

【解答】

- [設問 1] aー7, bー7
[設問 2] cー4
[設問 3] dー7, eー7, fー7

【解説】

8 ビットのデータ中の指定したビット位置にあるビットの値を検査して結果を返すプログラム、及び8ビットのデータ中にある1のビットの個数を返すプログラムをテーマにした問題である。問題文中にある一つ一つのプログラムは短いもので、プログラムの説明をよく読んで、ビットの論理演算やシフトなどの知識を活用しながら、ビット列の変化をトレースすれば解答できる内容である。また、設問3の処理量の問題については、繰返しが最小になる場合と最大になる場合について考えれば解答できる。問題の難易度はやや易しいといえる。

[設問 1]

最初に、問題文の例を使ってビット検査の方法を確認する。

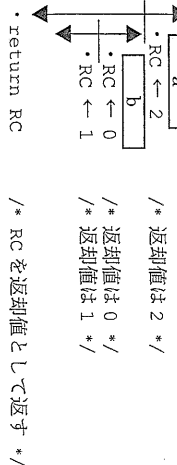
(例 1)			(例 2)		
ビット番号	7 6 5 4 3 2 1 0	ビット番号	7 6 5 4 3 2 1 0		
Data	0 1 0 1 0 1 0 1	Data	0 0 1 1 0 0 1 1		
Mask	1 1 1 0 0 0 0 0	Mask	0 0 0 1 0 0 0 1		
返却値	1	返却値	2		

(例 1) では、Mask のビット番号 7～5 の 3 ビットが 1 なので、Data のこの 3 ビットの値を検査する。そして、検査対象のビットには 0 と 1 が混在しているので、返却値は 1 となる。また、(例 2) では、Mask のビット番号 4 と 0 が 1 なので、Data のビット番号 4 と 0 の値を検査するが、どちらのビットも 1 なので返却値は 2 となる。解答群の内容を見ると、この問題では、Data (検査データ) と Mask (検査するビット位置が 1, 他のビットが 0: マスクビット) の論理積 (AND) が論理和 (OR) を求めて、対象となるビットの値を検査しているようである。どちらの演算なのかを考えると、(例 1), (例 2) の Data と Mask の論理積、論理和を求めると次のようになる。

(例 1)			(例 2)		
ビット番号	7 6 5 4 3 2 1 0	ビット番号	7 6 5 4 3 2 1 0		
Data	0 1 0 1 0 1 0 1	Data	0 0 1 1 0 0 1 1		
Mask	1 1 1 0 0 0 0 0	Mask	0 0 0 1 0 0 0 1		
論理積	0 1 0 0 0 0 0 0	論理積	0 0 0 1 0 0 0 1		
論理和	1 1 1 1 0 1 0 1	論理和	0 0 1 1 0 0 1 1		

この結果から分るように、論理積の場合は、Data 中のビット列から、Mask の値が 1 になっているビット位置の値だけがそのまま取り出され、その他のビットは全て 0 となる。一方、論理和の場合は、Mask の値が 0 になっているビット位置の値だけがそのまま取り出され、その他のビットは全て 1 となってしまう。このことから、あるビット列から特定のビット位置の値だけを取り出すには、検査対象のビット列と対応するビットに 1, それ以外のビットに 0 を設定したマスクビットとの論理積 (AND) を求めればよいことが分かる。

[プログラム 1]
○整数型関数: BitTest (8 ビット論理型: Data, 8 ビット論理型: Mask)
○整数型: RC
/* 返却値 */

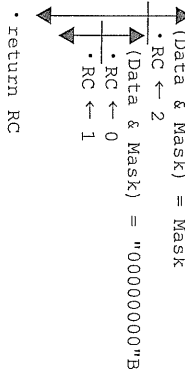


- 空欄 a: 空欄 a が真の場合は返却値に 2 が設定されているので、空欄 a では Data 中の検査したビットが全て 1 であることを判断できる条件を考える。前述の説明にもあるように、検査するビットの値だけを取り出すためには、検査対象のデータ (Data) とマスクビット (Mask) との論理積を求めると、その結果、検査したビットだけが 1, それ以外のビットは全て 0 であったときに、検査したビットが全て 1 であったと判断できる。そして、それは結果が Mask と同じ値ということである。したがって、条件式は (Data & Mask) = Mask と表現でき、(ア) が正解である。
- 空欄 b: 空欄 b が真の場合は返却値に 0 (検査した全てのビットが 0), 偽の場合は返却値に 1 (検査したビット中に 0 と 1 が混在) を設定している。前述の説明にもあるように、Mask と Data の論理積を求めると、検査対象のビットはそのままの値、対象外のビットは全て 0 になるので、検査対象のビットが全て 0 の場合の演算結果は "00000000"B に、1 が含まれる場合には、それ以外の値になる。したがって、空欄 b の条件式は (Data & Mask) = "00000000"B と表現でき、(イ) が正解である。

