CSc 252: Computer Organization
Fall 2018 (Lewis)

HW 1
due at the beginning of lecture on test day

Policy Reminders

• Include your CS username (a.k.a. NetID) on your page. You will lose a few points from your score if you do not include it.

• 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(h-j), 2(g-i), 3(i-k)

Problem 1 - 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.

- 1(a) 107
- 1(b) -3097
- 1(c) 21720
- 1(d) -1
- 1(e) 68
- 1(f) -6143
- 1(g) 937
- 1(h) **Turn in this one:** -129
- 1(i) **Turn in this one:** 1132
- 1(j) **Turn in this one:** 5555
Problem 2 - 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.

2(a)
\[ a = 0100 \ 0111 \ 0101 \ 1000 \]
\[ b = 1000 \ 0000 \ 1100 \ 0110 \]

2(b)
\[ a = 0001 \ 0000 \ 0011 \ 1000 \]
\[ b = 0111 \ 0010 \ 0100 \ 1011 \]

2(c)
\[ a = 0000 \ 0000 \ 0110 \ 1100 \]
\[ b = 0000 \ 0001 \ 1010 \ 1001 \]

2(d)
\[ a = 1101 \ 1011 \ 0011 \ 1111 \]
\[ b = 1010 \ 1100 \ 1100 \ 0001 \]

2(e)
\[ a = 0101 \ 0111 \ 1001 \ 0000 \]
\[ b = 1000 \ 1100 \ 0111 \ 0000 \]

2(f)
\[ a = 1001 \ 1000 \ 1100 \ 0001 \]
\[ b = 0100 \ 0111 \ 0111 \ 0110 \]
2(g) - Turn in this one
   a = 1101 1000 1000 1010
   b = 0010 0101 1101 1101

2(h) - Turn in this one
   a = 0010 0000 1001 1011
   b = 1110 0111 1101 0111

2(i) - Turn in this one
   a = 0101 0011 1010 0001
   b = 1001 0111 1001 1111
Problem 3 - 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

