LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Praktikum Rechnerarchitektur

Huffmankodierung (A405)

Projektaufgabe – Aufgabenbereich Theorie der Informationsverarbeitung

1 Organisatorisches

Auf den folgenden Seiten finden Sie die Aufgabenstellung zu Ihrer Projektaufgabe für das Praktikum. Die Rahmenbedingungen für die Bearbeitung werden in der Praktikumsordnung festgesetzt, die Sie über die Praktikumshomepage[1] aufrufen können.

Wie in der Praktikumsordnung beschrieben, sind die Aufgaben relativ offen gestellt. Besprechen Sie diese innerhalb Ihrer Gruppe und konkretisieren Sie die Aufgabenstellung. Die Teile der Aufgabe, in denen Assembler-Code anzufertigen ist, sind für die 64-Bit x86-Architektur (x86-64) unter Verwendung der SSE-Erweiterungen zu schreiben.

Der **Abgabetermin** ist der **24. Juli 2020, 11:59 Uhr (MESZ)**. Die Abgabe erfolgt per Git in das für Ihre Gruppe eingerichtete Projektrepository. Bitte beachten Sie die in der `README.md` angegebene Liste von abzugebenden Dateien.

Die **Abschlusspräsentationen** finden in der Zeit vom **10.08.2020 – 28.08.2020** statt. Weitere Informationen werden noch bekannt gegeben. Beachten Sie, dass die Folien für die Präsentation am obigen Abgabetermin im PDF-Format abzugeben sind. Geben Sie *außerdem* eine Datei mit Notizen zum Vortrag ab, mit deren Hilfe die Folien auch ohne den gehaltenen Vortrag nachvollziehbar sind. Der Kurzvortrag entfällt ersatzlos.

Sofern die Rahmenbedingungen (Hygiene-, Abstandsregeln etc.) für eine Präsenzprüfung nicht vorliegen, kann gemäß §13a APSO die geplante Prüfungsform auf eine virtuelle Präsentation (Videokonferenz) oder eine kurze schriftliche Fernprüfung umgestellt werden. Die Entscheidung über diesen Wechsel wird möglichst zeitnah, spätestens jedoch 14 Tage vor dem Prüfungstermin nach Abstimmung mit dem zuständigen Prüfungsausschuss bekannt gegeben.

Bei Fragen/Unklarheiten in Bezug auf den Ablauf und die Aufgabenstellung wenden Sie sich bitte an Ihren Tutor.

Wir wünschen Ihnen viel Erfolg und Freude bei der Bearbeitung Ihrer Aufgabe!

Mit freundlichen Grüßen
Die Praktikumsleitung

PS: Vergessen Sie nicht, sich rechtzeitig in TUMonline zur Prüfung anzumelden. Dies ist Voraussetzung für eine erfolgreiche Teilnahme am Praktikum im laufenden Semester.

[1] `https://gepasp.in.tum.de/eraweb/`

2 Huffmankodierung

2.1 Überblick

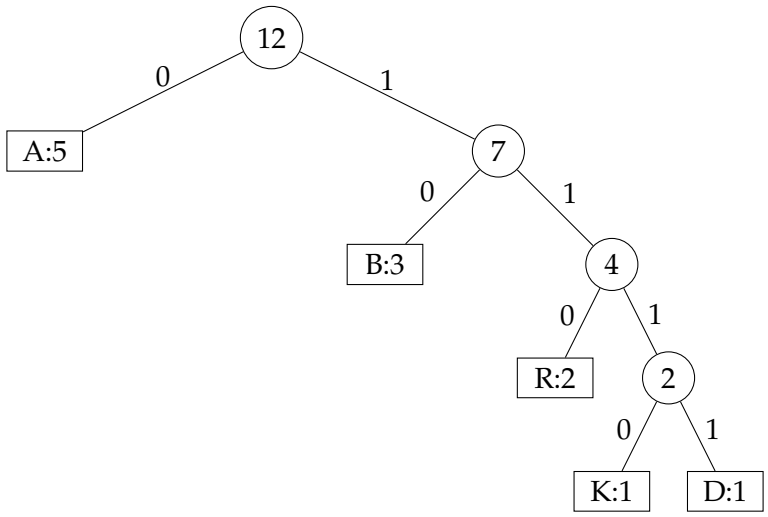
Die Informationstheorie ist ein Feld der Mathematik, in welchem man sich allgemein mit dem Kodieren von Nachrichten aufgrund von statistischen Gegebenheiten beschäftigt. Sie werden in Ihrer Projektaufgabe einen bestimmten Teilbereich näher beleuchten und einen bestimmten Algorithmus in Assembler implementieren.

