情報工学実験A（ハードウェア）春休み課題

09501502 池田海斗

April 11, 2021

1　C言語プログラミング

1.1　作成したプログラム

```
 1  #include <stdio.h>
 2
 3  int vsum(int *addr, int n)
 4  {
 5      int sum = 0;
 6      int i;
 7
 8      for (i = 0; i < n; i++)
 9      {
10          // ▼ My Answer ▼
11
12          sum += *addr++;
13
14          // ▲ My Answer ▲
15      }
16      return sum;
17  }
18
19  /**
20  * ============================================
21  *   For vsum test
22  * ============================================
23  */
24  int main(int n, char **argv)
25  {
26      int x[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
27      int n1 = 10;
28      int ans = vsum(x, n1);
29      printf("n = %d, ans = %d\n", n1, ans);
30  }
```

1.2 実行結果

```
n = 10, ans = 55
```

2 アセンブリ言語への変換

2.1 プログラム変換

2.1.1 作成したプログラム

```
 1  #include <stdio.h>
 2
 3  int vsum(int *addr, int n)
 4  {
 5      int sum = 0;
 6      int i;
 7
 8      // ▼ My Answer ▼
 9
10      i = 0;
11      while (i < n)
12      {
13          sum += *addr++;
14          i++;
15      }
16      // ▲ My Answer ▲
17
18      return sum;
19  }
20
21  /**
22  * ============================================
23  *   For vsum test
24  * ============================================
25  */
26  int main(int n, char **argv)
27  {
28      int x[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
29      int n1 = 10;
30      int ans = vsum(x, n1);
31      printf("n = %d, ans = %d\n", n1, ans);
32  }
```

2.1.2 実行結果

```
n = 10, ans = 55
```

2.2 レジスタの割当

2.2.1 作成したプログラム

```
 1  #include <stdio.h>
 2
 3  int vsum(int *addr, int n)
 4  {
 5      int sum = 0;
 6      int i;
 7
 8      // ▼ My Answer ▼
 9
10      i = 0;
11
12  _L1:
13      if (i == n) goto _L2;
14      sum += *addr++;
15      i++;
16      goto _L1;
17
18  _L2:
19      return sum;
20
21      // ▲ My Answer ▲
22  }
23
24  /**
25  * ================================================
26  *   For vsum test
27  * ================================================
28  */
29  int main(int n, char **argv)
30  {
31      int x[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
32      int n1 = 10;
33      int ans = vsum(x, n1);
34      printf("n = %d, ans = %d\n", n1, ans);
35  }
```

2.2.2 実行結果

```
n = 10, ans = 55
```

2.3 アドレス計算

2.3.1 作成したプログラム

```
 1      .data
 2      .align  2
 3  primes:
 4      .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
 5
 6
 7      .text
 8      .align      2
 9  main:
10      la      $a0, primes     # addr に配列の先頭アドレス
11      la      $a1, 10         # n = 10;
12
13      j     vsum
14
15  vsum:
16      li      $s0, 0          # sum = 0;
17      li      $t0, 0          # i = 0;
18
19  _L1:
20      slt     $t3, $t0, $a1   # $t3 = (i < n)
21      beqz    $t3, _L2        # if ($t3 == 0) goto _L2;
22
23      mul     $t1, $t0, 4     # $t1 = $t0 << 2; ($t1 = $t0 * 4)
24      addu    $t1, $a0, $t1   # $t1 はデータのアドレス
25      lw      $t2, 0($t1)     # データをロードし，$t2 に格納
26      add     $s0, $s0, $t2   # 加算
27      addi    $t0, $t0, 1     # i++;
28      j       _L1             # goto _L1;
29
30  _L2:
31      move    $v0, $s0        # $v0 = $s0
32      j       $ra             # return
```

3 コードサイズと実行ステップ数

命令数が 12 で 1 命令は 4 バイトであるため，アセンブリ言語プログラムのコードサイズは 48 である．またプログラムを実行する際の実行ステップ数は，固定的に実行される命令数が 6 であり，データ要素数 n に依存して繰り返し実行される命令数が 8 であるため，8n+6 となる．

