case文 / switch文の補足

1. はじめに

教科書の 93 頁の case 文/switch 文では，ジャンプ・テーブル（jump table）や jr 命令の記述があるが，具体的な case 文/switch 文のアセンブリ・コードは載っていない．このため講義では，具体的なアセンブリ・コードを示した．本資料では，QtSpim で実行するためのアセンブリ・コードについて解説する．

2. case 文/switch 文の例題

リスト 1 に示した C コードをアセンブリ・コードに変換せよ．変数 f から k は，レジスタ$s0 から$s5 に割り付けること．ジャンプ・テーブルを使った解答を示せ．

リスト 1 case-switch.c

```
1  switch (k) {
2     case 0: f = i + j;
3             break;
4     case 1: f = g + h;
5             break;
6     case 2: f = g - h;
7             break;
8     case 3: f = i - j;
9             break;
10 };
```

3. 例題の解答

解答をリスト 2 に示す．C コードの case 0 から case 3 を，アセンブリ・コードではラベルの L0 から L3 に対応付ける．リスト 2 の 2–5 行目がジャンプ・テーブルである．ラベル JTBL はジャンプ・テーブルの開始アドレス（ベースアドレス）である．

1 行目の “.data” は，アセンブリ・コードではなくて，アセンブラ指示命令であり，データセグメントの開始を指示している．ピリオドから始まる命令はアセンブラ指示命令である．2 行目の “.word” により，ラベル L0 のアドレスの値がメモリに格納される．7 行目の “.text” は，テキストセグメントの開始を指示している．

10 行目の la 命令，11 行目の li 命令と 27 行目の nop 命令は，アセンブラ命令ではなくて，擬似命令と呼ばれる命令である．la 命令は load address を意味し，ラベル JTBL の番地がレジスタ$s6 に格納される．li 命令は load immediate を意味し，レジスタ$s5 に 2 が格納される．27 行目の nop は，no operation を意味し，何もしない命令である．

リスト 2 case-switch.s

```
1          .data
2  JTBL:   .word L0
3          .word L1
4          .word L2
5          .word L3
6
7          .text
8          .globl __start
9  __start:
10     la $s6,JTBL         # $s6にJTBLの番地を格納する
11     li $s5,2            # $s5に2を設定する.
12     slt $t0,$s5,$zero   # Test if k < 0
13     bne $t0,$zero,Exit
14     slti $t0,$s5,4      # Test if k < 4
15     beq $t0,$zero,Exit  # if k >=4 then goto Exit
16     sll $t0,$s5,2
```

```
17      add $t0,$t0,$s6
18      lw $t0,0($t0)
19      jr $t0
20  L0: add $s0,$s3,$s4        # f = i + j
21      beq $zero,$zero,Exit   # break ;
22  L1: add $s0,$s1,$s2        # f = g + h;
23      beq $zero,$zero,Exit   # break ;
24  L2: sub $s0,$s1,$s2        # f = g - h;
25      beq $zero,$zero,Exit   # break ;
26  L3: sub $s0,$s3,$s4        # f = i - j;
27  Exit:nop
```

4. テキストセグメントとデータセグメント

4.1 テキストセグメント

リスト 2 を QtSpim で読み込んだときのテキストセグメントを**図 1** に示す.

