DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

Konzeptionierung eines Simulators für 8-bit Prozessoren

Studienarbeit

Bachelor of Science

Studiengang Informationstechnik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

Andreas Schmider, Nico Schrodt

Abgabedatum 28. Dezember 2021

Bearbeitungszeitraum	2 Semester
Kurs	TINF19B3
Betreuer der Ausbildungsfirma	Prof. Dr.-Ing. Kai Becher

Erklärung

Wir versichern hiermit, dass wir unsere Studienarbeit mit dem Thema:

Konzeptionierung eines Simulators für 8-bit Prozessoren

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Karlsruhe, 28. Dezember 2021

Ort, Datum Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

ALU - Arithmetic Logic Unit CISC - Complex Instruction Set Computer MISD - Multiple-instruction stream single-data stream MIMD - Multiple-instruction stream multiple-data stream RISC - Reduced Instruction Set Computer SISD - Single-instruction stream single-data stream SIMD - Single-instruction stream multiple-data stream

1 Einführung

Dieses Kapitel befasst sich vorwiegend mit relevanten Grundlagen der Arbeit. Unter anderem wird das Ziel spezifiziert, elementare Aspekte der Arbeitsweise eines Prozessors werden erläutert und die verschiedenen Werkzeuge mit denen das Ziel realisiert wird werden aufgeführt.

1.1 Ziel der Arbeit

In dieser Arbeit soll ein Simulationsprogramm geschrieben werden, mit dem mehrere unterschiedliche 8-Bit Prozessoren simuliert werden können. Dazu sollen die Grundlegenden Eigenschaften in kurzen Lernprogrammen erläutert werden. Ebenfalls soll es eine interaktive Einweisung geben wie der Simulator verwendet werden kann.

1.2 Theoretische Grundlagen

Als Vorbereitung für die Implementierung der verschiedenen Prozessoren werden einige allgemeingültige Architekturprinzipien eines Prozessors analysiert.

1.2.1 Architektur eines Prozessors

Der fundamentale Aufbau eines Prozessors lässt sich in folgende Bausteine einteilen:

Rechenwerk

Das Rechenwerk ist die zentrale Einheit mit der eingehende Befehle verarbeitet werden. Es erhält Werte aus dem Speicher und führt damit in der ALU Operationen durch. Zum Rechenwerk dazugehörig sind auch Hilfsregister die beispielsweise als naher Zwischenspeicher fungieren.

Steuerwerk

Das Steuerwerk ist für die korrekte Abarbeitung von Befehlen zuständig. Es besteht aus dem Befehlsdekoder, dem Befehlszähler und einem Statusregister.

Programmspeicher

Der Programmspeicher eines Prozessors enthält die einzelnen Befehle, welche vom Befehlsdekoder dekodiert werden. Dabei wird der Befehl verwendet der an der vom Befehlszähler spezifizierten Stelle im Speicher steht.

Ein-/Ausgabewerk

Das Ein-/Ausgabewerk ist für die Kommunikation des Prozessors mit anderen Systemkomponenten verantwortlich.

1.2.2 Befehlsformate

Für einen Prozessor wird zwischen vier verschiedenen Befehlsformaten unterschieden. Diese beziehen sich auf die Anzahl der Adressen, welche der ALU bei Beginn einer Operation zur Verfügung gestellt werden. **0-Adress-Anweisungen**

Von einem Akkumulator (Stack) wird der Wert der zwei obersten Adressfeldern entnommen und die Operation wird auf diese ausgeführt. Der Speicherort des Ergebnisses ist ebenfalls vorbestimmt. Daher werden keine Adressen benötigt zum ausführen von Operationen.

1-Adress-Anweisungen

Wie bei einer 0-Adress-Anweisung wird ein Wert aus dem Akkumulator entnommen. Ein zweiter Wert wird aus dem Speicher der übergebenen Adresse entnommen. Der Speicherort des Ergebnisses ist nach wie vor fest.

2-Adress-Anweisungen

txt

3-Adress-Anweisungen

txt

1.2.3 CISC und RISC

komplettes Kapitel aus Buch
Ein Prozessor unterstützt immer nur eine gewisse Menge an Befehlen, diese werden Instruction Set genannt. Heutzutage gibt es zwei grundlegende Prozessorarchitekturen, Complex Instruction Set Computer (CISC) und Reduced Instruction Set Computer (RISC). Früher konnten Prozessoren genau einer dieser Gruppen zugeordnet werden, allerdings ist das bei den heutigen Prozessoren nicht mehr möglich, da sowohl RISC als auch CISC Befehle dem Prozessor zur Verfügung gestellt werden um die Vorteile von beiden zu haben.

