2.1

addi $t0, $s2, -5 # t0=h-5
add $s0, $s1, $t0 # f=g+t0

2.3

sub $t0, $s3, $s4 # t0=i-j
sll $t0, $t0, 2 # t0=4*[i-j]
add $t1, $t0, $s6 # t1=s6+4*(i-j), t1=&A[i-j]
lw $t2, 0($t1) # t2=A[i-j]
sw $t2, 32($s7) # B[8]=t2

2.4

B[g] = A[f] + A[f+1]

2.7

sll $t0, $s3, 3 # t0=i*8*
add $t1, $t0, $s6 # t1=&A[i]
lw $t3, 0($t1) # t3=A[i]
*sll $t0, $s4, 3 # t0=j*8
add $t2, $t0, $s6 # t2=&A[j]
lw $t4, 0($t2) # t4=A[j]
add $t0, $t3, $t4 # t0=A[i]+A[j]
sw $t0, 64($s7) # B[8]=A[i]+A[j]

2.8

A[1] = &A[0]
f = &A[0] + &A[0]

2.16

2.16.1

- op: 在R型指令中，op段只用来指示这条指令是R型指令，因此本段长度不变

- rs: 原本有32个寄存器,需要五位数来指明寄存器编号, 现在有128个寄存器，翻了四倍,因此需要七位来指明寄存器编号
- rt, rd:与rs相同，都扩展为7位
- shamt: 用于指示移位位数，由于字长没有改变，最长移位位数仍为32，因此保持5位不变
- func: 用于指明指令功能， 因为指令数量翻了四倍， 因此此处也应翻四倍，扩展为8位
- 综上所述，扩展之后的R型指令位数为$6 + 7 * 3 + 5 + 8 = 40$

2.16.2

- op: I型指令的功能取决于op段，因为指令数量翻了4倍，op段也应该对应扩展两位，为8位
- rs，rt：同R型，扩展为7位
- constant/address：用于给出立即数作为操作数或地址，可以不扩展，但为遵循MIPS保持指令长度一致的思想，可扩展为25位

2.16.3

- 增加
  两种改变都直接增加了指令的长度， 也就相应的增加了MIPS汇编程序的大小
- 减少
  在寄存器增多，并且指令集指令数的增加的条件下，每条指令可以实现的功能就更丰富了，原本可能需要多条指令完成的功能，在扩展后的指令集中可能只需要一条对应的指令， 这样能够减少总指令的条数，也就减小了MIPS汇编程序的大小

2.17

2.17.1

（假设题目印刷错误，应为左移4位）
sll $t2, $t0, 4
移位后t2值为0xAAAA AAA0
or $t2, $t2, $t1
t2:1010 1010 1010 1010 1010 1010 1010 0000
t1:0001 0010 0011 0100 0101 0110 0111 1000
=> 1011 1010 1011 1110 1111 1110 1111 1000
故$t2为0xBABE FEF8

（假设题目印刷无误，应为左移44位）
sll $t2, $t0, 44
移位后t2值为0x0000 0000
or $t2, $t2, $t1

t2 = t1 or 0
故$t2 = 0x1234 5678

2.17.2

sll $t2, $t0, 4
移位后t2值为0xAAAA AAA0
andi $t2, $t2, -1
-1表示为0xFFFFFFFF
故$t2为0xAAAA AAA0

2.17.3

srl $t2, $t0, 3
移位后t2为0x1555 5555
andi $t2, $t2, 0xFFEF
t2:0001 0101 0101 0101 0101 0101 0101 0101
:0000 0000 0000 0000 1111 1111 1110 1111
=> 0000 0000 0000 0000 0101 0101 0100 0101
故$t2为0x0000 5545

2.22

2.22.1

jal 前6为为操作码，26位为地址，由于MIPS按字对齐，故26位可右移4位表示28位的地址，再将高四位替换为PC的地址，组成32位地址
故范围为
[0x2000 0000, 0x2FFF FFFC]

2.22.2

bne 相对PC寻址，偏移量为16位，最大为0x7FFF,最小为0x8000
故范围为
[0x1FFE 0000, 0x2001 FFFC]

2.24

2.24.1

仍为0，因为循环体内没有对$s0的操作

2.24.2

```
for(i = 10; i>0; i--)
{
    B += 2;
}
```

2.24.3

$5 * N + 2$

2.27

(假设最后一行分支指令中$s0应为$0)

```
for(i = 0; i<100 ; i++)
{
    result = MemArray[i];
}
```

2.39

2.39.1

设一个时钟周期的长度是1
则原本用时为：

$$time_1 = 5 * 10^8 * 1 + 3 * 10^8 * 10 + 1 * 10^8 * 3 = 3.8 * 10^9$$

修改后，算数指令有$3 * 10^8$条，时钟周期长度为1.1
修改后用时为：

$$time_2 = 3.75 * 10^8 * 1.1 + 3 * 10^8 * 11 + 1 * 10^8 * 3.3 = 4.0425 * 10^9$$

修改后用时变长，因此不是好的设计选择

2.39.2

- 两倍

$$CPI_2 = 0.5$$

$$time_3 = 5 * 10^8 * 0.5 + 3 * 10^8 * 10 + 1 * 10^8 * 3 = 3.55 * 10^9$$

$$\text{加速倍数}_2 = \frac{time_1}{time_3} = \frac{3.8 * 10^9}{3.55 * 10^9} \approx 1.07$$

- 10倍

$$CPI_{10} = 0.1$$

$$time_4 = 5 * 10^8 * 0.1 + 3 * 10^8 * 10 + 1 * 10^8 * 3 = 3.35 * 10^9$$

$$\text{加速倍数}_2 = \frac{time_1}{time_4} = \frac{3.8 * 10^9}{3.35 * 10^9} \approx 1.13$$

2.40

2.40.1

$$CPI_{aveg} = 2 * 0.7 + 6 * 0.1 + 3 * 0.2 = 2.6$$

2.40.2

设优化后算数指令CPI为x

$$0.7 * x + 6 * 0.1 + 3 * 0.2 = 2.6/(1 + 25\%)$$

解得$x \approx 1.26$
即算数指令平均需要1.26个时钟周期

2.40.3

设优化后算数指令CPI为y

$$0.7 * y + 6 * 0.1 + 3 * 0.2 = 2.6/(1 + 50\%)$$

解得$y \approx 0.76$
即算数指令平均需要0.76个时钟周期