乘以4

add $t3, $t3, $s6 # 加上A的基地址

# 加载A[i]的值到$t4中

lw $t4, 0($t2)

# 加载A[j]的值到$t5中

lw $t5, 0($t3)

# 计算A[i] + A[j]的结果并将结果存储在$t6中

add $t6, $t4, $t5

# 将结果存储在B[8]中

sll $t7, $s7, 2 # B中每个元素占4字节，因此需要乘以4

add $t7, $t7, $s7 # 加上B的基地址

addi $t7, $t7, 32 # 加上偏移量32（8 \* 4）

sw $t6, 0($t7)

(2) # 将B[4]的地址加载到$t0中

sll $t0, $s7, 2 # B中每个元素占4字节，因此需要乘以4

add $t0, $t0, $s7 # 加上B的基地址

addi $t0, $t0, 16 # 加上偏移量16（4 \* 4）

# 加载A[B[4]]的值到$t1中

lw $t1, 0($t0)

# 将g的值加载到$t2中

add $t2, $zero, $s1

# 计算g - A[B[4]]的结果并将结果存储在$f中

sub.s $f0, $t2, $t1

1. 将以下C语句转换为MIPS汇编指令序列，假设变量a, b, i, j分别对应寄存器$s0, $s1, $t0, $t1, $s2保存着数组D的起始地址。

a）for ( i=0; i<10; i++)

a += b;

b) while (a<10) {

D[a] = b + a;

a += 1;

}

(a) # 初始化循环计数器i为0

addi $t0, $zero, 0

# 初始化累加变量a为0

add $s0, $zero, $zero

# 将b的值加载到$t1中

add $t1, $zero, $s1

# 循环开始标签

loop:

# 检查循环条件 i < 10

slti $t2, $t0, 10

beqz $t2, exit # 如果 i >= 10，则跳出循环

# a += b

add $s0, $s0, $t1

# i += 1

addi $t0, $t0, 1

# 跳转回循环开始处

j loop

# 循环结束标签

exit:

(b) # 循环开始标签

loop:

# 检查循环条件 a < 10

slti $t0, $s0, 10

beqz $t0, exit # 如果 a >= 10，则跳出循环

# 计算 D[a] = b + a

add $t2, $s1, $s0 # b + a 的结果存储在$t2中

sll $t3, $s0, 2 # D中每个元素占4字节，因此需要乘以4

add $t3, $t3, $s2 # 加上D的起始地址

sw $t2, 0($t3) # 将结果存储到D[a]

# a += 1

addi $s0, $s0, 1

# 跳转回循环开始处

j loop

# 循环结束标签

exit:

1. 把下面的MIPS代码翻译成C代码。假设变量f,g,h,i和j分别赋值给寄存器$s0,$s1,$s2,$s3和$s4，数组A和数组B的基地址分别存放在$s6和$s7中。

addi $t0,$s6,4

add $t1,$s6,$0

sw $t1,0($t0)

lw $t0,0($t0)

add $s0,$t1,$t0

int\* t0 = (int\*)($s6 + 4);

int\* t1 = (int\*)($s6 + $0);

\*t0 = \*t1;

$t0 = \*t0;

$s0 = \*t1 + \*t0;

1. 假定字变量f, g, h, i, j分别对应寄存器$s0, $s1, $s2, $s3, $s4, 并且字数组A和B的起始地址分别存放在寄存器$s6, $s7中，请分别采用MIPS汇编指令实现其功能：

a) f=g+h+B[4]; b) f=g-A[B[4]];

c) f=g+h+B[1]; d) f=A[B[g]+1];

(a) # 将B[4]的地址加载到$t0中

sll $t0, $s7, 2 # B中每个元素占4字节，因此需要乘以4

add $t0, $t0, $s7 # 加上B的基地址

addi $t0, $t0, 16 # 加上偏移量16（4 \* 4）

# 加载B[4]的值到$t0中

lw $t0, 0($t0)

# f = g + h + B[4]

add $s0, $s1, $s2 # g + h

add $s0, $s0, $t0 # + B[4]

