課題のまとめ方

1. はじめに

計算機構成では，学期末試験の他にアセンブリプログラミングのレポート課題を課している．この資料では，アセンプリプログラムについて，どような内容を書けばよいかについて解説する．私がレポートで重視することは，次の 2 つである．
(1) 第三者がレポートを読んで，同じプログラムを作成できるか？
(2) プログラムが正しく動作することを示しているか？
本資料では，これら 2 つをどのようにまとめるかを解説する．

2. 同じプログラムが作成できるか

同じプログラムを作成できるようにするために，レポートにはアルゴリズムとデータ構造の解説を含める．科目 “プログラミング言語演習” の教科書，科目 “データ構造とアルゴリズム” の教科書が参考になるだろう．課題のアルゴリズムは複雑ではないので，箇条書きやフローチャートなどを使って解説すればよい．

課題で扱うデータ構造は，配列といった基本的なデータ構造だけである．詳しい解説は不要だが，アルゴリズムとテストデータを理解するために必要な解説は含めること．

計算機構成の教科書を参考に，作成したアセンブリプログラムの解説を書く．プログラムには適宜コメントを加える．レジスタの割当や役割も含めること．

3. プログラムが正しく動作することの確認

3.1 例：階乗を求める関数

解説の例として，教科書に載っている階乗を求める関数を取り上げる．リスト 1 は配布資料 [1] に載っている．テストデータは，アルゴリズムとアセンブリプログラムから決まる関数の定義域（入力変数の範囲）と値域（関数値の範囲）を考えて作成する．

リスト 1 fact.s

```
1      .data
2  text1:.asciiz "\nx="
3  text2:.asciiz "fact("
4  text3:.asciiz ")="
5
6      .text
7      .globl __start
8  __start:
9      ori $v0,$zero,4      # print_string
10     la $a0,text1
11     syscall
12     ori $v0,$zero,5      # read_int
13     syscall
14     add $a0,$zero,$v0
15     add $s0,$zero,$v0
16     jal fact
17     add $s1,$zero,$v0
18     ori $v0,$zero,4      # print_string
19     la $a0,text2
20     syscall
21     ori $v0,$zero,1      # print_int
22     add $a0,$zero,$s0
23     syscall
24     ori $v0,$zero,4      # print_string
25     la $a0,text3
26     syscall
27     ori $v0,$zero,1      # print_int
28     add $a0,$zero,$s1
29     syscall
30     j __start
31
32 fact:sub $sp,$sp,8
33      sw $ra,4($sp)
34      sw $a0,0($sp)
35      slti $t0,$a0,1 # 符号付きの比較
36      beq $t0,$zero,fact1
37      addi $v0,$zero,1 # 符号付きの加算
38      addi $sp,$sp,8
39      jr $ra
40 fact1:sub $a0,$a0,1
41      jal fact
42      lw $a0,0($sp)
43      lw $ra,4($sp)
44      addi $sp,$sp,8
45      mul $v0,$a0,  $v0# 32 ビットの符号付き乗算
46      jr $ra
```

fact 関数の引数と戻り値は 32 ビットの符号付き整数である．階乗は指数関数よりも早く増加するため，正しく計算できる引数の範囲は 32 ビットの符号付き整数の正数の範囲（$0 \sim 2^{31}-1$）よりもずっと狭い．すなわち，12 の階乗は $12! = 479001600 = (1C8CFC00)_{16}$ は 32 ビットの符号付き整数の範囲であるが，$13! = 6227020800 = (17328CC00)_{16}$ は範囲外となる．これらを踏まえて，テストデータは**表 1** のようになる．テストデータ 4 は，12 までは正しい計算結果が得られ，（アルゴリズムとアセンブリプログラムより）13 以上は正しく計算できないことを確認している．13 のときの戻り値 ($v0) は，$(17328CC00)_{16}$ の下から 32 ビットの $(7328CC00)_{16} = 1932053504$ である．

fact 関数の引数と戻り値は 32 ビットの符号なし整数とすれば 13 以上の階乗も正しく計算できるか検討してみよ．

表 1 fact のテストデータ

番号	入力	出力		注釈
	$a0	正しい計算結果	$v0	
1	0	1	同左	定義 0! = 1
2	1	1	同左	40 行目を実行する場合
3	12	479001600	同左	符号付き 32 ビットで収まる最大値
4	13	6227020800	1932053504	符号付き 32 ビットで収まらない．$(17328CC00)_{16}$

