

## 情報実験第三課題 1.A

情報工学科 15\_03602 柿沼 建太郎

情報工学科 15\_10588 中田 光

平成 29 年 5 月 9 日

# 各課題担当者

各課題と担当者を表として以下に示す。

課題番号/名前	柿沼	中田
1	○	○
2	○	○
3	○	○
4	○	○
5(test_io1)		○
5(test_io2)		○
5(test_calc1)	○	

# 課題プログラムレポート(1,2,3,4)

## 倍精度乗算

柿沼

流れについての説明

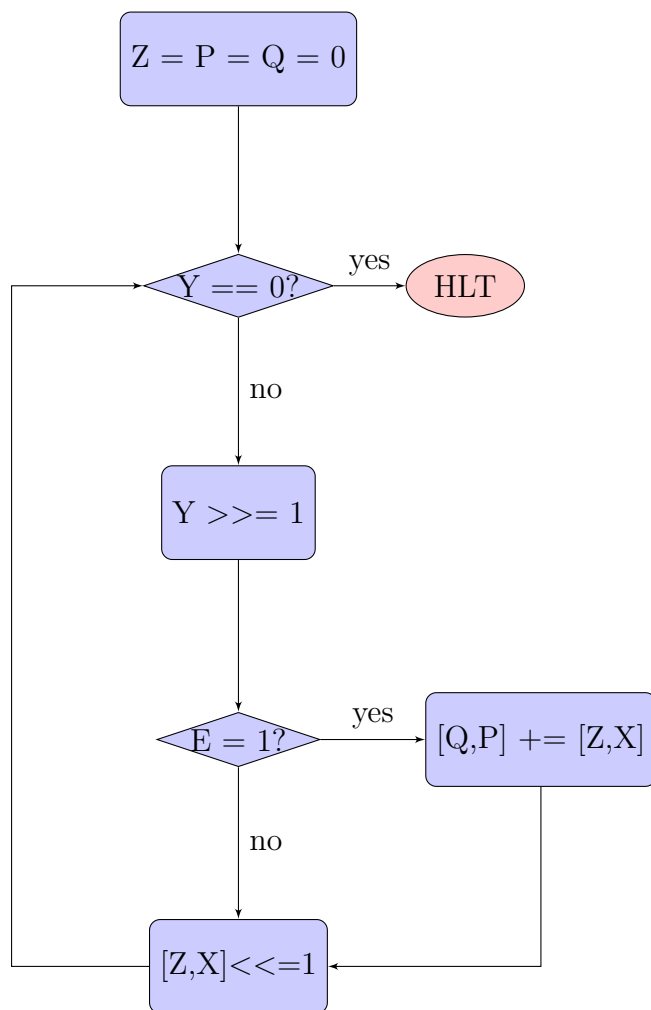
使用した変数について, 以下に示す。

変数名	説明
X	乗算の対象
Y	乗算の対象
Z	乗算の対象 (X の拡張用)
P	乗算の結果 (下位 16bit)
Q	乗算の結果 (上位 16bit)

乗算は筆算式に実現した。Y を右シフトしながら X を左シフトしていき、Y の末尾が 1 だった場合にはその時の X の値を結果に加算していく。このとき、X は 16 回左にシフトするが、結果は倍精度とするためそのままでは上位 16bit の情報が失われてしまう。

そこで、新しく Z という変数を使い [Z,X] を 32bit の変数として扱うことで倍精度計算を実現した。

以下にフローチャートを示す。



### 工夫点

$[Q,P] += [Z,X]$  を実現する際、P と X の加算によってあふれた bit が E レジスタに入ることを利用し、加算の後 E レジスタの値が 1 だった場合には 1 回 INC することで実現した。

### ソースコードと総命令数

Listing 1: report1.1.asm

```

1 | ORG 10
2 | / Y == 0 ? HLT : goto LY

```

```

3  L0,
4    CLE
5    LDA Y
6    SZA
7    BUN LY
8    HLT
9    / Y >>= 1, E == 0 ? goto LX : goto LP
10 LY,
11   CIR
12   STA Y
13   SZE
14   BUN LP
15   / [Z,X] <<= 1, goto L0
16 LX,
17   LDA X
18   CIL
19   STA X
20   LDA Z
21   CIL
22   STA Z
23   BUN L0
24   / [Q,P] += [Z,X]
25 LP,
26   LDA X
27   ADD P
28   STA P
29   LDA Z
30   SZE
31   INC
32   ADD Q
33   STA Q
34   CLE
35   BUN LX
36 / data
37 X, DEC 65535
38 Y, DEC 65535
39 Z, DEC 0
40 P, DEC 0
41 Q, DEC 0
42 END

```

総命令数:26

入力 ( $X \times Y$ )	ステップ数
$11 \times 13$	90
$0 \times 30$	114
$30 \times 0$	4
$65535 \times 65535$	403

## 異なる入力に対する実行命令ステップ数

### EX3 命令セットで改良すべき点

桁あふれしたときに E レジスタの中身を変えてくれるような INC 命令が欲しいと感じた。それを別の形で実現しようと考え、1 を ADD する方法も考えたが、即値が扱えないためそれをするのにも手間がかかる。E レジスタの中身を変える INC がなくても、即値を扱える機構さえあれば良いと思う。

## 中田

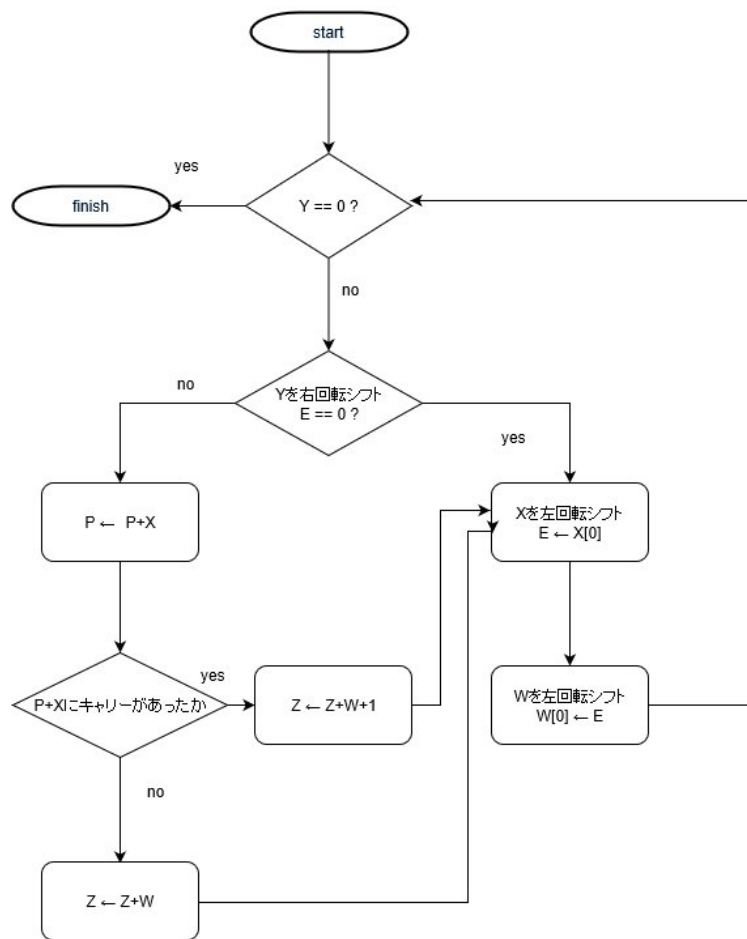
### 流れについての説明

使用した変数について、以下に示す。

変数名	説明
X	乗算の対象
W	乗算の対象 (X の拡張用)
Y	乗算の対象
P	乗算の結果 (下位 16bit)
Z	乗算の結果 (上位 16bit)

乗算は筆算法によって計算している。入力は共に 16bit 符号なし整数である。また、16bit の X を 32bit の [W;X] に拡張することで倍精度計算を行っている。Y を右回転シフトし E レジスタが 1 の場合は、[Z,P] に [W;X] を加算する。そして、[W;X] を左回転シフトする。これを Y が 0 になるまで繰り返し、筆算法を実現している。

