演習１：

下図の Func 関数および Sum 関数のCコードを変換した右図のアセンブリコードの空欄を埋めよ。

```
int Sum(int n) {
  int s = 0;
  int i;
  for (i=-n; i<=n; i++) {
    s += Func(i);
  }
  return s;
}

int Func(int n) {
  if (n>=0) {
    return n;
  }
  else {
    return 0;
  }
}
```

```
Sum:    addi    $sp,    $sp,    -16
        sw      $ra,    12($sp)
        sw      $s1,    8($sp)
        sw      $s2,    4($sp)
        sw      $s3,    0($sp)
        add     $s1,    $a0,    $zero
        add     $s2,    $zero,  $zero
        sub     $s3,    $zero,  $s1
Loop:   slt     $t0,    $s1,    $s3
        bne     $t0,    $zero,  Exit
        add     $a0,    $s3,    $zero
        (a)     Func
        add     $s2,    $s2,    (b)
        addi    $s3,    $s3,    1
        j       Loop
Exit:   add     $v0,    $s2,    $zero
        lw      (c)     0($sp)
        lw      (d)     4($sp)
        lw      (e)     8($sp)
        lw      (f)     12($sp)
        addi    $sp,    $sp,    (g)
        (h)     (i)
```

```
Func:   slt     $t0,    $a0,    $zero
        (j)     $t0,    $zero,  Else
        add     $v0,    $zero,  $a0
        jr      $ra
Else:   add     $v0,    $zero,  $zero
        jr      $ra
```

演習１解答：

どの部分が何をやっているのかを理解した上で詳細を検討する。

```
int Sum(int n) {
  int s = 0;
  int i;
  for (i=-n; i<=n; i++) {
    s += Func(i);
  }
  return s;
}

int Func(int n) {
  if (n>=0) {
    return n;
  }
  else {
    return 0;
  }
}
```

```
Sum:    addi    $sp,    $sp,    -16
        sw      $ra,    12($sp)
        sw      $s1,    8($sp)
        sw      $s2,    4($sp)
        sw      $s3,    0($sp)      レジスタ退避
        add     $s1,    $a0,    $zero   n = 引数
        add     $s2,    $zero,  $zero   s = 0
        sub     $s3,    $zero,  $s1     I = -n
Loop:   slt     $t0,    $s1,    $s3     n<iなら
        bne     $t0,    $zero,  Exit    ループ脱出
        add     $a0,    $s3,    $zero   Func(i)
        jal     Func
        add     $s2,    $s2,    $v0     s+=戻り値
        addi    $s3,    $s3,    1       i++
        j       Loop
Exit:   add     $v0,    $s2,    $zero   戻り値 = s
        lw      $s3     0($sp)      レジスタ復帰
        lw      $s2     4($sp)
        lw      $s1     8($sp)
        lw      $ra     12($sp)
        addi    $sp,    $sp,    16
        jr      $ra
```

```
Func:   slt     $t0,    $a0,    $zero   n<0なら
        bne     $t0,    $zero,  Else    Else
        add     $v0,    $zero,  $a0     戻り値 = n
        jr      $ra
Else:   add     $v0,    $zero,  $zero   戻り値 = 0
        jr      $ra
```

演習２：

オブジェクトファイル main.o および proc1.o がそれぞれ右図のように与えられているとき、これらをリンクした下図の実行ファイルの空欄にあてはまる数値を16進数で答えよ。但し、下図のアドレス(c)の jal 命令は main.o のjal 命令に対応し、下図のアドレス(e)および(g)の lw 命令およびsw 命令はそれぞれ、proc1.o のlw 命令および sw命令に対応するものとする。また、$gp の値は 10008000_{16} とせよ。

main.o

ヘッダ	名前	main	
	テキスト・サイズ	500_{16}	
	データ・サイズ	50_{16}	
テキスト・セグメント	アドレス		
	0_{16}	lw $t0,0($gp)	
	…	…	
	30_{16}	jal 0	
	…	…	
データ・セグメント	0_{16}	(X)	
	…	…	
リロケーション情報	アドレス	命令タイプ	依存関係
	0_{16}	lw	X
	30_{16}	jal	proc1
シンボル表	ラベル	アドレス	
	X	-	
	proc1	-	

