基礎ソフト 実験レポート 2

C 過程 S-15 組 1TE20137W 2022/10/24

柳 鷹

問 4

(1)

求めるプログラムは以下である。

```
************************************************
**
** アドレスレジスタ a1 で指定されるアドレスから始まる連続する 2n バイトのメモリの内容をアドレスレジスタ a2 で指定されるアドレスから始まる領域へと転送するサブルーチンの作成
**
************************************************
.section .text
************************************************
** メインルーチン
************************************************
start:
        jsr   COPY    /* サブルーチンへ */
        stop  #0x2700  /* プログラム終了 */

************************************************
** データを転送するサブルーチン
** a1:読み込みデータのアドレス
** a2:書き込みデータのアドレス
** d0:書き込み回数
************************************************

COPY:
        movem.l %d0-%d7/%a0-%a6,-(%sp) /* レジスタ退避 */
        lea.l   READ, %a1 /* 読み込みデータ READ の先頭アドレスを a1 レジスタへ */
        lea.l   WRITE, %a2 /* 書き込み先 WRITE の先頭アドレスを a2 レジスタへ */

        moveq.l #0, %d0 /* 書き込み回数初期化 */

LOOP:
        move.b  (%a1)+, (%a2)+          /* READ から WIRTE へ */
        addq.l  #1, %d0 /* 書き込み回数 +1 */
        cmp.l   #LENGTH, %d0 /* 書き込み回数 < LENGTH ならば */
        bne     LOOP /* LOOP へ */

        movem.l (%sp)+,%d0-%d7/%a0-%a6 /* レジスタ復帰 */
        rts /* サブルーチンから復帰 */
************************************************
** データエリア
************************************************
.equ      LENGTH,20

.section .data
READ:
        .ascii  "TAROU          "   /* NAME */
```

```
WRITE:
        .ds.b 20        /* 書き込みデータ出力先 */

.end
```

(2)

プログラムの目的：

アドレスレジスタ a1 で指定されるアドレスから始まる連続する 2n バイトのメモリの内容をアドレスレジスタ a2 で指定されるアドレスから始まる領域へと転送するサブルーチンの作成

レジスタ用途：

a1:読み込みデータのアドレス

a2:書き込みデータのアドレス

d0:書き込み回数

メモリ内のデータ配置：

0x000400 番地から 0x00042d まではテキスト領域で、0x000430 番地から 0 がデータ領域（0x000430+データサイズまでが読み込みデータ、それ以降書き込み領域）である。

(3)

・読み込みデータ（転送元）を「.section .data　READ:」以下に記述する

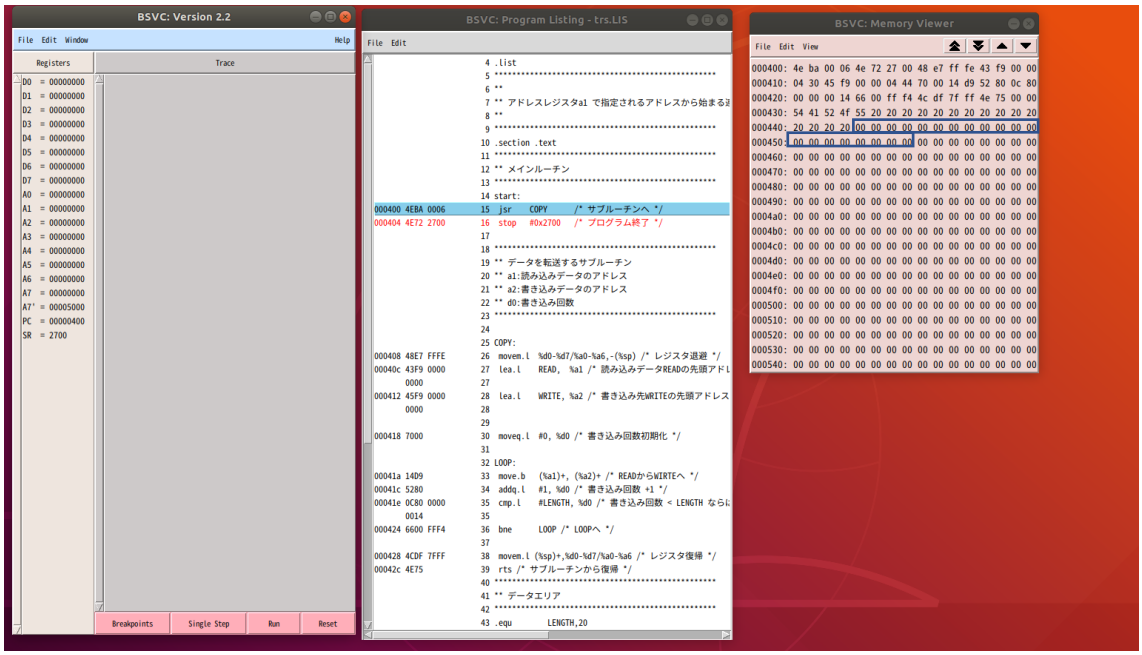
・読み込みデータのサイズ（何文字か）をシンボルである LENGTH に入れる

（ASCII コードに変換して 2 バイトを 1 としている）

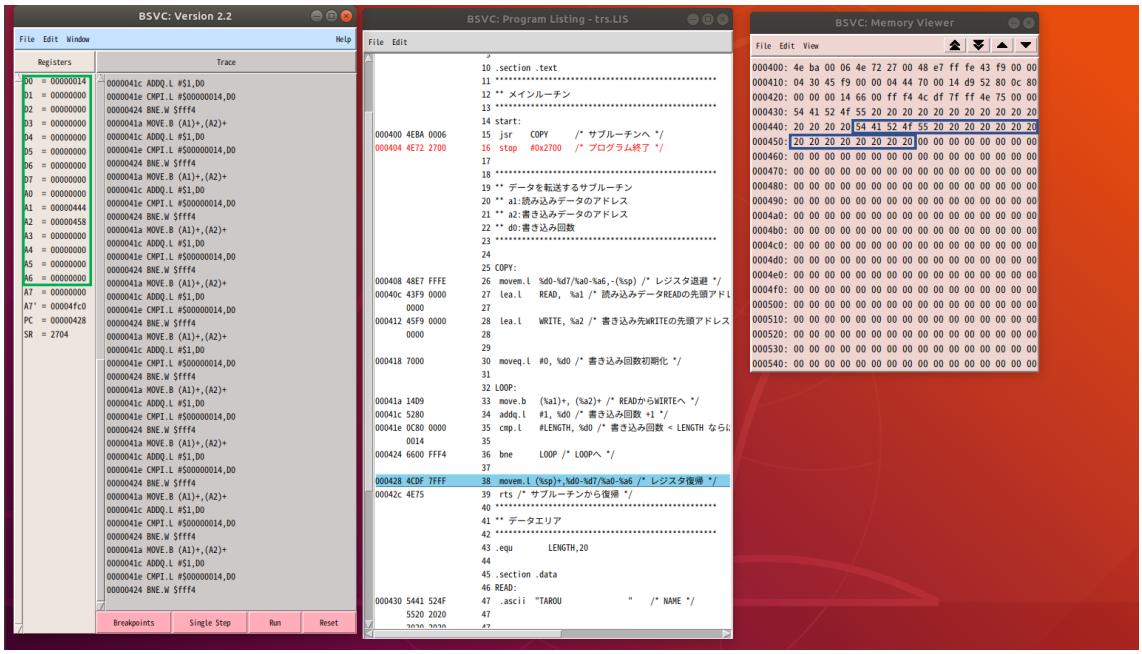
(4)