(b) # 将B[4]的地址加载到$t0中

sll $t0, $s7, 2 # B中每个元素占4字节，因此需要乘以4

add $t0, $t0, $s7 # 加上B的基地址

addi $t0, $t0, 16 # 加上偏移量16（4 \* 4）

# 加载B[4]的值到$t0中

lw $t0, 0($t0)

# 将A[B[4]]的地址加载到$t1中

sll $t1, $t0, 2 # A中每个元素占4字节，因此需要乘以4

add $t1, $t1, $s6 # 加上A的基地址

# 加载A[B[4]]的值到$t1中

lw $t1, 0($t1)

# f = g - A[B[4]]

sub $s0, $s1, $t1

(c) # 将B[1]的地址加载到$t0中

sll $t0, $s7, 2 # B中每个元素占4字节，因此需要乘以4

add $t0, $t0, $s7 # 加上B的基地址

addi $t0, $t0, 4 # 加上偏移量4（1 \* 4）

# 加载B[1]的值到$t0中

lw $t0, 0($t0)

# f = g + h + B[1]

add $s0, $s1, $s2 # g + h

add $s0, $s0, $t0 # + B[1]

(d) # 将g的值加载到$t0中

add $t0, $zero, $s1

# 将B[g]的地址加载到$t1中

sll $t2, $t0, 2 # B中每个元素占4字节，因此需要乘以4

add $t2, $t2, $s7 # 加上B的基地址

lw $t1, 0($t2)

# A[B[g] + 1]的地址存储在$t3中

addi $t1, $t1, 1

sll $t3, $t1, 2 # A中每个元素占4字节，因此需要乘以4

add $t3, $t3, $s6 # 加上A的基地址

# 加载A[B[g] + 1]的值到$s0中

lw $s0, 0($t3)

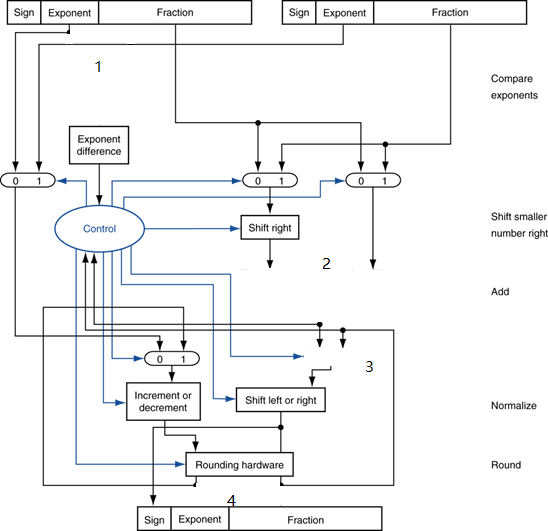
1. 请画出未经优化的乘法器结构图（含部件名称、位长、是否具有移位以及移位方向，部件间的连接），假设机器字长为4位，并填写无符号二进制数0101ⅹ1011的计算细节步骤。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **步骤** | **乘数寄存器** | **被乘数寄存器** | **乘积寄存器** |
| 0 | 初始值 | 0000 | 0101 | 0000 0000 |
| 1 | 1 | 0000 | 1011 | 0000 0000 |
| 2 | 0000 | 1011 | 0000 0000 |
| 3 | 0000 | 1011 | 0000 0000 |
| 2 | 1 | 0000 | 1011 | 0000 0000 |
| 2 | 0001 | 0110 | 0000 0000 |
| 3 | 0011 | 1011 | 0000 0000 |
| 3 | 1 | 0111 | 0110 | 0000 0000 |
| 2 | 1111 | 1101 | 0110 1101 |
| 3 |  |  |  |
| 4 | 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |

