

CSc 252: Computer Organization

HW 1

due at 5pm, two days **before** the test

Turn in through GradeScope

Solutions

Policy Reminders

- You must turn in a single PDF to GradeScope. Other file formats will not be accepted. (Sorry, this is necessary for the sanity of the TAs!)
- You are allowed to work with other students on this homework, as we will not be grading it for correctness. However, **each student must turn in their own copy of the homework.**
- **Show your work for all problems.** While we won't be grading for correctness, you will not receive full credit unless you show your work.

After all, showing your work is required on the test - and homeworks are intended to help you practice for the test!

Required Problems:

1.1(all), 1.2(k-m), 1.3(j-l), 1.4(i-k)

Problem 1.1 - Encoding in Binary

For each power below, give an approximate value for it. For instance, for the number 2^{32} , give "4 billion."

$$2^{29} \approx 512 \text{ million}$$

$$2^{42} \approx \underline{\quad 4 \text{ trillion} \quad} \quad 2^{17} \approx \underline{\quad 128 \text{ thousand} \quad}$$

$$2^{20} \approx \underline{\quad 1 \text{ million} \quad} \quad 2^{16} \approx \underline{\quad 64 \text{ thousand} \quad}$$

(the homework continues on the next page)

Problem 1.2 - Encoding in Binary

Convert the following decimal numbers to binary. You may use any method, but make sure to **show your work**. Give the answer as in 16-bit 2's complement form.

- (a) 107
- (b) -3097
- (c) 21720
- (d) -1
- (e) 68
- (f) -6143
- (g) 937
- (h) -129
- (i) 1132
- (j) 5555
- (k) **Turn in this one:** 7544

Solution: $7544 = 4096 + 3448$
 $7544 = 4096 + 2048 + 1400$
 $7544 = 4096 + 2048 + 1024 + 376$
 $7544 = 4096 + 2048 + 1024 + 256 + 120$
 $7544 = 4096 + 2048 + 1024 + 256 + 64 + 56$
 $7544 = 4096 + 2048 + 1024 + 256 + 64 + 32 + 24$
 $7544 = 4096 + 2048 + 1024 + 256 + 64 + 32 + 16 + 8$

 $7544_{10} = 0001\ 1101\ 0111\ 1000_2$

- (l) **Turn in this one:** 2974

Solution: $2974 = 2048 + 926$
 $2974 = 2048 + 512 + 414$
 $2974 = 2048 + 512 + 256 + 158$
 $2974 = 2048 + 512 + 256 + 128 + 30$
 $2974 = 2048 + 512 + 256 + 128 + 16 + 14$
 $2974 = 2048 + 512 + 256 + 128 + 16 + 8 + 6$
 $2974 = 2048 + 512 + 256 + 128 + 16 + 8 + 4 + 2$

 $2974_{10} = 0000\ 1011\ 1001\ 1110_2$

- (m) **Turn in this one:** -675

Solution: The starting value is negative, so we will first convert the positive value and then perform 2's complement:

$$675 = 512 + 163$$

$$675 = 512 + 128 + 35$$

$$675 = 512 + 128 + 32 + 3$$

$$675 = 512 + 128 + 32 + 2 + 1$$

$$675_{10} = 0000\ 0010\ 1010\ 0011_2$$

2's complement:

0000 0010 1010 0011	original
1111 1101 0101 1100	bitwise negation
1111 1101 0101 1101	plus one

$$-675_{10} = 1111\ 1101\ 0101\ 1101_2$$

Problem 1.3 - Some Binary Arithmetic and Conversions

For each of the pairs of numbers below, compute:

- hexadecimal (base 16) equivalents for both **a** and **b**; assume unsigned numbers.
- octal (base 8) equivalents for both **a** and **b**; assume unsigned numbers.
- decimal (base 10) equivalents for both **a** and **b**; assume **signed** numbers.
- **a+b** - Indicate if overflow and/or carry-out occurs; explain your answer.
Assume signed numbers.
(Do not convert to another base; do your work, and also give your answer, in binary.)
- **a-b** by negating **b** and adding. Indicate if overflow and/or carry-out occurs; explain your answer.
Assume signed numbers.
(Do not convert to another base; do your work, and also give your answer, in binary.)

NOTE: Assume that 16-bit binary numbers are being used in this problem. Signed numbers are always encoded using two's complement.

