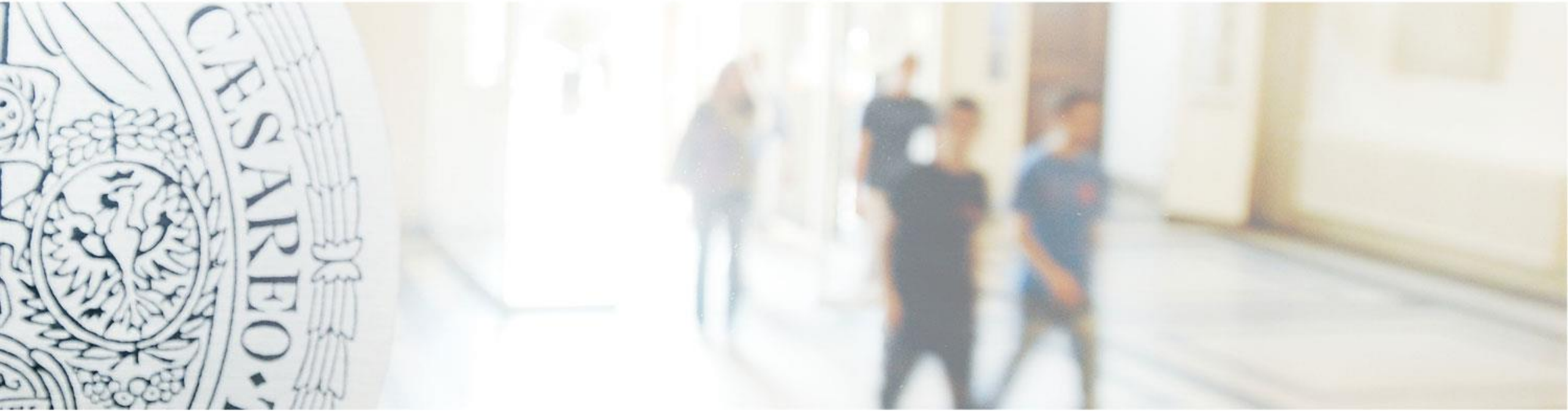
universität innsbruck



ARM Simulator, Interpreter und Debugger als Webanwendung

Finalpräsentation

Zangerl Dominik
Betreuer: Alexander Schlögl

Gliederung

- Einleitung und Motivation
- Theorie
  - ARMv5
  - Parser
- Implementation
- Evaluation und Zusammenfassung
- Referenzen

ARMv5 im ersten Semester

- ARMv5 [2] als Beispiel einer Befehlssatzarchitektur

ARMv5 im ersten Semester

- ARMv5 [2] als Beispiel einer Befehlssatzarchitektur
- Schreiben von Assembler-Programme und Ausführung auf einer ARMv5 Architektur

ARMv5 im ersten Semester

- ARMv5 [2] als Beispiel einer Befehlssatzarchitektur
- Schreiben von Assembler-Programme und Ausführung auf einer ARMv5 Architektur
- Simulation mit GNU Toolchain [1]:

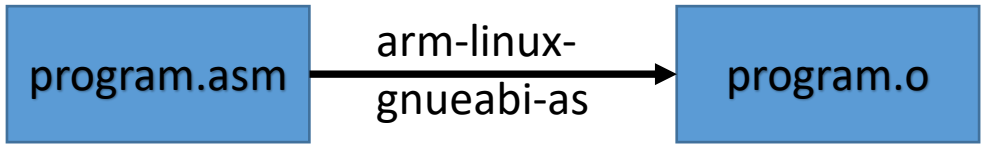
ARMv5 im ersten Semester

- ARMv5 [2] als Beispiel einer Befehlssatzarchitektur
- Schreiben von Assembler-Programme und Ausführung auf einer ARMv5 Architektur
- Simulation mit GNU Toolchain [1]:

program.asm

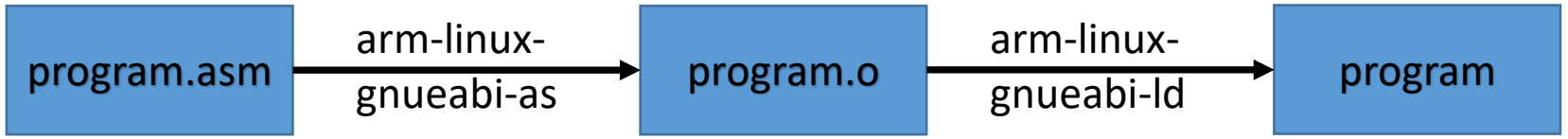
ARMv5 im ersten Semester

- ARMv5 [2] als Beispiel einer Befehlssatzarchitektur
- Schreiben von Assembler-Programme und Ausführung auf einer ARMv5 Architektur
- Simulation mit GNU Toolchain [1]:



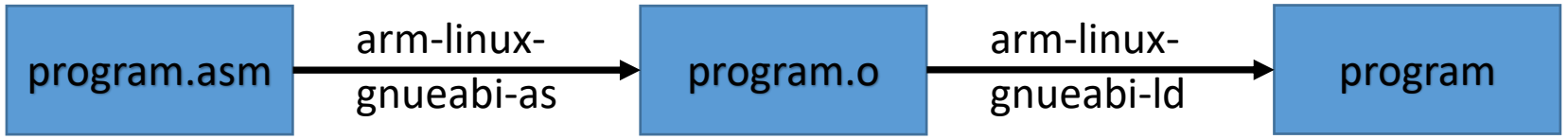
ARMv5 im ersten Semester

- ARMv5 [2] als Beispiel einer Befehlssatzarchitektur
- Schreiben von Assembler-Programme und Ausführung auf einer ARMv5 Architektur
- Simulation mit GNU Toolchain [1]:



ARMv5 im ersten Semester

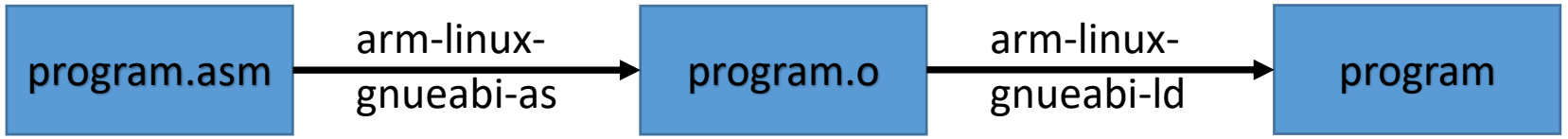
- ARMv5 [2] als Beispiel einer Befehlssatzarchitektur
- Schreiben von Assembler-Programme und Ausführung auf einer ARMv5 Architektur
- Simulation mit GNU Toolchain [1]:



- Ausführen mit QEMU User-Space-Emulator [13]

ARMv5 im ersten Semester

- ARMv5 [2] als Beispiel einer Befehlssatzarchitektur
- Schreiben von Assembler-Programme und Ausführung auf einer ARMv5 Architektur
- Simulation mit GNU Toolchain [1]:



- Ausführen mit QEMU User-Space-Emulator [13]
- Vereinfachung mit Skript und Ausführung über virtuelle Maschine oder WSL [11]

Debugging

- Größter Zeitaufwand bei Fehlersuche im Programm

Debugging

- Größter Zeitaufwand bei Fehlersuche im Programm
- Kann zusammen mit dem Gnu Debugger [12] verwendet werden:

Debugging

- Größter Zeitaufwand bei Fehlersuche im Programm
- Kann zusammen mit dem Gnu Debugger [12] verwendet werden:

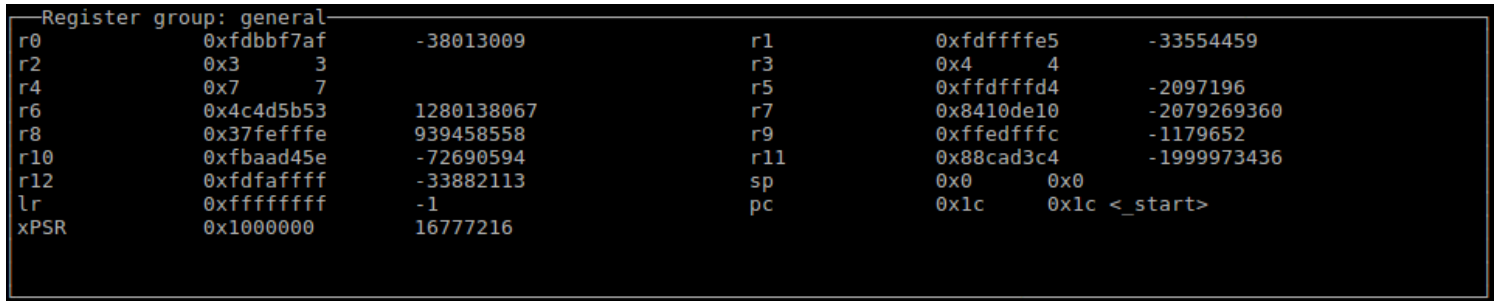


Abbildung: Use GDB on an ARM assembly program [15]

Debugging

- Größter Zeitaufwand bei Fehlersuche im Programm
- Kann zusammen mit dem Gnu Debugger [12] verwendet werden:

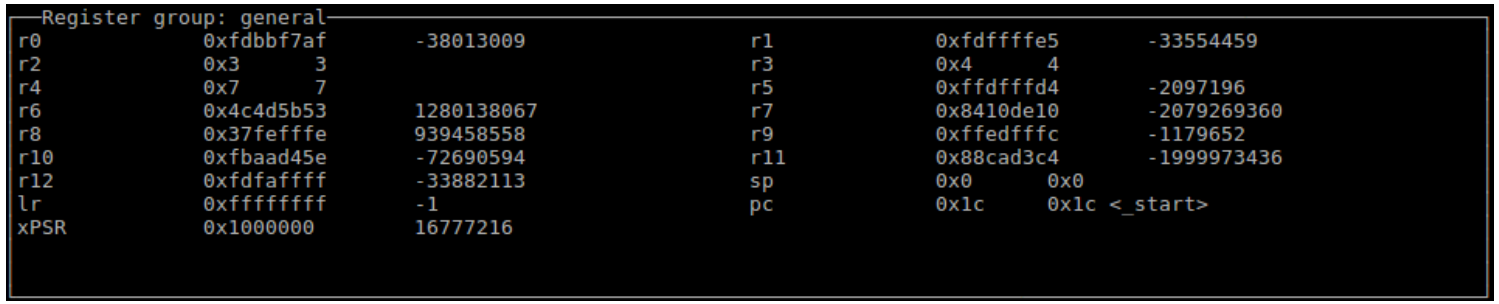


Abbildung: Use GDB on an ARM assembly program [15]

- Arbeiten mit Debuggern im ersten Semester oft schwierig

Debugging

- Größter Zeitaufwand bei Fehlersuche im Programm
- Kann zusammen mit dem Gnu Debugger [12] verwendet werden:

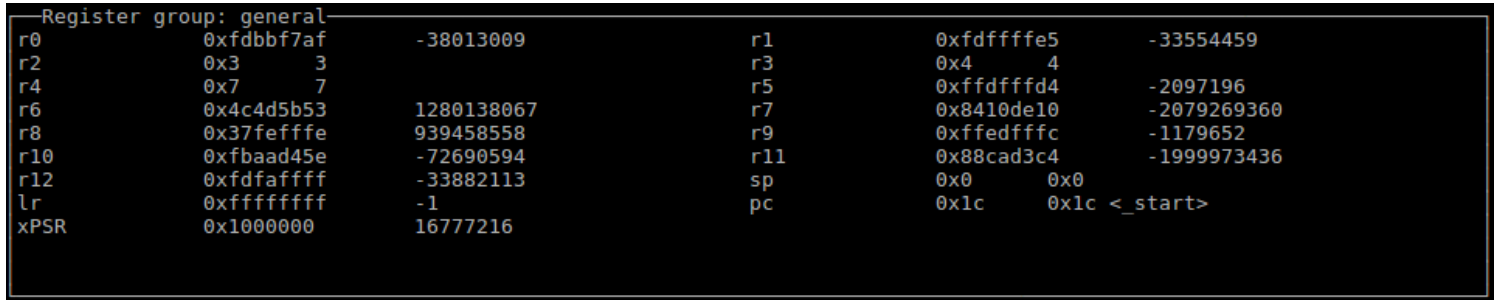


Abbildung: Use GDB on an ARM assembly program [15]

