Prüfung Rechnerarchitektur WS 2021/22

Im Wintersemester 2021/2022 wird die Prüfungsleistung für das Modul Rechnerarchitektur nicht durch eine Klausur erbracht, sondern in Form einer Programmieraufgabe mit schriftlicher Ausarbeitung (Ausarbeitung im Umfang von bis zu 4 A4-Seiten).

Bitte lesen Sie sich diese Anleitung zur Prüfungsleistung gründlich durch!

Bei Fragen schreiben Sie bitte eine Email an `moeller@techfak.de`

Aufgabenstellung

Realisieren Sie ein Programm zur Sortierung eines Datensatzes mithilfe des **bitweisen LSB-Radix-Sortierverfahrens in Assembler für Intel 64-Bit-Prozessoren (x86-64).**

Es liegt ein **Datensatz** vor, dessen Elemente jeweils aus einem Key (dem Schlüssel, nach dem sortiert wird) und einem Payload (eine zusätzliche Information, bspw. ein Index, die im Sortiervorgang stets gemeinsam mit dem Key transportiert wird) bestehen. Der Key ist eine vorzeichenbehaftete ganze Zahl in Zweierkomplement-Kodierung (2K-Zahl) mit 4 Bytes. Der Payload ist eine vorzeichenlose ganze Zahl in Dual-Kodierung (Dualzahl) mit 4 Bytes. Der Key liegt im Speicher vor dem Payload. Key und Payload schließen im Speicher unmittelbar aneinander an. Aufeinanderfolgende Elemente schließen im Speicher ebenfalls unmittelbar aneinander an.

Dieser Datensatz soll mit **bitweisem LSB-Radix-Sort** sortiert werden, wobei eine Sortierung sowohl in aufsteigender als auch in absteigender Reihenfolge möglich sein soll. Beim bitweisen LSB-Radix-Sort (auch "binäres LSD-Radix-Sort", LSD = "least significant digit") wird der Datensatz in jedem Durchgang vollständig durchmustert und dabei nach einem bestimmten Bit des Keys sortiert. Nach einem Durchgang liegen bspw. alle Elemente, bei denen das betrachtete Bit des Keys 0 ist, vor allen Elementen, bei denen das betrachtete Bit des Keys 1 ist (bzw. umgekehrt je nach Sortierrichtung). Der erste Durchgang sortiert die Daten nach dem niederwertigsten ("least significant") Bit (LSB) des Keys. In den folgenden Durchgängen werden die Bits in aufsteigender Reihenfolge der Signifikanz behandelt. Der letzte Durchgang sortiert die Daten nach dem höchstwertigen ("most significant") Bit (MSB) des Keys. In jedem Durchgang wird der *gesamte* Datensatz durchmustert, es erfolgt also *keine* Aufteilung in zwei Gruppen, die dann separat behandelt werden. Das Verfahren funktioniert allerdings nur unter der **Voraussetzung, dass Elemente mit gleichem Key-Bit ihre Reihenfolge untereinander in einem Durchgang nicht verändern**.

Ihr Programm soll in der Lage sein, den Datensatz sowohl aufwärts als auch abwärts anhand des Keys zu sortieren. Für die Sortierung ist der **Wert des Keys** entscheidend, nicht dessen Bitfolge. Es ist also Vorzeichen und Betrag der Zweierkomplement-Zahl zu berücksichtigen (bitte finden Sie dafür eine effiziente Lösung).

Bitte beachten Sie bei der Implementation, dass es effizienter sein kann, wenn schwer vorhersagbare Sprünge ("branch prediction") eliminiert und bspw. durch bedingte Transportbefehle (`cmov*`) ersetzt werden.

Anmeldung und Abgabe

Es ist keine Anmeldung erforderlich.

