

計算機構成論 第8回

—命令の実行(2)—

大連理工大学・立命館大学 国際情報ソフトウェア学部

大森 隆行

講義内容

■ 命令の実行

- コンパイラ

- アセンブラ

- リンカ

- ロード

■ Cプログラムからアセンブリコードへの変換

- swap, sortを例に

C言語での開発

main.c

```
int main() {  
    ...  
    printf("Hello");  
    ...  
    proc();  
    ...  
}
```

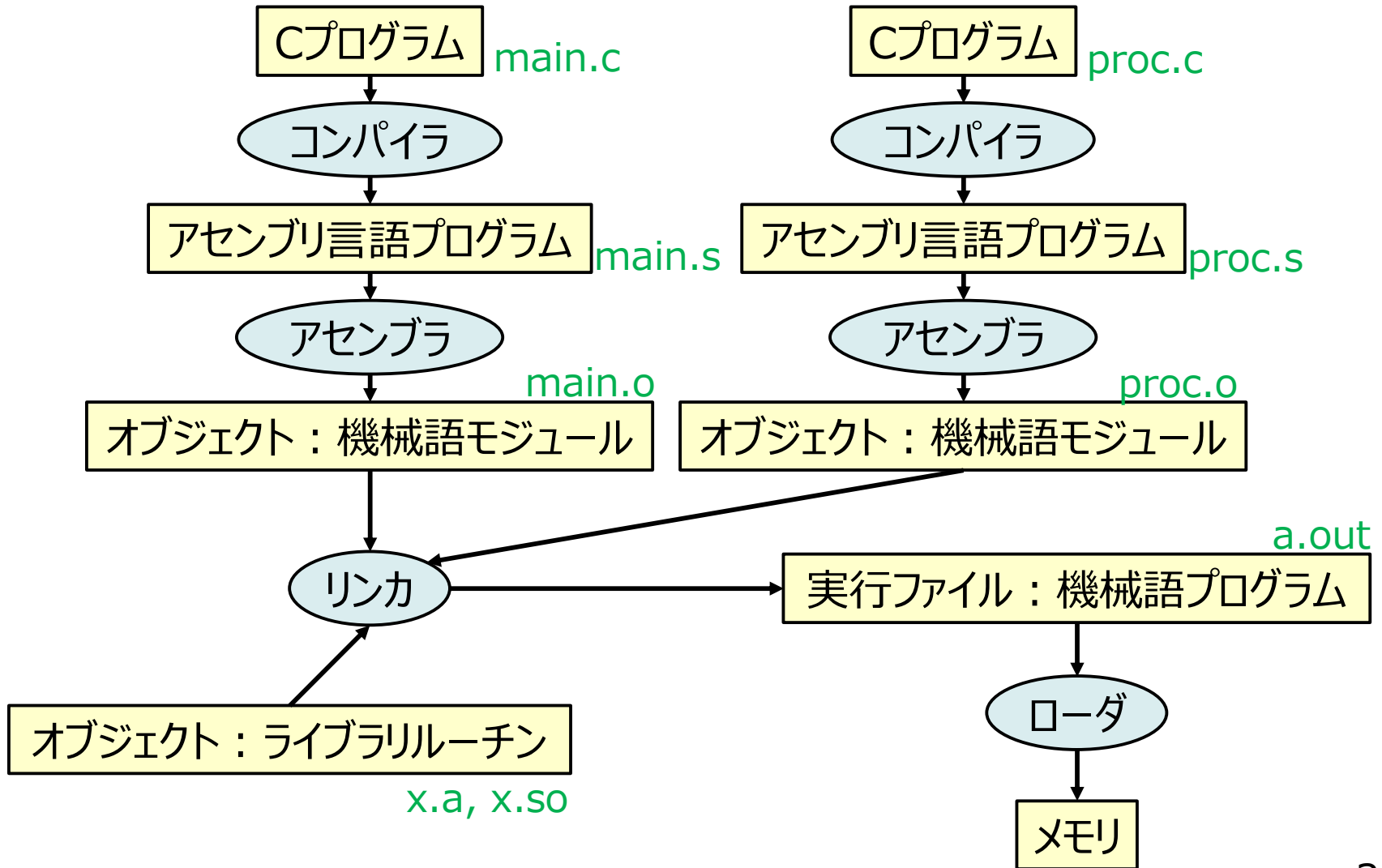
標準ライブラリ内の関数

proc.c

```
void proc() {  
    ...  
    ...  
    ...  
}
```

- C言語でのプログラム開発では、複数のファイルを使うのが一般的
- 標準ライブラリも使用
 - stdio.h, stdlib.h 等々

Cプログラムが起動するまで



コンパイラ

- ソースコードをアセンブリコードに変換
(コンパイルする)
- アセンブリコード
 - 機械が解釈できるコードを
シンボル(記号)で表したもの
 - (基本的に)機械語と一対一変換可能
 - アセンブリコードには擬似命令が
含まれることがある
 - 擬似命令：アセンブリ言語の命令セットには
含まれないが、簡便化のため用意されている命令
e.g., blt (branch less than),
move (レジスタ間のデータコピー)

講義内容

■ 命令の実行

- コンパイラ

- ➡ ■ アセンブラ

- リンカ

- ロード

■ Cプログラムからアセンブリコードへの変換

- swap, sortを例に