4　メモリ間のデータコピー

4.1　C 言語でのプログラミング

4.1.1　作成したプログラム

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  void *mymemcpy(void *dest, void *src, int n)
 5  {
 6      int i = 0;
 7
 8  _L1:
 9      if (i == n) goto _L2;
10
11      *((char *)dest + i) = *((char *)src + i);
12
13      i++;
14      goto _L1;
15
16  _L2:
17      return dest;
18  }
19
20  /**
21  * ============================================
22  *   For mymemcpy test
23  * ============================================
24  */
25
26  /* メモリアドレス addr から n バイトのデータを表示 */
27  void memdump(void *addr, int n)
28  {
29      int i, j;
30      char *p = (char *)addr;
31
32      printf("%16s ", "");
33      for (j = 0; j < 8; j++)
34          printf(" +%x", j);
35      printf(" -");
36      for (j = 8; j < 16; j++)
37          printf(" +%x", j);
38      printf("\n");
39      for (i = 0; i < n; i += 16)
40      {
```

```
41          printf("%16p:", (void *)p);
42          for (j = 0; j < 8; j++)
43          {
44              printf(" %02x", *(p++));
45          }
46          printf(" -");
47          for (j = 0; j < 8; j++)
48          {
49              printf(" %02x", *(p++));
50          }
51          printf("\n");
52      }
53  } /* void memdump(...) */
54
55  int main(int argc, char *argv[])
56  {
57      /* メモリ領域を確保 */
58      void *m1 = (char *)malloc(sizeof(char) * 4096);
59      void *m2 = (char *)malloc(sizeof(char) * 4096);
60      void *result; /* mymemcpy の返値を格納する変数 */
61      int v = 0;
62      int i;
63
64      /* コピー元のメモリ領域の初期化 */
65      for (i = 0; i < 100; i++)
66      {
67          *((char *)m1 + i) = i;
68      }
69      /* コピー先のメモリ領域の初期化 */
70      for (i = 0; i < 100; i++)
71      {
72          *((char *)m2 + i) = 0;
73      }
74      /* コピー前のメモリ領域のデータを表示 */
75      printf("(before copy)\n");
76      printf("--- m1 ---\n");
77      memdump(m1, 128); /* メモリ領域 m1 の先頭から 128 バイトを表
示 */
78      printf("--- m2 ---\n");
79      memdump(m2, 128); /* メモリ領域 m2 の先頭から 128 バイトを表
示 */
80
81      /* アドレス m1 から始まるの 50 バイトのデータをアドレス m2 か
ら
82       * 始まる領域にコピー */
83      result = mymemcpy(m2, m1, 50);
```

```
84
85      /* コピー後のメモリ領域のデータを表示 */
86      printf("(after copy)\n");
87      printf("--- m1 ---\n");
88      memdump(m1, 128); /* メモリ領域 m1 の先頭から 128 バイトを表
示 */
89      printf("--- m2 ---\n");
90      memdump(m2, 128); /* メモリ領域 m2 の先頭から 128 バイトを表
示 */
91  } /* int main(...) */
```

4.1.2 実行結果

```
(before copy)
--- m1 ---
            +0 +1 +2 +3 +4 +5 +6 +7 - +8 +9 +a +b +c +d +e +f
  0x22fd010: 00 01 02 03 04 05 06 07 - 08 09 0a 0b 0c 0d 0e 0f
  0x22fd020: 10 11 12 13 14 15 16 17 - 18 19 1a 1b 1c 1d 1e 1f
  0x22fd030: 20 21 22 23 24 25 26 27 - 28 29 2a 2b 2c 2d 2e 2f
  0x22fd040: 30 31 32 33 34 35 36 37 - 38 39 3a 3b 3c 3d 3e 3f
  0x22fd050: 40 41 42 43 44 45 46 47 - 48 49 4a 4b 4c 4d 4e 4f
  0x22fd060: 50 51 52 53 54 55 56 57 - 58 59 5a 5b 5c 5d 5e 5f
  0x22fd070: 60 61 62 63 00 00 00 00 - 00 00 00 00 00 00 00 00
  0x22fd080: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
--- m2 ---
            +0 +1 +2 +3 +4 +5 +6 +7 - +8 +9 +a +b +c +d +e +f
  0x22fe020: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
  0x22fe030: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
  0x22fe040: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
  0x22fe050: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
  0x22fe060: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
  0x22fe070: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
  0x22fe080: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
  0x22fe090: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
(after copy)
--- m1 ---
            +0 +1 +2 +3 +4 +5 +6 +7 - +8 +9 +a +b +c +d +e +f
  0x22fd010: 00 01 02 03 04 05 06 07 - 08 09 0a 0b 0c 0d 0e 0f
  0x22fd020: 10 11 12 13 14 15 16 17 - 18 19 1a 1b 1c 1d 1e 1f
  0x22fd030: 20 21 22 23 24 25 26 27 - 28 29 2a 2b 2c 2d 2e 2f
  0x22fd040: 30 31 32 33 34 35 36 37 - 38 39 3a 3b 3c 3d 3e 3f
  0x22fd050: 40 41 42 43 44 45 46 47 - 48 49 4a 4b 4c 4d 4e 4f
  0x22fd060: 50 51 52 53 54 55 56 57 - 58 59 5a 5b 5c 5d 5e 5f
  0x22fd070: 60 61 62 63 00 00 00 00 - 00 00 00 00 00 00 00 00
  0x22fd080: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
--- m2 ---
```