以下にフローチャートを示す。



## 工夫点

$[P;Z] += [W;X]$  において、 $Z += X$  の繰り上がりが E レジスタに格納されることを利用して計算した。 $Z += X$  の計算後、 $P += Z+E$  を計算している。

## ソースコードと総命令数

Listing 2: report1\_1\_nakata.asm

```

1      ORG 10          / program entry point
2  / check M[Y] == 0 ?
3  L0,
4      CLE              / E <- 0
5      LDA Y            / AC <- M[Y]
6      SZA              / (AC == 0) ? skip next
7      BUN LY          / goto LY
8      HLT
9  / M[Y] >>= 1
10 LY,
11     CIR              / {AC[15:0], E} <- {E, AC[15:0]}
12     STA Y            / M[Y] <- AC
13     SZE              / (E == 0) ? skip next
14     BUN LP          / goto LP
15 / M[X] <<= 1
16 LX,
17     CLE              / E <- 0
18     LDA X            / AC <- M[X]
19     CIL              / {E, AC[15:0]} <- {AC[15:0], E}
20     STA X            / M[X] <- AC
21     LDA W            / AC <- M[W]
22     CIL              / {E, AC[15:0]} <- {AC[15:0], E}
23     STA W            / M[W] <- AC
24     BUN L0          / goto L0
25 / {M[Z], M[P]} += {M[W], M[X]}
26 LP,
27     CLE
28     LDA X            / AC <- M[X]
29     ADD P            / AC <- AC + M[P]
30     STA P            / M[P] <- AC
31     LDA W            / AC <- M[W]
32     SZE              / (E == 0) ? skip next
33     INC              / ++AC
34     ADD Z            / AC <- AC + M[Z]
35     STA Z            / M[Z] <- M[Z]
36     BUN LX          / goto LX
37
38 / data
39 X,    DEC 65535      / X = 65535 -> {W,X}:init
40 W,    DEC 0          / (init : 0)
41 Y,    DEC 65535      / Y = 65535 -> Y:init
42 P,    DEC 0          / (init : 0) {Z,P} = {W,X} * Y : result
43 Z,    DEC 0
44 END

```

総命令数:28



異なる入力に対する実行命令ステップ数

入力 ( $X \times Y$ )	ステップ数
$11 \times 13$	94
$0 \times 30$	119
$30 \times 0$	4
$65535 \times 65535$	419

EX3 命令セットで改良すべき点

特になし。

## 剰余算

柿沼

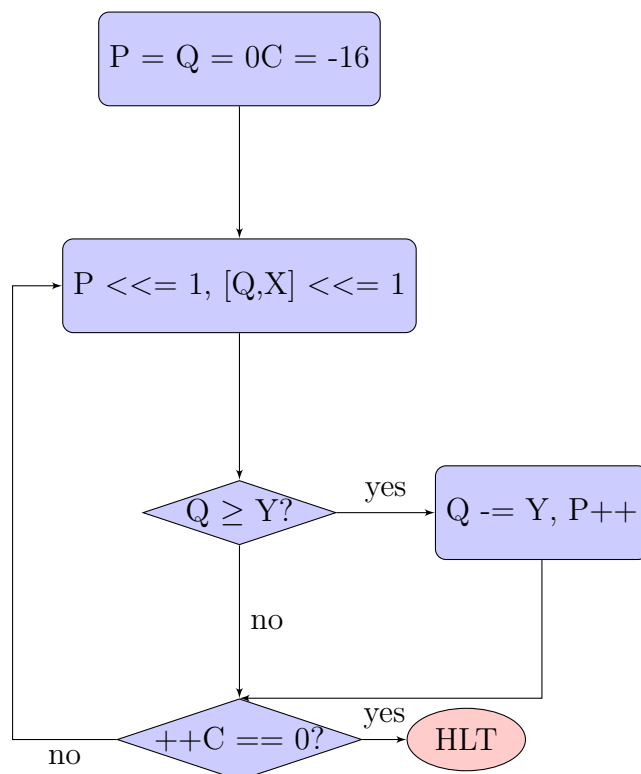
流れについての説明

使用した変数について, 以下に示す。

変数名	説明
X	徐算の対象
Y	徐算の対象
P	徐算の商
Q	徐算の余り
C	徐算の余り

徐算は筆算式に実現した。 $[Q, X]$  を左シフトしながら商も左シフトしていき、Q に溜まった数が Y 以上となったときに Q から Y を引き、商の末尾ビットを立てる。これを 16bit ぶん繰り返した。

以下にフローチャートを示す。



## 工夫点

$Q \geq Y$  の部分で  $Q - Y$  を計算するため、それをそのまま  $Q -= Y$  に転用した。

$Q \geq Y$  の計算は符号なし 16bit 整数として計算しなければならないため、E レジスタを含めた符号あり 17bit と考えて、適切な計算の後 SZE で判定するという手法をとった。

Y を E レジスタを含めた 17bit 符号あり整数と考えた場合、符号反転後はほとんどの場合最上位ビット (E レジスタ) が立っていることになるが、唯一 Y が 0 のときのみ符号を反転しても E レジスタは 0 のままなため、反転前の Y に対して SZA で E レジスタの値を定めた。

## ソースコードと総命令数

Listing 3: report1.2.asm

```
1 | ORG 10
```

```

2 / P<<= 1, [Q,X] <<= 1
3 LD0,
4   CLE
5   LDA P
6   CIL
7   STA P
8   LDA X
9   CIL
10  STA X
11  LDA Q
12  CIL
13  STA Q
14 / Q >= Y ? goto LD1 : goto LD2
15  LDA Y
16  CLE
17  SZA
18  CME
19  CMA
20  INC
21  ADD Q
22  SZE
23  BUN LD1
24  BUN LD2
25 / Q -= Y, P++
26 LD1,
27  STA Q
28  LDA P
29  INC
30  STA P
31 / ++C == 0 ? HLT : goto LD0
32 LD2,
33  ISZ C
34  BUN LD0
35  HLT
36 / data
37 X, DEC 65535
38 Y, DEC 65535
39 P, DEC 0
40 Q, DEC 0
41 C, DEC -16
42 END

```

総命令数:28

入力 ( $X \div Y$ )	ステップ数
$30 \div 7$	340
$0 \div 15$	336
$65535 \div 1$	400
$65535 \div 65535$	340

異なる入力に対する実行命令ステップ数

EX3 命令セットで改良すべき点

EX3の命令セットでは、スキップ命令が直感に反するものが多いと感じたため、スキップ条件が逆の命令セットがあると書きやすくなると思う。

中田

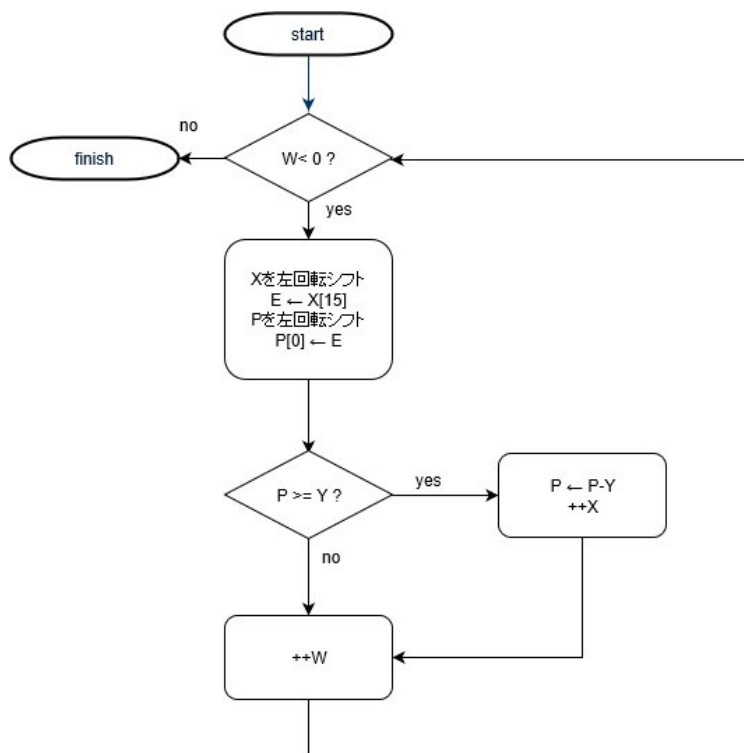
流れについての説明

使用した変数について, 以下に示す。

変数名	説明
X	徐算の対象 計算後、剰余の商を出力
Y	徐算の対象
P	剰余の余り
W	カウントする変数

徐算は筆算式によって計算している。 $[P;X]$  を左回転シフトし、P が Y 以上になったら  $P-Y$  をし、X に商のビットを立てる。これを 16 回繰り返す。

