

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 WRITE 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)

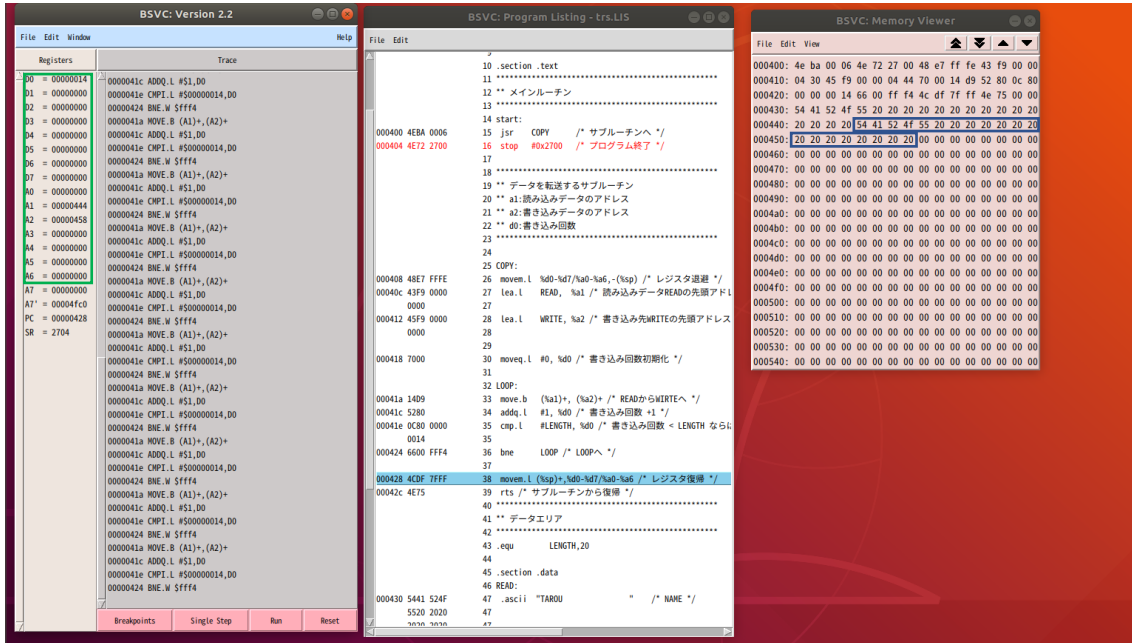
プログラムの実行結果をメインルーチン実行前、サブルーチンレジスタ復帰直前、メインルーチン実行後のメモリ・レジスタの様子を比較しながら示す。

「メインルーチン実行前」

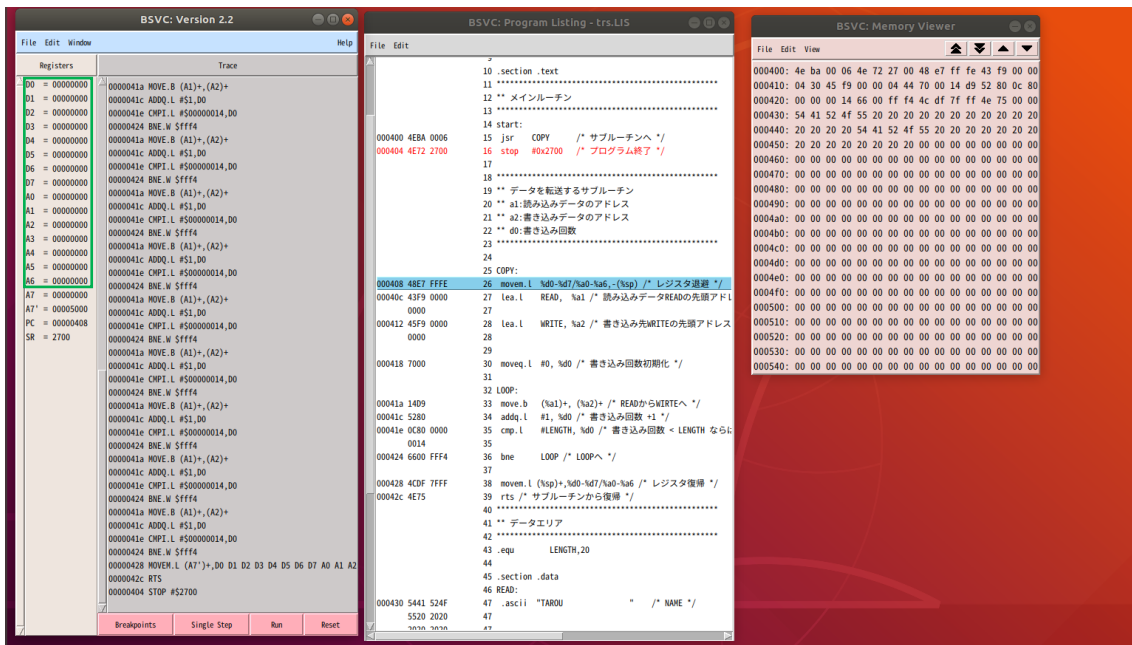
The screenshot displays the BSVC debugger interface with three main panels:

