Grafischer Prozessor-Simulator

Studienarbeit

Studiengang Informatik

OST – Ostschweizer Fachhochschule

Campus Rapperswil-Jona

Herbstsemester 2020

Autoren: Yves Boillat, Eliane Schmidli

Betreuer: Prof. Stefan Richter

Danksagung:

Wir möchten hier allen Personen danken, die uns bei der Studienarbeit unterstützt haben. Vor allem danken wir unserem Studienarbeitsbetreuer Prof. Stefan Richter und Florian Bruhin für die vielen wertvollen Inputs und die grosse Unterstützung. Zudem danken wir den Eltern von Eliane Schmidli für das Korrekturlesen der Arbeit.

1 Inhalt

1.1 Bildverzeichnis

1.2 Tabellenverzeichnis

2 Zusammenfassung

2.1 Aufgabenstellung

Studienarbeit «Prozessor-Simulator»

Einführung

Für das Verständnis von Betriebssystemen und hardwarenaher Software ist ein Verständnis der Hardware auf abstraktem Niveau unabdinglich. Dafür soll ein Prozessor-Simulator entwickelt werden, der es Studierenden ermöglicht, die Funktionsweise der Hardware abstrakt nachzuvollziehen.

Es gibt bereits einige Prozessor-Simulatoren, z.B. https://sourceforge.net/projects/johnnysimulator/ oder https://draemm.li/various/mirac/. Leider sind diese aber detaillierter als benötigt und simulieren selten echte CPU-Sätze. Sie sind eher für das Studium des Prozessorbaus interessant denn für hardware-nahe Programmierung.

Aufgabe

Ziel dieser Arbeit soll es sein, einen grafischen Prozessor-Simulator zu bauen, der

- das für das Modul Betriebssysteme 1 benötigte Subset an x86-64 Prozessor-Instruktionen schrittweise ausführen
- den Zustand aller Register, den Speicherbus und festlegbare Speicherbereiche grafisch aufzeigen
- Änderungen daran grafisch ansprechend darstellen kann.

Der entwickelte Code soll Open-Source sein. Es sollen und dürfen soweit als möglich bestehende Open-Source-Systeme und -Bibliotheken verwendet werden, bspw: https://www.unicorn-engine.org/. Die Wahl der Tools und Programmiersprache obliegt den Studierenden. Die entwickelte Software soll möglichst direkt von den Studierenden einsatzbar sein; Stabilität und Benutzbarkeit der Software sowie deren Erweiterbarkeit sind wichtiger als der Funktionsumfang.

Die Studierenden sollen strukturiert und ingenieurmässig vorgehen und wissenschaftlich nachvollziehbare Untersuchungen durchführen und dokumentieren.

Termine

Die Studienarbeit beginnt am 15.9.2020. Abgabetermin ist der 18.12.2020.

Betreuung

Die Studienarbeit wird durch Prof. Stefan Richter betreut. Jeden Dienstag von 18:00 bis 19:00 findet eine Besprechung statt, in der die Studierenden den Fortschritt der Arbeit präsentieren.

Fragen und Angelegenheiten können ausserhalb dieses Termins auch per E-Mail erörtert werden.

Bewertung

Die Arbeit wird anhand der folgenden fünf gleichgewichteten Punkte bewertet:

1. Organisation, Durchführung (Projektplanung u. Nachführung Arbeit gemäss Projektplan, Selbständigkeit, Einsatz, Zusammenarbeit mit Auftraggeber, Betreuer)
2. Bericht (Inhalt des Projektschlussberichts, Gliederung, Darstellung, Sprache der gesamten Dokumentation)
3. Problemanalyse (Vorstudie, Literaturstudium, Anforderungsspezifikation, Anforderungsanalyse, Domainanalyse)
4. Lösungsentwurf (Lösungsvarianten und deren Beurteilung, Variantenentscheid, Konzept, Entwurf)
5. Realisierung und Test

Hinweise

Die folgende Seite bietet zahlreiche nützliche Informationen zum Schreiben einer wissenschaftlichen oder technischen Arbeit:

https://www.ifs.hsr.ch/index.php?id=13194&L=4metadata%2Foai_dc_1.dc

2.2 Abstract

Die Arbeitsweise des Prozessors ist Teil des Moduls "Betriebssysteme 1" aus dem ersten Semester des Informatik Studiums an der OST - Ostschweizer Fachhochschule. Für das bessere Verständnis ist ein grafischer Prozessor-Simulator hilfreich. Bereits existierende Simulatoren sind entweder didaktisch oder vom Abstraktionsniveau her unpassend für die Zielgruppe. Das Ziel dieses Projektes ist es, den Studierenden ein Hilfsmittel zur Verfügung zu stellen, das das Verständnis für die Ausführung von Instruktionen durch den Prozessor und damit zusammenhängende Details unterstützt. Um das Problem der Unverständlichkeit zu lösen, entwickelten wir ein stark abstrahiertes und dem Lerninhalt des Moduls entsprechendes Design. Die Benutzbarkeit und Verständlichkeit unseres Ansatzes wurden mithilfe von Usability Tests mit den betreffenden Studierenden sichergestellt. Da das Problem der bestehenden Ansätze am GUI und nicht an der Emulation des Prozessors liegt, verwenden wir die Unicorn Engine als Emulator. Damit erhalten wir nach der Ausführung einer Instruktion die aktuellen Daten für das Memory und die Register. Aus unserer Arbeit resultiert ein verifiziertes Design und ein MVP. Die Applikation kann zur Erklärung der einzelnen Komponenten eines Prozessors und seiner Funktionsweise eingesetzt werden. Unsere Implementierung bildet die Grundlage, um unser Entwurf in einer Folgearbeit vollständig umzusetzen und um die Möglichkeit, eigenen Code einzugeben, zu ergänzen.

2.3 Lay Summary

Der Prozessor ist ein zentraler Bestandteil des Computers. Er führt Befehle aus und schreibt zum Beispiel Daten in den Speicher oder rechnet damit. Im Bereich Informatik ist besonders interessant, wie der Prozessor einen Programmcode ausführt. An der OST - Ostschweizer Fachhochschule wird das den Informatik Studierenden im ersten Semester nähergebracht.

Um die Arbeitsweise eines Prozessors besser zu verstehen, ist ein grafischer Prozessor-Simulator hilfreich. Die bereits existierenden Simulatoren sind jedoch zu detailliert und decken sich nicht mit dem Stoff der betreffenden Vorlesung. Unser Ziel ist deshalb, ein Tool zu entwickeln, das den Studierenden hilft das komplexe Thema besser zu verstehen. Es soll ihnen zeigen, wie der Prozessor schrittweise Befehle eines Programms lädt, verarbeitet und ausführt.

Als erstes überlegten wir uns, wie die Funktionsweise eines Prozessors visuell dargestellt werden kann und entwickelten daraus ein anschauliches Design. Die Verständlichkeit unserer Idee testeten wir mit Studierenden aus dem ersten Semester. Mit den daraus resultierenden Erkenntnissen programmierten wir dann eine erste Version des Prozessor-Simulators.

Das Resultat der Arbeit ist ein funktionierender Designansatz und der Prozessor-Simulator als Programm. Dieser kann verwendet werden, um den Prozessor und seine Arbeitsweise den Studierenden schrittweise zu erklären. Der Simulator entspricht jedoch noch nicht vollständig dem entworfenen Design. Unser Resultat bildet die Grundlage, dass unser Entwurf in einer Folgearbeit vollständig umgesetzt und um die Möglichkeit, eigenen Code einzugeben, erweitert werden kann.

3 Ausgangslage

Die Arbeitsweise eines Prozessors wird an der OST - Ostschweizer Fachhochschule im Modul “Betriebssysteme 1” (Bsys1) unterrichtet. Dieses wird von Informatik Studierenden im ersten Semester besucht. Der Ablauf, wie der Prozessor nacheinander Instruktionen aus dem Hauptspeicher anfordert und verarbeitet, wird Prozessor-Zyklus genannt. Damit die Studierenden diesen besser verstehen, wird ein grafischer Prozessor-Simulator gefordert.

In dieser Arbeit kombinieren wir die Gebiete User Experience Design und Software Engineering und entwickeln einen Ansatz, wie ein grafischer Prozessor-Simulator verständlich dargestellt und implementiert werden kann. Ausserdem setzen wir eine minimale Version dieses Ansatzes um.

Es existieren bereits einige Simulatoren, die man online oder als Desktop Applikation nutzen kann. Diese Simulatoren sind jedoch meistens zu detailliert und eher für die Elektrotechnik geeignet. Ausserdem wurde jeweils viel Zeit in die Programmierung des Emulators investiert und weniger in das Design des Graphical User Interface (GUI). Dies kann man beispielsweise bei den folgenden Prozessor-Simulatoren sehen.

Johnny Simulator[1]: Open Source Simulator, der in FreePascal programmiert ist. Er läuft auf den Betriebssystemen Windows und Linux. [1]

Unicorn.js[2]: Open Source Simulator, der auf dem CPU-Emulator Framework Unicorn[3] basiert. Die Engine wurde für JavaScript umprogrammiert und läuft im Webbrowser. Die Darstellung des Memory funktioniert jedoch noch nicht. [2]

SIMAS: Wurde als Java-Applet programmiert und kann in jedem Java-fähigen Webbrowser ausgeführt werden. Der Simulator soll den Studierenden ein Verständnis für die Arbeitsweise des Prozessors vermitteln. Durch die Verwendung im Web ist der SIMAS Prozessor auch für Distance Learning einsetzbar. [3]

[1] Johnny Simulator : https://sourceforge.net/projects/johnnysimulator/
[2] Unicorn.js : https://alexaltea.github.io/unicorn.js/demo.html?arch=x86
[3] Unicorn Engine : https://www.unicorn-engine.org/

Simple 8-bit Assembler Simulator with Angular.js[4]: Dieser Simulator ist in JavaScript programmiert. Er bietet eine vereinfachte Assemblersprache, die auf NASM basiert. [4]

HTML5 CPU Simulator (8 bit binary implementation of Little Man Computer)[5]: Dieser Simulator wurde in JavaScript programmiert und bietet die Instruktionen des Little Man Computers[6] an. [5]

MIRAC The Minecraft Redstone Automatic Computer[7]: JavaScript Emulation des Minecraft Redstone Automatic Computer. Er benutzt die MIDAS Assembly Language. [6]

Diese Simulatoren haben alle gemeinsam, dass sie den Prozessor-Zyklus anders aufzeigen, als er im Modul Bsys1 vorgestellt wird. Ihre GUIs sind sehr unübersichtlich und stark mit Informationen überladen. Für die Kommunikation zwischen der CPU und dem Memory verwenden die einen eine Darstellung des Speicherbusses und die anderen Datenpakete, die sich durch Kanäle bewegen. Jedoch werden bei keinem Simulator ungenutzte Elemente ausgeblendet.

Die Animationen laufen viel zu schnell ab und es ist unklar, was die Animation initiiert hat. Die meisten Simulatoren färben nur die zuletzt verwendeten Elemente im Speicher ein. So sieht man nicht, warum sich das Element geändert hat oder wieso es sich bewegt, beispielsweise bei der Übertragung vom Speicher in die Register.

Zusätzlich erwecken die Designs einen altmodischen Eindruck. Beispielsweise werden einzelne Elemente oft nur in 2D dargestellt. Die verschiedenen Komponenten werden visuell nicht voneinander getrennt. Zudem wird in allen existierenden Simulatoren nicht visuell zwischen Bytes und Adressen unterschieden, was bei der ersten Betrachtung zu Verwirrung führt. Es werden unklare Abstraktionen gemacht, zum Beispiel wird oft der Assembly Code als separates Element dargestellt, sodass nicht klar ist, dass die Instruktionen in Wirklichkeit im Memory gespeichert werden. Zudem wird die Umwandlung von Assembly zu Maschinencode oftmals weggelassen und man sieht nicht, dass der Prozessor mit dem Maschinencode arbeitet.

[4] Simple 8-bit Assembler Simulator : http://schweigi.github.io/assembler-simulator/
[5] CPU Simulator : https://tools.withcode.uk/cpu/
[6] Little Man Computer : https://en.wikipedia.org/wiki/Little_man_computer
[7] Mirac-Emulator : https://draemm.li/various/mirac/

4 Anforderungen

Die bestehenden Simulatoren sind nicht für das Informatik Studium geeignet. Deshalb soll ein neuer Prozessor-Simulator für die Informatik Studierenden des Moduls Bsys1 aus dem ersten Semester erstellt werden. Er soll ihnen helfen den Prozessor-Zyklus besser zu verstehen. Um den Lernerfolg für die Informatik Studierenden zu gewährleisten, muss das GUI verständlicher und moderner gestaltet werden als bei den bestehenden Simulatoren.

4.1 Anforderungen an das GUI Design

Vom GUI wird erwartet, dass die Zustände aller Register und Speicherbereiche sinnvoll dargestellt sind und Zustandsänderungen durch eine ansprechende Animation verdeutlicht werden. Darüber hinaus wollen wir ein modernes Design, das visuell mit aktuellen Apps verglichen werden kann. Wir haben folgende Problemstellungen definiert, die durch das GUI Design gelöst werden müssen.

Für den Überblick über den Prozessor-Zyklus soll klar erkennbar sein, in welchem Schritt sich der Prozessor gerade befindet. Wir wollen den Studierenden helfen, zu erkennen welche Elemente in den Prozessor-Zyklus involviert sind und ob sie zum Prozessor oder zum Memory gehören. Beispiele dafür sind Adressen, Daten oder Register. Diese Elemente sollen sich klar voneinander unterscheiden. Jedoch müssen gleiche Elemente auch visuell gleich aussehen, so soll zum Beispiel ein Byte im Speicher gleich dargestellt werden wie ein Byte im Register.

Es soll ersichtlich werden, dass die CPU und der Speicher getrennt sind und nur über einen bestimmten Kanal kommunizieren können. Wir müssen zudem einen Weg finden, darzustellen, wie und warum Daten über diesen Kanal ausgetauscht werden.

Wir wollen zeigen, dass Instruktionen und Daten am selben Ort liegen, nämlich im Speicher. Die beiden Abschnitte des Speichers müssen jedoch unterscheidbar sein, z.B. in dem die Instruktionen in Assembly und die Daten in ASCII übersetzt werden.

Bei den bestehenden Simulatoren haben wir festgestellt, dass alle Daten und Elemente mit derselben visuellen Präsenz dargestellt werden, auch wenn sie im Moment nicht benötigt werden. Beispielsweise werden ungenutzte Bytes als “00” angezeigt und unterscheiden sich

nicht von benutzten Bytes. Dadurch füllt sich der ganze Speicher und somit ein grosser Teil des Bildschirms mit Nullen. Wir wollen einen Weg finden, Daten und Elemente, die aktuell nicht verwendet werden, so darzustellen, dass sie visuell nicht mit wichtigeren Teilen des Designs interferieren. Die Schwierigkeit besteht hier darin, dass die unterschiedliche Darstellung von gleichen Elementen nicht verwirren darf.

4.2 Anforderungen an den Prozessor-Simulator

In diesem Abschnitt werden wichtige Anforderungen an den Prozessor-Simulator kurz vorgestellt. Die genauen funktionalen und nicht-funktionalen Anforderungen werden im Anhang A beschrieben.

Eine vorgegebene Anforderung an den Prozessor-Simulator ist, dass er eine Teilmenge an x86_64 Prozessor-Instruktionen schrittweise ausführen kann. Diese Instruktionen werden in der Bsys1 Vorlesung verwendet. Der Code soll zudem Open-Source sein und es sollen soweit wie möglich bestehende Open-Source-Systeme und -Bibliotheken verwendet werden.

Da die Zielgruppe möglicherweise das erste Mal mit der Installation von Programmen oder Ausführung von Skripten in Kontakt kommt, sind komplizierte Installationsprozesse zu vermeiden. Damit die Benutzenden das Programm ohne Zeitverlust verwenden können, sollte es direkt per Doppelklick gestartet werden können. Das Programm kann somit ohne zusätzlichen Aufwand oder grosse Erklärungen während einer Übungsstunde eingesetzt werden.

5 Lösungskonzept

In diesem Kapitel erläutern wir die wichtigsten Architektur- und GUI-Designentscheidungen, die wir getroffen haben.

5.1 Deployment

Es gibt verschiedene Möglichkeiten eine Software auszuliefern. Eine Variante ist, ein Skript auszuliefern, das auf mehreren Betriebssystemen in einer separaten Laufzeitumgebung laufen kann. Die Laufzeitumgebung muss aber je nachdem zuerst installiert werden. Alternativ kann eine vorkompilierte und ausführbare Datei erstellt werden, die jeweils direkt auf einem Betriebssystem läuft.

5.1.1 Beschreibung der Möglichkeiten

Die Installation von Laufzeitumgebungen, die nicht mit dem Betriebssystem ausgeliefert werden, sind oft mühsam. Beispielsweise ist die Installation der Java Runtime Engine (JRE) für Unerfahrene kompliziert, da die Oracle Webseite unserer Meinung nach nicht sehr übersichtlich ist. Die Installation von Python ist eher für Fortgeschrittene, da es nur über die Konsole verfügbar ist. Je nach Einstellungen können die Skriptdateien auch nach der Installation der Laufzeitumgebung nicht per Doppelklick gestartet werden. Zum Beispiel werden bei Windows die ".jar" Dateien mit Doppelklick standardmässig als ".zip" Datei entpackt und nicht gestartet.

Webtechnologien decken die Anforderung, dass das Programm einfach zu installieren und zu starten sein soll, sehr gut ab. Jedes Desktop oder Mobile Betriebssystem ist mit einem Browser ausgestattet und hat somit die Laufzeitumgebung integriert. Zusätzlich können Webtechnologien in eine einzelne HTML Datei gebündelt werden, die per Doppelklick gestartet werden kann.

Bei einer vorkompilierten Datei nennt Oracle als Nachteil, dass die Auslieferung aufwändiger für die Benutzenden ist, als die Anwendung aus dem Web zu starten: "Unlike web deployment, the user experience is not about 'launch the application from the web'. It is more one of 'download, install, and run' process, in which the user might need to go through additional steps to get the application launched." [7]

5.1.2 Bewertung

Die Tabelle 1 listet die verschiedenen Möglichkeiten für die Auslieferung auf und bewertet sie mithilfe der im obigen Abschnitt erwähnten Kriterien und Überlegungen. Nach unseren Kriterien hat die Auslieferung als HTML Datei am besten abgeschnitten.

Punkte von 0 bis 10	**Gewichtung**	**Python (.py)**	**Java (.jar)**	**Programm (.exe)**	**Web (.html)**
Unsere Vorkenntnisse der Technologie	6	1	8	7	9
Auf einem Desktop ausführbar mit Doppelklick	5	0	5 (je nach Installation und OS)	10	10
Einfache Installation (inklusive der Runtime Umgebung)	4	3	5	10	10
Nutzbare Programmiersprachen	3	3	0	10	10
Multiplattform	2	10	10	0	10
Lauffähig auf mobilen Plattformen	1	0	0	0	10
Gewichtete Summe		47	113	162	204
Resultat (Punkte von 0.0 bis 10.0)		**2.2**	**5.4**	**7.7**	**9.7**

Tabelle 1 Bewertung der verschiedenen Auslieferungsarten mit gewichteten Kriterien

5.1.3 Entscheidung

Wir entschieden uns für die Webtechnologien, da sie unseren Anforderungen am besten entsprechen. Die Webtechnologien erlauben uns die Verwendung einer Vielzahl von Tools und Libraries. Zudem vereinfacht sich die ganze CI/CD Pipeline, da wir mit einer Pipeline gleich alle Architekturen abdecken können und nicht nur ein einzelnes Betriebssystem.

Für die Animationen verwenden wir CSS-Animationen, da diese für unsere Zwecke ausreichen und in allen Browsern nativ unterstützt sind. Wir arbeiten mit TypeScript, um die Typsicherheit zu gewährleisten. Die Applikation wird als einzelnes HTML File ausgeliefert. Das bedeutet, dass alle Ressourcen wie Styling, Bilder und Schriftarten in dasselbe HTML File gepackt werden und so lokal im Browser mit dem “file://” Protokoll geöffnet werden können. Dies hat zum Vorteil, dass die Applikation direkt via Doppelklick gestartet werden kann und keine komplizierte Infrastruktur vonnöten ist. Bei Bedarf besteht die Möglichkeit das fertige Webprojekt in eine ausführbare Elektron App zu konvertieren, um zusätzlich alle Vorteile von einem ausführbaren Programm zu erhalten [8].

5.2 Ansätze zum Bau des Prozessor-Simulator

Bei den in der Ausgangslage beschriebenen Prozessor-Simulatoren wurden unterschiedliche Ansätze verwendet. Entweder wurde ein bestehender CPU-Emulator, beispielsweise für x86 Emulation, verwendet und damit ein Simulator programmiert. Oder der ganze Simulator wurde von Grund auf selbst programmiert, meistens anhand einer eigenen vereinfachten Assembly Syntax. Ein weiterer Ansatz wäre, für einen bestehenden Prozessor-Simulator ein neues GUI zu implementieren. In den folgenden Abschnitten werden diese Möglichkeiten mit ihren Vor- und Nachteilen genauer beschrieben.

5.2.1 Verwendung eines bestehenden CPU-Emulators

Um einen bestehenden CPU-Emulator zu verwenden, muss zuerst ein geeigneter gefunden werden. Wir haben die folgenden CPU-Emulatoren berücksichtigt.

Libemu:
Library in C, die x86 Emulation anbietet. Wurde ursprünglich von Paul Baecher und Markus Koetter entwickelt. Ist jedoch fast nicht dokumentiert. [9]

LibCPU:
Open Source Library, die CPU-Architekturen emuliert und sich als CPU-Core für verschiedene Arten von Simulator Projekten anbietet. Ist unserer Meinung nach zu wenig dokumentiert. [10]

Unicorn Engine:

Bietet x86 Emulation an. Basiert auf dem Open Source Maschinen Emulator QEMU[8] und arbeitet mit Maschinencode. Bietet Native Support für Windows, Linux, Mac OS X. In C programmiert. C kann aber mithilfe von Emscripten[9] in WebAssembly kompiliert werden, was beim Simulator Unicorn.js[10] gemacht wurde. Somit kann die Unicorn Engine auch im Browser laufen. [11]

Da die Unicorn Engine gut dokumentiert ist, plattformunabhängig ist und bereits ein Web Beispiel mit Unicorn.js existiert, haben wir uns für diesen Emulator entschieden.

Nun stellt sich die Frage, was die Vor- und Nachteile von der Verwendung eines bestehenden Emulators beim Bau eines Prozessor-Simulators sind.

Vorteile:

- Korrekte und komplette Funktionalität: Ausführung Maschinencodes mit Registern, Memory etc.
- Zeitersparnis gegenüber selbst programmieren

Nachteile:

- Wir müssen Hooks nutzen, um an die Assembly Interpretation der Instruktion zu kommen, damit wir die Animation richtig darstellen können.
- Umwandlung Maschinencode zu Assembly und zurück muss über separates Tool gemacht werden (zum Beispiel Capstone.js[11] und Keystone.js[12])

5.2.2 Kompletter Simulator selbst schreiben

Der komplette Simulator kann ohne die Verwendung eines Emulators geschrieben werden. Dafür interpretiert man selbst den eingegebenen Assembly Code und programmiert dessen Auswirkung auf den Zustand des Speichers und Prozessors.

Vorteile:

- Zuordnung von Instruktion zur entsprechenden Animation ist einfach

Nachteile:

[8] QEMU: http://www.qemu.org/
[9] Emscripten: https://emscripten.org/
[10] Unicorn.js: https://alexaltea.github.io/unicorn.js/
[11] Capstone.js : https://alexaltea.github.io/capstone.js/
[12] Keystone.js : https://alexaltea.github.io/keystone.js/

- Für die Grösse des geplanten Instruktionssatzes sehr aufwändig und fehleranfällig
- Datenstrukturen und Funktionalität für alle Komponenten zum Beispiel Memory und Register müssen implementiert werden

5.2.3 GUI eines bestehenden Simulators durch eigenes ersetzen

Es gibt bereits viele Prozessor-Simulatoren, die gut funktionieren, jedoch ein schlechtes GUI haben. Statt die ganze Arbeit noch einmal zu machen, könnte man also direkt ein neues GUI für einen bestehenden Simulator programmieren.

Vorteile:

- Der Simulator funktioniert bereits
- Das Hauptproblem an den bestehenden Simulatoren ist das GUI und nicht der Simulator selbst.

Nachteil:

- Wir haben zwar Simulatoren gefunden, die im Web laufen. Diese nutzen aber alle das JavaScript Framework AngularJS[13]. Da wir jedoch keine Erfahrung mit diesem Framework haben und es sich dabei um ein sehr grosses und umfangreiches Tool handelt, eignet es sich nicht für unsere Arbeit. Wir ziehen es deshalb vor, mit dem uns bereits bekannten Framework Vue.js[14] zu arbeiten.

5.2.4 Entscheidung

Wir haben uns dazu entschieden, unseren Prozessor-Simulator mit der Unicorn Engine als Emulator zu bauen. So können wir die korrekte Abbildung der Daten im Memory und in den Registern ohne grossen Aufwand gewährleisten. Dies bedeutet, dass wir auch Instruktionen, die wir noch nicht animiert haben, trotzdem verarbeiten können. Wir haben uns entschieden Unicorn.js, den JavaScript Port der Unicorn Engine, zu verwenden, damit der Simulator im Browser laufen kann.