以下にフローチャートを示す。



## 工夫点

$P \geq Y$  を判定する際に  $P-Y$  の正負の判定を用いた。  
 ここで、 $P-Y$  を SNA によって判定すると 16bit 符号あり整数として扱ってしまうため誤りが生じる。  
 そのため、E レジスタを 17bit 目とすることで、16bit 符号なし整数として計算を行った。E が 1 の時  $P-Y$  を正と判定し、0 の時は負と判定する。

## ソースコードと総命令数

Listing 4: report1\_2\_nakata.asm

```

1      ORG 10          / program entry point
2  / count 16 W: -16 --> 0
3  L0,
4      LDA W          / AC ← M[W]
5      SNA             / (AC < 0) ? next skip
6      HLT
7  / main loop

```

```

8  L1,
9      CLE      / E <- 0
10     CLA      / AC <- 0
11     LDA X    / AC <- M[X]
12     CIL      / {E, AC[15:0]} <- {AC[15:0], E}
13     STA X    / M[X] <- AC
14     CLA      / AC <- 0
15     LDA P    / AC <- M[P]
16     CIL      / {E, AC[15:0]} <- {AC[15:0], E}
17     STA P    / M[P] <- AC
18     LDA Y    / AC <- M[Y]
19     CMA      / AC <- ~AC
20     INC      / ++AC
21     ADD P    / AC <- AC + M[P]
22     SZE      / (E == 0) ? skip next
23     BUN L2   / goto L2
24 / ++M[W]
25 L3,
26     CLA      / AC <- 0
27     LDA W    / AC <- M[W]
28     INC      / ++AC
29     STA W    / M[W] <- AC
30     CLA      / AC <- 0
31     BUN L0   / goto L0
32 / P - Y >= 0 --> ++M[X]
33 L2,
34     STA P    / M[P] <- AC
35     CLA      / AC <- 0
36     LDA X    / AC <- M[X]
37     INC      / ++AC
38     STA X    / M[X] <- AC
39     BUN L3   / goto L3
40 / data
41 X,      DEC 65535      / X = 65535 --> X:init
--> X = X / Y : result
42 Y,      DEC 65535      / Y = 65535 --> Y:init
43 P,      HEX 0          / (init : 0) P = X mod Y : result
44 W,      DEC -16 / counter (init : -16)
45 END

```

総命令数:30

入力 ( $X \div Y$ )	ステップ数
$30 \div 7$	362
$0 \div 15$	336
$65535 \div 1$	467
$65535 \div 65535$	362

異なる入力に対する実行命令ステップ数

EX3 命令セットで改良すべき点

INC 命令によってカウントするために W を-16 としたが、負の数を用いると直感的ではない。

そのため、デクリメントを行う命令が欲しいと感じた。

## 16進 → 10進

柿沼

流れについての説明

使用した変数について, 以下に示す。使用した定数について, 以下に示す。

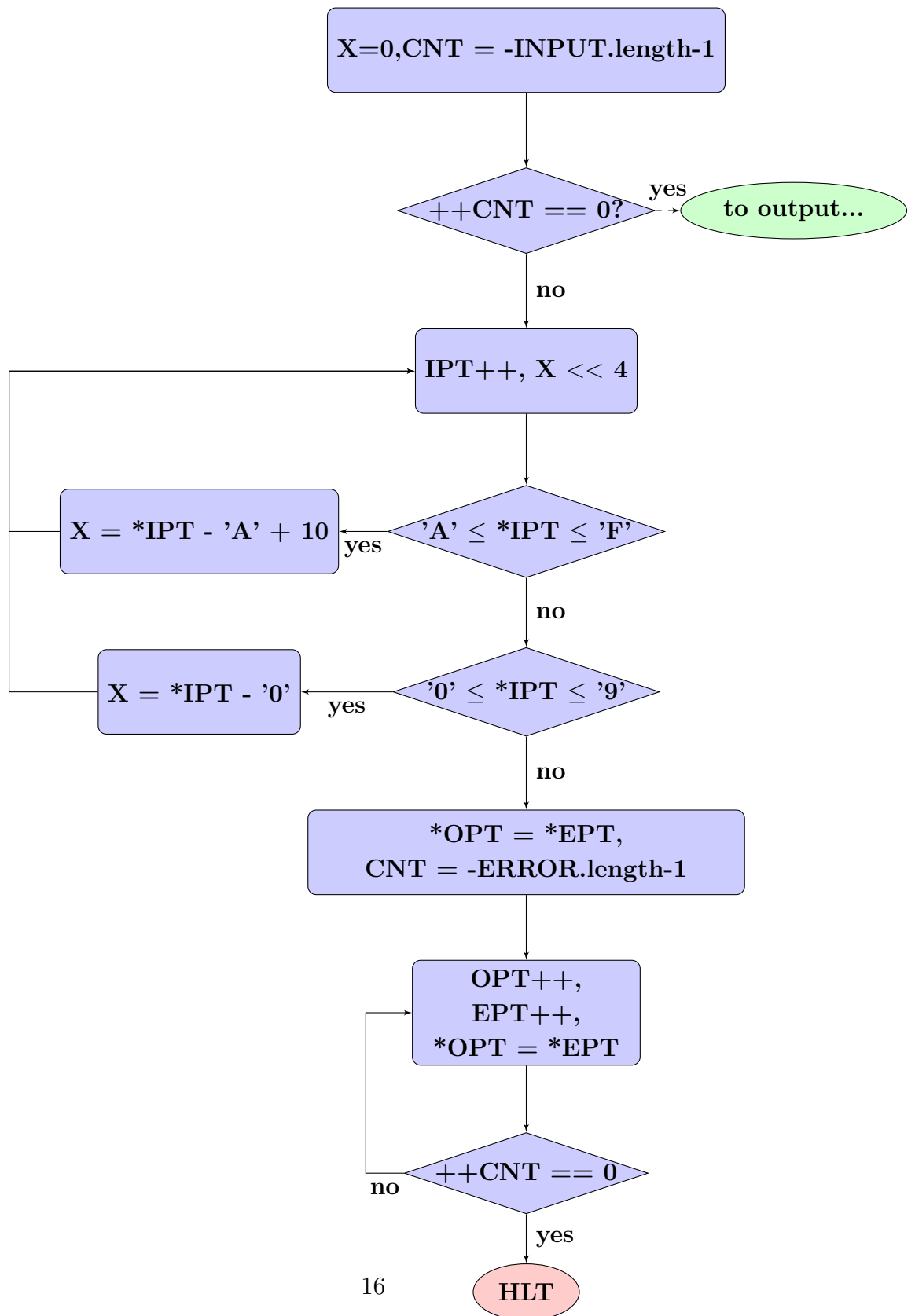
変数名	説明
OUTPUT	出力文字列
IPT	入力文字列の先頭へのポインタ
OPT	出力文字列の先頭へのポインタ
EPT	エラー文字列の先頭へのポインタ
CNT	文字列のカウント用変数
X	徐算の対象
Y	徐算の対象
P	徐算の商
Q	徐算の余り
C	徐算の余り