1. 请画出优化后的乘法器结构图（含部件名称、位长、是否具有移位以及移位方向，部件间的连接），假设机器字长为4位，并填写无符号二进制数0101ⅹ1011的计算细节步骤。

|  |  |  |  |
| --- | --- | --- | --- |
|  | **步骤** | **被乘数寄存器** | **乘积寄存器** |
| 0 | 初始值 | 0000 | 0101 |
| 1 | 1 | 0000 | 1011 |
| 2 | 0000 | 1011 |
| 2 | 1 | 0000 | 1011 |
| 2 | 0000 | 1011 |
| 3 | 1 | 0001 | 0110 |
| 2 | 0011 | 1011 |
| 4 | 1 | 0111 | 0110 |
| 2 | 1111 | 1101 |

1. 请补充完成下面的浮点加法器的结构图，将图中数字1~4的部位绘制完整。然后以下图所示的浮点加法硬件计算（0.5）10+（-0.4375）10，精度为4位，采用IEEE 754单精度格式表示浮点数。请完成以下工作：（1）结合图中关键部件和箭头处标明相关步骤序号并在空白处说明，重点讲明数据的变换和传输，关键部件的输入和输出。（2）相关数据转换，如十进制转换为二进制等，如对阶计算过程请在空白处详细给出。



浮点加法器结构图描述：

输入：

A（0.510）和 B（-0.437510）：待加的两个浮点数。

e1和e2：A和B的指数部分。

精度：4位。

关键部件：

二进制补码表示转换器：用于将十进制数转换为二进制补码表示。

对阶操作器：用于将指数小的数左移（或减小的指数）以使指数相等。

尾数加法器：用于将两个数的尾数相加得到结果的尾数。

舍入器：根据精度要求对尾数进行舍入。

规格化器：将结果尾数和指数调整到适当的范围。

输出：

结果的符号位（正或负）。

结果的尾数（mantissa）。

结果的指数（exponent）。

步骤：

输入数据通过二进制补码表示转换器转换为二进制补码表示。转换包括符号位、指数和尾数的转换。例如，0.510转换为二进制补码表示为0 10000001 01000000000100100111111，-0.437510转换为二进制补码表示为1 10000001 1011010001111110110101。

通过比较A和B的指数e1和e2，确定对阶操作。由于A的指数e1较小，需要将A的尾数左移若干位，使其指数与B相等。左移位数为e2 - e1。例如，如果e2 - e1 = 2，则将A的尾数左移两位。

尾数加法器将对阶后的A和B的尾数相加，得到结果的尾数。如果结果尾数超过精度范围，需要进行舍入操作。舍入方式可以根据精度要求选择不同的舍入方式，如四舍五入、向下舍入等。

规格化器将结果调整到IEEE 754单精度浮点数的范围，包括符号位、指数和尾数。如果结果尾数为零，则指数减小；否则，指数不变。规格化后的结果即为最终的浮点加法结果。

输出：最终结果的符号位、尾数和指数通过相应的接口输出。

计算结果：通过上述步骤，我们可以得到（0.5）10+（-0.4375）10的IEEE 754单精度浮点数表示为（−0.875）10，即二进制表示为 1 10000001 11010100011（由于精度限制，这里只显示了部分尾数）。

* 16. 对于虚地址13048，如果页的大小是2KB，那么请问相应的虚页号和页内偏移各自是什么？如果上述的虚页映射到12号物理页，请问其物理地址是什么？

如果页的大小是2KB，虚地址13048可以表示为：

虚地址 = 虚页号 × 页大小 + 页内偏移

根据这个关系，我们可以将虚地址13048拆分为虚页号和页内偏移。

虚页号 = 虚地址 / 页大小 = 13048 / 2KB = 6

页内偏移 = 虚地址 % 页大小 = 13048 % 2KB = 304

因此，虚地址13048对应的虚页号是6，页内偏移是304。

如果该虚页映射到12号物理页，物理地址可以计算如下：

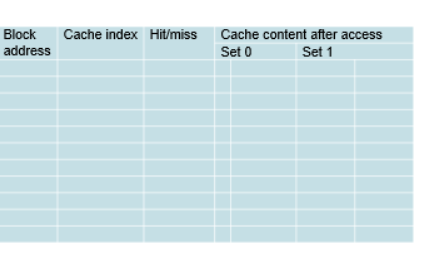
物理地址 = 物理页号 × 页大小 + 页内偏移

假设物理页大小也是2KB，那么物理地址可以计算如下：

物理页号 = 12

物理地址 = 12 × 2KB + 304 = 24576 + 304 = 24880

因此，虚页号为6的虚页映射到物理页号为12的物理页时，其物理地址为24880。

* 17. 对于4个块的cache，如果采用2路组相联方式，并假定刚开始cache没有有效数据，当访问0，1，2，3，3，6，7，6，7，0，1块时，请问此时命中率是多少？

