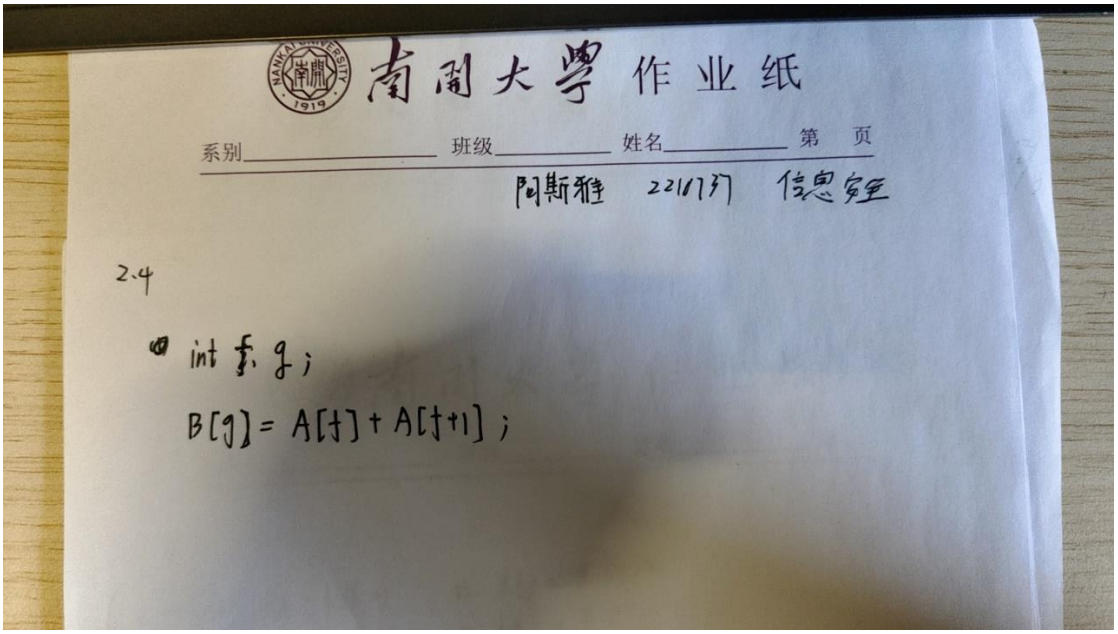
2.4


南開大學 作业纸

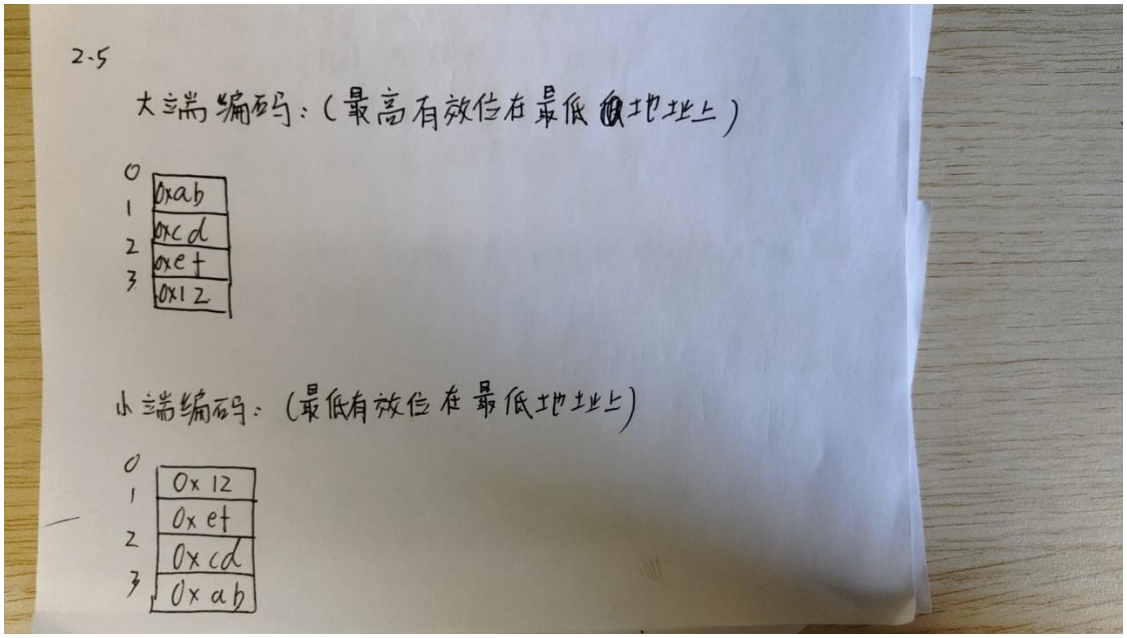
系别 班级 姓名 第 页

阿斯雅 2211737 信息安全

2.4

```
int i, j;
B[j] = A[i] + A[i+1];
```

2.5


2.5

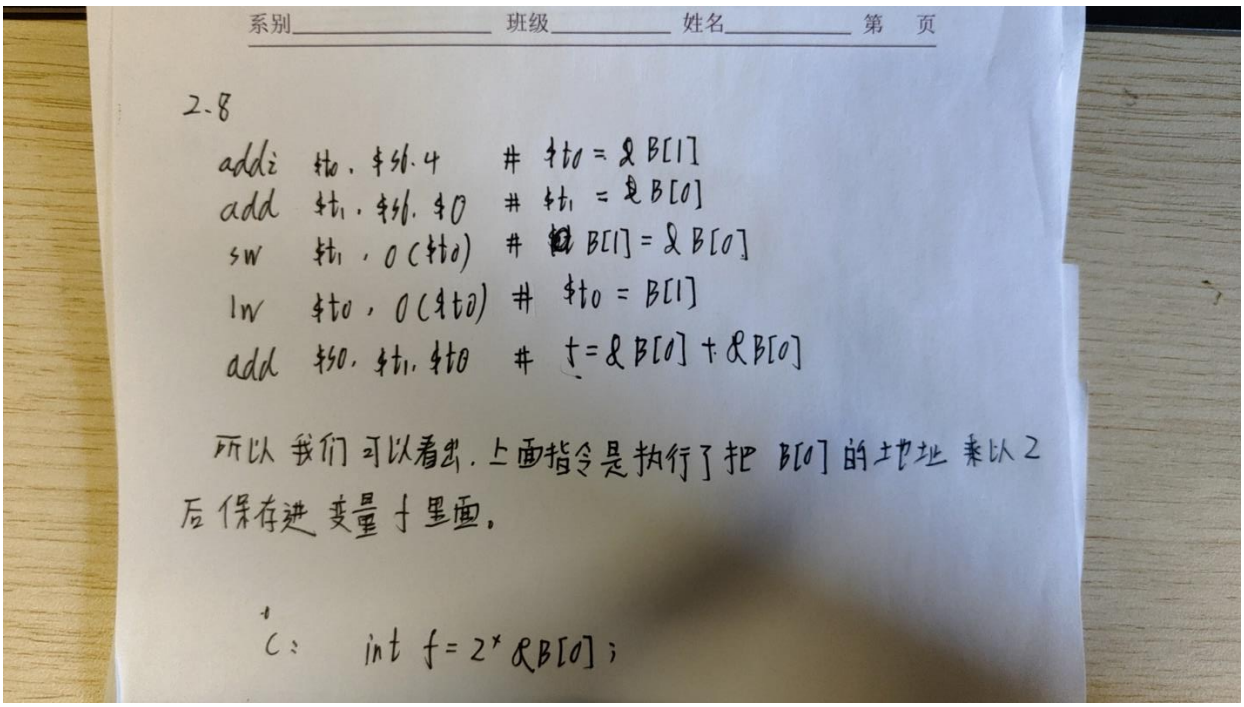
大端编码：（最高有效位在最低地址上）

0	0xab
1	0xcd
2	0xef
3	0x12

小端编码：（最低有效位在最低地址上）

0	0x12
1	0xef
2	0xcd
3	0xab

2.8



系别_____ 班级_____ 姓名_____ 第 页