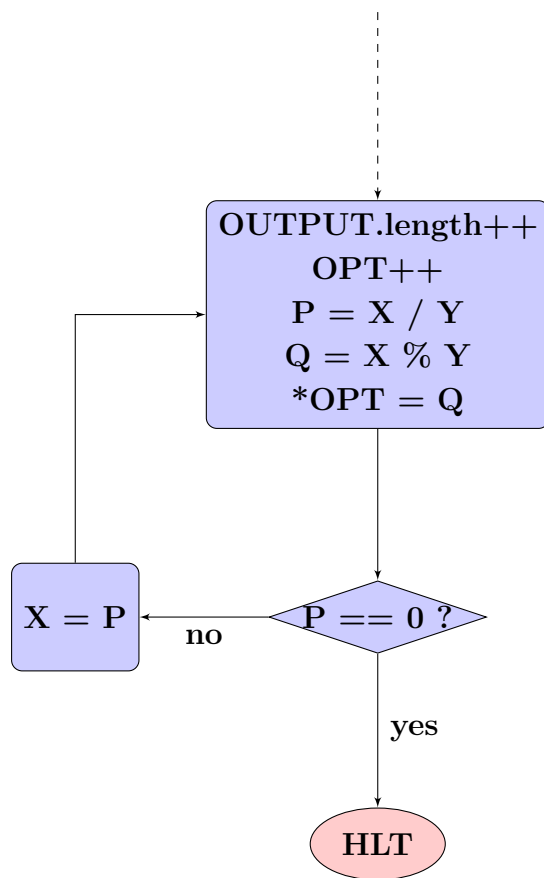
定数名	説明
TEN	即値「10」
A	即値「'A'」
F	即値「'F'」
ZERO	即値「'0'」
NINE	即値「'9'」
CINIT	即値「-16」
INPUT	入力文字列
ERROR	エラー文字列

16進数文字列として格納されたINPUTを数値としてXに格納し、1桁ずつOUTPUTに入れていく。Xへの格納に不備が発生した場合にはERRORをOUTPUTにコピーする。なお、このプログラムでは出力文字列は逆順に格納される。

以下にフローチャートを示す。







工夫点

特になし

ソースコードと総命令数

Listing 5: report1.3.asm

```

1  ORG 10
2  / CNT = -INPUT.length-1
3  CLA
4  ADD IPT I
5  CMA
6  STA CNT
7  / ++CNT == 0 ? LO : goto L1
8  L0,
9  ISZ CNT

```

```

10  BUN L1
11  BUN LO
12  / IPT++
13  L1,
14  ISZ IPT
15  / X <<= 4
16  LDA X
17  CIL
18  CIL
19  CIL
20  CIL
21  STA X
22  / AC = INPUT
23  CLA
24  ADD IPT I
25  / input > 'F' goto LE
26  CMA
27  INC
28  ADD F
29  SPA
30  BUN LE
31  / input >= 'A' goto LA
32  LDA A
33  CMA
34  INC
35  ADD IPT I
36  SNA
37  BUN LA
38  / input < '0' goto LE
39  LDA ZERO
40  CMA
41  INC
42  ADD IPT I
43  SPA
44  BUN LE
45  / input <= '9' goto LN
46  CLA
47  ADD IPT I
48  CMA
49  INC
50  ADD NINE
51  SNA
52  BUN LN
53  / output 'ERROR'
54  / *OPT = *EPT, CNT = -ERROR.length-1
55  LE,
56  CLA
57  ADD EPT I
58  STA OPT I

```

```

59      CMA
60      STA CNT
61      / EPT++, OPT++, *OPT = *EPT
62      LE2,
63      ISZ EPT
64      ISZ OPT
65      CLA
66      ADD EPT I
67      STA OPT I
68      / ++CNT == 0 ? HLT : goto LE2
69      ISZ CNT
70      BUN LE2
71      HLT
72      / AlphaBet to Dec
73      / X += input - 'A' + 10, goto L0
74      LA,
75      LDA A
76      CMA
77      ADD TEN
78      BUN LX
79      / X += input - '0', goto L0
80      LN,
81      LDA ZERO
82      CMA
83      LX,
84      INC
85      ADD IPT I
86      ADD X
87      STA X
88      BUN L0
89      /binary -> dec
90      /OUTPUT.length++, OPT++, P = X/Y, Q = X%Y, *OPT = Q
91      LO,
92      ISZ OUTPUT
93      ISZ OPT
94      BSA LD
95      LDA Q
96      STA OPT I
97      / P == 0 ? HLT
98      LDA P
99      SZA
100     BUN LD3
101     HLT
102     / X = P
103     LD3,
104     STA X
105     BUN LO
106     LD,
107     HEX 0

```

```

108   CLA
109   STA P
110   STA Q
111   LDA TEN
112   STA Y
113   LDA CINIT
114   STA C
115   / P<<= 1, [Q,X] <<= 1
116 LD0,
117   CLE
118   LDA P
119   CIL
120   STA P
121   LDA X
122   CIL
123   STA X
124   LDA Q
125   CIL
126   STA Q
127   / Q >= Y ? goto LD1 : goto LD2
128   LDA Y
129   CLE
130   SZA
131   CME
132   CMA
133   INC
134   ADD Q
135   SZE
136   BUN LD1
137   BUN LD2
138   / Q -= Y, P++
139 LD1,
140   STA Q
141   LDA P
142   INC
143   STA P
144   / ++C == 0 ? HLT : goto LD0
145 LD2,
146   ISZ C
147   BUN LD0
148   BUN LD I
149   / data
150 TEN, DEC 10
151 A,  CHR A
152 F,  CHR F
153 ZERO, CHR 0
154 NINE,  CHR 9
155 CINIT, DEC -16
156 INPUT, DEC 4

```

```

157 CHR A
158 CHR B
159 CHR C
160 CHR X
161 OUTPUT, DEC 0
162   CHR -
163   CHR -
164   CHR -
165   CHR -
166   CHR -
167 ERROR, DEC 5
168   CHR R
169   CHR O
170   CHR R
171   CHR R
172   CHR E
173 IPT, SYM INPUT
174 OPT, SYM OUTPUT
175 EPT, SYM ERROR
176 X, DEC 0
177 Y, DEC 10
178 P, DEC 0
179 Q, DEC 0
180 C, DEC -16
181 CNT, DEC 0
182 END

```

総命令数:121

異なる入力に対する実行命令ステップ数

入力 ( $X$ )	ステップ数
$0F$	783
$FFFF$	1957
$XX$	67
$ABCX$	157

EX3 命令セットで改良すべき点

INC の逆で、-1 を足す命令が欲しいと感じた。

## 中田

### 流れについての説明

使用した変数について,以下に示す。使用した定数について,以下に示す。

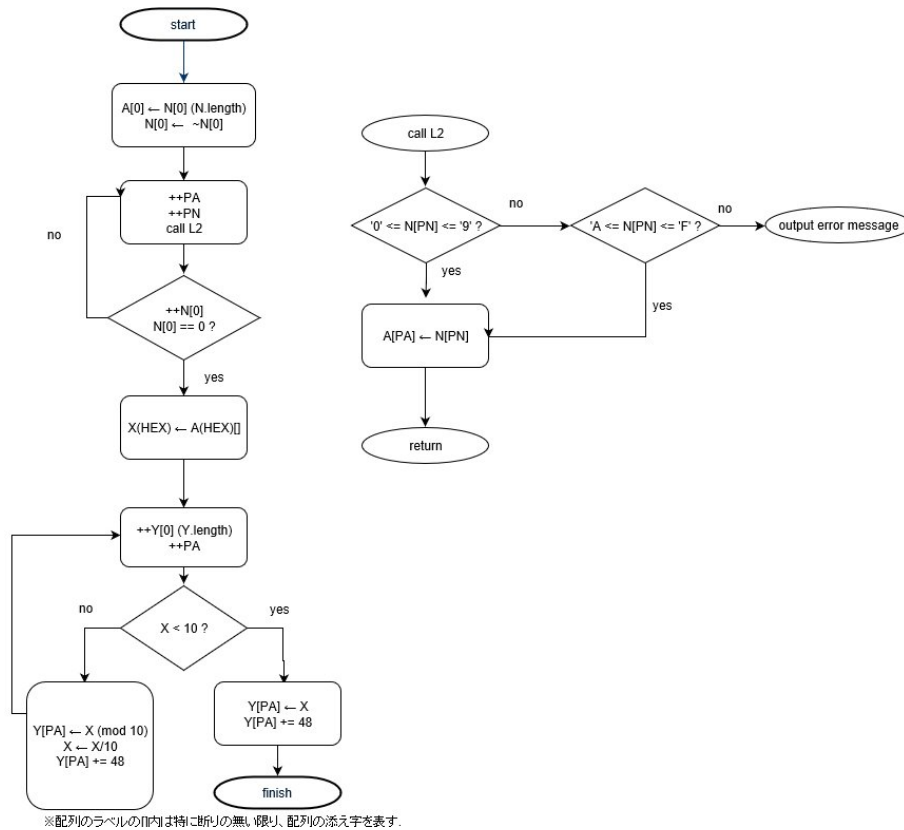
変数名	説明
N	入力文字列、先頭は文字数
PN	入力文字列へのポインタ
X	入力された数を 16 進数で保持
A	ASCII コードを 16 進数に変換した文字列、先頭は文字数
PA	A へのポインタ
Y	出力文字列
PY	出力文字列へのポインタ
PEMG	エラー文字列へのポインタ
W	除算でカウントする変数
W1	W の初期値を保持
COUNT	カウントする変数

