計算機論理設計

3. 数値演算プログラミング(2)

一色 剛

工学院情報通信系

isshiki@ict.e.titech.ac.jp

単精度乗算

	A
×	B
CH	C

無視

Nビット × Nビット = 2Nビット
→単精度乗算は積の
下位Nビットだけを出力
→ 積がNビットに収まるように
にプログラムで調整が必要

```
int a, b, c;
c = a * b;
```

32ビット × 32ビット →
32ビット積 （C言語）

■ Nビット単精度乗算:「符号なし」データとして考える

$$A = \sum_{i=0}^{N-1} A_i \cdot 2^i$$

$$B = \sum_{i=0}^{N-1} B_i \cdot 2^i$$

$$A \cdot B = A \cdot \sum_{i=0}^{N-1} B_i \cdot 2^i = \sum_{i=0}^{N-1} B_i \cdot (A \cdot 2^i)$$

(N=8)	00001001	(A) 9
	00001101	(B) 13
	00001001	27
0	00000000	9
00	00100100	(C) 117
000	01001000	
0000	00000000	
00000	00000000	
000000	00000000	
0000000	00000000	
00000000	01110101	

単精度乗算プログラム

$$A \cdot B = \sum_{i=0}^{N-1} B_i \cdot (A \cdot 2^i)$$

$A \cdot 2^i$: A の論理左シフト
B_i : B の i ビット目
$B_i = 1$ の場合 $A \cdot 2^i$を C に加算

00001001	A
00001101	B
00001001	$B_0 \cdot A \cdot 2^0$
00000000	$B_1 \cdot A \cdot 2^1$
00100100	$B_2 \cdot A \cdot 2^2$
01001000	$B_3 \cdot A \cdot 2^3$
00000000	$B_4 \cdot A \cdot 2^4$
00000000	$B_5 \cdot A \cdot 2^5$
00000000	$B_6 \cdot A \cdot 2^6$
00000000	$B_7 \cdot A \cdot 2^7$
01110101	C

a = A, b = B とする

```
c = 0;
for (k = 0; k < N; k++) {
    bi = b & 1;
    b = b >> 1;
    if (bi)
        c = c + a;
    a = a << 1;
}
```

bi : B の i ビット目

$A \cdot 2^i$を C に加算

a = $A \cdot 2^i$

単精度乗算プログラム

```
c = 0; k = –N;
do {
   bi = b & 1;
   b = b >> 1;
   if (bi)
      c = c + a;
   a = a << 1;
   k = k + 1;
} while (k != 0);
```

```
L0,  LDA  B
     CIR
     STA  B
     SZE
     BUN  L1
     BUN  L2
L1,  LDA  C
     ADD  A
     STA  C
     CLE
L2,  LDA  A
     CIL
     STA  A
     ISZ  K
     BUN  L0
     HLT
<データ宣言部>
```

$M[B] \leftarrow M[B] \gg 1$

$E \leftarrow B_i$

SKIP-NEXT $(E = 0)$

$M[C] \leftarrow M[C] + M[A]$

$M[A] \leftarrow M[A] \ll 1$

$M[K] \leftarrow M[K] + 1,$

SKIP-NEXT $(M[K] + 1 = 0)$

```
<データ宣言部>
A,    DEC  9
B,    DEC  13
C,    DEC  0
K,    DEC  –16
```

単精度乗算プログラム

```
c = 0; k = –N;
do {
   bi = b & 1;
   b = b >> 1;
   if (bi)
      c = c + a;
   a = a << 1;
   k = k + 1;
} while (k != 0);
```

<データ宣言部>

```
A,     DEC  9
B,     DEC  13
C,     DEC  0
K,     DEC  –16
```

B_iビット判定

C += A

A <<=1

終了判定

```
L0,  LDA  B
     CIR
     STA  B
     SZE
     BUN  L1
     BUN  L2
L1,  LDA  C
     ADD  A
     STA  C
     CLE
L2,  LDA  A
     CIL
     STA  A
     ISZ  K
     BUN  L0
     HLT
```

