Bit manipulations

Operate on the bits of integers (0,...,31 for 4-byte integer)

Single-bit functions (b=bit#):

`btest(i,b) - .true.` or `.false.`

`ibset(i,b),ibclr(i,b) - integer`

All-bit functions (pair-wise on two integers):

`iand(i,j),ior(i,j),ieor(i,j) - integer`

```
function bits(int)
integer :: i,int
character(32) :: bits
bits='00000000000000000000000000000000'
do i=0,31
  if (btest(int,i)) bits(32-i:32-i)='1'
enddo
end function bits
```

Processor time subroutine

`cpu_time(t)` - `t` = seconds after start of execution

```
integer :: i,nloop
real(8) :: sum
real    :: time0,time1

print*,'Number of operations in each loop’
read*,nloop

sum=0.0d0; call cpu_time(time0)
do i=1,nloop
  sum=sum+dfloat(i)*dfloat(i)
enddo
call cpu_time(time1)
print*,'Time used for s=s+i*i: ',time1-time0
```

Files

- A file has a name on disk, associated unit number in program
- File "connected" by `open` statement

```
open(unit=10,file='a.dat')
```

associates unit 10 with file a.dat

```
open(10,file='a.dat')
```

"`unit`" does not have to be written out

```
open(10,file='a.dat',status='old')
```

'`old`' file already exists ('`new`', '`replace`')

```
open(10,file='a.dat',status='old',access='append')
```

to append existing file with new data

Reading and writing files:

```
read(10,*)a
write(10,*)b
```

Output formatting

```
aa(1)=1; aa(2)=10; aa(3)=100; aa(4)=1000
bb(1)=1.d0; bb(2)=1.d1; bb(3)=1.d2; bb(4)=1.d3
print'(4i5)',aa
write(*,'(4i5)')aa
write(*,10)aa
10 format(4i5)
print'(4i3)',aa
print'(a,i1,a,i2,a,i3)',' one:',aa(1),' ten:',aa(2)
print'(4f12.6)',bb
```

```
    1   10  100 1000
    1   10  100 1000
    1   10  100 1000
  1 10100***
one:1 ten:10
    1.000000   10.000000  100.000000 1000.000000
```

Allocatable arrays

Mechanism to assign the size of an array when running the program (i.e., not fixed when compiling)

```
integer :: m,n
real(8), allocatable :: matr(:,:)

write(*,*)'Give matrix dimensions m,n: ';
read*,m,n
allocate(matr(m,n))
…

deallocate(matr)
```

To change the size of an already allocated array, it first has To be de-allocated, then allocated again.

Variable-sized arrays, interfaces, assumed-shape, and automatic

```
integer :: m,n
real(8), allocatable :: matr(:,:)

Interface                               ! Declaring the interface
  subroutine checkmatr(matr)            ! of a procedure - include in
  real(8) :: matr(:,:)                  ! all procedures that need it,
  end subroutine checkmatr              ! e.g., when using “assumed shape”
end interface

write(*,*)'Give matrix dimensions m,n: '; read*,m,n
allocate(matr(m,n))
call checkmatr(matr)
end

subroutine checkmatr(matr)
real(8) :: matr(:,:)    ! Assumed shape
real(8) :: localmatr(size(matr,1),size(matr,2)) ! “Automatic”
print*,size(localmatr)
print*,shape(localmatr)
end subroutine checkmatr
```

Random number generators

How can deterministic algorithms give random numbers?

- pseudo-random numbers generators

Linear congruential generators; recurrence relation

$$x_{n+1} = \mathrm{mod}(a \cdot x_n + c, m)$$

can generate all numbers 0,...,m-1 in seemingly random order (for suitable a, m, c odd). Test with small m=2^k, c=1:

m=4	a	sequence:
	1	0 1 2 3 0
	2	0 1 3 3 3
	3	0 1 0 1 0
	4	0 1 1 1 1

m=8	a	sequence:
	1	0 1 2 3 4 5 6 7 0
	2	0 1 3 7 7 7 7 7 7
	3	0 1 4 5 0 1 4 5 0
	4	0 1 5 5 5 5 5 5 5
	5	0 1 6 7 4 5 2 3 0
	6	0 1 7 3 3 3 3 3 3
	7	0 1 0 1 0 1 0 1 0
	8	0 1 1 1 1 1 1 1 1

m=16	a	sequence:
	3	0 1 4 13 8 9 12 5 0 1 4 13 8 9 12 5 0
	5	0 1 6 15 12 13 2 11 8 9 14 7 4 5 10 3 0
	11	0 1 12 5 8 9 4 13 0 1 12 5 8 9 4 13 0
	13	0 1 14 7 12 13 10 3 8 9 6 15 4 5 2 11 0

On the computer, integer overflow is a modified modulus 2^{32} (or 2^{64}) operation; can be used for random numbers:

```
n=69069*n+1013904243
ran=0.5d0+n*0.23283064d-9
```

Many systems use this type of intrinsic random number generator

- don't use in serious work (period too short, not random enough)
- 64-bit integer version is quite a good generator

This one is recommended:

```
n=2862933555777941757*n+1013904243
ran=0.5d0+dble(n)*dmul
```

Where `dmul` is precalculated as

```
dmul=1.d0/dble(2*(2_8**62-1)+1)
```

Addition, subtraction can also be used, e.g.,

$$x_{n+1} = \mathrm{mod}(x_{n-3} - x_{n-1}, m) \qquad m = 2^k \text{ - prime}$$

Mixed generators; longer periods, more random, e.g.,

```
mzran=iir-kkr
if (mzran < 0) mzran=mzran+2147483579
iir=jjr; jjr=kkr; kkr=mzran
nnr=69069*nnr+1013904243
mzran=mzran+nnr
rand=0.5d0+mzran*0.23283064d-9
```

Four seeds; `iir,jjr,kkr,nnr`. Period > 10^{28}