5.3 Softwarearchitektur

Aufgrund der oben genannten Entscheidungen ist die in Abbildung 1 abgebildete Architektur entstanden.

[13] AngularJS: https://angularjs.org/
[14] Vue.js: https://vuejs.org/

Abbildung 1 Architekturdiagramm Prozessor-Simulator

Wir verwenden das Framework Vue.js, da wir dieses schon aus einem früheren Projekt kennen. Es wird eine Vue Single Page Applikation eingesetzt. Ausserdem verwenden wir die Libraries Capstone.js und Keystone.js. Keystone.js wandelt den vom Benutzer eingegebenen Assembly Code in Maschinencode um. Dieser kann dann von der Unicorn Engine verarbeitet werden. Capstone.js wandelt den Maschinencode wieder in Assembly um, damit zum Beispiel die Übersetzung der Instruktionen im GUI angezeigt werden können.

5.3.1 Material Design

Für das GUI wird die Bibliothek Quasar[15] verwendet, die sich an die Guidelines von Material Design hält. Material ist eine Designsprache von Google und ein Leitfaden, wie Animationen und Elemente dargestellt werden müssen, damit sie für den Benutzenden verständlich und intuitiv sind. Ein wichtiges Element der Sprache sind die Schatten, die die Elemente werfen. [12]

Viele der bestehenden Simulatoren verzichten auf dreidimensionale Elemente, was laut der Nielsen Norman Group zu Usability Problemen führen kann. Zum Beispiel ist nicht mehr erkennbar, welche Elemente klickbar sind und welche nicht und die Effizienz des Benutzenden wird gesenkt. Die Nielsen Norman Group erwähnt, dass Material Design die Prioritäten richtig setzt. Durch die Verwendung von physikalischen Prinzipien erkennt man visuelle Hierarchien. [13]

Deshalb verwenden auch wir Material Design, um ein modernes GUI und gut verständliche Animationen zu bieten. Somit ist dann auch der Prozessor-Zyklus besser zu verstehen. Das GUI wird nach dem Desktop First Konzept gebaut. Die Applikation kann später um ein Responsive Design erweitert werden, um kleinere Screens wie Tablets zu bedienen.

5.3.2 Details zur Architektur

Der Emulator, die Services und die Vue Komponenten sind so getrennt, dass man sie austauschen kann, ohne dass viel an den anderen Komponenten geändert werden muss. Vue Komponenten ermöglichen durch HTML-Templating, wiederverwendbare und abgekapselte Web-Elemente zu erstellen. Die Vue Komponenten können auf Änderungen von übergebenen Daten reagieren und sie dynamisch auf der Webseite anzeigen. Ein Beispiel für eine Komponente, die wir mehrfach wiederverwenden ist die ByteVue Komponente, die den Inhalt eines Bytes darstellt. [14]

Weiter können die Komponenten verschachtelt werden, um komplexe Hierarchien abzubilden. Das oberste Element in unserer Komponenten Hierarchie ist die Simulator-Vue-Komponente. Sie beinhaltet unteranderem eine CPU und eine Memory Komponente, die wiederum verschachtelte Unterkomponenten enthalten. Die genaue Hierarchie unserer Vue Komponenten ist in der Abbildung 2 ersichtlich.

[15] Quasar: https://quasar.dev/

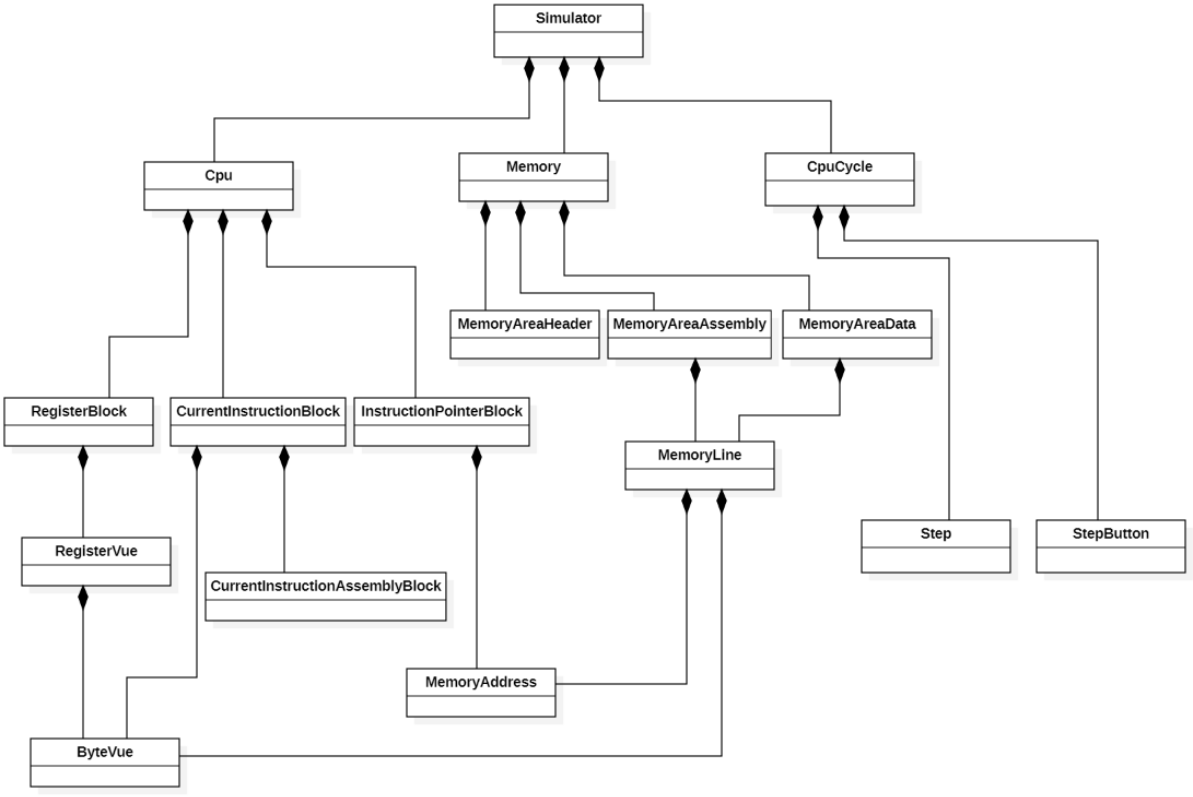


Abbildung 2 Hierarchie der Vue Komponenten im Prozessor-Simulator

Der Emulator wird über den “emulatorService” gesteuert. Der aktuelle Zustand des Prozessors wird im “State” Objekt gespeichert (siehe Abbildung 3). Die Data-Services sind dafür zuständig, die benötigten Daten beim Emulator zu holen und diese in das Objekt abzufüllen. Wenn der Prozessor-Simulator gestartet wird, erstellt die Simulator-Vue-Komponente eine Instanz des Controllers und startet den Emulator via “emulatorService”. Dann fragt sie den State des Prozessors beim Controller ab, der dann mithilfe der Data-Services das State Objekt mit Daten aus dem Emulator füllt und der Simulator-Vue-Komponente zurückgibt. Diese gibt das State Objekt an ihre Unterkomponenten weiter, die diese dann auf der Webseite abbilden.

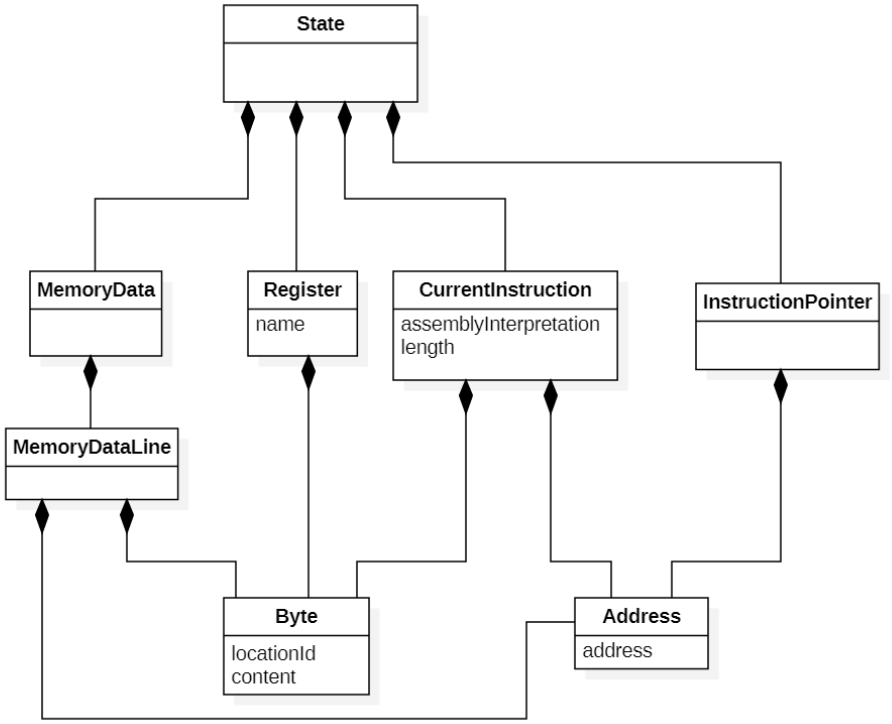


Abbildung 3 Aufbau des State Objekts

Wenn der nächste Schritt im Simulator ausgeführt werden soll, ruft die Simulator-Vue-Komponente den Controller auf, der je nach dem den Emulator die nächste Instruktion ausführen lässt und das State Objekt mit dem aktuellen Zustand des Prozessors aktualisiert. Ausserdem startet er die Animation der betroffenen Elemente im “animationService”. Sobald die Simulator-Vue-Komponente das neue State Objekt erhält, passen sich die Daten der Unterkomponenten automatisch an.

Für die Animationen müssen wir direkt auf die betroffenen Elemente zugreifen können. Dafür setzen wir in jeder Vue Komponente eine einzigartige HTML ID. Da die IDs global sind können wir direkt aus dem “animationService” das CSS-Styling der Elemente ändern, um die Animationen zu bauen und zu starten. Zusätzlich können wir bei Bedarf ganze Elemente samt Inhalt klonen und nach der Anpassung der ID wieder einfügen und später löschen. Dies ist zum Beispiel hilfreich, um die ByteVue Komponente bei einer “mov” Instruktion zu bewegen, ohne dass das Quell- und Ziel-Byte verändert wird.

5.4 Ansätze GUI Design

Das Design des GUI ist der Hauptbestandteil der Arbeit, da davon abhängt, ob die Studierenden den Prozessor-Zyklus besser verstehen werden. Da wir auch Animationen entwerfen

und testen wollten, erstellten wir den Designansatz in Microsoft PowerPoint. In den folgenden Abschnitten wird die Entwicklung unseres Designansatzes und unsere Überlegungen dargelegt.

5.4.1 Erste Überlegungen

Unser erster Ansatz wird in Abbildung 4 dargestellt. Er trennt die CPU und den Speicher und verbindet die beiden mit einem Kanal, durch den die Daten fliessen können. Der Speicherbus wurde bewusst weggelassen, um die Übersicht zu vereinfachen. Die Register und der Speicher haben jeweils ASCII-Interpretationen auf derselben Zeile.

Der Benutzende kann im Voraus seinen Programmcode eingeben, danach wird die Übersicht angezeigt und die Instruktionen in den Speicher geladen. Die Instruktionen werden nacheinander ausgeführt und pro Instruktion der Prozessor-Zyklus einmal durchlaufen. Um den Zyklus besser zu erklären, werden Infoboxen angezeigt, die den aktuellen Schritt beschreiben.

Es gibt drei Schritte: Instruktion holen, Instruktion ausführen und Befehlszeiger erhöhen. Danach beginnt der Zyklus wieder von vorne und die nächste Instruktion wird ausgeführt. Die Elemente, die in eine Animation involviert sind, leuchten kurz auf oder pulsieren. So kann man den Animationen besser folgen. Für die einzelnen Schritte werden jeweils nicht alle Komponenten gebraucht, deshalb grauen wir gewisse Teile aus.

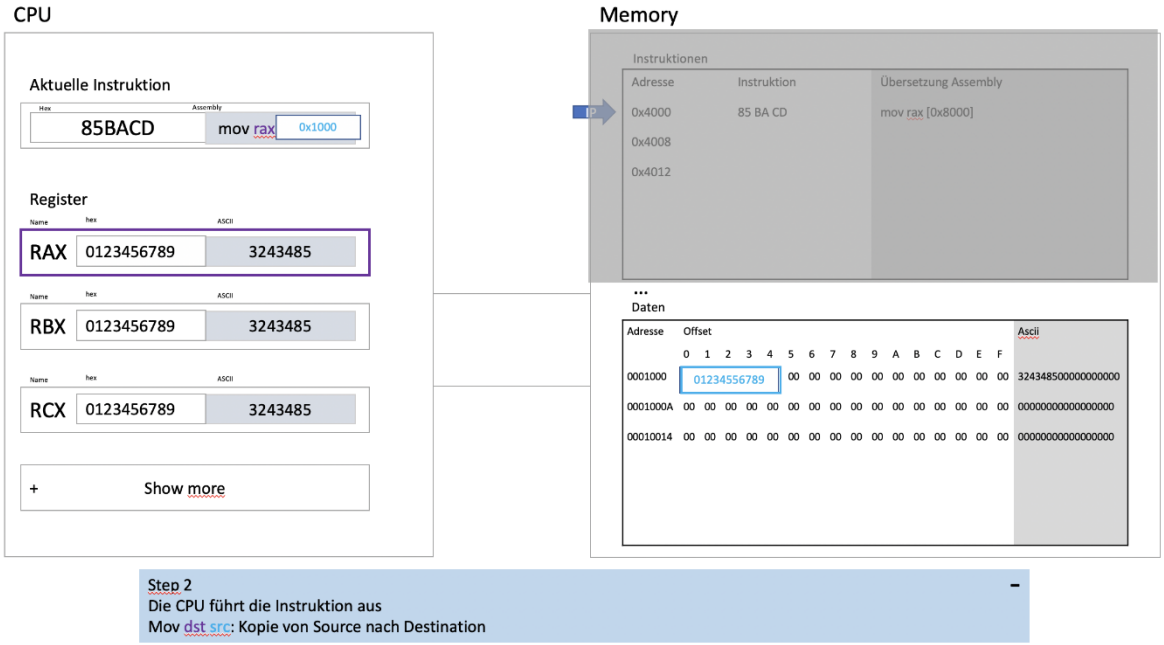


Abbildung 4 Erster Ansatz für das GUI Design

In diesem Ansatz sieht man nicht, woher der Befehlszeiger kommt. Ausserdem fehlt die visuelle Differenzierung von Adressen und Daten innerhalb des Speichers beispielsweise durch verschiedene Farben. Das Design dieses Ansatzes ähnelt dem Fenstersystem von Windows 10 und muss noch weiter modernisiert werden.

5.4.2 Ansatz zur Darstellung von Bytes und Adressen

Um Adressen und Daten besser zu unterscheiden, haben wir einen neuen Ansatz entwickelt, der in Abbildung 5 ersichtlich ist. Bei den Daten werden die Bytes getrennt voneinander dargestellt und bilden visuelle Einheiten. In der ASCII Interpretation wird ebenfalls jedes Zeichen einzeln abgebildet (siehe Abbildung 6).

Abbildung 5 Ansatz zur Darstellung Bytes und Adressen

Die Adresse wird in einer anderen Farbe dargestellt und aufgeteilt. Der erste Teil der Adresse, der für alle Bytes in einer Zeile gleich ist, wird an den Anfang der Zeile geschrieben. An die Bytes werden dann die dazugehörigen zweiten Teile der Adresse angehängt. Auf diese Weise kann die Adresse visuell zusammengesetzt werden und es ist für jedes Byte klar, an welcher Adresse es liegt.

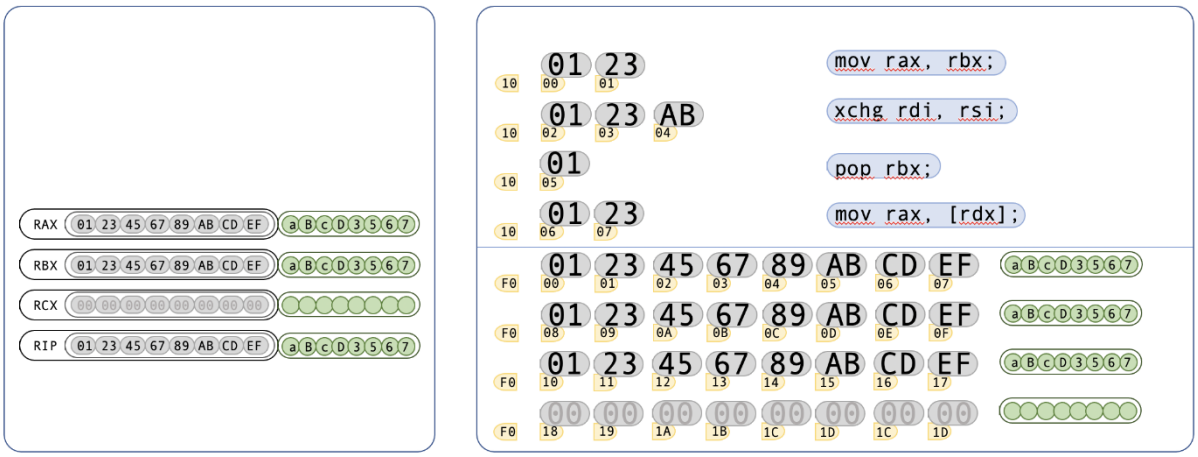


Abbildung 6 Design CPU und Memory mit Ansatz zur Darstellung Bytes und Adressen

In Abbildung 6 wird das Design der CPU und des Memory dargestellt, wenn der vorher beschriebene Ansatz angewendet wird. Die Idee mit den Adressen weist einige Mängel auf. Erstens benötigt der Speicher in dieser Darstellung zu viel Platz auf dem Bildschirm. Zweitens können in einer Zeile ungünstige Übergänge von Adressen, wie zum Beispiel von "0FFF" auf "0A00", auftreten. Man müsste in dieser Zeile bei der ersten Adresskennung eine Ziffer weglassen und bei den Adressen der Bytes eine hinzufügen. Das beansprucht jedoch noch mehr Platz. Dieses Problem liesse sich vermeiden, da die verwendeten Adressbereiche eingeschränkt werden können, aber das Platzproblem bleibt. Deshalb haben wir den Ansatz mit den Adressen verworfen.

5.4.3 Grundlayout

Als nächstes haben wir den ersten Ansatz mit dem Design der einzelnen Bytes kombiniert, das entstandene Resultat ist in Abbildung 7 abgebildet. Damit die ASCII- und Assembly-Übersetzungen nicht als Bestandteile des Speichers und der Register interpretiert werden, sind sie in einer darüberliegenden Ebene dargestellt. Sie können ausgeblendet werden, wenn sie nicht verwendet werden. Durch die Farbcodierung wird die ASCII-Interpretation von der Assembly-Interpretation unterschieden. Der Befehlszeiger wird als Pfeil dargestellt, der von der CPU auf die entsprechende Adresse im Speicher zeigt.

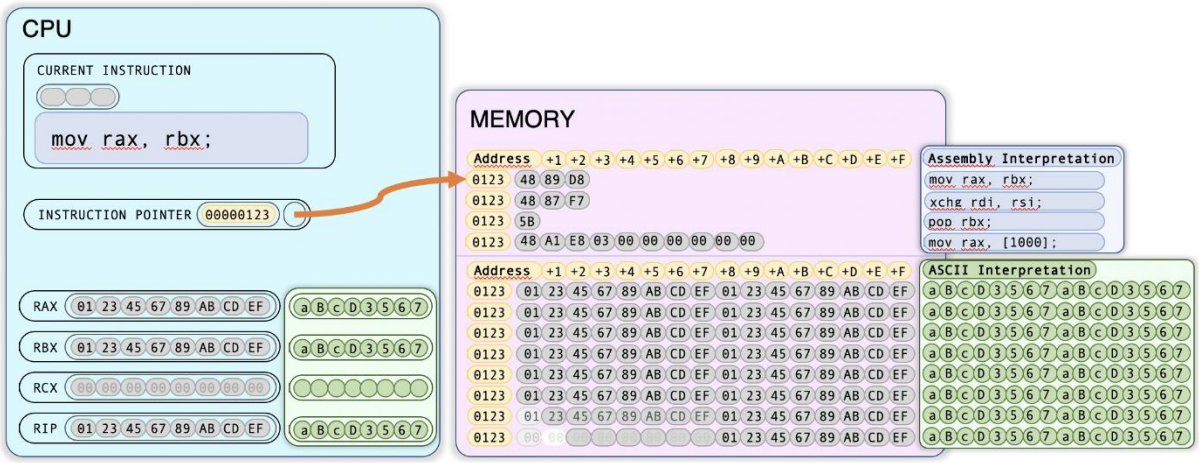


Abbildung 7 Grundlayout für den Prozessor-Simulator

5.4.4 Verbesserung Grundlayout

Nachdem wir das Grundlayout bestimmt hatten, versuchten wir, das Design durch die Verwendung von Schatten und modernen Farben zu modernisieren. Das entstandene Design wird in der Abbildung 8 dargestellt.

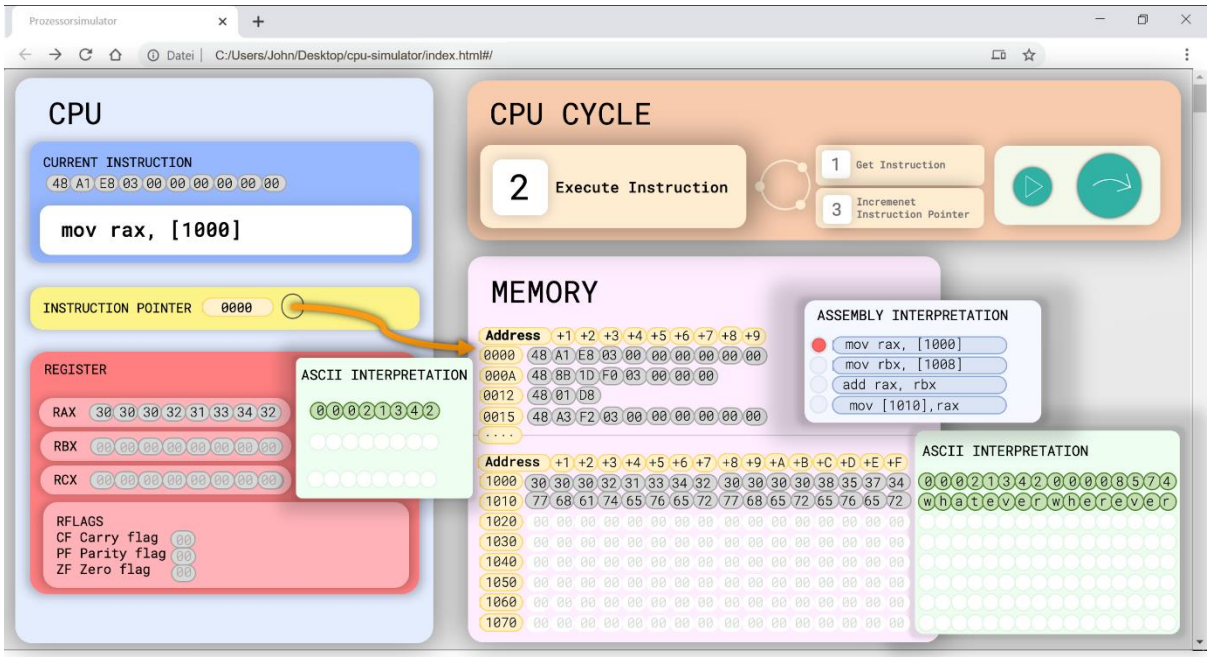


Abbildung 8 Design des Prozessor-Simulators

Damit der Benutzende zu Beginn nicht die ganze Übersicht auf einmal angezeigt bekommt, animierten wir noch ein Intro, bei dem der zuvor eingegebene Code ins Memory geladen wird und dann die anderen Elemente langsam nacheinander eingeblendet werden.

Um den aktuellen Schritt im Prozessor-Zyklus anzuzeigen, fügten wir neben den Erklärungstexten, die als Pop-up erscheinen, eine CPU Cycle Komponente hinzu. Der aktuelle Schritt wird dort jeweils in gross dargestellt. Beim Übergang zum nächsten Schritt werden alle Elemente in der CPU Cycle Komponente im Uhrzeigersinn rotiert. Wir haben zwei Steuerschaltflächen zum Durchlaufen der Zustände hinzugefügt. Eine Schaltfläche führt jeweils nur einen Schritt aus und die andere spielt das ganze Programm ab, ohne anzuhalten oder die Pop-up Fenster zu öffnen.