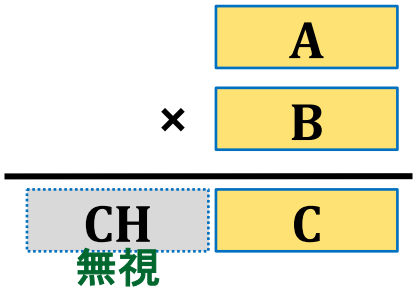
1 2 3 4

<データ宣言部>

M[B]の最上位ビットにE(不定値)が代入されるが、B_iビット判定で不具合は発生しない

$$A \cdot B = \sum_{i=0}^{N-1} B_i \cdot (A \cdot 2^i)$$

単精度乗算（符号の有無）



Nビット × Nビット = 2Nビット
→単精度乗算は積の
下位Nビットだけを出力

```
c = a * b;
```

■ 単精度乗算:「符号なし」と「符号付き」（2の補数）は同じ計算方法が適用できる

11111001	249
10001101	141
11111001	249
00000000	996
11100100	249
11001000	35109
00000000	
00000000	
00000000	
10000000	
00100101	

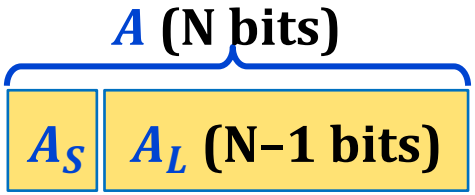
	unsigned	signed
11111001	249	–7
10001101	141	–115

(249:unsigned) * (141:unsigned) =	35109	10001001 00100101
(–7:signed) * (141:unsigned) =	–987	11111100 00100101
(249:unsigned) * (–115:signed) =	–28635	10010000 00100101
(–7:signed) * (–115:signed) =	805	00000011 00100101

単精度乗算（符号の有無）

■ 単精度乗算:「符号なし」と「符号付き」（2の補数）は同じ計算方法が適用できる

$A = \{A_S, A_L\}$: Nビットデータ
A_S : 最上位ビット
A_L : 下位(N – 1)ビットデータ



符号なし：$u.A = A_S \cdot 2^{N-1} + A_L$
2の補数：$s.A = -A_S \cdot 2^{N-1} + A_L$

$$u.A \cdot u.B = \left(A_S \cdot 2^{N-1} + A_L\right)\left(B_S \cdot 2^{N-1} + B_L\right)$$
$$u.A \cdot u.B = A_S B_S \cdot 2^{2N-2} + (A_S B_L + B_S A_L) \cdot 2^{N-1} + A_L B_L$$

$$s.A \cdot u.B = -A_S B_S \cdot 2^{2N-2} + (-A_S B_L + B_S A_L) \cdot 2^{N-1} + A_L B_L$$

$$u.A \cdot s.B = -A_S B_S \cdot 2^{2N-2} + (A_S B_L - B_S A_L) \cdot 2^{N-1} + A_L B_L$$

$$s.A \cdot s.B = A_S B_S \cdot 2^{2N-2} + (-A_S B_L - B_S A_L) \cdot 2^{N-1} + A_L B_L$$