- BSVC: Version 2.2 (Registers):** Shows the state of registers D0 through SR. All registers are at their initial zero or default values (e.g., D0 = 00000000, PC = 00000400).
- BSVC: Program Listing - trs.LIS:** Displays the assembly code for the main routine. The program starts at address 000400 and ends at 00042C. Key instructions include:
 - 15 `jsr COPY` /* サブルーチンへ */
 - 16 `stop #0x2700` /* プログラム終了 */
 - 26 `movem.l %d0-%d7,%a0-%a6,~(%sp)` /* レジスタ退避 */
 - 27 `lea.l READ, %a1` /* 読み込みデータREADの先頭アドレス
 - 28 `lea.l WRITE, %a2` /* 書き込み先WRITEの先頭アドレス
 - 30 `moveq.l #0, %d0` /* 書き込み回数初期化 */
 - 32 `LOOP:`
 - 33 `move.b (%a1)+, (%a2)+` /* READからWRITEへ */
 - 34 `addq.l #1, %d0` /* 書き込み回数 +1 */
 - 35 `cmp.l #LENGTH, %d0` /* 書き込み回数 < LENGTH なら
 - 36 `bne LOOP` /* LOOPへ */
 - 38 `movem.l (%sp)+, %d0-%d7,%a0-%a6` /* レジスタ復帰
 - 39 `rts` /* サブルーチンから復帰 */
 - 41 `/* データエリア`
 - 42 `/*`
 - 43 `.equ LENGTH, 20`
- BSVC: Memory Viewer:** Shows the memory dump starting at address 000400. The first few lines of memory contain the program code, and subsequent lines are filled with zeros, indicating the data area.

「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。

(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。

The screenshot shows the BSV-C: Memory Viewer interface. The title bar reads "BSV-C: Memory Viewer". Below the title bar is a menu bar with "File", "Edit", and "View". To the right of the menu bar are three icons: a home icon, a double arrow icon, and a single arrow icon. The main area displays a list of memory addresses and their corresponding data. The address 0004ff is highlighted in green, and the data "44 77 ee" is highlighted in red.

| Address | Data |
|------------|--|
| 00041f: 01 | 4e 75 00 00 ee ee ee ee ee 44 77 ee ee 44 |
| 00044b: ee | ee ee ee ee 00 00 00 00 ee ee ee ee ee ee ee |
| 0004c4: 03 | 03 01 03 03 03 03 03 03 01 01 01 01 01 01 |
| 0004c8: 01 | 01 01 02 02 02 02 02 02 02 00 00 00 00 00 |
| 0004e6: 00 | 03 03 03 03 03 03 01 01 01 02 02 02 00 00 04 |
| 0004ff: ee | ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee |
| 00050f: ee | ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee |
| 00051a: ee | ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee |
| 00052f: ee | ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee |
| 00053f: ee | ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee |
| 00054f: ee | ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee |
| 00055f: ee | ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee |
| 00056f: ee | ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee |
| 00057f: ee | ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee |
| 00058f: ee | ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee |
| 00059f: ee | ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee |
| 0005af: ee | ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee |
| 0005bf: ee | ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee |
| 0005cf: ee | ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee |
| 0005df: ee | ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee |
| 0005ef: ee | ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee |

BSVC: Memory Viewer

File Edit View

0004ff: 01 4e 75 00 00 ee ee ee ee ee 44 77 ee ee 44

0004fb: ee ee ee ee ee 00 00 00 00 00 00 00 00 00 00

0004fc: 03 03 03 03 03 03 03 03 03 03 01 01 01 01 01

0004fd: 01 01 01 02 02 02 02 02 02 02 00 00 00 00 00

0004fe: 00 03 03 03 03 03 01 01 01 01 02 02 02 00 04

0004ff: ee ee ee ee ee ee ee ee ee ee ee ee ee ee

000500: ee ee ee ee ee ee ee ee ee ee ee ee ee ee

000501: ee ee ee ee ee ee ee ee ee ee ee ee ee ee

000502: ee ee ee ee ee ee ee ee ee ee ee ee ee ee

000503: ee ee ee ee ee ee ee ee ee ee ee ee ee ee

000504: ee ee ee ee ee ee ee ee ee ee ee ee ee ee

000505: ee ee ee ee ee ee ee ee ee ee ee ee ee ee

000506: ee ee ee ee ee ee ee ee ee ee ee ee ee ee

000507: ee ee ee ee ee ee ee ee ee ee ee ee ee ee

000508: ee ee ee ee ee ee ee ee ee ee ee ee ee ee

000509: ee ee ee ee ee ee ee ee ee ee ee ee ee ee

00050a: ee ee ee ee ee ee ee ee ee ee ee ee ee ee

00050b: ee ee ee ee ee ee ee ee ee ee ee ee ee ee

00050c: ee ee ee ee ee ee ee ee ee ee ee ee ee ee

00050d: ee ee ee ee ee ee ee ee ee ee ee ee ee ee

00050e: ee ee ee ee ee ee ee ee ee ee ee ee ee ee

[illegible]

BSV::C Memory Viewer

File Edit View

0004af: 01 4e 75 00 00 ee ee ee ee 44 77 ee ee 44

0004b0: 00 00 00 ee ee ee ee ee ee ee ee ee ee ee ee

0004b1: 01 01 01 02 02 02 02 02 01 01 01 01 01 01 01

0004b2: 00 00 00 ee ee ee ee ee ee ee ee ee ee ee ee

0004ff: ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee

00050f: ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee

00051f: ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee

00052f: ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee

00053f: ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee

00054f: ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee

00055f: ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee

00056f: ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee

00057f: ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee

00058f: ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee

00059f: ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee

0005af: ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee

0005bf: ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee

0005cf: ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee

0005df: ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee

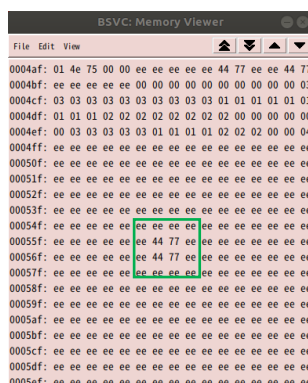
0005ef: ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee

[illegible][illegible]