[設問 2]

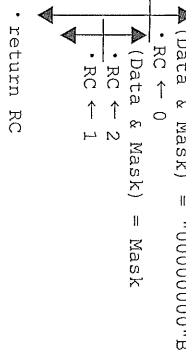
プログラム 1 では、Mask 中に 1 のビットが 1 個以上あることを前提としていたが、この前提を取り除いて、Mask 中の 1 のビットが 0 個の場合は、返却値 0 を返すようにプログラムを変更する。この変更のための修正案は①～③の三つあり、適切なものを選択する。なお、修正案中の空欄 a, b は、それぞれ設問 1 で解答した“(Data & Mask) = Mask”, “(Data & Mask) = "00000000"B”が入ることに注意する。
Mask 中の 1 のビットが 0 個の場合に Data と Mask の論理積を求めると、演算結果のビット列は全て 0 になるため、その判定条件は (Data & Mask) = "00000000"B と表現できる。また、Mask 自体のビット列が "00000000"B であるため、(Data & Mask) = Mask と表現できる。このことを基に修正案を順番に検証していく。

修正案① (変更なし)



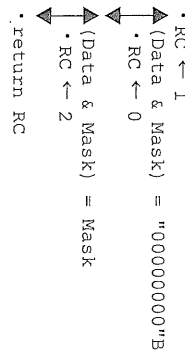
修正案①では、Mask のビット列が全て 0 の場合、空欄 a ((Data & Mask) = Mask) が真となるので、本来は返却値として 0 を返すべきところを 2 を返すことになり、正しい結果とはならない。

修正案②



修正案②では、最初の判定条件が“(Data & Mask) = "000000000"B”なので、Mask のビット列が全て 0 の場合には返却値が 0 となる。また、検査した全てのビットが 0 の場合もこの条件が成り立つので、返却値 0 に関する判定は正しく行うことができる。そして、この条件が偽の場合には、続いて“(Data & Mask) = Mask”の判定を行うが、その結果は、検査したビットが全て 1 の場合は Mask と同じビット列に、0 と 1 が混在する場合は Mask と異なるビット列になるため、真のときに 2, 偽のときに 1 と正しい返却値を返している。したがって、正しく動作する。

修正案③



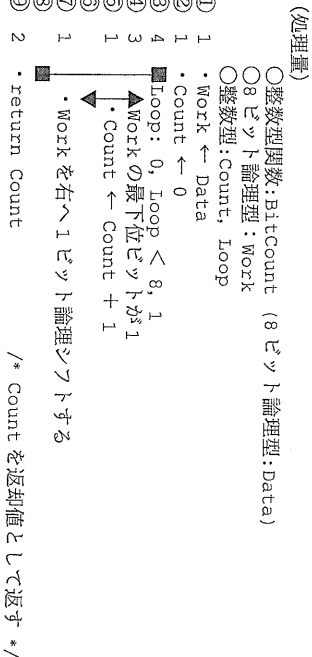
修正案③では、最初の判定条件である“(Data & Mask) = "000000000"B”が真となった場合にも、続けて“(Data & Mask) = Mask”の判定を行うことになる。修正案①の説明にもあるように、Mask 中の 1 のビットが 0 個の場合にはこの二つの判定条件はどちらも真となるため、結果として返却値が 2 となってしまう、正しい結果とはならない。

以上の内容から、正しく動作するのは修正案②だけであることが分かり、空欄 c は (イ) が正解である。

[設問 3]

プログラム 2, 3 の処理効率について考える。

[プログラム 2]



※○付き数字は行番号を表す。

- 空欄 d: 整数型関数 BitCount は、8 ビットのデータ中にある 1 のビットの個数を返す。プログラム 2 は、最下位ビットが 1 かどうかを判断することによって 1 のビットの個数をカウントするものであるが、ビットの判定の都度、ビット列を 1 ビットずつ右へシフトすることで、ビット番号 2, ビット番号 3, ..., ビット番号 7 を順番に最下位ビットへ移動させ、8 ビット全てについての判定を行う。そのため、行番号③の繰返しの判断は 9 回 (Loop は 0 から始まり Loop=8 で繰返し処理を抜ける), 行番号④, ⑦の処理は 8 回実行されるので、実行回数が最小になるのは Work のビットが全て 0 の場合、逆に最大になるのは Work のビットが全て 1 の場合であり、実行回数はそれぞれ 0 回, 8 回となる。これを整理すると、最小、最大の処理量は次のようになる。

