

Technische Informatik 1 – Übung 3

Compilations-Vorgang (Computerübung)

David Hasenfratz
27.-28. Oktober 2010



Ziele der Übung

Aufgabe 1

- **Aufbau von Objekt-Dateien (ELF-Format)**
- **Verteilung eines Objekt-Dateien auf RAM und ROM**

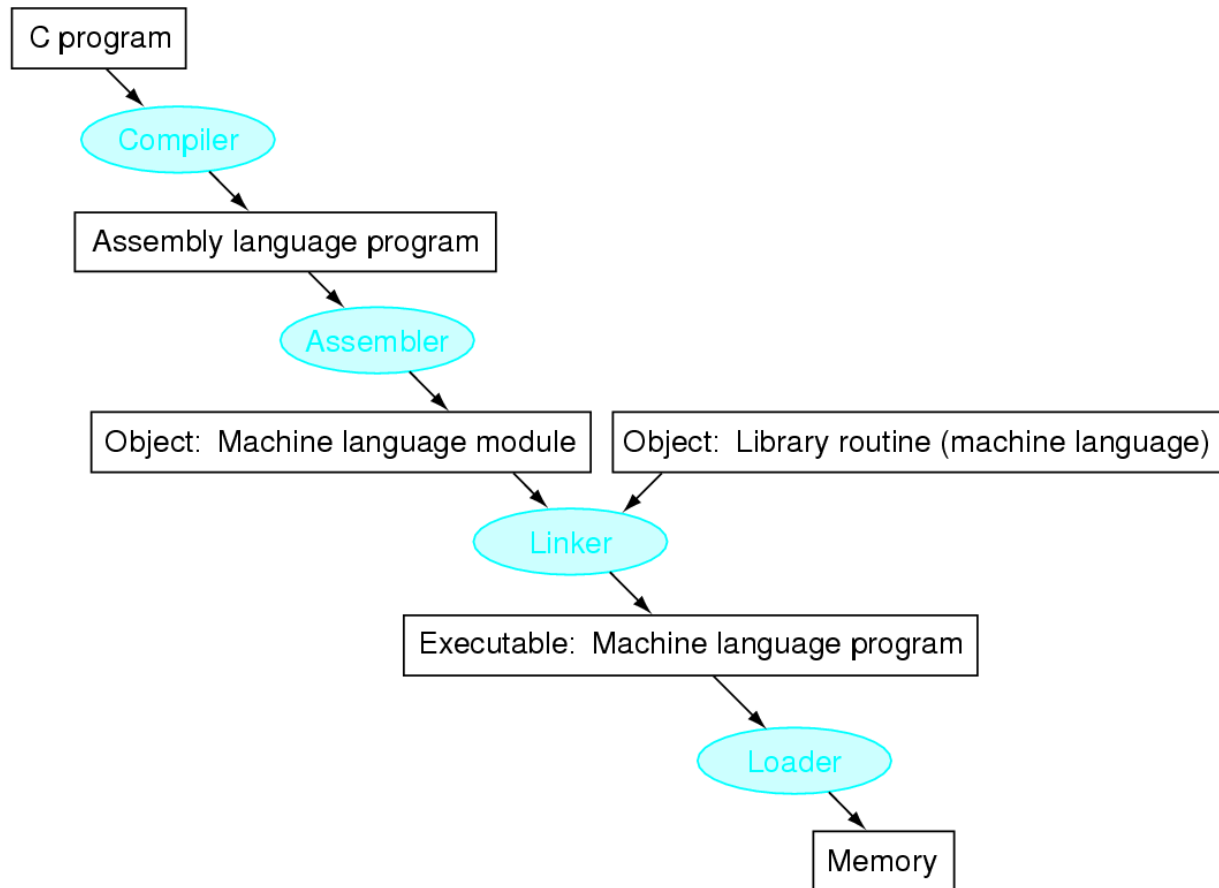
Aufgabe 2

- **Assemblieren, Assembler-Direktiven**
- **Kompilieren, Linken**

Aufgabe 3

- **Analyse einer Objekt-Datei**

Programmübersetzung - Übersicht



Aufgabe 1: Anlegung von Objekt-Dateien im Speicher

Lösung Aufgabe 1 a)

- **.text, .rodata nicht veränderbar ⇒ ROM**
- **.bss, .data veränderbar ⇒ RAM**

Lösung Aufgabe 1 b)

Möglicher Boot-Prozess:

1. Prozessorregister initialisieren (z.B. `sp`, `fp`, `excep`)
2. RAM initialisieren
 - Werte aus `.data` von ROM nach RAM kopieren
 - `.bss` Daten mit 0 initialisieren
3. Programm aus `.text` ausführen

Aufgabe 2: C → Assembler

Lösung Aufgabe 2.1 a)

- **a = 7**
 - **.data**: Wert ist veränderbar, initialisiert mit Wert ungleich null
- **b = 0**
 - **.bss**: Wert ist veränderbar, wird mit 0 initialisiert
- **c = 33 (const)**
 - **.rodata**: Konstante, Wert kann zur Laufzeit nicht verändert werden

Lösung Aufgabe 2.1 b)

d ist eine lokale Variable die pro Funktionsaufruf angelegt wird. Dementsprechend wird d im Stack-Speicher angelegt und nicht im Daten-Segment des Programms.