Wir suchten nach Ideen, um die visuelle Überlastung des Bildschirms zu reduzieren. Die unbenutzten Datenbytes im Speicher und in den Registern gestalteten wir so, dass sie visuell nicht von anderen, wichtigeren Elementen ablenken (siehe Abbildung 9). Sie ändern ihr Design, sobald sie benutzt werden und Daten enthalten. Darüber hinaus wird während einer Animation die Opazität von Elementen erhöht, die für diese spezifische Animation nicht wichtig sind. So kann sich der Benutzer besser auf die essenziellen Elemente konzentrieren. Ausserdem kürzten wir die Adressen zur Vereinfachung von 64 Bit auf 16 Bit.

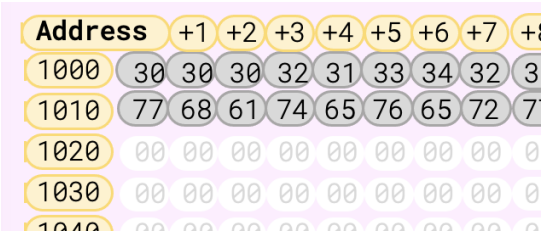


Abbildung 9 Darstellung unbenutzter Datenbytes

Um zu zeigen, wie und warum bei einer “mov”-Instruktion Daten verschoben werden, animierten wir zwei Pfeile, die von der Instruktion aus auf die Quelle und das Ziel zeigen. Die Daten schweben dann von der Quelle über die Anweisung in der CPU zu ihrem Ziel (siehe Abbildung 10). So wird deutlich gemacht, von wo die Animation ausgelöst wurde und wie sie abläuft.

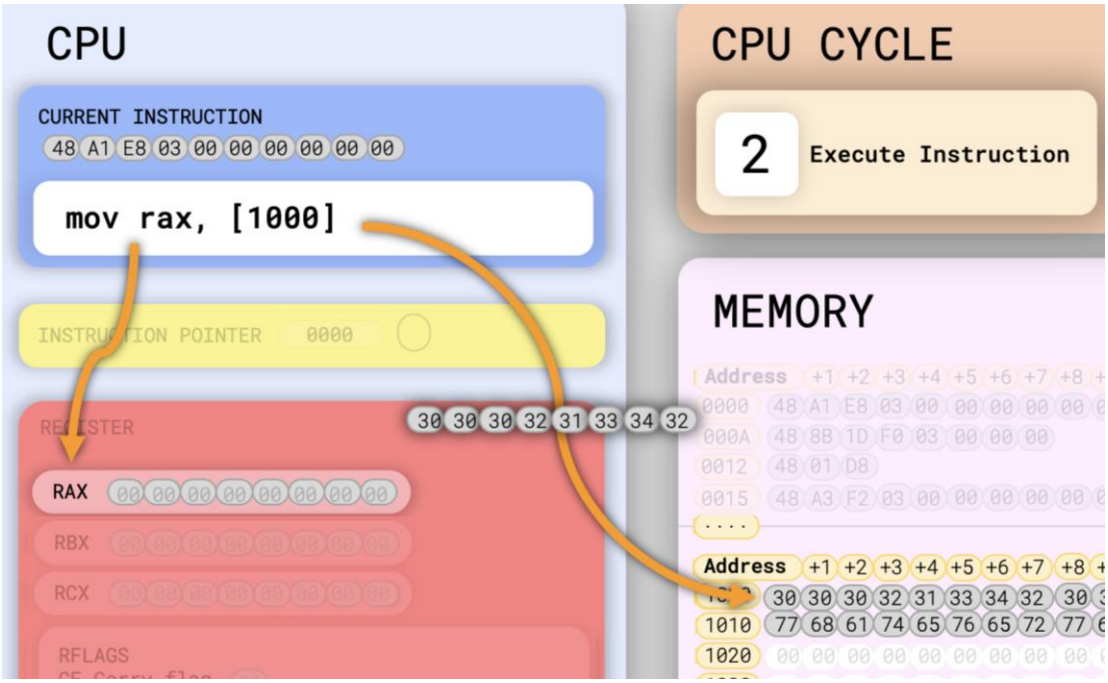


Abbildung 10 Animation "mov"-Instruktion

Beim letzten Schritt, die Erhöhung des Befehlszeigers, wollten wir zeigen, dass er um die Länge der aktuellen Instruktion erhöht wird. Dafür haben wir die Byte-Elemente der aktuellen Anweisung so animiert, dass sie sich in einen Zähler oberhalb des Befehlszeigers hinein bewegen (siehe Abbildung 11). Die resultierende Zahl wird dann zur Adresse hinzugezählt und der Pfeil verschiebt sich zur neuen Adresse.

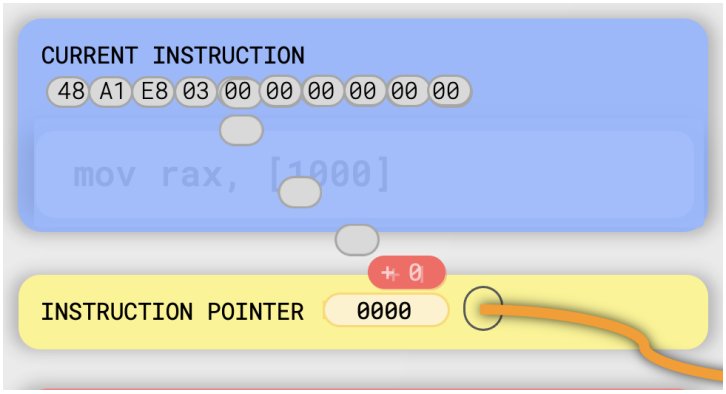


Abbildung 11 Animation Erhöhung des Befehlszeigers

Die Adress- und Byte-Elemente bedürfen noch einiger geringfügiger Anpassungen, um moderner auszusehen, was mit dem von uns verwendeten Tool aus Zeitgründen nicht möglich war. Auch die Pfeile, die für den Befehlszeiger und zur Animation der Datenbewegung verwendet werden, müssen separat neugestaltet werden, um ihren Zweck besser zu unterstützen.

6 Umsetzung

Um aus unseren Ideen ein Produkt zu entwickeln, sind wir folgendermassen vorgegangen. Als erstes testeten wir unseren GUI Designansatz mit unserer Zielgruppe. Mit den daraus resultierenden Erkenntnissen setzten wir unser Konzept für die Architektur um, so dass die Komponenten im GUI nun den aktuellen Prozessorzustand anzeigen. Anschliessend implementierten wir erste Animationen für die verschiedenen Schritte aus dem Prozessor-Zyklus. Um die Erfüllung der Anforderungen an die Architektur zu gewährleisten, wurden verschiedenen Analyse Tools verwendet. Die Testvorgänge und wichtige Implementierungsdetails werden in den folgenden Abschnitten erläutert.

6.1 Usability Tests für GUI Design

Der wichtigste Bestandteil der Arbeit ist das GUI Design, von dem der Lernerfolg der Studierenden abhängt. Um unseren Ansatz zu testen, wurde ein Usability Test mit sieben Bsys1 Studierenden mit verschiedenen Vorkenntnissen durchgeführt. Dafür wurde das im Microsoft PowerPoint animierte Design verwendet. Die Studierenden haben vor und nach der Präsentation des Designansatzes Fragen zum Prozessor-Zyklus beantwortet. Zudem wurden sie zu spezifischen Designelementen befragt. Der genaue Hergang und alle Rückmeldungen sind im Anhang B ersichtlich.

Unsere Tests zeigen, dass die Studierenden das Design mögen und sich bereits nach wenigen Sekunden darin zurechtfinden. Ausserdem beantworteten die Studierenden die Fragen in der zweiten Runde etwas besser.

Aus den Fragen, die zu den Designelementen gestellt wurden, ging hervor, dass die Bedeutung der Buttons unklar ist (siehe Abbildung 12). Das Problem ist, dass das Symbol des rechten Buttons, der den nächsten Schritt des Prozessor-Zyklus startet, als “Überspringen” interpretiert wird. Deshalb entschieden wir uns im Minimum Viable Product (MVP) nur einen Button zu verwenden.

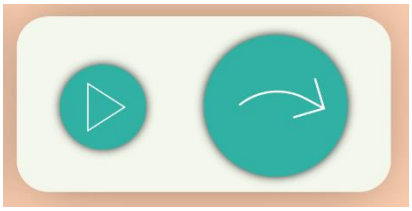


Abbildung 12 Buttons aus Designansatz

Für weitere Verwirrung sorgte die ASCII Interpretation, die wir neben dem Memory und den Registereinträgen (siehe Abbildung 13) dargestellt haben. Da die in Bsys1 verwendeten Instruktionen meistens mit Zahlen und nicht mit einzelnen Ziffern oder Buchstaben agieren, ist die ASCII Interpretation verwirrend und unnötig. Wir haben deshalb entschieden, die ASCII Interpretation wegzulassen.

Abbildung 13 ASCII Interpretation der Register im Designansatz

Zur Vereinfachung blendeten wir nicht benötigte Elemente aus. Dies trägt grundsätzlich zum besseren Verständnis bei. Jedoch führt die komplette Ausblendung des Instruction Pointer zu Unklarheiten, da er in der Realität konstant auf die aktuelle Instruktion zeigt. Der Instruction Pointer und die Pfeile für die Darstellung der Prozessor-Aktionen müssen deshalb noch einmal überarbeitet werden, so dass sie sich gut unterscheiden und der Instruction Pointer konstant eingeblendet werden kann. Die aktuelle Darstellung der Pfeile ist in Abbildung 14 und Abbildung 15 ersichtlich.

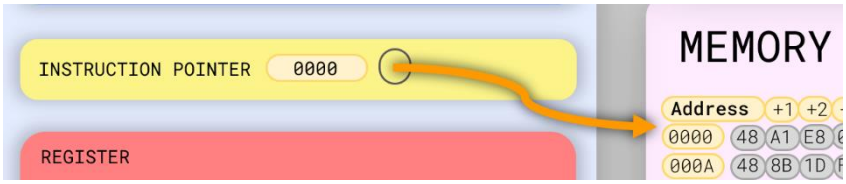


Abbildung 14 Instruction Pointer im Designansatz

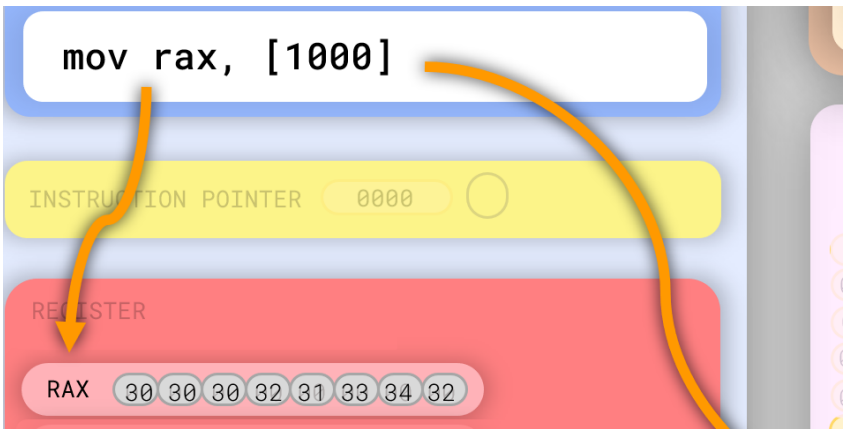


Abbildung 15 Pfeile für die Darstellung der Prozessor-Aktionen im Designansatz

6.2 Architektur

Nach dem Usability Test bauten wir das MVP mit einer vereinfachten Version des Designs. Als Übersicht über die Abhängigkeiten zwischen den Packages und Klassen im MVP wurde ein Abhängigkeitsgraph erstellt, der im Anhang C zu finden ist. Es handelt sich um einen gerichteten azyklischen Graphen.

6.2.1 Libraries

In unserer Architektur verwenden wird die Unicorn.js Library die auf dem Unicorn Framework basiert, das unter anderem von Nguyen Anh Quynh und Dang Hoang Vu als Open Source Projekt entwickelt wurde [2]. Ausserdem verwenden wir die Capstone.js Library, die auf dem Capstone Framework basiert, dass ebenfalls von Nguyen Anh Quynh entwickelt wurde [15]. Für die Einbindung der beiden Libraries haben wir in den Files “emulatorService.ts”[16] und “disassemblerService.ts”[17] Code von Alexandro Sánchez Bach übernommen und für die TypeScript Umgebung angepasst. Er hat beide Frameworks mithilfe von Emscripten für die Webentwicklung zugänglich gemacht.

Aus zeitlichen Gründen war es nicht möglich, den Code Editor zu programmieren. Dieser sollte zu Beginn des Programms angezeigt werden und dem Benutzenden ermöglichen sein eigenes Programm einzugeben und nachher zu simulieren. Deshalb wurde auch die angedachte Library Keystone.js nicht zum Endprodukt hinzugefügt. Diese hätte den Assembler Code in Maschinenbytes umgewandelt, die dann vom Emulator verarbeitet werden können. Wir testeten jedoch, ob sie mit den anderen verwendeten Libraries funktioniert und sie kann problemlos hinzugefügt werden.

Um den aktuellen Befehl zu erhalten, hätte man eine Hook Funktion innerhalb des Emulators verwenden müssen. Diese Einrichtung des Hooks ist aber sehr zeitaufwändig, weshalb wir uns für den folgenden Weg entschieden haben. Bevor alle Befehle in den Speicher des Emulators geschrieben werden, lassen wir diese vom Capstone-Disassembler analysieren. Dieser liefert die Instruktionen und die entsprechenden Adressen, an denen sie sich befinden. Während der Ausführung des Emulators suchen wir anhand der Adresse des Befehlszeigers den aktuellen Befehl in den von Capstone gelieferten Instruktionen.

[16] Original File Emulator : https://github.com/AlexAltea/unicorn.js/blob/master/src/unicorn-wrapper.js
[17] Original File Capstone : https://github.com/AlexAltea/capstone.js/blob/master/src/capstone-wrapper.js

Dieser Weg bringt jedoch einen Nachteil mit sich. Das Nachschlagen funktioniert in unserem Kontext sehr gut, würde aber fehlschlagen, wenn die Befehle während der Ausführung des Emulators im Speicher verändert oder erweitert werden. Falls dies bei einer Erweiterung umgesetzt wird, muss für einen stabileren Ansatz auf das Hook-System umgestiegen werden.

6.2.2 Interfaces vs. Klassen

In TypeScript werden Interfaces und Klassen anders verwendet, als man sich das von der Objektorientierten Programmierung gewöhnt ist. Beide Konstrukte definieren wie ein Objekt aussieht und werden in TypeScript verwendet, um Variablen zu typisieren. Im Gegensatz zu Klassen hat das Interface keinen Konstruktor und definiert nur, welche Namen und Typen die Attribute des Objekts haben sollen. Bei der Erstellung des Objekts wird also nur die Übereinstimmung der Typen getestet und keine weitere Validierung durchgeführt. Bei Klassen kann man im Konstruktor neben den Typen auch die übergebenen Werte validieren. [16]

Da es in unserem Fall keine Rolle spielt, verwenden wir Interfaces für die grundlegenden Datenelemente, aus denen sich das “State” Objekt zusammensetzt. Um trotzdem eine Validierung bei der Erstellung der Objekte zu machen, haben wir Konvertierungsfunktionen geschrieben. Diese nehmen die Eingabe entgegen, prüfen sie, konvertieren sie in das richtige Format und erstellen das Objekt, das zurückgegeben wird. Beispielsweise wird für die Erstellung eines Bytes eine Nummer als Adresse mitgegeben, die dann validiert und in eine vierstellige Hex Zahl als String umgewandelt wird.

6.3 Qualitätssicherung

In diesem Abschnitt werden die Tests und Analysen beschrieben, die wir für unser Projekt verwendet haben. Die erreichten Endergebnisse unseres Projektes, werden im Kapitel Ergebnisse beschrieben.

6.3.1 Unit- und Systemtests

Um die Services zu testen wurden Unittests für die einzelnen Funktionen mithilfe der Chai Assertion Library[18] geschrieben. Um die StepController Klasse zu testen wird der Emulator gestartet und eine Instruktion schrittweise ausgeführt und die Änderungen im State validiert.

[18] Chai Assertion Library: https://www.chaijs.com/

Der AnimationService wird nicht mit Unit Tests getestet. Für diesen würde sich ein Regression Test beispielsweise mithilfe von Selenium[19] eignen. Da im MVP nur wenige Animationen angedacht sind, lohnt sich der Aufwand unserer Meinung nach nicht. Die Animationen werden jedoch in den Systemtests überprüft. Diese werden manuell mindestens vor und nach einem Merge Request durchgeführt. Im Anhang G befindet sich das Protokoll des Systemtests, den wir mit dem MVP durchgeführt haben.

6.3.2 Performance Test

Um die RAM- und CPU-Auslastung zu analysieren, nehmen wir jeweils den Safari- und Chrome-Prozess, der für unseren Browser Tab verantwortlich ist und messen die Auslastung im Prozessmanager des Betriebssystems.

Im Safari können wir die Framerate der Animationen untersuchen, da er diese direkt anzeigt. Ein weiterer wichtiger Aspekt ist die Seitenladezeit. Dieser Wert kann mit dem Lighthouse-Benchmark[20] im Chrome-Browser ermittelt werden, der das Laden und Öffnen einer Website simuliert. Wir erwarten jedoch, dass unsere Applikation schlecht klassifiziert wird, da wir keine normale Webseite entwickeln, sondern alle Ressourcen auf einmal aus dem HTML File laden.

6.3.3 Code Analyse

Um die Codequalität sicherzustellen, wurden neben gegenseitigen Code Reviews auch Analysen mit den Tools DeepScan[21] und Better Code Hub[22] durchgeführt. Better Code Hub hat unseren Code nach verschiedenen Kriterien überprüft, wie zum Beispiel kurze und simple Code Units. Da Better Code Hub Vue Komponenten ignoriert, haben wir zusätzlich DeepScan verwendet, das auf JavaScript spezialisiert ist. So können wir auch die Anzahl Codezeilen bestimmen. Für die statische Codeanalyse haben wir ESLint[23] verwendet, das bei jedem Build den Code überprüft.

[19] Selenium: https://www.selenium.dev/
[20] Lighthouse: https://developers.google.com/web/tools/lighthouse
[21] DeepScan: https://deepscan.io/
[22] Better Code Hub: https://bettercodehub.com/
[23] ESLint: https://eslint.org/

7 Ergebnisse

Das Ergebnis dieser Arbeit besteht aus einem verifizierten in Microsoft PowerPoint animierten Designvorschlag, dem Code Repository und dem per Doppelklick startbaren Prozessor-Simulator als HTML File. Zudem wurde eine Anleitung erstellt (siehe Anhang D).

7.1 Animationen

Auch wenn das visuelle Ergebnis der Animationen innerhalb des Browsers mit blossem Auge gut aussieht, gibt es einige Verbesserungsmöglichkeiten (siehe Resultate in Anhang E). Sowohl Chrome als auch Safari zeigen an, dass die Leistung bei laufenden Animationen gut ist und meistens bei über 30 Bilder pro Sekunde liegt. Es gibt jedoch einen erheblichen Rückgang der Framerate, wenn die zu animierenden Elemente ins Document Object Model (DOM) hinzugefügt und entfernt werden.

Weitere Untersuchungen haben ergeben, dass die Stelle im DOM, an der die Elemente eingefügt werden, suboptimal für die Performance ist. Das Einfügen und Entfernen an dieser Stelle führt zu einer vollständigen Neuberechnung des gesamten Layouts. Dies könnte umgangen werden, indem die Position der Einfügung an eine günstigere Stelle im DOM verschoben würde.

Darüber hinaus könnte sich die Leistung verbessern, wenn wir die zu animierenden Elemente nicht klonen und einfügen, sondern sie auf Vorrat ausserhalb des sichtbaren Bereichs abbilden und nur den Inhalt anpassen. So kann die Layout-Engine die Animationen im Voraus berechnen. Trotz der Verbesserungsmöglichkeiten wurde unser Ziel von 30 Bildern pro Sekunde erreicht.

7.2 Architektur

Um zu überprüfen, ob unsere Architektur die Anforderungen erfüllt, führten wir diverse Tests und Analysen durch. In diesem Abschnitt werden die Resultate und unsere Interpretation aufgeführt. Die genauen Ergebnisse der Analyse sind im Anhang E ersichtlich.

Die Download-Grösse der Applikation beträgt 6 MB und liegt innerhalb der 15 MB-Grenze, die wir in unseren nicht-funktionalen Anforderungen festgelegt haben. Es gibt somit immer

noch mehr als genug Spielraum, um Bibliotheken wie Keystone hinzuzufügen, die in der Messung noch nicht enthalten sind.

Die CPU-Auslastung überschreitet während einer Animation unser Limit von 20 %. In Chrome beträgt die CPU-Auslastung maximal 32 % und in Safari 34 %. Da zwischen den Animationen die CPU-Nutzung unter 0.1 % fällt, nehmen wir an, dass der Grund für die Überschreitungen des Limits ebenfalls die unvorteilhaft programmierten Animationen sind. Deshalb könnten sich die im Abschnitt 7.1 erläuterten Änderungsvorschläge auch positiv auf diesen Wert auswirken. Bei der RAM-Ausnutzung unserer Applikation meldet der Safari Prozess maximal 480 MB und der Chrome Prozess 360 MB, was beides innerhalb der 500 MB Grenze liegt.

Um die Seitenladezeit zu messen wurde ein Test im Lighthouse-Benchmark im Chrome Browser gemacht. Wie zu erwarten war, steht unsere Webseite bei diesem Benchmark im Vergleich zu anderen sehr schlecht da. Wir erläutern in den folgenden Abschnitten die Ergebnisse und ihre Angemessenheit für unseren Anwendungsfall. Im Anhang E befindet sich der genaue Bericht von Lighthouse.

Der Test zeigt eine Seitenladezeit von fünf Sekunden, was für normale Webseiten sehr schlecht wäre. Bei unseren Architekturentscheidungen, alle Ressourcen der Webseite in dieselbe Datei zu packen, ist diese Ladezeit jedoch unvermeidlich. In unserem Anwendungsfall liegt die Ladezeit innerhalb unserer Erwartungen von zehn Sekunden. Vergleicht man diese Download- und Ausführungsgeschwindigkeit mit klassischen ausführbaren Dateien, so ist unser Ansatz um Grössenordnungen schneller. Im Hinblick auf die anderen negativen Beurteilungen, die der Lighthouse-Test liefert, stellen wir fest, dass die meisten von ihnen durch die enorme Dateigrösse unserer Anwendung erklärt werden können.

Eine weitere interessante Beobachtung ist die massive DOM-Grösse von fast 14'000 Elementen. Die meisten dieser Elemente lassen sich auf die Darstellung des Memory zurückführen, das rund 4000 Byte Elemente beinhaltet. Der Grund für die Integration so vieler Speicherelemente liegt darin, dass der Emulator eine Begrenzung hat, die es nicht erlaubt, weniger Speicher zu mappen. Man könnte jedoch, sobald eigener Code eingegeben werden kann, den Speicherbereich für den Benutzer einschränken. Dies würde die Komplexität des DOM und somit die gesamte Darstellungs- und Animationsleistung erheblich verbessern.

7.3 Codeanalyse

Bei DeepScan erreichten wir den höchsten Grad “Good” und haben genau eine Schwachstelle mit einer geringen Auswirkung. Diese Schwachstelle kann nicht behoben werden, da sonst der Linter fehlschlägt. Bei Better Code Hub erfüllen wir zehn von zehn Kriterien. Die maximale Funktionslänge, die gemessen wurde, beträgt 21 Zeilen und die maximale gefundene Parameteranzahl ist drei. Im Anhang F befinden sich die genauen Auswertungen der beiden Analysetools.

Die beiden Tools halfen während des ganzen Projekts Probleme zu erkennen und die Qualität zu verbessern. Sie beweisen jedoch nicht, dass die Qualität des Codes wirklich gut ist. Zum Beispiel können sie nicht erkennen, ob Kommentare sinnvoll sind oder nicht.

7.4 GUI Design

Der Designvorschlag mit dem die Usabilitiytests gemacht wurden, erfüllt fast komplett die Usecases, die definiert wurden. Die einzelnen Schritte der Ausführung wurden graphisch ansprechend und verständlich dargestellt, was mithilfe der Usabilitytests belegt wurde. Einzig die Steuerung der Ausführung muss bei einer Weiterentwicklung überdacht werden.

Wir haben uns das Ziel gesetzt, dass vier von fünf Erstsemestrige den Prozessor-Zyklus nach der Verwendung der Applikation für 30 Minuten verstehen. Nach unserer Fünfminütigen Demo im Usability Test haben bereits Fünf von sieben Erstsemestrigen verstanden, wie der Prozessor-Zyklus abläuft. Wir gehen deshalb davon aus, dass wir mit unserem Designansatz das gesetzte Ziel erreichen würden, wenn die Studierenden 30 Minuten Zeit hätten und ihren eigenen Code eingeben könnten. Durch unseren einfachen Installationsprozess und mithilfe eines geeigneten Steuerungssystems müssten in Zukunft alle in der Lage sein, die Applikation nach fünf Minuten zu bedienen.

Leider entspricht der entstandene MVP noch nicht ganz dem getesteten Design. Dieses wurde bewusst zu umfangreich erstellt, damit die Grundidee getestet werden kann und die Usability Tests in einer Folgearbeit nicht wiederholt werden müssen.