アセンブラ

- アセンブリコードを機械語に変換
(アセンブルする)
 - オブジェクトファイル(目的ファイル)に出力 (後述)
- 擬似命令を実際の命令に置き換える
e.g., `blt $t0, $t1, Label`
→ `slt $t2, $t0, $t1`
 `bne $t2, $zero, Label`
`move $t0, $t1`
→ `add $t0, $zero, $t1`

アセンブラ

■ 遠くへの分岐を、分岐＋ジャンプに変換

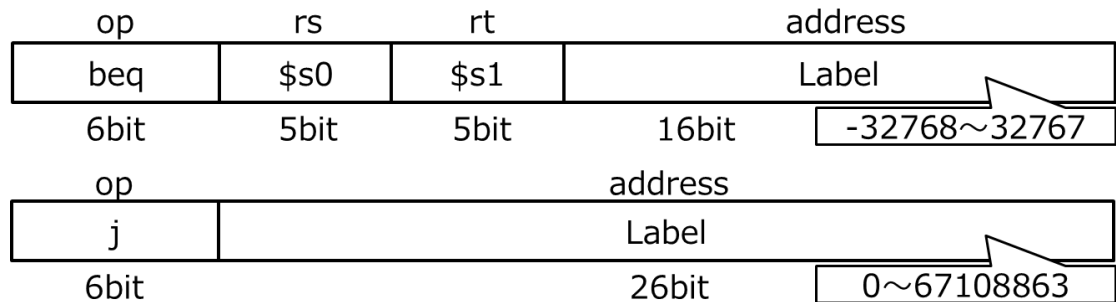
beq \$s0, \$s1, L1



bne \$s0, \$s1, L2

j L1

L2:



■ シンボル・テーブルの生成

- ラベル名と命令のアドレスの対応を管理

アセンブラ

■ オブジェクトファイルの生成

■ 以下の6セクションで構成 (UNIXの場合)

- オブジェクト・ファイル・ヘッダー (object file header)
 - テキスト・セグメント、静的データ・セグメントのサイズを示す
- テキスト・セグメント (text segment)
 - 機械語のプログラムコード
- 静的データ・セグメント (static data segment)
 - 実行中に割り当てられるデータ
- リロケーション情報 (relocation information)
 - プログラムをメモリにロードしたときの絶対アドレスに依存する命令語、データ語を示す
- シンボル・テーブル (symbol table)
 - 未定義のラベルを保持(外部参照も含む)
- デバッグ情報 (debugging information)
 - コンパイルに関する情報。デバッガ等で使用される