(a)

a = 0100 0111 0101 1000
b = 1000 0000 1100 0110

(b)

a = 0001 0000 0011 1000
b = 0111 0010 0100 1011

(c)

a = 0000 0000 0110 1100
b = 0000 0001 1010 1001

(d)

a = 1101 1011 0011 1111
b = 1010 1100 1100 0001

(e)

a = 0101 0111 1001 0000
b = 1000 1100 0111 0000

(f)

a = 1001 1000 1100 0001
b = 0100 0111 0111 0110

(g)

a = 1101 1000 1000 1010
b = 0010 0101 1101 1101

(h)

a = 0010 0000 1001 1011
b = 1110 0111 1101 0111

(i)

a = 0101 0011 1010 0001
b = 1001 0111 1001 1111

(j) - Turn in this one

a = 0000 1001 0010 1111
b = 0101 1100 0001 1100

Solution:

Hexadecimal

a = 0x092f
b = 0x5c1c

Octal

a = 004457 octal
b = 056034 octal

Decimal

$a = 2^{11} + 2^8 + 2^5 + 2^3 + 2^2 + 2^1 + 2^0$
 $a = 2048 + 256 + 32 + 8 + 4 + 2 + 1$
 $a = 2304 + 32 + 15$
 $a = 2304 + 47$
 $a = 2351$
 $b = 2^{14} + 2^{12} + 2^{11} + 2^{10} + 2^4 + 2^3 + 2^2$
 $b = 16384 + 4096 + 2048 + 1024 + 16 + 8 + 4$
 $b = 16384 + 7168 + 28$
 $b = 23552 + 28$
 $b = 23580$

a+b

```
          11      111 1
a:      0000 1001 0010 1111
b:      + 0101 1100 0001 1100
-----
          0110 0101 0100 1011
```

No overflow. We started with a positive value, and added a positive, and the result was positive.

a-b

We calculate the 2's complement of b:

```
      b   = 0101 1100 0001 1100
      ~b  = 1010 0011 1110 0011
      -b = ~b+1 = 1010 0011 1110 0100
```

```
          111 11 1 1
a:      0000 1001 0010 1111
-b:      + 1010 0011 1110 0100
-----
          1010 1101 0001 0011
```

No overflow. We started with a positive value, subtracted a positive, so overflow is impossible.

(k) - Turn in this one

```
a = 0011 1001 1110 1111
b = 0011 1101 0010 0010
```

Solution:

Hexadecimal

```
a = 0x39ef
b = 0x3d22
```

Octal

```
a = 034757      octal
b = 036442      octal
```

Decimal

$$\begin{aligned}a &= 2^{13} + 2^{12} + 2^{11} + 2^8 + 2^7 + 2^6 + 2^5 + 2^3 + 2^2 + 2^1 + 2^0 \\a &= 8192 + 4096 + 2048 + 256 + 128 + 64 + 32 + 8 + 4 + 2 + 1 \\a &= 12288 + 2048 + 384 + 96 + 15 \\a &= 14336 + 480 + 15 \\a &= 14336 + 495 \\a &= 14831 \\b &= 2^{13} + 2^{12} + 2^{11} + 2^{10} + 2^8 + 2^5 + 2^1 \\b &= 8192 + 4096 + 2048 + 1024 + 256 + 32 + 2 \\b &= 12288 + 3072 + 256 + 34 \\b &= 15360 + 290 \\b &= 15650\end{aligned}$$

a+b

$$\begin{array}{r} \\ \\ a: \\ b: 0011 \\ \hline 0111 \end{array}$$

No overflow. We started with a positive value, and added a positive, and the result was positive.

a-b

We calculate the 2's complement of b:

$$\begin{aligned}b &= 0011 \ 1101 \ 0010 \ 0010 \\ \sim b &= 1100 \ 0010 \ 1101 \ 1101 \\ -b &= \sim b + 1 = 1100 \ 0010 \ 1101 \ 1110\end{aligned}$$

$$\begin{array}{r} \\ \\ a: \\ -b: 1100 \\ \hline 1111 \end{array}$$

No overflow. We started with a positive value, subtracted a positive, so overflow is impossible.