定数名	説明
M	即値「10」
VM1	即値「-1」
VM48	即値「-48」
VP48	即値「48」
VM55	即値「-55」
VM6	即値「-6」
VM10	即値「-10」
EMG	エラー文字列
ECNT	エラーの文字数

ASCII コードで保持されている文字列 N を 16 進数文字列 A へと変換し,16 進数の数値として X に保持する。  
X を繰り返し 10 で割ることで 10 進数文字列 Y へと変換し、それをさらに ASCII コードの文字列へと変換する。  
入力された文字列が 16 進数として扱えない場合はエラーを出力する。

また、入力の 16 進数には大文字のアルファベットのみを扱うものとし、出力は逆順に出力されるものとする。

以下にフローチャートを示す。



工夫点

特になし。

ソースコードと総命令数

Listing 6: report1\_3\_nakata.asm

```

1      ORG 10          / program entry point
2      SIO             / serial port select
3      / A ← N.length , N ← ~N
4      L0,

```



```

5          CLA      / AC <- 0
6          LDA N    / AC <- M[N]
7          STA A     / M[A] <- AC
8          CMA      / AC <- ~AC
9          INC       / ++AC
10         STA N     / M[N] <- N
11         CLA      / AC <- 0
12 / ++M[PA], ++M[PN], call L2
13 L1,
14     ISZ PA / ++M[PA]
15     ISZ PN / ++M[PN]
16     BSA L2 / call L2
17     ISZ N   / ++M[N] (M[N] == 0) ? skip next
18     BUN L1 / goto L1
19 / X(HEX) <- A(CHR) []
20 LX,
21     CLA      / AC <- 0
22     LDA A     / AC <- M[A]
23     CMA      / AC <- ~AC
24     INC       / ++AC
25     STA COUNT / M[COUNT] <- AC
26     ADD PA    / AC += M[PA]
27     STA PA    / M[PA] <- AC
28     ISZ COUNT / ++M[COUNT]
29     CLA      / AC <- 0
30     ISZ PA    / ++M[PA]
31     ADD PA I  / AC += M[M[PA]]
32 LX1,
33     CIL      / {E,AC[15:0]} <- {AC[15:0],E}
34     CIL      / {E,AC[15:0]} <- {AC[15:0],E}
35     CIL      / {E,AC[15:0]} <- {AC[15:0],E}
36     CIL      / {E,AC[15:0]} <- {AC[15:0],E}
37     ISZ PA    / ++M[PA]
38     ADD PA I  / AC += M[M[PA]]
39     ISZ COUNT / ++M[COUNT]
40     BUN LX1 / goto LX1
41     STA X     / M[X] <- AC
42 / Y(CHR) [] <- X(HEX)
43 LY,
44     LDA W1    / AC <- M[W1]
45     STA W     / M[W] <- AC
46     CLA      / AC <- 0
47     ISZ Y     / ++M[Y]
48     ISZ PY    / ++M[PY]
49     LDA X     / AC <- M[X]
50     ADD VM10  / AC -= 10
51     CME      / E <- ~E
52     SZE      / (E == 0) ? skip next
53     BUN LY5  / goto LY5

```

```

54 LY0,
55     LDA W / AC <- M[W]
56     SNA   / (AC < 0) ? skip next
57     BUN LY4 / goto LY4
58 LY1,
59     CLE   / E <- 0
60     CLA   / AC <- 0
61     LDA X / AC <- M[X]
62     CIL   / {E,AC[15:0]} <- {AC[15:0],E}
63     STA X / M[X] <- AC
64     CLA   / AC <- 0
65     LDA PY I / AC <- M[M[PY]]
66     CIL   / {E,AC[15:0]} <- {AC[15:0],E}
67     STA PY I / M[M[PY]] <- AC
68     LDA M / AC <- M[M]
69     CMA   / AC <- ~AC
70     INC   / ++AC
71     ADD PY I / AC += M[M[PY]]
72     SZE   / (E == 0) ? skip next
73     BUN LY2 / goto LY2
74 LY3,
75     CLA   / AC <- 0
76     LDA W / AC <- M[W]
77     INC   / ++AC
78     STA W / M[W] <- AC
79     CLA   / AC <- 0
80     BUN LY0 / goto LY0
81 LY2,
82     STA PY I / M[M[PY]] <- AC
83     CLA   / AC <- 0
84     LDA X / AC <- M[X]
85     INC   / ++AC
86     STA X / AC <- M[X]
87     BUN LY3 / goto LY3
88 LY4,
89     LDA PY I / AC <- M[M[PY]]
90     ADD VP48 / AC += 48
91     STA PY I / AC <- M[M[PY]]
92     BUN LY / goto M[LY]
93 LY5,
94     CLA   / AC <- 0
95     LDA X / AC <- M[X]
96     ADD VP48 / AC += 48
97     STA PY I / M[M[PY]] <- AC
98     HLT
99 / A(HEX)[] <- N(CHR)[] , error check
100 L2,
101     HEX 0 / return address
102     LDA PN I / AC <- M[M[PN]]

```

```

103      ADD VM48 / AC -= 48
104      STA PA I / M[M[PA]] <- AC
105      SPA      / (AC >= 0) ? skip next
106      BUN ERR / goto ERR
107      ADD VM10 / AC -= 10
108      SPA      / (AC >= 0) ? skip next
109      BUN L2 I / goto L2
110      LDA PN I / AC <- M[PN]
111      ADD VM55 / AC -= 55
112      STA PA I / AC <- M[M[PA]]
113      ADD VM10 / AC -= 10
114      SPA      / (AC >= 0) ? skip next
115      BUN ERR / goto ERR
116      ADD VM6 / AC -= 6
117      SPA      / (AC >= 0) ? skip next
118      BUN L2 I / goto L2
119      BUN ERR / goto ERR
120  ERR,
121      LDA PEMG I / AC [PEMG] PEMG: error message address
122      OUT      / OUTR <- AC
123      ISZ PEMG / ++M[PEMG]
124      ISZ ECNT / ++M[ECNT] (M[ECNT] == 0) ? skip next
125      BUN ERR / goto ERR
126      HLT
127  / data
128  N,      DEC 4 / length init: 4
129          CHR F / N = FFFF input (HEX)
130          CHR F
131          CHR F
132          CHR F
133  PN,     SYM N / N pointer address
134  X,      HEX 0 /
135  A,      HEX 0 / length init: 0
136          HEX 0 / N(CHR) --> A(HEX) init: 0000
137          HEX 0
138          HEX 0
139          HEX 0
140  PA,     SYM A / A pointer address
141  Y,      HEX 0 / length init: 0
142          HEX 0 / A(HEX) --> Y(DEC) init: 00000
143          HEX 0
144          HEX 0
145          HEX 0
146          HEX 0
147  PY,     SYM Y / Y pointer address
148  / error message
149  EMG,    HEX 45 / 'E'
150          HEX 52 / 'R'
151          HEX 52 / 'R'

```

```

152          HEX 4f          / 'O'
153          HEX 52          / 'R'
154 ECNT, DEC -5 / error counter
155 PEMG,  SYM EMG / error message pointer address
156 W,      HEX 0
157 W1,     DEC -16
158 M,      DEC 10
159 COUNT,  HEX 0
160 VM1,    DEC -1
161 VM48,   DEC -48
162 VP48,   DEC 48
163 VM55,   DEC -55
164 VM6,    DEC -6
165 VM10,   DEC -10
166 END