Im MVP wird der Code eines vorgegebenen Programms ausgeführt und man sieht die Zustandsveränderungen der Komponenten im GUI. Das Laden der Instruktion und die Erhöhung des Instruction Pointer sind bereits animiert. Bei der Ausführung der Instruktion ist die

‘mov’ Instruktion zwischen Registern und Memory animiert. Die Animation ist noch nicht ganz verständlich, da die Pfeile nicht implementiert wurden. Diese sollten anzeigen, von wo die Daten geholt werden und wohin sie sich bewegen. Weitere Instruktionen sind noch nicht animiert; die Daten in den Komponenten werden jedoch entsprechend angepasst. Um eigenen Code einzugeben, muss dieser im Moment selbst in Maschinenbytes übersetzt und dann manuell im Source Code eingefügt werden. Des Weiteren muss die Ausführungssteuerung verbessert und Beschreibungstexte ergänzt werden.

Durch das schlichte Design, die stark abstrahierte Darstellung der Komponenten und das angemessene Animationstempo löst unser Prozessor-Simulator die meisten Probleme, die die bestehenden Simulatoren ungeeignet für die Bsys1 Studierenden machen. Ausserdem wurde von den Test-Teilnehmenden mehrfach erwähnt, dass man sich einen solchen Simulator wünsche, was die Weiterentwicklung unserer Arbeit berechtigt.

8 Fazit und Ausblick

Der Prozessor-Simulator kann im aktuellen Zustand verwendet werden, um die Schritte des Prozessor-Zyklus den Bsys1 Studierenden näher zu bringen. Man kann damit erklären, welche Elemente daran beteiligt sind und ob diese zum Prozessor, oder zum Memory gehören. Unserer MVP trägt somit zu einem besseren Verständnis bei.

Bis auf die CPU-Auslastung wurden alle nicht-funktionalen Anforderungen an die Architektur erfüllt. Die erreichte Downloadgrösse und Startzeit fördern einen einfachen und schnellen Startprozess. Die Animationen scheinen für die Benutzenden flüssig, müssen jedoch für die Erhöhung der Performance noch verbessert werden.

Um einen besseren Lerneffekt zu erreichen, muss es für den Benutzenden möglich sein, seinen eigenen Code einzugeben. Wie in unserem Lösungskonzept angedacht, kann man den Assembly Code mithilfe der Keystone.js Library in Maschinencode umwandeln und dem Simulator übergeben. Des Weiteren müssen noch mehr Instruktionen animiert werden und die verschiedenen CPU-Flags angezeigt werden. Zum Beispiel wäre ein Wunsch der Bsys1 Studierenden, dass die Jump Instruktion animiert wird, um das Konzept der Rekursion besser zu verstehen.

Ein wichtiger Zwischenschritt aus dem Prozessor-Zyklus wird in unserem Design vereinfacht dargestellt. Der Prozessor liest nicht alle Bytes einer Instruktion auf einmal, sondern interpretiert diese einzeln. Dies könnte man in einer späteren Version ebenfalls noch ergänzen.

Wir haben hiermit gezeigt, dass ein Prozessor-Simulator im von uns gestalteten Stil von den Studierenden gewünscht wird und sie beim Lernen unterstützt. Der resultierende Designansatz und MVP bilden eine gute Grundlage für eine Weiterentwicklung in einer Folgearbeit.

9 Literaturverzeichnis

[1] P. Dauscher, «Johnny Simulator,» 10 Juli 2012. [Online]. Available: https://sourceforge.net/projects/johnnysimulator/. [Zugriff am 28 September 2020].

[2] A. S. Bach, «Unicorn.js,» 19 April 2017. [Online]. Available: https://alexaltea.github.io/unicorn.js/. [Zugriff am 14 September 2020].

[3] N. Jovanovic, D. Markovic, D. Živković und R. Popovic, «SIMAS: A web-based computer system simulator,» *International Journal of Engineering Education,* Bd. 26, pp. 930-937, Januar 2010.

[4] M. Schweighauser, «Simple 8-bit Assembler Simulator,» 01 Februar 2015. [Online]. Available: https://github.com/Schweigi/assembler-simulator. [Zugriff am 28 September 2020].

[5] P. Dring, «CPU Simulator,» 25 September 2017. [Online]. Available: https://tools.withcode.uk/cpu/. [Zugriff am 28 September 2020].

[6] R. Rytz, «MIRAC-Emulator,» 21 Juli 2014. [Online]. Available: https://draemm.li/various/mirac/. [Zugriff am 28 Oktober 2020].

[7] Oracle, «Self-Contained Application Packaging,» März 2015. [Online]. Available: https://docs.oracle.com/javase/8/docs/technotes/guides/deploy/self-contained-packaging.html. [Zugriff am 03 Oktober 2020].

[8] OpenJS Foundation, «Electron,» 15 September 2020. [Online]. Available: https://www.electronjs.org/. [Zugriff am 28 September 2020].

[9] A. Dell'Aera und A. Skovoroda, «libemu,» 05 Dezember 2019. [Online]. Available: https://github.com/buffer/libemu. [Zugriff am 28 September 2020].

[10] G. Guida und viele mehr, «libcpu,» 09 Januar 2014. [Online]. Available: https://github.com/libcpu/libcpu. [Zugriff am 28 September 2020].

[11] N. A. Quynh, D. H. Vu und viele mehr, «Unicorn,» 21 September 2020. [Online]. Available: https://www.unicorn-engine.org/. [Zugriff am 28 September 2020].

[12] Google, «Introduction,» 08 2020. [Online]. Available: https://material.io/design. [Zugriff am 02 Oktober 2020].

[13] K. Moran, «Flat Design: Its Origins, Its Problems, and Why Flat 2.0 Is Better for Users,» 27 September 2015. [Online]. Available: https://www.nngroup.com/articles/flat-design/. [Zugriff am 28 Oktober 2020].

[14] E. You, «Components Basic,» 27 April 2016. [Online]. Available: https://vuejs.org/v2/guide/components.html. [Zugriff am 15 Oktober 2020].

[15] A. S. Bach, «Capstone.js,» 01 Februar 2017. [Online]. Available: https://alexaltea.github.io/capstone.js/. [Zugriff am 14 September 2020].

[16] T. Motto, «Classes vs Interfaces in TypeScript,» 22 September 2018. [Online]. Available: https://ultimatecourses.com/blog/classes-vs-interfaces-in-typescript. [Zugriff am 28 Oktober 2020].

Anhang A Anforderungen

Anforderungen

Persona

Andreas (21):
Andreas hat nach seiner Informatiklehre frisch an der OST begonnen, Informatik zu studieren. Er besucht zusammen mit den anderen Erstsemestrigen das Basispflichtmodul Bsys1. In der zweiten Woche wird er in der Vorlesung in das Prozessor Modell eingeführt. Er freut sich auf dieses Thema, da er gerne mehr über Hardwarenahe Programmierung erfahren möchte.

Abbildung A 1 Andreas (Bild wurde erstellt auf https://thispersondoesnotexist.com/)

Use Cases

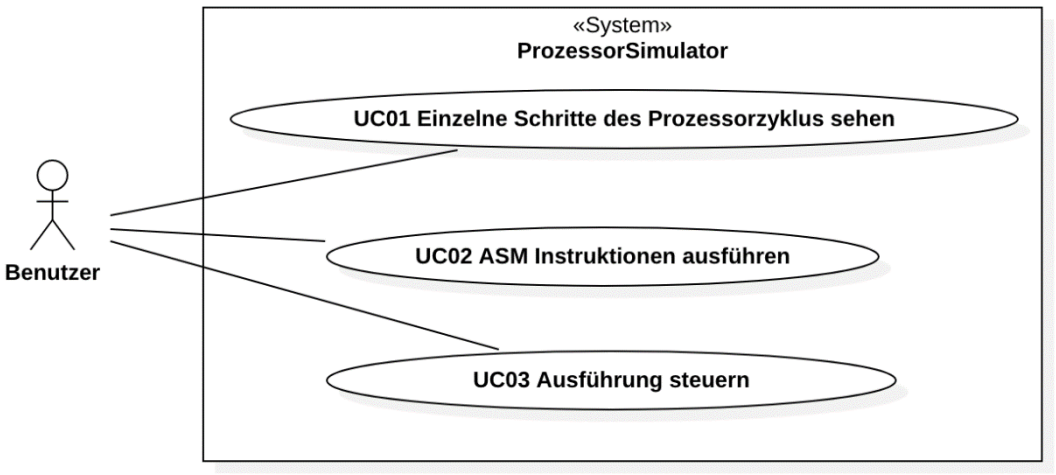


Abbildung A 2 Use Case Diagramm

UC01 Einzelne Schritte des Prozessor-Zyklus sehen

Als Benutzer möchte ich die einzelnen Schritte des Prozessor-Zyklus sehen, um besser zu verstehen, was im Prozessor genau passiert.

Id	**Titel**
01	Aktuelle Instruktion sehen
02	Veränderungen im Memory sehen
03	Veränderungen im Register sehen
04	Sehen, in welchem Schritt man sich befindet
05	Zustandsübergänge graphisch ansprechend sehen

UC02 ASM Instruktionen ausführen

Als Benutzer möchte ich mein eigenes Programm eingeben und ausführen, um die verschiedenen Instruktionen besser zu verstehen.

Id	**Titel**
06	Simulation starten
07	Demo Instruktion starten
08	ASM Instruktionen eingeben

UC03 Ausführung steuern

Als Benutzer möchte ich die Ausführung steuern können, um in meinem Tempo die Animationen zu sehen.

Id	**Titel**
09	Ausführung starten und stoppen
10	In Schritten vorwärts gehen
11	Animationstempo einstellen

Nicht-funktionale Anforderungen

Maintainability

Nr	Anforderung	Beschreibung
01	Testability	Das Testing kann auf Entwickler- und Benutzerebene durchgeführt werden.
02	Testability	Alle Test Cases müssen erfolgreich sein und reviewed werden, bevor die Code Änderungen akzeptiert werden.
03	Modifiability	Die Applikation kann angepasst werden, ohne das produktive System zu beeinflussen
04	Modularity	Die Applikationskomponenten werden modular aufgebaut, um einen schnellen und einfachen Austausch zu ermöglichen, damit neue Technologien rasch eingebaut werden können.

Security

Nr	Anforderung	Beschreibung
05	Integrity	Wenn die Applikation von der originalen Plattform heruntergeladen wird, kann man sicher sein, dass sie nicht von Dritten modifiziert wurde.
06	Confidentiality	Es werden keine persönlichen Daten erfasst oder versendet.

Performance

Nr	Anforderung	Beschreibung
07	Resource Behaviour	Die Applikation darf in ihrem Chrome Prozess eine Gesamt-RAM-Nutzung von 500 MB nicht überschreiten. Die Applikation darf in ihrem Chrome Prozess eine Gesamt-CPU-Nutzung von 20% nicht überschreiten.
08	Resource Behaviour	Die Applikation hat eine Downloadgrösse von maximal 15 MB
09	Time Behaviour	Die Applikation startet innerhalb von 10 Sekunden nach dem Download und nimmt Benutzereingaben entgegen.
10	Time Behaviour	Die Animationen sind flüssig und stocken nicht. Die Animationen laufen mindestens mit 30 Frames pro Sekunde.

Usability

Nr	Anforderung	Beschreibung
11	Availability	Die Applikation ist offline lauffähig
12	Learnability	Informatik Erstsemestrige sollen die Applikation nach maximal 5 Minuten verstehen und verwenden können.
13	Learnability	4 von 5 Informatik Erstsemestrige sollen nach dem Benutzen der Applikation für 30 Minuten verstehen, wie ein Prozessor-Zyklus abläuft

Portability

Nr	Anforderung	Beschreibung
14	Installability	Das Produkt kann ohne Installation auf Windows in der neusten Version von Chrome genutzt werden.

Anhang B Dokumentation Usability Tests

Dokumentation Usability Test

Datum: 4. November 2020

Mithilfe dieses Usability Tests wollen wir untersuchen, wo genau die Verständnisprobleme beim Prozessor-Zyklus liegen und ob unser Design Abhilfe schafft. Dazu wurden Studierende des Moduls “Betriebssysteme 1” (Bsys1) befragt und unser aktuelles Design mit ihnen getestet. Wir haben angenommen, dass bei den anderen Simulatoren vor allem die Überladung an Informationen und das altmodische Design das Problem ist und dass wir das in unserem Design besser gelöst haben.

Unser Ziel ist es, dass die Studierenden problemlos die Applikation bedienen können und den Ablauf des Prozessor-Zyklus so schnell wie möglich verstehen. Ausserdem möchten wir ihnen ein ansprechendes, modernes Design bieten.

In diesem Dokument beschreiben wir zuerst unser Vorgehen, danach fassen wir die Resultate der einzelnen Befragungen zusammen und interpretieren sie und zum Schluss listen wir die konkreten Rückmeldungen und Verbesserungsvorschläge auf. Im Anhang I befinden sich die genauen Protokolle der Tests.

Vorgehen

Die Vorlesung zum Prozessor-Zyklus fand in der Unterrichtswoche zwei statt, unser Usability Testing in der Woche acht. Die Bsys1 Studierenden, die sich für den Test angemeldet haben, wurden via Teams einzeln befragt. Es stand den Teilnehmenden frei die Kamera einzuschalten. Sie wurden darüber informiert, dass der Call zur Ergänzung des Testprotokolls aufgezeichnet wird, die Aufzeichnungen jedoch nicht mit der Arbeit abgegeben werden. Zur Anonymisierung wurden den Studierenden in der Arbeit eine Nummer (z.B. “S001”) zugeordnet. Die Studierenden wurden für die Teilnahme am Usability Test nicht entschädigt.

Als erstes haben wir zusammen mit ihnen einen Fragebogen zu ihren allgemeinen Informatikvorkenntnissen und einen zu ihren Kenntnissen bezüglich Prozessor-Zyklus ausgefüllt. Ziel der Fragen war, die Schwierigkeiten der Thematik zu erkennen und zu sehen, ob unser Simulator dabei hilft, diese zu lösen. Deshalb wurde auch erst ganz am Schluss die Richtigkeit der gegebenen Antworten aufgelöst.

Danach haben wir ihnen unseren Design Ansatz gezeigt, den wir in Microsoft PowerPoint animiert haben. Die Testpersonen wurden aufgefordert laut zu denken und zu erklären, was in jedem Schritt passiert und Fragen zu stellen, falls etwas unklar sei. Am Schluss der Animation wurden sie befragt, wie sie die Knöpfe interpretieren. Ausserdem wurde zum Memory

im Design gefragt, was ein grauer Kreis darstellt und welche Adresse so ein Byte im Memory hat.

Der Simulator wurde dann wieder ausgeblendet und die Studierenden mussten noch einmal dieselben Fragen wie zu Beginn beantworten. Danach haben wir die Antworten mit ihnen zusammen korrigiert und falls nötig noch weitere Erläuterungen zum Prozessor-Zyklus gegeben. Zur Veranschaulichung wurde der Simulator dabei wieder eingeblendet. Anschliessend haben wir die Studierenden noch um Feedback zum Simulator und zum Design gebeten.

Resultate der Befragung

Teilnehmende

Bei den Teilnehmenden handelt es sich um eine Frau und sechs Männer, die alle Bsys1 besuchen und im 1. Semester des Informatikstudiums sind. Sie haben unterschiedliche Ausbildungen vor dem Studium gemacht und die meisten schätzen ihre Informatikvorkenntnisse als gut und ein paar als mittelmässig ein. Die meisten gaben an, die Vorlesung zum Prozessor-Zyklus gut verstanden zu haben und jemand kannte ihn bereits vor dem Studium. Zwei konnten der Vorlesung nur einigermassen folgen. Die genaue Auswertung zu den Teilnehmenden ist in der Tabelle B 1 ersichtlich.

Was hast du vor dem Studium gemacht?	
4	Lehre als Informatiker/Informatikerin
1	Kaufmännische Lehre (Mediamatiker)
1	Gymnasium
1	Elektroniker Lehre
Wie würdest du deine Vorkenntnisse für das Informatikstudium einschätzen?	
0	Sehr gut, ich bin unterfordert im Informatikstudium
4	Gut
2	Mittelmässig
1	Ich habe vor dem Studium den Programmiervorkurs besucht
Wie schätzt du deine Kenntnisse über den Prozessor-Zyklus ein?	
1	Was in Bsys1 zum Prozessor-Zyklus unterrichtet wird, wusste ich bereits vorher schon.
4	Ich konnte den Vorlesungen zum Prozessor-Zyklus gut folgen und habe das meiste verstanden
2	Ich habe die Vorlesungen von Bsys1 zum Prozessor-Zyklus einigermassen verstanden.
0	Was ist ein Prozessor-Zyklus?

Tabelle B 1 Auswertung Fragen zur Person

Fragen zum Prozessor-Zyklus

Um die Schwierigkeiten besser zu erkennen, haben wir den Probanden Fragen zum Prozessor-Zyklus gestellt. Für eine Aussage über den Lernerfolg müsste jedoch eine Studie mit viel mehr Probanden gemacht werden.

Folgende Fragen zum Prozessor-Zyklus wurden gestellt:

1. Beschreibe den Ablauf des Prozessor-Zyklus
2. Woher holt der Prozessor die Instruktionen, die du in dein Programm geschrieben hast?
3. Beschreibe schrittweise was der Prozessor bei der folgenden Assembly Instruktion macht: mov rax, [0x1000].
4. Mit welchem/r Code/Sprache arbeitet der Prozessor?
5. Was ist ein Register?

Die Auswertung der Antworten zum Prozessor-Zyklus ist in der Tabelle B 2 ersichtlich. Die Fragen wurden insgesamt in der zweiten Runde etwas besser beantwortet. Die Frage vier wurde nach der Präsentation schlechter beantwortet. Bei dieser Frage ging es uns darum, herauszufinden, ob der Unterschied Assembler und Maschinencode klar ist. Wir gehen davon aus, dass die Frage etwas gemein gestellt war und die Teilnehmenden verunsichert waren, da wir kein Feedback gegeben haben.

Ausserdem ist uns aufgefallen, dass diejenigen Fragen, bei denen die Teilnehmenden sich sehr sicher bei ihrer falschen Antwort waren, in der zweiten Runde noch einmal gleich beantwortet wurden. Diejenigen Fragen, bei denen sie unsicher waren, wurden in der zweiten Runde besser beantwortet. Für ein nächstes Usability Testing wäre es noch spannend, wenn die Studierenden ihre Antworten mithilfe des Simulators selbst korrigieren müssten.

Nr.	Stichwort	Vorher Richtig	Vorher Falsch	Nachher Richtig	Nachher Falsch	Tendenz
1	Ablauf Zyklus	4	3	6	1	besser
2	Lage Instruktion	4	3	5	2	besser
3	mov rax, [0x1000]	6	1	6	1	gleich
4	Prozessor Code	3	4	2	5	schlechter
5	Register	5	2	5	2	gleich
Total		**22**	**13**	**24**	**11**	**besser**

Tabelle B 2 Auswertung Fragen Prozessor-Zyklus

Fragen zum Design

Die Auswertung der Fragen zu den Design Elementen ist in Tabelle B 3 und die Ergebnisse der allgemeinen Fragen zum Design in Tabelle B 4 ersichtlich. Fast alle finden den Prozessor-Simulator toll. Nur zwei finden ihn okay, da sie sich nicht sofort zurechtgefunden haben und die Knöpfe nicht ganz klar waren. Einer wünscht sich zudem noch ein technisches Design. Alle können sich vorstellen, dass der Simulator den Studierenden hilft und es wurde teilweise betont, dass man sich so ein Simulator wünscht.

Die Symbole der Buttons müssen verbessert werden, da sie nicht klar sind. Der gebogene Pfeil wurde oft nicht als nächster Schritt, sondern als Überspringen verstanden. Die grauen Ovale und die Darstellung der Adresse war den meisten sofort klar. Ausserdem ist uns aufgefallen, dass die Instruction Pointer Erhöhung mit der Länge der Instruktion dank unserer Animation sehr gut verstanden wurde.

Buttons klar	
2	Ja
5	Nein
Graue Ovale klar	
6	Ja
1	Nein
Adresse (+..) klar	
5	Ja
2	Nein

Tabelle B 3 Auswertung Fragen Design Elemente

Gefällt dir unser Design für den Prozessor-Simulator?	
5	Ja, finde ich toll
2	Ich finde es okay
0	Naja…
0	Ich mag das Design nicht
Hast du dich in unserem Design gut zurechtgefunden?	
6	Ja
1	Nein
Kannst du dir vorstellen, dass unser Prozessor-Simulator den Studierenden aus Bsys1 helfen kann, den Prozessor-Zyklus besser zu verstehen?	
7	Ja
0	Nein

Tabelle B 4 Auswertung Fragen Design

Rückmeldungen

Die Rückmeldungen aus den Usability Tests werden hier zusammengefasst und zusätzlich mit Rückmeldungen aus dem Studienarbeitsmeeting mit dem Betreuer ergänzt (siehe Anhang H).

Positive Rückmeldungen

- Sehr übersichtlich, man sieht alles was man braucht, nachvollziehbar
- Viele Dinge, jedoch nur das Nötigste
- Interessant
- Selbsterklärend
- Farbtrennung gut
- Farben gut
- Man versteht die Übersicht nach ein paar Sekunden
- Hilft definitiv den Bsys1 Studierenden
- Das langsame Animationstempo hilft

Rückmeldungen, die umgesetzt werden müssen

Die folgenden Rückmeldungen sind unserer Meinung nach sehr wichtig und müssen eingebaut werden, bevor der Prozessor-Simulator im Unterricht eingesetzt wird.

- CPU Cycle Design ändern, um zu vermeiden, dass er gleich aussieht wie die CPU und Memory Objekte.
- CPU Cycle wäre besser mit einer Art Uhr in der Mitte, die sich dreht, statt dass man die Boxen dreht und mühsam animiert.
- ASCII Übersetzung verwirrt.
    - Wir lassen sie vorerst komplett weg.
- Farben innerhalb von einer Komponente einheitlicher, zum Beispiel keine roten Register.
- Farbkonzept noch einmal überarbeiten
- Hex Notation Adressen mit 0x ergänzen (0x1000).
- Zeigen, dass nach Intro Animation noch nichts passiert ist (es verwirrt, dass Current Instruction noch leer ist, obwohl CPU Cycle “Get Instruction” anzeigt)
- Beim Popup Dialog sollte man “src” und “dest” als “Source” und “Destination” ausschreiben, dass klar ist, was gemeint ist.
- Verschwinden von Instruction Pointer Pfeil unklar. Der Pfeil soll sich ausserdem von anderen Pfeilen unterscheiden
    - Wir designen IP so, dass man ihn nicht mehr ganz ausblenden muss
    - Die anderen Pfeile müssen sich in Aussehen und Animation klar davon unterscheiden
- Hover Funktionalität für Buttons
- Button Symbole sind unklar

    - Wir werden nur einen Button machen und dazu ein Toggle, um den schnellen Ablauf ein- und auszuschalten
- Flags sind unklar, wenn sie leer sind und nicht benötigt werden
- Alle Flags anzeigen, die im Unterricht behandelt werden
- Es wird gewünscht, dass man selbst Instruktionen eingeben kann
    - War sowieso schon angedacht
- Font muss angepasst werden, sodass [ ] besser erkennbar
    - Alles in Ovalen (Memory Einträge, Adressen, Register Einträge, Flags) werden weiterhin mit dem Font Roboto Mono dargestellt
    - Alle Texte und Assembly Code wird in anderem Font dargestellt
- Statt Popup Fenster, dass man wegdrücken muss, besser Log
    - Wir finden die Fenster gut, da man so zu Beginn gezwungen wird, diese zu lesen. Es soll jedoch möglich sein, die Popups auszuschalten
    - Falls wir ein Logging machen, dann nur für Execute Instruction und nicht für die Schritte, die immer gleichbleiben.
- Memory Adressen der Instructions wäre besser bis +F statt nur bis +9

Rückmeldungen für eine Erweiterung

Die folgenden Rückmeldungen könnten in einer späteren Erweiterung umgesetzt werden.

- Einblendung der Komponenten zu Beginn ist zu schnell
- Text Beschreibung bei Intro Animation
- ASCII Übersetzung soll bei Hover erscheinen
- Jump Instruktion animieren
- Stack zeigen
- Speicherbus wird gewünscht (z.B. Button, mit dem man ihn einschalten könnte)
- Eine Tabelle bei den Registern wäre noch schön, die anzeigt von wo bis wo EAX usw. wäre
- Benennung Prozessor als CPU evtl. nicht klar
    - Evtl. könnte man eine deutsche Übersetzung anbieten
- Zusammenhang Instruktion und Bytes gewünscht (48=mov)
    - Wäre wünschenswert, ist jedoch nur schwierig umzusetzen

Weitere Rückmeldungen

Diese Rückmeldungen werden wir aus den angegebenen Gründen nicht umsetzen.