(1) - Turn in this one

$$\begin{aligned}a &= 1111 \ 0010 \ 0010 \ 0111 \\ b &= 0010 \ 0011 \ 0011 \ 0011\end{aligned}$$

Solution:

Hexadecimal

a = 0xf227
b = 0x2333

Octal

a = 171047 octal
b = 021463 octal

Decimal

The high bit of a is 1, so the number is negative. We invert and add one to get the positive value.

a = 1111 0010 0010 0111
~a = 0000 1101 1101 1000
-a = ~a+1 = 0000 1101 1101 1001

$-a = 2^{11} + 2^{10} + 2^8 + 2^7 + 2^6 + 2^4 + 2^3 + 2^0$
 $-a = 2048 + 1024 + 256 + 128 + 64 + 16 + 8 + 1$
 $-a = 3072 + 384 + 80 + 9$
 $-a = 3456 + 89$ $-a = 3545$ $a = -3545$
 $b = 2^{13} + 2^9 + 2^8 + 2^5 + 2^4 + 2^1 + 2^0$
 $b = 8192 + 512 + 256 + 32 + 16 + 2 + 1$
 $b = 8192 + 768 + 48 + 3$
 $b = 8960 + 51$
 $b = 9011$

a+b

```
      1 11   1   1   111
a:    1111 0010 0010 0111
b:    + 0010 0011 0011 0011
-----
      0001 0101 0101 1010
```

No overflow. We started with a negative value, and added a positive, so overflow was impossible.

a-b

We calculate the 2's complement of b:

b = 0010 0011 0011 0011
~b = 1101 1100 1100 1100
-b = ~b+1 = 1101 1100 1100 1101

```

      1 111          1 111
a:    1111 0010 0010 0111
-b:  + 1101 1100 1100 1101
-----
      1100 1110 1111 0100

```

No overflow. We started with a negative value, subtracted a positive, and got a negative result.

Problem 1.4 - Basic MIPS

This question assumes the following MIPS code, which sets up memory locations `hermit`, `kaibab`, `tanner`, `clear`, `creek`, `ribbon`, `falls`, and `tonto`. The code then loads the values of some of these variables into the indicated MIPS registers. In answering these questions, you can assume this code has already been executed, and that the value of some of the variables are already in the indicated registers.

Each question is **independent** of the other questions - that is, assume that the program has started over from scratch each time.

Do not modify any `sX` register, unless specifically instructed.

```
.data
hermit:    .word    xxx          # hidden so you can't hard-code values!
kaibab:    .word    xxx
tanner:    .word    xxx
clear:     .word    xxx
creek:     .word    xxx
ribbon:    .word    xxx
falls:     .word    xxx
tonto:     .word    xxx

.text
main:
    # set $s3 = tonto
    la    $t0, tonto
    lw    $s3, 0($t0)

    # set $s4 = hermit
    la    $t0, hermit
    lw    $s4, 0($t0)

    # set $s5 = clear
    la    $t0, clear
    lw    $s5, 0($t0)

    # set $s6 = creek
    la    $t0, creek
    lw    $s6, 0($t0)
```

(a)

Put `clear + creek` in register `$t9`

(b)

Put `hermit - creek - clear + tonto` in register `$t2`

(c)

Put `hermit + falls` in register `$t1`

(d)

Put `tonto - clear + hermit` in memory location `ribbon`

(e)

If (tonto != hermit), put tonto + clear in register \$s2

(f)

If (creek >= clear), put ribbon + clear in register \$s2

(g)

If kaibab - tanner == falls - tonto, put tonto in register \$s7.

(h)

Add 10 to ribbon, and store the updated value back into the variable.

(i) - Turn in this one

Put tonto+hermit into the variable ribbon.

Solution:

```
add    $t0, $s3,$s4    # t0 = tonto+hermit
la     $t1, ribbon     # t1 = &ribbon
sw     $t0, 0($t1)    # ribbon = tonto+hermit
```

(j) - Turn in this one

If (clear == falls), put 1 into register \$s3; otherwise, put 0 into it.

Solution:

```
la     $t1, falls     # t1 = &falls
lw     $t1, 0($t1)    # t1 = falls

bne    $s5,$t1, NOT_EQUAL # if (clear != falls) jump ahead

addi   $s3, $zero,1   # s3 = 1
j      AFTER

NOT_EQUAL:
addi   $s3, $zero,0   # s3 = 0

AFTER:
```

(k) - Turn in this one

Put tonto*3 + kaibab - creek into register \$t7.