プログラムの実行結果をメインルーチン実行前、サブルーチンレジスタ復帰直前、メインルーチン実行後のメモリ・レジスタの様子を比較しながら示す。

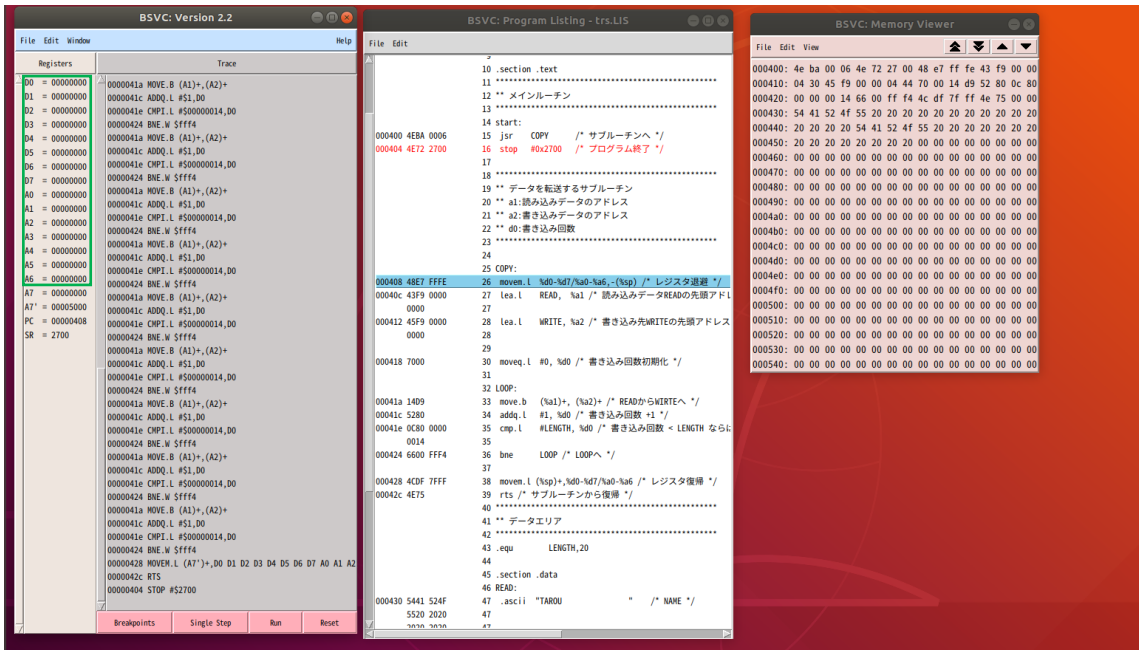
「メインルーチン実行前」


BSVC: Version 2.2
File Edit Window Help
Registers
Trace
D0 = 00000000
D1 = 00000000
D2 = 00000000
D3 = 00000000
D4 = 00000000
D5 = 00000000
D6 = 00000000
D7 = 00000000
A0 = 00000000
A1 = 00000000
A2 = 00000000
A3 = 00000000
A4 = 00000000
A5 = 00000000
A6 = 00000000
A7 = 00000000
A7' = 00005000
PC = 00000400
SR = 2700
Breakpoints
Single Step
Run
Reset
BSVC: Program Listing - trs.LIS
File Edit
4 .list
5 ********
6 **
7 ** アドレスレジスタa1 で指定されるアドレスから始まる
8 **
9 ********
10 .section .text
11 ********
12 ** メインルーチン
13 ********
14 start:
000400 4EBA 0006 15 jsr COPY /* サブルーチンへ */
000404 4E72 2700 16 stop #0x2700 /* プログラム終了 */
17
18 ********
19 ** データを転送するサブルーチン
20 ** a1:読み込みデータのアドレス
21 ** a2:書き込みデータのアドレス
22 ** d0:書き込み回数
23 ********
24
25 COPY:
000408 48E7 FFFE 26 movem.l %d0-%d7/%a0-%a6,-(%sp) /* レジスタ退避 */
00040c 43F9 0000 27 lea.l READ, %a1 /* 読み込みデータREADの先頭アド
0000
000412 45F9 0000 28 lea.l WRITE, %a2 /* 書き込み先WRITEの先頭アドレス
0000
29
000418 7000 30 moveq.l #0, %d0 /* 書き込み回数初期化 */
31
32 LOOP:
00041a 14D9 33 move.b (%a1)+, (%a2)+ /* READからWIRTEへ */
00041c 5280 34 addq.l #1, %d0 /* 書き込み回数 +1 */
00041e 0C80 0000 35 cmp.l #LENGTH, %d0 /* 書き込み回数 < LENGTH ならば
0014
000424 6600 FFF4 36 bne LOOP /* LOOPへ */
37
000428 4CDF 7FFF 38 movem.l (%sp)+,%d0-%d7/%a0-%a6 /* レジスタ復帰 */
00042c 4E75 39 rts /* サブルーチンから復帰 */
40 ********
41 ** データエリア
42 ********
43 .equ LENGTH,20
BSVC: Memory Viewer
File Edit View
000400: 4e ba 00 06 4e 72 27 00 48 e7 ff fe 43 f9 00 00
000410: 04 30 45 f9 00 00 04 44 70 00 14 d9 52 80 0c 80
000420: 00 00 00 14 66 00 ff f4 4c df 7f ff 4e 75 00 00
000430: 54 41 52 4f 55 20 20 20 20 20 20 20 20 20 20 20
000440: 20 20 20 20 00 00 00 00 00 00 00 00 00 00 00 00
000450: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000460: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000470: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000480: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000490: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0004a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0004b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0004c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0004d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0004e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0004f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000500: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000510: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000520: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000530: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000540: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