Die Abgabe muss bis zum **Sonntag, 6. März 2022, 23:59** erfolgen. Eine verspätete Abgabe gilt als Nichtteilnahme. Die Deadline ist strikt. Die Abgabe erfolgt über die "E-Prüfung" im Lernraum der Rechnerarchitektur-Vorlesung (Zugang via eKVV). Bitte kennzeichnen Sie Programmdatei und Ausarbeitung mit Ihrem Account-Namen an der Technischen Fakultät. Für Erika Mustermann wäre dies z.B. `emustermann`. Die Programmdatei muss in diesem Beispiel den Dateinamen `lsbsort_emustermann.asm`, die Ausarbeitung den Dateinamen `ausarbeitung_emustermann.pdf` haben. Achten Sie bitte auf die korrekte Angabe des Account-Namens, da wir diesen ggf. auch für die Email-Kommunikation verwenden. Die bereitgestellte Datei `support_lsbsort.asm` wird nicht mit abgegeben. Weitere Quellcode-Dateien können nicht mit abgegeben werden. Sie können Ihre Abgabe bis zur Deadline beliebig oft verändern. Bitte beachten Sie die Hinweise in der bereitgestellten Anleitung `Anleitung_E-Pruefung_RA.PDF`. *Hinweis: Bitte testen Sie den Abgabevorgang deutlich vor der Deadline.*

Wiederholung

Eine einmalige Wiederholung (2. Versuch) im Wintersemester 2021/2022 kann zur Notenverbesserung in Anspruch genommen werden und ist **nur bei Teilnahme am 1. Versuch** möglich, da dieselbe Aufgabe in einem verkürzten Zeitraum bearbeitet wird. Bitte beachten Sie, dass die Leistung des 1. Versuchs auch bei einer Wiederholung der Prüfung eingetragen wird.

Die Wiederholung erfolgt durch **Überarbeitung von Programm und Ausarbeitung mit demselben Thema**. Für die Überarbeitung wird eine **Frist von 2 Wochen** nach der Mitteilung der Note eingeräumt.

In nachfolgenden Semestern gilt die dann festgelegte Prüfungsform (in der Regel eine Klausur).

Programm

Fügen Sie Ihren Programmcode in das **vorgegebene Rahmenprogramm** `lsbsort_emustermann.asm` ein (s.u.). Dieses erzeugt in Zusammenspiel mit dem Code in der bereitgestellten Datei `support_lsbsort.asm` zunächst zufällige Daten nach den Vorgaben der Definitionen `ELEMS` (Anzahl der Elemente) und `SEED` (Initialwert Zufallszahlengenerator) in `support_lsbsort.asm`. Dann wird Ihr Sortier-Unterprogramm (Marke `lsbsort`) aufgerufen. Dabei legt eine weitere Definition `UP` in `support_lsbsort.asm` die Sortierrichtung fest (1 = aufwärts, 0 = abwärts), die Ihr Programm realisieren soll. Im vorgegebenen Code des Unterprogramms werden die Daten hier lediglich kopiert; bitte ersetzen Sie diesen Code durch Ihre Implementation des Sortierverfahrens. Ist das Symbol `STORE` definiert, wird der sortierte Datensatz unter dem Namen `lsb_sorted.dat` abgespeichert. Abschließend erfolgt eine Überprüfung, ob die Daten tatsächlich sortiert sind; dies wird durch eine Ausgabe auf der Standardausgabe angezeigt.

Die bereitgestellte Datei `support_lsbsort.asm` wird nicht mit abgegeben. Bitte legen Sie auch **keine weiteren Quellcode-Dateien** an; diese können nicht abgegeben werden, da nur zwei Dokumente (eine Assembler-Datei und eine PDF-Datei mit der Ausarbeitung) elektronisch eingereicht werden können. Kann Ihr Programm wegen fehlender weiterer Quellcode-Dateien nicht assembliert werden, wird es als nicht funktionstüchtig bewertet (s.u.).

Gefordert ist ein Assemblerprogramm für **Intel x86-64-Prozessoren im Intel-Syntax** (nicht IA-32, nicht AT&T-Syntax). Ihr Programm muss mit der vorgegebenen Version des Assemblers `nasm` assemblierbar sein (s.u.). Es ist nicht gestattet, Assemberprogramme oder Teile davon durch Compilieren aus Hochsprachen zu erzeugen (Betrugsversuch, s.u.). Es ist nicht gestattet, Bibliotheken einzubinden.

Das Programm soll nur **auf einer einzigen CPU (bzw. Core)** laufen. Vermeiden Sie also eine Parallelisierung auf mehreren CPUs / Cores.

