**第二章**

**2.1　[5]<2.2> 下面的C语言表达式对应的MIPS汇编语言代码是什么？假设给定变量f、g、h和i，像在C程序中声明的一样它们都是32位的整数，使用最少的MIPS汇编指令。**

**f = g + (h-5) ；**

解：

addi f, h, -5 或 addi i, h, -5

add f, f, g add f, g, i

**2.3　[5]<2.2, 2.3> 下面的C语言表达式对应的MIPS汇编代码是什么？假设变量f、g、h、i和j分别赋值给寄存器$s0、$s1、$s2、$s3和$s4。假设数组A和B的基地址分别在寄存器$s6和$s7中。**

**B[8] = A [i-j];**

解：

sub $t0, $s3, $s4 #t0=i-j

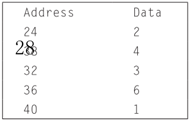
sll $t0, $t0, 2 #t0=t0\*4字节地址，32位占4个地址

add $t0, $t0, $s6 #t0 = t0+A基地址

lw $t1, 0($t0) #t1=A[i-j]

sw $t1, 32($s7) #t1即A[i-j]存入B[8] （考试必须写注释）

**2.6　下表表示在主存中存放的一个数组的32位数据。**



**2.6.1　[5]<2.2, 2.3>基于上表中数据在存储器中的位置，编写一段C代码，将数据从小到大排序，最小的数放在地址最低的位置。（假设这段数据代表了C的一个Int型数组Array，并且这台特别的机器是按照字节寻址的，且一个字包含4字节。）**

**2.6.2　[5]<2.2, 2.3>基于上表中数据在存储器中的位置，编写一段MIPS代码，将数据从小到大排序，最小的数放在地址最低的位置。（使用最少的MIPS汇编指令，假设Array的基地址保存在寄存器$s6中。）**

**原答案**

解：

1. temp = Array[0];

temp2 = Array[1];

Array[0] = Array[4];

Array[1] = temp;

Array[3] = temp2;

Array[4] = Array[3];

1. lw $t0, 0($s6)

lw $t1, 4($s6)

lw $t2, 16($s6)

sw $t2, 0($s6)

sw $t0, 4($s6)

sw $t1, 12($s6)

lw $t0, 12($s6)

sw $t0, 16($s6)

**更新后的答案**

① temp = Array[0];

temp2 = Array[1];

**temp3 = Array[3];**

Array[0] = Array[4];

Array[1] = temp;

Array[3] = temp2;

Array[4] = **temp3;**

② lw $t0, 0($s6) **#temp**

lw $t1, 4($s6)  **#temp2**

lw $t2, 16($s6) **#Array[4]**

**lw $t3, 8($s0)**   **#temp3**

sw $t2, 0($s6)

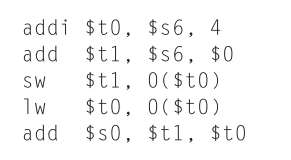
sw $t0, 4($s6)

sw $t1, 12($s6)

~~lw $t0, 12($s6)~~

sw $t**3**, 16($s6)

**2.10　[5]<2.2, 2.3> 把下面的MIPS代码翻译为C代码。假定变量f、g、h、i和j分别赋值给寄存器$s0、$s1、$s2、$s3和$s4。假定数组A和数组B的基地址分别存放在$s6和$s7中。**



解：

f=2\*(&A)

**2.12　假定寄存器$s0和$s1分别存放数值0x80000000和0xD0000000。**

**2.12.1　[5]<2.4> 下面汇编代码的$t0的值是多少？**

**add $t0, $s0, $s1**

**2.12.2　[5]<2.4> $t0中的结果是期望的结果还是发生溢出后的结果？**

**2.12.3　[5]<2.4> 对于上面定义的寄存器$s0和$s1的内容，下面汇编代码的$t0的值是多少？**

**sub $t0, $s0, $s1**

**2.12.4　[5]<2.4> $t0中的结果是期望的结果还是发生溢出后的结果？**

**2.12.5　\[5\]<2.4> 对于上面定义的寄存器$s0和$s1的内容，下面汇编代码的$t0的值是多少？**

**add $t0, $s0, $s1**

**add $t0, $t0, $s0**

**2.12.6　[5]<2.4> $t0中的结果是期望的结果还是发生溢出后的结果？**

解：

1. 0x8 = 1000(2进制) 0xD = 1101(2进制)

0x8 + 0xD = 1 0101(2进制)