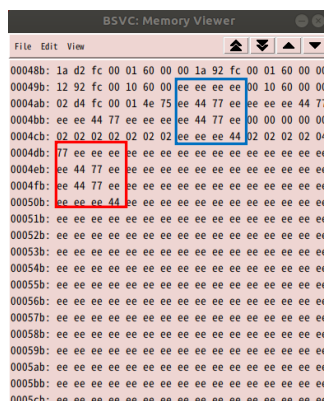
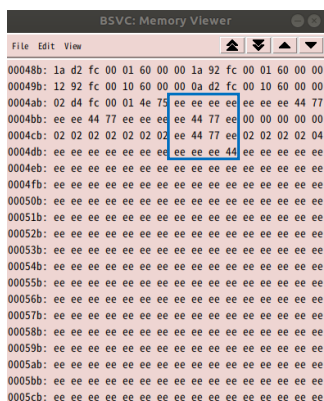
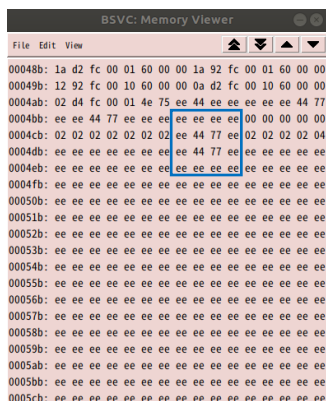
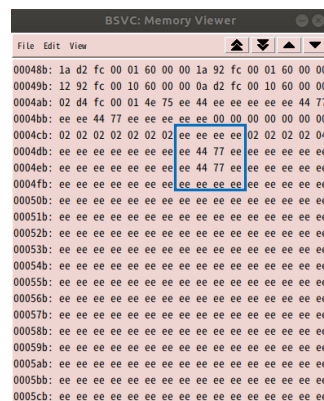
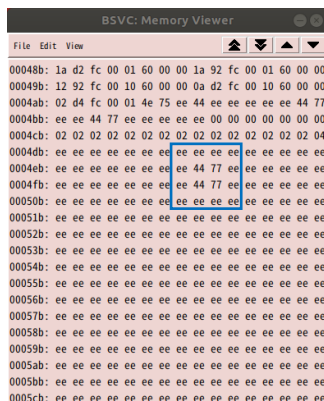
The screenshot shows the 'BSV-C: Memory Viewer' window. It has a menu bar with 'File', 'Edit', and 'View'. Below the menu bar are four navigation icons: a home icon, a double arrow pointing down, a double arrow pointing up, and a dropdown arrow. The main area displays a list of memory addresses and their data. The address 0005f7f is highlighted in green.