Kommentieren Sie Ihr Programm ausgiebig **in deutscher Sprache** (s.u.). Geben Sie auch **Kommentare für jedes Unterprogramm** an, die dessen Eingabe- und Ausgabe-Parameter, genutzte "globale" Parameter und die von der Funktion modifizierten ("zerstörten") Register angeben. Parameter können dabei Speicherbereiche oder Register sein.

Bitte nutzen Sie **keine bedingte Einbindung** von Code (z.B. `%if` oder `%ifdef`). Bei der Zählung der Instruktionen (s.u.) werden solche Konstrukte nicht berücksichtigt; es würden also auch nicht eingebundene Instruktionen mitgezählt.

Bitte beachten Sie: Ihr Programm muss den **Vektor `data` unverändert lassen** und die **sortierten Daten im Vektor `sorted_data`** ablegen!

Bitte behalten Sie die Kodierung des Quellcodes im UTF-8-Unicode-Format bei.

Ausarbeitung

Format

Auf einer vorangestellten Titelseite geben Sie bitte "Ausarbeitung Rechnerarchitektur Wintersemester 2021/22" mit dem Titel "Assembler-Implementierung eines bitweisen LSB-Radix-Sort" sowie **Ihren Namen, Vornamen, Studiengang und Ihre Matrikelnummer** an. Fügen Sie darunter bitte folgenden Text ein:

> "Hiermit erkläre ich, dass ich das Programm und die vorliegende Ausarbeitung selbstständig verfasst habe. Ich habe keine anderen Quellen als die angegebenen benutzt und habe die Stellen in Programm und in der Ausarbeitung, die anderen Quellen entnommen wurden, in jedem Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht. Diese Erklärung gilt auch ohne meine Unterschrift, sobald ich das Programm und die Ausarbeitung über die E-Prüfung der Vorlesung Rechnerarchitektur im Lernraum des eKVV an der Universität Bielefeld und unter Angabe meiner Matrikelnummer in der Ausarbeitung eingereicht habe."

Die eigentliche Ausarbeitung umfasst ***bis zu* 4 A4-Seiten** mit einer Textbreite von 16 cm und einer Texthöhe von 23 cm. Der Text ist im Zeilenabstand "ein-einhalb-zeilig" zu formatieren. Als Schriftart wählen Sie dabei bitte Times-Roman in der Größe 12 pt.

Bitte beachten Sie: Bei Abgaben, welche die Seitenbegrenzung überschreiten, wird nur der Teil bis zum Erreichen der maximalen Seitenzahl bewertet. Bei Abgaben, die Zeilenabstand und / oder Schriftgröße verringern, wird nur der Teil bis zu einer entsprechenden Wortzahl bewertet. Das Erreichen der maximalen Seitenzahl ist nicht erforderlich (bewertet wird die inhaltliche Vollständigkeit, nicht der Umfang).

Die **Liste Ihrer Quellen** (Literatur, Webseiten etc.) geben Sie bitte nachfolgend zum eigentlichen Text an, beginnend auf einer neuen Seite.

Die Titelseite und die Seite(n) mit der Liste der Quellen werden nicht bei der Seitenzählung berücksichtigt.

Das eingereichte **PDF-Dokument muss ein Format aufweisen, aus dem der Text extrahiert werden kann** (für den Plagiatstest). Vermeiden Sie deshalb bitte Rastergrafiken für Text und verschlüsselte Formate. Die Abgabe wird als nicht bestanden bewertet, wenn auf einmalige Nachfrage per Email keine für die Plagiatsprüfung geeignete Fassung zur Verfügung gestellt wird. *Hinweis: Sie können die Extrahierbarkeit von Text z.B. mit dem Linux-Tool `pdftotext` testen.*

Sie können das bereitgestellte **LaTeX-Vorlage** `ausarbeitung_emustermann.tex` verwenden. Diese enthält bereits einen Vorschlag für eine inhaltliche Gliederung nach den untenstehenden Angaben. Behalten Sie diese bitte möglichst bei.

Inhalt

Bitte *vermeiden* Sie eine Einführung, in der Sie die Aufgabenstellung rekapitulieren oder grundlegendes Wissen zu Sortierverfahren, zur Assemblerprogrammierung oder zur Rechnerarchitektur wiedergeben. Folgen Sie bitte möglichst der vorgegebenen Gliederung in der LaTeX-Vorlage.