Solution:

```
add    $t7, $s3,$s3      # t7 = tonto*2
add    $t7, $t7,$s3      # t7 = tonto*3

la     $t0, kaibab       # t0 = &kaibab
lw     $t0, 0($t0)       # t0 = kaibab

add    $t7, $t7,$t0      # t7 = tonto*3 + kaibab
sub    $t7, $t7,$s6      # t7 = tonto*3 + kaibab - creek
```

EXAMPLES

(begin on next page)

Example: Problem 1.2(a)

$$\begin{aligned}107 &= 64 + 43 \\107 &= 64 + 32 + 11 \\107 &= 64 + 32 + 8 + 3 \\107 &= 64 + 32 + 8 + 2 + 1 \\107_{10} &= 0000\ 0000\ 0110\ 1011_2\end{aligned}$$

Example: Problem 1.2(b)

Since the number is negative, we convert the positive number to binary, and then do 2's complement.

$$\begin{aligned}3097 &= 2048 + 1049 \\3097 &= 2048 + 1024 + 25 \\3097 &= 2048 + 1024 + 16 + 9 \\3097 &= 2048 + 1024 + 16 + 8 + 1 \\3097_{10} &= 0000\ 1100\ 0001\ 1001_2 \\2\text{'s complement:}\end{aligned}$$

0000 1100 0001 1001	(positive value)
1111 0011 1110 0110	(negated)
1111 0011 1110 0111	(add one)

ANSWER:

$$1111\ 0011\ 1110\ 0111$$

Example: Problem 1.2(c)

$$\begin{aligned}21720 &= 16384 + 5336 \\21720 &= 16384 + 4096 + 1240 \\21720 &= 16384 + 4096 + 1024 + 216 \\21720 &= 16384 + 4096 + 1024 + 128 + 88 \\21720 &= 16384 + 4096 + 1024 + 128 + 64 + 24 \\21720 &= 16384 + 4096 + 1024 + 128 + 64 + 16 + 8 \\21720_{10} &= 0101\ 0100\ 1101\ 1000_2\end{aligned}$$

Example: Problem 1.2(d)

-1 is one of the "magic" numbers - we don't need to do any work to find it:

$$-1_{10} = 1111\ 1111\ 1111\ 1111_2$$

Example: Problem 1.2(e)

$$\begin{aligned}68 &= 64 + 4 \\68_{10} &= 0000\ 0000\ 0100\ 0100_2\end{aligned}$$

Example: Problem 1.2(f)

Since the value is negative, we'll convert the positive value first.

```
6143 = 4096 + 2047
6143 = 4096 + 1024 + 1023
6143 = 4096 + 1024 + 512 + 511
6143 = 4096 + 1024 + 512 + 256 + 255
6143 = 4096 + 1024 + 512 + 256 + 128 + 127
6143 = 4096 + 1024 + 512 + 256 + 128 + 64 + 63
6143 = 4096 + 1024 + 512 + 256 + 128 + 64 + 32 + 31
6143 = 4096 + 1024 + 512 + 256 + 128 + 64 + 32 + 16 + 15
6143 = 4096 + 1024 + 512 + 256 + 128 + 64 + 32 + 16 + 8 + 7
6143 = 4096 + 1024 + 512 + 256 + 128 + 64 + 32 + 16 + 8 + 4 + 3
6143 = 4096 + 1024 + 512 + 256 + 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1
614310 = 0001 0111 1111 11112
2's complement:
```

0001 0111 1111 1111	(positive value)
1110 1000 0000 0000	(negated)
1110 1000 0000 0001	(add one)

ANSWER:

```
1110 1000 0000 0001
```

Example: Problem 1.2(g)

```
937 = 512 + 425
937 = 512 + 256 + 169
937 = 512 + 256 + 128 + 41
937 = 512 + 256 + 128 + 32 + 9
937 = 512 + 256 + 128 + 32 + 8 + 1
```

```
93710 = 0000 0011 1010 10012
```

Example: Problem 1.2(h)

Since the value is negative, we'll convert the positive value first.

```
129 = 128 + 1
12910 = 0000 0000 1000 00012
2's complement:
```

0000 0000 1000 0001	(positive value)
1111 1111 0111 1110	(bitwise negation)
1111 1111 0111 1111	(add one)

ANSWER:

```
1111 1111 0111 1111
```

Instructor's Note: Did you notice that $-129 = -1 - 128$? Can you see how that gets encoded in the binary format?