2.2 Funktionsweise

Die Huffmankodierung ist ein zur Datenkompression eingesetztes Entropiekodierungschema. Bei der Kompression wird zunächst eine Häufigkeitsanalyse der zu kodierenden Nachricht durchgeführt. Anschließend wird ein sogenannter Huffmanbaum erstellt, der Symbolen entsprechend ihren Häufigkeiten unterschiedliche Bitsequenzen zuweist. Wir betrachten dazu ein einfaches Beispiel mithilfe des Wortes *ABRAKADABRAB*.
Zunächst führt man die Häufigkeitsanalyse durch:

Zeichen	A	B	R	K	D
Häufigkeit	5	3	2	1	1

Anschließend konstruiert man den Huffmanbaum als präfix-freien Baum, in welchem die häufigsten Symbole die kürzesten Codewörter zugewiesen bekommen.



Der fertige Baum kann dann verwendet werden, um mit den einzelnen Bits an den Ästen die Eingabe binär zu kodieren. Dazu erstellt man ein sogenanntes *Dictionary*:

Symbol	Kodierung
A	0
B	10
R	110
K	1110
D	1111

Das gewählte Beispiel wird dann mithilfe der Tabelle buchstabenweise kodiert:

```
ABRAKADABRAB = 0 10 110 0 1110 0 1111 0 10 110 0 10
```

Beachten Sie, dass die Gruppierung der Bits nur zur visuellen Verdeutlichung erfolgt. Das Eingabewort kann dann statt mit $12 \cdot 8 = 96$ Bit mit $1+2+3+1+4+1+4+1+2+3+1+2 = 25$ Bit dargestellt werden. Natürlich muss zu diesen 25 Bit noch zusätzlich der Baum als Datenstruktur gespeichert werden, dennoch lässt sich aber leicht einsehen, dass die Huffmankodierung für längere Texte eine gute Kompressionsrate erreichen kann.

2.3 Aufgabenstellungen

Ihre Aufgaben lassen sich in die Bereiche Konzeption (theoretisch) und Implementierung (praktisch) aufteilen. Sie können (müssen aber nicht) dies bei der Verteilung der Aufgaben innerhalb Ihrer Arbeitsgruppe ausnutzen. Antworten auf konzeptionelle Fragen sollten an den passenden Stellen in Ihrer Ausarbeitung in angemessenem Umfang erscheinen. Entscheiden Sie nach eigenem Ermessen, ob Sie im Rahmen Ihres Abschlussvortrags auch auf konzeptionelle Fragen eingehen. Die Antworten auf die Implementierungsaufgaben werden durch Ihrem Code reflektiert.

2.3.1 Theoretischer Teil

- Erarbeiten Sie anhand geeigneter Sekundärliteratur die genaue Funktionsweise der Huffmankodierung. Berechnen Sie dazu zunächst einige Beispiele auf Papier. Warum müssen die Bits der Ausgabesymbole nicht durch spezielle Trennzeichen markiert werden?
- Erarbeiten Sie ein geeignetes Format zum Speichern des Huffmanbaums im Arbeitsspeicher. Schlagen Sie dazu nach, wie Bäume in C üblicherweise dargestellt werden.
- Erarbeiten Sie ein geeignetes Ausgabeformat, in welchem die komprimierte Nachricht gespeichert werden soll. Versuchen Sie, möglichst effizient zu arbeiten indem Sie möglichst wenig Bits verbrauchen. Das Erreichen der optimalen Darstellung ist ein wichtiger Teil Ihrer Aufgabe!
- Untersuchen Sie Ihre fertige Implementierung auf Performanz. Errechnen Sie auch die durchschnittliche Kompressionsrate für Eingabendaten Ihrer Wahl.

2.3.2 Praktischer Teil

- Implementieren Sie im Rahmenprogramm I/O-Operationen in C, mit welchen Sie Ihrem Assemblerprogramm die Inhalte einer eingelesenen Datei als Pointer übergeben können.

  Eine unkomprimierte Nachricht ist als ASCII-Text in der Eingabedatei gespeichert.

