CS/MATH–4414 Homework #1
Computer Arithmetic

January 31, 2012

Implement each of the following programs in Fortran or C. Run them. The homework will carefully display the results and fully discuss/explain them.

Problem. Run the following program program several times (with i = 1,2,3,4,5) and explain the results.

```
program test_int
      implicit none
      integer :: m,i
      m = 2147483645
      do i=1,10
        print*,'i=',i,'. m+i=',m+i
      end do
end program test_int
```

Run the program several times, with i = 1,2,3,4,5 and explain the results.

Problem. Run the following program and explain the results.

```
    program test
      real :: a, p
      double precision :: b
      print*, 'please provide p:'
      read*, p
      b = (1.99d0+p)*(2.d0**127)
      print*, b
```

```
      a = b
      print*, a
      print*, 'normal end here !'
    end program test
```

Problem.

```
program test
real :: a=1.0, b=-1.0E+8, c=9.999999E+7
read :: d, r1, r2
  d  = sqrt(b**2-4.0*a*c)
  r1 = (-b+d)/(2.0*a)
  r2 = (-b-d)/(2.0*a)
  print*, r1, r2
end program test
```

The exact results are -1 and $-c$, and we expect the numerical results to be close approximations. Run the code and explain the results. What kind of errors are corrupting the result? Which is the critical stage of the calculation?

Re-run the code with all variables declared double precision. Explain the results.

To overcome the cancellation implement the mathematically equivalent formulas:

$$e_{1,2} = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}} \ .$$

Problem. Consider the Fortran code

```
 program test
   real :: x=100000.0, y=100000.1, z
   z = y-x
   print*, 'z=',z
 end program test
```

We would expect the output

$$Z = 0.1000000$$

What do we get? Explain.

Problem. Run the code and explain the results:

```
program test
  real :: x=12345.6, y=45678.9, z=98765432.1
  real :: w1, w2
  w1 = x*y/z
  w2 = y*(x*(1.0/z))
  print*, w1-w2
end program test
```

Problem. Write a program that computes the smallest positive floating point number (2^{-p}) in single and in double precision. What is p in each case?

Problem. The following F90 code fragment is an attempt to find the ϵ-machine in double precision.

```
   double precision function wlamch( )

      integer   :: i
      double precision  :: sum, eps

      eps = 1.0d0
      do i = 1, 80
          eps = eps*0.5d0
          sum = 1.0 + eps
          if (sum.le.1.0d0) then
             wlamch = eps*2
             return
          end if
      end do
      print*,'error in wlamch. eps < ',eps

   end function wlamch
```

Run the program. The returned accuracy is smaller than expected (in fact, it corresponds to extended precision).

Another implementation (see LAPACK) can be considered along the following lines:

```
double precision function wlamch( )

   integer   :: i
   double precision  :: sum, eps
   double precision, parameter :: one=1.0d0

   eps = one
   do i = 1, 80
       eps = eps*0.5d0
       call wlamch_add(one,eps,sum)
       if (sum.le.one) then
          wlamch = eps*2
          return
       end if
   end do
   print*,'error in wlamch. eps < ',eps

end function wlamch

subroutine wlamch_add( a, b, sum )
   double precision a, b, sum
   sum = a + b
end subroutine wlamch_add
```

Now the reported accuracy is in line with our expectations.

Explain in detail what causes the difference in results between the 2 implementations.