講義内容

■ 命令の実行

- コンパイラ

- アセンブラ

➡ ■ リンカ

- ロード

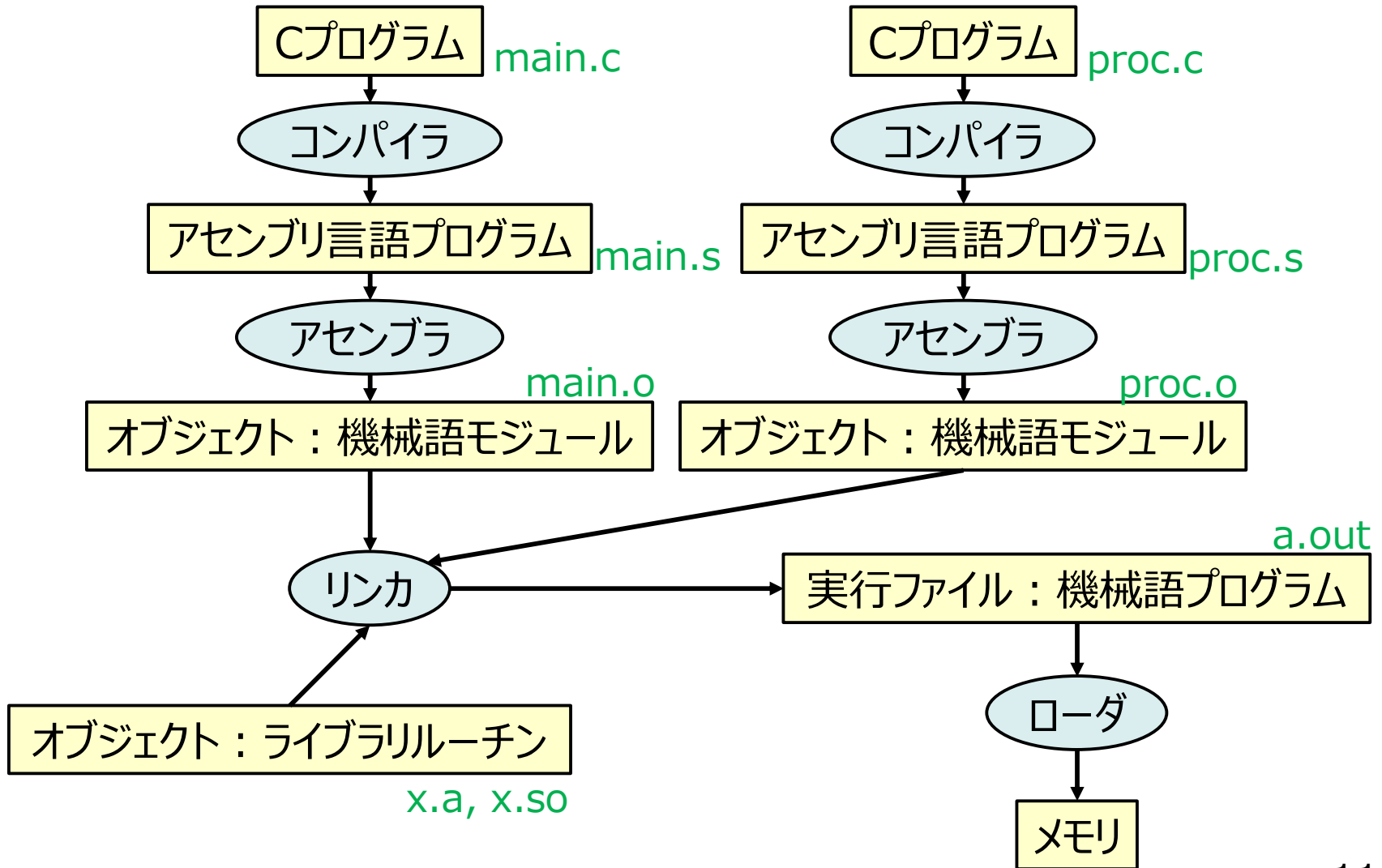
■ Cプログラムからアセンブリコードへの変換

- swap, sortを例に

リンカ

- それぞれ個別にコンパイル、アセンブルされたプログラムをつなぎ合わせて、1つの実行可能なオブジェクトファイルを作る
 - 大規模なプログラムの全体をコンパイルし直す必要がなくなる
 - ライブラリのような再利用性の高いものを後からリンクする
- 外部参照を解決(e.g., 他のファイルにあるラベル名)
- メモリ上のプログラムの配置を決定
 - 絶対アドレッシング(ベース相対アドレッシングでないもの)のアドレスを調整(relocation)

