Master in HPC

Problem Sheet 1 - Bit manipulation routines

In what follows, to help the students, the exercises are presented with basic pseudo-codes. Herafter a pseudocode is a set of several lines of istructions with the aim of illustrating the general procedure. The solution of the problems requires the conversion of these pseudocodes into a computer program , it is left to the student the choice of the programming language. The pseudocodes presented here will be written in Fortran-style language.

The first exercise is the construction of some bit manipulation routines. The aim of this simple problem is to acquire some familiarity with the kind of bit manipulation which will be used later in the forthcoming exercises.

- Integer variables will be declared as 64-bit integer long. This is not mandatory but it allows more flexibility in the manipulation of the bits
- Hereafter for the bit position we will use the notation $0-63$, with the least significant bit starting from the left. So that if K is an integer with $K = 2^p$ and p=1, the bit representation of K is $K = \{0, 1, \ldots\}$.
- To construct the bit manipulation routines make use of the following bitwise operations:

LOGICAL BITWISE OPERATIONS

bit 1	bit 2	OR(|)	AND($\&$)	XOR(^)
0	0	0	0	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	0

- In addition, it also useful to use the logical operation NOT, which inverts each bit of the variable. So that if $K = \{0, 1, 0, 1, 0, 0, \ldots\}$, then $(.NOT.K) = \{1, 0, 1, 0, 1, 1, \ldots\}$ and so on.

We can now construct the following routines (p is an integer, the function in parenthesis indicates the Fortran equivalent)

- *logical function mybitf(K,p)* : returns $TRUE$ if bit p of K is one, $FALSE$ otherwise (BTEST)
- *long integer function mybitset(K,p)* : set bit p of K to one (IBSET)
- *long integer function mybitclr(K,p)* : set bit p of K to zero (IBCLR)
- *long integer function mybits(K,p,lbit)* : from bit p of K extract $lbit$ bits (IBITS)
- *long integer function mybitshft(K,len)* : move the bits of K to the right($len > 0$) or to the left $(len < 0)$ by len positions (ISHFT)

Write a program which include the above routines and check them by performing the following operations on the variable K

- set $K = 0$

(a) - set bit in position 3 to one ($p = 3, K = mybitset(K, p), K = 8 = \{0, 0, 0, 1, 0, \ldots, 0, 0\}$)

(b) - check the non-zero bits: $mybitf(K, 3) = TRUE, mybitf(K, p) = FALSE,\ p \neq 3$

- set $K = 0$

(c) - now set to one the first three bits : $K = mybitset(K, p), p = 0, 1, 2 - K = 7 = \{1, 1, 1, 0, 0, \ldots\}$

(d) - move them to the right by two positions $K = mybitshft(K, 2) - K = 28 = \{0, 0, 1, 1, 1, 0, 0, \ldots\}$

(e) - is bit 4 = 1 ? $mybitf(K, 4) = TRUE$

(f) - now from position 2 extract three bits $K2 = mybits(K, 2, 3)\ K2 = 7 = \{1, 1, 1, 0, 0, \ldots\}$

(g) - set bit at position 3 to be zero $K = mybitclr(K, 3) - K = 20 = \{0, 0, 1, 0, 1, 0, 0, \ldots\}$

(h) - now perform a logical mask with another integer ($K2 = 16 = \{0, 0, 0, 0, 1, 0, 0, \ldots\} - K3 = 16 = K2.AND.K$

(i) - move the result to the left by three positions $K2 = mybitshft(K2, -3) = 2 = \{0, 1, 0, 0, , 0, \ldots\}$

Print the value of K after each operation.
Here are given the corresponding pseudocodes.

Algorithm 1 Bit manipulation

1: **procedure** BIT ▷

Global:

Require: int $p,\ lbit,\ len,\ nlen$

Require: long int $K,\ K2,\ K3$ ▷ working variables

Require: long int $mybitset,\ mybits,\ mybitclr$ ▷ bit functions

Require: logical $mybif$ ▷ logical bit function

Begin

2: $K := 0$ ▷ set to zero

▷ This function returns the bit length of K: nlen=64, it depends on the language

3: $nlen := BITSIZE(K)$

▷ (a)

4: $p := 3$ ▷ set bit position

5: $K := mybitset(K, p)$ ▷ now K=0,0,0,1,0,...

