Rechnerorganisation SS18
Testat P.2

TECHNISCHE UNIVERSITÄT DARMSTADT

Prof. Dr.-Ing. Michael Goesele, M.Sc. Daniel Thürck
Version 1 vom 3. Juni 2018

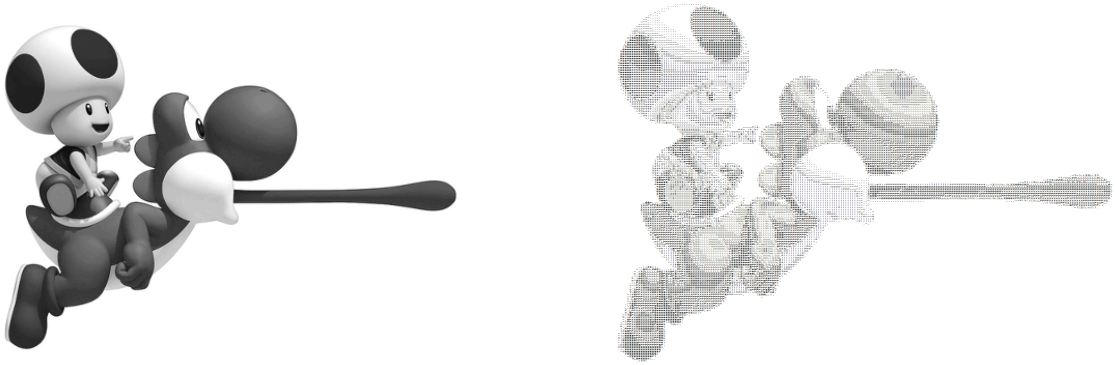
Hinweis: Die Abgabe ihres Codes erfolgt per Moodle; die **Abgabefrist ist der 17.06. um 23:30 Uhr**. Näheres zur anschließenden Besprechung Ihrer Ergebnisse mit Ihrer Tutorin bzw. Ihrem Tutor wird in Kürze bekannt gegeben.
Dieses Praxistestat soll individuell bearbeitet und abgegeben werden; es gelten die üblichen Richtlinien zum Plagiarismus des Fachbereichs 20, siehe `https://www.informatik.tu-darmstadt.de/studium_fb20/im_studium/studienbuero/plagiarismus/index.de.jsp`.

Einführung: ASCII-Art

Hinweis: Verwenden Sie die **bereitgestellte Vorlage** und bearbeiten Sie die Aufgaben in der **angegebenen Reihenfolge**, da diese aufeinander aufbauen. Lesen Sie sich zunächst die ganze Aufgabenstellung durch, um einen Überblick zu erhalten.
Nehmen Sie sich **ausreichend Zeit** zur Bearbeitung und kommentieren Sie Ihren Code zwecks späterer Nachvollziehbarkeit.
Die Aufgaben wurden ausschließlich auf den **ISP - Poolrechnern** getestet. Ihre Abgabe wird von den Tutoren bzw. Tutorinnen ebenso auf diesen Rechnern bewertet - es gibt keinen Support für eigene Konfigurationen.

In diesem Programmiertestat soll ein IA32 - Programm geschrieben werden, das ein Graustufenbild ausliest, verkleinert und anschließend ein Bild in Form einer ASCII-Textdatei speichert. “ASCII-Art” bezeichnet hier die Darstellung von Pixeln eines Bildes durch ASCII-Zeichen, die durch ihre unterschiedliche Form für den Betrachter wie verschiedene Graustufen wirken.

Als Eingabeformat wird ein vereinfachtes PGM-Format verwendet; alle Eingabebilder müssen kleiner als 1024^2 Pixel sein; Breit und Höhe müssen darüberhinaus ohne Rest durch 8 teilbar sein. Der Dateiname muss `input.pgm` sein. Das Codeframework enthält bereits einige Beispieldateien.



Beispiel für ein 256 x 256 px Bild in ASCII-Art (zur Quelle des Bildes siehe Codeframework).

Coderahmen

Für diese Aufgabe steht Ihnen ein Coderahmen zur Verfügung. Alle Änderungen durch Sie sind in der Datei `ascii_art.s` im markierten Bereich nach dem label `main` vorzunehmen.