Erklären Sie Ihre Implementation des bitweisen LSB-Radix-Sort. Gehen Sie dabei auch auf folgende Aspekte Ihrer Implementation ein:

- Implementation des gesamten Sortiervorgangs
- Implementation des Sortierdurchgangs für ein Bit
- Lösung für das Beibehalten der Reihenfolge von Elementen mit gleichen Key-Bits in jedem Durchgang
- Lösung für die Behandlung von Vorzeichen und Betrag der Keys (effiziente Lösung gesucht!)
- Lösung für die Prüfung einzelner Bits im Datenwort
- Lösung für das Sortieren in beide Richtungen
- Vergleich der Laufzeit ("user"-Anteil ohne "system"-Anteil) Ihres Verfahrens mit der des bereitgestellten Vergleichsprogramms `stlsort` (s.u.) für verschiedene Größen des Datensatzes
- Schwachstellen und Verbesserungsmöglichkeiten.

Hinweise zur Einzelleistung und zu Betrugsversuchen

Die Programmieraufgabe mit Ausarbeitung ist ein Ersatz für eine Klausur. In einer Klausur ist sichergestellt, dass die Leistung als **Einzelleistung** erbracht wird. Derselbe Maßstab gilt auch für die Programmieraufgabe mit Ausarbeitung. Eine **Zusammenarbeit mit anderen Personen in jeglicher Form gilt deshalb als Betrugsversuch**. Für Programmdatei und Ausarbeitung wird ein elektronischer Plagiatstest durchgeführt. Dadurch aufgedeckte **Betrugsversuche werden geahndet**: Gespräch mit dem Modulverantwortlichen, ggf. Meldung ans Prüfungsamt mit Vermerk, Eintragung der Leistung als "nicht bestanden", Ausschluss vom zweiten Versuch im WS 2021/2022, Schreiben vom Dekan, mögliche Exmatrikulation bei weiteren (vorherigen oder nachfolgenden) Betrugsversuchen.

Wir recherchieren im Vorfeld mögliche **Quellen im Internet**. Bitte denken Sie daran, dass eventuell auch andere TeilnehmerInnen dieselben Quellen verwenden. Auch wenn wir diese Quellen nicht kennen sollten, würde dadurch ein Betrugsversuch aufgedeckt.

Bitte wenden Sie sich mit Fragen zur Aufgabenstellung oder bei technischen Fragen nur an den Veranstalter (Email an `moeller@techfak.de`), nicht an KommilitonInnen oder an TutorInnen.

Sowohl im Quellcode Ihres Programms als auch in Ihrer Ausarbeitung müssen **alle verwendeten Quellen gekennzeichnet** sein. Im **Quellcode** sind alle Programmteile, die aus anderen Quellen (bspw. Hilfeseiten im Internet) stammen, zu kennzeichnen. Diese Teile gelten allerdings nicht als selbst erbrachte Leistung; die erreichte Punktzahl wird entsprechend reduziert. Wird die Quelle nicht angegeben, liegt ein Betrugsversuch vor. Wenn Sie selbst Teile Ihres Programms öffentlich im Internet zugänglich machen, werten wir Ihre Abgabe als Plagiat der Internet-Quelle (die Internet-Quelle gilt dabei als nicht selbst verfasst, auch wenn sie unter Ihrem Klarnamen veröffentlicht wurde). In der **Ausarbeitung** müssen wörtliche Zitate (auch Übersetzungen)

gekennzeichnet sein. Bei zu großem Anteil (gekennzeichneter) wörtlicher Übernahmen gelten diese Teile der Ausarbeitung nicht mehr als selbst erbrachte Leistung; die erreichte Punktzahl wird entsprechend reduziert.

Es gilt als **Betrugsversuch, wenn das Assemblerprogramm durch Compilieren aus einer Hochsprache erzeugt wurde**.

Bewertungskriterien

Die Leistung ist benotet. Abweichungen von dieser Vorgabe sind mit dem Prüfungsamt zu klären.