- Weniger Border Radius und Shadow
    - Unserer Meinung nach machen diese beide Effekte das moderne Design aus.
- Assembly sollte keine externe Komponente sein, sondern direkt neben Maschinencode ins Memory geschrieben werden, wie in einem Hex Editor.
    - Machen wir nicht, da man denken könnte, dass ASCII Zeichen im Memory stehen.
- Assembly Interpretation gehört in CPU Cycle Box
    - Finden wir verwirrend, da Assembly Übersetzung wichtig ist, um den Maschinencode im Memory zu erkennen

- Zu simpel, sieht zu wenig technisch aus, Prozessor soll wie Prozessor geformt sein
    - Da es ein Modell ist, muss die Darstellung nicht der Realität entsprechen.
    - Wir finden, dass die Vereinfachung hilft, sich die Komponenten besser zu merken und die Simulation besser nachzuvollziehen
- Graue Ovale (Bytes) unnötig
    - Finden wir nicht, da während dem Usability Test den meisten direkt klar war, dass es sich um ein Byte handelt. Zudem haben viele dadurch gleich gemerkt, wie viele Bytes beim mov im Memory geholt werden und dass das der Grösse vom Register Rax entspricht.

Fazit

Grundsätzlich kann man sagen, dass unsere Designidee gut ist, den Studierenden hilft und ihnen gefällt. Ausserdem wurde auch ersichtlich, dass dieser Prozessor-Simulator und die Eingabe eigener Programme gewünscht wird und diese Arbeit berechtigt ist. Das Ziel, dass die Studierenden unsere Applikation gut bedienen können, haben wir noch nicht erreicht, da die Buttons noch nicht klar sind.

Wir haben sehr viele hilfreiche Inputs erhalten, was uns sehr freut und weiterbringt. Zusätzlich haben wir auch bereits wünschenswerte Funktionen des Prozessor-Simulators gesammelt, z.B. den Speicherbus anzeigen. Wir werden versuchen, so viele Rückmeldungen wie möglich bereits im ersten Inkrement umzusetzen. Zum Beispiel werden wir die ASCII Übersetzungstabelle weglassen, da sie mehr verwirrt als hilft. Man kann dann später überlegen, ob diese nötig ist und wie sie besser umgesetzt werden kann.

Schlussendlich haben wir festgestellt, dass bereits eine nicht animierte Ansicht des Zustands den Studierenden hilft, sich das ganze besser vorzustellen. Deshalb wird bei der Implementation, die Darstellung der einzelnen Komponenten höher priorisiert als die Animationen der einzelnen Schritte.

Anhang C Abhängigkeitsgraph

In der Abbildung C 1 wird der Abhängigkeitsgraph des MVP dargestellt. Bei den eingefärbten Felder wurden zur Vereinfachung mehrere Klassen und Files und deren Abhängigkeiten in einem Feld zusammengefasst.

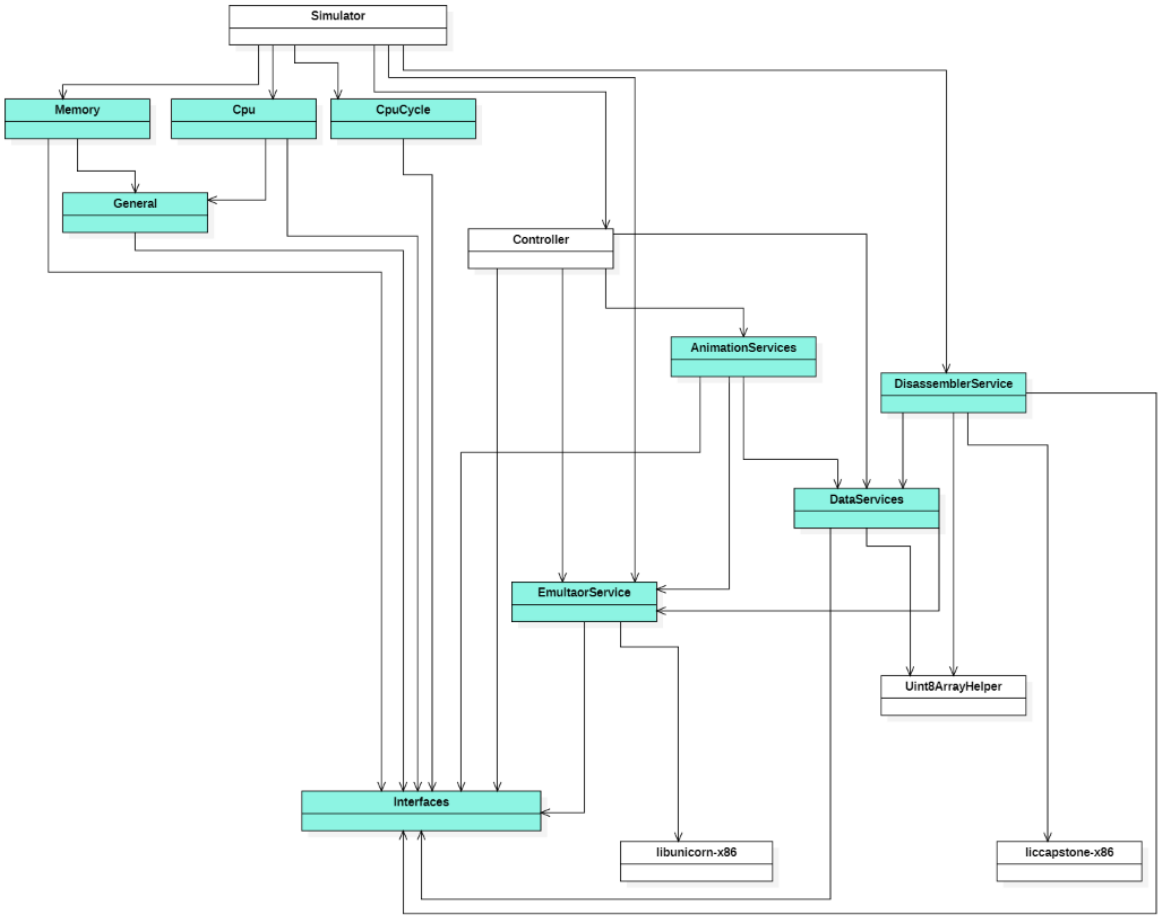
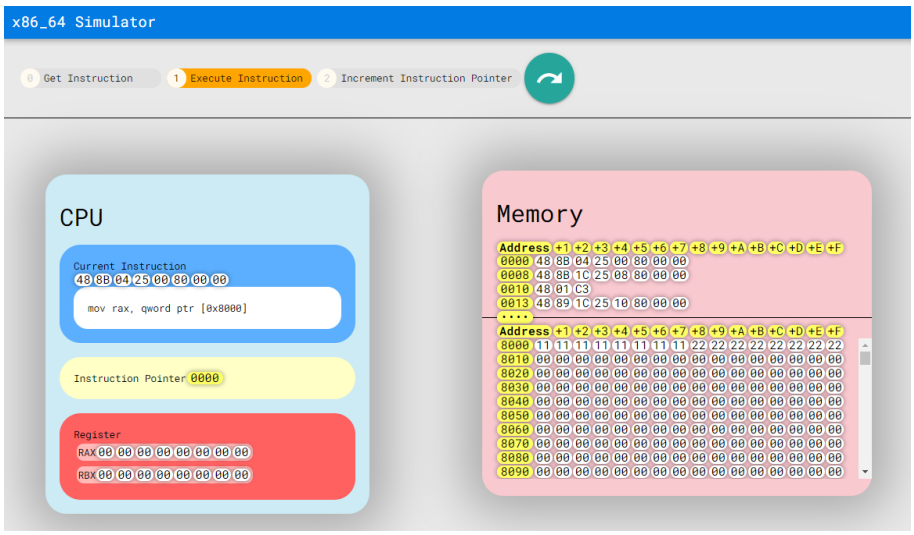


Abbildung C 1 Abhängigkeitsgraph Prozessor-Simulator

Anhang D Anleitung

Anleitung



Anleitung Prozessor-Simulator

Um den Prozessor-Simulator zu starten, braucht man das HTML File. Dieses kann zum Beispiel über den Skripte Server oder per E-Mail ausgeliefert werden.

Achtung: Der Prozessor-Simulator ist nur für den Desktop ausgerichtet und nicht für Mobiltelefone oder Tablets. Er wurde für den Browser Google Chrome entwickelt, sollte aber auch in anderen Browsern problemlos ausführbar sein.

Nach dem Herunterladen des HTML Files, kann es per Doppelklick gestartet werden. Danach sollte sich der Simulator im Standardbrowser öffnen. Leider ist es bis jetzt noch nicht möglich, eigenen Code einzugeben. Es wurde stattdessen ein Demo Programm hinterlegt:

mov rax, [0x8000]
mov rbx, [0x8008]
add rbx, rax
mov [0x8010], rbx

Über den Button kann dieses Programm schrittweise ausgeführt werden. Zu beachten ist noch, dass die Instruktion ‘add rax, rbx’ nicht animiert ist.

Anleitung Entwicklung

Für die Weiterentwicklung befindet sich eine Anleitung im Readme des Repository. Die Anleitung ist auf die Entwicklungsumgebung WebStorm von JetBrains ausgerichtet.

Anhang E Ergebnisse Webseiten Analyse

Ergebnisse Webseiten Analyse

Datum: 8. Dezember 2020
Tester: Yves Boillat

Hier werden die Ergebnisse der Webseiten Analyse mithilfe von Browser Tools dokumentiert. Die Tests wurden mit dem Endstand des Codes durchgeführt.

Ergebnis Chrome Performance Analyse

Um die Performance der Webseite während aktiver Nutzung zu messen, wurden 6 Sekunden mit dem Chrome Performance Tab analysiert. Die analysierte Zeitspanne beinhaltet den Ruhezustand sowie eine Animation. Von Sekunde 1.2 bis 3.2 befindet sich die Webseite im Ruhezustand und wartet auf eine Eingabe. Bei Sekunde 3.2 wird eine Animation gestartet, die bis zum Ende der Zeitachse im Gang ist.

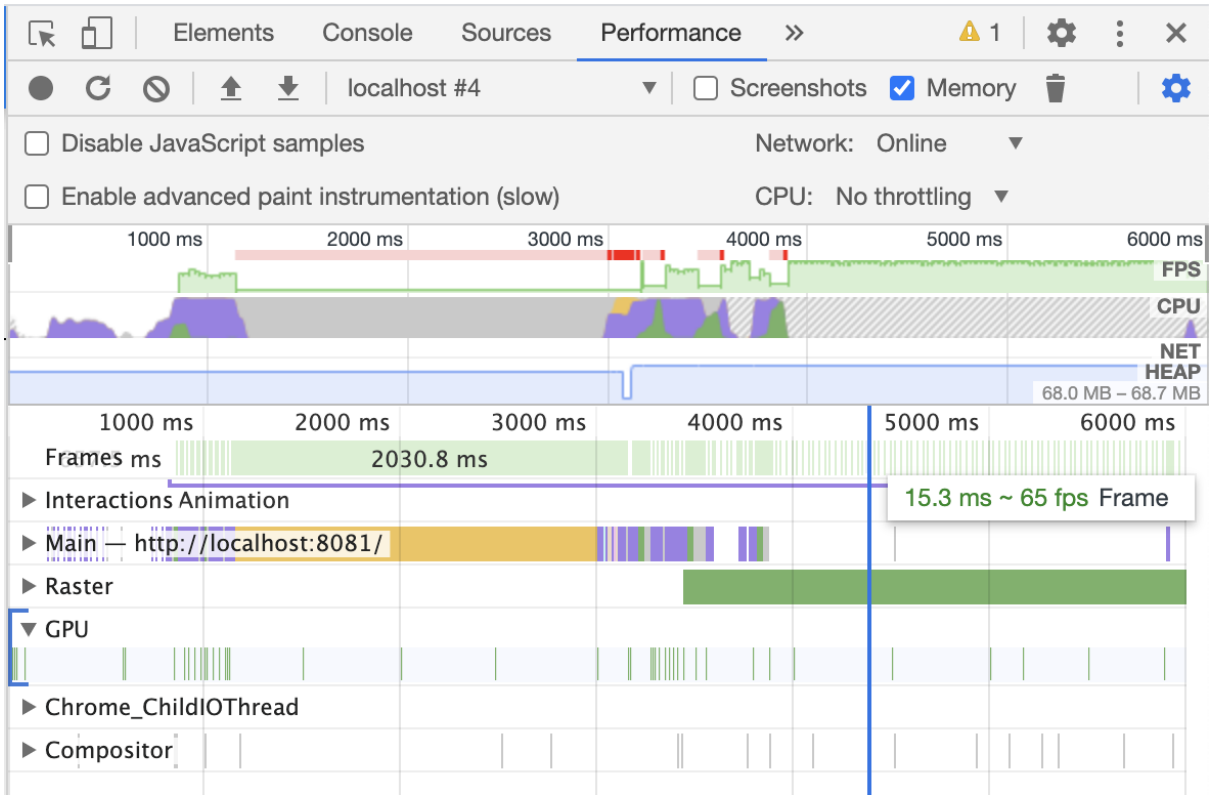


Abbildung E 1 Chrome Performance Tab

Framerate

Im oberen Bereich der Abbildung E 1 sieht man in hellgrün den Verlauf der Frames per Second (FPS) über die Zeit. Die Achse der FPS geht von 0 (unten) bis 60 (oben). Im Ruhezustand beträgt die FPS 0, da sich nichts bewegt oder ändert und somit keine Bildaktualisierungen nötig sind. Ab dem Start der Animation bei Sekunde 3.2 bis 3.9 sieht man, dass die FPS sehr stark zwischen 8 und 60 FPS schwankt. Ab Sekunde 3.9 ist die CSS-Animation fertig aufgebaut und im vollen Gange, wobei die Framerate bei 60 FPS liegt.

CPU-Nutzung

Im oberen Bereich der Abbildung E 1 sieht man in violett und dunkelgrün den Verlauf der CPU-Nutzung. Die Achse der CPU-Nutzung zeigt in Millisekunden die benötigte Rechenzeit an, jedoch ohne konkrete Minimal- oder Maximalwerte. Im Ruhezustand beträgt die CPU-Last 0. Ab dem Start der Animation bei Sekunde 3.2 bis 3.9 sieht man, dass die CPU-Nutzung auf das Maximum ansteigt. Die Detailansicht von “Main” im unteren Bereich in violett zeigt, dass die meiste Zeit verwendet wird, um dem “Layer Tree” zu aktualisieren. Ab Sekunde 3.9 ist die CSS-Animation fertig aufgebaut und im vollen Gange, wobei die CPU nicht mehr ausgelastet wird.

Ergebnis Chrome Lighthouse Test

Um die Seitenladezeit und andere Diagnostiken zu analysieren, verwenden wir den Chrome Lighthouse Test. Wir stellen unsere Webseite als “index.html” Datei lokal mit einem Python “SimpleHTTPServer” zur Verfügung, da der Lighthouse Test die Webseite nicht über das “File” Schema testen kann. Der Test misst die Seitenladezeit, in dem er die Datenrate eines 3G-Mobiltelefonnetz simuliert. Es wird eine TCP Round Trip Time (RTT) von 40 ms und eine Datenrate von 10,240 kbps simuliert. Es spielt somit keine Rolle, ob unser Server lokal auf dem System läuft oder über das Internet kontaktiert wird.

Seitenladezeit

Die Performancemessung von Lighthouse fällt mit 20 von möglichen 100 Punkten sehr schlecht aus. Das Ergebnis ist in der Abbildung E 2 ersichtlich. Der grösste Einfluss auf dieses Resultat hat der “Speed Index” von 4.9 Sekunden, der die Seitenladezeit widerspiegelt. Der “First Contentful Paint” von 4.9 Sekunden zeigt, wie schnell das Grundlayout auf dem Bildschirm angezeigt wird.

Die “Time To Interactive” von 5.7 Sekunden zeigt, dass weitere 0.8 Sekunden vergehen, bis man mit der Webseite interagieren kann, um zum Beispiel den ersten Schritt im Simulator zu starten. Der “Cumulative Layout Shift” von 0.003 zeigt, dass sich das Layout der Webseite nicht gross verändert oder hin- und herspringt, sobald die Seite angezeigt wird, was unschön aussehen würde.

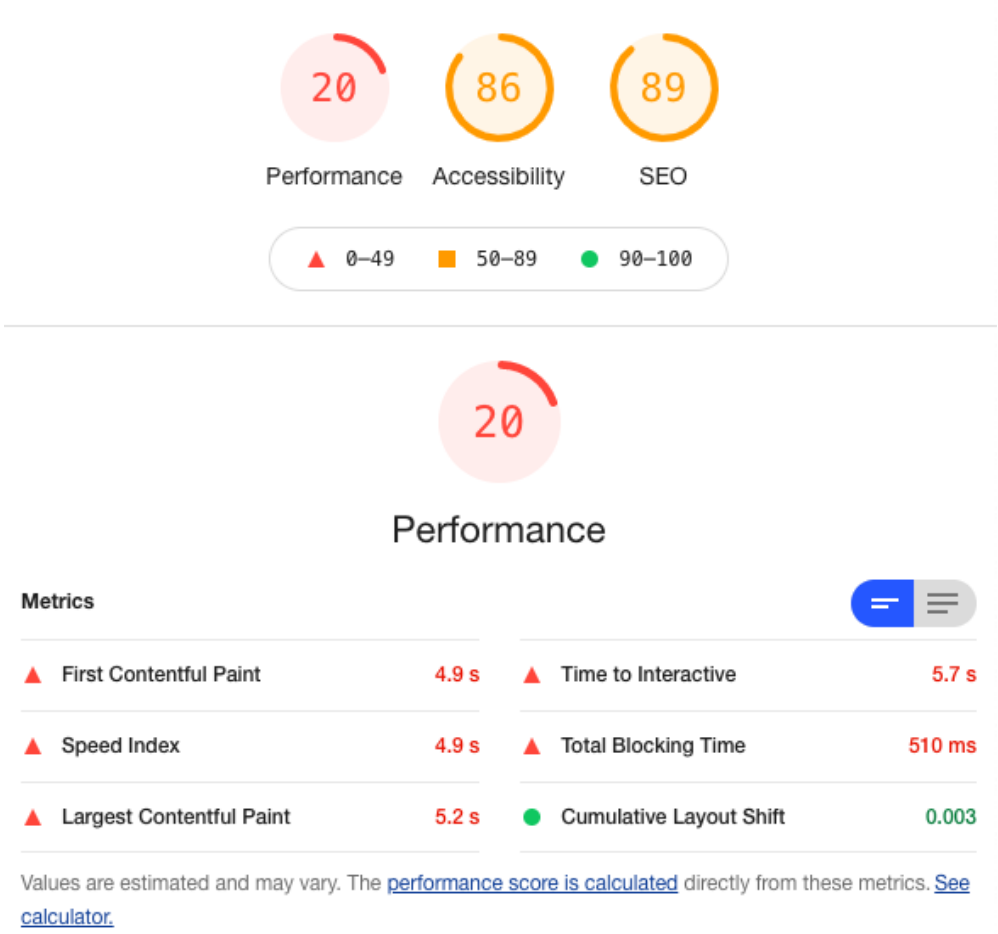


Abbildung E 2 Chrome Lighthouse Test - Performancemessung

Weitere Resultate

Wie in der Abbildung E 3 zu sehen ist, zeigt der Lighthouse Test weiteres Verbesserungspotenzial der Webseite an. Die Webseite beinhaltet 13’875 DOM Elemente. Der Test bemängelt auch die grosse Netzwerklast. Zusätzlich soll einiger JavaScript Code direkt nach dem Laden der Seite unbenutzt sein.

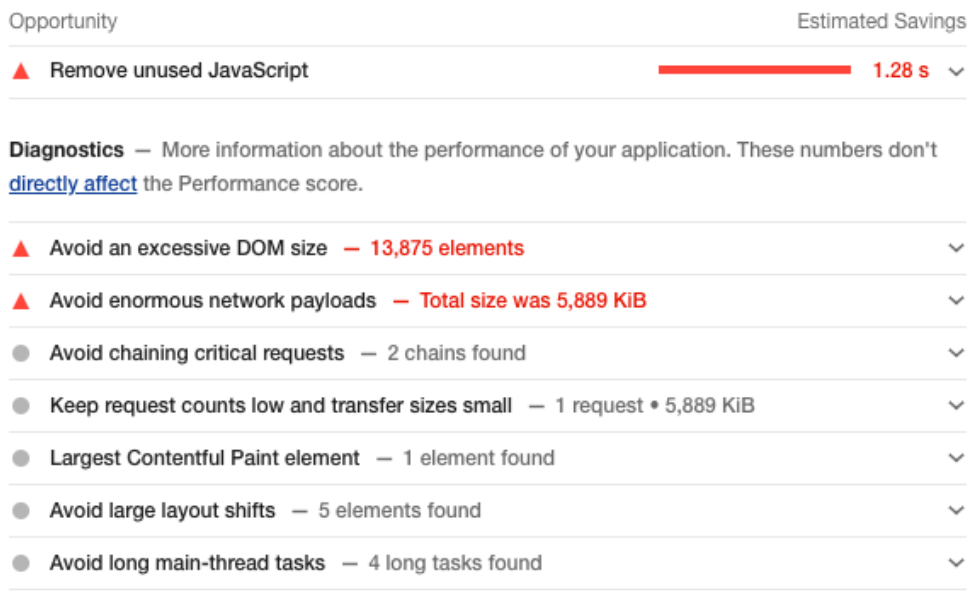


Abbildung E 3 Chrome Lighthouse Test - Weitere Resultate

Anhang F Ergebnisse Codeanalyse Tools

Ergebnisse Codeanalyse Tools

Datum: 8. Dezember 2020
Tester: Eliane Schmidli

Hier werden die Analysen der Codeanalyse Tools mit dem Code des MVP dokumentiert.

Ergebnis Better Code Hub

Wie in der Abbildung F 1 zu sehen ist, haben wir das Resultat 10 von 10 erhalten. Ein paar Details und die verwendete Konfiguration sind weiter unten beschrieben.

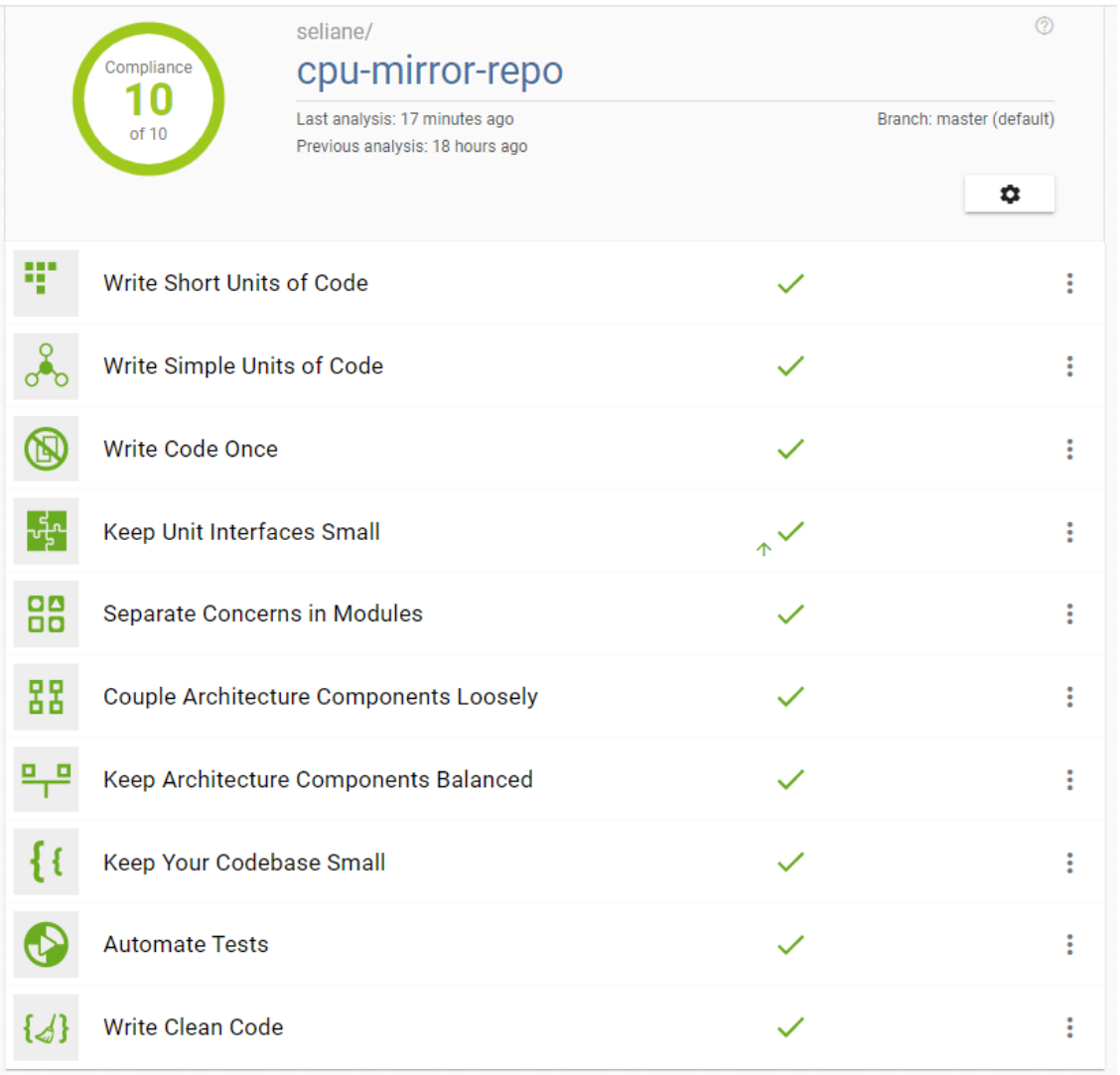


Abbildung F 1 Ergebnis Better Code Hub

In zwei Kategorien hat Better Code Hub noch ein paar Unschönheiten gefunden, die jedoch nicht als schlimm eingestuft werden. In der Abbildung F 2 ist die Kategorie „Write Short Units of Code“ abgebildet, die Funktionen mit mehr als 15 Zeilen Code anzeigt. Die maximale Anzahl Zeilen, die unsere Funktionen haben beträgt 21. In der Abbildung F 3 werden Funktionen mit mehr als zwei Parametern aufgelistet. Unsere maximale Anzahl Parameter beträgt 3.