Im Codeframework ist zusätzlich zu den Ihnen aus der Vorlesung bekannten Segmenten `.data` und `.text` noch das Segment `.bss` benutzt. In diese Segment kann Speicherplatz byte-weise für globale Variablen angelegt werden. Alle hier angelegten Variablen werden vom Assembler mit 0 initialisiert.

Funktionenübergreifend stehen Ihnen folgende Variablen bereit:

- `in_width`, `out_width` - Breite des Eingabe- bzw. verkleinerten Bildes (Fomat: `int`),
- `in_height`, `out_height` - Höhe des Eingabe- bzw. verkleinerten Bildes (Fomat: `int`),
- `in_buffer`, `out_buffer`, `out_ascii_buffer` - reservierte Speicherplätze der Größe 1024^2 Byte für den Inhalt des Eingabebildes, des verkleinerten Bildes sowie der auszugebenden ASCII-Art (Fomat: `byte` - Array),
- `fpu_exchg` - Reservierte 4 - Byte Variable zur Kommunikation mit der FPU, da diese nicht direkt auf die Allzweckregister zugreifen kann (Fomat: `int`).

Das `.bss` Segment in `ascii_art.s` darf von Ihnen beliebig mit weiteren Variablen erweitert werden. Hilfe dazu erhalten Sie im Manual des GNU Assemblers.

In `io.s` stehen Ihnen darüberhinaus eine Reihe von global definierten Funktionen zur Verfügung:

- `copy_image` - kopiert das Eingabebild im Format `in_width x in_height` aus `in_buffer` nach `out_buffer`; Metadaten werden ebenso kopiert,
- `write_output` - schreibt das verkleinerte Bild im Format `out_width x out_height` aus `out_buffer` als `output.pgm` auf die Festplatte,
- `write_ascii` - schreibt das in ASCII-Art konvertierte, evtl. verkleinerte Bild im Format `out_width x out_height` aus `out_ascii_buffer` als `output.html` auf die Festplatte; die ausgegebene Datei stellt einen HTML-Viewer da, mit dem Sie Ihre Ausgabe im Browser (wichtig: herauszoomen!) begutachten können.

Die Funktionen werden im gegebenen Rahmen bereits an den richtigen Stellen aufgerufen.

Assembly und Debugging

Da die Funktionen des Coderahmens auf mehrere Dateien verteilt sind, ist beim Bau des Programmes ein Extraschritt notwendig (*linken*). Der gesamte Bauprozess ist durch das beiliegende Makefile allerdings automatisiert.

Auf den ISP - Poolrechnern können Sie im Ordner des entpackten Coderahmens mittels

```
> make
```

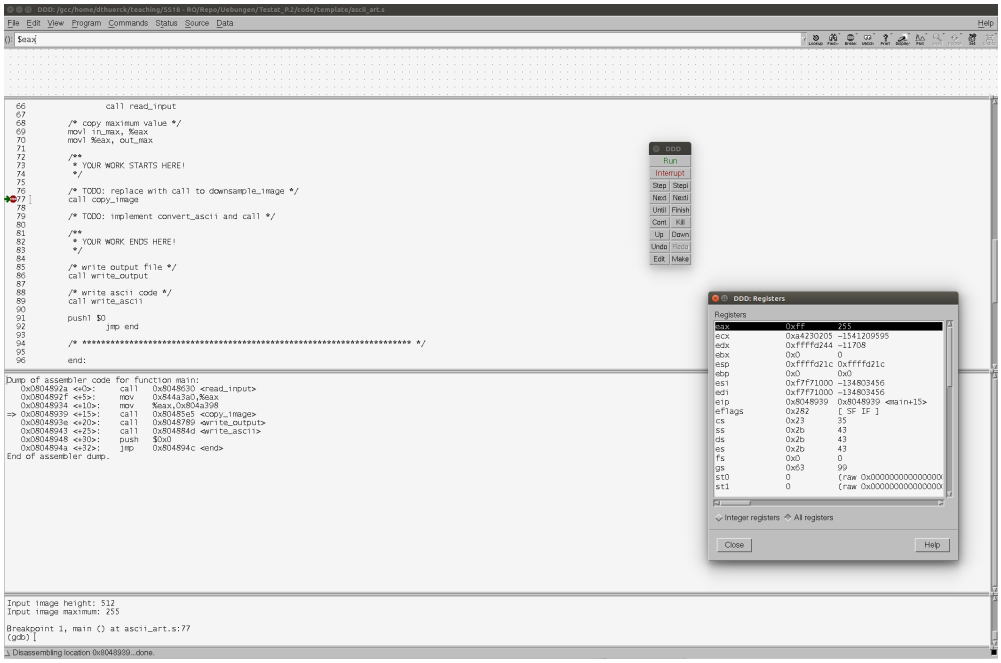
alle Schritte automatisiert ausführen. Zum Zwecke des Debuggings ermöglicht der Coderahmen, den Befehl

```
> make debug
```