Cプログラムが起動するまで 再掲



(例) リンカの働き

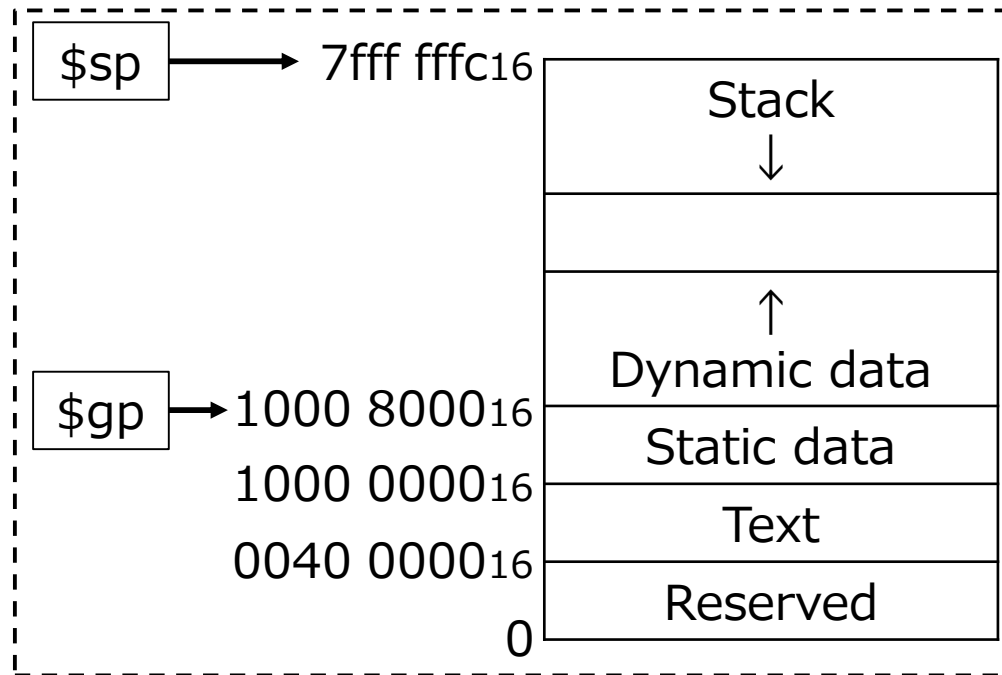
※赤の箇所はリンク過程で更新されるデータ

ProcA.o			
ヘッダー	名前		ProcA
	テキスト・サイズ		100 16
	データ・サイズ		20 16
テキスト・セグメント	アドレス	命令	
	0	lw \$a0, 0(\$gp)	
	4	jal 0	
	
静的データセグメント	アドレス	データ	
	0	(x)	
	
リロケーション情報	アドレス	命令タイプ	依存関係
	0	lw	x
	4	jal	ProcB
	
シンボルテーブル	ラベル	アドレス	
	x	-	
	procB	-	
	

ProcB.o			
ヘッダー	名前		ProcB
	テキスト・サイズ		200 16
	データ・サイズ		30 16
テキスト・セグメント	アドレス	命令	
	0	sw \$a1, 0(\$gp)	
	4	jal 0	
	
静的データセグメント	アドレス	データ	
	0	(y)	
	
リロケーション情報	アドレス	命令タイプ	依存関係
	0	sw	y
	4	jal	ProcA
	
シンボルテーブル	ラベル	アドレス	
	y	-	
	procA	-	
	

(例) リンカの働き

実際のメモリ上の配置(仮定)