proc1.o

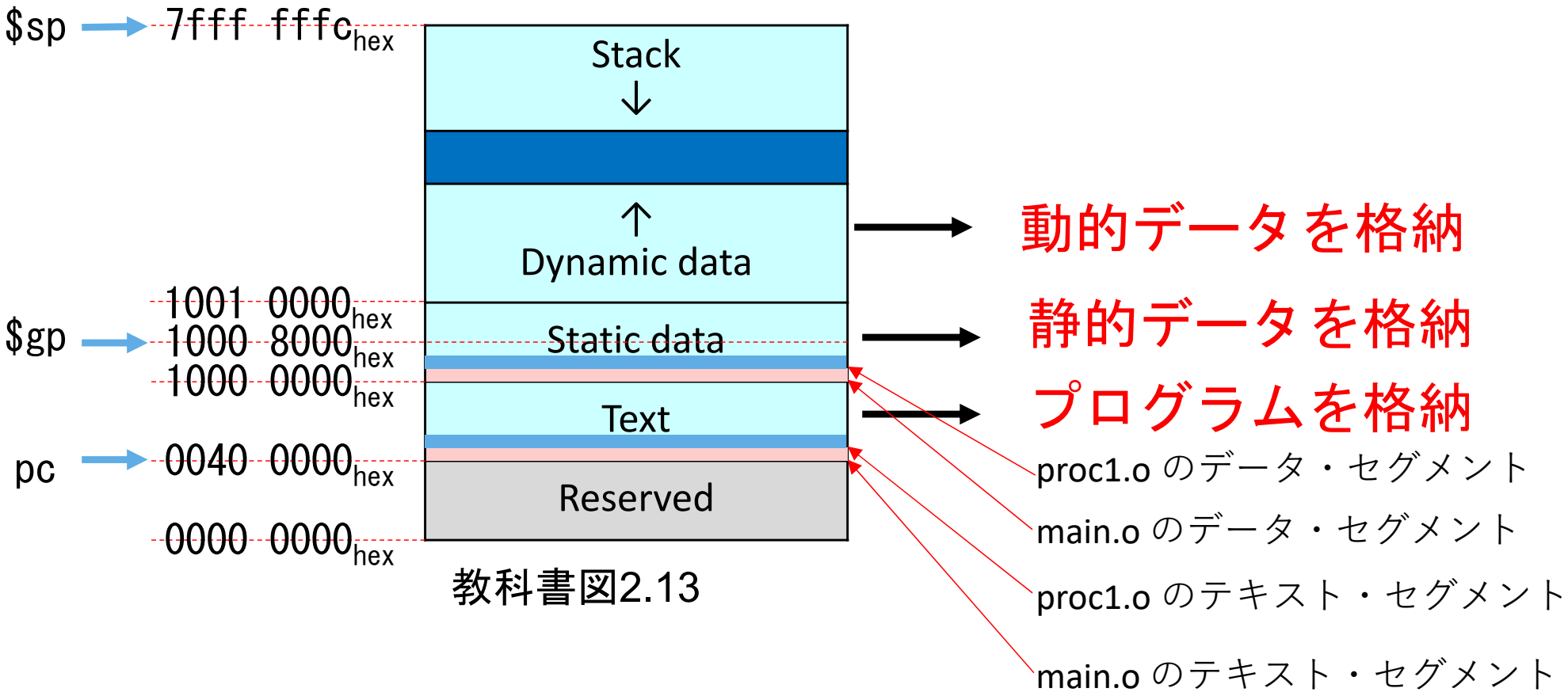
ヘッダ	名前	proc1	
	テキスト・サイズ	300_{16}	
	データ・サイズ	30_{16}	
テキスト・セグメント	アドレス		
	0_{16}	lw $t0,0($gp)	
	…	…	
	70_{16}	sw $t1,0($gp)	
	…	…	
データ・セグメント	0_{16}	(Y)	
	4_{16}	(Z)	
	…	…	
リロケーション情報	アドレス	命令タイプ	依存関係
	0_{16}	lw	Y
	70_{16}	sw	Z
シンボル表	ラベル	アドレス	
	Y	-	
	Z	-	
	…	-	

リンク結果

ヘッダ		
	テキスト・サイズ	(a)
	データ・サイズ	(b)
テキスト・セグメント	アドレス	
	00400000_{16}	lw $t0,$8000_{16}$($gp)
	…	…
	(c)	jal (d)
	…	…
	(e)	lw $t0, (f) ($gp)
	…	…
	(g)	sw $t1, (h) ($gp)
	…	…
データ・セグメント	10000000_{16}	(X)
	…	…
	(i)	(Y)
	(j)	(Z)
	…	…

【再掲】

メモリ上でのプログラムとデータの配置（MIPSの場合）



教科書図2.13

リンカは、与えられたオブジェクトファイルに書かれているプログラム（テキスト・セグメント）や静的データ（データセグメント）のメモリ上での配置（アドレス）を決定し、それが書かれた「実行可能なオブジェクトファイル」を生成する。