```

総命令数:107

異なる入力に対する実行命令ステップ数

入力 ( $X$ )	ステップ数
$000F$	487
$FFFF$	1724
$00XX$	74
$ABCX$	107

EX3 命令セットで改良すべき点

即値を ADD などの命令で直接扱えるようにして欲しいと感じた。

## 素数計算

柿沼

流れについての説明

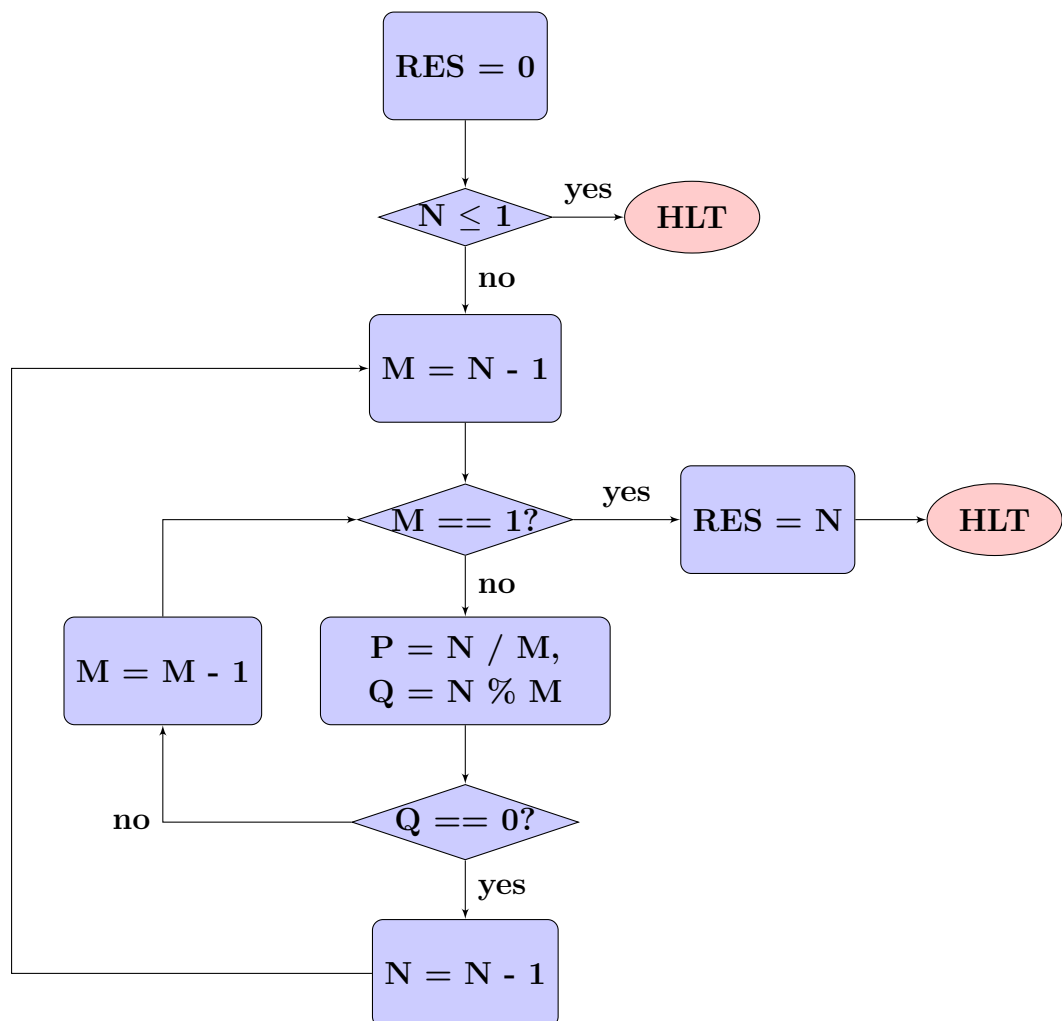
使用した変数について, 以下に示す。使用した定数について, 以下に示す。

変数名	説明
N	現在の素数候補
M	N を割る数
RES	結果
X	徐算の対象
Y	徐算の対象
P	徐算の商
Q	徐算の余り
C	徐算の余り

定数名	説明
CINIT	即値「-16」

自然数  $N$  が素数であるかの判定には  $N$  未満の自然数すべてに対して徐算を実行することで実現した。

以下にフローチャートを示す。



### 工夫点

$Q \geq Y$  の部分で  $Q - Y$  を計算するため、それをそのまま  $Q -= Y$  に転用した。

$Q \geq Y$  の計算は符号なし 16bit 整数として計算しなければならないため、E レジスタを含めた符号あり 17bit と考えて、適切な計算の後 SZE で判定するという手法をとった。

Y を E レジスタを含めた 17bit 符号あり整数と考えた場合、符号反転後はほとんどの場合最上位ビット (E レジスタ) が立っていることになるが、唯一 Y が 0 のときのみ符号を反転しても E レジスタは 0 のままなため、反転前の Y に対して SZA で E レジスタの値を定めた。

## ソースコードと総命令数

Listing 7: report1\_4.asm

```
1  ORG 10
2  /N <= 1 ? HLT
3    LDA N
4    SZA
5    CME
6    CMA
7    ADD TWO
8    SZE
9    HLT
10 / M = N-1
11 L0,
12   LDA N
13   BSA DC
14   STA M
15 / M == 1 ? RES = N, HLT : goto L2
16 L1,
17   LDA M
18   BSA DC
19   SZA
20   BUN L2
21   LDA N
22   STA RES
23   HLT
24 / P = N / M, Q = N % M, Q == 0 ? N--, goto L0 : goto L3
25 L2,
26   LDA N
27   STA X
28   LDA M
29   STA Y
30   BSA LD
31   LDA Q
32   SZA
33   BUN L3
34   LDA N
35   BSA DC
36   STA N
37   BUN L0
38 / M--, goto L1
39 L3,
40   LDA M
41   BSA DC
42   STA M
43   BUN L1
44 /P = 0, Q = 0, C = -16
45 LD,
```

```

46      HEX 0
47      CLA
48      STA P
49      STA Q
50      LDA CINIT
51      STA C
52      / P<<= 1, [Q,X] <<= 1
53 LD0,
54      CLE
55      LDA P
56      CIL
57      STA P
58      LDA X
59      CIL
60      STA X
61      LDA Q
62      CIL
63      STA Q
64      / Q >= Y ? goto LD1 : goto LD2
65      LDA Y
66      CLE
67      SZA
68      CME
69      CMA
70      INC
71      ADD Q
72      SZE
73      BUN LD1
74      BUN LD2
75      / Q -= Y, P++
76 LD1,
77      STA Q
78      LDA P
79      INC
80      STA P
81      / ++C == 0 ? HLT : goto LD0
82 LD2,
83      ISZ C
84      BUN LD0
85      BUN LD I
86      /AC = AC-1
87 DC,
88      HEX 0
89      CMA
90      INC
91      INC
92      CMA
93      INC
94      BUN DC I

```



```

95 / data
96 CINIT, DEC -16
97 TWO, DEC 2
98 RES, DEC 0
99 N, DEC 65535
100 M, DEC 0
101 X, DEC 0
102 Y, DEC 0
103 P, DEC 0
104 Q, DEC 0
105 C, DEC 0
106 END