Lösung Aufgabe 2.1 c)

- **Definition in swap.s**
 - *.globl swap*
- **Objekt-Datei swap.o**
 - Enthält Adresse von *swap*
- **sorter.s**
 - Aufruf der Funktion *swap* mit einer *jump-and-link* Instruktion auf das entsprechende Label

Lösung Aufgabe 2.2 a)

- **Definierte Symbole**
 - *a, b, c, main, print_array, sort, unsorted_array*
 - Sind im eigenen Modul definiert

- **Undefinierte Symbole**
 - *printf, swap*
 - Sind in anderen Modulen definiert

Lösung Aufgabe 2.2 b)

- **Vor dem Linker sind die effektiven Positionen (Adresse) der Funktionen nicht bekannt**
- **Die Sprungziele aller *jal* Instruktionen werden daher vorläufig auf die Adresse 0 gesetzt**
- **Der Linker ist für das korrekte Setzen der Sprungziele verantwortlich**

Lösung Aufgabe 2.2 c)

- Die Relokationstabelle der Objekt-Datei gibt an, an welchen Programminstruktionen der Linker Adressen ersetzen muss
- Beispiel: *00000044 R_MIPS_26 swap*
 - Der Linker muss an der (relativen) Adresse *00000044* im Programmcode, die der Instruktion *jal swap* entspricht, das Sprungziel ersetzen

Lösung Aufgabe 2.2 d)

- **Executable app (Ergebnisse programmabhängig)**
 - swap: a00202b8
 - print array: a002047c
- **Zwei grundlegende Aktionen des Linkers**
 - Programmcode der verschiedenen Objekt-Dateien in eine einzelne ausführbare Datei zusammenfügen
 - Alle Adressen von Funktionsaufrufen und Variablen ersetzen (Relokationstabelle abarbeiten)

Lösung Aufgabe 2.2 e)

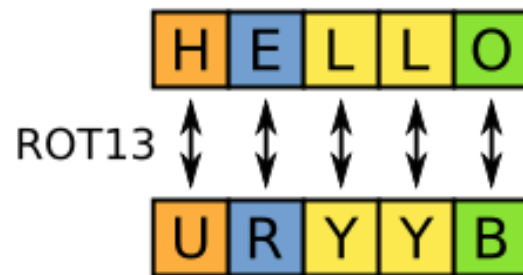
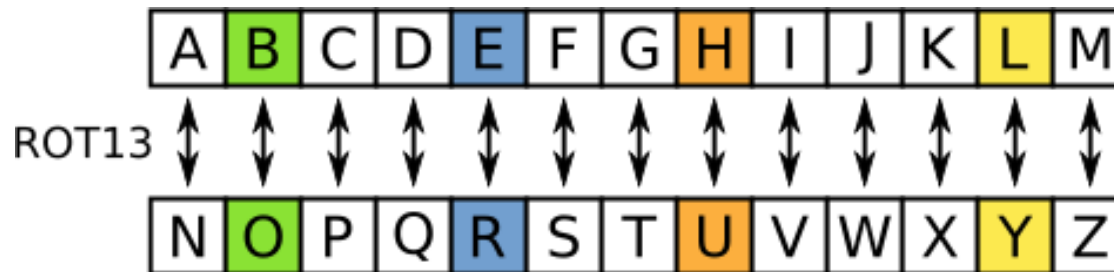
Neben dem übersetzten Programmcode von *swap.o* und *sorter.o* enthält die ausführbare Datei *app* ausserdem die Bibliotheksfunktion *printf*.

Aufgabe 3: Programmanalyse mit objdump

Lösung Aufgabe 3

ROT13 Verfahren:

Verschiebe jeden Buchstaben um 13 Positionen im Alphabet



Lösung Aufgabe 3

```
int encrypt_char ( int c ) {  
    if (((c >= 'a') && (c <= 'm')) ||  
        ((c >= 'A') && (c <= 'M'))) {  
        return c + 13;  
    }  
    else if (((c > 'm') && (c <= 'z')) ||  
             ((c > 'M') && (c <= 'Z'))) {  
        return c - 13;  
    }  
    else {  
        return c;  
    }  
}
```

Lösung Aufgabe 3

```
encrypt_char:
    addiu    v0, a0, -97           # ASCII 'a' = 97
    sltiu   v0, v0, 13            # check if 'a' <= c <= 'm'
    bnez    v0, 'add13'          # if yes goto add13
    nop

    addiu   v0, a0, -65           # ASCII 'A' = 65
    sltiu   v0, v0, 13            # check if 'A' <= c <= 'M'
    beqz    v0, 'check_upper'     # if no goto check_upper
    nop

add_13:
    addiu   a0, a0, 13            # c = c + 13
    j      'end'                  # goto end
    nop
```

Lösung Aufgabe 3

check_upper:

```
    addiu    v0, a0, -110        # ASCII 'n' = 110
    sltiu   v0, v0, 13          # check if 'n' <= c <= 'z'
    bnez    v0, 'sub13'        # if yes goto sub13
    nop
```

```
    addiu    v0, a0, -78        # ASCII 'N' = 78
    sltiu   v0, v0, 13          # check if 'N' <= c <= 'Z'
    beqz    v0, 'end'          # if no goto end
    nop
```

sub13:

```
    addiu    a0, a0, -13        # c = c - 13
```

end:

```
    move    v0, a0              # store result
    jr      ra                  # return
```