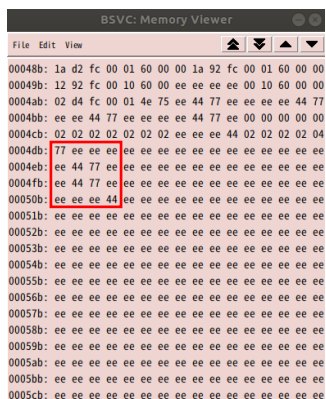
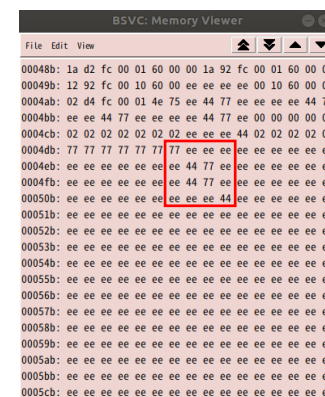
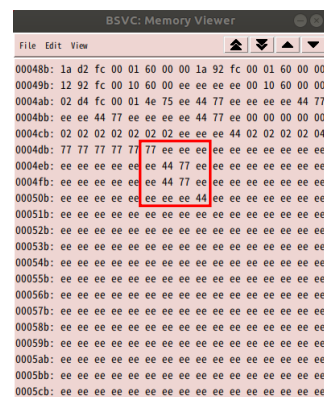
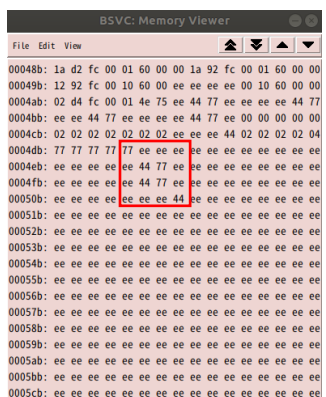
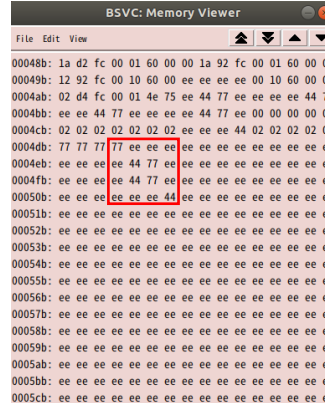
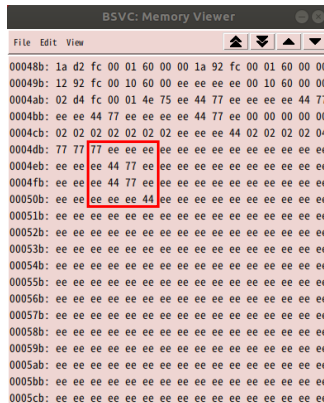
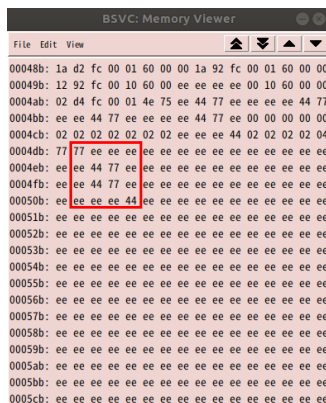
| Address | Data |
|--|------|
| 000404f: 01 4e 75 00 00 ee ee ee ee 44 77 ee ee 44 | |
| 000408f: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | |
| 00040cf: 03 03 03 03 03 03 03 03 01 01 01 01 01 01 | |
| 00040df: 01 01 01 02 02 02 02 02 02 02 02 02 02 02 | |
| 00041ff: 00 00 03 03 03 03 01 01 01 01 02 02 02 04 | |
| 00042ff: ee ee ee ee ee ee ee ee ee ee ee ee ee ee | |
| 000505f: ee ee ee ee ee ee ee ee ee ee ee ee ee ee | |
| 000501f: ee ee ee ee ee ee ee ee ee ee ee ee ee ee | |
| 000502f: ee ee ee ee ee ee ee ee ee ee ee ee ee ee | |
| 000503f: ee ee ee ee ee ee ee ee ee ee ee ee ee ee | |
| 000504f: ee ee ee ee ee ee ee ee ee ee ee ee ee ee | |
| 000505f: ee ee ee ee ee ee ee ee ee ee ee ee ee ee | |
| 000506f: ee ee ee ee ee ee ee ee ee ee ee ee ee ee | |
| 000507f: ee ee ee ee ee ee ee ee ee ee ee ee ee ee | |
| 000508f: ee ee ee ee ee ee ee ee ee ee ee ee ee ee | |
| 000509f: ee ee ee ee ee ee ee ee ee ee ee ee ee ee | |
| 00050af: ee ee ee ee ee ee ee ee ee ee ee ee ee ee | |
| 00050bf: ee ee ee ee ee ee ee ee ee ee ee ee ee ee | |
| 00050cf: ee ee ee ee ee ee ee ee ee ee ee ee ee ee | |
| 00050df: ee ee ee ee ee ee ee ee ee ee ee ee ee ee | |
| 00050ef: ee ee ee ee ee ee ee ee ee ee ee ee ee ee | |

[illegible][illegible]



(2)

[illegible]



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_Queue, %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_Queue: .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_Queue:」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。