Abbildung F 2 Details zu Write Short Units of Code

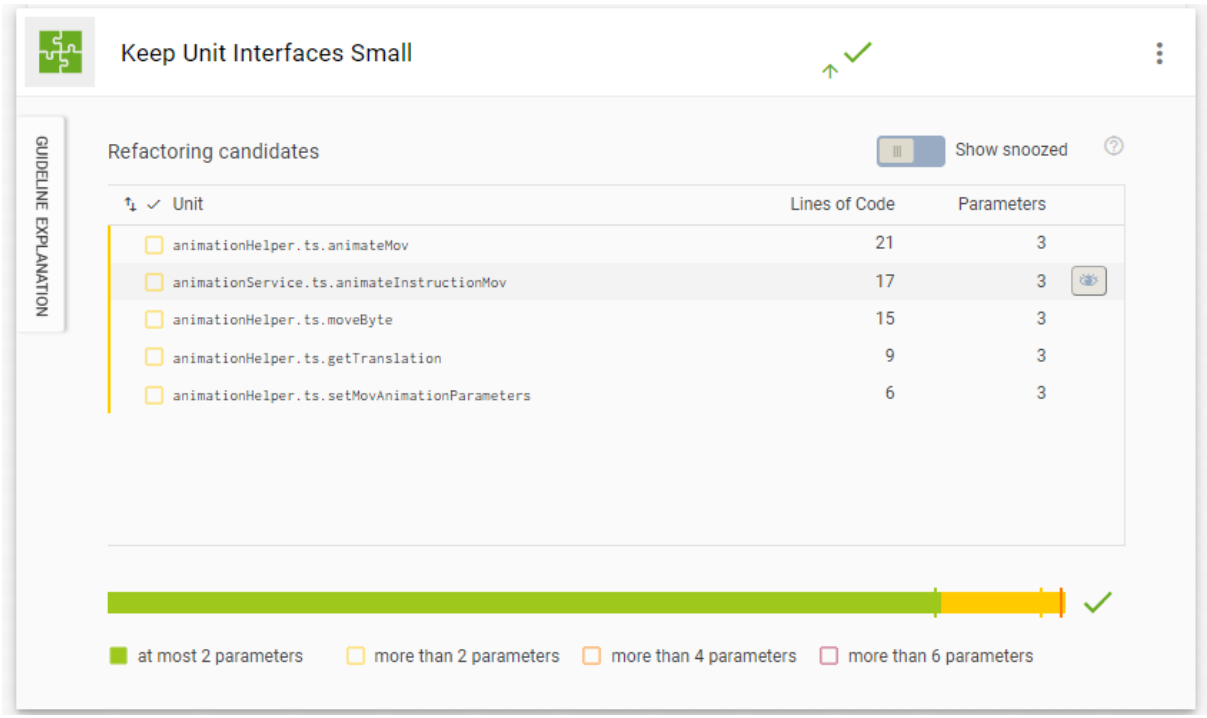


Abbildung F 3 Details zu Keep Unit Interfaces Small

Die Abbildung F 4 zeigt die von uns verwendete Konfiguration an. Wir haben die Files für den Emulator und den Disassembler von der Analyse ausgeschlossen, da wir diese mehrheitlich nicht selbst geschrieben haben. Better Code Hub ignoriert ausserdem den Code von unseren Vue Komponenten, da es die Dateiendung „.vue“ nicht kennt.

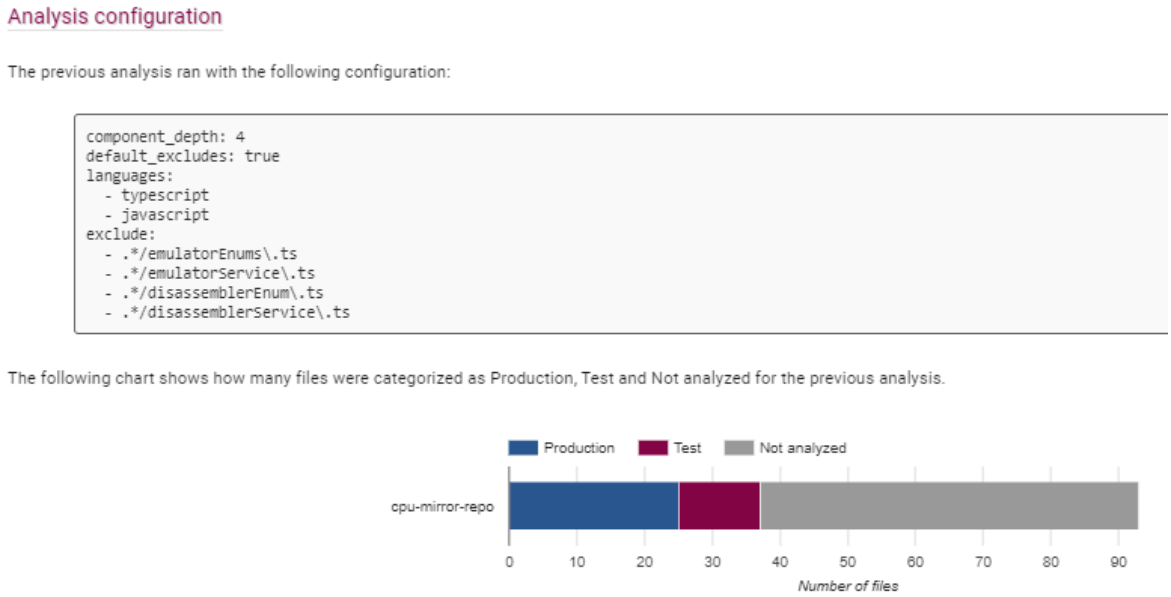


Abbildung F 4 Verwendete Konfiguration für Better Code Hub Analyse

Ergebnis DeepScan

Die Abbildung F 5 zeigt unser Ergebnis von der Analyse mit DeepScan. Wir haben den besten Grad „Good“ erhalten. Von den 2’020 Zeilen Code sind 1’341 Zeilen Source Code und 679 Zeilen Test Code.

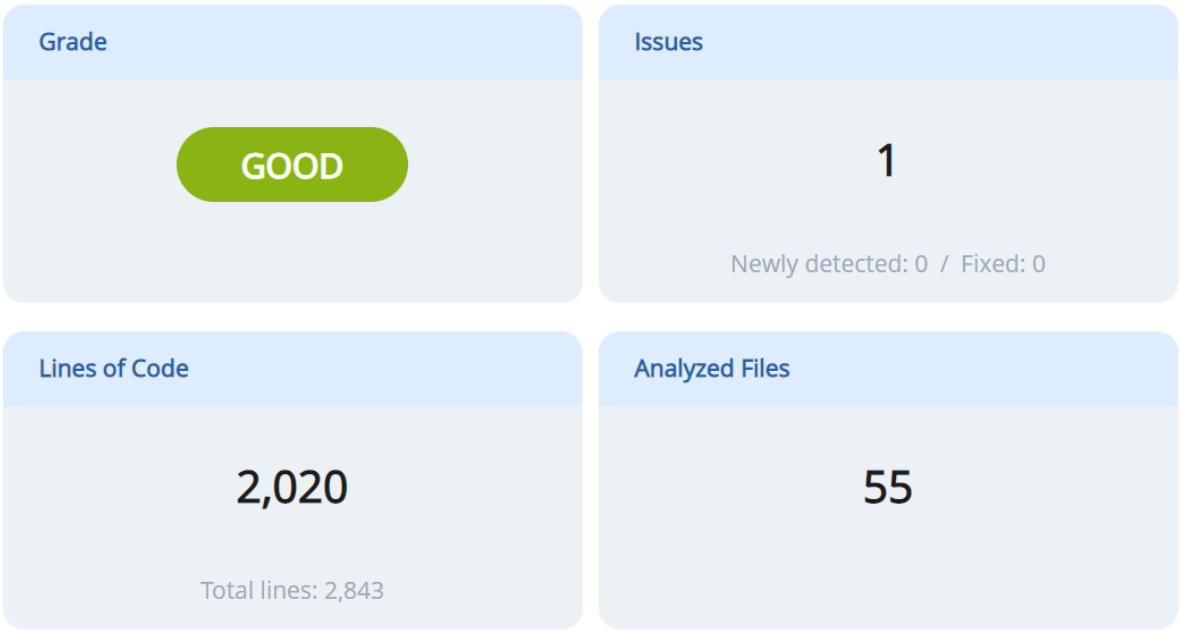


Abbildung F 5 Ergebnis DeepScan

DeepScan hat eine Schwachstelle in unserem Code gefunden. Diese wird in Abbildung F 6 genauer dargestellt. Wir können diese jedoch nicht beheben, da sonst der Linter fehlschlägt.

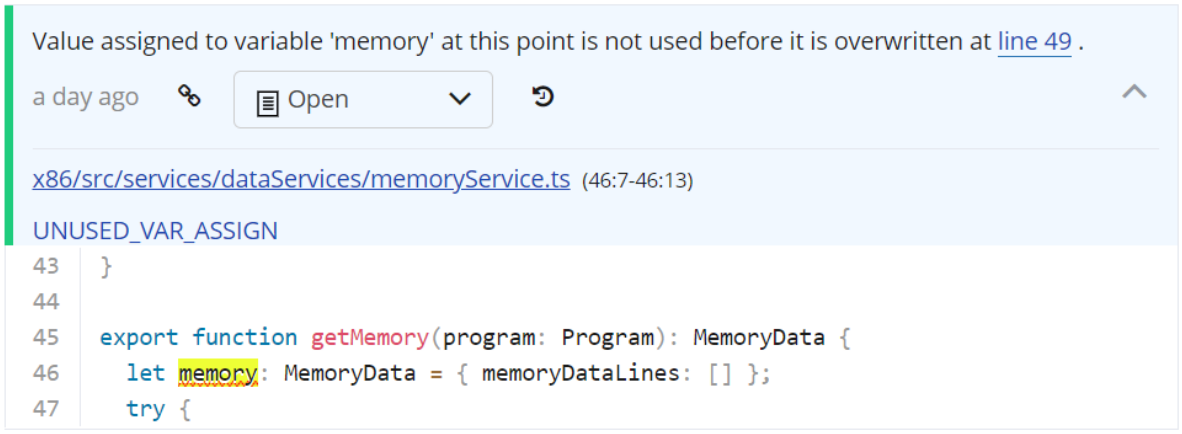


Abbildung F 6 Details zu Issue

Anhang G Protokoll Systemtest

Durchführung 9 - 14.12.2020

Test auf Branch “master” mit Pipeline-Artefakt “index.html”. Getestet mit Betriebssystem MacOS im Browser Safari.

Datum: 14.12.2020
Tester: Yves Boillat

Bekannte Einschränkungen

- Assembly Interpretation von mov rax, [0x8000] usw. werden als mov rax, qword ptr [0x8000] angezeigt.
- Bei der Ausführung als Datei werden im Web-Inspector Fehler angezeigt bezüglich des Ladens von lokalen Ressourcen: “Not allowed to load local resource: file:///favicon.ico” und “Not allowed to load local resource: file://...*.js.map”. Diese zählen nicht als Fehlermeldung für diesen Systemtest.

Protokoll

ST-UC01-01 - Simulator anzeigen

Aktionen	**Erwartetes Resultat**	**Kommentar von Tester**	**Status**
Doppel klicke Datei index.html	Header: “x86_64 Simulator” wird angezeigt		OK
	3 CPU Steps und Step Button werden angezeigt.		OK
	Step: 0 Get Instruction ist anders eingefärbt als die anderen Schritte		OK
	Cpu Komponente mit Titel “CPU” und Komponenten Current Instruction, Instruction Pointer und Registern wird angezeigt		OK
	Current Instruction hat Titel “Current Instruction”, zeigt 8 Bytes mit Inhalt “00” an, und ein leeres Instruktionsfeld.		OK
	Instruction Pointer hat Titel “Instruction Pointer” und ein vierstelliges Adressfeld mit auf “0000”		OK

	Register hat Titel “Register” und 3 Register mit Namen “RAX” und “RBX” und jeweils 8 Bytes mit Inhalt “00”		OK
	Memory hat Titel “Memory” Instruction und Memorydatenbereich		OK
	Instructiondaten im Memory haben Adresszeile von +1 bis +F Jede Datenzeile beinhaltet 4-Stellige Hex Adresse und Bytes.		OK
	Memorydaten haben Adresszeile von +1 bis +F Jede Datenzeile beinhaltet 4-Stellige Hex Adresse und Bytes.		OK
Scrolle Memory Daten	Scrollen von 8000 bis zur Adresse 8FF0 ist möglich. Alle Zeilen sind mit Bytes gefüllt		OK

ST-UC01-02 - Simulator anzeigen

Aktionen	**Erwartetes Resultat**	**Kommentar von Tester**	**Status**
Doppel klicke Datei index.html	Simulator wird angezeigt		OK
Inspector öffnen	In Console wird kein Fehler angezeigt.		OK
Step Button drücken	Erste Zeile von Instruktionen (Adresse 0000) wird geclont und bewegt sich zu Current Instruction und ersetzt die Bytes dort.		OK
	Step Button ist während Animation disabled		OK
	Assembly Übersetzung “mov rax, qword ptr [0x8000]” wird angezeigt		OK
	Step ändert auf “1 Execute Instruction”		OK
	Alle anderen Daten bleiben unverändert.		OK
	In der Konsole des Inspectors wird kein Fehler angezeigt.		OK
Step Button drücken	Erste 8 Bytes von Memory Daten (Adresse 8000) werden geclont und bewegen sich zu Current Instruction und dann zum Register “RAX”. Die Bytes in Register werden ersetzt.		OK

	Step Button ist während Animation disabled		OK
	Register “RAX” zeigt “11 11 11 11 11 11 11 11”		OK
	Step ändert auf “2 Increment Instruction Pointer”		OK
	Alle anderen Daten bleiben unverändert.		OK
	In der Konsole des Inspectors wird kein Fehler angezeigt.		OK
Step Button drücken	Current Instruction Bytes fliegen zu Instruction Pointer und legen sich übereinander. Der Instruction Pointer ändert auf “0008”		OK
	Step Button ist während Animation disabled		OK
	Step ändert auf “0 Get Instruction”		OK
	Alle anderen Daten bleiben unverändert.		OK
	In der Konsole des Inspectors wird kein Fehler angezeigt.		OK
Step Button drücken	Zweite Zeile von Instruktionen (Adresse 0008) wird geclont und bewegt sich zu Current Instruction und ersetzt die Bytes dort.		OK
	Step Button ist während Animation disabled		OK
	Assembly Übersetzung “mov rbx, qword ptr [0x8008]” wird angezeigt		OK
	Step ändert auf “1 Execute Instruction”		OK
	Alle anderen Daten bleiben unverändert.		OK
	In der Konsole des Inspectors wird kein Fehler angezeigt.		OK
Step Button drücken	8 Bytes von Memory Daten (Adresse 8008) werden geclont und bewegen sich zu Current Instruction und dann zum Register “RBX”. Die Bytes in Register werden ersetzt.		OK
	Step Button ist während Animation disabled		OK
	Register “RBX” zeigt “22 22 22 22 22 22 22 22” an		OK
	Register “RAX” zeigt “11 11 11 11 11 11 11 11” an		OK

	Step ändert auf “2 Increment Instruction Pointer”		OK
	Alle anderen Daten bleiben unverändert.		OK
	In der Konsole des Inspectors wird kein Fehler angezeigt.		OK
Step Button drücken	Current Instruction Bytes fliegen zu Instruction Pointer und legen sich übereinander. Der Instruction Pointer ändert auf “0010”		OK
	Step Button ist während Animation disabled		OK
	Step ändert auf “0 Get Instruction”		OK
	Alle anderen Daten bleiben unverändert.		OK
	In der Konsole des Inspectors wird kein Fehler angezeigt.		OK
Step Button drücken	Dritte Zeile von Instruktionen (Adresse 0010) wird geclont und bewegt sich zu Current Instruction und ersetzt die Bytes dort.		OK
	Step Button ist während Animation disabled		OK
	Assembly Übersetzung “add rbx, rax” wird angezeigt		OK
	Step ändert auf “1 Execute Instruction”		OK
	Alle anderen Daten bleiben unverändert.		OK
	In der Konsole des Inspectors wird kein Fehler angezeigt.		OK
Step Button drücken	Register RBX und RAX werden addiert (ohne Animation)		OK
	Step Button ist während Animation disabled		OK
	Register “RBX” zeigt “33 33 33 33 33 33 33 33” an		OK
	Register “RAX” zeigt “11 11 11 11 11 11 11 11” an		OK
	Step ändert auf “2 Increment Instruction Pointer”		OK
	Alle anderen Daten bleiben unverändert.		OK
	In der Konsole des Inspectors wird kein Fehler angezeigt.		OK

Step Button drücken	Current Instruction Bytes fliegen zu Instruction Pointer und legen sich übereinander. Der Instruction Pointer ändert auf “0013”		OK
	Step Button ist während Animation disabled		OK
	Step ändert auf “0 Get Instruction”		OK
	Alle anderen Daten bleiben unverändert.		OK
	In der Konsole des Inspectors wird kein Fehler angezeigt.		OK
Step Button drücken	Vierte Zeile von Instruktionen (Adresse 0013) wird geclont und bewegt sich zu Current Instruction und ersetzt die Bytes dort.		OK
	Step Button ist während Animation disabled		OK
	Assembly Übersetzung “mov qword ptr [0x8010], rbx” wird angezeigt		OK
	Step ändert auf “1 Execute Instruction”		OK
	Alle anderen Daten bleiben unverändert.		OK
	In der Konsole des Inspectors wird kein Fehler angezeigt.		OK
Step Button drücken	8 Bytes von Register “RBX” werden geclont und bewegen sich zu Current Instruction und dann zu Memory Daten (Adresse 8010). Die Bytes werden in die zweite Reihe eingesetzt.		OK
	Step Button ist während Animation disabled		OK
	Register “RBX” zeigt “33 33 33 33 33 33 33 33” an		OK
	Step ändert auf “2 Increment Instruction Pointer”		OK
	Alle anderen Daten bleiben unverändert.		OK
	In der Konsole des Inspectors wird kein Fehler angezeigt.		OK
Step Button drücken	Alle Steps sind ausgegraut		OK
	Step Button ändert Symbol zu Reload		OK
	Alle anderen Daten bleiben unverändert.		OK

	In der Konsole des Inspectors wird kein Fehler angezeigt.		OK
Step Button drücken	Seite wird neu geladen		OK
	Get Instruciton Schritt wird angezeigt		OK
	Step Button hat wieder Step Symbol		OK
	Alle Komponenten werden wie zu Beginn eingeblendet		OK

ST-UC01-03 - Reload Seite

Aktionen	**Erwartetes Resultat**	**Kommentar von Tester**	**Status**
Doppel klicke Datei index.html	Simulator wird angezeigt		OK
Step Button drücken	Get Instruction wird ausgeführt		OK
Seite Refreshen	Seite wird neu geladen		OK
	Status quo wird hergestellt, Step = Get Instruction usw.		OK

ST-UC01-04 - Memory Scrollen

Aktionen	**Erwartetes Resultat**	**Kommentar von Tester**	**Status**
Doppel klicke Datei index.html	Simulator wird angezeigt		OK
Step Button drücken	Get Instruction wird ausgeführt		OK
Scrollen im Memory nach unten	Scrollen ist möglich und Daten werden angezeigt		OK
Step Button drücken	Memory wird in 1. Zeile gescrollt und execute Instruction ausgeführt		OK

Anhang H Rückmeldung Betreuer zu Design

Rückmeldung Design Betreuer

SA Meeting W08

Rückmeldung Stefan Richter

- Entspricht nicht dem was er sich vorgestellt hat, er findet es aber gut
- Assembly Code hat keine Semikolons am Ende der Zeile
- ASCII Übersetzung ist etwas verwirrend und nicht nötig
    - Besser wäre, dass man Bytes markiert und die ASCII Übersetzung dafür sehen kann
- Er fragt sich, ob die einzelnen Byte Boxen notwendig sind
    - wir finden, dass diese hilfreich sind, um die Länge einer Sequenz zu sehen
- Font ist ungeeignet um [ ] bei Assembly darzustellen. Man sieht nicht gut, dass es sich um eckige Klammern handelt
- Increment Instruction: sollte zuerst die Adresse in CPU hochzählen und dann Pfeil verschieben
- Popup Fenster findet er eher mühsam, da man das wegklicken muss. Besser wäre ein Log Fenster, wo man im Nachhinein hochscrollen könnte zum Log Eintrag, den man noch einmal lesen möchte.
- CPU Cycle könnte man auch mit einer Art Uhr in der Mitte abbilden, die dreht, statt dass man die Boxen dreht.
- Flags: Parity braucht es nicht unbedingt, aber Carry, Zero, Sign, Overflow
    - könnte man auch nur mit Abkürzungen darstellen
- Eine Tabelle bei den Registern wäre noch schön, die anzeigt von wo bis wo EAX usw. wäre
- Farbkonzept passt noch nicht ganz
- Instruction Memory wäre besser bis +F

Rückmeldungen Florian Bruhin

- Ihm gefällt das Design
- seiner Meinung nach sind die Farben okay und tragen auch zum Lerneffekt bei.

Für den Usability Test haben wir folgende Punkte bereits umgesetzt

- Increment Instruction Pointer: Wird zuerst hochgezählt in CPU
- Semikolon in Assembly entfernt

Die restlichen Punkte wurden zusammen mit den Rückmeldungen aus dem Usability Test diskutiert und je nachdem umgesetzt.

Anhang I Protokolle Usability Tests

Usability Test Vorlage Fragebogen

Datum: 04. November 2020

Im Rahmen unserer Studienarbeit erstellen wir einen Prozessor-Simulator, welcher in Zukunft den Bsys1 Studierenden helfen soll, die einzelnen Schritte des Prozessor-Zyklus besser zu verstehen. Um dieses Ziel zu erreichen, sind wir auf deine Hilfe angewiesen. Wir möchten herausfinden, wo genau die Verständnisprobleme liegen und ob unsere Applikation dabei helfen kann.

Herzlichen Dank für deine Mithilfe!
Yves Boillat & Eliane Schmidli

Angaben zu deiner Person

Vorname: ……...
In welchem Semester bist du? ……...

Was hast du vor dem Studium gemacht?
☐ Lehre als Informatiker/Informatikerin
☐ Kaufmännische Lehre
☐ Gymnasium
☐ Anderes: ……...

Wie würdest du deine Vorkenntnisse für das Informatikstudium einschätzen?
☐ Sehr gut, ich bin unterfordert im Informatikstudium
☐ Gut
☐ Mittelmässig
☐ Ich habe vor dem Studium den Programmiervorkurs besucht

Wie schätzt du deine Kenntnisse über den Prozessor-Zyklus ein?
☐ Was in Bsys1 zum Prozessor-Zyklus unterrichtet wird, wusste ich bereits vorher schon.
☐ Ich konnte den Vorlesungen zum Prozessor-Zyklus gut folgen und habe das meiste verstanden
☐ Ich habe die Vorlesungen von Bsys1 zum Prozessor-Zyklus einigermassen verstanden.
☐ Was ist ein Prozessor-Zyklus?

Falls dir gerade etwas in den Sinn kommt, was du nicht verstanden hast, schreibe das bitte hierhin:

Fragen zum Prozessor-Zyklus Teil 1

Beschreibe den Ablauf des Prozessor-Zyklus:

Woher holt der Prozessor die Instruktionen, welche du in dein Programm geschrieben hast?

Aus dem Memory (auch okay, da wo IP hinzeigt)

Beschreibe schrittweise was der Prozessor bei der folgenden Assembly Instruktion macht:
mov rax, [0x1000]

Kopie Wert an Adresse 0x1000 in Register RAX

Mit was für einem Code/Sprache arbeitet der Prozessor?

Maschinen Code (Ziel ist es zu erkennen, ob sie Assembly und Maschinencode unterscheiden können)

Was ist ein Register?

temporärer Speicher, Gehört zu Prozessor

Fragen zum Prozessor-Zyklus Teil 2

Beantworte hier noch einmal die Fragen von oben:
Beschreibe den Ablauf des Prozessor-Zyklus:

Woher holt der Prozessor die Instruktionen, welche du in dein Programm geschrieben hast?

Beschreibe schrittweise was der Prozessor bei der folgenden Assembly Instruktion macht:
mov rax, [0x1000]

Mit was für einem Code/Sprache arbeitet der Prozessor?

Was ist ein Register?