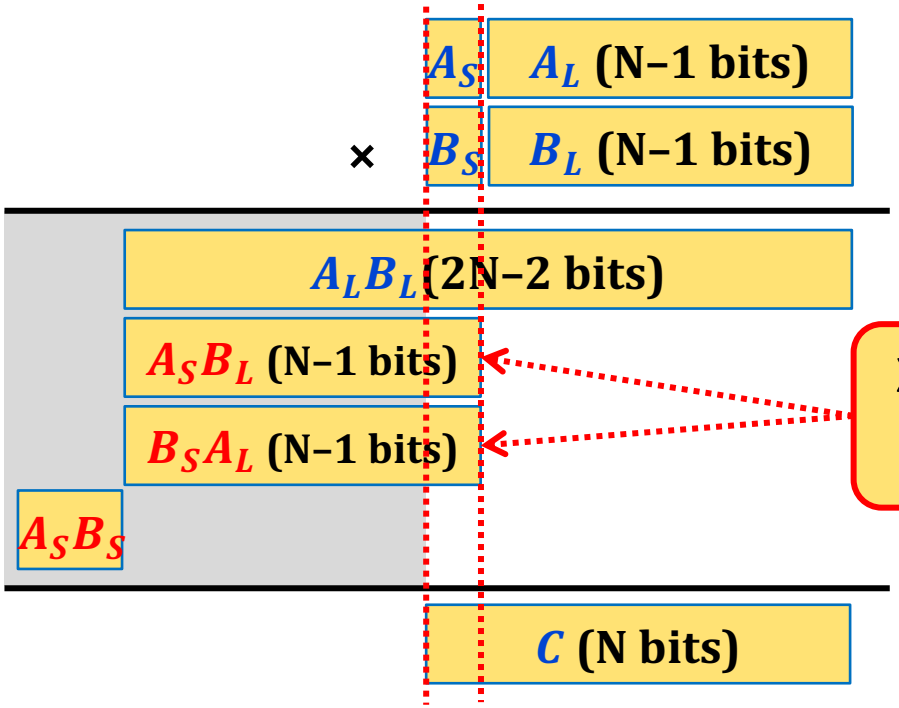
単精度乗算（符号の有無）

$$u.A \cdot u.B = A_S B_S \cdot 2^{2N-2} + (A_S B_L + B_S A_L) \cdot 2^{N-1} + A_L B_L$$

$$s.A \cdot u.B = -A_S B_S \cdot 2^{2N-2} + (-A_S B_L + B_S A_L) \cdot 2^{N-1} + A_L B_L$$

$$u.A \cdot s.B = -A_S B_S \cdot 2^{2N-2} + (A_S B_L - B_S A_L) \cdot 2^{N-1} + A_L B_L$$

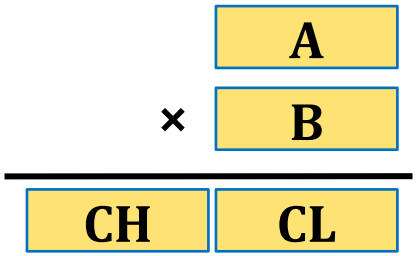
$$s.A \cdot s.B = A_S B_S \cdot 2^{2N-2} + (-A_S B_L - B_S A_L) \cdot 2^{N-1} + A_L B_L$$



Xと-Xの<u>最下位ビットは一致する</u>

$$-X = \overline{X} + 1$$

符号なし倍精度乗算



Nビット × Nビット = 2Nビット

```
unsigned int a, b;   uint64_t c;
c = (uint64_t)a * (uint64_t)b;
```

■ Nビット倍精度乗算: 符号の有無によって計算が異なる

(N=8)	11111001	(A)	249
(符号なし)	10001101	(B)	141
00000000	11111001		249
00000000	00000000		996
00000011	11100100		249
00000111	11001000	(C)	35109
00000000	00000000		
00000000	00000000		
00000000	00000000		
01111100	10000000		
10001001	00100101	C= {CH, CL}	

{ AH, A }

符号なしの場合：

$A \cdot 2^i$: Aの倍精度論理左シフト

（$A \cdot 2^i$の上位ワードをAHに格納）

B_i : B の i ビット目

$B_i = 1$ の場合 : C に$A \cdot 2^i$を 加算

→ C と $A \cdot 2^i$ の倍精度加算

（C = { CH, CL } : 2ワード）

符号なし倍精度乗算プログラム

単精度乗算

B_iビット判定

```
L0,  LDA  B
     CIR
     STA  B
     SZE
     BUN  L1
     BUN  L2
```

1

C += A

```
L1,  LDA  C
     ADD  A
     STA  C
```

2

A <<=1

```
     CLE
L2,  LDA  A
     CIL
     STA  A
```

3

終了判定

```
     ISZ  K
     BUN  L0
     HLT
```

4

<データ宣言部>

符号なし倍精度乗算

B_iビット判定

```
L0,  LDA  B
     CIR
     STA  B
     SZE
     BUN  L1
     BUN  L2
```