- Arbeiten mit Debuggern im ersten Semester oft schwierig
- Großer Zeitaufwand zusammen mit Aufsetzen der Toolchain

ARMv5 Umgebung und Debugging als Webanwendung

- Bachelorprojekt: Simuliere ARMv5 Entwicklungsumgebung und Debugger als Webanwendung

ARMv5 Umgebung und Debugging als Webanwendung

- Bachelorprojekt: Simuliere ARMv5 Entwicklungsumgebung und Debugger als Webanwendung
- ARMv5 Entwicklungsumgebung

ARMv5 Umgebung und Debugging als Webanwendung

- Bachelorprojekt: Simuliere ARMv5 Entwicklungsumgebung und Debugger als Webanwendung
- ARMv5 Entwicklungsumgebung
  - Simulierte CPU und Hauptspeicher

ARMv5 Umgebung und Debugging als Webanwendung

- Bachelorprojekt: Simuliere ARMv5 Entwicklungsumgebung und Debugger als Webanwendung
- ARMv5 Entwicklungsumgebung
  - Simulierte CPU und Hauptspeicher
  - Assembler-Code in Webanwendung schreiben und direkt im Browser ausführen

ARMv5 Umgebung und Debugging als Webanwendung

- Bachelorprojekt: Simuliere ARMv5 Entwicklungsumgebung und Debugger als Webanwendung
- ARMv5 Entwicklungsumgebung
  - Simulierte CPU und Hauptspeicher
  - Assembler-Code in Webanwendung schreiben und direkt im Browser ausführen
  - Dauerhafte Anzeige von Registern und Stacks

ARMv5 Umgebung und Debugging als Webanwendung

- Bachelorprojekt: Simuliere ARMv5 Entwicklungsumgebung und Debugger als Webanwendung
- ARMv5 Entwicklungsumgebung
  - Simulierte CPU und Hauptspeicher
  - Assembler-Code in Webanwendung schreiben und direkt im Browser ausführen
  - Dauerhafte Anzeige von Registern und Stacks
- Debugger

ARMv5 Umgebung und Debugging als Webanwendung

- Bachelorprojekt: Simuliere ARMv5 Entwicklungsumgebung und Debugger als Webanwendung
- ARMv5 Entwicklungsumgebung
  - Simulierte CPU und Hauptspeicher
  - Assembler-Code in Webanwendung schreiben und direkt im Browser ausführen
  - Dauerhafte Anzeige von Registern und Stacks
- Debugger
  - Breakpoints

ARMv5 Umgebung und Debugging als Webanwendung

- Bachelorprojekt: Simuliere ARMv5 Entwicklungsumgebung und Debugger als Webanwendung
- ARMv5 Entwicklungsumgebung
  - Simulierte CPU und Hauptspeicher
  - Assembler-Code in Webanwendung schreiben und direkt im Browser ausführen
  - Dauerhafte Anzeige von Registern und Stacks
- Debugger
  - Breakpoints
  - Zeilenweise Abarbeitung

ARMv5 Architektur

- Rechner mit reduziertem Befehlssatz (RISC [4]):

ARMv5 Architektur

- Rechner mit reduziertem Befehlssatz (RISC [4]):
  - Load/Store-Architektur

ARMv5 Architektur

- Rechner mit reduziertem Befehlssatz (RISC [4]):
  - Load/Store-Architektur
  - Datenverarbeitende Instruktionen arbeiten nur mit Inhalten der Register

ARMv5 Architektur

- Rechner mit reduziertem Befehlssatz (RISC [4]):
  - Load/Store-Architektur
  - Datenverarbeitende Instruktionen arbeiten nur mit Inhalten der Register
  - Einheitliche Form und Länge der Kodierung

