

6502 Emulations & Debuggingtool

Eingebettete Systeme

Präsentiert von Younes Oubkis

Agenda

- Motivation
- Projektziel
- Grundlagen
- Durchführung
- Ergebnis
- Ausblick
- Fragen

Motivation

Oberziel:

- Entwicklung einer 6502 „Konsole“
- Anbindung an Hardware für Grafik & Audio sowie Input

Lernressource: 6502 Prozessor

- Preisgünstig
- Umfassend dokumentiert
- Seit über 30 Jahren in Produktion
- Weniger komplex, dennoch mächtig

Entwicklung in Assembler:

- Intransparent
- Abstrakt
- Klobig
- Fehleranfällig

Projektziel

Entwicklung eines Emulationstools, welcher Debuggingmöglichkeiten wie Breakpoints und Memorysnapshots ermöglicht

- *Entwicklung eines Emulators*
- *Schrittweises Emulieren und Zustandsausgabe*

Grundlagen

MOS 6502

- Mikroprozessorfamilie seit 1970er
- 16 Bit Adressierung
- 1-4 MHz Taktrate



6502 Assembler

- Primitivste CPU-Operationen
- Kurze, direkte Ansteuerung Logischen Einheiten

Register und Speicher

- 2 Indexregister, 1 Akkumulator
- Statusregister - Befehlsergebnisse
- Stack mit Pointeradresse

6502 Befehle und Adressierungen

- Alle Befehle benötigen
- 1 Register und/oder
- 1 Adress-/Parameter

CLC

LDA \$BF

ADC #\$0A

Beispielanweisung:
Addition von 0x0A zum Wert
an Adresse 0xBF

Adressierungsmethoden

- Implizit
- Akkumulator
- Absolut
- Relativ
- Zero Page
- ...

HI	LO-NIBBLE															
	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	BRK impl	ORA X,ind				ORA zpg	ASL zpg		PHP impl	ORA #	ASL A			ORA abs	ASL abs	
1-	BPL rel	ORA ind,Y				ORA zpg,X	ASL zpg,X		CLC impl	ORA abs,Y				ORA abs,X	ASL abs,X	
2-	JSR abs	AND X,ind			BIT zpg	AND zpg	ROL zpg		PLP impl	AND #	ROL A		BIT abs	AND abs	ROL abs	
3-	BMI rel	AND ind,Y				AND zpg,X	ROL zpg,X		SEC impl	AND abs,Y				AND abs,X	ROL abs,X	
4-	RTI impl	EOR X,ind				EOR zpg	LSR zpg		PHA impl	EOR #	LSR A		JMP abs	EOR abs	LSR abs	
5-	BVC rel	EOR ind,Y				EOR zpg,X	LSR zpg,X		CLI impl	EOR abs,Y				EOR abs,X	LSR abs,X	
6-	RTS impl	ADC X,ind				ADC zpg	ROR zpg		PLA impl	ADC #	ROR A		JMP ind	ADC abs	ROR abs	
7-	BVS rel	ADC ind,Y				ADC zpg,X	ROR zpg,X		SEI impl	ADC abs,Y				ADC abs,X	ROR abs,X	
8-		STA X,ind			STY zpg	STA zpg	STX zpg		DEY impl		TXA impl		STY abs	STA abs	STX abs	
9-	BCC rel	STA ind,Y			STY zpg,X	STA zpg,X	STX zpg,Y		TYA impl	STA abs,Y	TXS impl			STA abs,X		
A-	LDY #	LDA X,ind	LDX #		LDY zpg	LDA zpg	LDX zpg		TAY impl	LDA #	TAX impl		LDY abs	LDA abs	LDX abs	
B-	BCS rel	LDA ind,Y			LDY zpg,X	LDA zpg,X	LDX zpg,Y		CLV impl	LDA abs,Y	TSX impl		LDY abs,X	LDA abs,X	LDX abs,Y	
C-	CPY #	CMP X,ind			CPY zpg	CMP zpg	DEC zpg		INY impl	CMP #	DEX impl		CPY abs	CMP abs	DEC abs	
D-	BNE rel	CMP ind,Y				CMP zpg,X	DEC zpg,X		CLD impl	CMP abs,Y				CMP abs,X	DEC abs,X	
E-	CPX #	SBC X,ind			CPX zpg	SBC zpg	INC zpg		INX impl	SBC #	NOP impl		CPX abs	SBC abs	INC abs	
F-	BEQ rel	SBC ind,Y				SBC zpg,X	INC zpg,X		SED impl	SBC abs,Y				SBC abs,X	INC abs,X	