「サブルーチンレジスタ直前」



「メインルーチン実行後」



まず、メインルーチン実行前とサブルーチンレジスタ復帰前を比較すると、青枠で囲まれたメモリ（0x000444 番地～0x000457 番地）に 0x000430 番地～0x000443 番地のデータが転送されているのが確認できる。

また、サブルーチンレジスタ復帰前とメインルーチン実行後を比較すると、緑枠で囲まれた d0 から d7、a1 から a6 のレジスタが復帰している（サブルーチン実行前と同じ状態になっている）ことが分かる。

(5)

転送後に、転送元の領域を 0 で上書きする必要があると考える。
なぜなら、「複製」ではなく「転送」であるからである。確かに上記のコードでは a1 レジスタで指定されるアドレスから始まるメモリの領域と a2 レジスタで指定されるアドレスから始まる領域は「複製」された同じ状態になっている。そして、レジスタ退避して転送元の先頭アドレスは実行後には分からない状態にある。しかしながら、メモリを探索すれば転送元のデータは元の状態と変わらないため、「転送」とは言えない。(データを特定のメモリ領域から転送することで削除したつもりでも、メモリアドレスを記憶しているレジスタから削除しただけでメモリから完全に削除しきれていないということ)

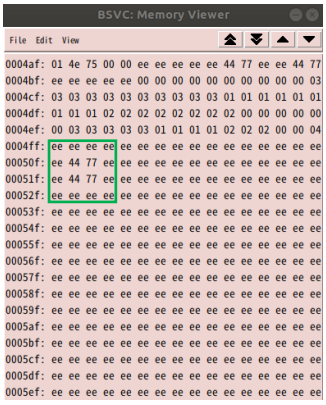
(6)

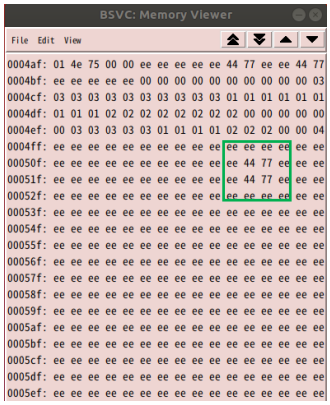
今後可能なら改良すべき点として、本課題の前提条件であった「アドレスレジスタ a1 で指定される領域と、アドレスレジスタ a2 で指定される領域は重ならないものとする」を考慮したプログラムの作成が考えられる。例えば、メモリ全体の大きさの把握し、転送先の領域の確保を転送前に行うなどが挙げられる。

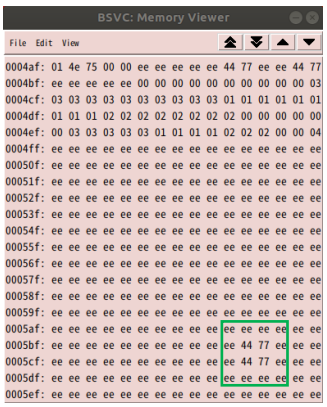
問 5

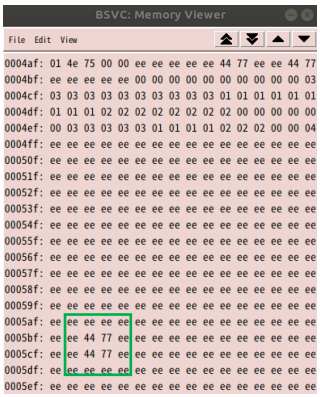
(1)

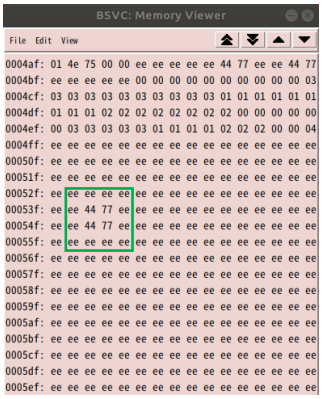
描画する絵を動かす方向を示すデータ領域の「CONTROL」の数値が変わるタイミングを比較することで与えられたプログラムの動作確認を行う。

