計算機システム演習　第４回レポート

17B13541　　細木隆豊

1　課題１の議論

caller-save では、サブルーチンを 1 回呼び出すのでレジスタの退避・復帰を 1 回行う。

callee-save では、今回のサブルーチン read1bit で$s0, $s1 は使われないため、read1bit でレジスタを退避・復帰しなくて良い。main ルーチンでは$s0, $s1 を使うので、退避・復帰をする。よって、callee-save での退避・復帰は 1 回である。

以上から、今回の２進１０進変換プログラムではレジスタの退避・復帰回数はどちらも 1 回であるから、コードの効率の違いは生まれないと考えられる。ただ read1bit で$t0, $t1 を使わないために退避・復帰する必要がなく、callee-save の方が有効だと考えられる。

2　説明・工夫

プログラム 1: assignment1.s

```
    .text
main:
    addi   $sp, $sp, -12      # レジスタの退避領域作成
    sw     $ra, 0($sp)        # $ra を退避
    sw     $s0, 4($sp)        # $s0 を退避
    sw     $s1, 8($sp)        # $s1 を退避

    li     $s0, 0             # $s0 に初期値を設定
                              # $s0 は結果の記録
    li     $s1, 0             # $s1 に初期値を設定
                              # $s1 はループ回数の記録
loop:
    jal    read1bit           # サブルーチンの呼び出し
```

```
    add   $s0, $s0, $s0      # 今までの結果を２倍
    add   $s0, $s0, $v0      # 入力値を足す
    addi  $s1, $s1, 1        # ループ回数に 1 足す
    blt   $s1, 8, loop       # ループ回数を比較

    move  $a0, $s0
    li    $v0, 1
    syscall                  # 結果を表示

    lw    $ra, 0($sp)        # $ra を復帰
    lw    $s0, 4($sp)        # $s0 を復帰
    lw    $s1, 8($sp)        # $s1 を復帰
    addi  $sp, $sp, 12       # 退避領域開放
    jr    $ra                # main ルーチン終了

read1bit:
    li    $v0, 5
    syscall                  # 入力値を読み込み
    jr    $ra                # サブルーチン終了
```

プログラム 2: assignment2.s

```
    .data                    # data の設定
cnt:                         # 配列の要素数を入力させる
    .asciiz "What is number of the array?-"
array:                       # 配列の要素を入力させる
    .asciiz "type elements of the array.\n"
sum:                         # 配列の要素の和を出力
    .asciiz " is the sum."

    .text
main:
          ~ 退避の部分により省略 ~

    li    $v0, 4
    la    $a0, cnt
    syscall                  # 文字列 cnt を表示

    li    $v0, 5
    syscall
```

```
    move   $s0, $v0            # 配列の要素数を読み込み

    li     $v0, 4
    la     $a0, array
    syscall                    # 文字列 array を表示

    move   $a0, $s0            # サブルーチンの引数を設定
    jal    create_array        # $a0 は配列の要素数

    move   $a0, $v0            # create_array の戻り値を
                               # calc_sum の引数に設定
                               # $a0 は配列の先頭アドレス
    move   $a1, $s0            # 引数を設定
    jal    calc_sum            # $a1 は配列の要素数
    move   $s0, $v0            # calc_sum の結果を代入

    li     $v0, 1
    move   $a0, $s0
    syscall                    # 配列の和を表示

    li     $v0, 4
    la     $a0, sum
    syscall                    # 文字列 sum を表示

           ~ 復帰の部分により省略 ~

create_array:
           ~ 退避の部分により省略 ~
    move   $s0, $a0            # $s0 は残りのループ回数を記録

    li     $v0, 9
    add    $a0, $a0, $a0
    add    $a0, $a0, $a0       # 要素数 * 4 Byte
    syscall                    # メモリの確保
    move   $s1, $v0            # メモリの先頭アドレス

    sw     $v0, 8($sp)         # 先頭アドレスを記憶
loopa:
    li     $v0, 5
```

```
    syscall                 # 入力値を読み込み
    sw     $v0, 0($s1)      # 入力値を配列に追加

    addi   $s1, $s1, 4      # 次の要素が入るメモリ
    addi   $s0, $s0, -1     # 残りのループ回数を減らす
    blt    $zero, $s0, loopa # ループするか判定

           ~ 復帰の部分により省略 ~
    jr     $ra

calc_sum:
           ~ 退避の部分により省略 ~

    li     $s0, 0           # $s0 は配列の要素を読み込む
    li     $s1, 0           # $s1 はそれまでの和
loopb:
    lw     $s0, 0($a0)      # $s0 に要素を呼び出す
    add    $s1, $s1, $s0    # $s0 に $s1 を足す
    addi   $a0, $a0, 4      # 次の要素があるアドレス
    addi   $a1, $a1, -1     # 残りのループ回数
    blt    $zero, $a1, loopb # ループするか判定

    move   $v0, $s1         # 戻り値に結果を代入
           ~ 復帰の部分により省略 ~
    jr     $ra
```

課題 2 は callee-save で実装しました。

3 実行結果

assignment1.s

```
1
1
0
0
1
1
0
0
```

```
204
```

```
1
0
0
0
0
0
0
0
128
```

assignment2.s

```
What is number of the array?-5
type elements of the array.
1
2
3
4
5
15 is the sum.
```

```
What is number of the array?-10
type elements of the array.
1
2
3
4
5
6
7
8
9
10
55 is the sum.
```

4 感想・質問

難しかったが、理解できたので大変良かった。