Anweisungsmatrix des „klassischen“ 6502

Durchführung

.NET 8.0 Desktop Anwendung

- C# Programmiersprache
- Commandline Tool
- Eingabe von Anweisungen oder Debugbefehlen
- Ausgabe von Prozessor & Speicherstatus

Prozessorstatus

- Register – Bytes
- Stack – Byte Array
- „Externer“ Speicher – Byte Array

```
public ushort PC;  
public byte A, X, Y;  
public byte Flags;  
public byte SP = 255;  
public byte _sp = 0;  
public byte[] Stack;
```

```
private readonly byte[] _data;
```

Abbildung des Prozessorstatus in C#

Problem – Jede Adressierung ist „real“ ein eigener Befehl

- Aus 56 Befehlen werden ca. 160
- Unterschiede in Taktzyklen
- Jeden Befehl X mal Implementieren?
- Switch für jeden Befehl?

HI	LO-NIBBLE														
	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E
0-	BRK impl	ORA X,ind				ORA spg	ASL spg		FRP impl	ORA #	ASL A			ORA abs	ASL abs
1-	BPL rel	ORA ind,Y				ORA spg,X	ASL spg,X		CLC impl	ORA abs,Y				ORA abs,X	ASL abs,X
2-	JSR abs	AND X,ind			BIT spg	AND spg	ROL spg		PLP impl	AND #	ROL A		BIT abs	AND abs	ROL abs
3-	BMI rel	AND ind,Y				AND spg,X	ROL spg,X		SEC impl	AND abs,Y				AND abs,X	ROL abs,X
4-	RTI impl	EOR X,ind				EOR spg	LSR spg		PHA impl	EOR #	LSR A		JMF abs	EOR abs	LSR abs
5-	BVC rel	EOR ind,Y				EOR spg,X	LSR spg,X		CLI impl	EOR abs,Y				EOR abs,X	LSR abs,X
6-	RTS impl	ADC X,ind				ADC spg	ROR spg		PLA impl	ADC #	ROR A		JMP ind	ADC abs	ROR abs
7-	BVS rel	ADC ind,Y				ADC spg,X	ROR spg,X		IMI impl	ADC abs,Y				ADC abs,X	ROR abs,X
8-		STA X,ind			STY spg	STA spg	STX spg		DEY impl		TXA impl		STY abs	STA abs	STX abs
9-	BCC rel	STA ind,Y			STY spg,X	STA spg,X	STX spg,Y		TYA impl	STA abs,Y	TXS impl			STA abs,X	
A-	LDY #	LDA X,ind	LDX #		LDY spg	LDA spg	LDX spg		TXA impl	LDA #	TAX impl		LDY abs	LDA abs	LDX abs
B-	SCS rel	LDA ind,Y			LDY spg,X	LDA spg,X	LDX spg,Y		CLV impl	LDA abs,Y	TSX impl		LDY abs,X	LDA abs,X	LDX abs,Y
C-	CPY #	CMP X,ind			CPY spg	CMP spg	DEC spg		INY impl	CMP #	DEX impl		CPY abs	CMP abs	DEC abs
D-	BNE rel	CMP ind,Y				CMP spg,X	DEC spg,X		CLE impl	CMP abs,Y				CMP abs,X	DEC abs,X
E-	CPX #	SBC X,ind			CPX spg	SBC spg	INC spg		INX impl	SBC #	NOP impl		CPX abs	SBC abs	INC abs
F-	SEQ rel	SBC ind,Y				SBC spg,X	INC spg,X		SED impl	SBC abs,Y				SBC abs,X	INC abs,X