1

{CH,CL} += {AH,A}

```
L1,  LDA  CL
     ADD  A
     STA  CL
     CLA
     CIL
     ADD  CH
     ADD  AH
     STA  CH
```

2'

講義資料2 P11参照

講義資料2 P16参照

{AH,A} <<=1

```
     CLE
L2,  LDA  A
     CIL
     STA  A
     LDA  AH
     CIL
     STA  AH
```

3'

終了判定

```
     ISZ  K
     BUN  L0
     HLT
```

4

```
A,   DEC  249
B,   DEC  141
CL,  DEC  0
CH,  DEC  0
AH,  DEC  0
K,   DEC  -16
```

2の補数倍精度乗算

	A
×	B
CH	CL

```
int a, b;   int64_t c;
c = (int64_t)a * (int64_t)b;
```

符号なし：$u.B = B_S \cdot 2^{N-1} + B_L$
2の補数：$s.B = -B_S \cdot 2^{N-1} + B_L$

$$A \cdot B = B_{N-1} \cdot \left(-A \cdot 2^{N-1}\right) + \sum_{i=0}^{N-2} B_i \cdot \left(A \cdot 2^i\right)$$

(N=8)	11111001	(A)	−7
(2の補数)	10001101	(B)	−115
11111111	11111001		35
00000000	00000000		7
11111111	11100100		7
11111111	11001000	(C)	805
00000000	00000000		
00000000	00000000		
00000000	00000000		
00000011	10000000		
00000011	00100101		

$-A \cdot 2^{N-1}$

2の補数の場合：
$A \cdot 2^i$：Aの倍精度算術左シフト
（$A \cdot 2^i$の上位ワードAHを符号拡張）
B_i：B の i ビット目
$B_i = 1$ の場合：C に$A \cdot 2^i$を 加算
→ Cと $A \cdot 2^i$ の倍精度加算
最上位ビット$B_{N-1} = 1$ の場合：
C に$(-A \cdot 2^{N-1})$を加算

2の補数倍精度乗算プログラム

AH 符号拡張

B_i ビット判定

最上位ビット判定

```
        LDA  A
        SNA
        BUN  L0
        CLA
        CMA
        STA  AH
L0,     LDA  B
        CIR
        STA  B
        SZE
        BUN  L1
        BUN  L2
L1,     LDA  K
        INC
        SZA
        BUN  L1B
```

0 1 2a

{AH,A} *= −1

{CH,CL} += {AH,A}

```
        LDA  A
        CMA
        CIL
        LDA  AH
        CMA
        SZE
        INC
        STA  AH
L1B,    LDA  CL
        ADD  A
        STA  CL
        CLA
        CIL
        ADD  CH
        ADD  AH
        STA  CH
```

2b 2'

{AH,A} <<=1

終了判定

```
        CLE
L2,     LDA  A
        CIL
        STA  A
        LDA  AH
        CIL
        STA  AH
        ISZ  K
        BUN  L0
        HLT
A,      DEC  −7
B,      DEC  −115
CL,     DEC  0
CH,     DEC  0
AH,     DEC  0
K,      DEC  −16
```

3' 4

符号なし		1			2'	3'	4
2の補数	0	1	2a	2b	2'	3'	4

2の補数倍精度乗算プログラム

0

```
LDA  A
SNA
BUN  L0
CLA
CMA
STA  AH
```

AH符号拡張

IF (A < 0) :
CLA : AC ← 0
CMA : AC ← 0xFFFF
STA AH (0xFFFF)

2a

```
L1,  LDA  K
     INC
     SZA
     BUN  L1B
```

最上位ビット判定

SKIP-NEXT IF
(M[K] = −1)

2b

```
LDA  A
CMA
CIL
LDA  AH
CMA
SZE
INC
STA  AH
```

$E = \overline{A(15)}$

{AH,A} *= −1

AH	A
11111100	10000000

−AH	−A
00000011	10000000

Aの最上位ビット以外は0
→ $-A = A$

AH	A
11111100	00000000

−AH	−A
00000100	00000000

$-AH = \overline{AH} + \overline{A(15)}$

2の補数倍精度乗算プログラム

IF (A < 0) :
CLA : AC ← 0
CMA : AC ← 0xFFFF
STA AH (0xFFFF)