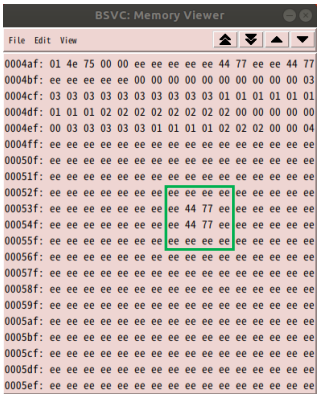


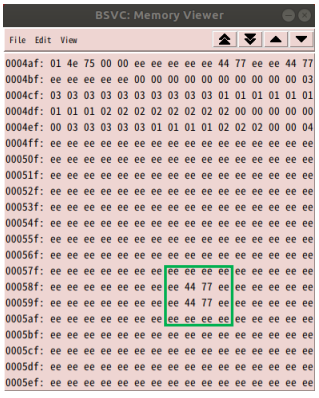


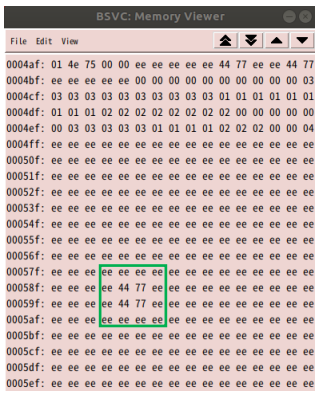


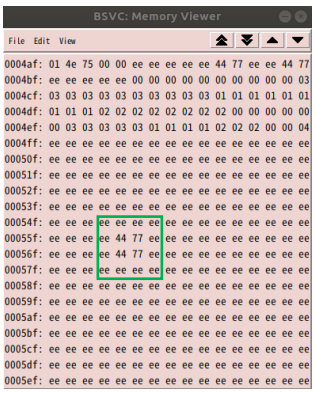


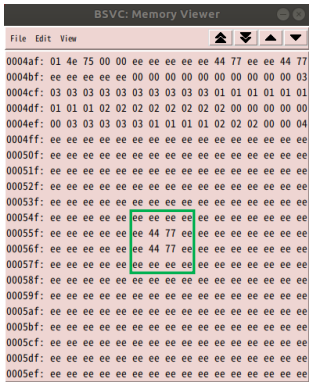


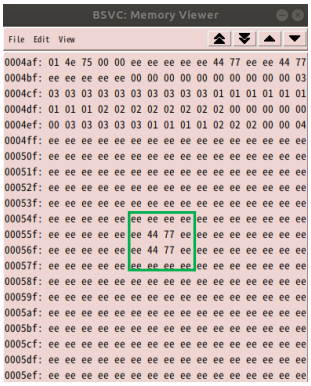








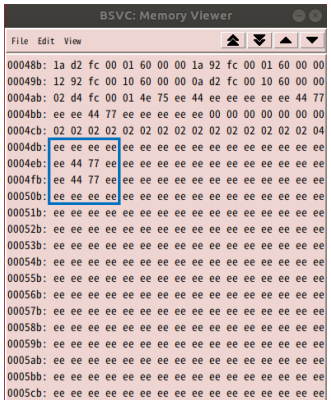


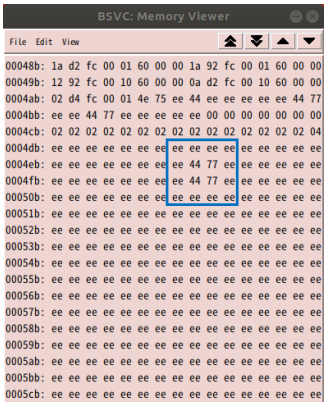


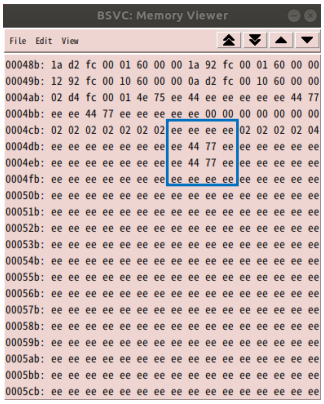
以上の画面のハードコピー中、緑枠で囲まれた絵が”ee”に埋め尽くされた領域を時計回りに渦を巻くように動いていることが確認できる。

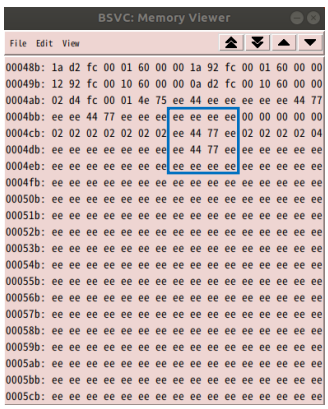
(2)

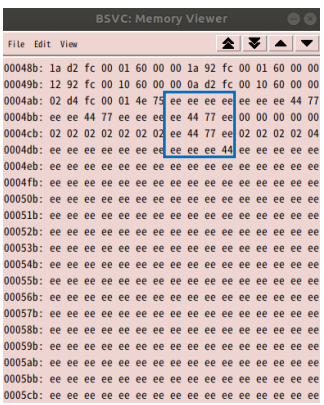
まず、(1)と同様にメモリの領域の変化を以下に示す。

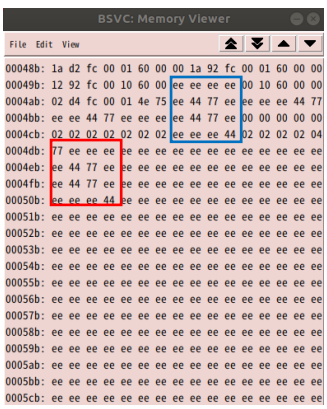


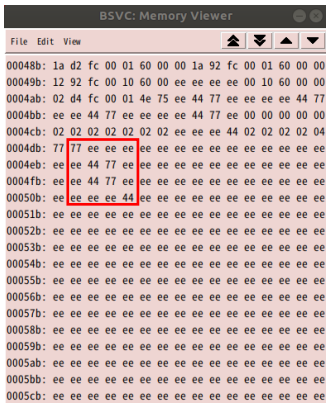


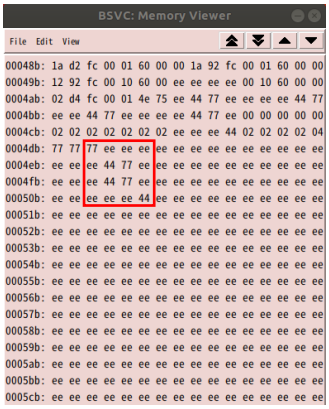


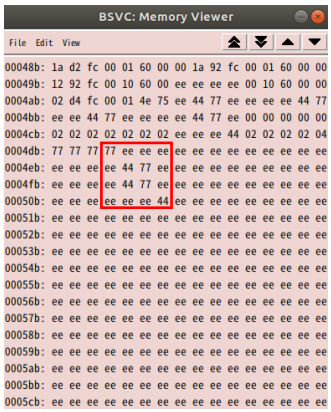


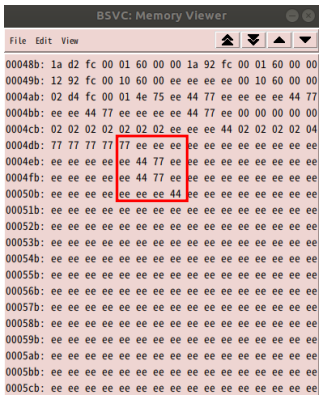


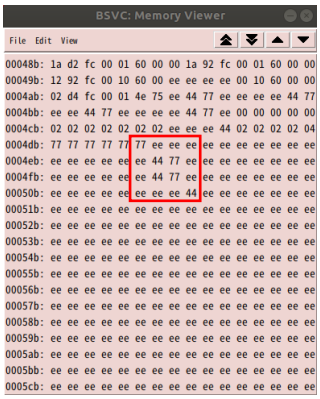


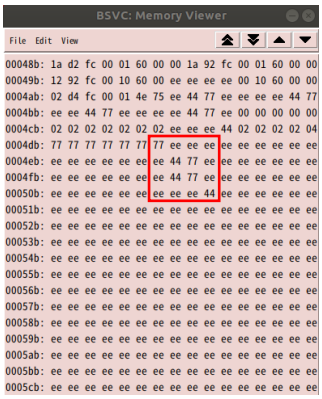


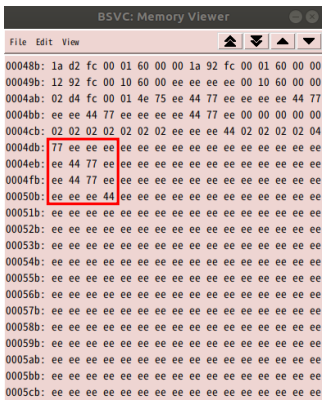












まず、以上の画面のハードコピー中、青枠で囲まれた絵が(1)同様に描画する絵を動かす方向を示すデータ領域の「CONTROL」の数値に従って動いていることが確認できる。このとき、ハードコピー3 枚目から 5 枚目にかけて、本来描画されるべき範囲である「VIEWTOP」以下の範囲を超えて、他のデータ領域に侵食してしまっている。