Fragen zum Design

Gefällt dir unser Design für den Prozessor-Simulator?
☐ Ja, finde ich toll
☐ Ich finde es okay
☐ Naja…
☐ Ich mag das Design nicht
Verbesserungsvorschlag/Kommentar:

Hast du dich in unserem Design gut zurechtgefunden?
☐ Ja, weil….

☐ Nein, weil ....

Kannst du dir vorstellen, dass unser Prozessor-Simulator den Studierenden aus Bsys1 helfen kann, den Prozessor-Zyklus besser zu verstehen?
☐ Ja, weil …

☐ Nein, weil …

Was fehlt deiner Meinung nach noch, um den Prozessor-Zyklus besser zu verstehen?

Gibt es etwas, das du uns sonst noch sagen möchtest?

Usability Test S001

Datum: 04. November 2020

Im Rahmen unserer Studienarbeit erstellen wir einen Prozessor-Simulator, welcher in Zukunft den Bsys1 Studierenden helfen soll, die einzelnen Schritte des Prozessor-Zyklus besser zu verstehen. Um dieses Ziel zu erreichen, sind wir auf deine Hilfe angewiesen. Wir möchten herausfinden, wo genau die Verständnisprobleme liegen und ob unsere Applikation dabei helfen kann.

Herzlichen Dank für deine Mithilfe!
Yves Boillat & Eliane Schmidli

Angaben zu deiner Person

Vorname: S001...
In welchem Semester bist du? …1.

Was hast du vor dem Studium gemacht?
☐ Lehre als Informatiker/Informatikerin
☐ Kaufmännische Lehre
☒ Gymnasium
☐ Anderes: ……...

Wie würdest du deine Vorkenntnisse für das Informatikstudium einschätzen?
☐ Sehr gut, ich bin unterfordert im Informatikstudium
☐ Gut
☒ Mittelmässig
☐ Ich habe vor dem Studium den Programmiervorkurs besucht

Wie schätzt du deine Kenntnisse über den Prozessor-Zyklus ein?
☐ Was in Bsys1 zum Prozessor-Zyklus unterrichtet wird, wusste ich bereits vorher schon.
☒ Ich konnte den Vorlesungen zum Prozessor-Zyklus gut folgen und habe das meiste verstanden
☐ Ich habe die Vorlesungen von Bsys1 zum Prozessor-Zyklus einigermassen verstanden.
☐ Was ist ein Prozessor-Zyklus?

Falls dir gerade etwas in den Sinn kommt, was du nicht verstanden hast, schreibe das bitte hierhin:

-

Fragen zum Prozessor-Zyklus Teil 1

Beschreibe den Ablauf des Prozessor-Zyklus:
- Prozessor holt etwas aus Speicher
- Verarbeitet das mit Operation
- Schreibt wieder in Speicher

Woher holt der Prozessor die Instruktionen, welche du in dein Programm geschrieben hast?
- ?

Beschreibe schrittweise was der Prozessor bei der folgenden Assembly Instruktion macht:
mov rax, [0x1000]

- geht zu Speicherstelle 1000 holt wert
- kopiert ihn an Speicherstelle des Register RAX

Mit was für einem Code/Sprache arbeitet der Prozessor?
- Assembly

Was ist ein Register?
- kleiner Speicher auf/im Prozessor

Fragen zum Prozessor-Zyklus Teil 2

Beantworte hier noch einmal die Fragen von oben:
Beschreibe den Ablauf des Prozessor-Zyklus:

- nimmt Instruktion aus Speicher
- führt Instruktion aus (holt Daten aus Memory, schreibt in Register und schreibt in Speicher je nach dem)
- geht zur nächsten Instruktion

Woher holt der Prozessor die Instruktionen, welche du in dein Programm geschrieben hast?
- Speicher

Beschreibe schrittweise was der Prozessor bei der folgenden Assembly Instruktion macht:
mov rax, [0x1000]
- Instruktion wird geholt, springt zu 1000, kopiert Wert in Register, IP wird erhöht

Mit was für einem Code/Sprache arbeitet der Prozessor?
- Assembly

Was ist ein Register?
- kleiner Speicher im Prozessor

Fragen zum Design

Gefällt dir unser Design für den Prozessor-Simulator?
☒ Ja, finde ich toll
☐ Ich finde es okay
☐ Naja…
☐ Ich mag das Design nicht
Verbesserungsvorschlag/Kommentar:

- sehr übersichtlich, man sieht alles was man braucht, nachvollziehbar

Hast du dich in unserem Design gut zurechtgefunden?
☒ Ja, weil….

- selbsterklärend

☐ Nein, weil ....

Kannst du dir vorstellen, dass unser Prozessor-Simulator den Studierenden aus Bsys1 helfen kann, den Prozessor-Zyklus besser zu verstehen?
☒ Ja, weil …

- definitiv, ich hätte gern so ein Tool gehabt.

☐ Nein, weil …

Was fehlt deiner Meinung nach noch, um den Prozessor-Zyklus besser zu verstehen?

- dass man instruktionen aus der Vorlesung eingeben kann

Gibt es etwas, das du uns sonst noch sagen möchtest?
-

Usability Test Protokoll S001

Datum: 04. November 2020

Vorame: S001

Vor der Vorführung

Erkenntnisse aus dem Gespräch

- Weiss was Prozessor-Zyklus ist

Während der Vorführung

Protokoll zur Vorführung

Notizen zu Aussagen und zum Verhalten des Kandidaten

Assembly

- Erkennt Assembly
- Versteht Code

Intro animation

- Erkennt die verschiedenen Komponenten
- Am Anfang zuviel auf einmal
- Separation der Komponenten gut

Get instruction

- “CPU liest Instruktion beim Pointer”
- “CPU schreibt in Register”

Execute instruction

- Aussage Kandidat: “CPU springt wie beim Instruktions Pointer zu 1000”
- “Springen” ist der falsche Begriff und impliziert, dass CPU bei der Ausführung einen Pointer hat, welcher auf das Memory zeigt.

Increment IP

- “IP wird erhöht um Länge der Instruktion”

Fragestellungen zum Design

Spezifische Fragen, um unsere Design Entscheidungen auf Verständlichkeit zu testen.

Graue Ovale

- **Frage**: Für was stehen die grauen Ovale im Design?
  - Richtige Antwort: Ein Byte.
  - Antwort des Kandidaten: Zwei Hex Zahlen also 1 Byte
- War unsere Absicht klar: **Ja**

Address eines Bytes

- **Frage**: Im Memory auf der zweiten Zeile und in der dritten Spalte, welche Adresse hat das graue Oval “1D”?
  - Richtige Antwort: 000C
  - Antwort des Kandidaten: 000C
- War unsere Absicht klar: **Ja**

Knöpfe

- **Frage**: Wenn du nun in der Animation einen Schritt weitergehen möchtest, wohin würdest du klicken?
- Unsere Absicht:
  - Step Knopf: Geht zum nächsten Animationsschritt.
  - Play Knopf: Startet die Animation, ohne bei einem Schritt zu halten.
- Interpretation des Kandidaten:
  - Step Knopf: Instruktion überspringen
  - Play Knopf: Step machen
- War unsere Absicht klar: **Nein**

Nach der Vorführung

Anmerkungen

- Step Knopf sollte ein gerader Pfeil sein mit einem vertikalen Strich
- Findet Design super und übersichtlich

Verbesserungen im Verständnis

- Sagt immernoch “springen” bei der zweiten Fragerunde.

Wünsche

- Instruktionen selber eingeben

Usability Test S002

Datum: 04. November 2020

Im Rahmen unserer Studienarbeit erstellen wir einen Prozessor-Simulator, welcher in Zukunft den Bsys1 Studierenden helfen soll, die einzelnen Schritte des Prozessor-Zyklus besser zu verstehen. Um dieses Ziel zu erreichen, sind wir auf deine Hilfe angewiesen. Wir möchten herausfinden, wo genau die Verständnisprobleme liegen und ob unsere Applikation dabei helfen kann.

Herzlichen Dank für deine Mithilfe!
Yves Boillat & Eliane Schmidli

Angaben zu deiner Person

Vorname: S002...
In welchem Semester bist du? 1.

Was hast du vor dem Studium gemacht?
☒Lehre als Informatiker/Informatikerin
☐ Kaufmännische Lehre
☐ Gymnasium
☐ Anderes: ……...

Wie würdest du deine Vorkenntnisse für das Informatikstudium einschätzen?
☐ Sehr gut, ich bin unterfordert im Informatikstudium
☒Gut
☐ Mittelmässig
☐ Ich habe vor dem Studium den Programmiervorkurs besucht

Wie schätzt du deine Kenntnisse über den Prozessor-Zyklus ein?
☒ Was in Bsys1 zum Prozessor-Zyklus unterrichtet wird, wusste ich bereits vorher schon.
☐ Ich konnte den Vorlesungen zum Prozessor-Zyklus gut folgen und habe das meiste verstanden
☐ Ich habe die Vorlesungen von Bsys1 zum Prozessor-Zyklus einigermassen verstanden.
☐ Was ist ein Prozessor-Zyklus?

Falls dir gerade etwas in den Sinn kommt, was du nicht verstanden hast, schreibe das bitte hierhin:

Fragen zum Prozessor-Zyklus Teil 1

Beschreibe den Ablauf des Prozessor-Zyklus:
- Schaut, wo ist nächste Instruktion
- Instruktion ausführen
- Ort für nächste Instruktion ändert oder IP erhöht

Woher holt der Prozessor die Instruktionen, welche du in dein Programm geschrieben hast?
- holt vom Memory, Pointer zeigt wo

Beschreibe schrittweise was der Prozessor bei der folgenden Assembly Instruktion macht:
mov rax, [0x1000]
- holt vom RAM an Position 1000 die folgenden 7 Bytes und schreibt in Register RAX

Mit was für einem Code/Sprache arbeitet der Prozessor?
- Maschinencode

Was ist ein Register?
- Stück schnell zugreifbarer Speicher

Fragen zum Prozessor-Zyklus Teil 2

Beantworte hier noch einmal die Fragen von oben:
Beschreibe den Ablauf des Prozessor-Zyklus:

- Instruktion laden
- Instruktion ausführen
- IP erhöhen
- repeat

Woher holt der Prozessor die Instruktionen, welche du in dein Programm geschrieben hast?
- beim IP, zeigt aufs Ram oder Register?

Beschreibe schrittweise was der Prozessor bei der folgenden Assembly Instruktion macht:
mov rax, [0x1000]
- kopiert Werte von 1000-1007 in Register rax

Mit was für einem Code/Sprache arbeitet der Prozessor?
- Maschinencode

Was ist ein Register?

- Stück Speicher in CPI

Fragen zum Design

Gefällt dir unser Design für den Prozessor-Simulator?
☒Ja, finde ich toll
☐ Ich finde es okay
☐ Naja…
☐ Ich mag das Design nicht
Verbesserungsvorschlag/Kommentar:

- Cpu und Memory gut
- Cpu Cycle wirkt wie weitere Komponente
- Farbtrennung gut
- erschlagen am Anfang
    - besser: kommentieren: das ist die CPU mit dem und dem , dass ist das Memory
- Zusammenhang mov 48
- was bedeuten die Knöpfe

Hast du dich in unserem Design gut zurechtgefunden?
☒ Ja, weil….

- nach ein paar Sekunen

☐ Nein, weil ....

Kannst du dir vorstellen, dass unser Prozessor-Simulator den Studierenden aus Bsys1 helfen kann, den Prozessor-Zyklus besser zu verstehen?
☒ Ja, weil …

- wenn es nicht im Schnellzug Tempo ist, kann man sich es besser vorstellen

☐ Nein, weil …

Was fehlt deiner Meinung nach noch, um den Prozessor-Zyklus besser zu verstehen?
-

Gibt es etwas, das du uns sonst noch sagen möchtest?

- Flags sind noch nicht so ganz klar: stört nicht, man fragt sich aber was macht das dort

Usability Test Protokoll S002

Datum: 04. November 2020

Vorame: S002

Vor der Vorführung

Erkenntnisse aus dem Gespräch

- Der Kandidat sagt insgesamt fühle er sich sicher in Bsys aber es gäbe immer wieder den Moment, wenn man etwas im Detail anschaut, und man merkt “ah stimmt, der holt ja dann nicht nur 1 Byte von der Adresse, sondern auch noch die folgenden 7”
- Aussage: “Bei Intel holt CPU 8 Byte bei mov”
    - 8 Bytes werden geholt, wegen des Registers Rax

Während der Vorführung

Protokoll zur Vorführung

Notizen zu Aussagen und zum Verhalten des Kandidaten

Assembly

Intro animation

- Kandidat bewegt Augen stark
- Fände Ton cool
- Nennt als Komponenten:
    - CPU Übersicht
    - RAM mit Sachen drin
- Ist sich unsicher was orangen Pfeil ist (IP)

Get instruction

- Kandidat nickt, “Genau, CPU holt Instruktion, die wir wohl ausführen, an Adresse.

Execute instruction

- Findet, dass Pfeil von Instruktion zu Memory verkehrt ist.
- Er merkt, dass der Wert kopiert wird
- Er merkt, dass seine Antwort mit 8 geholten Bytes korrekt war.

Increment IP

- “Pointer wird erhöht, um die Länge der aktuellen Instruktion”
- “Pointer zeigt auf nächste Instruktion”

Fragestellungen zum Design

Spezifische Fragen, um unsere Design Entscheidungen auf Verständlichkeit zu testen.

Graue Ovale

- **Frage**: Für was stehen die grauen Ovale im Design?
    - Richtige Antwort: Ein Byte.
    - Antwort des Kandidaten: Ein Byte (ist sich aber nicht sicher)
- War unsere Absicht klar: **nein**

Address eines Bytes

- **Frage**: Im Memory auf der zweiten Zeile und in der dritten Spalte, welche Adresse hat das graue Oval “1D”?
    - Richtige Antwort: 000C
    - Antwort des Kandidaten: 000C
- War unsere Absicht klar: **Ja**

Knöpfe

- **Frage**: Wenn du nun in der Animation einen Schritt weitergehen möchtest, wohin würdest du klicken?
- Unsere Absicht:
    - Step Knopf: Geht zum nächsten Animationsschritt.
    - Play Knopf: Startet die Animation, ohne bei einem Schritt zu halten.
- Interpretation des Kandidaten:
    - Step Knopf: Step
    - Play Knopf: Spielt ab und verwandelt sich in ein Stopp
- War unsere Absicht klar: Kandidat ist sich nicht ganz sicher und würde über Buttons hovern, **nein**

Nach der Vorführung

Anmerkungen

- Button Symbole unklar
- Zu Beginn wird man erschlagen
- Man hat sich nach ein paar Sekunden zurechtgefunden
- Flags verwirren etwas, da sie nicht benötigt werden

- CPU und Memory sauber getrennt
- CPU Cycle sieht aus wie eine Komponente aus dem Computer

Verbesserungen im Verständnis

- Weiss nun das Register zur CPU gehört
- Ist sich nicht mehr sicher, ob Instuction Pointer auf Memory oder Register zeigt.

Wünsche

- Dass man genau sieht welcher Teil der Instruktion für mov steht (48)

Usability Test S003

Datum: 04. November 2020

Im Rahmen unserer Studienarbeit erstellen wir einen Prozessor-Simulator, welcher in Zukunft den Bsys1 Studierenden helfen soll, die einzelnen Schritte des Prozessor-Zyklus besser zu verstehen. Um dieses Ziel zu erreichen, sind wir auf deine Hilfe angewiesen. Wir möchten herausfinden, wo genau die Verständnisprobleme liegen und ob unsere Applikation dabei helfen kann.

Herzlichen Dank für deine Mithilfe!
Yves Boillat & Eliane Schmidli

Angaben zu deiner Person

Vorname: S003
In welchem Semester bist du? 1. ..

Was hast du vor dem Studium gemacht?
☒ Lehre als Informatiker/Informatikerin
☐ Kaufmännische Lehre
☐ Gymnasium
☐ Anderes: ……...

Wie würdest du deine Vorkenntnisse für das Informatikstudium einschätzen?
☐ Sehr gut, ich bin unterfordert im Informatikstudium
☒ Gut
☐ Mittelmässig
☐ Ich habe vor dem Studium den Programmiervorkurs besucht

Wie schätzt du deine Kenntnisse über den Prozessor-Zyklus ein?
☐ Was in Bsys1 zum Prozessor-Zyklus unterrichtet wird, wusste ich bereits vorher schon.
☒ Ich konnte den Vorlesungen zum Prozessor-Zyklus gut folgen und habe das meiste verstanden
☐ Ich habe die Vorlesungen von Bsys1 zum Prozessor-Zyklus einigermassen verstanden.
☐ Was ist ein Prozessor-Zyklus?

Falls dir gerade etwas in den Sinn kommt, was du nicht verstanden hast, schreibe das bitte hierhin:
-

Fragen zum Prozessor-Zyklus Teil 1

Beschreibe den Ablauf des Prozessor-Zyklus:
- Anweisung wird vom Prozessor aus Speicher geholt
- Anweisung wird verarbeitet und benötigt je nach dem die Register
- Je nachdem Daten in Register kopiert, zurück speichern

Woher holt der Prozessor die Instruktionen, welche du in dein Programm geschrieben hast?
- aus dem Speicher, wird auf Speicherbus abgelegt

Beschreibe schrittweise was der Prozessor bei der folgenden Assembly Instruktion macht:
mov rax, [0x1000]
- Daten 1000- 1008
- Datenbus: schreibt in Datenbus wo er Daten holen will
- Daten werden auf Bus gelegt,
- Prozessor liest Daten und legt diese in Register Rax
- Speicher bleibt gleich, nur Kopie

Mit was für einem Code/Sprache arbeitet der Prozessor?
- Assembly

Was ist ein Register?
- Speicherplätze auf Prozessor, verwaltet diese, schneller Zugriff

---

Fragen zum Prozessor-Zyklus Teil 2

Beantworte hier noch einmal die Fragen von oben:
Beschreibe den Ablauf des Prozessor-Zyklus:
- Prozessor holt Kommando bei Adresse von IP
- Interpretiert Instruktion
- führt sie aus

Woher holt der Prozessor die Instruktionen, welche du in dein Programm geschrieben hast?
- Speicher

Beschreibe schrittweise was der Prozessor bei der folgenden Assembly Instruktion macht:
mov rax, [0x1000]
- lädt Inhalt zwischen 1000 und 1008 Adresse ins Register Rax

Mit was für einem Code/Sprache arbeitet der Prozessor?
- Assembly

Was ist ein Register?
- Speicher liegt in Cpu, Cpu verwaltet und verwendet es für Operationen als Zwischenspeicher

Fragen zum Design

Gefällt dir unser Design für den Prozessor-Simulator?
☐ Ja, finde ich toll
☒ Ich finde es okay
☐ Naja…
☐ Ich mag das Design nicht
Verbesserungsvorschlag/Kommentar:

- zu simpel, Prozessor sollte als Prozessor geformt sein
- sieht zu wenig technisch aus

Hast du dich in unserem Design gut zurechtgefunden?
☐ Ja, weil….

☒ Nein, weil ....

- Cpu Cycle verwirrt da Design gleich ist, technisch wäre besser
- Knöpfe waren verwirrend.
- Ascii Interpretation schwierig. Besser wenn man selber darüber hovert
- Auch Assembly Interpretation weglassen oder nur zu beginn zeigen. Gehört eher zum Cpu Cycle

Kannst du dir vorstellen, dass unser Prozessor-Simulator den Studierenden aus Bsys1 helfen kann, den Prozessor-Zyklus besser zu verstehen?
☒ Ja, weil …

☐ Nein, weil …

Was fehlt deiner Meinung nach noch, um den Prozessor-Zyklus besser zu verstehen?

- jump wäre auch spannend als Instruktion, rekursive Funktion
- Stack wäre auch spannend. 25% Studierende hat mühe. Stack pointer und Frame pointer.

Gibt es etwas, das du uns sonst noch sagen möchtest?

-

Usability Test Protokoll S003

Datum: 04. November 2020

Vorame: S003

Vor der Vorführung

Erkenntnisse aus dem Gespräch

- Er meint es sei alles klar.
- “Nach der Berechnung schreibt man die Daten zurück ins Memory.”
- Kennt die Grösse vom RAX Register.
- Kennt den Speicherbus und versteht register.

Während der Vorführung

Protokoll zur Vorführung

Notizen zu Aussagen und zum Verhalten des Kandidaten

Assembly

Intro animation

- Findet die Animation sei zu schnell.
- Sieht Cpu und Memory und findet Cpu Cycle “Objekt” unverständlich.

Get instruction

Execute instruction

- “Beim Popup Dialog sollte man mov src und dest ausschreiben, dass es klar ist was gemeint ist.”
- Ascii interpretation Popup sei unklar

Increment IP

- Versteht erhöhung des Instruction Pointer um länge der Instruction.

Fragestellungen zum Design

Spezifische Fragen, um unsere Design Entscheidungen auf Verständlichkeit zu testen.

Graue Ovale

- **Frage**: Für was stehen die grauen Ovale im Design?
  - Richtige Antwort: Ein Byte.
  - Antwort des Kandidaten: “Byte in hex”
- War unsere Absicht klar: **Ja**

Address eines Bytes

- **Frage**: Im Memory auf der zweiten Zeile und in der dritten Spalte, welche Adresse hat das graue Oval “1D”?
  - Richtige Antwort: 000C
  - Antwort des Kandidaten: Versteht Adressformat nicht komplett. Wenn nach Adresse gefragt rechnet er daten nach dezimal um.
- War unsere Absicht klar: **Nein**

Knöpfe

- **Frage**: Wenn du nun in der Animation einen Schritt weitergehen möchtest, wohin würdest du klicken?
- Unsere Absicht:
  - Step Knopf: Geht zum nächsten Animationsschritt.
  - Play Knopf: Startet die Animation, ohne bei einem Schritt zu halten.
- Interpretation des Kandidaten:
  - Step Knopf: Meint es überspringe Instruktion.
  - Play Knopf: Geht zum nächsten Animationsschritt.
- War unsere Absicht klar: **Nein**

Nach der Vorführung

Anmerkungen

- Findet die Simulation nicht komplett ohne Speicher Bus.
- Findet design ok, aber es sollte mehr nach physischen Elemente aussehen, wie eine CPU oder ein Speicherelement. Findet es sieht zu wenig technisch aus.
- CPU Cycle als Objekt sei unverständlich.
- Würde ASCII und Assembly interpretation nicht immer anzeigen, oder anders, zum Beispiel Assembly in CPU Cycle Box.
- Findet ~25% der Studierenden haben Stack Pointer und Stack nicht verstanden.
- Versteht frame pointer.
- Findet ~50% der Studierenden haben dem Frame Pointer nicht verstanden.

Verbesserungen im Verständnis

- Versteht besser, dass es sich bei “[1000]” um Inhalt handelt
- Versteht die Register Funktion ein wenig besser.

Wünsche

- Möchte gerne den Speicher Bus über einen Knopf einblenden.
- Würde gerne Sprünge sehen mit dem Stack Pointer.
- Würde gerne eigene Instruktionen ausführen

Usability Test S004

Datum: 04. November 2020

Im Rahmen unserer Studienarbeit erstellen wir einen Prozessor-Simulator, welcher in Zukunft den Bsys1 Studierenden helfen soll, die einzelnen Schritte des Prozessor-Zyklus besser zu verstehen. Um dieses Ziel zu erreichen, sind wir auf deine Hilfe angewiesen. Wir möchten herausfinden, wo genau die Verständnisprobleme liegen und ob unsere Applikation dabei helfen kann.

Herzlichen Dank für deine Mithilfe!
Yves Boillat & Eliane Schmidli

Angaben zu deiner Person

Vorname: S004...
In welchem Semester bist du? 1….

Was hast du vor dem Studium gemacht?
☒ Lehre als Informatiker/Informatikerin
☐ Kaufmännische Lehre
☐ Gymnasium
☐ Anderes: ……...

Wie würdest du deine Vorkenntnisse für das Informatikstudium einschätzen?
☐ Sehr gut, ich bin unterfordert im Informatikstudium
☒ Gut
☐ Mittelmässig
☐ Ich habe vor dem Studium den Programmiervorkurs besucht

Wie schätzt du deine Kenntnisse über den Prozessor-Zyklus ein?
☐ Was in Bsys1 zum Prozessor-Zyklus unterrichtet wird, wusste ich bereits vorher schon.
☒ Ich konnte den Vorlesungen zum Prozessor-Zyklus gut folgen und habe das meiste verstanden
☐ Ich habe die Vorlesungen von Bsys1 zum Prozessor-Zyklus einigermassen verstanden.
☐ Was ist ein Prozessor-Zyklus?