Lösung: Adressierung von eigentlicher Logik ausklinken

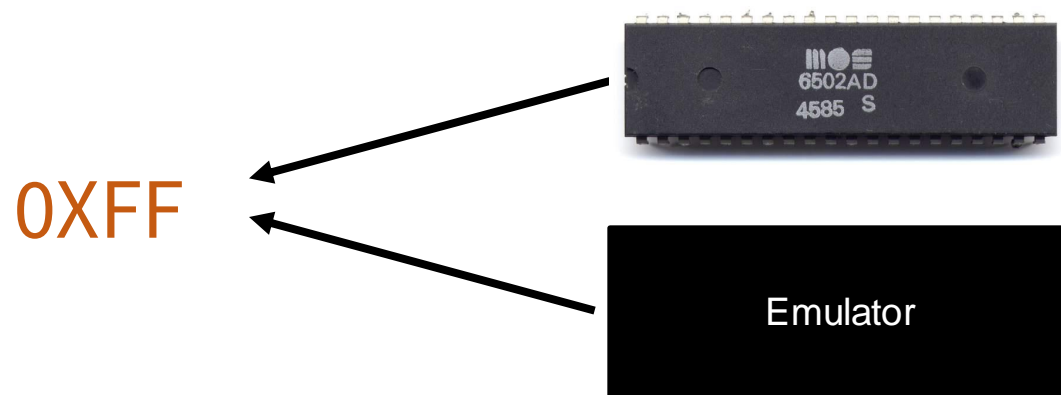
- Injektion einer Adressierungslogik
- Befehlslogik vereinfacht
- Erweiterte Loggingmöglichkeiten

```
public delegate ushort AddressParser(ushort value, ref ProcessorState state, ref Memory mem);
```

Delegatendefinition: Adressierungsauflösung

Emulation \neq Simulation

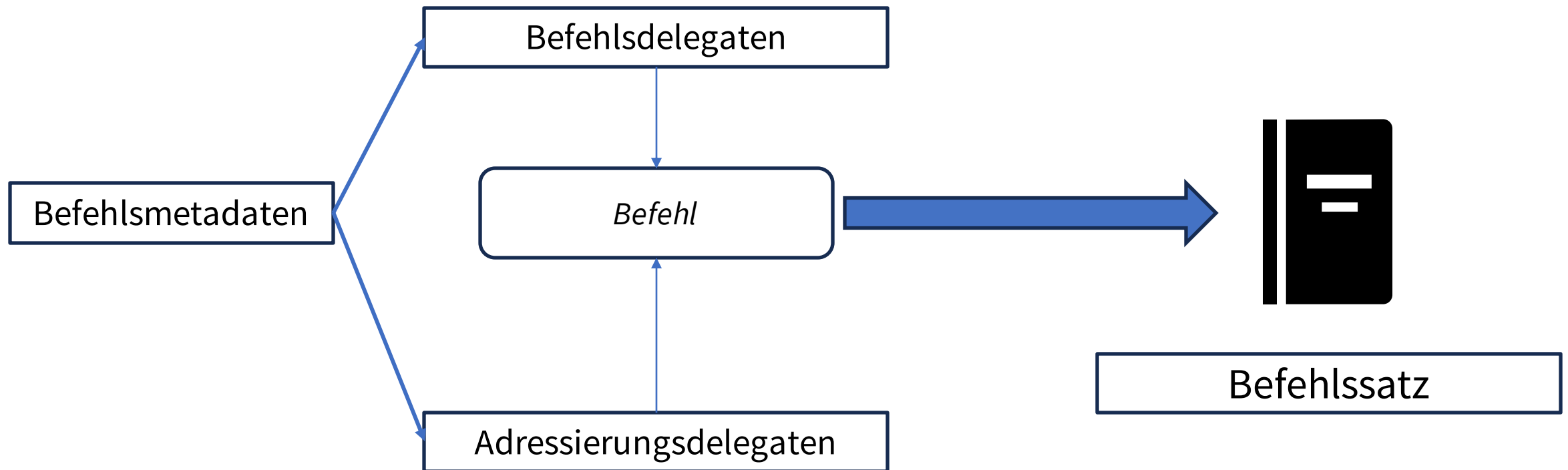
- Optimierungen für die Zielplattform
- Verbesserte Fehleranalyse: Aufteilung der Logik