3.2 例：2 倍長加算

配布資料 [2] では，オーバーフローの検出をして 2 倍長（64 ビット）加算のアセンブリプログラム [*1] を示した．リスト 2 は，配布資料で示した dadd 関数を少し改良したプログラムである．li 命令は擬似命令であり，レジスタに 32 ビットの即値を格納する．この dadd 関数のテストデータを考えてみよう．

dadd 関数のアルゴリズムは，下位 32 ビットの加算と上位 32 ビットの加算を計算し，下位 32 ビットの加算でオーバーフローが発生した場合は上位 32 ビットの加算結果に 1 を加える．リスト 2 では，ちょっと工夫したプログラムになっており，sltu でオーバーフローが発生したとき設定される$t0 を上位 32 ビットの加算結果に加えている．

32 ビットの加算は addu 命令で正しく計算されるので，テストデータはオーバーフローが発生する場合と発生しない場合を考えれば良い．テストデータの一部を**表 2** に示す．テストデータ 3 は，上位は 0 で下位の加算でオーバーフローが発生する場合である．テストデータ 4 から 5 は，下位の加算でオーバーフローが上位の加算結果に影響を与える場合である．オーバーフロー（言い換えれば，桁上がり）に関するテストデータなので，$(80000000)_{16}$，$(FFFFFFFF)_{16}$，$(00000001)_{16}$ といった数値を使う．

表 2 dadd のテストデータ

番号	入力		出力		注釈
	$a1,$a0	$a3,$a2	正しい計算結果	$v1,$v0	
1	00000000 00000001	00000000 00000001	00000000 00000010	同左	下位 32 ビットの加算
2	00000001 00000000	00000001 00000000	00000010 00000000	同左	上位 32 ビットの加算
3	00000000 80000000	00000000 80000000	00000001 00000000	同左	下位 32 ビットの加算と OV
4	00000001 80000000	00000000 80000000	00000010 00000000	同左	下位から上位への桁上がり 1
5	00000000 80000000	00000001 80000000	00000010 00000000	同左	下位から上位への桁上がり 2
6	00000001 80000000	00000001 80000000	00000011 00000000	同左	下位から上位への桁上がり 3

リスト 2 dadd.s

```
1     .text
2     .globl __start
3  __start:
4     li $a0,0x80000000
5     li $a1,0x00000001
6     li $a2,0x80000000
7     li $a3,0xFFFFFFFF
8     jal dadd
9     nop
10    break 2
11
12 dadd:addu $v0,$a0,$a2
13    addu $v1,$a1,$a3
14    nor $t0,$a0,$zero
15    sltu $t0,$t0,$a2
16    addu $v1,$v1,$t0
17    jr $ra
```

[*1] 実際は，リスト 1 のようにキーボードから数値を入力するようなプログラムにしなければならない．

4. 追加課題

配布した課題 [3] とは別に，追加課題を示す．必須ではないが，レポートとして提出すれば評価する．

リスト 3 は 4 倍長加算を計算するプログラムである．実行結果をリスト 4 に示す．qadd 関数と add 関数のアセンブリコードを示せ．レポートには，アルゴリズムやテストデータを含めること．提出するアセンブリコードのファイル名は “xxxx-y-1.s” とせよ．

リスト 3 qadd.c

```
#include <stdio.h>
#define U64BPrint(s) printf("%08x%08x%08x%08x\n",s[3],s[2],s[1],s[0]);

unsigned int add(unsigned int *c, unsigned int a, unsigned int b, unsigned int ov)
{
  *c = a + b + ov;
  if( ~b < ov) return 1;
  b = b + ov ;
  if( ~a < b ) return 1;
  return 0;
}

unsigned int qadd(unsigned int *c, unsigned int *a, unsigned int *b )
{
  unsigned int ov=0;
  for(int i=0; i<4; i++)
    ov = add(&c[i],a[i],b[i],ov);
  return ov;
}

int main() {
  unsigned int a[]={-1,-1,-1,0x7fffffff},b[]={256,1,4096,0},c[]={0,0,0,0};
  unsigned int cin=0;
  printf("sizeof␣%lu␣bytes\n", sizeof(unsigned int));
  if(qadd(c,a,b)) printf("Overflow\n");
  U64BPrint(a);
  U64BPrint(b);
  U64BPrint(c);
}
```

リスト 4 実行結果

```
$ cc qadd-main.c
$ ./a.out
sizeof 4 bytes
7fffffffffffffffffffffffffffffff
00000000000010000000000100000100
80000000000010000000000001000000ff
$
```

参考文献

[1] 冨澤眞樹：QtSpim によるアセンブリプログラミング演習 ❶ (2017–05–23).
[2] 冨澤眞樹：第 13 回 計算機構成 配布資料 (2017–07–11).
[3] 冨澤眞樹：平成 29 年度 計算機構成 期末課題 (2017–07–25).

