

CMPT 295 Assignment 2 Solutions (2%)

1. [6 marks] *Binary Conversions*

(a) [1 mark]

Method 1: Continual Subtraction

$$\begin{array}{r|l|l}
 106 - 64 & = & 42 \\
 42 - 32 & = & 10 \\
 10 - 8 & = & 2 \\
 2 - 2 & = & 0
 \end{array}
 \quad
 \begin{array}{r|l|l}
 128 - 128 & = & 0
 \end{array}
 \quad
 \begin{array}{r|l|l}
 150 - 128 & = & 22 \\
 22 - 16 & = & 6 \\
 6 - 4 & = & 2 \\
 2 - 2 & = & 0
 \end{array}$$

Hex Conversion:

$$\begin{array}{r|l|l}
 106 & = & 64 + 32 + 8 + 2 \\
 & = & 2^6 + 2^5 + 2^3 + 2^1 \\
 & = & 0110\ 1010_2 \\
 & = & 6\ A_{16}
 \end{array}
 \quad
 \begin{array}{r|l|l}
 128 & = & 2^7 \\
 & = & 1000\ 0000_2 \\
 & = & 8\ 0_{16}
 \end{array}
 \quad
 \begin{array}{r|l|l}
 150 & = & 128 + 16 + 4 + 2 \\
 & = & 2^7 + 2^4 + 2^2 + 2^1 \\
 & = & 1001\ 0110_2 \\
 & = & 9\ 6_{16}
 \end{array}$$

Method 2: Continual Division

$$\begin{array}{r|l|l}
 106 \div 2 & = & 53R0 \\
 53 \div 2 & = & 26R1 \\
 26 \div 2 & = & 13R0 \\
 13 \div 2 & = & 6R1 \\
 6 \div 2 & = & 3R0 \\
 3 \div 2 & = & 1R1 \\
 1 \div 2 & = & 0R1 \\
 0 \div 2 & = & 0R0
 \end{array}
 \quad
 \begin{array}{r|l|l}
 128 \div 2 & = & 64R0 \\
 64 \div 2 & = & 32R0 \\
 32 \div 2 & = & 16R0 \\
 16 \div 2 & = & 8R0 \\
 8 \div 2 & = & 4R0 \\
 4 \div 2 & = & 2R0 \\
 2 \div 2 & = & 1R0 \\
 1 \div 2 & = & 0R1
 \end{array}
 \quad
 \begin{array}{r|l|l}
 150 \div 2 & = & 75R0 \\
 75 \div 2 & = & 37R1 \\
 37 \div 2 & = & 18R1 \\
 18 \div 2 & = & 9R0 \\
 9 \div 2 & = & 4R1 \\
 4 \div 2 & = & 2R0 \\
 2 \div 2 & = & 1R0 \\
 1 \div 2 & = & 0R1
 \end{array}$$

... and the binary digits are read in reverse order for each case and subsequently converted to hex using the steps shown in Method 1.

(b) [2 marks] Using the results from part (a), take their 2's complement: flip all the bits and add 1!

$$\begin{array}{lll}
 1 = 0000\ 0001_2 & 106 = 0110\ 1010_2 & 128 = 1000\ 0000_2 \\
 \bar{1} = 1111\ 1110_2 & \bar{106} = 1001\ 0101_2 & \bar{128} = 0111\ 1111_2 \\
 -1 = \bar{1} + 1 = 1111\ 1111_2 & -106 = \bar{106} + 1 = 1001\ 0110_2 & -128 = \bar{128} + 1 = 1000\ 0000_2
 \end{array}$$

(c) [1 mark] *Unsigned:* $1010\ 1110_2 = 2^7 + 2^5 + 2^3 + 2^2 + 2^1 = 128 + 32 + 8 + 4 + 2 = 174_{10}$.

2's Complement: The 2's complement (or negative) of $1010\ 1110_2$ is $\bar{1010\ 1110} + 1 = 0101\ 0001 + 1 = 0101\ 0010_2$. This has a magnitude of $2^6 + 2^4 + 2^1 = 64 + 16 + 2 = 82$. Therefore, $1010\ 1110_2 = -82_{10}$.

(d) [2 marks]

$$\begin{array}{cccccc}
 1100\ 1110 & 0011 & 0111 & 1111\ 1010 & 1010 & 1110 \\
 C & E & 3 & 7 & F & A & A & E \\
 \\
 C^1 & E & & & F^1 & A \\
 + & 3 & 7 & & + & A & E \\
 \hline
 1 & 0 & 5 & & 1 & A & 8
 \end{array}$$

2. [5 marks] `leal`

- (a) [2 marks] Using only one instruction is fairly limiting in that there are only two useful forms:

`leal (, %edi, s), %edi` `leal (%edi, %edi, s), %edi`

where $s \in \{1, 2, 4, 8\}$. Thus the possible values of k are $\{1, 2, 4, 8\}$ in the first case, or $\{2, 3, 5, 9\}$ in the second case. Thus the final list of possible k 's is: $\{1, 2, 3, 4, 5, 8, 9\}$.

- (b) [3 marks] It's possible to feed the result of the first `leal` into the input of the second `leal`.

- $k = 13$

```
leal (%edi, %edi, 8), %eax
leal (%eax, %edi, 4), %edi
```
- $k = 20$

```
leal (%edi, %edi, 4), %edi
leal (, %edi, 4), %edi
```
- $k = 37$

```
leal (%edi, %edi, 8), %eax
leal (%edi, %eax, 4), %edi
```

3. [9 marks] *The Root of the Problem*

```
# The subroutine sqrt finds the integer square root of a 32-bit unsigned
# value, using binary search.
#
# pseudocode: result <- 0
#               for k from 15 downto 0 do:
#                   change the kth bit of result to 1
#                   if result * result > x then:
#                       change the kth bit of result back to 0
#               return result
```

```
sqrt:
    movl    $0, %eax          # result <- 0
    movl    $0x8000, %ecx     # ecx is used to toggle kth bit

loop:
                                # for k from 15 downto 0 do:
    xorl    %ecx, %eax        #   change the kth bit of result to 1
    movl    %eax, %esi
    imul    %esi, %esi
    cmpl    %esi, %edi
    jae     endif            #   if result * result > x then:
    xorl    %ecx, %eax        #       change kth bit back to 0

endif:
    shr     $1, %ecx          #   next k
    jnz     loop

ret
                                # return result
```