auszuführen. Dadurch werden Informationen zum Source Code in das entstehende Programm eingebunden, die dann vom Debugger ausgelesen werden können.

Zur Entwicklung in IA32 Assembler benötigen sie nur einen Texteditor (lies: *vim*) sowie den bereits installierten Assembler GAS, der im Makefile aufgerufen wird. Für mehr Entwicklungskomfort - Syntax Highlighting, graphischer Debugger - wird u.a. die IDE *SASM*[1] angeboten. SASM ist nicht auf den ISP-Poolrechnern vorinstalliert. Sollten Sie SASM nutzen, so wählen Sie entsprechend GAS und die AT&T Syntax aus.

[1] `https://dman95.github.io/SASM/english.html`



Sollten Sie den GNU Debugger `gdb` nutzen wollen, empfehlen wir den *Data Display Debugger*, kurz `ddd` als graphischen Aufsatz. `ddd` ist jedoch nicht auf den ISP-Poolrechnern vorinstalliert, sondern muss von Ihnen lokal gebaut werden.

Da `gdb` ein komplexes Werkzeug ist, seien die Einführungen

`https://www.cs.umb.edu/~cheungr/cs341/Using_gdb_for_Assembly.pdf`

sowie

`https://www.cs.umd.edu/~srhuang/teaching/cmsc212/gdb-tutorial-handout.pdf`

empfohlen.

Aufgaben

Aufgabe 1.1 Konvertierung von Graustufenwerten zu ASCII-Zeichen

Zunächst sollen Sie das Eingabebild der Größe `out_width x out_height` aus dem Array `out_buffer` zu entsprechenden ASCII-Zeichen umwandeln. Im Eingabeformat PGM wird jedes Pixel durch eine Ganzzahl im Bereich $[0, 255]$ im Speicher repräsentiert. Jedes Pixel belegt also genau 1 Byte.

Der Graustufenwert soll nun in ein ASCII-Zeichen mit ähnlichem "Deckungsgrad" der schwarzen Farbe umgewandelt werden. Dazu steht Ihnen als "Graustufentabelle" das Byte-Array `format_b2w` zur Verfügung, das 59 Graustufen abdeckt. Ihre Aufgabe ist es nun, jeden Graustufenwert im Array `in_buffer` von $g \in [0, 255]$ auf $ix \in [0, 58]$ (Array-indizes sind hier stets 0-basiert) abzubilden und das zugehörige Zeichen aus `format_b2w` an die entsprechende Position in `out_ascii_buffer` zu schreiben:

$$ix = \text{round}\left(\frac{g}{255} \cdot 58\right), \textbf{ ASCII: } \texttt{format\_b2w}[ix]$$

Implementieren Sie dazu ein Unterprogramm `convert_ascii`, das jeden Pixel wie oben beschrieben umwandelt und an der entsprechenden Stelle in `out_ascii_buffer` abspeichert. Um Fehler durch Integer-Arithmetik zu vermeiden, benutzen Sie zur Abbildung auf den Wertebereich $[0, 58]$ die FPU.

Eine gute Referenz mit den wichtigsten FPU-Befehlen finden Sie unter

`https://c9x.me/x86/`

FPU-Befehle erkennt man meist an dem Präfix "F". Beachten Sie inbesondere die FPU-Befehle zum Laden und Speichern von ganzen Zahlen (mit "I" als Teil des Mnemonic).