3 枚目では「CONTROL」の一部を変えているが、次の動きには影響がまだない。

4 枚目では実行済みの「CONTROL」の一部を変え、絵となるデータの一部を侵食し始めるが、上書きした部分が”ee”と同じであったため、次の動きに影響はない。

5 枚目では絵となるデータの一部を侵食し、異なる値で上書きした。

しかし、6 枚目ではここで絵となるデータの先頭アドレスを格納する a1 レジスタの値が変わり、絵となるデータの一部を他の値に変え、さらに、データ領域を超えてテキスト領域に侵食して UP と DOWN の一部に影響している。
ゆえに、データ領域において 6 枚目から 12 枚目の赤枠に囲まれた部分が RIGHT によって繰り返される。

以上が、「メモリ破壊」の影響と原因である。

(3)

上記のメモリ破壊を防ぐため、描画領域の境界条件として絵となるデータの先頭アドレスを格納する a1 レジスタの値と描画領域の大きさを示す#AREASIZE を考慮したプログラムに改良すべきである。

問 6

(設問 1),(設問 2)に必要なアセンブラプログラムを（まとめて）以下に記す。

```
.section .text
*****************************
** メインルーチン
** a0:書き込む/読み出すデータの始めアドレス
** a4:書き込みデータの一時保存アドレス（データ量が 256 バイトを超える場合のみ用いる）
** a5:読み出しデータの一時保存アドレス（データ量が 256 バイトを超える場合のみ用いる）
** d3:キューの書き込み上限
** d4:書き込み/読み出しデータ数
*****************************
Start:
        jsr    Init_Q /* キューの初期化処理 */
        lea.l  Data_to_Que, %a0 /* 書き込むデータの先頭アドレスを a0 レジスタへ */
        move.l #LENGTH, %d4 /* 書き込み回数を d4 レジスタへ */
        move.l #257, %d3 /* 書き込み上限 */

Loop1:
        subq.w #1, %d4 /* 書き込みデータ数 - 1 */
        bcs    End_put /* d4 - 1 < 0 ならば書き込み完全終了 */
        subq.w #1, %d3 /* 書き込み上限 - 1 */
        bcs    End_put /* d3 - 1 < 0 ならば書き込み一旦終了 */
        jsr    QueueIn /* 書き込み処理 */
        bra    Loop1 /* ループに戻る */
```

```
End_put:
        movea.l %a0,%a4 /* 次に書き込むデータの先頭アドレスを一時保存 */
        lea.l  COPY, %a0 /* 書き込むデータの先頭アドレスを a0 レジスタへ */
        move.l  #LENGTH, %d4 /* 読み出し回数を d4 レジスタへ */
        move.l  #257, %d3 /* 読み出し上限 */

Loop2:
        subq.w  #1, %d4 /* 読み出しデータ数 - 1 */
        bcs    End_program /* d4 - 1 < 0 ならば読み出し完全終了 */
        subq.w  #1, %d3 /* 読み出し上限 - 1 */
        bcs    End_get /* d3 - 1 < 0 ならば読み出し一旦終了 */
        jsr    QueueOut /* 読み出し処理 */
        bra    Loop2 /* ループに戻る */

End_get:
        movea.l %a4,%a0 /* 一時保存していた次に書き込むデータの先頭アドレスを a0 レジスタに戻す */
        move.l  #257, %d3 /* 書き込み上限 */
        bra    Loop1

End_program:
        stop   #0x2700 /* 終了 */
**********************
** キューの初期化処理（サブルーチン 1）(p15)
**********************
Init_Q:
        lea.l  BF_START, %a2 /* キューのデータ領域の先頭アドレスを a2 レジスタへ */
        move.l  %a2, PUT_PTR /* キューのデータ領域の先頭アドレス（a2）を書き込み用のポインタへ */
        move.l  %a2, GET_PTR /* キューのデータ領域の先頭アドレス（a2）を読み出し用のポインタへ */
        move.b  #0xff, PUT_FLG /* キューは「空」なので書き込み「許可」に設定 */
        move.b  #0x00, GET_FLG /* キューは「空」なので読み出し「禁止」に設定 */
        rts

**********************************
** QueueIn キューへのデータ書き込み（サブルーチン 2）
** a0:書き込むデータのアドレス
** d0:結果(00:失敗, 00 以外:成功)
**********************************
QueueIn:
        jsr PUT_BUF/* キューへの書き込み*/
        rts /* メインルーチンへ */
****************************************
** キューへのデータ書き込み PUT_BUF（サブルーチン 2-1）(p15)
** a0:書き込むデータのアドレス
** d0:結果(00:失敗, 00 以外:成功)
****************************************
PUT_BUF:
        movem.l %a1-%a4,-(%sp) /* レジスタ退避 */
        move.b  PUT_FLG, %d0 /* 書き込み許可フラグを d0 レジスタへ */
        cmp.b   #0x00, %d0 /* 書き込み許可フラグ 0x00:禁止 | 0xff:許可 */
```

```
        beq    PUT_BUF_Finish /* 0x00 で書き込み「禁止」なら終了 */
        movea.l PUT_PTR, %a1 /* 書き込み用のポインタアドレスを a1 レジスタへ */
        move.b  (%a0)+,(%a1)+ /* データをキューへ入れ、書き込むデータアドレスと書き込み用ポインタを更新（+2） */
        lea.l   BF_END, %a3 /* キューデータ領域の末尾アドレスを a3 レジスタへ */
        cmpa.l  %a3, %a1 /* 次書き込もうとしているアドレス a1 とキューデータ領域の末尾アドレス a3 を比較 */
        bls    PUT_BUF_STEP1 /* a1 < a3   ならば、そのまま PUT_BUF_STEP1 へ */
        lea.l  BF_START, %a2 /* 次書き込もうとしているアドレス a1 とキューデータ領域の末尾アドレス a3 を超えているならば、キューデータ領域の先頭アドレスを a2 レジスタへ*/
        movea.l %a2,%a1 /* 書き込み用ポインタ(a1)をキューデータ領域の先頭アドレス(a2)に更新 */
PUT_BUF_STEP1:
        move.l  %a1,PUT_PTR /* 書き込み用ポインタを更新 */
        cmpa.l  GET_PTR,%a1 /* 読み出し用ポインタと書き込み用ポインタ(a1)を比較する */
        bne     PUT_BUF_STEP2 /* 書き込み用ポインタと読み出し用ポインタが異なる、つまりまだ書き込むことができればそのまま PUT_BUF_STEP2 へ */
        move.b  #0x00,PUT_FLG /* 書き込み用ポインタと読み出し用ポインタが同じなら、キューは一杯なので書き込み用ポインタを書き込み「禁止」に */
PUT_BUF_STEP2:
        move.b  #0xff, GET_FLG /* キューが一杯でなくなったので読み出し用ポインタを「許可」に */
PUT_BUF_Finish:
        movem.l (%sp)+, %a1-%a4 /* レジスタ復帰 */
        rts /* サブルーチンを抜ける */



**********************************
** QueueOut キューからのデータ読み出し（サブルーチン 3）
** a0:読み出すデータのアドレス
** d0:結果(00:失敗, 00 以外:成功)
**********************************
QueueOut:
        jsr GET_BUF/* キューへの書き込み*/
        rts /* メインルーチンへ */
**************************************
** キューへからのデータ読み出し GET_BUF（サブルーチン 3-1）(p16)
** a0:読み出すデータのアドレス
** d0:結果(00:失敗, 00 以外:成功)
**************************************
GET_BUF:
        movem.l %a1-%a4,-(%sp) /* レジスタ退避 */
        move.b  GET_FLG, %d0 /* 読み出し許可フラグを d0 レジスタへ */
        cmp.b   #0x00, %d0 /* 読み出し許可フラグ 0x00:禁止｜0xff:許可 */
        beq     GET_BUF_Finish /* 0x00 で読み出し「禁止」なら終了 */
        movea.l GET_PTR, %a1 /* 読み出し用のポインタアドレスを a1 レジスタへ */
        move.b  (%a1)+,(%a0)+ /* データをキューへ入れ、読み出しデータアドレス a1 と読み出しデータ出力先のアドレス a0 を更新（+2） */
        lea.l   BF_END, %a3 /* キューデータ領域の末尾アドレスを a3 レジスタへ */
        cmpa.l  %a3, %a1 /* 次書き込もうとしているアドレス a1 とキューデータ領域の末尾アドレス a3 を比較
```

```
*/
        bls    GET_BUF_STEP1 /* a1 < a3   ならば、そのまま GET_BUF_STEP1 へ */
        lea.l  BF_START, %a2 /* 次読み込もうとしているアドレス a1 とキューデータ領域の末尾アドレス a3 を超えているならば、キューデータ領域の先頭アドレスを a2 レジスタへ*/
        movea.l %a2,%a1 /* 読み込み用ポインタ(a1)をキューデータ領域の先頭アドレス(a2)に更新 */
GET_BUF_STEP1:
        move.l  %a1,GET_PTR /* 読み出し用ポインタを更新 */
        cmpa.l  PUT_PTR,%a1 /* 書き込み用ポインタと読み出し用ポインタ(a1)を比較する */
        bne     GET_BUF_STEP2 /* 書き込み用ポインタと読み出し用ポインタが異なる、つまりまだ読み出すことができればそのまま GET_BUF_STEP2 へ */
        move.b  #0x00,GET_FLG /* 書き込み用ポインタと読み出し用ポインタが同じなら、キューは空なので書き込み用ポインタを読み出し「禁止」に */
GET_BUF_STEP2:
        move.b  #0xff, GET_FLG /* キューが空でなくなったので読み出し用ポインタを「許可」に */
GET_BUF_Finish:
        movea.l  %a0, %a5 /* 更新された読み出しデータ出力先のアドレス a0 を COPY に一時保存 */
        movem.l (%sp)+, %a1-%a4 /* レジスタ復帰 */
        rts /* サブルーチンを抜ける */




*******************************
.section .data
*******************************
**
**
** キュー用のメモリ領域確保
**
**
*******************************
**キューのデータ領域は 256 バイト(p12)
        .equ        B_SIZE, 256
*******************************
**キューデータ領域の先頭アドレス(p12)
BF_START:           .ds.b       B_SIZE-1
*******************************
**キューデータ領域の末尾アドレス(p13)
BF_END:             .ds.b       1
*******************************
**キューに書き込むべきデータアドレスを管理するポインタ(p13)
PUT_PTR: .ds.l      1
*******************************
**キューから読み出すデータアドレスを管理するポインタ(p13)
GET_PTR: .ds.l      1
*******************************
**書き込み許可フラグ(p14)
**0x00 -> 書き込み禁止（buffer FULL）
```

```
**0xFF -> 書き込み許可
PUT_FLG: .ds.b          1
*****************************
**読み出し許可フラグ(p14)
**0x00 -> 読み出し禁止（buffer EMPTY）
**0xFF -> 読み出し許可
GET_FLG: .ds.b          1
*****************************



*****************************
**
** 書き込むデータ（サンプル）
**
*****************************
** 書き込みデータの長さ
                .equ       LENGTH,3
*****************************
** 書き込むデータ
Data_to_Que:            .ascii          "ABC"
*****************************

*****************************
**
** 読み出し先
**
*****************************
COPY:
                .ds.b 20        /* 読み出しデータ出力先 */

.end
```

