University of Stirling
Department of Computing Science and Mathematics

CSCU9V4 Systems

Tutorial 2: Solutions

1. Convert the following decimal numbers into binary and hex, using two's complement notation with 8 bits:

10, -14, -29, -31

Now perform the following operations, in both binary and hexadecimal:

-14 + 10, -29 - 31

Convert the results back to decimal to check you got the right answers.

Answers: 00001010, 11110010, 11100011, 11100001

11110010	11100011
00001010	11100001
11111100 = -4	11000100 = -60

2. Images are often stored as sets of 24 bit values, where each 24 bit value codes the amount of red, green and blue at each picture element, each in 8 bits. Consider a digital photograph which is 1280 by 1024 picture elements in size. How many bytes does it occupy? How many ASCII characters could that hold (assume that each 7-bit character is held in an 8 bit byte)? How many Unicode characters could it hold? (Is a "picture worth a thousand words"?)

Answer:
1280*1024*3 = 3.75Mbytes.
3.75 Mcharacters
3.75/2 M_Unicode Characters.
A picture is worth rather more than 1000 words!

3. A computer used in a control application needs to generate a byte with bits 1 and 6 set, and the others all 0. How might you generate such a byte?
Another part of the application needs to set bit 4 if and only if bit 2 is set, and to clear bit 4 if bit 2 is not set. The other bits of the byte are not to be altered. How might this be achieved using masking and an `if` statement.

X = 01000010b, *or* X = 42x *or* X=66

```
if (X & 00000100b)   // bitwise AND
     X = X | 00010000b ; // bitwise OR
else X = X & 11101111b ; // BITWISE AND
```

Note (i) bitwise operators & | . Note also ^ (bitwise EOR), and ~ (bitwise NOT)

(ii) an expression that evaluates to any nonzero is taken as TRUE, and 0 is FALSE. If in doubt, use

if ((X & 01000010b) != 0) …

4. In the lectures, it was stated that multiplication could be achieved through repeated addition. Clearly, this is very inefficient (consider multiplying 1000 by 1000: you would need to perform 999 additions). Consider how one might do this more efficiently.

- We will multiply 2 positive 8 bit binary integers
- Think of them as binary strings of a particular length (8 bits) and placing the result into a larger (say 16 bit) word (why?)
- Assume that the ALU has a "shift left" operation, which moves all the bits one place to the left, inserting a 0 on the right, and discarding the leftmost bit.

Can you outline an algorithm which will multiply these two binary integers together reasonably efficiently? (Hint: long multiplication) How many additions and shift operations might you require?

(Harder Question) Can you do something similar for division?

Answer: shift and add:

Consider e.g. 1010 * 1100

```
76543210 (bit no)
    1010  0
   1010   0
  1010    1
 1010          1
_________
01111000
```

(Bold numbers show the ones that are added).

Algorithm: (answer = multiplier * multiplicand)
Answer = 0
While multiplicand != 0
    If least significant bit of multiplicand == 1
        Answer = answer + multiplier
    Shift multiplicand 1 place right
    Shift multiplier 1 place left
End

For divide, (Dividend / divisor = quotient + remainder)
Instead of simply subtracting the divisor from the dividend repeatedly, until the result is <0 and then adding it back once (all the time counting, so as to find the quotient), one starts by shifting the divisor left until its most significant bit is directly under the MSB of the dividend. The one does a test subtraction. If the result is –ve one adds the shifted divisor back, and shifts it one place right. If the result is +ve or 0 one puts a 1 in the (least significant bit of the) quotient. Next one shifts the divisor right, shifts the quotient right (inserting a 0 in the LSB), and repeats the process. Eventually, the

divisor is back where it started. After the last test subtraction, whatever is left in the dividend is the remainder.

Example:

```
11001010  / 101

      11001010
    - 101
      ________
      00101010    Quotient = 1
    -  101
      ________
      result -ve
                  Quotient = 10
      00101010
    -   101
      ________
      00000010    Quotient = 101
    -    101
      ________
      result -ve  Quotient = 1010

      00000010
   -      101
      ________
      result -ve  Quotient = 10100

      00000010
   -       101
      ________
      result -ve  Quotient = 101000, remainder = 10
```