Example: Problem 1.2(i)

$$1132 = 1024 + 108$$

$$1132 = 1024 + 64 + 44$$

$$1132 = 1024 + 64 + 32 + 12$$

$$1132 = 1024 + 64 + 32 + 8 + 4$$

$$1132_{10} = 0000\ 0100\ 0110\ 1100_2$$

Example: Problem 1.2(j)

$$5555 = 4096 + 1459$$

$$5555 = 4096 + 1024 + 435$$

$$5555 = 4096 + 1024 + 256 + 179$$

$$5555 = 4096 + 1024 + 256 + 128 + 51$$

$$5555 = 4096 + 1024 + 256 + 128 + 32 + 19$$

$$5555 = 4096 + 1024 + 256 + 128 + 32 + 16 + 3$$

$$5555 = 4096 + 1024 + 256 + 128 + 32 + 16 + 2 + 1$$

$$5555_{10} = 0001\ 0101\ 1011\ 0011_2$$

Example: Problem 1.3(a)

a = 0100 0111 0101 1000

b = 1000 0000 1100 0110

Hexadecimal

a = 0x4758_{hex}

b = 0x80c6_{hex}

Octal

a = 043530_{oct}

b = 100306_{oct}

Decimal

For a, the high bit is 0, so the number is positive.

$$a = 2^{14} + 2^{10} + 2^9 + 2^8 + 2^6 + 2^4 + 2^3$$

$$a = 16384 + 1024 + 512 + 256 + 64 + 16 + 8$$

$$a = 18264$$

For b, the high bit is 1, so the number is negative. We convert it to positive before conversion:

b = 1000 0000 1100 0110

0111 1111 0011 1001 (negated)

0111 1111 0011 1010 (plus one)

$$-b = 2^{14} + 2^{13} + 2^{12} + 2^{11} + 2^{10} + 2^9 + 2^8 + 2^5 + 2^4 + 2^3 + 2^1$$

$$-b = 16384 + 8192 + 4096 + 2048 + 1024 + 512 + 256 + 32 + 16 + 8 + 2$$

$$-b = 32570$$

$$b = -32570$$

a+b; check for overflow

```

          1111 1
a:      0100 0111 0101 1000
b:    + 1000 0000 1100 0110
-----
          1100 1000 0001 1110
```

No overflow. Overflow is not possible when we add a positive number to a negative.

a-b; check for overflow

```
b = 1000 0000 1100 0110
    0111 1111 0011 1001  (negated)
    0111 1111 0011 1010  (plus one)
```

```
      1111 111  1111
a:    0100 0111 0101 1000
+    0111 1111 0011 1010
-----
      1100 0110 1001 0010
```

Overflow! We started with a positive value, subtracted a negative (and thus expect positive), but the result was negative.

Example: Problem 1.3(b)

a = 0001 0000 0011 1000
b = 0111 0010 0100 1011

Hexadecimal

a = 0x1038_{hex}
b = 0x724b_{hex}

Octal

a = 010070_{oct}
b = 071113_{oct}

Decimal

For a, the high bit is 0, so the number is positive.

$$a = 2^{12} + 2^5 + 2^4 + 2^3$$
$$a = 4096 + 32 + 16 + 8$$
$$a = 4152$$

For b, the high bit is 0, so the number is positive.

$$b = 2^{14} + 2^{13} + 2^{12} + 2^9 + 2^6 + 2^3 + 2^1 + 2^0$$
$$b = 16384 + 8192 + 4096 + 512 + 64 + 8 + 2 + 1$$
$$b = 29259$$

a+b; check for overflow

```
          11      1111
a:      0001 0000 0011 1000
b:      + 0111 0010 0100 1011
-----
          1000 0010 1000 0011
```

Overflow! We added a positive to a positive, and got a negative.

a-b; check for overflow

```
b = 0111 0010 0100 1011
    1000 1101 1011 0100 (negated)
    1000 1101 1011 0101 (plus one)
```

```
          11
a:   0001 0000 0011 1000
    + 1000 1101 1011 0101
    -----
    1001 1101 1110 1101
```

No overflow. A positive, minus a positive, cannot result in overflow.

Example: Problem 1.3(c)

a = 0000 0000 0110 1100
b = 0000 0001 1010 1001

Hexadecimal

a = 0x006c_{hex}
b = 0x01a9_{hex}

Octal

a = 000154_{oct}
b = 000651_{oct}

Decimal

For a, the high bit is 0, so the number is positive.

$$a = 2^6 + 2^5 + 2^3 + 2^2$$

$$a = 64 + 32 + 8 + 4$$

$$a = 108$$

For b, the high bit is 0, so the number is positive.

$$b = 2^8 + 2^7 + 2^5 + 2^3 + 2^0$$

$$b = 256 + 128 + 32 + 8 + 1$$

$$b = 425$$

a+b; check for overflow

```

                11 11 1
a:      0000 0000 0110 1100
b:      + 0000 0001 1010 1001
-----
                0000 0010 0001 0101
```

No overflow. We added a positive to a positive, and the result was positive.

a-b; check for overflow

```
b = 0000 0001 1010 1001
    1111 1110 0101 0110 (negated)
    1111 1110 0101 0111 (plus one)
```

```
          1111 1
a:      0000 0000 0110 1100
+ 1111 1110 0101 0111
-----
    1111 1110 1100 0011
```

No overflow. A positive, minus a positive, cannot result in overflow.