.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)
```

3(a)
Put clear + creek in register $t9

3(b)
Put hermit - creek - clear + tonto in register $t2

3(c)
Put hermit + falls in register $t1

3(d)
Put tonto - clear + hermit in memory location ribbon
3(e)
If (tonto != hermit), put tonto + clear in register $s2$

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

3(g)
If kaibab - tanner == falls - tonto, put tonto in register $s7$.

3(h)
Add 10 to ribbon, and store the updated value back into the variable.

3(i) - Turn in this one
Put tonto+hermit into the variable ribbon.

3(j) - Turn in this one
If (clear == falls), put 1 into register $s3$; otherwise, put 1 into it.

3(k) - Turn in this one
Put tonto*3 + kaibab - creek into register $t7$. 

EXAMPLES
(begin on next page)
Example: Problem 1(a)

\[107 = 64 + 43\]
\[107 = 64 + 32 + 11\]
\[107 = 64 + 32 + 8 + 3\]
\[107 = 64 + 32 + 8 + 2 + 1\]
\[107_{10} = 0000 \ 0000 \ 0110 \ 1011_2\]

Example: Problem 1(b)

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

\[3097 = 2048 + 1049\]
\[3097 = 2048 + 1024 + 25\]
\[3097 = 2048 + 1024 + 16 + 9\]
\[3097 = 2048 + 1024 + 16 + 8 + 1\]

\[3097_{10} = 0000 \ 1100 \ 0001 \ 1001_2\]

2’s complement:

\[
\begin{array}{c}
0000 \ 1100 \ 0001 \ 1001 \ \text{(positive value)} \\
1111 \ 0011 \ 1110 \ 0110 \ \text{(negated)} \\
1111 \ 0011 \ 1110 \ 0111 \ \text{(add one)}
\end{array}
\]

ANSWER:

\[1111 \ 0011 \ 1110 \ 0111\]

Example: Problem 1(c)

\[21720 = 16384 + 5336\]
\[21720 = 16384 + 4096 + 1240\]
\[21720 = 16384 + 4096 + 1024 + 216\]
\[21720 = 16384 + 4096 + 1024 + 128 + 88\]
\[21720 = 16384 + 4096 + 1024 + 128 + 64 + 24\]
\[21720 = 16384 + 4096 + 1024 + 128 + 64 + 16 + 8\]

\[21720_{10} = 0101 \ 0100 \ 1101 \ 1000_2\]

Example: Problem 1(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(e)

Example: Problem 1(f)

Example: Problem 1(g)
Example: Problem 2(a)

\[ a = 0100 \ 0111 \ 0101 \ 1000 \]
\[ b = 1000 \ 0000 \ 1100 \ 0110 \]

Hexadecimal

\[ a = 0x4758_{\text{hex}} \]
\[ b = 0x80c6_{\text{hex}} \]

Octal

\[ a = 043530_{\text{oct}} \]
\[ b = 100306_{\text{oct}} \]

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:

\[
b = 1000 \ 0000 \ 1100 \ 0110
\]
\[
0111 \ 1111 \ 0011 \ 1001 \ \text{(negated)}
\]
\[
0111 \ 1111 \ 0011 \ 1010 \ \text{(plus one)}
\]

\[
-b = 2^{14} + 2^{13} + 2^{12} + 2^{11} + 2^{10} + 2^9 + 2^8 + 2^5 + 2^4 + 2^3 + 2^1
\]
\[
-b = 16384 + 8192 + 4096 + 2048 + 1024 + 512 + 256 + 32 + 16 + 8 + 2
\]
\[
-b = 32570
\]
\[
b = -32570
\]

\[ a + b; \text{ check for overflow} \]

\[
\begin{array}{cccccccc}
  & 1111 & 1 \\
 a: & 0100 & 0111 & 0101 & 1000 \\
 b: & + & 1000 & 0000 & 1100 & 0110 \\
 & \hline \\
 & 1100 & 1000 & 0001 & 1110 \\
\end{array}
\]

\textbf{No overflow.} Overflow is not possible when we add a positive number to a negative.
a−b; check for overflow

\[ b = 1000 \ 0000 \ 1100 \ 0110 \]
\[ 0111 \ 1111 \ 0011 \ 1001 \quad \text{(negated)} \]
\[ 0111 \ 1111 \ 0011 \ 1010 \quad \text{(plus one)} \]
\[ 1111 \ 111 \ 1111 \]

\[ a: \quad 0100 \ 0111 \ 0101 \ 1000 \]
\[ + \ 0111 \ 1111 \ 0011 \ 1010 \]

\[ \text{Overflow!} \quad \text{We started with a positive value, subtracted a negative (and thus expect positive), but the result was negative.} \]
Example: Problem 2(b)

\[
\begin{align*}
a &= 0001 \ 0000 \ 0011 \ 1000 \\
b &= 0111 \ 0010 \ 0100 \ 1011
\end{align*}
\]

Hexadecimal

\[
\begin{align*}
a &= 0x1038_{\text{hex}} \\
b &= 0x724b_{\text{hex}}
\end{align*}
\]

Octal

\[
\begin{align*}
a &= 010070_{\text{oct}} \\