Bei CISC-Prozessoren wird versucht soviel wie möglich in einem Befehl ausführen zu können. So gibt es viele verschieden Befehle, die auch unterschiedlich viel Zeit benötigen. Dadurch wird es aber auch möglich, komplexere Befehle direkt in der Hardware zu berechnen. Bei diesen Befehlen gibt es auch einige Adressierungsarten mehr als bei RISC-Prozessoren. Für vorbestimmte Aufgaben gibt es auch eigene Register, die nur dafür verwendet werden und davon auch nur wenige. Der Nachteil bei CISC-Prozessoren ist, dass die eigenen Befehle erst noch durch ein Mikroprogramm interpretiert werden müssen und dieses die komplexen Befehle in mehrere kleine Befehle aufteilen muss, welche erst dann vom Prozessor bearbeitet werden können. Dies kostet etwas mehr Zeit und verlangsamt die Ausführung. Die Mikroprogramme, die dafür verwendet werden, werden in einem kleinem Read-only Memory (ROM) gespeichert.

Bei RISC-Prozessoren wird versucht mit nur wenigen, kleinen Befehlen auszukommen. Diese sind wiederum sehr schnell, da sie meist fest verdrahtet sind, müssen aber mit anderen kombiniert werden um die Komplexität eines einzigen CISC-Befehls zu erreichen. Im Gegensatz zu CISC-Prozessoren besitzen RISC-Prozessoren viele Register die frei verwendbar sind und nicht für speziellen Operationen bestimmt sind. Ebenso können die meisten Befehle in nur einem einzigen Arbeitsschritt ausgeführt werden.

Da bei CISC-Prozessoren mit nur einem Befehl viel berechnet werden kann, sind diese optimal für Übersetzer oder Interpreter geeignet. Bei der Entwicklung können dann einzelne komplexe Befehle verwendet werden anstatt von vielen kleinen. Jedoch können RISC-Prozessoren schneller Befehle ausführen, da:

- es nur wenige Befehle gibt und diese schnell decodiert werden können
- die Befehle mithilfe von Pipelines effizienter abgearbeitet werden können
- kein Mikroprogramm die einzelnen Befehle erst noch interpretieren muss

1.2.4 Parallelität nach Flynn

Zur Unterscheidung von Parallelität wurden 1966 von Michael Flynn vier Arten definiert [Flynn 949]:

- Single-instruction stream, single-data stream (SISD)
- Single-instruction stream, multiple-data stream (SIMD)
- Multiple-instruction stream, single-data stream (MISD)
- Multiple-instruction stream, multiple-data stream (MIMD)

SISD

Diese Beschreibung trifft auf die einfachen Einprozessorsysteme zu. Dabei kann immer nur eine Operation gleichzeitig ausgeführt werden und diese werden in nur einer möglichen Reihenfolge aus einem Daten-Strom abgearbeitet.

SIMD

Bei SIMD werden Pipelines eingesetzt, die es ermöglichen mehrere korrekte Abfolgen von Programmbefehlen auszuführen. So können unterschiedliche Programme in sich selber in der richtigen Reihenfolge aber mit anderen Programmen abwechselnd ausgeführt werden.

MISD

Diese Variante scheint auf Anhieb keinen effizienten Nutzen zu besitzen, da mehrere Prozessoren alle die gleichen Befehle ausführen, die aus einem Daten-Strom stammen. Dies kann aber dazu verwendet werden um die Korrektheit durch Redundanzen zu bestätigen.

MIMD

Diese Architektur verwendet mehrere Prozessoren und mehrere Daten-Ströme. Heutzutage ist dies unter dem Begriff Mehrprozessorsysteme bekannt. So werden für jeden einzelnen Prozessor ein Daten-Strom erzeugt der unabhängig von den anderen Prozessoren arbeiten kann. Dabei ist es für die Prozessoren aber trotzdem möglich die Daten der anderen Prozessoren zu nutzen. Nur durch MIMD oder MISD, also die Ausführung mit mehreren unabhängigen Prozessoren, ist es möglich Programme

tatsächlich parallel ablaufen zu lassen. Mit SISD oder SIMD sind nur quasi parallele Ausführungen möglich.