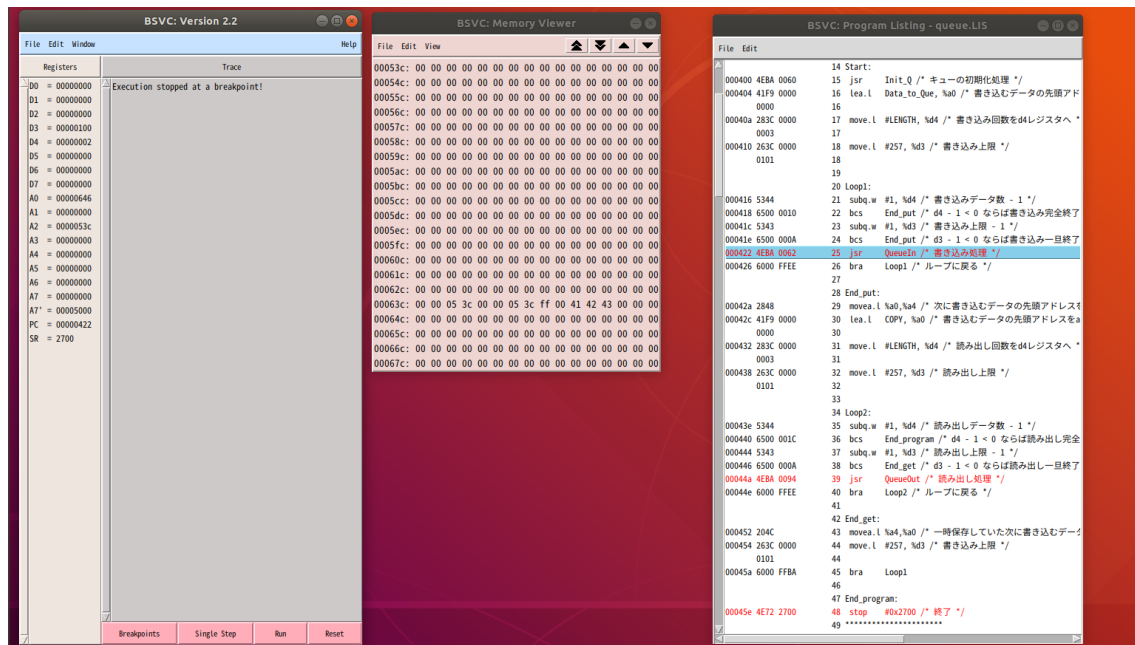
(設問 2)

求めるプログラムは上記のとおりである。

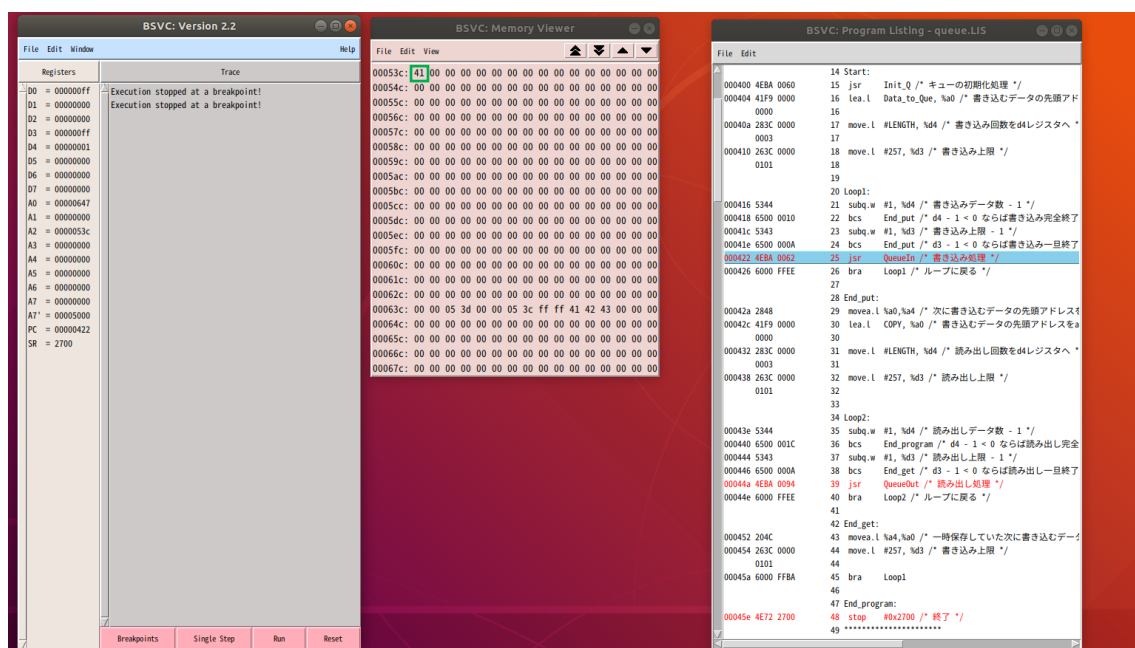
(設問 3a)