Die maximale erreichbare Punktzahl ist 100. Die Bestehensgrenze ist 50% (50 Punkte). 50 Punkte werden aus der Bewertung des Programms ermittelt, 50 Punkte aus der Bewertung der Ausarbeitung inkl. Kommentierung des Programms. Da Ausarbeitung und Programm in engem Bezug stehen, kann die Leistung nur bestanden werden, wenn für das **Programm mindestens 15 Punkte** *und* für die **Ausarbeitung mindestens 15 Punkte** erreicht werden; zudem müssen in der **Summe mindestens 50 Punkte** erreicht werden.

Bewertung des Programms

Die Bewertung des Programms erfolgt mit maximal 50 Punkten. Es müssen mindestens 15 Punkte erreicht werden. Beim Erwerb von Zusatzpunkten (s.u.) kann die maximale Punktzahl überschritten werden.

Es wird zunächst geprüft, ob Ihr Programm die **Aufgabenstellung erfüllt**, d.h. tatsächlich bitweises LSB-Radix-Sort in Assembler für Intel-x86-64-Prozessoren und im Intel-Syntax implementiert wurde. Ist dies nicht der Fall, gilt die Aufgabe als nicht erfüllt und die Prüfung damit als nicht bestanden. Bitte beachten Sie, dass auch Mischformen mit anderen Sortierverfahren oder Varianten (z.B. bitweises *MSB*-Radix-Sort oder *byteweises* LSB-Radix-Sort) nicht akzeptiert werden. Ansonsten erfolgt die Bewertung wie folgt:

Funktionstüchtigkeit des Programms (20 Punkte): Ihre Abgabe wird von uns mit dem "Netwide Assembler" `nasm` in der Version auf den GZI-Maschinen (2.13.02) zu einer ausführbaren Datei assembliert. Dazu verwenden wir eine Version von `support_lsbsort.asm` mit anderen Werten der Definitionen und zusätzlichem Testcode. Ein nicht assemblierbares Programm wird nicht weiter untersucht und gilt als "nicht funktionstüchtig" (0 Punkte). *Hinweis: Bitte testen Sie deshalb vor Abgabe auf einer GZI-Maschine, ob Ihr Programm mit diesem Assembler assemblierbar ist!* Ein assemblierbares Programm wird von uns mehreren Tests mit unterschiedlichen Werten für die Definitionen `ELEMS`, `SEED` und `UP` getestet (s.o.). Der Anteil der Testläufe, bei denen Ihr Programm eine korrekte Sortierung erzeugt, bestimmt den Grad der Funktionstüchtigkeit und damit die zugewiesene Punktzahl (bspw. funktionstüchtig in der Hälfte der Fälle: 10 Punkte). *Hinweis: Testen Sie deshalb Ihr Programm ausführlich mit unterschiedlichen Werten für die genannten Definitionen*.

Anzahl der Assembler-Instruktionen (15 Punkte): Bewertet wird die Anzahl der benötigten Assembler-Instruktionen. Bitte beachten Sie, dass bei der automatischen Zählung Konstrukte für die bedingte Einbindung (z.B. `%ifdef`) nicht berücksichtigt werden. Wenn Sie also bspw. separaten Code für die beiden Sortierrichtungen angeben, werden die Instruktionen in beiden Teilen gezählt. Unsere Musterlösung benötigt 40 Instruktionen. Bewertungsskala: bis zu 50 Instruktionen 15 Punkte, bis zu 70 Instruktionen 10 Punkte, bis zu 100 Instruktionen 5 Punkte, sonst 0 Punkte. Werden weniger als 40 Instruktionen benötigt, werden 5 zusätzliche Punkte vergeben. *Hinweis: Entfernen Sie unnötige Instruktionen. Ihr Unterprogramm `lsbsort` muss z.B. keine Register retten und restaurieren.*