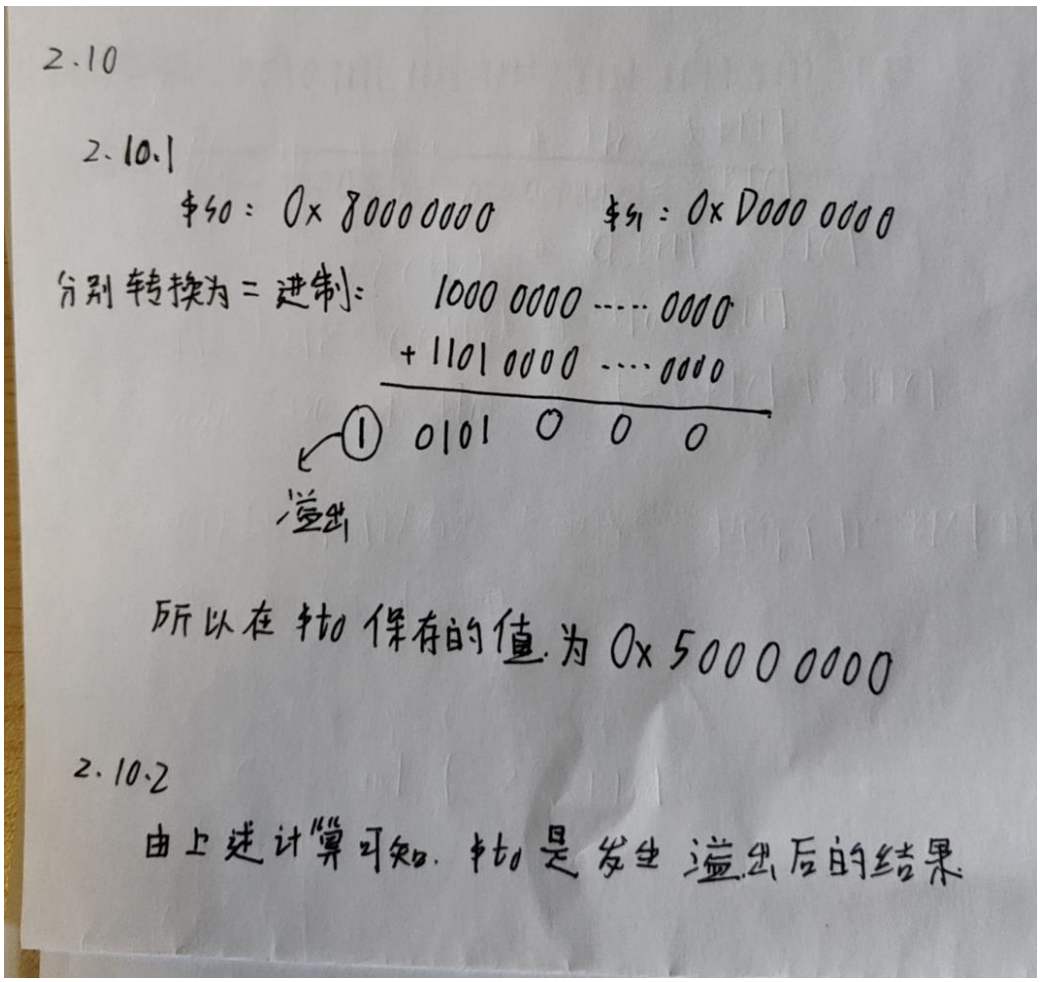
2.8

```
addi  $t0, $s0, 4    # $t0 = &B[1]
add   $t1, $s0, $0   # $t1 = &B[0]
sw    $t1, 0($t0)    # B[1] = &B[0]
lw    $t0, 0($t0)    # $t0 = B[1]
add   $s0, $t1, $t0  # f = &B[0] + &B[0]
```

所以我们可以看出，上面指令是执行了把 B[0] 的地址乘以2后保存进变量 f 里面。

C: int f = 2 * &B[0];

2.10



2.10

2.10.1

$s0: 0x80000000　　$s1: 0xD0000000

分别转换为二进制: 1000 0000 0000

+ 1101 0000 0000

① 0101 0 0 0

溢出

所以在 $t0 保存的值为 0x50000000

2.10.2

由上述计算可知，$t0 是发生溢出后的结果

系别

2.10.3.

先对 0xD000 0000 取 补码运算得:

1101 0000 0000 0000 0000 0000 0000 0000

先按位取反: 0010 1111 1111 1111 1111 1111 1111 1111

再加1: + 0001

0010 0000 0000 0000 0000 0000 0000 0000

然后再根 0x8000 0000 做加法:

1000 0000 0000

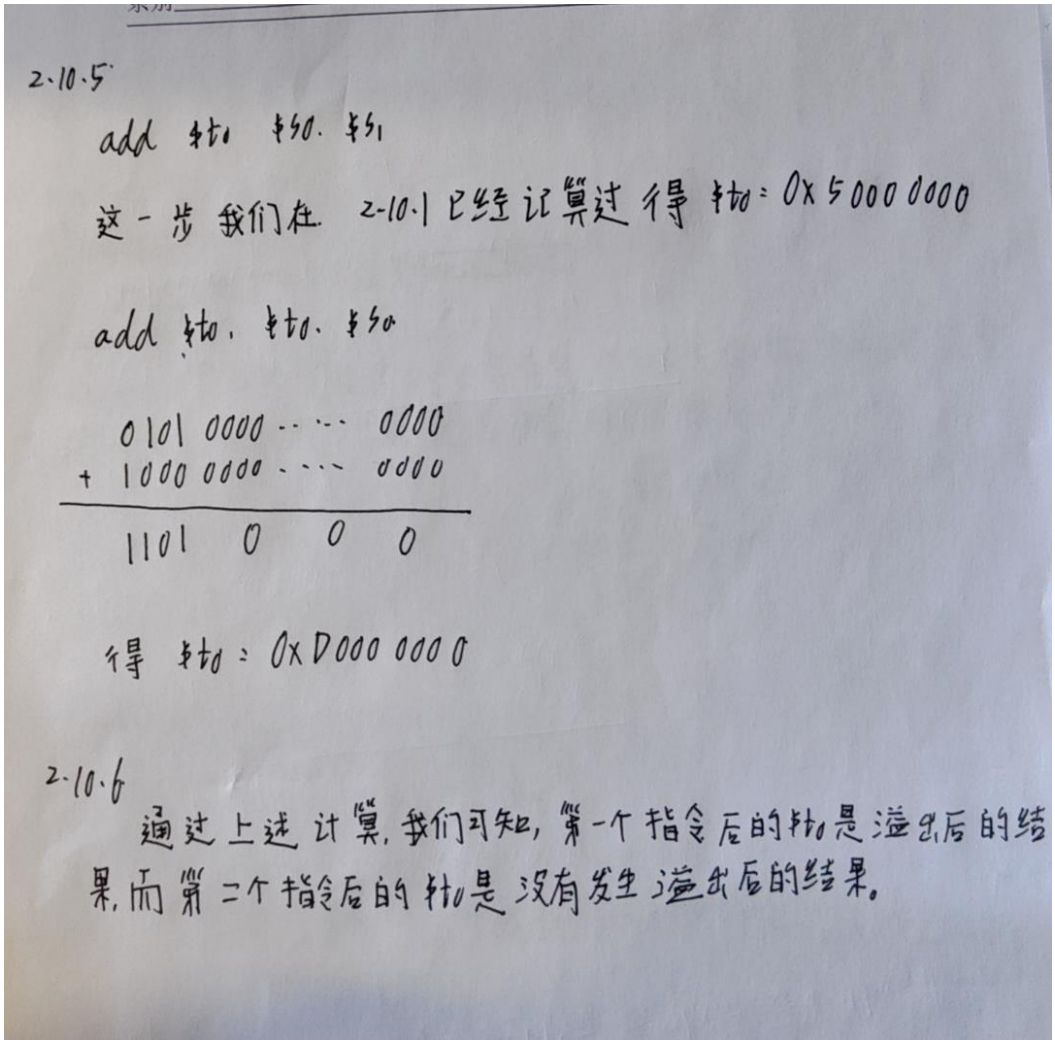
+ 0011 0000 0000

1011 0 0 0

所以 t1 的值为 0xB000 0000

2.10.4:

根据上面运算,可知 t1 是没有发生溢出的结果

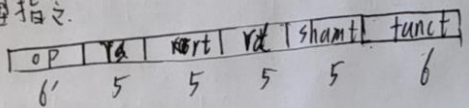

2.10.5
add $t0 $s0. $s1
这一步我们在 2-10-1 已经计算过得 $t0 = 0x5000 0000
add $t0. $t0. $s0
0101 0000 …… 0000
+ 1000 0000 …… 0000
1101 0 0 0
得 $t0 = 0xD000 0000
2.10.6
通过上述计算，我们可知，第一个指令后的$t0是溢出后的结果，而第二个指令后的$t0是没有发生溢出后的结果。

2.16

2.16

2.16.1

R型指令

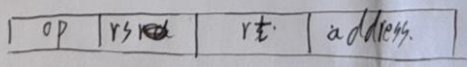


因为由32个寄存器变成了128个寄存器，所以5位不能满足，需要 $(2^7 = 128)$，7个位，也就是说 rd、rs、rt 要变成7位。

因为每个指令都有它特定的操作码，所以当指令数变成4倍时，操作码也要变成四倍，也就是向左移2位，变成8位。

2.16.2

I型指令



像上面讨论的一样，表示寄存器的位数要变成7位，也就是 rt 和 rs 要变成7位，而表示操作码的op字段要变成8位。

2.16.3

减少程序大小：首先更多的寄存器，可以有效的简化我们的代码，不需要每次重复的使用一个寄存器或使用栈保存数据，而更多的指令，可以让我们写代码的时候有了更多的选择，让我们可以选择合适的指令，而不是用好几个指令来去执行一个步骤。

增加程序：由 2.16.1 和 2.16.2 我们可以看到，由于更多的寄存器出现，我们的指令的位数也会相应的增加，而这会使我们的程序所占内存会增加，同样的道理，更多的指令数，我们需要更长的操作码去表示，也会使指令所占内存增大。