AH 符号拡張

```
      LDA  A
      SNA
      BUN  L0
      CLA
      CMA
      STA  AH
L0,   LDA  B
      CIR
      STA  B
      SZE
      BUN  L1
      BUN  L2
```

0

1

B_i ビット判定

最上位ビット判定

SKIP-NEXT IF (M[K]+1 = 0)

AH *= –1

{CH,CL} += {AH,A}

```
L1,   LDA  K
      INC
      SZA
      BUN  L1B
      LDA  AH
      CMA
      INC
      STA  AH
L1B,  LDA  CL
      ADD  A
      STA  CL
      CLA
      CIL
      ADD  CH
      ADD  AH
      STA  CH
```

2a

2b

2’

```
L2,   CLE
      LDA  A
      CIL
      STA  A
      LDA  AH
      CIL
      STA  AH
      ISZ  K
      BUN  L0
      HLT
A,    DEC  –7
B,    DEC  –115
CL,   DEC  0
CH,   DEC  0
AH,   DEC  0
K,    DEC  –16
```

3’

4

{AH,A} <<=1

終了判定

2b : A *= –1の処理は必要ない → P8参照

配列データ処理

■ 配列データ: 簡単かつ大変重要なデータ構造

- ほとんどのプログラミングで配列データは必須
- 配列データを効率的に処理するプログラミング技法は重要
  → 命令セットによってアプローチは異なる
- 本命令セットにおけるメモリ参照命令を効率的に使う方法を考える → 「間接アドレス」メモリ参照命令（ポインタ型データ）によって配列データのアクセスを行う

```
int d[100] = { 75, 62, ... , 23 };
int sum = 0, k;
for (k = 0; k < 100; k++)
    sum += d[i];
```

```
int d[100] = { 75, 62, ... , 23 };
int sum = 0, i, *pd = d;
for (k = 0; k < 100; k++, pd++)
    sum += *pd;
```

配列データの総和計算

配列データ総和計算

```
int d[100] = { 75, 62, ... , 23 };
int sum = 0, k = –100, *pd = d;
do {
    sum += *pd;
    pd = pd + 1;
    k = k + 1;
} while (k != 0);
```

アドレス指定：ORGに続くワードのアドレスを<16進定数>で指定

ORG <16進定数>

<命令ワード> or <データワード>

（アドレス指定のないワードは0番地から順次配置される）

アセンブリプログラム終端：

END

講義資料2 P9参照

データ100個

		アドレス
	CLA	000
L0,	ADD PD I	001
	ISZ PD	002
	ISZ K	003
	BUN L0	004
	STA SUM	005
	HLT	006
PD,	HEX 150	007
SUM,	DEC 0	008
K,	DEC –100	009
	ORG 150	
D,	DEC 75	150
	DEC 62	151
	. . . .	. . . .
	DEC 23	1B3
	END	

配列データ総和計算プログラム動作

M[ad] ← M[ad] + 1,
SKIP-NEXT IF (M[ad] + 1 = 0)

→

SKIP-NEXT IF ((++M[ad]) = 0)

			動作
	CLA		AC ← 0
L0,	ADD	PD I	AC ← AC + M[M[PD]]
	ISZ	PD	*SKIP-NEXT IF* ((++M[PD])=0)
	ISZ	K	*SKIP-NEXT IF* ((++M[K])=0)
	BUN	L0	PC ← L0
	STA	SUM	M[SUM] ← AC
	HLT		
PD,	HEX	150	
SUM,	DEC	0	
K,	DEC	–100	
	ORG	150	
D,	DEC	75	
	.....		