Laufzeit Ihres Programms (15 Punkte): Die von uns erarbeitete Musterlösung hat für 10 Mio. Elemente eine ähnliche Laufzeit wie das Vergleichsprogramm `stlsort` (siehe unten). Die Laufzeit Ihres Programms im Vergleich zu `stlsort` fließt in die Bewertung ein. Sie können die Laufzeit Ihres Programms ermitteln, indem Sie beim Start `time` auf der Kommandozeile voranstellen. Relevant ist dabei die ausgegebene "User"-Zeit. Bewertungsskala: bis zur 2-fachen Laufzeit im Vergleich zu `stlsort` 15 Punkte, bis zur 3-fachen Laufzeit 10 Punkte, bis zur 5-fachen Laufzeit 5 Punkte, sonst 0 Punkte. Ist Ihr Programm schneller als `stlsort`, werden 5 zusätzliche Punkte vergeben. Die relativen Laufzeiten (Radix-Sort zu `stlsort`) werden bei der Bewertung über mehrere Messungen gemittelt. Ein nicht assemblierbares oder nicht zumindest bei einem Laufzeittest funktionstüchtiges Programm erhält hierbei 0 Punkte.

Bewertung der Ausarbeitung und Kommentierung

Die Bewertung der Ausarbeitung und Kommentierung erfolgt mit maximal 50 Punkten. Es müssen mindestens 15 Punkte erreicht werden.

Ausarbeitung in deutscher Sprache (40 Punkte): Die Ausarbeitung ist in deutscher Sprache vorzulegen (zu Ausnahmen s.u.). Bewertet wird die inhaltliche Korrektheit (u.a. auch Übereinstimmung mit der Implementation), die Vollständigkeit, die Verständlichkeit der Darstellung sowie die Präzision und Konsistenz der Formulierungen. Bitte achten Sie auf eine verständliche Beschreibung. Wenn z.B. Register `r12` als Zeiger auf das nächste zu sortierende Element verwendet wird, dann führen Sie dies z.B. als "Quelladresse (hier Register `r12`)" ein und nehmen Sie in der folgenden Beschreibung jeweils auf die "Quelladresse" Bezug statt auf `r12`. *Hinweis: Bitte verwenden Sie die Rechtschreibprüfung Ihres Editors und lesen Sie Ihre Ausarbeitung vor der Abgabe nochmals gründlich durch.*

Kommentierung des Programms in deutscher Sprache (10 Punkte): Bewertet wird, wie ausführlich Ihr Programm kommentiert wurde. Dabei muss jede Programmzeile (Instruktion) mit einem verständlichen Kommentar in deutscher Sprache (zu Ausnahmen s.u.) versehen sein; Ausnahmen sind nur offensichtliche Befehlsabfolgen wie bspw. `push`-/`pop`-Sequenzen am Anfang und Ende von Unterprogrammen. Bitte vermeiden Sie dabei eine Beschreibung der Assembleranweisung (z.B. `inc rdx ; erhöhe Register rdx um 1`), sondern erläutern Sie mit dem Kommentar Ihren Algorithmus (z.B. `inc rdx ; Adresse des nächsten Bytes im Eingabepuffer`). Verwenden Sie möglichst dieselben Bezeichnungen (z.B. "Quelladresse", s.o.) wie in der Ausarbeitung.

Hinweis: Sowohl die Ausarbeitung als auch die Kommentierung wird in deutscher Sprache eingefordert, um einfacher Bezüge zwischen den Kommentaren und der Beschreibung Ihres Verfahrens in der Ausarbeitung herstellen zu können. In begründeten Ausnahmefällen kann beides in Englisch erfolgen; dies ist aber vor Beginn der Bearbeitung mit dem Prüfenden zu klären.

Bereitgestellte Materialien

Es werden folgende Textdokumente, Programme und Skripte (für das Betriebssystem Linux) auf der Webseite der Veranstaltung

https://www.ti.uni-bielefeld.de/html/teaching/WS2122/techinf1/index.html

bereitgestellt:

- `Anleitung_E-Pruefung_RA.PDF`: Anleitung für die E-Prüfung.
- `ausarbeitung_emustermann.tex` **(bitte Account-Namen im Dateinamen sowie Angaben zu Ihrer Person in der Datei vor der Abgabe anpassen)**: Latex-Vorlage für die Ausarbeitung. Die daraus erzeugte PDF-Datei liegt ebenfalls bei.

- `lsbsort_emustermann.asm` (**bitte Account-Namen im Dateinamen sowie Angaben zu Ihrer Person in der Datei vor der Abgabe anpassen**): Rahmen für Ihr Assembler-Programm. Bitte fügen Sie Ihre Unterprogramme und deren Aufruf im gekennzeichneten Bereich (Marke `lsbsort`) ein. Bitte lassen Sie die Einbindung von `support_lsbsort.asm` unverändert.