(設問 1)

求めるサブルーチンプログラムは、上記の QueueIn と QueueOut である。

プログラムの目的：

サブルーチン化した QueueIn 目的はキューへの指定されたメモリアドレスのデータの書き込み機能の作成である。また同様にサブルーチン化した QueueOut の目的はキューからのデータの指定されたメモリアドレスのデータの書き込み機能の作成である。

レジスタ用途：

a0:書き込む/読み出すデータの始めアドレス

a4:書き込みデータの一時保存アドレス（データ量が 256 バイトを超える場合のみ用いる）

a5:読み出しデータの一時保存アドレス（データ量が 256 バイトを超える場合のみ用いる）

d3:キューの書き込み上限

d4:書き込み/読み出しデータ数

メモリ内のデータ配置：

0x000400 番地から 0x00053b まではテキスト領域で、0x00053c 番地以降がデータ領域である。（入力データによってデータ領域は可変である。）

プログラムの入力：

・読み込みデータ（転送元）を 0x000646 番地「.section .data　Data_to_Que:」以下に記述する

・読み込みデータのサイズ（何文字か）をシンボルである LENGTH に入れる

（ASCII コードに変換して 2 バイトを 1 としている）

プログラムの出力：

・0x00053c 番地から 0x00063b はキューを表す

・0x000646 番地から(0x000646 + LENGTH)番地はデータの入力

・(0x000646 + LENGTH)番地から(0x000646 + 2×LENGTH)番地がキューからの読み出し

プログラムの工夫した点：

本プログラムで工夫した点は二つある。

一つ目は、入力されたデータがなくなったらキューへの書き込みをやめ、直ちにキューからの読み出しを可能にしている点である。

二つ目は、入力されたデータがキューのサイズよりも大きかった場合、キューが一杯になった時点で、キューの読み出しを行い、またキューの書き込み・読み出しの続行を可能にしている点である。これを達成するために、a4,a5 レジスタをアドレス一時保存用とし、レジスタ退避・復帰を行った。