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」



The image shows the BSVC (Binary Static Viewer) interface. The left pane displays the 'Registers' window, showing the values of various registers (D0 through SR). The right pane displays the 'File Edit View' window, showing the assembly code for 'Program Listing - queue.LIS'. The code includes instructions like 'jsr Init_q', 'lea DataQueue, a0', 'move.l #LENGTH, %d4', and 'jsr QueueIn'. The code is organized into sections like 'Loop1' and 'Loop2'.

Registers Window (Left Pane):

| Register | Value |
|---------------|-------|
| D0 = 000000ff | |
| D1 = 00000000 | |
| D2 = 00000000 | |
| D3 = 00000000 | |
| D4 = 00000100 | |
| D5 = 00000002 | |
| D6 = 00000000 | |
| D7 = 00000000 | |
| A0 = 00000640 | |
| A1 = 00000000 | |
| A2 = 0000053c | |
| A3 = 00000000 | |
| A4 = 00000649 | |
| A5 = 00000000 | |
| A6 = 00000000 | |
| A7 = 00000000 | |
| A* = 00000540 | |
| PC = 00000444 | |
| SR = 2700 | |

File Edit View Window (Right Pane):

```

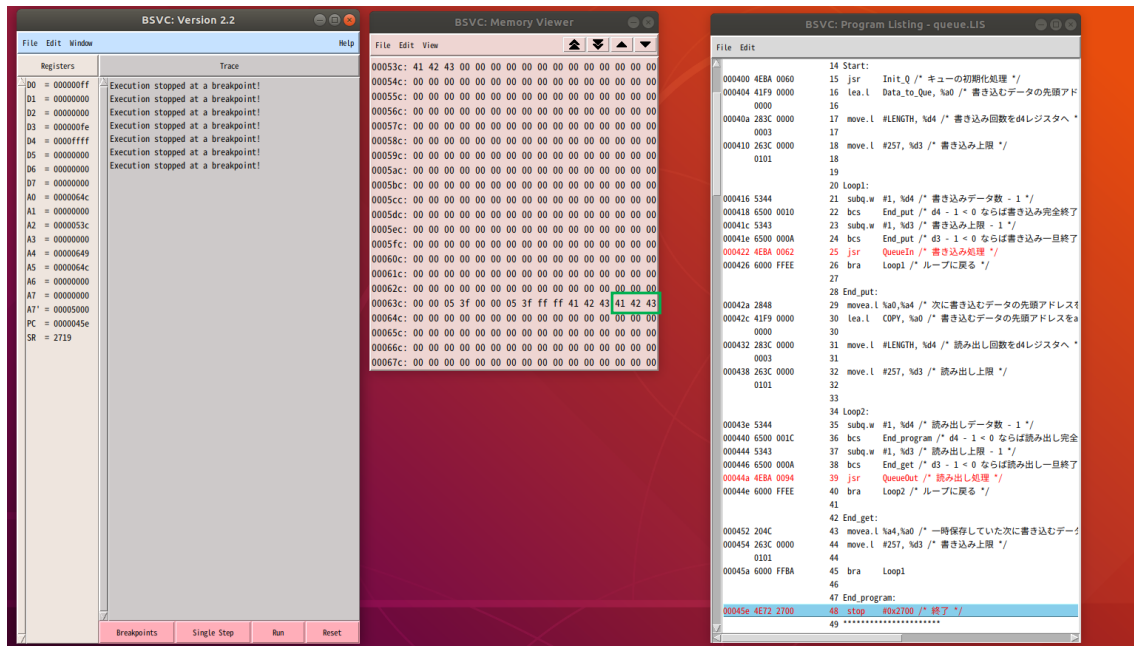
14 Start:
000400 4EBA 0060 15 jsr  Init_q  /* キューの初期化処理 */
000404 41F9 0000 16 lea.l  DataQueue, a0 /* 書き込むデータの先頭アドレス */
000000 0000 16
000404 263C 0000 17 move.l #LENGTH, %d4 /* 書き込み回数をd4レジスタへ */
000000 0000 17
000410 263C 0000 18 move.l #257, %d3 /* 書き込み上限 */
000000 0101 18
000000 0000 19
20 Loop1:
000416 5344 21 subq.w #1, %d4 /* 書き込みデータ数 - 1 */
000418 6500 0010 22 bcs  End_put /* d4 - 1 < 0 ならば書き込み完了終了 */
00041C 5343 23 subq.w #1, %d3 /* 書き込み上限 - 1 */
00041E 6500 000A 24 bcs  End_put /* d3 - 1 < 0 ならば書き込み一旦終了 */
000422 4EBA 0062 25 jsr  QueueIn /* 書き込み処理 */
000426 6000 FFEE 26 bra  Loop1 /* ループに戻る */
000000 0000 27
28 End_put:
00042A 2848 29 movea.l %a0, %a1 /* 次に書き込むデータの先頭アドレス */
00042C 41F9 0000 30 lea.l  COPR, %a0 /* 書き込むデータの先頭アドレスをa */
000000 0000 30