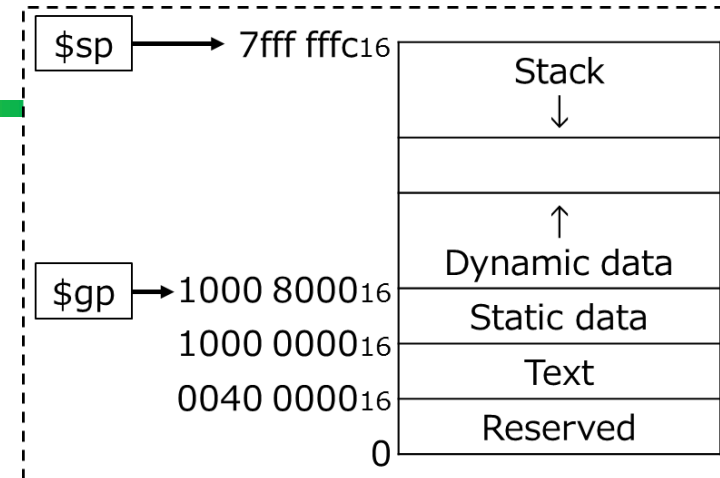
(例) リンカの働き

Linked.o

ヘッダー	テキスト・サイズ	300 16
	データ・サイズ	50 16
テキスト セグメント	アドレス	命令
	0040 0000 ₁₆	lw \$a0, 8000 ₁₆ (\$gp)
	0040 0004 ₁₆	jal 40 0100 ₁₆

	0040 0100 ₁₆	sw \$a1, 8020 ₁₆ (\$gp)
	0040 0104 ₁₆	jal 40 0000 ₁₆
	...	
静的 データ セグメント	アドレス	データ
	1000 0000 ₁₆	(x)

	1000 0020 ₁₆	(y)



ProcA.o

ヘッダー	名前		ProcA
	テキスト・サイズ		10016
	データ・サイズ		2016
テキスト・セグメント	アドレス	命令	
	0	lw \$a0, 0(\$gp)	
	4	jal 0	
	
静的データセグメント	アドレス	データ	
	0	(x)	
	
リロケーション情報	アドレス	命令タイプ	依存関係
	0	lw	x
	4	jal	ProcB
	
シンボルテーブル	ラベル		アドレス
	x		-
	procB		-

ProcB.o

ヘッダー	名前	ProcB	
	テキスト・サイズ	20016	
	データ・サイズ	3016	
テキスト・セグメント	アドレス	命令	
	0	sw \$a1, 0(\$gp)	
	4	jal 0	
	
静的データセグメント	アドレス	データ	
	0	(y)	
	
リロケーション情報	アドレス	命令タイプ	依存関係
	0	sw	y
	4	jal	ProcA
	
シンボルテーブル	ラベル	アドレス	
	y	-	
	procA	-	
	

(例) リンカの働き

\$sp → 7fff fffc₁₆

$$1000\ 8000_{16} + 8000_{16} = 1000\ 0000_{16}$$

$$\begin{array}{r} 0001\ 0000\ 0000\ 0000\ 1000\ 0000\ 0000\ 0000 \\ +\ 1111\ 1111\ 1111\ 1111\ 1000\ 0000\ 0000\ 0000 \\ \hline 0001\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000 \end{array}$$

Linked.o

ヘッダー

テキスト

データ・サイズ

テキスト
セグメント

アドレス	命令
0040 0000 ₁₆	lw \$a0, 8000 ₁₆ (\$gp)
0040 0004 ₁₆	jal 40 0100 ₁₆
...	...
0040 0100 ₁₆	sw \$a1, 8020 ₁₆ (\$gp)
0040 0104 ₁₆	jal 40 000
...	