総和をACで保持
（メモリ退避なし）

SKIPは（まず）発生しない

100回後にSKIP

```
int d[100] = { 75, 62, ... , 23 };
int sum = 0, k = –100, *pd = d;
do {
    sum += *pd;
    pd = pd + 1;
    k = k + 1;
} while (k != 0);
```

→ PDとKをISZ命令でインクリメント
→ 総和をACに保持(メモリ退避なし)
→ 高速実行

配列データ総和計算プログラム動作

		アドレス
	CLA	000
L0,	ADD PD I	001
	ISZ PD	002
	ISZ K	003
	BUN L0	004
	STA SUM	005
	HLT	006
PD,	HEX 150	007
SUM,	DEC 0	008
K,	DEC –100	009
	ORG 150	
D,	DEC 75	150
	DEC 62	151
	.....	....
	DEC 23	1B3
	END	

	CLA	AC ← 0
L0,	ADD PD I	AC ← AC + M[0x150]
	ISZ PD	M[PD] ++ (0x151)
	ISZ K	M[K] ++ (–99)
	BUN L0	PC ← 0x001
L0,	ADD PD I	AC ← AC + M[0x151]
	ISZ PD	M[PD] ++ (0x152)
	ISZ K	M[K] ++ (–98)
	BUN L0	PC ← 0x001
	.....	.....
	ISZ PD	M[PD] ++ (0x1B3)
	ISZ K	M[K] ++ (–1)
	BUN L0	PC ← 0x001
L0,	ADD PD I	AC ← AC + M[0x1B3]
	ISZ PD	M[PD] ++ (0x1B4)
	ISZ K	M[K] ++ (0)
	BUN L0	*SKIPPED*
	STA SUM	M[SUM] ← AC
	HLT	

サブルーチン処理とプログラム構造化

- 大規模なソフトウェアではサブルーチン処理によるプログラム構造化は必須
- サブルーチン呼出し機構は命令セットに大きく依存する
- 本命令セットでは、BSA命令とBUN命令（間接アドレス）によってサブルーチン呼出し・復帰を行う

```
int SH4(int a) {
   return a << 4;
}
. . .
x = SH4(0x1234);
y = SH4(0x4321);
```

```
        LDA  X
        BSA  SH4
        STA  X
        LDA  Y
        BSA  SH4
        STA  Y
        HLT
X,      HEX  1234
Y,      HEX  4321
SH4,    HEX  0
        CIL
        CIL
        CIL
        CIL
        AND  MSK
        BUN  SH4  I
MSK,    HEX  FFF0
        END
```

サブルーチン処理動作

```
                          アドレス
        LDA  X            000
        BSA  SH4          001
        STA  X            002
        LDA  Y            003
        BSA  SH4          004
        STA  Y            005
        HLT               006
X,      HEX  1234         007
Y,      HEX  4321         008
SH4,    HEX  0            009
        CIL               00A
        CIL               00B
        CIL               00C
        CIL               00D
        AND  MSK          00E
        BUN  SH4  I       00F
MSK,    HEX  FFF0         010
        END
```

2. 各命令の実行時は、すでに $PC \leftarrow PC + 1$ が実行されている

講義資料2 P5参照

BUN adi	$PC \leftarrow ad$
BSA adi	$M[ad] \leftarrow PC, PC \leftarrow ad + 1$

LDA X	AC ← M[X] (0x1234)
BSA SH4	M[SH4] ← PC(0x002), PC ← SH4 + 1
CIL x 4回	AC の4ビット左シフト（下位4ビットには不定値とAC上位3ビットが入る）
AND MSK	AC ← AC & M[MSK] (0xFFF0のANDにより下位4ビットが0000になる)
BUN SH4 I	PC ← M[SH4] (0x002)
STA X	M[X] ← AC (0x2340)

サブルーチン呼出しと復帰

```
                          アドレス
        LDA  X            000
        BSA  SH4          001
        STA  X            002
        LDA  Y            003
        BSA  SH4          004
        STA  Y            005
        HLT               006
X,      HEX  1234         007
Y,      HEX  4321         008
SH4,    HEX  0            009
        CIL               00A
        CIL               00B
        CIL               00C
        CIL               00D
        AND  MSK          00E
        BUN  SH4  I       00F
MSK,    HEX  FFF0         010
        END
```