Example: Problem 1.3(d)

a = 1101 1011 0011 1111
b = 1010 1100 1100 0001

Hexadecimal

a = 0xdb3f
b = 0xacc1

Octal

a = 155477 (octal)
b = 126301 (octal)

Decimal

For both numbers, the MSB is 1, so the number is negative. We convert each to positive before conversion:

a = 1101 1011 0011 1111
 $\sim a$ = 0010 0100 1100 0000
 $-a = \sim a + 1 = 0010 0100 1100 0001$

$-a = 2^{13} + 2^{10} + 2^7 + 2^6 + 2^0$
 $-a = 8192 + 1024 + 128 + 64 + 1$
 $-a = 9409$
 $a = -9409$

b = 1010 1100 1100 0001
 $\sim b$ = 0101 0011 0011 1110
 $-b = \sim b + 1 = 0101 0011 0011 1111$

$-b = 2^{14} + 2^{12} + 2^9 + 2^8 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$
 $-b = 16384 + 4096 + 512 + 256 + 32 + 16 + 8 + 4 + 2 + 1$
 $-b = 21311$
 $b = -21311$

a+b; check for overflow

```
1 1111 1111 1111 111
a:   1101 1011 0011 1111
b: + 1010 1100 1100 0001
-----
    1000 1000 0000 0000
```

No overflow. We added a negative to a negative, and got a negative result.

a-b; check for overflow

We have already calculated the 2's complement of b above:

$$\begin{array}{r} -b = 0101\ 0011\ 0011\ 1111 \\ \\ \begin{array}{r} 1\ 1\ 1\ 11\ 111\ 111 \\ a: \quad 1101\ 1011\ 0011\ 1111 \\ + 0101\ 0011\ 0011\ 1111 \\ \hline 0010\ 1110\ 0111\ 1110 \end{array} \end{array}$$

No overflow. We started with a negative value, and subtracted a negative, so overflow was impossible.

Example: Problem 1.3(e)

a = 0101 0111 1001 0000
b = 1000 1100 0111 0000

Hexadecimal

a = 0x5790
b = 0x8c70

Octal

a = 053620 (octal)
b = 106160 (octal)

Decimal

$a = 2^{14} + 2^{12} + 2^{10} + 2^9 + 2^8 + 2^7 + 2^4$
 $a = 16384 + 4096 + 1024 + 512 + 256 + 128 + 16$
 $a = 22416$

The high bit of b is 1, so the number is negative. We invert and add one to get the positive value.

b = 1000 1100 0111 0000
 $\sim b$ = 0111 0011 1000 1111
 $-b = \sim b + 1 = 0111 0011 1001 0000$

$-b = 2^{14} + 2^{13} + 2^{12} + 2^9 + 2^8 + 2^7 + 2^4$
 $-b = 16384 + 8192 + 4096 + 512 + 256 + 128 + 16$
 $-b = 29584$

a+b; check for overflow

```
      1 1111 111
a:    0101 0111 1001 0000
b:    + 1000 1100 0111 0000
-----
      1110 0100 0000 0000
```

No overflow. We added a negative to a positive, and so no overflow was possible.

a-b; check for overflow

We have already calculated the 2's complement of b above:

```
      111 1111 1
a:    0101 0111 1001 0000
+ 0111 0011 1001 0000
-----
      1100 1011 0010 0000
```

Overflow! We started with a positive value, subtracted a negative, and got a negative result.

Example: Problem 1.3(f)

a = 1001 1000 1100 0001
b = 0100 0111 0111 0110

Hexadecimal

a = 0x98c1
b = 0x4776

Octal

a = 114301\$_8\$
b = 043566\$_8\$

Decimal

The high bit of a is 1, so the number is negative. We invert and add one to get the positive value.

a = 1001 1000 1100 0001
 $\sim a$ = 0110 0111 0011 1110
 $-a = \sim a + 1 = 0110 0111 0011 1111$

$-a = 2^{14} + 2^{13} + 2^{10} + 2^9 + 2^8 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$
 $-a = 16384 + 8192 + 1024 + 512 + 256 + 32 + 16 + 8 + 7 + 4 + 2 + 1$
 $-a = 26431$ $a = -26431$