0040 0000₁₆
0

Reserved

ProcA.o

名前	ProcA
ヘッダー	
テキスト・サイズ	100 ₁₆
データ・サイズ	20 ₁₆
アドレス	命令
0	lw \$a0, 0(\$gp)
4	jal 0
...	...
アドレス	データ
0	(x)

ProcB.o

名前	ProcB
ヘッダー	
テキスト・サイズ	200 ₁₆
データ・サイズ	30 ₁₆
アドレス	命令
0	sw \$a1, 0(\$gp)
4	jal 0
...	...
アドレス	データ
0	(y)

静的
データ
セグメント

アドレス
1000 00
...
1000 00
...

$$1000\ 8000_{16} + 8020_{16} = 1000\ 0020_{16}$$

$$\begin{array}{r} 0001\ 0000\ 0000\ 0000\ 1000\ 0000\ 0000\ 0000 \\ +\ 1111\ 1111\ 1111\ 1111\ 1000\ 0000\ 0010\ 0000 \\ \hline 0001\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010\ 0000 \end{array}$$

講義内容

■ 命令の実行

- コンパイラ

- アセンブラ

- リンカ

- ➡ ■ ロード

■ Cプログラムからアセンブリコードへの変換

- swap, sortを例に

ロード

- リンク後のオブジェクトファイルをメモリに読み込む
- main関数を実行する環境を用意
- 最初の命令を呼び出す

- 動的にリンクされるライブラリ
(DLL: dynamically linked library)
 - 呼び出すためのダミールーチンのみ用意して、必要になったときに、必要な命令をテキスト領域に配置する

確認問題

- 次の各文は、それぞれ何について説明したものか答えよ
 - ソースコードをアセンブリコードに変換するソフトウェア
 - アセンブリコードを機械語に変換するソフトウェア
 - 複数の機械語のコードをつなぎあわせて1つのオブジェクトファイルを作るソフトウェア
 - 動的にリンクされるライブラリ

- 次の各項目は、オブジェクトファイルを構成する6つのセクションに保持されるものを示している。それぞれ、どのセクションか答えよ

- テキスト・セグメント、静的データ・セグメントのサイズ
- 機械語のプログラムコード
- 実行中に割り当てられるデータ
- プログラムをメモリにロードしたときの絶対アドレスに依存する命令語、データ語
- 未定義のラベル

- (a) 静的データ・セグメント
- (b) リロケーション情報
- (c) オブジェクト・ファイル・ヘッダ
- (d) シンボル・テーブル
- (e) デバッグ情報
- (f) テキスト・セグメント

- デバッガ等で使用されるコンパイルに関する情報



講義内容

■ 命令の実行

- コンパイラ

- アセンブラ

- リンカ

- ロード

■ Cプログラムからアセンブリコードへの変換

➡ ■ swap, sortを例に

Cプログラムからアセンブリコードへの変換

- 以下のC言語のソースコードをアセンブリコードに直せ。

```
void swap(int v[], int k){  
    int t;  
    t = v[k];  
    v[k] = v[k+1];  
    v[k+1] = t;  
}
```

レジスタ割り付け：
\$a0:v \$a1:k
\$t0:t

```
swap:  
    sll $t1, $a1, (1)    #$t1 = k*4  
    add $t1, (2), (3)    #$t1 = &(v[k])  
    lw  $t0, (4)         #t = v[k]  
    lw  $t2, (5)         #$t2 = v[k+1]  
    sw  $t2, (6)         #v[k] = $t2  
    sw  $t0, (7)         #v[k+1] = t  
    jr  $ra              #return
```



Cプログラムからアセンブリコードへの変換

- 以下のC言語のソースコードをアセンブリコードに直せ。

```
void sort(int v[], int n){  
    int i, j;  
    for(i=0; i<n; i++){  
        for(j=i-1; j>=0 && v[j]>v[j+1]; j--){  
            swap(v, j);  
        }  
    }  
}
```

```
$a0:v $a1:n  
$s0:i $s1:j
```

- 手順

- ①レジスタの退避
- ②引数の退避
- ③外側のループ
- ④内側のループ
- ⑤swap呼出
- ⑥内側のループ
- ⑦外側のループ
- ⑧レジスタ復元～

Cプログラムからアセンブリコードへの変換

■ 以下のC言語のソースコードをアセンブリコードに直せ。

```
void sort(int v[], int n){
    int i, j;
    for(i=0; i<n; i++){
        for(j=i-1; j>=0 && v[j]>v[j+1]; j--){
            swap(v, j);
        }
    }
}
```

```
sort:
    addi $sp, $sp, -20
    sw    $ra, 16($sp)
    sw    $s3, 12($sp)
    sw    $s2, 8($sp)
    sw    $s1, 4($sp)
    sw    $s0, 0($sp)
```