2 Projektplanung

Platzhalter

2.1 Zeitplan

2.2 Auswahl geeigneter Varianten

2.3 Instruction Sets

Die meisten Prozessoren besitzen Befehle aus den folgenden fünf Gruppen:

- Daten Transfer Gruppe
- Arithmetik Gruppe
- Logischen Gruppe
- Zweig Gruppe
- Stapel, Ein- Ausgänge und Maschinen Kontroll- Gruppe

Die Befehle der Daten Transfer Gruppe bewegen Daten zwischen Registern und/oder Speicher wie zum Beispiel mit den MOVE Befehlen. In der Arithmetik Gruppe werden, wie der Name schon sagt, Befehle mit arithmetische Operationen wie Addition verwendet. In der Logischen Gruppen werden Operation wie das logische Oder verwendet. In der Zweig Gruppe gibt es die Befehle, welche den Standardmäßigen Programmfluss ändern und das Programm nicht zwangsläufig in der nächsten Zeile fortgesetzt wird wie bei bedingten Sprüngen. In der letzten Gruppe liegen die Befehle, die Eingänge und Ausgänge beachten oder den Stack bearbeiten. [Intel 8080 S.46 (4-1)]

2.4 Intel 8080

Der Intel 8080 ist in der Lage Befehle, die aus einem, zwei oder drei Bytes bestehen, auszuführen. Dabei gibt das erste Byte immer den Opcode oder Operation Code an. In Byte zwei und drei werden nur Daten oder Adressen gespeichert. Dabei werden die zwei Byte großen Adressen so gespeichert, dass das niederwertige Byte vor dem höherwertigem gespeichert wird. Die Adressen können dabei über vier verschiedene Modi verwendet werden.

- Direct
- Register

- Register Indirect
- Immediate

Bei "Direkt“ wird der Wert in dem Speicher mit der angegebenen Adresse verwendet. Hier werden das Low-Byte im zweiten und das High-Byte im dritten Byte gespeichert. Bei "Register“ wird auf ein oder zwei Register verwiesen und verhält sich wie bei Direkt. Bei "Register Indirect“ wird der Wert aus der Adresse aus dem zweiten und dritten Byte des Befehls gelesen. Dieser Wert wird als Adresse verarbeitet und erst der Wert aus dieser Adresse ist der zu verwendete Wert. Bei Ïmmediate“ steht im zweiten und/oder dritten Byte ein Wert mit dem gearbeitet wird (Lowbyte im zweiten Byte). [Intel 8080 S.47 (4-2)]

Bei Interrupts und Branch Befehlen gibt es nur den "Direct“ und "Register indirect“ Modus [Intel 8080 S.47 (4-2)].

Der Prozessor besitzt fünf Condition Flags. Das Zero flag, das angibt ob das Ergebnis eines Befehls den Wert 0 hatte. Das Sign flag, welches angibt ob Bit 8, das Most Signifikant-Bit, des letzten Ergebnisses den Wert 1 hat. Das Paritäts flag, welches gesetzt ist, wenn das letzte Ergebnis einen Modulo 2 Wert von 0 hat, also der Wert gerade ist. Das Carry flag, das einen Übertrag bei einer Addition oder einen Abzug bei einer Subtraktion oder Vergleich anzeigt und noch das Auxiliary Carry flag, welches ebenfalls einen Übertrag oder Abzug anzeigt aber zwischen dem vierten (Bit 3) und fünften Bit (Bit 4). [Intel 8080 S.47f (4-2)]

2.4.1 Data Transfer Group

Für den Prozessor sind 13 Befehle aus dieser Gruppe bekannt. Bei keinem dieser Befehle werden die Condition Flags gesetzt oder zurückgesetzt.

- Move Register
- Move from Memory
- Move to Memory
- Move immediate
- Move to Memory Immediate
- Load register pair immediate
- Load Accumulator direct
- Store Accumulator direct
- Load H and L direct
- Store H and L direct
- Load Acumulator indirect
- Store Accumulator indirect
- Exchange H and L with D and E

2.4.2 Arithmetic Group

20 Befehle Unless indicated otherwise, all instructions in this group affect the Zero, Sign, Parity, Carry, and Auxiliary Carry flags accord ing to the standard rules. All subtraction operations are performed via two's complement arithmetic and set the carry flag to one to indicate a borrow and clear it to indicate no borrow