- Implementieren Sie in der Datei mit dem Assemblercode eine Funktion

  ```
  int huffman_encode(char *data, char *result, unsigned int result_size)
  ```

  welche einen Pointer auf die unkomprimierten Eingabedaten `data` übergeben bekommt und die Huffmankomprimierten Daten in das Array `result` speichert. Denken Sie daran, dass die Huffmankompression sowohl Daten als auch das Dictionary zurückgeben muss. Wählen Sie hierfür ein sinnvolles Format (es ist Ihnen hierfür erlaubt, die Parameter der Funktion gegebenenfalls zu erweitern, sollte es Ihnen sinnvoll erscheinen). Das Argument `result_size` gibt die maximale Größe des für `result` reservierten Speicherbereichs an. Der Rückgabewert soll im Erfolgsfall die Länge der kodierten Daten sein. Wenn die Funktion fehlschlägt (beispielsweise, weil das Ergebnisarray nicht groß genug gewählt wurde) soll -1 zurückgegeben werden. Speichern Sie die komprimierte Nachricht im von Ihnen gewählten Format in einer vom Benutzer bestimmbaren Datei im Rahmenprogramm (d.h. in C).

2.3.3 Zusatzaufgabe

Implementieren Sie in der Datei mit dem Assemblercode eine Funktion

```
int huffman_decode(char *data, char *result, unsigned int result_size)
```

welche einen Pointer auf die Huffmankomprimierten Eingabedaten `data` übergeben bekommt und die unkomprimierten Daten in das Array `result` speichert. Sie können die Funktionssignatur gegebenenfalls erweitern, um das Dictionary komfortabel übergeben zu können. Der Parameter `result_size` gibt die maximale Größe des für `result` reservierten Speicherbereichs an. Der Rückgabewert soll im Erfolgsfall die Länge der unkomprimierten Daten sein. Wenn die Funktion fehlschlägt (beispielsweise, weil das Dictionary nicht vollständig ist) soll -1 zurückgegeben werden. Speichern Sie die dekomprimierte Nachricht in einer vom Benutzer frei wählbaren Datei.

2.4 Allgemeine Bewertungshinweise

Die folgende Liste soll Ihnen als Gedächtnisstütze beim Bearbeiten der Aufgaben dienen. Beachten Sie ebenfalls die in der Praktikumsordnung angegebenen Hinweise.

- Stellen Sie unbedingt sicher, dass Ihre Abgabe auf der Referenzplattform des Praktikums (`lxhalle`) kompiliert und funktionsfähig ist.

- Fügen Sie Ihrem Projekt ein funktionierendes `Makefile` hinzu, welches durch den Aufruf von `make` Ihr Projekt kompiliert.
- Verwenden Sie keinen Inline-Assembler.
- Verwenden Sie SIMD-Befehle, wenn möglich.
- Verwenden Sie keine x87-FPU- oder MMX-Instruktionen. Sie dürfen alle SSE-Erweiterungen bis SSE4.2 benutzen. AVX-Instruktionen dürfen Sie benutzen, sofern Ihre Implementierung auch auf Prozessoren ohne AVX-Erweiterungen lauffähig ist.
- Sie dürfen die Signatur der in Assembler zu implementierenden Funktion nur dann ändern, wenn Sie dies (in Ihrer Ausarbeitung) rechtfertigen können.
- I/O-Operationen dürfen grundsätzlich in C implementiert werden.
- Denken Sie daran, das Laufzeitverhalten Ihres Codes zu testen (Sichere Programmierung, Performanz) und behandeln Sie alle möglichen Eingaben, auch Randfälle. Ziehen Sie ggf. alternative Implementierungen als Vergleich heran.
- Eingabedateien, welche Sie generieren, um Ihre Implementierungen zu testen, sollten mit abgegeben werden.
- Verwenden Sie für die Ausarbeitung die bereitgestellte TEX-Vorlage und legen Sie sowohl die PDF-Datei als auch sämtliche TEX-Quellen in das Repository.
- Stellen Sie Performanz-Ergebnisse nach Möglichkeit grafisch dar.
- Vermeiden Sie unscharfe Grafiken und Screenshots von Code.
- Geben Sie die Folien für Ihre Abschlusspräsentation im PDF-Format ab. Achten Sie auf hinreichenden Kontrast (schwarzer Text auf weißem Grund!) und eine angemessene Schriftgröße. Verwenden Sie 4:3 als Folien-Format.
- Zusatzaufgaben (sofern vorhanden) müssen nicht implementiert werden. Es gibt keine Bonuspunkte.