所以$t0的值为50000000

1. 溢出
2. 0x8 - 0xD = 1011(2进制) = 0xB

所以$t0的值为B0000000

1. 未溢出
2. 根据①，第一行执行后$t0的值为50000000(溢出后结果)

0x5 = 0101(2进制) 0x8 = 1000(2进制)

0x5 + 0x8 = 1101(2进制) = 0xD

所以$t0的值为D0000000

1. 溢出

**2.18　假设可以将MIPS寄存器扩展到128个寄存器，并将指令集中的指令数扩展为原来的4倍。**

**2.18.1　[5]<2.5> 这将如何影响R型指令的每个位字段的大小？**

**2.18.2　[5]<2.5> 这将如何影响I型指令的每个位字段的大小？**

**原答案**

解：寄存器32\*4 -> 128个，所以寄存器位数+2

指令数扩展为原来的4倍，所以操作码位数+2

1. 操作码变为8bits, rs,rt,rd,变为7bits
2. 操作码变为8bits, rs,rt变为7bits

**更新后的答案**

**①刚开始时，指令数介于25=32（指令数严格大于32）和26=64之间（因为我们需要6位来表示op）。将指令数增加4倍后，我们将需要8位 （因此，指令数将严格大于32×4=27且小于或等于64×4=28）。**

**同样128=27，所以我们需要7位来表示寄存器。这意味着我们将需要7位来分别表示rs，rt和rd。**

**现在，我们已经使用了8 + 3 × 7 = 29位。 我们剩下3位来表示shamt和funct。 我们可以分别用1位和2位表示它们，反之亦然。 因此，有两种可能性：**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **8位** | **7位** | **7位** | **7位** | **1位** | **2位** |

**或者**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **8位** | **7位** | **7位** | **7位** | **2位** | **1位** |

**②与第一问相同。即op需要8位，而rs和rt分别需要7位。所以最后一个字段将占用32-8-7-7 = 10位。因此，指令如下所示：**

|  |  |  |  |
| --- | --- | --- | --- |
| **8位** | **7位** | **7位** | **10位** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **8** | **7** | **7** | **7** | **5** | **6** |

|  |  |  |  |
| --- | --- | --- | --- |
| **8** | **7** | **7** | **16** |

**2.22　[5]<2.6> 对于下面的C语言表达式，写一个能够完成同样操作的最短MIPS汇编指令程序段。假设$t1=A,$t2=B,$s1是C的基地址。**

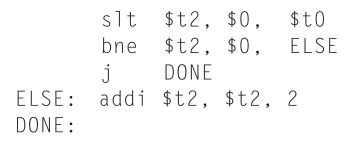
**A = C[0] << 4**

解：

lw $t2, 0($s1)

sll $t1, $t2, 4

**2.23　[5]<2.7> 假设$t0中存放数值0x00101000，在执行下列指令后$t2的值是多少？**



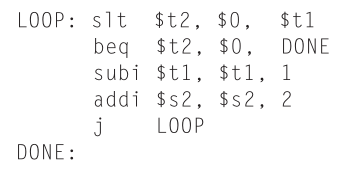
解：

0 < t0 -> t2 = 1

t2 != 0 -> ELSE

t2 = t2 + 2 =3

**2.26　考虑如下的MIPS循环：**

****

**2.26.1　[5]<2.7> 假设寄存器$t1的初始值为10，假设$s2初始值为0，则循环执行完毕时寄存器$s2的值是多少？**

**2.26.2　[5]<2.7>对于上面的循环体，写出等价的C代码例程。假定寄存器$s1、 $s2、$t1和$t2分别为整数A、B、i和temp。**

**2.26.3　[5]<2.7>假定寄存器$t1的初始值为N，上面的MIPS汇编循环执行了多少条指令？**

**原答案:**

解：

1. 20 (s2+=2 循环10次)
2. i=10

do{

B+=2;

i=i-1;

} while(i>0)