- Add Register
- Add Memory
- Add immediate
- Add Register with carry
- Add Memory with carry
- Add immediate with carry
- Sustract Register
- Subtract Memory
- Subtract immediate
- Subtract Register with borrow
- Subtract Memory with borrow
- Subtract immediate with borrow
- Increment Register
- Increment Memory
- Decrement Register
- Decrement Memory
- Increment register pair
- Decrement register pair
- Add register pair to H and L
- Decimal Adjust Accumulator

2.4.3 Logical Group

19 Befehle Unless indicated otherwise, all instructions in this group affect the Zero, Sign, Parity, Auxiliary Carry, and Carry flags according to the standard rules

- AND Register
- AND Memory
- AND immediate
- Exclusive OR Register
- Exclusive OR Memory
- Exclusive OR immediate
- OR Register
- OR Memory
- OR immediate
- Compare Register
- Compare Memory
- Compare immediate
- Rotate left
- Rotate right
- Rotate left through Carry
- Rotate right through Carry
- Complement Accumulator
- Complement Carry

- Set Carry

2.4.4 Branch Group

8 Befehle Flags are not affected

- Jump
- Conditional Jump
- Call
- Conditional Call
- Return
- Conditional Return
- Restart
- Jump H and L indirect - move H and L to PC

2.4.5 Stack, I/O and Machine Control Group

12 Befehle Unless otherwise specified, condition flags are not affected by any instructions in this group

- Push
- Push Processor status word
- Pop
- Pop processor status word
- Exchange stack top with H and L
- Move HL to SP
- Input
- Output
- Enable Interrupts
- Disable Interrupts
- Halt
- No op

2.4.6 Maschinenzyklen und Typen

Jeder Befehl oder Befehlszyklus benötigt mindestens einen und maximal fünf Maschinenzyklen zum kompletten Ausführen des Befehls. Dabei kann jeder Befehl aus mehreren Typen bestehen, wobei der erste immer ein Fetch-Befehlstyp ist. Die existierenden Befehlstypen sind:

- Fetch

- Memory Read
- Memory Write
- Stack Read
- Stack Write
- Input
- Output
- Interrupt
- Halt
- Halt Interrupt

Jeder Maschinenzyklus besteht nochmal aus drei bis fünf Zuständen. So kann ein Befehl insgesamt zwischen vier und achtzehn Zuständen andauern. In dem ersten Zustand jedes Maschinenzykluses wird immer der Befehlstyp während des SYNC-Taktes kodiert. Die folgende Abbildung zeigt, wie die Befehlstypen über die **Data Lines** kodiert werden. [Intel 8080 S17ff. 2-3]

Abbildung 1: Kodierung der Befehlstypen beim SYNC Takt

STATUS WORD CHART

TYPE OF MACHINE CYCLE

DATA BUS BIT	STATUS INFORMATION	① INSTRUCTION FETCH	② MEMORY READ	③ MEMORY WRITE	④ STACK READ	⑤ STACK WRITE	⑥ INPUT READ	⑦ OUTPUT WRITE	⑧ INTERRUPT ACKNOWLEDGE	⑨ HALT ACKNOWLEDGE	⑩ INTERRUPT ACKNOWLEDGE WHILE HALT
											Ⓝ STATUS WORD
D_0	INTA	0	0	0	0	0	0	0	1	0	1
D_1	$\overline{WO}$	1	1	0	1	0	1	0	1	1	1
D_2	STACK	0	0	0	1	1	0	0	0	0	0
D_3	HLTA	0	0	0	0	0	0	0	0	1	1
D_4	OUT	0	0	0	0	0	0	1	0	0	0
D_5	M_1	1	0	0	0	0	0	0	1	0	1
D_6	INP	0	0	0	0	0	1	0	0	0	0
D_7	MEMR	1	1	0	1	0	0	0	0	1	0

2.5 Beispielprozessor 2

2.6 Beispielprozessor 3 ...

3 Umsetzung

Platzhalter

3.1 Abstraktion der Architektur

3.1.1 ALU

3.1.2 Akkumulator

3.1.3 Instruction Register

3.1.4 DataBus

3.1.5 etc.

3.2 Ablauf der Simulation

3.3 Tutorials

4 Fazit und Ausblick

Platzhalter

Literatur

[1] Google: https://www.google.com

[2] Grundlagen der Informatik: Herold, Lurz, Wohlrab und Hopf; 3. aktualisierte Auflage (2017), Pearson

[3] Instruction formats: https://www.geeksforgeeks.org/computer-organization-instruction-formats-zero-one-two-three-address-instruction/, zuletzt abgerufen: 25.12.2021