b &= 071113_{\text{oct}}
\end{align*}
\]

Decimal

For \(a\), the high bit is 0, so the number is positive.
\[
\begin{align*}
a &= 2^{12} + 2^5 + 2^4 + 2^3 \\
a &= 4096 + 32 + 16 + 8 \\
a &= 4152
\end{align*}
\]

For \(b\), the high bit is 0, so the number is positive.
\[
\begin{align*}
b &= 2^{14} + 2^{13} + 2^{12} + 2^9 + 2^6 + 2^3 + 2^1 + 2^0 \\
b &= 16384 + 8192 + 4096 + 512 + 64 + 8 + 2 + 1 \\
b &= 29259
\end{align*}
\]

\(a+b\); check for overflow

\[
\begin{array}{c}
11 \\
a: \quad 0001 \ 0000 \ 0011 \ 1000 \\
b: \quad + \ 0111 \ 0010 \ 0100 \ 1011 \\
\hline \\
1000 \ 0010 \ 1000 \ 0011
\end{array}
\]

\textbf{Overflow!} We added a positive to a positive, and got a negative.
a–b; check for overflow

\[
\begin{align*}
 b &= 0111 \ 0010 \ 0100 \ 1011 \\
 &\quad 1000 \ 1101 \ 1011 \ 0100 \quad (\text{negated}) \\
 &\quad 1000 \ 1101 \ 1011 \ 0101 \quad (\text{plus one})
\end{align*}
\]

\[
\begin{align*}
11 \\
a: &\quad 0001 \ 0000 \ 0011 \ 1000 \\
+ &\quad 1000 \ 1101 \ 1011 \ 0101 \\
\hline \\
&\quad 1001 \ 1101 \ 1110 \ 1101
\end{align*}
\]

No overflow. A positive, minus a positive, cannot result in overflow.
Example: Problem 2(c)

\[
\begin{align*}
  a &= 0000\ 0000\ 0110\ 1100 \\
  b &= 0000\ 0001\ 1010\ 1001
\end{align*}
\]

**Hexadecimal**

\[
\begin{align*}
  a &= 0x006c_{\text{hex}} \\
  b &= 0x01a9_{\text{hex}}
\end{align*}
\]

**Octal**

\[
\begin{align*}
  a &= 000154_{\text{oct}} \\
  b &= 000651_{\text{oct}}
\end{align*}
\]

**Decimal**

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

\[
\begin{align*}
  a &= 2^6 + 2^5 + 2^3 + 2^2 \\
  a &= 64 + 32 + 8 + 4 \\
  a &= 108
\end{align*}
\]

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

\[
\begin{align*}
  b &= 2^8 + 2^7 + 2^5 + 2^3 + 2^0 \\
  b &= 256 + 128 + 32 + 8 + 1 \\
  b &= 425
\end{align*}
\]

\( a+b; \) **check for overflow**

\[
\begin{array}{c}
  \begin{array}{c}
    11\ 11\ 1
  \end{array} \\
  \begin{array}{c}
    \begin{array}{c}
      a: \ 0000\ 0000\ 0110\ 1100 \\
      b: \ +\ 0000\ 0001\ 1010\ 1001
    \end{array} \\
  \end{array} \\
  \begin{array}{c}
    \begin{array}{c}
      \hline
      0000\ 0010\ 0001\ 0101
    \end{array} \\
  \end{array}
\end{array}
\]

**No overflow.** We added a positive to a positive, and the result was positive.
\(a-b;\) check for overflow

\[
\begin{align*}
b &= 0000 \ 0001 \ 1010 \ 1001 \\
1111 \ 1110 \ 0101 \ 0110 \quad \text{(negated)} \\
1111 \ 1110 \ 0101 \ 0111 \quad \text{(plus one)}
\end{align*}
\]

\[
\begin{align*}
1111 \ 1 \\
a: \quad 0000 \ 0000 \ 0110 \ 1100 \\
+ \quad 1111 \ 1110 \ 0101 \ 0111 \\
\hline
1111 \ 1110 \ 1100 \ 0011
\end{align*}
\]

No overflow. A positive, minus a positive, cannot result in overflow.
Example: Problem 2(d)

a = 1101 1011 0011 1111
b = 1010 1100 1100 0001
Example: Problem 2(e)

\[ a = 0101 \ 0111 \ 1001 \ 0000 \]
\[ b = 1000 \ 1100 \ 0111 \ 0000 \]
Example: Problem 2(f)

\[ a = 1001 \ 1000 \ 1100 \ 0001 \]
\[ b = 0100 \ 0111 \ 0111 \ 0110 \]

Example: Problem 3(a)

Problem: Put clear + creek in register $t9$

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

Example: Problem 3(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 3(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 3(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 3(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 3(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 3(g)

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

Example: Problem 3(h)

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