\$spを必要なだけ進めて、
手続き内で使用するレジスタ
の値を退避

①レジスタの退避

②引数の退避

③外側のループ

④内側のループ

⑤swap呼出

⑥内側のループ

⑦外側のループ

⑧レジスタ復元～

\$a0:v \$a1:n
\$s0:i \$s1:j

Cプログラムからアセンブリコードへの変換

- 以下のC言語のソースコードをアセンブリコードに直せ。

```
void sort(int v[], int n){  
    int i, j;  
    for(i=0; i<n; i++){  
        for(j=i-1; j>=0 && v[j]>v[j+1]; j--){  
            swap(v, j);  
        }  
    }  
}
```

```
move $s2, $a0  
move $s3, $a1
```

swap(v, j) (5行目)で実引数を
設定するときにa0, a1を使用する
ので、値をコピーしておく

- ①レジスタの退避
- ②**引数の退避**
- ③外側のループ
- ④内側のループ
- ⑤swap呼出
- ⑥内側のループ
- ⑦外側のループ
- ⑧レジスタ復元～

```
$a0:v  $a1:n  
$s0:i  $s1:j  
$s2:v  $s3:n
```

Cプログラムからアセンブリコードへの変換

- 以下のC言語のソースコードをアセンブリコードに直せ。

```
void sort(int v[], int n){  
    int i, j;  
    for(i=0; i<n; i++){  
        for(j=i-1; j>=0 && v[j]>v[j+1]; j--){  
            swap(v, j);  
        }  
    }  
}
```

```
move $s0, $zero  
for1tst:  
    slt  $t0, $s0, $s3  
    beq  $t0, $zero, exit1
```

i=0

i<n が満たされ
なければexit1へ
ジャンプ

- ①レジスタの退避
- ②引数の退避
- ③外側のループ
- ④内側のループ
- ⑤swap呼出
- ⑥内側のループ
- ⑦外側のループ
- ⑧レジスタ復元～

\$a0:v \$a1:n
\$s0:i \$s1:j
\$s2:v \$s3:n

Cプログラムからアセンブリコードへの変換

- 以下のC言語のソースコードをアセンブリコードに直せ。

```
void sort(int v[], int n){
    int i, j;
    for(i=0; i<n; i++){
        for(j=i-1; j>=0 && v[j]>v[j+1]; j--){
            swap(v,j);
        }
    }
}
```

```
    addi $s1, $s0, -1
for2tst:
    slti $t0, $s1, 0
    bne $t0, $zero, exit2
    sll $t1, $s1, 2
    add $t2, $s2, $t1
    lw $t3, 0($t2)
    lw $t4, 4($t2)
    slt $t0, $t4, $t3
    beq $t0, $zero, exit2
```

j=i-1

j<0であれば、
exit2へ

\$t1=j*4
\$t2=&(v[j])

\$t3=v[j]
\$t4=v[j+1]

\$t3≤\$t4で
あればexit2へ

- ①レジスタの退避
- ②引数の退避
- ③外側のループ
- ④内側のループ
- ⑤swap呼出
- ⑥内側のループ
- ⑦外側のループ
- ⑧レジスタ復元～

\$a0:v \$a1:n
\$s0:i \$s1:j
\$s2:v \$s3:n

Cプログラムからアセンブリコードへの変換

- 以下のC言語のソースコードをアセンブリコードに直せ。

```
void sort(int v[], int n){  
    int i, j;  
    for(i=0; i<n; i++){  
        for(j=i-1; j>=0 && v[j]>v[j+1]; j--){  
            swap(v, j);  
        }  
    }  
}
```

```
move $a0, $s2  
move $a1, $s1  
jal  swap
```

引数を設定して、
swap呼び出し

- ①レジスタの退避
- ②引数の退避
- ③外側のループ
- ④内側のループ
- ⑤**swap呼出**
- ⑥内側のループ
- ⑦外側のループ
- ⑧レジスタ復元～

```
$a0:v  $a1:n  
$s0:i  $s1:j  
$s2:v  $s3:n
```