初始状态下，缓存为空，所以第一次访问任何块都会导致缓存未命中。

访问块0：未命中（缓存为空），将块0加载到缓存中。

访问块1：未命中（缓存为空），将块1加载到缓存中。

访问块2：未命中（缓存为空），将块2加载到缓存中。

访问块3：未命中（缓存为空），将块3加载到缓存中。

访问块3：命中（块3已在缓存中）。

访问块6：未命中（缓存中没有块6），将块6加载到缓存中。

访问块7：未命中（缓存中没有块7），将块7加载到缓存中。

访问块6：命中（块6已在缓存中）。

访问块7：命中（块7已在缓存中）。

访问块0：命中（块0已在缓存中）。

访问块1：命中（块1已在缓存中）。

在上述访问序列中，共进行了11次访问，其中有6次命中。

因此，命中率为：命中次数 / 总访问次数 = 6 / 11 ≈ 0.545（约为54.5%）。

* 18. 如果有一个处理器的理想CPI=1，结构设计师正在为它选择存储部件，方案1: I-cache 缺失率2.2% D-cache缺失率3.9%，缺失代价100个时钟周期；方案2 I-cache缺失率2.6% D-cache缺失率3.6%，缺失代价85个时钟周期。目标程序中访问内存的指令占38%。请问选择哪个方案会获得更好的性能？

方案1的I-cache缺失率为2.2%，D-cache缺失率为3.9%，缺失代价为100个时钟周期。

方案2的I-cache缺失率为2.6%，D-cache缺失率为3.6%，缺失代价为85个时钟周期。

目标程序中访问内存的指令占总指令数的38%。

首先，计算平均CPI（Cycles Per Instruction）。

方案1的平均CPI = 1 + 0.38 \* 2.2% \* 100 = 1.076

方案2的平均CPI = 1 + 0.38 \* 2.6% \* 85 = 1.065

可以看出，方案2的平均CPI略低于方案1，这意味着方案2的性能更好。较低的平均CPI表示指令的平均执行时间更短。

19. 如果处理器中$t0=4100、$t1=200当程序发出lw $t1,100($t0)指令时，程序访问的物理内存是那个单元？假设此时页表（页的大小为4KB）的部分内容如下

|  |  |
| --- | --- |
| 有效位 | 物理页/硬盘上 |
| 1 | 5 |
| 1 | 2 |
| 0 | 硬盘 |
| 1 | 6 |

从 $t0 寄存器中取得基地址：4100。

从 $t1 寄存器中取得偏移量：100。

将基地址和偏移量相加：4100 + 100 = 4200。

检查页表中的情况：

有效位为1表示页面在物理内存中，使用相应的物理页号。

有效位为0表示页面不在物理内存中，需要从硬盘加载。

根据给定的页表，可以看到：

有效位为1的物理页号是5，对应的虚拟页号（页表中的索引）为5。

有效位为1的物理页号是2，对应的虚拟页号（页表中的索引）为2。

有效位为1的物理页号是6，对应的虚拟页号（页表中的索引）为6。

偏移量为100，说明访问的是页内的第100个字节。

虚拟页号5，对应的物理页号是2。所以，物理内存地址为：

物理内存地址=物理页号×页大小+偏移量

物理内存地址=物理页号×页大小+偏移量

物理内存地址=2×4096+100=8192+100=8292

物理内存地址=2×4096+100=8192+100=8292

因此，当执行 lw $t1, 100($t0) 指令时，程序访问的物理内存单元是8292。在这个情况下，虚拟页号5与物理页号2的映射是有效的。