  In der bereitgestellten Form erzeugt das Programm einen Datensatz mit Zufallszahlen je nach den Werten der Definitionen von `ELEMS` und `SEED`, ruft das Sortier-Unterprogramm auf (Marke `lsbsort`, gewünschte Sortierrichtung nach Definition `UP`), schreibt die sortierten Daten in die Ausgabedatei `lsb_sorted.dat` und prüft, ob die Sortierung korrekt war (Ausgabe auf der Standardausgabe). Der Aufruf der ausführbaren Datei erfolgt ohne Kommandozeilen-Parameter.
- `support_lsbsort.asm`: Enthält Funktionen zum Erzeugen der Daten, für Systemrufe und zur Überprüfung auf Sortiertheit sowie das Hauptprogramm, aus dem Ihr Unterprogramm `lsbsort` aufgerufen wird. Diese Datei wird nicht mit abgegeben. Sie dürfen nur die Werte der Definitionen ändern (s.o.). *Hinweis: Prüfen Sie vor Abgabe von `lsbsort_emustermann.asm` nochmals, ob das Programm mit der unveränderten Datei `support_lsbsort.asm` assemblierbar und funktionstüchtig ist.*
- `Makefile`: einfaches Makefile zum Assemblieren Ihrer Lösung und zum Compilieren der Hilfsprogramme `stlsort` und `prtdata`. Der Aufruf erfolgt auf der Kommandozeile wie folgt:
  - `make lsbsort_emustermann`: Erzeugt eine ausführbare Datei mit demselben Rumpfnamen wie die Quelldatei (`lsbsort_emustermann.asm`). Benötigt den Assembler `nasm`.
  - `make stlsort`: Erzeugt eine ausführbare Daten `stlsort` aus `stlsort.C`. Benötigt den C++-Compiler `g++`.
  - `make prtdata`: Erzeugt eine ausführbare Datei `prtdata` aus `prtdata.c`. Benötigt den C-Compiler `gcc`.

  Hinweis: Vorgesehen ist die Verwendung des Makefiles und der Aufruf von `make` von der Kommandozeile. Bei Einsatz einer Entwicklungsumgebung können wir keine Unterstützung leisten. Sie können die Entwicklung auf einem der GZI-Rechner durchführen, falls auf Ihrem eigenen Rechner die benötigten Programme nicht installiert bzw. nicht installierbar sind.
- `prtdata.c`: C-Programm zum Anzeigen von Ausgabedateien. Das Programm zeigt jeweils den Index, den Wert des Keys (dezimal), das Bitmuster des Keys (hexadezimal) sowie den Payload (hexadezimal) jedes Elements an. Aufruf mit Angabe des Dateinamens, z.B.: `prtdata lsb_sorted.dat` oder `prtdata stl_sorted.dat`.
- `stlsort.C`: C++-Programm zum Sortieren einer Datei mithilfe der Funktion `std::sort()` aus der C++ Standard Template Library (STL). `stlsort` weist ebenfalls Definitionen für `ELEMS`, `SEED` und `UP` auf, die entsprechend angepasst werden müssen. Der Aufruf erfolgt ohne Kommandozeilen-Parameter. Es wird die Ausgabedatei `stl_sorted.dat` erzeugt (falls das Symbol `STORE` definiert ist). Bitte beachten Sie, dass Elemente mit gleichen Keys anders sortiert sein können als bei bitweisem LSB-Radix-Sort.

Wichtiger Hinweis zur Arbeit im GZI

Die erzeugten Dateien mit den sortierten Daten können sehr groß sein (z.B. 100 Mio. Elemente mit je 8 Bytes: fast 1 GByte), um Laufzeiten im Sekundenbereich zu realisieren. Bitte erzeugen Sie diese Dateien nur temporär und löschen Sie diese nach Benutzung wieder aus dem Dateisystem. Sie können das Abspeichern durch Auskommentieren der Definition von `STORE` unterbinden.

Alternativ können Sie die Dateien auch im Verzeichnis `/tmp` ablegen. Dort werden sie beim täglichen Neustart der GZI-Maschinen gelöscht.