Cプログラムからアセンブリコードへの変換

- 以下のC言語のソースコードをアセンブリコードに直せ。

```
void sort(int v[], int n){  
    int i, j;  
    for(i=0; i<n; i++){  
        for(j=i-1; j>=0 && v[j]>v[j+1]; j--){  
            swap(v, j);  
        }  
    }  
}
```

```
addi $s1, $s1, -1  
j     for2tst
```

j--して
内側ループの条件
判定へ戻る

- ①レジスタの退避
- ②引数の退避
- ③外側のループ
- ④内側のループ
- ⑤swap呼出
- ⑥内側のループ
- ⑦外側のループ
- ⑧レジスタ復元～

```
$a0:v $a1:n  
$s0:i $s1:j  
$s2:v $s3:n
```

Cプログラムからアセンブリコードへの変換

- 以下のC言語のソースコードをアセンブリコードに直せ。

```
void sort(int v[], int n){  
    int i, j;  
    for(i=0; i<n; i++){  
        for(j=i-1; j>=0 && v[j]>v[j+1]; j--){  
            swap(v, j);  
        }  
    }  
}
```

```
exit2:  
    addi $s0, $s0, 1  
    j     for1tst
```

i++して
外側ループの条件
判定へ戻る

- ①レジスタの退避
- ②引数の退避
- ③外側のループ
- ④内側のループ
- ⑤swap呼出
- ⑥内側のループ
- ⑦外側のループ
- ⑧レジスタ復元～

```
$a0:v $a1:n  
$s0:i $s1:j  
$s2:v $s3:n
```

Cプログラムからアセンブリコードへの変換

■ 以下のC言語のソースコードをアセンブリコードに直せ。

```
void sort(int v[], int n){
    int i, j;
    for(i=0; i<n; i++){
        for(j=i-1; j>=0 && v[j]>v[j+1]; j--){
            swap(v, j);
        }
    }
}
```

- ①レジスタの退避
- ②引数の退避
- ③外側のループ
- ④内側のループ
- ⑤swap呼出
- ⑥内側のループ
- ⑦外側のループ
- ⑧レジスタ復元～

```
exit1:
    lw    $s0, 0($sp)
    lw    $s1, 4($sp)
    lw    $s2, 8($sp)
    lw    $s3, 12($sp)
    lw    $ra, 16($sp)
    addi  $sp, $sp, 20
    jr    $ra
```

レジスタ復元

呼び出し元へ戻る

<pre> sort: addi \$sp, \$sp, -20 sw \$ra, 16(\$sp) sw \$s3, 12(\$sp) sw \$s2, 8(\$sp) sw \$s1, 4(\$sp) sw \$s0, 0(\$sp) move \$s2, \$a0 move \$s3, \$a1 move \$s0, \$zero for1tst: slt \$t0, \$s0, \$s3 beq \$t0, \$zero, exit1 addi \$s1, \$s0, -1 for2tst: slti \$t0, \$s1, 0 bne \$t0, \$zero, exit2 sll \$t1, \$s1, 2 add \$t2, \$s2, \$t1 </pre>	<pre> lw \$t3, 0(\$t2) lw \$t4, 4(\$t2) slt \$t0, \$t4, \$t3 beq \$t0, \$zero, exit2 move \$a0, \$s2 move \$a1, \$s1 jal swap addi \$s1, \$s1, -1 j for2tst exit2: addi \$s0, \$s0, 1 j for1tst exit1: lw \$s0, 0(\$sp) lw \$s1, 4(\$sp) lw \$s2, 8(\$sp) lw \$s3, 12(\$sp) lw \$ra, 16(\$sp) addi \$sp, \$sp, 20 jr \$ra </pre>
---	---

参考文献

- コンピュータの構成と設計 上 第5版
David A.Patterson, John L. Hennessy 著、
成田光彰 訳、日経BP社
- 山下茂 「計算機構成論 1」 講義資料