„Just in Time“ Kompilierung

- Reguläre Ausdrücke zum parsen von OpCodes
- Injektion der Adressierungslogik
- Direktausgabe von status

Ablauf: dynamischen Befehlssatz bilden



Parsen mit regulären Ausdrücken

LDA_#\$FF_

$^(\backslash w\{3\})\backslash s(.*)\backslash n\$$

#\$FF

$\#([\$ \%]\{0,1\})([A-Za-z0-9]*)$

Ablauf: Parsen des Assembler Codes

LDA #\$FF

Regex

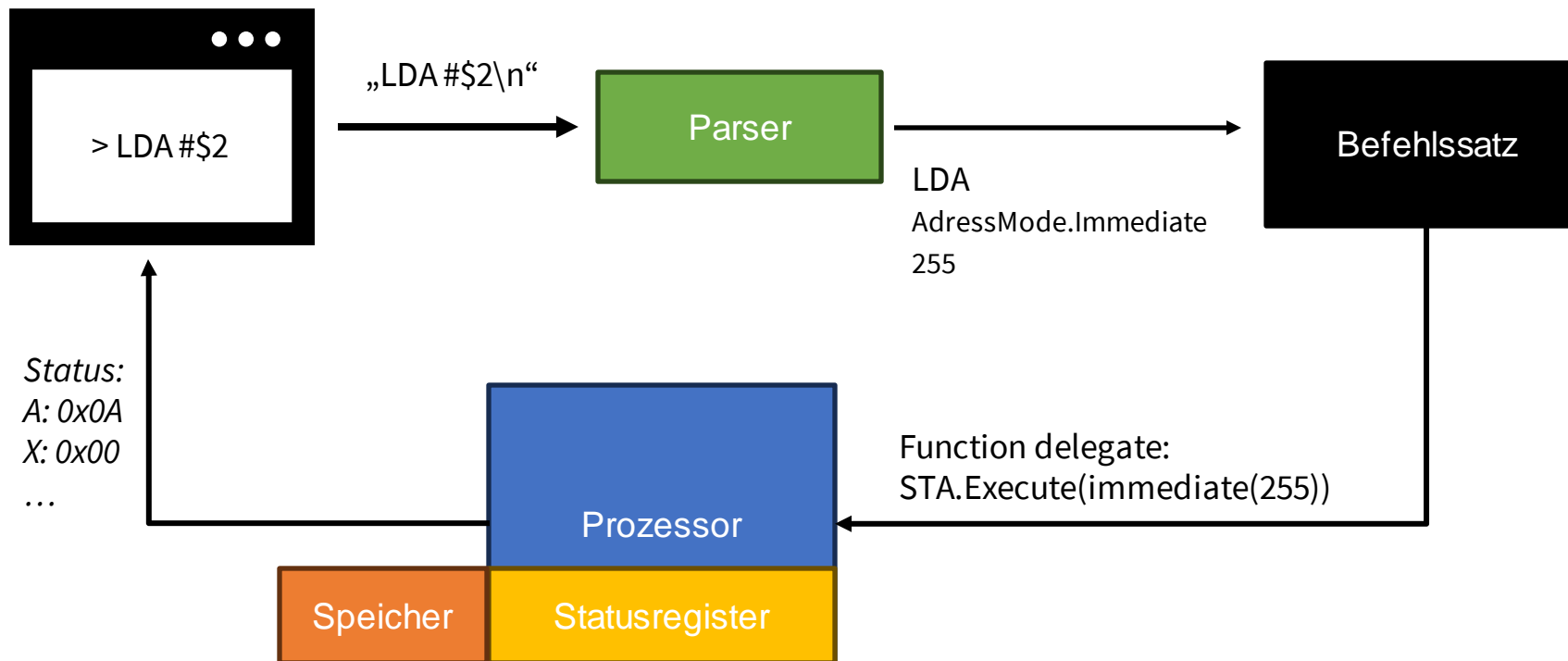
LDA

(Immediate)

\$ (Hexadezimal)

0xFF (255)

Prozessoremulation (Just In Time)



Ergebnis

Ergebnis

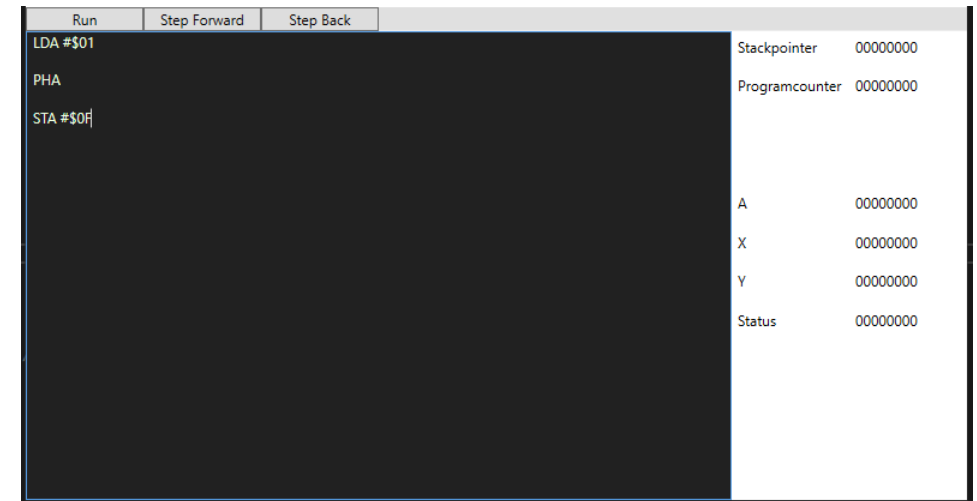
- Lauffähiger Prototyp
- 20 Befehle abgebildet
- Funktioninsfähiger Stack und Speicher

```
InstructionSet of 6502 loaded with 18 OpCodes.  
Please Enter ASM6502 Line or "close" to exit Application  
LDA #$F0  
Instruction with Byte LDA  
and Mode IMMEDIATE  
with Argument 0xF0  
  
---  
Load Accumulator.  
Valid Codes:8  
---  
  
LDA #$FF  
Instruction with Byte LDA  
and Mode IMMEDIATE  
with Argument 0xFF  
  
---  
Load Accumulator.  
Valid Codes:8  
---  
  
status  
Status Report  
A:0xFF X:0x00 Y:0x00  
Flags:00000000  
      NV BDIZC  
StackPointer:0x00  
Program Counter:0x0000  
---  
  
reset  
Processor Reset
```

Ausblick

Ausblick

- Restliche 25 Befehle implementieren
- UI „IDE“
- Step Funktion



Vielen Dank für die Aufmerksamkeit

Noch Fragen?

6502 Emulations & Debuggingtool

Eingebettete Systeme

Präsentiert von Younes Oubkis