1. 5\*N

**更新后的答案**

② **解法二：**

i=10

**While(i>0){**

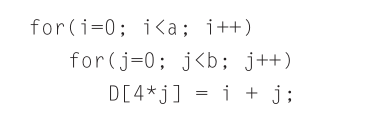
**i=i-1;**

**B+=2;**

**}**

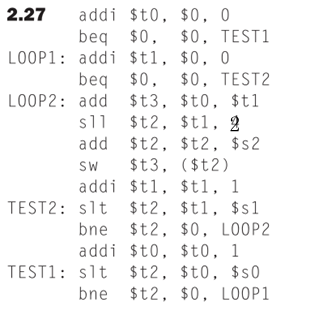
③5\*N**+2**

**2.27　[5]<2.7>将下面的C代码翻译为MIPS汇编代码。要求使用的指令数目最少。假设值a、b、i和j分别存放在寄存器$s0、$s1、$t0和$t1中。另外假设寄存器$s2中存放着数组D的基地址。**

****

**原答案**

解：



# i清0（$t0存i）

# 转test1

# j清0（$t1存j）

# 转test2

# t3 = i+j

# t2 = j 左移2位 即j=j\*4

# t2 = t2+s2（s2存D基址）

# D(4\*j) = i+j

# j = j+1

# t1(即j)<b t2 = 1

# t2≠0 Loop2

# i = i+1

# t0<a 则t2=1

# t2≠0 Loop1

**更新后的答案**

# i清0（$t0存i）

# 转test1

# j清0（$t1存j）

# 转test2

# t3 = i+j

# t2 = j 左移2位 即j=j\*4,**字节地址再乘以4，所以左移4位**

# t2 = t2+s2（s2存D基址）

# D(4\*j) = i+j

# j = j+1

# t1(即j)<b t2 = 1

# t2≠0 Loop2

# i = i+1

# t0<a 则t2=1

# t2≠0 Loop1

addi$t0，$0，0

beq $0，$0，TEST1

Loop1: addi $t1，$0，0

beq $0，$0，TEST2

Loop2: add $t3，$t0，$t1

sll $t2，$t1，**4**

add $t2，$t2，$s2

sw $t3，**0**（$t2）

addi $t1，$t1，1

TEST2: slt $t2，$t1，$s1

bne $t2，$0，LOOP2

addi $t0，$t0，1

TEST1: slt $t2，$t0，$s0

bne $t2，$0，LOOP1

**2.40　[5]<2.6,2.10>如果当前PC值是0x00000000，可以使用单独的跳转指令跳转到练习题2.39中所指定的PC地址吗？**

解：

2.39中所指定的PC地址：0010 0000 0000 0001 0100 1001 0010 0100（2）

=0x20014924

跳转指令 j target 其中target26位（j指令因为高位不是符号位扩展）

所以依据其寻址方式，它最远跳到：

0000 1111 1111 1111 1111 1111 1111 1100 （来自PC，26位target，移位，见上课ppt）

即：0x0FFFFFFC

0x 0000 0000+? = 0x 20014924

? = 0x 20014924 > 0x 0FFFFFFC 所以不能跳