000432 263C 0000 31 move.l #LENGTH, %d4 /* 読み出し回数をd4レジスタへ */
000000 0000 31
000438 263C 0000 32 move.l #257, %d3 /* 読み出し上限 */
000000 0101 32
000000 0000 33
34 Loop2:
00043E 5344 35 subq.w #1, %d4 /* 読み出しデータ数 - 1 */
000440 6500 001C 36 bcs  End_program /* d4 - 1 < 0 ならば読み出し完全 */
000444 5343 37 subq.w #1, %d3 /* 読み出し上限 - 1 */
000446 6500 000A 38 bcs  End_get /* d3 - 1 < 0 ならば読み出し一旦終了 */
00044A 4EBA 0094 39 jsr  QueueOut /* 読み出し処理 */
00044E 6000 FFEE 40 bra  Loop2 /* ループに戻る */
000000 0000 41
42 End_get:
000452 204C 43 movea.l %a0, %a1 /* 一時保存していた次に書き込むデー */
000454 263C 0000 44 move.l #257, %d3 /* 書き込み上限 */
000000 0101 44
00045A 6000 FFBA 45 bra  Loop1
000000 0000 46
47 End_program:
00045E 4E72 2700 48 stop  #0x2700 /* 終了 */
000000 0000 49 .....

```

The image shows the BSVC (Binary Software Viewer) interface, which is used for analyzing binary data and program execution. It consists of two main windows:

- BVC: Memory Viewer:** This window displays a memory dump. On the left, there is a 'Registers' pane showing the values of various registers (D0 through SR). On the right, there is a 'Trace' pane showing the execution of instructions. The memory dump itself is a table with columns for address, data, and comments. The address column shows values from 00053c to 00067c. The data column shows hexadecimal values. The comments column shows the meaning of the data, such as 'Execution stopped at a breakpoint!'.
- BVC: Program Listing - queue.LIS:** This window displays the assembly code corresponding to the memory dump. It has a 'File Edit' menu and a list of instructions. The instructions are numbered and include comments. For example, instruction 14 is 'Init Q / キューの初期化処理', instruction 15 is 'jsr Init_Q, %a0 / Data_to_Que, %a0 / 書き込むデータの先頭アドレス', and instruction 25 is 'jsr QueueIn / 書き込み処理'.

「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。