```
          +0 +1 +2 +3 +4 +5 +6 +7 - +8 +9 +a +b +c +d +e +f
0x22fe020: 00 01 02 03 04 05 06 07 - 08 09 0a 0b 0c 0d 0e 0f
0x22fe030: 10 11 12 13 14 15 16 17 - 18 19 1a 1b 1c 1d 1e 1f
0x22fe040: 20 21 22 23 24 25 26 27 - 28 29 2a 2b 2c 2d 2e 2f
0x22fe050: 30 31 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
0x22fe060: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
0x22fe070: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
0x22fe080: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
0x22fe090: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
```

4.1.3 作成したプログラム

```
 1      .data
 2      .align  2
 3  dest:
 4      .space 40
 5  src:
 6      .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
 7
 8
 9      .text
10      .align      2
11  main:
12      la      $a0, dest       # dest の配列の先頭アドレス
13      la      $a1, src        # src の配列の先頭アドレス
14      la      $a2, 10         # n = 10;
15
16      j     mymemcpy
17
18  mymemcpy:
19      li      $t0, 0          # i = 0;
20
21  _L1:
22      slt     $t3, $t0, $a2   # $t3 = (i < n)
23      beqz    $t3, _L2        # if ($t3 == 0) goto _L2;
24
25      mul     $t1, $t0, 4     # $t1 = $t0 << 2; ($t1 = $t0 * 4)
26      addu    $t1, $a1, $t1   # $t1 は src のデータのアドレス
27      lw      $t2, 0($t1)     # データをロードし，$t2 に格納
28      mul     $t1, $t0, 4     # $t1 = $t0 << 2; ($t1 = $t0 * 4)
29      addu    $t1, $a0, $t1   # $t1 は dest のデータのアドレス
30      sw      $t2, 0($t1)     # データをストア
31      addi    $t0, $t0, 1     # i++;
32      j       _L1             # goto _L1;
33
34  _L2:
```

```
35      la      $v0, dest       # $v0 = $s0
36      j       $ra             # return
```

4.1.4 実行結果

```
User data segment [10000000]..[10040000]
[10000000]..[1000ffff]  00000000
[10010000]    00000001  00000002  00000003  00000004    . . . . .
[10010010]    00000005  00000006  00000007  00000008    . . . . .
[10010020]    00000009  0000000a  00000001  00000002    . . . . .
[10010030]    00000003  00000004  00000005  00000006    . . . . .
[10010040]    00000007  00000008  00000009  0000000a    . . . . .
[10010050]..[1003ffff]  00000000


PC       = 400020
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 24

R0  [r0] = 0
R1  [at] = 4
R2  [v0] = a
R3  [v1] = 0
R4  [a0] = 10010000
R5  [a1] = 10010028
R6  [a2] = a
R7  [a3] = 0
R8  [t0] = a
R9  [t1] = 10010024
R10 [t2] = a
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
```

```
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
R28 [gp] = 10008000
R29 [sp] = 7ffffdf4
R30 [s8] = 0
R31 [ra] = 400018
```

上記の実行結果より，きちんとメモリ間でのデータコピーが行われていることがわかる．また，実行後のメモリアドレスより，きちんと 10 回ループが行われたことがわかる．