**2.41　[5]<2.6,2.10>如果当前PC值是0x00000600，可以使用单独的分支指令跳转到练习题2.39中所指定的PC地址吗？**

**原答案**

解：不能

**更新后的答案**

**答案一：**

**不能**

**分支指令bne rs,rt,offset其中offset为16位有符号整数，所以依据其寻址方式，它最远跳到：PC+4+offset\*4**

**当offset为正的时候,offset最大为0111 1111 1111 1111(共计15个1,符号位为0表示正数)**

**offset\*4并符号位扩展后为**

**0000 0000 0000 0001 1111 1111 1111 1100(32bits,且向右移位2bits)=0x1FFFC**

**因此正的时候最远跳到PC+4+offset\*4=0x600+0x4+0x1FFFC=0x00020600**

**当offset为负的时候,offset最大为-2^15=1000 0000 0000 0000(补码 共计15个0,符号位为1表示负数)**

**offset\*4为-2^17=0x00020000**

**因此负的时候最远跳到PC+4-offset\*4=0x604-0x00020000**

**=0xFFFE0604**

**所以范围是0xFFFE06040到x00020600**

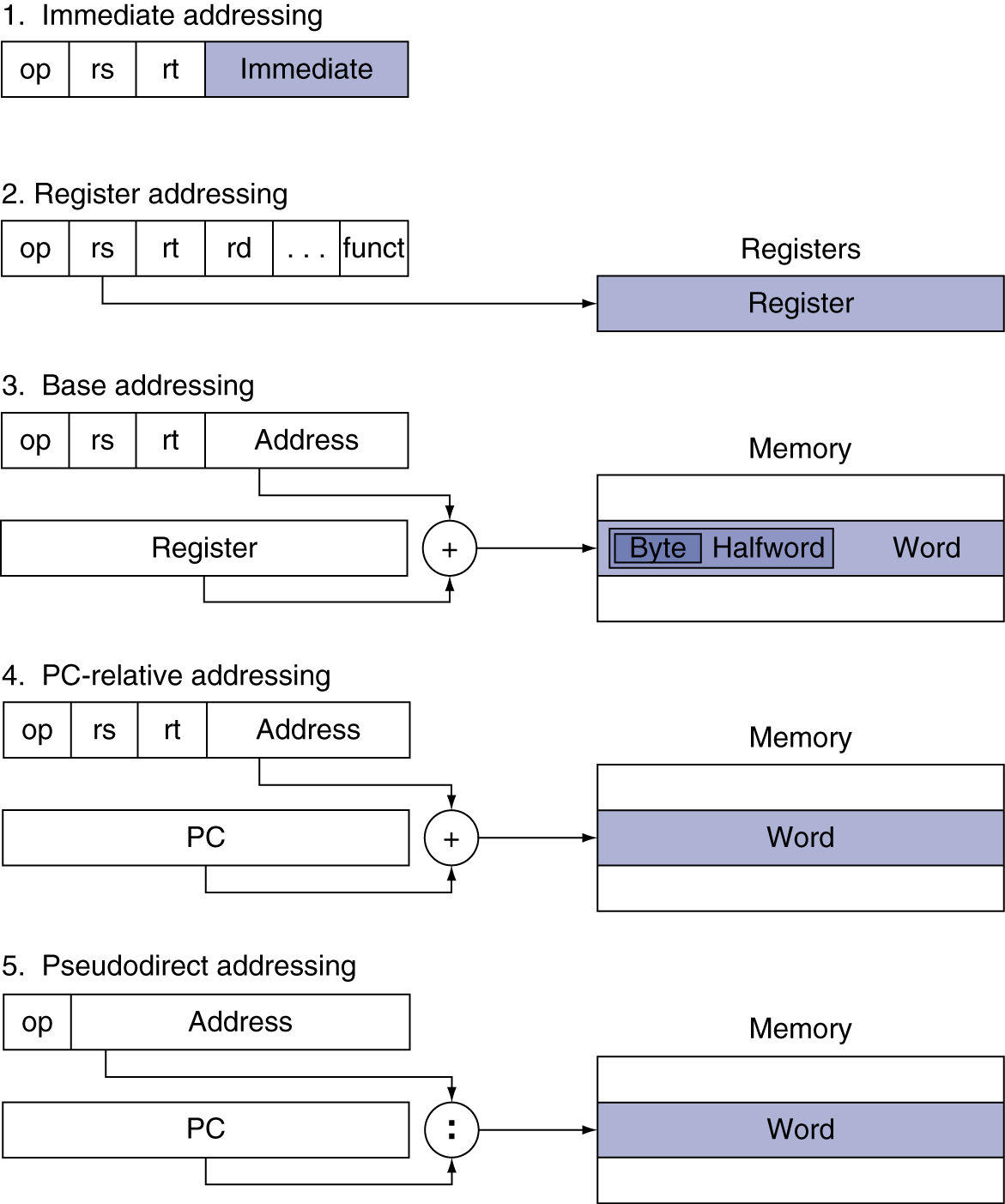
**这时,我们发现 0010 0000 0000 0001 0100 1001 0010 01002=0x20014924>0x00020600,不在范围内,所以不可能跳转到。**

**（0x604 + 0x1FFFC = 0x0002 0600**

**0x604 – 0x20000 = 0xFFFE 0604.**

**理解：指令中offset存的值无论大小正负，系统并不判断。**

**PC+4+offset+4 那么若offset取补码 1000 0000 0000 0000 则为负的绝对值最大值，左移2位，即-217 ，则最远负向跳转到0x0604+(-217)=0x0604-(217)=0x604 – 0x20000,即负向最远的地址。4位：补码1000为-8即-23。同理，补码10 0000 0000 0000 0000=-217转为十六进制 -0x20000 即10 0000 0000 0000 0000补码大小为-0x2000,所以PC+4+10 0000 0000 0000 0000=PC+4+（-0x2000））**



地图的截图

描述已自动生成

图片包含 游戏机, 截图

描述已自动生成

P77例题

pc=80016  
exit=80024  
80024=80016+8  
8/4=2  
loop=80000  
pc=80024  
32位2进制：0000 0000 0000 0001 0011 1000 1000 00 00  
去掉最后两位是1 0011 1000 1000 00 也就是20000