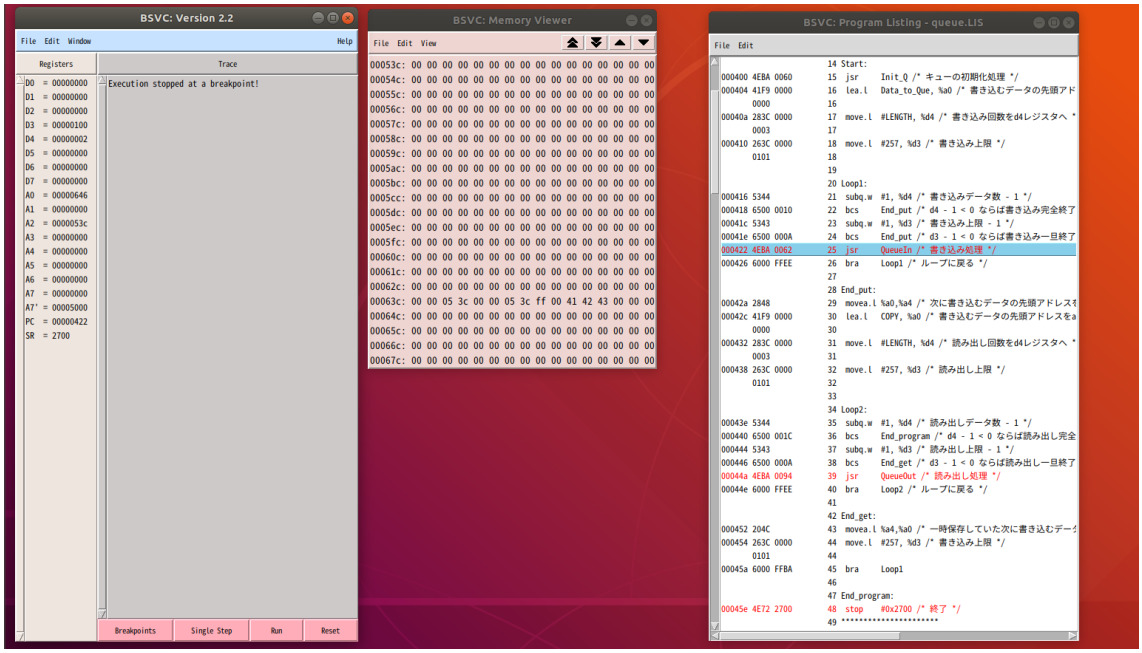
(設問 2)

求めるプログラムは上記のとおりである。

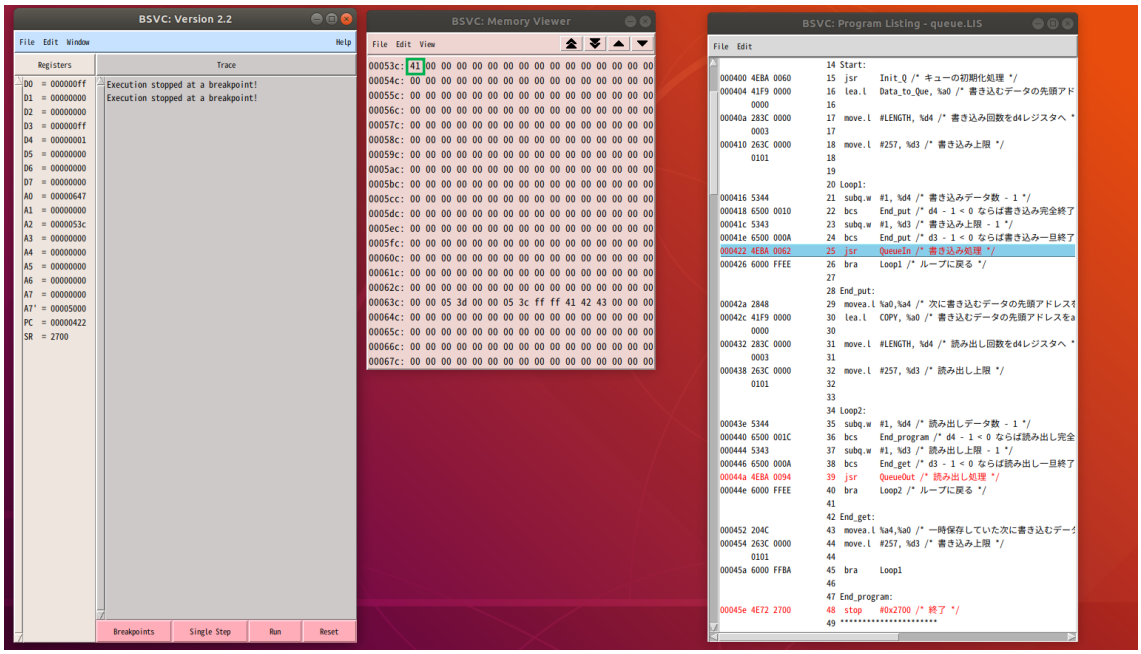
(設問 3a)