Falls dir gerade etwas in den Sinn kommt, was du nicht verstanden hast, schreibe das bitte hierhin:

Fragen zum Prozessor-Zyklus Teil 1

Beschreibe den Ablauf des Prozessor-Zyklus:
- holt sich aktuelle Instruktion
- ladet memory Offstet von IP
- decoding phase,
- Instruktion mit flags/addressing Bytes
- Microcode mov etc aus kleineren Instruktionen
- mov: kopieren oder noch mehrere Zyklen je nach Operation

Woher holt der Prozessor die Instruktionen, welche du in dein Programm geschrieben hast?
- main memory, oder Pipeline / Cache

Beschreibe schrittweise was der Prozessor bei der folgenden Assembly Instruktion macht:
mov rax, [0x1000]
- lädt 8 Bytes von Adresse 1000
- Rax = 8 Bytes
- kopiert Werte in Rax

Mit was für einem Code/Sprache arbeitet der Prozessor?
- x86 assembly Sprache / micro Code

Was ist ein Register?
- memory location direkt im Prozessor, schneller als Arbeitsspeicher

---

Fragen zum Prozessor-Zyklus Teil 2

Beantworte hier noch einmal die Fragen von oben:
Beschreibe den Ablauf des Prozessor-Zyklus:
- holt Instruktion aus Memory
- führt Instruktion aus
- schaut wie gross Instruktion war und updatet IP

Woher holt der Prozessor die Instruktionen, welche du in dein Programm geschrieben hast?
- Memory

Beschreibe schrittweise was der Prozessor bei der folgenden Assembly Instruktion macht:
mov rax, [0x1000]
- IP zeigt auf Instruktion, Prozessor lädt diese
- Prozessor kopiert Wert an Adresse 1000 in Rax
- vergrössert IP

Mit was für einem Code/Sprache arbeitet der Prozessor?
- Maschinencode, Assembler

Was ist ein Register?
- memory Feld innerhalb Prozessor welches effizient und schnell ist.

Fragen zum Design

Gefällt dir unser Design für den Prozessor-Simulator?
☒ Ja, finde ich toll
☐ Ich finde es okay
☐ Naja…
☐ Ich mag das Design nicht
Verbesserungsvorschlag/Kommentar:

- shadows und border radius zu stark
- Farben sind gut
- Assembly Interpretation ist klar, aber wäre besser wenn es direkt daneben steht

Hast du dich in unserem Design gut zurechtgefunden?
☒ Ja, weil….

☐ Nein, weil ....

Kannst du dir vorstellen, dass unser Prozessor-Simulator den Studierenden aus Bsys1 helfen kann, den Prozessor-Zyklus besser zu verstehen?
☒ Ja, weil …

☐ Nein, weil …

Was fehlt deiner Meinung nach noch, um den Prozessor-Zyklus besser zu verstehen?

- color code was zeigt mov was zeigt Rax von Instruktion.

Gibt es etwas, das du uns sonst noch sagen möchtest?
-

Usability Test Protokoll S004

Datum: 04. November 2020

Vorame: S004

Vor der Vorführung

Erkenntnisse aus dem Gespräch

- Er fühlt sich sicher
- Kennt Memory und Caching
- Weiss, dass Rax 8 Byte Register

Während der Vorführung

Protokoll zur Vorführung

Notizen zu Aussagen und zum Verhalten des Kandidaten

Assembly

Intro animation

- Er sieht Hex Dump von Memory
- Oben sind Instruction
- Unten ist Program Memory
- Versteht alles

Get instruction

- Versteht Vorgang
- Unklar: “Wie weiss CPU, wie Lange Instruktion ist. CPU muss ja zuerst Opcode anschauen”

Execute instruction

- 8 Bytes, weil Register Rax 8 Byte

Increment IP

-

Fragestellungen zum Design

Spezifische Fragen, um unsere Design Entscheidungen auf Verständlichkeit zu testen.

Graue Ovale

- **Frage**: Für was stehen die grauen Ovale im Design?
    - Richtige Antwort: Ein Byte.
    - Antwort des Kandidaten: 1 Byte
- War unsere Absicht klar: **Ja**

Address eines Bytes

- **Frage**: Im Memory auf der zweiten Zeile und in der dritten Spalte, welche Adresse hat das graue Oval “1D”?
    - Richtige Antwort: 000C
    - Antwort des Kandidaten: 000C
- War unsere Absicht klar: **Ja**

Knöpfe

- **Frage**: Wenn du nun in der Animation einen Schritt weitergehen möchtest, wohin würdest du klicken?
- Unsere Absicht:
    - Step Knopf: Geht zum nächsten Animationsschritt.
    - Play Knopf: Startet die Animation, ohne bei einem Schritt zu halten.
- Interpretation des Kandidaten:
    - Step Knopf:Nächster Schritt
    - Play Knopf: Startet Animation ohne Unterbrechung
- War unsere Absicht klar: **Ja**

Nach der Vorführung

Anmerkungen

- Hätte lieber Übersetzung direkt neben dem Memory ohne separate Komponente
- Design ist hübsch
- Border Radius und Shadow etwas too much

Verbesserungen im Verständnis

-

Wünsche

- Genaue Erklärung wie CPU weiss, wie lange Instruktion ist
- Color Coding für Instruktion, welches Byte steht zum Beispiel für mov

Usability Test S005

Datum: 04. November 2020

Im Rahmen unserer Studienarbeit erstellen wir einen Prozessor-Simulator, welcher in Zukunft den Bsys1 Studierenden helfen soll, die einzelnen Schritte des Prozessor-Zyklus besser zu verstehen. Um dieses Ziel zu erreichen, sind wir auf deine Hilfe angewiesen. Wir möchten herausfinden, wo genau die Verständnisprobleme liegen und ob unsere Applikation dabei helfen kann.

Herzlichen Dank für deine Mithilfe!
Yves Boillat & Eliane Schmidli

Angaben zu deiner Person

Vorname: S005.
In welchem Semester bist du? ……1.

Was hast du vor dem Studium gemacht?
☐ Lehre als Informatiker/Informatikerin
☒ Kaufmännische Lehre (Mediamatiker)
☐ Gymnasium
☐ Anderes: …

Wie würdest du deine Vorkenntnisse für das Informatikstudium einschätzen?
☐ Sehr gut, ich bin unterfordert im Informatikstudium
☐ Gut
☒ Mittelmässig
☒ Ich habe vor dem Studium den Programmiervorkurs besucht

Wie schätzt du deine Kenntnisse über den Prozessor-Zyklus ein?
☐ Was in Bsys1 zum Prozessor-Zyklus unterrichtet wird, wusste ich bereits vorher schon.
☐ Ich konnte den Vorlesungen zum Prozessor-Zyklus gut folgen und habe das meiste verstanden
☒ Ich habe die Vorlesungen von Bsys1 zum Prozessor-Zyklus einigermassen verstanden.
☐ Was ist ein Prozessor-Zyklus?

Falls dir gerade etwas in den Sinn kommt, was du nicht verstanden hast, schreibe das bitte hierhin:
-

Fragen zum Prozessor-Zyklus Teil 1

Beschreibe den Ablauf des Prozessor-Zyklus:

- Durchlauf von einer Anweisung?

Woher holt der Prozessor die Instruktionen, welche du in dein Programm geschrieben hast?

- bekommt er von einem Bus

Beschreibe schrittweise was der Prozessor bei der folgenden Assembly Instruktion macht:
mov rax, [0x1000]

- schreibt hex Wert 1000 in Register Rax

Mit was für einem Code/Sprache arbeitet der Prozessor?

- Maschinencode

Was ist ein Register?
-

Fragen zum Prozessor-Zyklus Teil 2

Beantworte hier noch einmal die Fragen von oben:
Beschreibe den Ablauf des Prozessor-Zyklus:

- get instruction
- execute
- increment ip

Woher holt der Prozessor die Instruktionen, welche du in dein Programm geschrieben hast?
- instruction pointer

Beschreibe schrittweise was der Prozessor bei der folgenden Assembly Instruktion macht:
mov rax, [0x1000]
- hex Wert 1000 ins Register Rax kopieren

Mit was für einem Code/Sprache arbeitet der Prozessor?
- Maschinencode

Was ist ein Register?
- Speicherbereich, im Ram

Fragen zum Design

Gefällt dir unser Design für den Prozessor-Simulator?
☐ Ja, finde ich toll
☒ Ich finde es okay
☐ Naja…
☐ Ich mag das Design nicht
Verbesserungsvorschlag/Kommentar:

- zu Beginn war die Frage was ist das, mittlerweile komme ich klar

Hast du dich in unserem Design gut zurechtgefunden?
☒ Ja, weil….

- nach einer gewissen Zeit ja, ganz am Anfang nicht
- mehr Beschreibung mit Text oder Ton

☐ Nein, weil ....

Kannst du dir vorstellen, dass unser Prozessor-Simulator den Studierenden aus Bsys1 helfen kann, den Prozessor-Zyklus besser zu verstehen?
☒ Ja, weil …

- ja bestimmt

☐ Nein, weil …

Was fehlt deiner Meinung nach noch, um den Prozessor-Zyklus besser zu verstehen?
-

Gibt es etwas, das du uns sonst noch sagen möchtest?
-

Usability Test Protokoll S005

Datum: 04. November 2020

Vorame: S005

Vor der Vorführung

Erkenntnisse aus dem Gespräch

- Hat Mühe im Studium
- Prozessor-Zyklus: rät richtig, dass es sich um den Durchlauf einer Anweisung handelt
- Kennt Maschinen Code
- Kann nicht direkt erklären was Register ist
- “Schreibt hex Wert 1000 ins Register”

Während der Vorführung

Protokoll zur Vorführung

Notizen zu Aussagen und zum Verhalten des Kandidaten

Assembly

Intro animation

- Alles auf einmal
- Block CPU hat ihn am Anfang verwirrt
- Meint Instruction Pointer müsse auf Adresse 1000 zeigen, da die aktuell Instruktion mov rax [1000] ist

Get instruction

- Versteht nun warum IP auf Adresse 0000 zeigt und nicht 1000

Execute instruction

- Hex Wert wurde in Rax geschrieben

Increment IP

- Ihm gefällt die Animation

Fragestellungen zum Design

Spezifische Fragen, um unsere Design Entscheidungen auf Verständlichkeit zu testen.

Graue Ovale

- **Frage**: Für was stehen die grauen Ovale im Design?
  - Richtige Antwort: Ein Byte.
  - Antwort des Kandidaten: Sagt zuerst Adresse, dann 1 Byte
- War unsere Absicht klar: **Ja**

Address eines Bytes

- **Frage**: Im Memory auf der zweiten Zeile und in der dritten Spalte, welche Adresse hat das graue Oval “1D”?
  - Richtige Antwort: 000C
  - Antwort des Kandidaten: seufzt, meint dann +2
- War unsere Absicht klar: **Ja**

Knöpfe

- **Frage**: Wenn du nun in der Animation einen Schritt weitergehen möchtest, wohin würdest du klicken?
- Unsere Absicht:
  - Step Knopf: Geht zum nächsten Animationsschritt.
  - Play Knopf: Startet die Animation, ohne bei einem Schritt zu halten.
- Interpretation des Kandidaten:
  - Step Knopf: überspringt Instruction
  - Play Knopf: Nächster Schritt
- War unsere Absicht klar: **Nein**

Nach der Vorführung

Anmerkungen

- Spricht auch beim zweiten Durchlauf noch von Hex Wert statt Adresse 1000
- Er findet die Grafik helfe
- Design findet er okey, da er sich nicht sofort zurechtgefunden hat
- Am Anfang wäre ein Text noch hilfreich, der beschreibt, was was ist
- Einer der Knöpfe muss angepasst werden

Verbesserungen im Verständnis

- Meint auch in zweiter Fragerunde, dass [1000] für Wert und nicht Adresse steht

Wünsche

Usability Test S006

Datum: 04. November 2020

Im Rahmen unserer Studienarbeit erstellen wir einen Prozessor-Simulator, welcher in Zukunft den Bsys1 Studierenden helfen soll, die einzelnen Schritte des Prozessor-Zyklus besser zu verstehen. Um dieses Ziel zu erreichen, sind wir auf deine Hilfe angewiesen. Wir möchten herausfinden, wo genau die Verständnisprobleme liegen und ob unsere Applikation dabei helfen kann.

Herzlichen Dank für deine Mithilfe!
Yves Boillat & Eliane Schmidli

Angaben zu deiner Person

Vorname: S006 (weiblich)
In welchem Semester bist du? …1.

Was hast du vor dem Studium gemacht?
☐ Lehre als Informatiker/Informatikerin
☐ Kaufmännische Lehre
☐ Gymnasium
☒ Anderes: Elektroniker Lehre...

Wie würdest du deine Vorkenntnisse für das Informatikstudium einschätzen?
☐ Sehr gut, ich bin unterfordert im Informatikstudium
☒ Gut
☐ Mittelmässig
☐ Ich habe vor dem Studium den Programmiervorkurs besucht

Wie schätzt du deine Kenntnisse über den Prozessor-Zyklus ein?
☐ Was in Bsys1 zum Prozessor-Zyklus unterrichtet wird, wusste ich bereits vorher schon.
☒ Ich konnte den Vorlesungen zum Prozessor-Zyklus gut folgen und habe das meiste verstanden
☐ Ich habe die Vorlesungen von Bsys1 zum Prozessor-Zyklus einigermassen verstanden.
☐ Was ist ein Prozessor-Zyklus?

Falls dir gerade etwas in den Sinn kommt, was du nicht verstanden hast, schreibe das bitte hierhin:

Fragen zum Prozessor-Zyklus Teil 1

Beschreibe den Ablauf des Prozessor-Zyklus:

- program counter -> aktueller Stand
- geht an Adresse, lädt Instruktion von Speicher in Register
- liest Operator aus und dann Operanden aus Instruktion
- sucht richtige Operation aus
- führt Operation aus
- schreibt Resultat in Hauptspeicher
- erhöht Pc

Woher holt der Prozessor die Instruktionen, welche du in dein Programm geschrieben hast?
- aus dem Hauptspeicher

Beschreibe schrittweise was der Prozessor bei der folgenden Assembly Instruktion macht:
mov rax, [0x1000]

- geht in Hauptspeicher an Adresse 1000 hex, kopiert Wert in Register Rax und speichert ihn zwischen, evtl noch Zwischenregister

Mit was für einem Code/Sprache arbeitet der Prozessor?

- Sequenzen welche er aus Assembler macht
- Maschinen Sequenzen

Was ist ein Register?

- Speicherbereich des Prozessor, wo er Werte zwischenspeichern kann ca. 16

Fragen zum Prozessor-Zyklus Teil 2

Beantworte hier noch einmal die Fragen von oben:
Beschreibe den Ablauf des Prozessor-Zyklus:

- holt Instruktion aus Hauptspeicher
- führt Instruktion aus
- erhöht Pc

Woher holt der Prozessor die Instruktionen, welche du in dein Programm geschrieben hast?
- Hauptspeicher

Beschreibe schrittweise was der Prozessor bei der folgenden Assembly Instruktion macht:
mov rax, [0x1000]
- holt Adresse 1000 hex kopiert in Register

Mit was für einem Code/Sprache arbeitet der Prozessor?
- Assembler

Was ist ein Register?
- Zwischenspeicher im Prozessor

Fragen zum Design

Gefällt dir unser Design für den Prozessor-Simulator?
☒ Ja, finde ich toll
☐ Ich finde es okay
☐ Naja…
☐ Ich mag das Design nicht
Verbesserungsvorschlag/Kommentar:

- Benennung Prozessor <-> CPU
- viele Sachen, aber reduziert aufs Wichtigste

Hast du dich in unserem Design gut zurechtgefunden?
☒ Ja, weil….

- zuerst nicht, aber nach 3 Sekunden,
- Kasten sind schnell klar

☐ Nein, weil ....

Kannst du dir vorstellen, dass unser Prozessor-Simulator den Studierenden aus Bsys1 helfen kann, den Prozessor-Zyklus besser zu verstehen?
☒ Ja, weil …

- ja ich denke schon

☐ Nein, weil …

Was fehlt deiner Meinung nach noch, um den Prozessor-Zyklus besser zu verstehen?

- Adressen in hex
- je nachdem mit Flags, alle wichtig

Gibt es etwas, das du uns sonst noch sagen möchtest?

- interessant

Usability Test Protokoll S006

Datum: 04. November 2020

Vorame: S006

Vor der Vorführung

Erkenntnisse aus dem Gespräch

- Findet Hardware Dinge verständlich
- Beschreibt Instruktionsausführung extrem detailliert
- Ist sich nicht sicher, ob beim Mov noch Zwischenregister verwendet werden neben Rax
- Erwähnt, das es ca. 16 Register gibt

Während der Vorführung

Protokoll zur Vorführung

Notizen zu Aussagen und zum Verhalten der Kandidatin

Assembly

- Merkt, dass Adressen im Assembly Code noch mit 0x beginnen müssen
- Versteht Assembly

Intro animation

- Erwähnt CPU Abschnit
- Aktuelle Instruktion, welche noch leer ist
- Register, welche auch noch leer sind
- Ist verwirrt über Ascii Interpretation und fragt sich, woher diese kommt und warum sie dort ist.

Get instruction

- Wir hatten technische Übertragungsprobleme
- Hat verstanden was passiert ist

Execute instruction

- Findet, dass Ascii Interpretation nun Sinn macht

Increment IP

- Erkennt, dass der Pointer um die Länge der Instruktion erhöht wird
- Überall einen Schritt weiter

Fragestellungen zum Design

Spezifische Fragen, um unsere Design Entscheidungen auf Verständlichkeit zu testen.

Graue Ovale

- **Frage**: Für was stehen die grauen Ovale im Design?
  - Richtige Antwort: Ein Byte.
  - Antwort der Kandidatin: Ein Byte
- War unsere Absicht klar: **Ja**

Address eines Bytes

- **Frage**: Im Memory auf der zweiten Zeile und in der dritten Spalte, welche Adresse hat das graue Oval “1D”?
  - Richtige Antwort: 000C
  - Antwort der Kandidatin: 000C
- War unsere Absicht klar: **Ja**

Knöpfe

- **Frage**: Wenn du nun in der Animation einen Schritt weitergehen möchtest, wohin würdest du klicken?
- Unsere Absicht:
  - Step Knopf: Geht zum nächsten Animationsschritt.
  - Play Knopf: Startet die Animation, ohne bei einem Schritt zu halten.
- Interpretation der Kandidatin:
  - Step Knopf: nächster Schritt
  - Play Knopf: Startet Animation ohne Unterbrechung
- War unsere Absicht klar: **Ja**

Nach der Vorführung

Anmerkungen

- Fragt sich, ob für alle klar ist, dass die Begriffe CPU und Prozessor dasselbe bedeuten
- Es ist verständlich
- Es hat viele Dinge, aber nicht zuviele
- Nach ein paar Sekunden findet man sich zurecht

- Besser als in der Übung
- “Schön, dass es vereinfacht dargestellt ist”
- Adressen müssen mit 0x beginnen

Verbesserungen im Verständnis

- Beschreibt Instruktion weniger detailliert

Wünsche

-

Usability Test S007

Datum: 04. November 2020

Im Rahmen unserer Studienarbeit erstellen wir einen Prozessor-Simulator, welcher in Zukunft den Bsys1 Studierenden helfen soll, die einzelnen Schritte des Prozessor-Zyklus besser zu verstehen. Um dieses Ziel zu erreichen, sind wir auf deine Hilfe angewiesen. Wir möchten herausfinden, wo genau die Verständnisprobleme liegen und ob unsere Applikation dabei helfen kann.

Herzlichen Dank für deine Mithilfe!
Yves Boillat & Eliane Schmidli

Angaben zu deiner Person

Vorname: S007...
In welchem Semester bist du? 1. …

Was hast du vor dem Studium gemacht?
☒ Lehre als Informatiker/Informatikerin
☐ Kaufmännische Lehre
☐ Gymnasium
☐ Anderes: ……...

Wie würdest du deine Vorkenntnisse für das Informatikstudium einschätzen?
☐ Sehr gut, ich bin unterfordert im Informatikstudium
☐ Gut
☒ Mittelmässig
☐ Ich habe vor dem Studium den Programmiervorkurs besucht

Wie schätzt du deine Kenntnisse über den Prozessor-Zyklus ein?
☐ Was in Bsys1 zum Prozessor-Zyklus unterrichtet wird, wusste ich bereits vorher schon.
☐ Ich konnte den Vorlesungen zum Prozessor-Zyklus gut folgen und habe das meiste verstanden
☒ Ich habe die Vorlesungen von Bsys1 zum Prozessor-Zyklus einigermassen verstanden.
☐ Was ist ein Prozessor-Zyklus?

Falls dir gerade etwas in den Sinn kommt, was du nicht verstanden hast, schreibe das bitte hierhin:

- Grösstes Problem: Speicher, Register

Fragen zum Prozessor-Zyklus Teil 1

Beschreibe den Ablauf des Prozessor-Zyklus:

- Datei vom Linker mit Code
- include Variablen einlesen
- Zeile für Zeile ausführen

Woher holt der Prozessor die Instruktionen, welche du in dein Programm geschrieben hast?
- IP zeigt auf Zeile wo Instruktion ist.
- in Speicher und Register

Beschreibe schrittweise was der Prozessor bei der folgenden Assembly Instruktion macht:
mov rax, [0x1000]
- Prozessor sieht das mov ist
- kopiert Wert wo in Adresse 0x1000 liegt
- kopiert in Rax, Register 64bit

Mit was für einem Code/Sprache arbeitet der Prozessor?
- Assembler

Was ist ein Register?
- kleiner Speicherblock im Arbeitsspeicher

Fragen zum Prozessor-Zyklus Teil 2

Beantworte hier noch einmal die Fragen von oben:
Beschreibe den Ablauf des Prozessor-Zyklus:
- Prozessor holt Instruktion aus Register wo IPzeigt
- führt Instruktion aus
- IP geht auf nächste Instruktion anhand Grösse

Woher holt der Prozessor die Instruktionen, welche du in dein Programm geschrieben hast?
- Register im ram wo IP hinzeigt

Beschreibe schrittweise was der Prozessor bei der folgenden Assembly Instruktion macht:
mov rax, [0x1000]
- Wert an 1000 wird kopiert in Rax

Mit was für einem Code/Sprache arbeitet der Prozessor?
- Assembler

Was ist ein Register?
- Speicherbereich

Fragen zum Design

Gefällt dir unser Design für den Prozessor-Simulator?
☒ Ja, finde ich toll
☐ Ich finde es okay
☐ Naja…
☐ Ich mag das Design nicht
Verbesserungsvorschlag/Kommentar:

- verständlich,
- Register nicht klar von Farbe, dass sie zur Cpu gehören, sollte anderes blau sein

Hast du dich in unserem Design gut zurechtgefunden?
☒ Ja, weil…. nach gewisser zeit

☐ Nein, weil ....

Kannst du dir vorstellen, dass unser Prozessor-Simulator den Studierenden aus Bsys1 helfen kann, den Prozessor-Zyklus besser zu verstehen?
☒ Ja, weil …

☐ Nein, weil …

Was fehlt deiner Meinung nach noch, um den Prozessor-Zyklus besser zu verstehen?

- Stack fehlt, das wäre noch hilfreich

Gibt es etwas, das du uns sonst noch sagen möchtest?
-

Usability Test Protokoll S007

Datum: 04. November 2020

Vorame: S007

Vor der Vorführung

Erkenntnisse aus dem Gespräch

- Sein grösstes Problem ist Uterscheidung von Speicher und Register
- Verwechselt Adressen und Wert konstant ([1000] und 1000)
- Beschreibt C Tool Chain statt Prozessor-Zyklus
- Instruction liegt im Register oder Memory
- Rax Register ist 64 bit
- Register liegen im Arbeitsspeicher

Während der Vorführung

Protokoll zur Vorführung

Notizen zu Aussagen und zum Verhalten des Kandidaten

Assembly

- Beschreibt add als Addition von Adressen 1000 + 1008

Intro animation

- Register mit Flags in CPU
- Beschreibt Cycle richtig
- Sagt den Adresszeilen in Memory Register
- Namen und Farben helfen laut ihm fürs Verständnis

Get instruction

- CPU liest aus IP und holt Instruktion

Execute instruction

- Wert von Register 1000 in Rax geschrieben

Increment IP

- CPU liest Länge von Instruktion und erhöht Pointer

Fragestellungen zum Design

Spezifische Fragen, um unsere Design Entscheidungen auf Verständlichkeit zu testen.

Graue Ovale

- **Frage**: Für was stehen die grauen Ovale im Design?
  - Richtige Antwort: Ein Byte.
  - Antwort des Kandidaten: Ein Byte
- War unsere Absicht klar: **Ja**

Address eines Bytes

- **Frage**: Im Memory auf der zweiten Zeile und in der dritten Spalte, welche Adresse hat das graue Oval “1D”?
  - Richtige Antwort: 000C
  - Antwort des Kandidaten: Ist verwirrt und versucht Inhalt in Dezimalwert umzurechnen
- War unsere Absicht klar: **Nein**

Knöpfe

- **Frage**: Wenn du nun in der Animation einen Schritt weitergehen möchtest, wohin würdest du klicken?
- Unsere Absicht:
  - Step Knopf: Geht zum nächsten Animationsschritt.
  - Play Knopf: Startet die Animation, ohne bei einem Schritt zu halten.
- Interpretation des Kandidaten:
  - Step Knopf: Überspringen
  - Play Knopf: Nächster schritt
- War unsere Absicht klar: **Nein**, wahrscheinlich weil er noch fast nie einen Debugger genutzt hat

Nach der Vorführung

Anmerkungen

- Findet Design gut
- Die rote Farbe der Register führt dazu, dass man nicht merkt, dass sie zur CPU gehören
- Nach gewisser Zeit hat er sich zurechtgefunden

- Insgesamt hat die Animation ihm geholfen

Verbesserungen im Verständnis

-

Wünsche

- Stack sollte noch ersichtlich sein