```

総命令数:72

異なる入力に対する実行命令ステップ数

入力 ( $N$ )	ステップ数
2	27
64	61307
255	337101
65535	245377052

EX3 命令セットで改良すべき点

可変長のデータを保持する機能がないので、何か方法が欲しいと思った。スタックなどがあればよいが、簡単さを重視した命令セットでは難しいのかもしれない。

中田

流れについての説明

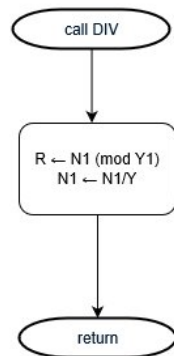
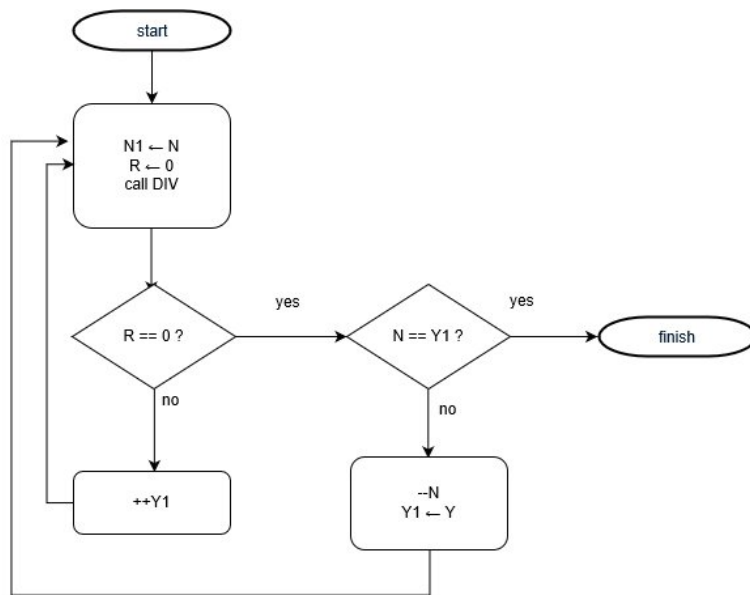
使用した変数について、以下に示す。使用した定数について、以下に示す。

自然数  $N$  が素数であるかの判定には  $N$  未満の自然数すべてに対して徐算を実行することで実現した。

変数名	説明
N	入力および素数の候補、処理終了後において結果
N1	N を計算の間、保持する
Y	最初に割る数 (2)
Y1	割る数
R	徐算の余り
W	徐算でカウントする変数
W	W1 の初期値を保持

定数名	説明
VM1	即値「-1」

以下にフローチャートを示す。



工夫点

特になし。

ソースコードと総命令数

Listing 8: report1\_4\_nakata.asm

```

1      ORG 10          / program entry point
2  / N1 ← N, call DIV, M[R] == 0 ?
3  LD0,

```

```

4      LDA N   / AC <- M[N]
5      STA N1  / M[N1] <- AC
6      CLA     / AC <- 0
7      STA R   / M[R] <- AC
8      BSA DIV / goto DIV
9      LDA R   / AC <- M[R]
10     SZA     / (AC == 0) ? skip next
11     BUN LD1 / goto LD1
12 / M[Y1] == M[N] ?
13     LDA Y1  / AC <- M[Y1]
14     CMA     / AC <- ~AC
15     INC     / ++AC
16     ADD N   / AC += M[N]
17     SZA     / (AC == 0) ? skip next
18     BUN LD2 / goto LD2
19     HLT
20 / --M[N1] , M[Y1] <- M[Y]
21 LD2,
22     LDA N   / AC <- M[N]
23     ADD VM1 / AC -= 1
24     STA N   / M[N] <- AC
25     LDA Y   / AC <- M[Y]
26     STA Y1  / M[Y1] <- AC
27     BUN LD0 / goto LD0
28 / ++M[Y1]
29 LD1,
30     ISZ Y1  / ++M[Y1]
31     BUN LD0
32 / M[R] <- M[N1] (mod M[Y1]) , M[N1] <- M[N1]/M[Y1]
33 DIV,
34     HEX 0   / return address
35     LDA W1  / AC <- M[W1]
36     STA W   / M[W] <- AC
37     CLA     / AC <- 0
38 DIV0,
39     LDA W   / AC <- M[W]
40     SNA     / (AC < 0) ? skip next
41     BUN DIV I / goto DIV
42 DIV1,
43     CLE     / E <- 0
44     CLA     / AC <- 0
45     LDA N1  / AC <- M[N1]
46     CIL     / {E, AC[15:0]} <- {AC[15:0], E}
47     STA N1  / M[N1] <- AC
48     CLA     / AC <- 0
49     LDA R   / AC <- M[R]
50     CIL     / {E, AC[15:0]} <- {AC[15:0], E}
51     STA R   / M[R] <- AC
52     LDA Y1  / AC <- M[Y1]

```

```

53      CMA      / AC <- ~AC
54      INC      / ++AC
55      ADD R    / AC += M[R]
56      SZE      / (E == 0) ? skip next
57      BUN DIV2 / goto DIV2
58 DIV3,
59      CLA      / AC <- 0
60      LDA W    / AC <- M[W]
61      INC      / ++AC
62      STA W    / M[W] <- AC
63      CLA      / AC <- 0
64      BUN DIV0 / goto DIV0
65 DIV2,
66      STA R    / M[R] <- AC
67      CLA      / AC <- 0
68      LDA N1   / AC <- M[N1]
69      INC      / ++AC
70      STA N1   / M[N1] <- AC
71      BUN DIV3 / goto DIV3
72 / data
73 N,      HEX ffff    N = ffff --> X:init    --> result
74 N1,     DEC 0
75 Y,      DEC 2
76 Y1,     DEC 2
77 R,      HEX 0
78 W,      DEC 0
79 W1,     DEC -16
80 VM1,    DEC -1
81 END

```

総命令数:57

異なる入力に対する実行命令ステップ数

入力 (N)	ステップ数
2	378
64	24240
255	100047
65535	24769799

### EX3 命令セットで改良すべき点

任意の大きさの素数のテーブルを容易に作成できるように、動的にメモリの確保できる配列を扱えるようにして欲しいと思った。

# サンプルプログラム解析レポート (5)

## 0.1 test\_io1.asm

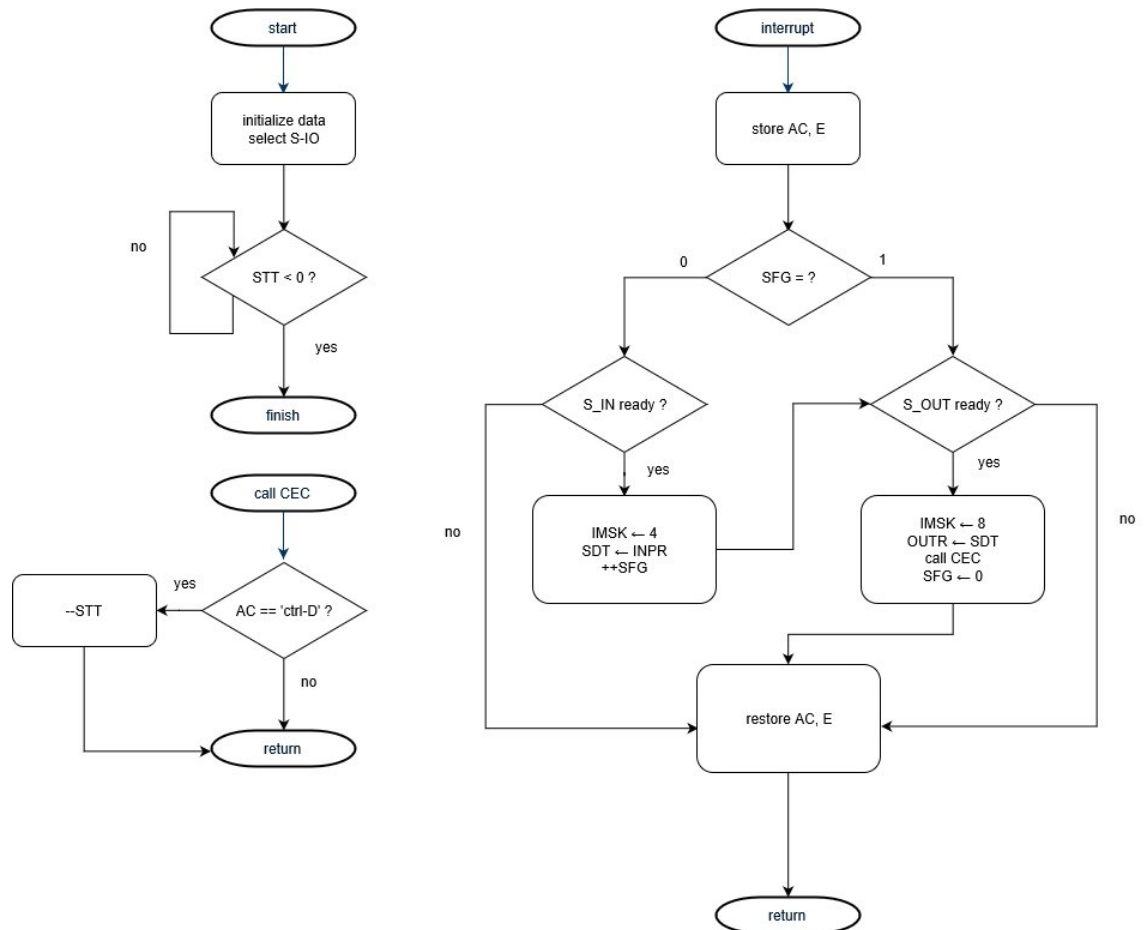
### 0.1.1 プログラムの概要

入力された文字を返すプログラムである。  
入出力ポートとしてシリアルポートが選択され、割り込みが発生すると、 $S_{IN}$  を通して入力された文字を  $SDT$  に格納し、 $SDT$  に格納された文字を  $S_{OUT}$  を通して出力する。  
 $ctrl - D$  が入力されるとプログラムが終了する。