▷ (b)

6: **for** $p \leftarrow 0, nlen - 1$ **do** ▷ now check the non-zero bits

7: **if** $mybitf(K, p)$ **then**

8: print p

9: **end if**

10: **end for**

▷ (c)

11: $K := 0$ ▷ set to zero

12: **for** $p \leftarrow 0, 2$ **do** ▷ set to one the first three bits

13: $K := mybitset(K, p)$ ▷ now K=1,1,1,0,0,...

14: **end for**

▷ (d)

15: $lbit = 2$ ▷ move to the right by two positions

16: $K := mybitshft(K, lbit)$ ▷ now K=0,0,1,1,1,0,0,...

▷ (e)

17: $p = 4$ ▷ check bit 4

18: print $mybitf(K, p)$

▷ (f)

19: $p = 2$ ▷ extract three bits from bit pos p=2

20: $len = 3$

21: $K2 := mybits(K, p, len)$ ▷ now K2=1,1,1,0,0,0,...

22: print $K2$

▷ (g)

23: $p = 3$ ▷ set bit pos p=3 to zero

24: $K := mybitclr(K, p)$ ▷ now K=0,0,1,0,1,0,0,...

25: print K

▷ (h)

26: $K2 := 16$ ▷ K2=0,0,0,0,1,0,0,0,... ▷ now perform a logical mask with another integer

27: $K3 := K.AND.K2$ ▷ now K3=0,0,0,0,1,0,0,...

28: print $K3$

▷ (i)

29: $lpos = -3$ ▷ move the result to the left by three positions

$K3 = mybitshft(K3, lpos)$ ▷ K3= 0,1,0,0,0,...

30: print $K3$

31: **end procedure**

32: **procedure** MYBITSET(ARG,P)

Require: long int $mybitset,\ arg$

Require: int p

Local:

Require: long int $mask$

33: $mask := 2^p$

34: $mybitset = arg.OR.mask$

35: **end procedure**

1: **procedure** MYBITF(ARG,P)
Require: logical mybitf
Require: long int arg
Require: int p
Local:
Require: long int $mask$
2: $mask := 2^p$
3: $mybitf = (arg.AND.mask) > 0$
4: **end procedure**
5: **procedure** MYBITCLR(ARG,P)
Require: long int $mybitclr,\ arg$
Require: long int arg
Require: int p
Local:
Require: long int $mask,\ maskc$
6: $mask := 2^p$
7: $maskc : (.NOT.mask)$ ▷ bit inverse
8: $mybitclr = arg.AND.maskc$
9: **end procedure**

```
10: procedure MYBITS(ARG,LPOS,LBIT)
Require: long int mybits, arg
Require: int lpos, lbit
  Local:
Require: long int mask, maskc
Require: int p, sumbit
11:     sumbit := 0
12:     for p ← lpos, lpos + lbit − 1 do
13:         mask := 2^p
14:         maskc := 2^(p−lpos)
15:         if ARG.AND.MASK > 0 then
16:             sumbit := sumbit.OR.maskc
17:         end if
18:     end for
19:     mybits := sumbit
20: end procedure
```

```
21: procedure MYBITSHFT(ARG,LPOS)
Require: long int mybitshft, arg
Require: int lpos
  Local:
Require: long int mask, maskc
Require: int p, nb, sumbit
22:     sumbit := 0
23:     nb := BITSIZE(arg)
24:     if lpos < 0 then
25:         for p ← −lpos, nb − 1 do
26:             mask := 2^p
27:             maskc := 2^(p+lpos)
28:             if ARG.AND.MASK > 0 then
29:                 sumbit := sumbit.OR.maskc
30:             end if
31:         end for
32:     else if
        then
33:         for p ← 0, nb − 1 − lpos do
34:             mask := 2^p
35:             maskc := 2^(p+lpos)
36:             if ARG.AND.MASK > 0 then
37:                 sumbit := sumbit.OR.maskc
38:             end if
39:         end for
40:     end if
41:     mybitshft := sumbit
42: end procedure
```