プログラムの実行結果を、キューの変化が分かりやすいフェーズのメモリ配置を示すことで報告する。

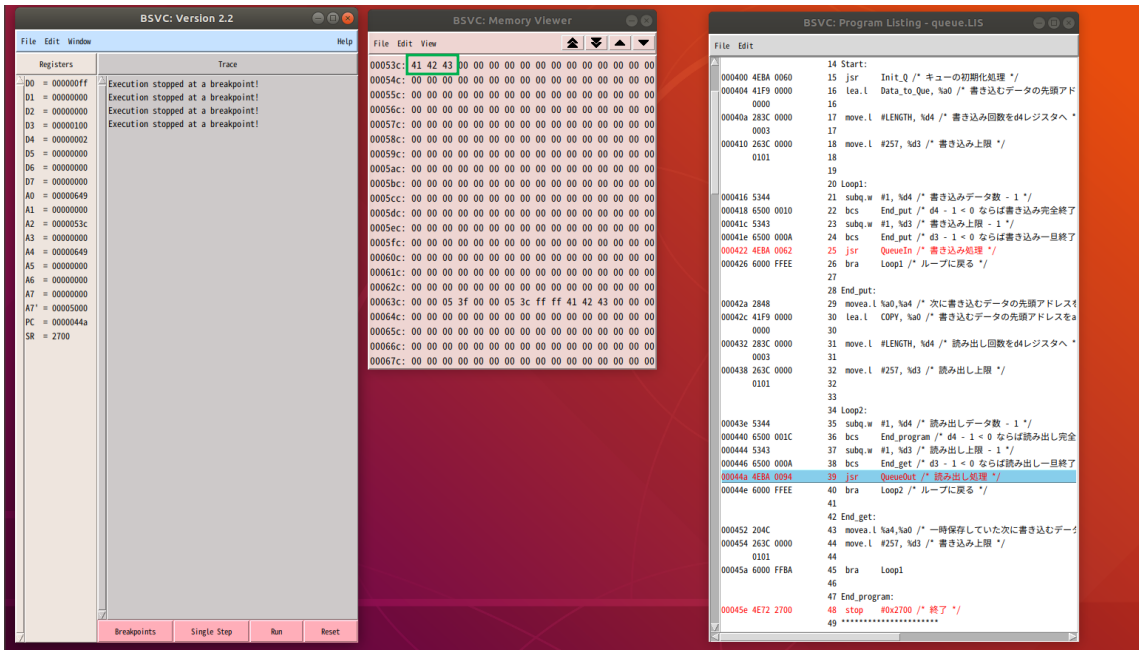
「キュー初期化後」



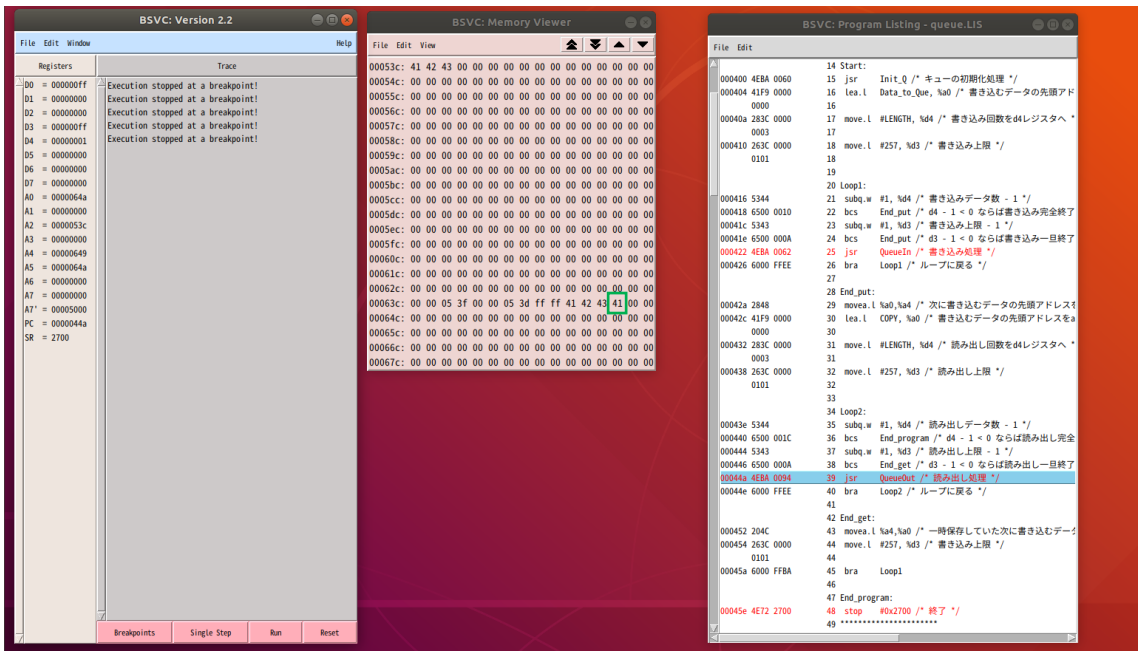
「キューへの書き込み一回目終了」



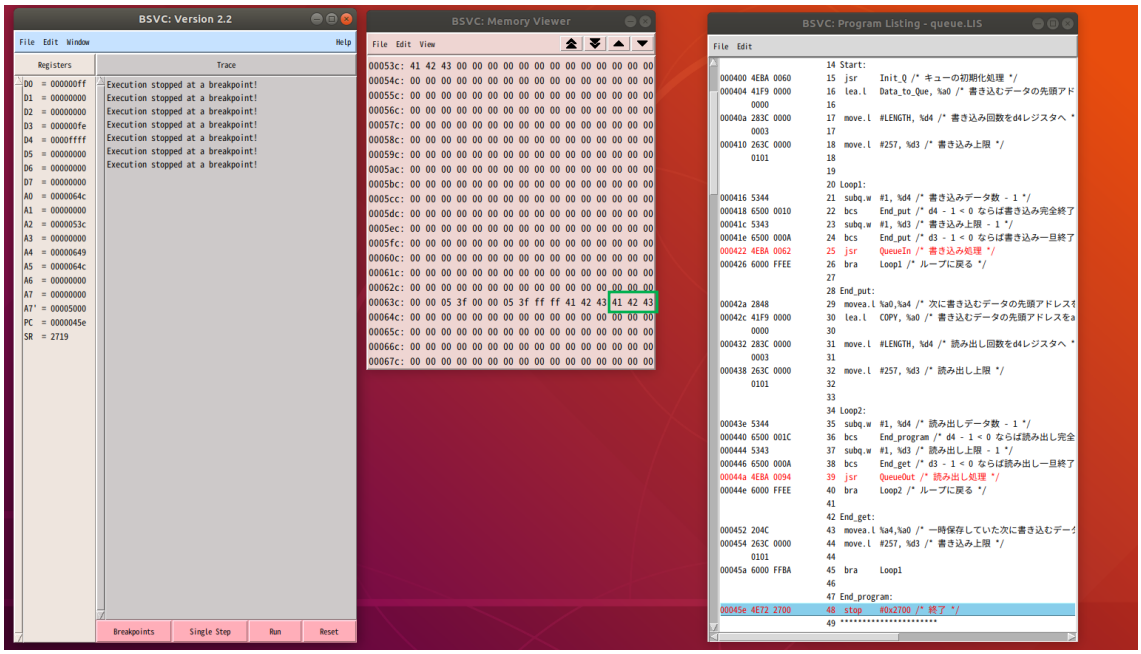
「キューへの書き込み終了・キューからの読み出し開始前」



「キューからの読み出し一回目終了」



「キューからの読み出し終了・プログラム終了直前」



(設問 3b)

必要なキューへの書き出し・キューからの読み出しを行えている。

(設問 4)

まず、作成したプログラムはキューへの書き込みをキューからの読み出しに先行して行っている。そのためデータが空になる入力データがない場合であり、読み出し許可フラグで制御している。

そして、キューが一杯の時であるが、(設問 1)でも述べた通り、入力されたデータがキューのサイズよりも大きかった場合、書き込みを続けキューが一杯になった時点で、キューの読み出しを行い、またキューの書き込み・読み出しの続行を可能にしている。これを達成するために、a4 レジスタはレジスタで退避し、a5 レジスタは退避していない。

(設問 5)

感想：

本項（レポート 2 問 6）ではアセンブラ実習 6 で扱ったキューのプログラムを改良することでプログラム作成をした。改良前はサブルーチン化しなくてもジャンプ命令などで実装可能だったが、改良後は（a0 レジスタを書き込む/読み出すデータの始めアドレスレジスタと固定する場合）サブルーチン化しないと実装ができなかった。つまりシステムスタックへの退避を有効利用できたように思える。