BSA SH4	M[SH4] ← PC(0x002), PC ← SH4 + 1

BSA命令の次のアドレスがSH4番地に格納される（復帰アドレス）

SH4+1番地にサブルーチンの最初の命令が配置されている

SH4サブルーチン内部処理（引数：AC）

BUN SH4 I	PC ← M[SH4] (0x002)

SH4番地に格納されている復帰アドレスを間接アドレスによってPCに代入（間接アドレス無条件分岐命令）

サブルーチン戻り値：AC

複数のサブルーチン引数の引渡し方法

- 第1引数 : ACで引渡す
- 第2引数以降 : BSA命令の次の番地から配置する
- サブルーチン内部での第2引数以降は間接アドレスメモリ参照命令で読み出す

第 1引数

第 2引数

```
int OR(int a, int b) {
   return a | b;
}
. . .
x = OR(0x7B95, 0x3AF6);
```

0x7B95 | 0x3AF6 = 0x7BF7

$$X \mid Y = \overline{\overline{X} \,\&\, \overline{Y}}$$

```
        LDA  X
        BSA  OR
        HEX  3AF6
        STA  Y
        HLT
X,      HEX  7B95
Y,      HEX  0
OR,     HEX  0
        CMA
        STA  TMP
        LDA  OR I
        CMA
        AND TMP
        CMA
        ISZ  OR
        BUN  OR I
TMP,    HEX  0
        END
```

複数引数のサブルーチン処理動作

```
                          アドレス
        LDA  X            000
        BSA  OR           001
        HEX  3AF6         002
        STA  Y            003
        HLT               004
X,      HEX  7B95         005
Y,      HEX  0            006
OR,     HEX  0            007
        CMA               008
        STA  TMP          009
        LDA  OR  I        00A
        CMA               00B
        AND TMP           00C
        CMA               00D
        ISZ  OR           00E
        BUN  OR  I        00F
TMP,    HEX  0            010
        END
```

LDA X	AC ← M[X] (0x7B95) (X) 第 1引数
BSA OR	M[OR] ← PC (0x002), PC ← OR + 1
CMA STA TMP	M[TMP] ← $\overline{AC}$ (0x846A) ($\overline{X}$)
LDA OR I	AC ← M[M[OR]] 第 2引数 (AC ← M[0x002] (0x3AF6)) (Y)
CMA	AC ← $\overline{AC}$ (0xC509) ($\overline{Y}$)
AND TMP	AC ← AC & M[TMP] ($\overline{X}$ & $\overline{Y}$) (0xC509 & 0x846A → 0x8408)
CMA	AC ← $\overline{AC}$ (0x07BF7) ($\overline{\overline{X} \& \overline{Y}}$)
ISZ OR	M[OR] ← M[OR] + 1 (0x003)
BUN OR I	PC ← M[OR] (0x003)
STA Y	M[Y] ← AC (0x07BF7)

複数引数のサブルーチン処理動作

```
                          アドレス
        LDA  X            000
        BSA  OR           001
        HEX  3AF6         002
        STA  Y            003
        HLT               004
X,      HEX  7B95         005
Y,      HEX  0            006
OR,     HEX  0            007
        CMA               008
        STA  TMP          009
        LDA  OR  I        00A
        CMA               00B
        AND TMP           00C
        CMA               00D
        ISZ  OR           00E
        BUN  OR  I        00F
TMP,    HEX  0            010
        END
```

LDA X	AC ← M[X] (0x7B95) (*X*)
BSA OR	M[OR] ← PC (0x002), PC ← OR + 1

第1引数 : ACで引渡す

LDA OR I	AC ← M[M[OR]] (AC ← M[0x002] (0x3AF6)) (*Y*)

第2引数 : M[OR]番地に格納されているデータをORラベルの間接アドレスで読み出す

ISZ OR	M[OR] ← M[OR] + 1 (0x003)

サブルーチン復帰アドレスは、BSA OR命令のアドレス（0x001）の2つ先（0x003）であるので、M[OR]をインクリメントする

BUN OR I	PC ← M[OR] (0x003)