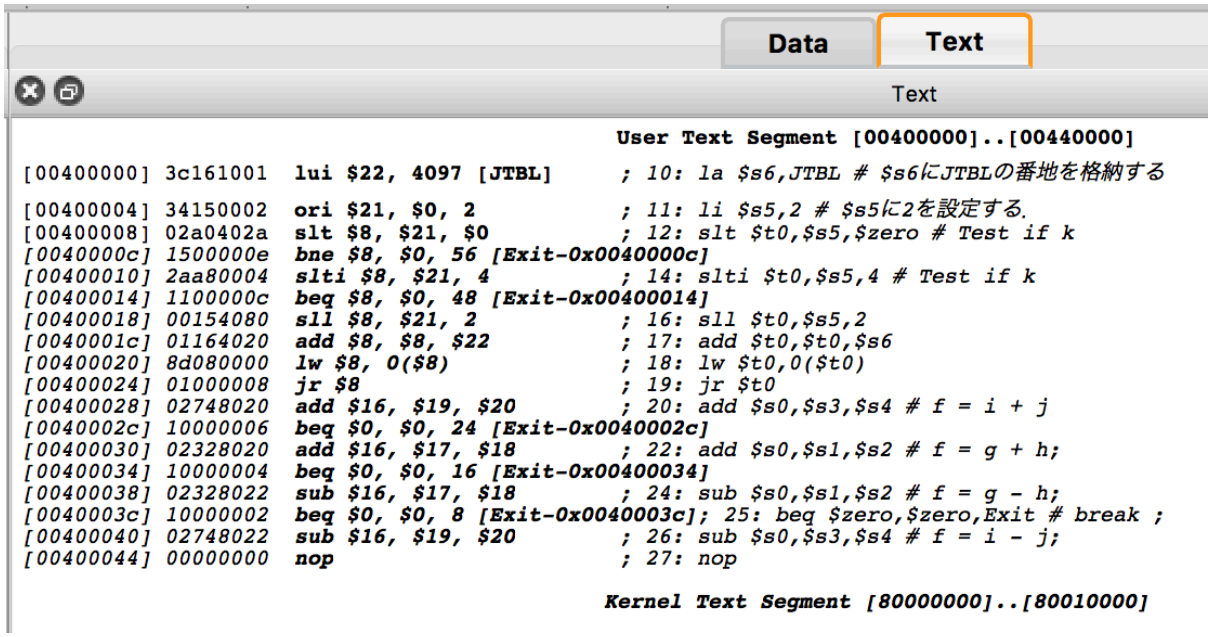

図 1 Text Segment

テキストセグメントから分かることは次の通り.

(1) テキストセグメントは 0x00400000 番地から開始されること. すなわち, ラベル__start の値は 0x00400000 である.

(2) 0x00400000 番地を見ると, 擬似命令 la は実際には lui 命令であること.

(3) 0x00400004 番地を見ると, 擬似命令 li は実際には ori 命令であること.

(4) 0x00400044 番地を見ると, 擬似命令 nop の機械語は $(00000000)_{16}$ であること.

(5) 0x00400028 番地が L0, 0x00400030 番地 が L1, 0x00400038 番地が L2, 0x00400040 番地が L3 に対応すること.

4.2 データセグメント

リスト 2 を QtSpim で読み込んだときのデータセグメントを**図 2** に示す．

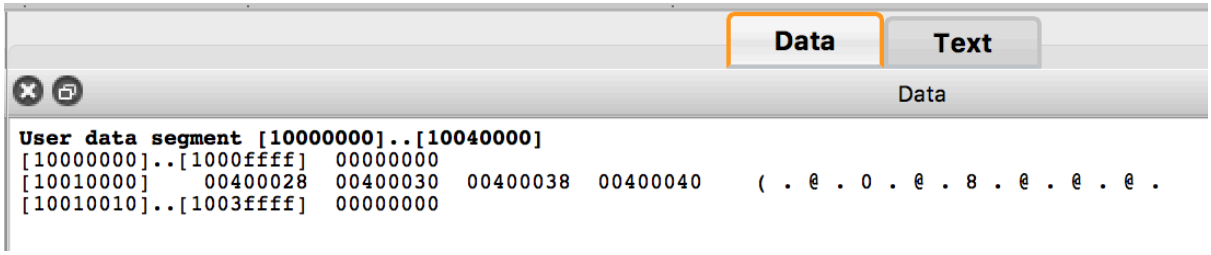

図 2 Data Segment

データセグメントから分かることは次の通り．

(1) データセグメントは 0x10000000 番地から開始されること．
(2) ジャンプ・テーブルの開始アドレスは，0x10010000 番地であること．
(3) ジャンプ・テーブルの内容が， 0x00400028， 0x00400030，0x00400038，0x00400040 であること．

5. 検討課題

5.1 if-else 文との比較

case 文/switch 文を if-else 文に書き直してから，アセンブリ・コードに変換する方法もある．if-else 文を使った場合と，case 文/switch 文を使った場合を比較せよ．

5.2 k の境界チェック

アセンブリ・コードでは，k が $0 \leqq k \leqq 3$ の範囲に収まっていることをチェックしている．このチェックを境界チェック (bounds checking) と呼ぶ．リスト 2 の 12–15 行目を見ると，k<0 と k<4 をチェックしている．k は整数なので，$k \leqq 3$ ではなくて，k<4 をチェックしてもよい．k と 3 を使って $k \leqq 3$ をチェックするようなアセンブリ・コードを示し，k<4 のチェックと $k \leqq 3$ をチェックのコードを比較せよ．

5.3 境界チェックの簡便法

リスト 2 を教科書に載っている境界チェックの簡便法を使うように修正せよ．簡便法では，$k \geqq 4$ または k<0 をチェックする．境界チェックについて，リスト 2 と簡便法を比較せよ．簡便法はどんな場合でも正しく動作するだろうか？

5.4 nop 命令

擬似命令 nop の機械語は $(00000000)_{16}$ である．この機械語をアセンブリ言語に変換せよ．

6. おわりに

QtSpim は，自分の書いたアセンブリプログラムの動作確認や，教科書や講義で示されたプログラムの動作が分からないとき使うと良い．アセンブラ指示命令と擬似命令については，QtSpim のヘルプに記載があるので，目を通しておくこと．