$b = 2^{14} + 2^{10} + 2^9 + 2^8 + 2^6 + 2^5 + 2^4 + 2^2 + 2^1$
 $b = 16384 + 1024 + 512 + 256 + 64 + 32 + 16 + 4 + 2$
 $b = 18294$

a+b; check for overflow

```
      11 1111 1
a:    1001 1000 1100 0001
b:  + 0100 0111 0111 0110
-----
      1110 0000 0011 0111
```

No overflow. We added a negative to a positive, and so no overflow was possible.

a-b; check for overflow

We calculate the 2's complement of b:

b = 0100 0111 0111 0110
 $\sim b$ = 1011 1000 1000 1001
 $-b = \sim b + 1 = 1011 1000 1000 1010$

Example: Problem 1.3(g)

a = 1101 1000 1000 1010
b = 0010 0101 1101 1101

Hexadecimal

a = 0xd88a
b = 0x25dd

Octal

a = 154212 octal
b = 022735 octal

Decimal

The high bit of a is 1, so the number is negative. We invert and add one to get the positive value.

a = 1101 1000 1000 1010
 \sim a = 0010 0111 0111 0101
 $-a$ = \sim a+1 = 0010 0111 0111 0110

$-a = 2^{13} + 2^{10} + 2^9 + 2^8 + 2^6 + 2^5 + 2^4 + 2^2 + 2^1$
 $-a = 8192 + 1024 + 512 + 256 + 64 + 32 + 16 + 4 + 2$
 $-a = 9216 + 768 + 96 + 22$
 $-a = 9984 + 118$
 $-a = 10102$
 $a = -10102$

$b = 2^{13} + 2^{10} + 2^8 + 2^7 + 2^6 + 2^4 + 2^3 + 2^2 + 2^0$
 $b = 8192 + 1024 + 256 + 128 + 64 + 16 + 8 + 4 + 1$
 $b = 9216 + 384 + 80 + 13$
 $b = 9216 + 477$
 $b = 9693$

a+b

```

              11  11
a:    1101 1000 1000 1010
b:    + 0010 0101 1101 1101
-----
      1111 1110 0110 0111
```

No overflow. We added a positive to a negative, and so no overflow was possible.

a-b

We calculate the 2's complement of b:

```
    b   = 0010 0101 1101 1101
   ~b   = 1101 1010 0010 0010
  -b = ~b+1 = 1101 1010 0010 0011
```

```
      1 1 11           1
a:    1101 1000 1000 1010
-b:  + 1101 1010 0010 0011
-----
      1011 0010 1010 1101
```

No overflow! We started with a negative value, subtracted a positive, and got a negative result.

Example: Problem 1.3(h)

a = 0010 0000 1001 1011
b = 1110 0111 1101 0111

Hexadecimal

a = 0x209b
b = 0xe7d7

Octal

a = 020233 octal
b = 163727 octal

Decimal

$a = 2^{13} + 2^7 + 2^4 + 2^3 + 2^1 + 2^0$
 $a = 8192 + 128 + 16 + 8 + 2 + 1$
 $a = 8192 + 144 + 11$
 $a = 8347$

The high bit of b is 1, so the number is negative. We invert and add one to get the positive value.

b = 1110 0111 1101 0111
 $\sim b$ = 0001 1000 0010 1000
 $-b = \sim b + 1 = 0001 1000 0010 1001$

$-b = 2^{12} + 2^{11} + 2^5 + 2^3 + 2^0$
 $-b = 4096 + 2048 + 32 + 8 + 1$
 $-b = 6144 + 41$
 $-b = 6185$
 $b = -6185$

a+b

```
      1 11   1111   11 111
a:    0010 0000 1001 1011
b:    + 1110 0111 1101 0111
-----
      0000 1000 0111 0010
```

No overflow. We added a negative to a positive, and so no overflow was possible.

a-b

We calculate the 2's complement of b :

```
    b   = 1110 0111 1101 0111
   ~b   = 0001 1000 0010 1000
-b = ~b+1 = 0001 1000 0010 1001
```

```
                111  11
a:      0010 0000 1001 1011
-b:    + 0001 1000 0010 1001
-----
      0011 1000 1100 0100
```

No overflow. We started with a positive value, subtracted a negative, and got a positive result.