#### 各サブルーチンの機能

サブルーチン名	機能の説明
INI	プログラム本体のエントリーポイント。各種変数の初期化をする。
ST0	割り込みハンドラのエントリーポイント
L0	メインループ。入力可能状態で待機し、ctrl-D が入力されるとプログラムを終了する。
CEC	入力された文字が ctrl-D かをチェックする
$I_{HND}$	割り込みハンドラ。 $S_{IN}$ を通して入力された文字を $SDT$ に格納し、格納された文字を $S_{OUT}$ を通して出力する。

## 処理の流れ



## 0.2 test\_calc1.asm

### 0.2.1 プログラムの概要

入力された文字を返すプログラムである。初期状態ではパラレルポートが選択される。

割り込みが発生するごとに、次の2つの動作が交互に発生する。

- ・  $S_{IN}$  を通して入力された文字を  $SDT$  に格納し、 $SDT$  に格納された文字を  $P_{OUT}$  を通して出力する。
- ・  $P_{IN}$  を通して入力された文字を  $PDT$  に格納し、 $PDT$  に格納された文字を  $S_{OUT}$  を通して出力する。

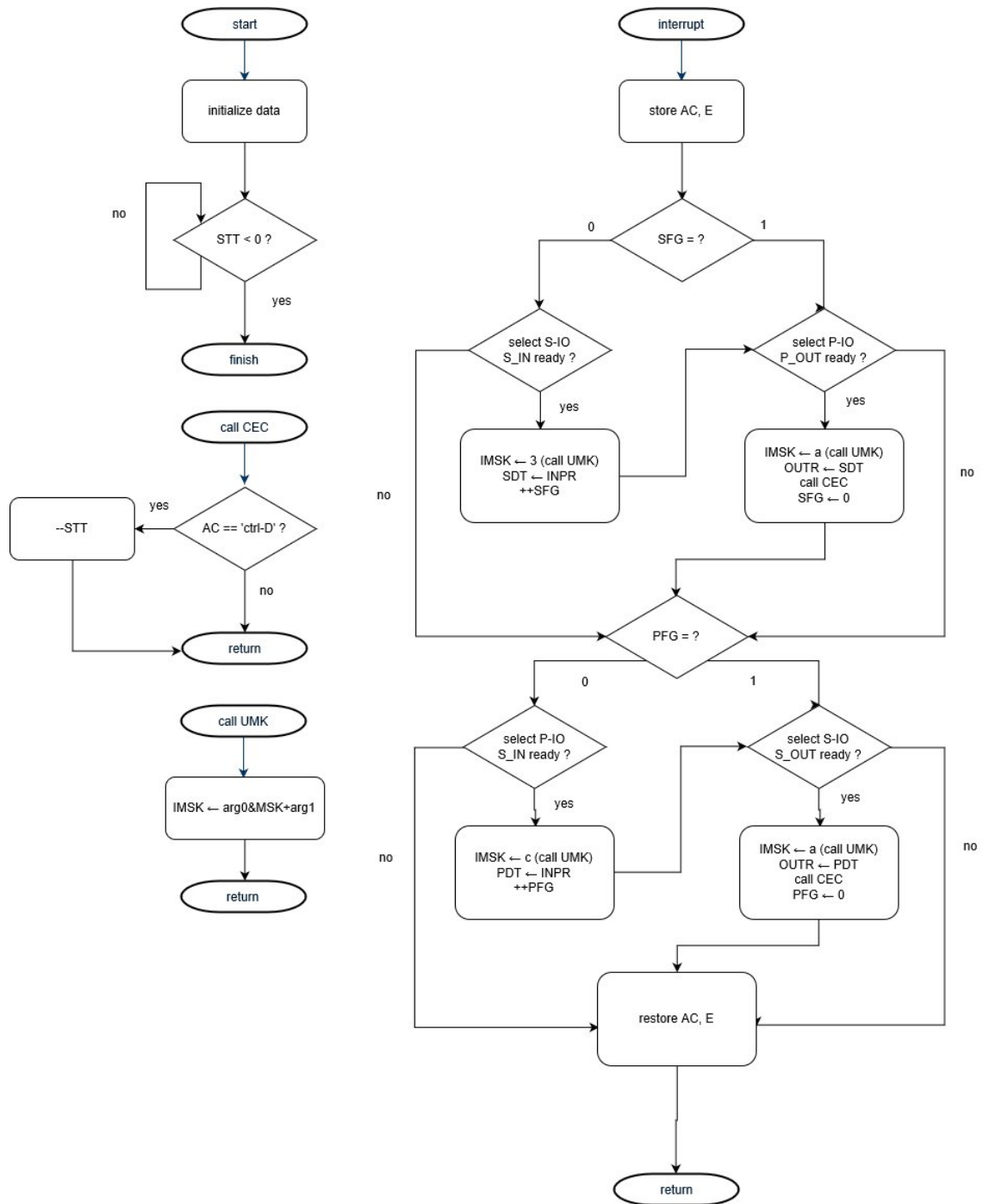
$ctrl - D$  が入力されるとプログラムが終了する。



## 各サブルーチンの機能

サブルーチン名	機能の説明
INI	プログラム本体のエントリーポイント。各種変数の初期化をする。
ST0	割り込みハンドラのエントリーポイント
L0	メインループ。入力可能状態で待機し、ctrl-D が入力されるとプログラムを終了する。
CEC	入力された文字が ctrl-D かをチェックする
UMK	IMSK を更新する。
$I_HND$	割り込みハンドラ。割り込みが発生するごとに、次の2つの動作が交互に発生する。・ $S_I N$ を通して入力された文字を $SDT$ に格納し、 $SDT$ に格納された文字を $P_OUT$ を通して出力する。・ $P_I N$ を通して入力された文字を $PDT$ に格納し、 $PDT$ に格納された文字を $S_OUT$ を通して出力する。

## 処理の流れ



## 0.3 test\_calc1.asm

### 0.3.1 プログラムの概要

入力された数値の計算をさせるプログラムである。ユーザーは最大 4 桁の 16 進数表記の数値の足し算と引き算ができる。オーバーフローやアンダーフローは無視し、下位 16bit を答えとして出力する。入力する数値が 4 桁を超えたりするとエラー文を出力して終了する。

#### 各サブルーチンの機能

サブルーチン名	機能の説明
INI	プログラム本体のエントリーポイント。各種変数の初期化をした後、入力可能状態で待機し、BYE が 1 になると停止する。
INI.ST	各種変数の初期化
ST0	割り込みハンドラのエントリーポイント
CHK_CH	AC と TMI が等しいかどうかを返す。
SET_MSG	PTR_MG に AC の指すアドレスを入れ、その指す文字列長を CNT に入れる。
READ_HX	TMI に入った文字が 16 進数として有効な文字なら HXI にその値を入れ、 $AC \geq$ にする。無効なら $AC < 0$ にする。このとき、文字が改行文字ならば=に置き換える。
CHK_DGT	引数として AC とその他に 2 つの定数を取り、AC がそれらの間に入っているかどうかを返す。
WRITE_Z	Z に入った数値を文字列として ZMG に入れる。

## 処理の流れ