行番号	処理量	最小の処理量	最大の処理量
①	1	1×1 回=1	1×1 回=1
②	1	1×1 回=1	1×1 回=1
③	4	4×9 回=36	4×9 回=36
④	3	3×8 回=24	3×8 回=24
⑤	1	1×0 回=0	1×8 回=8
⑦	1	1×8 回=8	1×8 回=8
⑨	2	2×1 回=2	2×1 回=2
	合計	72	合計 80

【解答】

- [設問 1] aー7, bー7
[設問 2] cー4
[設問 3] dー7, eー7, fー7

【解説】

8 ビットのデータ中の指定したビット位置にあるビットの値を検査して結果を返すプログラム、及び8ビットのデータ中にある1のビットの個数を返すプログラムをテーマにした問題である。問題文中にある一つ一つのプログラムは短いもので、プログラムの説明をよく読んで、ビットの論理演算やシフトなどの知識を活用しながら、ビット列の変化をトレースすれば解答できる内容である。また、設問3の処理量の問題については、繰返しが最小になる場合と最大になる場合について考えれば解答できる。問題の難易度はやや易しいといえる。

[設問 1]

最初に、問題文の例を使ってビット検査の方法を確認する。

(例 1)			(例 2)		
ビット番号	7 6 5 4 3 2 1 0	ビット番号	7 6 5 4 3 2 1 0		
Data	0 1 0 1 0 1 0 1	Data	0 0 1 1 0 0 1 1		
Mask	1 1 1 0 0 0 0 0	Mask	0 0 0 1 0 0 0 1		
返却値	1	返却値	2		

(例 1) では、Mask のビット番号 7～5 の 3 ビットが 1 なので、Data のこの 3 ビットの値を検査する。そして、検査対象のビットには 0 と 1 が混在しているので、返却値は 1 となる。また、(例 2) では、Mask のビット番号 4 と 0 が 1 なので、Data のビット番号 4 と 0 の値を検査するが、どちらのビットも 1 なので返却値は 2 となる。解答群の内容を見ると、この問題では、Data (検査データ) と Mask (検査するビット位置が 1, 他のビットが 0: マスクビット) の論理積 (AND) が論理和 (OR) を求めて、対象となるビットの値を検査しているようである。どちらの演算なのかを考えるために、(例 1), (例 2) の Data と Mask の論理積、論理和を求めると次のようになる。

(例 1)			(例 2)		
ビット番号	7 6 5 4 3 2 1 0	ビット番号	7 6 5 4 3 2 1 0		
Data	0 1 0 1 0 1 0 1	Data	0 0 1 1 0 0 1 1		
Mask	1 1 1 0 0 0 0 0	Mask	0 0 0 1 0 0 0 1		
論理積	0 1 0 0 0 0 0 0	論理積	0 0 0 1 0 0 0 1		
論理和	1 1 1 1 0 1 0 1	論理和	0 0 1 1 0 0 1 1		

この結果から分るように、論理積の場合は、Data 中のビット列から、Mask の値が 1 になっているビット位置の値だけがそのまま取り出され、その他のビットは全て 0 となる。一方、論理和の場合は、Mask の値が 0 になっているビット位置の値だけがそのまま取り出され、その他のビットは全て 1 となってしまう。このことから、あるビット列から特定のビット位置の値だけを取り出すには、検査対象のビット列と対応するビットに 1, それ以外のビットに 0 を設定したマスクビットとの論理積 (AND) を求めればよいことが分かる。

【プログラム 1】

○整数型関数: BitTest (8 ビット論理型: Data, 8 ビット論理型: Mask)

○整数型: RC

RC ← 2

a

b

RC ← 0

RC ← 1

/* 返却値は 2 */

/* 返却値は 0 */

/* 返却値は 1 */

• return RC

/* RC を返却値として返す */

- 空欄 a: 空欄 a が真の場合は返却値に 2 が設定されているので、空欄 a では Data 中の検査したビットが全て 1 であることを判断できる条件を考える。前述の説明にもあるように、検査するビットの値だけを取り出すためには、検査対象のデータ (Data) とマスクビット (Mask) との論理積を求めると、その結果、検査したビットだけが 1, それ以外のビットは全て 0 であったときに、検査したビットが全て 1 であったと判断できる。そして、それは結果が Mask と同じ値ということである。したがって、条件式は (Data & Mask) = Mask と表現でき、(ア) が正解である。
- 空欄 b: 空欄 b が真の場合は返却値に 0 (検査した全てのビットが 0), 偽の場合は返却値に 1 (検査したビット中に 0 と 1 が混在) を設定している。前述の説明にもあるように、Mask と Data の論理積を求めると、検査対象のビットはそのままの値、対象外のビットは全て 0 になるので、検査対象のビットが全て 0 の場合の演算結果は "00000000"B に、1 が含まれる場合には、それ以外の値になる。したがって、空欄 b の条件式は (Data & Mask) = "00000000"B と表現でき、(イ) が正解である。