演習２解答：

リンク結果

ヘッダ		
	テキスト・サイズ	(a)
	データ・サイズ	(b)
テキスト・セグメント	アドレス	
	00400000_{16}	lw $t0, 8000_{16}($gp)
	…	…
	(c)	jal (d)
	…	…
	(e)	lw $t0, (f) ($gp)
	…	…
	(g)	sw $t1, (h) ($gp)
	…	…
データ・セグメント	10000000_{16}	(X)
	…	…
	(i)	(Y)
	(j)	(Z)
	…	…

※ Lecture 7の演習①も参考にして下さい。

演習３：

Lecture 8 で扱った「最終バージョンの乗算アルゴリズム」に沿って、4ビットの被乗数 7_{10} と乗数5_{10} の積を求める過程を書け。

サイクル	ステップ	被乗数レジスタ	積レジスタ
0	初期値	0111	
1	積レジスタのLSBが1なら被乗数を積レジスタの左半分に足す	0111	
	積レジスタを右シフト	0111	
2	積レジスタのLSBが1なら被乗数を積レジスタの左半分に足す	0111	
	積レジスタを右シフト	0111	
3	積レジスタのLSBが1なら被乗数を積レジスタの左半分に足す	0111	
	積レジスタを右シフト	0111	
4	積レジスタのLSBが1なら被乗数を積レジスタの左半分に足す	0111	
	積レジスタを右シフト	0111	

演習３解答：

Lecture 8 で扱った「最終バージョンの乗算アルゴリズム」に沿って、4ビットの被乗数 7_{10} と乗数5_{10} の積を求める過程を書け。
⇒ 答えが $35_{10} = 100011_2$ になっていることを確認しよう

サイクル	ステップ	被乗数レジスタ	積レジスタ
0	初期値	0111	00000101
1	積レジスタのLSBが1なら被乗数を積レジスタの左半分に足す	0111	01110101
	積レジスタを右シフト	0111	00111010
2	積レジスタのLSBが1なら被乗数を積レジスタの左半分に足す	0111	00111010
	積レジスタを右シフト	0111	00011101
3	積レジスタのLSBが1なら被乗数を積レジスタの左半分に足す	0111	10001101
	積レジスタを右シフト	0111	01000110
4	積レジスタのLSBが1なら被乗数を積レジスタの左半分に足す	0111	01000110
	積レジスタを右シフト	0111	00100011

※ Lecture 8 Part 2の補足説明も参考にして下さい。

演習４：

(1) 10進数の 1.0 および 2.0 を IEEE-754規格の単精度浮動小数点形式で表現した時の32ビットのビット列をそれぞれ8桁の16進数で表せ。

(2) 10進数の 1.0 および 2.0 を IEEE-754規格の倍精度浮動小数点形式で表現した時の64ビットのビット列をそれぞれ16桁の16進数で表せ。

(3) 10進数の 256.5 を IEEE-754規格の単精度浮動小数点形式で表現した時の32ビットのビット列を8桁の16進数で表せ。

(4) 10進数の –65536.125 を IEEE-754規格の倍精度浮動小数点形式で表現した時の64ビットのビット列を16桁の16進数で表せ。

演習４解答：

(1) 10進数の 1.0 および 2.0 を IEEE-754規格の単精度浮動小数点形式で表現した時の32ビットのビット列をそれぞれ8桁の16進数で表せ。

- 1.0_{10} → 0011 1111 1000 0000 0000 0000 0000 0000 → 3f800000
- 2.0_{10} → 0100 0000 0000 0000 0000 0000 0000 0000 → 40000000

(2) 10進数の 1.0 および 2.0 を IEEE-754規格の倍精度浮動小数点形式で表現した時の64ビットのビット列をそれぞれ16桁の16進数で表せ。

- 1.0_{10} → 0011 1111 1111 0000 0000 0000 0000 0000 0000 0…0 (32桁)
  → 3ff0000000000000
- 2.0_{10} → 0100 0000 0000 0000 0000 0000 0000 0000 0000 0…0 (32桁)
  → 4000000000000000

(3) 10進数の 256.5 を IEEE-754規格の単精度浮動小数点形式で表現した時の32ビットのビット列を8桁の16進数で表せ。

- $256.5_{10} = 100000000.1_2 = 1.000000001_2 \times 2^8$
  → 0100 0011 1000 0000 0100 0000 0000 0000 → 43804000

(4) 10進数の –65536.125 を IEEE-754規格の倍精度浮動小数点形式で表現した時の64ビットのビット列を16桁の16進数で表せ。

- $-65536.125_{10} = -1000000000000000.001_2$
  $= -1.0000000000000001_2 \times 2^{16}$
  → 1100 0000 1111 0000 0000 0000 0000 0010 0…0 (32桁)
  → c0f0000200000000