Example: Problem 1.3(i)

a = 0101 0011 1010 0001
b = 1001 0111 1001 1111

Hexadecimal

a = 0x53a1
b = 0x979f

Octal

a = 051641 octal
b = 113637 octal

Decimal

$a = 2^{14} + 2^{12} + 2^9 + 2^8 + 2^7 + 2^5 + 2^0$
 $a = 16384 + 4096 + 512 + 256 + 128 + 32 + 1$
 $a = 20480 + 896 + 33$
 $a = 20480 + 929$
 $a = 21409$

The high bit of b is 1, so the number is negative. We invert and add one to get the positive value.

b = 1001 0111 1001 1111
 $\sim b$ = 0110 1000 0110 0000
 $-b = \sim b + 1 = 0110 1000 0110 0001$

$-b = 2^{14} + 2^{13} + 2^{11} + 2^6 + 2^5 + 2^0$
 $-b = 16384 + 8192 + 2048 + 64 + 32 + 1$
 $-b = 24576 + 2048 + 97$
 $-b = 26624 + 97$
 $-b = 26721$
 $b = -26721$

a+b

```

          1 1111 111 111
a:      0101 0011 1010 0001
b:      + 1001 0111 1001 1111
-----
          1110 1011 0100 0000
```

No overflow. We started with a positive value, and added a negative, so overflow was impossible.

a-b

We calculate the 2's complement of b:

```
    b   = 1001 0111 1001 1111
   ~b   = 0110 1000 0110 0000
  -b = ~b+1 = 0110 1000 0110 0001
```

```
           111 11    1
a:      0101 0011 1010 0001
-b:    + 0110 1000 0110 0001
-----
      1011 1100 0000 0010
```

Overflow! We started with a positive value, subtracted a negative, and got a negative result.

Example: Problem 1.4(a)

Problem: Put `clear + creek` in register `$t9`

```
add    $t9, $s5, $s6    # t9 = clear + creek
```

Example: Problem 1.4(b)

Problem: Put `hermit - creek - clear + tonto` in register `$t2`

```
sub    $t2, $s4, $s6    # t2 = hermit - creek
sub    $t2, $t2, $s5    # t2 = hermit - creek - clear
add    $t2, $t2, $s3    # t2 = hermit - creek - clear + tonto
```

Example: Problem 1.4(c)

Problem: Put `hermit + falls` in register `$t1`

```
la    $t1, falls      # t1 = &falls
lw    $t1, 0($t1)     # t1 = falls
add   $t1, $s4, $t1    # t1 = hermit + falls
```

Example: Problem 1.4(d)

Problem: Put `tonto - clear + hermit` in memory location `ribbon`

```
sub    $t0, $s3, $s5    # t0 = tonto - clear
add    $t0, $t0, $s4    # t0 = tonto - clear + hermit
la    $t1, ribbon      # t1 = &ribbon
sw    $t0, 0($t1)     # ribbon = tonto - clear + hermit
```

Example: Problem 1.4(e)

Problem: If (`tonto != hermit`), put `tonto + clear` in register `$s2`

```
beq    $s3, $s4, AFTER_IF # if (tonto == hermit) skip ahead
add    $s2, $s3, $s5      # if (tonto != hermit) s2 = tonto + clear
AFTER_IF:
```

Example: Problem 1.4(f)

Problem: If (`creek >= clear`), put `ribbon + clear` in register `$s2`

```
slt    $t0, $s6, $s5    # t0 = (creek < clear)
bne    $t0, $zero, AFTER_IF # if (creek < clear) skip ahead
la    $t0, ribbon      # t0 = &ribbon
lw    $t0, 0($t0)     # t0 = ribbon
add    $s2, $t0, $s5    # s2 = ribbon + clear
AFTER_IF:
```

Example: Problem 1.4(g)

If `kaibab - tanner == falls - tonto`, put `tonto` in register `$s7`.

```
la    $t0, kaibab      # t0 = &kaibab
lw    $t0, 0($t0)      # t0 = kaibab
la    $t1, tanner      # t1 = &tanner
lw    $t1, 0($t1)      # t1 = tanner
sub   $t2, $t0, $t1    # t2 = kaibab - tanner

la    $t3, falls      # t3 = &falls
lw    $t3, 0($t3)      # t3 = falls
sub   $t4, $t3, $s3    # t4 = falls - tonto

bne   $t2, $t4, AFTER  # if (kaibab - tanner != falls - tonto) jump
add   $s7, $s3, $zero  # s7 = tonto
```

AFTER:

Example: Problem 1.4(h)

Add 10 to `ribbon`, and store the updated value back into the variable.

```
la    $t0, ribbon      # t0 = &ribbon
lw    $t1, 0($t0)      # t1 = ribbon
addi  $t1, $t1, 10     # t1 = ribbon+10
sw    $t1, 0($t0)      # ribbon = ribbon+10
```