[設問 2]

プログラム 1 では、Mask 中に 1 のビットが 1 個以上あることを前提としていたが、この前提を取り除いて、Mask 中の 1 のビットが 0 個の場合は、返却値 0 を返すようにプログラムを変更する。この変更のための修正案は①～③の三つあり、適切なものを選択する。なお、修正案中の空欄 a, b は、それぞれ設問 1 で解答した“(Data & Mask) = Mask”, “(Data & Mask) = "00000000"B”が入ることに注意する。
Mask 中の 1 のビットが 0 個の場合に Data と Mask の論理積を求めると、演算結果のビット列は全て 0 になるため、その判定条件は (Data & Mask) = "00000000"B と表現できる。また、Mask 自体のビット列が "00000000"B であるため、(Data & Mask) = Mask と表現できる。このことを基に修正案を順番に検証していく。

修正案① (変更なし)

(Data & Mask) = Mask

RC ← 2

(Data & Mask) = "00000000"B

RC ← 0

(Data & Mask) = Mask

RC ← 1

• return RC

修正案①では、Mask のビット列が全て 0 の場合、空欄 a ((Data & Mask) = Mask) が真となるので、本来は返却値として 0 を返すべきところを 2 を返すことになり、正しい結果とはならない。

修正案②

(Data & Mask) = "00000000"B

RC ← 0

(Data & Mask) = Mask

RC ← 2

(Data & Mask) = Mask

RC ← 1

• return RC

修正案②では、最初の判定条件が“(Data & Mask) = "00000000"B”なので、Mask のビット列が全て 0 の場合には返却値が 0 となる。また、検査した全てのビットが 0 の場合もこの条件が成り立つので、返却値 0 に関する判定は正しく行うことができる。そして、この条件が偽の場合には、続いて“(Data & Mask) = Mask”の判定を行うが、その結果は、検査したビットが全て 1 の場合は Mask と同じビット列に、0 と 1 が混在する場合は Mask と異なるビット列になるため、真のときに 2, 偽のときに 1 と正しい返却値を返している。したがって、正しく動作する。

修正案③

RC ← 1

(Data & Mask) = "00000000"B

RC ← 0

(Data & Mask) = Mask

RC ← 2

• return RC

修正案③では、最初の判定条件である“(Data & Mask) = "00000000"B”が真となった場合にも、続けて“(Data & Mask) = Mask”の判定を行うことになる。修正案①の説明にもあるように、Mask 中の 1 のビットが 0 個の場合にはこの二つの判定条件はどちらも真となるため、結果として返却値が 2 となってしまう、正しい結果とはならない。

以上の内容から、正しく動作するのは修正案②だけであることが分かり、空欄 c は (イ) が正解である。

[設問 3]

プログラム 2, 3 の処理効率について考える。

【プログラム 2】

(処理量)

○整数型関数: BitCount (8 ビット論理型: Data)

○8 ビット論理型: Work

○整数型: Count, Loop

1

Count ← 0

Loop: 0, Loop < 8, 1

3

Work の最下位ビットが 1

1

Count ← Count + 1

1

Work を右へ 1 ビット論理シフトする

8

2

return Count

/* Count を返却値として返す */

※○付き数字は行番号を表す。

- 空欄 d: 整数型関数 BitCount は、8 ビットのデータ中にある 1 のビットの個数を返す。プログラム 2 は、最下位ビットが 1 かどうかを判断することでのビットの個数をカウントするものであるが、ビットの判定の都度、ビット列を 1 ビットずつ右へシフトすることで、ビット番号 2, ビット番号 3, ..., ビット番号 7 を順番に最下位ビットへ移動させ、8 ビット全てについての判定を行う。そのため、行番号③の繰返しの判断は 9 回 (Loop は 0 から始まり Loop = 8 で繰返し処理を抜ける), 行番号④, ⑦の処理は 8 回実行されるので、実行回数が最小になるのは Work のビットが全て 0 の場合、逆に最大になるのは Work のビットが全て 1 の場合であり、実行回数はそれぞれ 0 回, 8 回となる。これを整理すると、最小、最大の処理量は次のようになる。

行番号	処理量	最小の処理量	最大の処理量
①	1	1×1 回=1	1×1 回=1
②	1	1×1 回=1	1×1 回=1
③	4	4×9 回=36	4×9 回=36
④	3	3×8 回=24	3×8 回=24
⑤	1	1×0 回=0	1×8 回=8
⑦	1	1×8 回=8	1×8 回=8
⑨	2	2×1 回=2	2×1 回=2
	合計	72	合計 80