Ergänzen Sib abschließend den Unterprogrammaufruf für `convert_ascii` in `main`. Funktioniert Ihr Code korrekt, so sollten Sie ihr Ergebnis durch Öffnen von `output.pgm` in einem modernen Browser ansehen können.

Aufgabe 1.2 Downsampling des Eingabebildes

Selbst moderne Browser erlauben kein ausreichendes Zoomen, um Ihr Werk in Gänze betrachten zu können. In dieser Teilaufgabe soll daher das Eingabebild zunächst verkleinert werden, ohne zu viele Details zu verlieren, sog. *downsampling*.

Sie sollen dazu ein 2x2 - Boxfiltering implementieren. Aus einem 8×2 großen Eingabebild mit den Graustufenwerten

0	1	2	3	4	5	6	7
10	11	12	13	14	15	16	17

wird durch 2x2 - Boxfiltering

$\frac{1}{4}(0+1+10+11)$	$\frac{1}{4}(2+3+12+13)$	$\frac{1}{4}(4+5+14+15)$	$\frac{1}{4}(6+7+16+17)$

Die Pixel des Eingabebildes werden also horizontal und vertikal in 2 x 2 Pixel große "Blöcke" eingeteilt, die je einem Pixel im Ausgabebild entsprechen. Der Wert dieses Pixels ist der gerundete Durchschitt aller Graustufenwerte im Block.

Nach Ausführen von `read_input` liegen die Pixel zeilenweise nacheinander in Speicher. Die Arbeit auf Byte-Ebene bringt allerdings mit sich, dass auf die Speicherorganisation Rücksicht genommen werden muss; die im ISP-Pool verwendeten Intel-Systeme folgen alle der *little endian* Organisation, die Sie bereits aus der Vorlesung kennen. Ein 8×2 Pixel großes Bild mit folgenden Graustufenwerten:

0	1	2	3	4	5	6	7
10	11	12	13	14	15	16	17

wird im Speicher also folgendermaßen abgespeichert (mit wachsenden Speicheradressen):

3 2 1 0 7 6 5 4 13 12 11 10 17 16 15 14

Um das Handling dieses Umstandes zu vereinfachen, werden Sie immer 2 Blöcke auf einmal verarbeiten, bei denen dann je eine Zeile einem doubleword entspricht.

a) Implementieren Sie ein Unterprogramm `downsample_doublepixel`, welches 2 Blöcke downsampled und die resultierenden Bytes in `%ah` und `%al` zurückgibt.

   Als Eingabe soll die Funktion die Adresse des oberen doublewords der Eingabeblöcke auf dem Stack erhalten.

   Ein Beispiel: Gegeben Sei obiges 8×2 Bild, das ab der Adresse 0x200 im Speicher liegt. Als Eingabe erhält die Funktion die Speicheradresse 0x204, lädt also die beiden doublewords an den Adressen 0x204 sowie 0x20C. Damit verkleinert sie die beiden $2x2$ - Blöcke

4	5	6	7
14	15	16	17

   und setzt

   - `%ah` $= \text{round}(\frac{1}{4}(6+7+16+17))$
   - `%al` $= \text{round}(\frac{1}{4}(4+5+14+15))$.

   Berechnen Sie, ähnlich der obigen Aufgabe, die Durchschnittswerte per FPU.

b) Wenden Sie nun Ihre bereits implementierte Funktion `downsample_doublepixel` an, um das komplette Eingabebild zu verkleinern und im Array `out_buffer` zu speichern. Erstellen Sie dafür ein neues Unterprogramm `downsample_image` und ersetzen Sie den Aufruf von `copy_image`.

   Machen Sie sich dabei noch einmal klar, welche Byte-Reihenfolge im Speicher vorliegt. Eine falsche Reihenfolge macht sich durch Zacken im Ausgabebild bemerkbar.

   Passen Sie auch die Metadaten `out_width` und `out_height` entsprechend an.

   Haben Sie `downsample_image` korrekt implementiert, so enthält die Ausgabedatei `output.pgm` ihr verkleinertes Bild.