ARMv5 Architektur

- Rechner mit reduziertem Befehlssatz (RISC [4]):
  - Load/Store-Architektur
  - Datenverarbeitende Instruktionen arbeiten nur mit Inhalten der Register
  - Einheitliche Form und Länge der Kodierung

- Zusätzlich bei ARM:

ARMv5 Architektur

- Rechner mit reduziertem Befehlssatz (RISC [4]):
  - Load/Store-Architektur
  - Datenverarbeitende Instruktionen arbeiten nur mit Inhalten der Register
  - Einheitliche Form und Länge der Kodierung

- Zusätzlich bei ARM:
  - Meiste Instruktionen haben Zugriff ALU und Barrel-Shifter

ARMv5 Architektur

- Rechner mit reduziertem Befehlssatz (RISC [4]):
  - Load/Store-Architektur
  - Datenverarbeitende Instruktionen arbeiten nur mit Inhalten der Register
  - Einheitliche Form und Länge der Kodierung

- Zusätzlich bei ARM:
  - Meiste Instruktionen haben Zugriff ALU und Barrel-Shifter
  - Adressierungsarten, die Adresse automatisch inkrementieren/dekrementieren

ARMv5 Architektur

- Rechner mit reduziertem Befehlssatz (RISC [4]):
  - Load/Store-Architektur
  - Datenverarbeitende Instruktionen arbeiten nur mit Inhalten der Register
  - Einheitliche Form und Länge der Kodierung

- Zusätzlich bei ARM:
  - Meiste Instruktionen haben Zugriff ALU und Barrel-Shifter
  - Adressierungsarten, die Adresse automatisch inkrementieren/dekrementieren
  - Instruktionen zum Laden/Speichern von mehreren Registern

ARMv5 Register

- 31 Universalregister mit einer Breite von 32 Bit

ARMv5 Register

- 31 Universalregister mit einer Breite von 32 Bit
  - 16 sichtbar, je nach Ausführungsmodus

Ausführungsmodi

Privilegierte Ausführungsmodi

Ausnahmenmodi

User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

zeigt an, dass normale Register aus User- und Systemmodus durch alternative Register spezifisch für den jeweiligen Ausnahmemodus ersetzt wurden

Abbildung: Register je nach Ausführungsmodi [2]

ARMv5 Register

- 31 Universalregister mit einer Breite von 32 Bit
  - 16 sichtbar, je nach Ausführungsmodus
  - R13 – Stapelzeiger

Ausführungsmodi

Privilegierte Ausführungsmodi

Ausnahmenmodi

User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

zeigt an, dass normale Register aus User- und Systemmodus durch alternative Register spezifisch für den jeweiligen Ausnahmemodus ersetzt wurden

Abbildung: Register je nach Ausführungsmodi [2]

ARMv5 Register

- 31 Universalregister mit einer Breite von 32 Bit
  - 16 sichtbar, je nach Ausführungsmodus
  - R13 – Stapelzeiger
  - R14 – Link-Register

Ausführungsmodi

Privilegierte Ausführungsmodi

Ausnahmenmodi

User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

zeigt an, dass normale Register aus User- und Systemmodus durch alternative Register spezifisch für den jeweiligen Ausnahmemodus ersetzt wurden

Abbildung: Register je nach Ausführungsmodi [2]

ARMv5 Register

- 31 Universalregister mit einer Breite von 32 Bit
  - 16 sichtbar, je nach Ausführungsmodus
  - R13 – Stapelzeiger
  - R14 – Link-Register
  - R15 – Befehlszähler

Ausführungsmodi

Privilegierte Ausführungsmodi

Ausnahmenmodi

User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

zeigt an, dass normale Register aus User- und Systemmodus durch alternative Register spezifisch für den jeweiligen Ausnahmemodus ersetzt wurden

Abbildung: Register je nach Ausführungsmodi [2]

ARMv5 Register

- 31 Universalregister mit einer Breite von 32 Bit
  - 16 sichtbar, je nach Ausführungsmodus
  - R13 – Stapelzeiger
  - R14 – Link-Register
  - R15 – Befehlszähler
- Status-Register (CPSR) mit Status-Flags NZCV:

Ausführungsmodi

Privilegierte Ausführungsmodi

Ausnahmenmodi

User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

zeigt an, dass normale Register aus User- und Systemmodus durch alternative Register spezifisch für den jeweiligen Ausnahmemodus ersetzt wurden

Abbildung: Register je nach Ausführungsmodi [2]

ARMv5 Register

- 31 Universalregister mit einer Breite von 32 Bit
  - 16 sichtbar, je nach Ausführungsmodus
  - R13 – Stapelzeiger
  - R14 – Link-Register
  - R15 – Befehlszähler
- Status-Register (CPSR) mit Status-Flags NZCV:
  - N – Negativ

Ausführungsmodi

Privilegierte Ausführungsmodi

Ausnahmenmodi

User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

zeigt an, dass normale Register aus User- und Systemmodus durch alternative Register spezifisch für den jeweiligen Ausnahmemodus ersetzt wurden

Abbildung: Register je nach Ausführungsmodi [2]

ARMv5 Register

- 31 Universalregister mit einer Breite von 32 Bit
  - 16 sichtbar, je nach Ausführungsmodus
  - R13 – Stapelzeiger
  - R14 – Link-Register
  - R15 – Befehlszähler
- Status-Register (CPSR) mit Status-Flags NZCV:
  - N – Negativ
  - Z – Null (Zero)

Ausführungsmodi

Privilegierte Ausführungsmodi

Ausnahmenmodi

User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

zeigt an, dass normale Register aus User- und Systemmodus durch alternative Register spezifisch für den jeweiligen Ausnahmemodus ersetzt wurden

Abbildung: Register je nach Ausführungsmodi [2]

ARMv5 Register

- 31 Universalregister mit einer Breite von 32 Bit
  - 16 sichtbar, je nach Ausführungsmodus
  - R13 – Stapelzeiger
  - R14 – Link-Register
  - R15 – Befehlszähler
- Status-Register (CPSR) mit Status-Flags NZCV:
  - N – Negativ
  - Z – Null (Zero)
  - C – Übertrag (Carry)

Ausführungsmodi

Privilegierte Ausführungsmodi

Ausnahmenmodi

User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

zeigt an, dass normale Register aus User- und Systemmodus durch alternative Register spezifisch für den jeweiligen Ausnahmemodus ersetzt wurden

Abbildung: Register je nach Ausführungsmodi [2]

ARMv5 Register

- 31 Universalregister mit einer Breite von 32 Bit
  - 16 sichtbar, je nach Ausführungsmodus
  - R13 – Stapelzeiger
  - R14 – Link-Register
  - R15 – Befehlszähler
- Status-Register (CPSR) mit Status-Flags NZCV:
  - N – Negativ
  - Z – Null (Zero)
  - C – Übertrag (Carry)
  - V – Überlauf (Overflow)

Ausführungsmodi

Privilegierte Ausführungsmodi

Ausnahmenmodi

User	System	Supervisor	Abort	Undefined	Interrupt	Fast interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

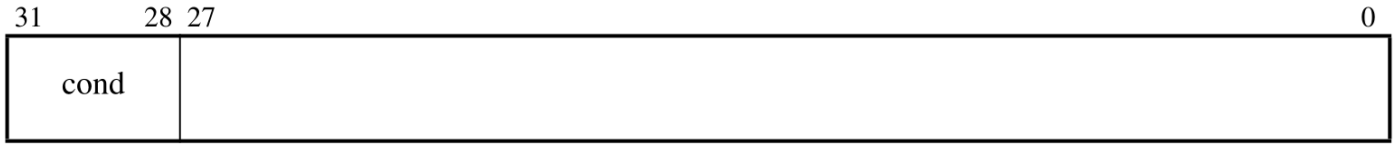
zeigt an, dass normale Register aus User- und Systemmodus durch alternative Register spezifisch für den jeweiligen Ausnahmemodus ersetzt wurden

Abbildung: Register je nach Ausführungsmodi [2]

Bedingungen

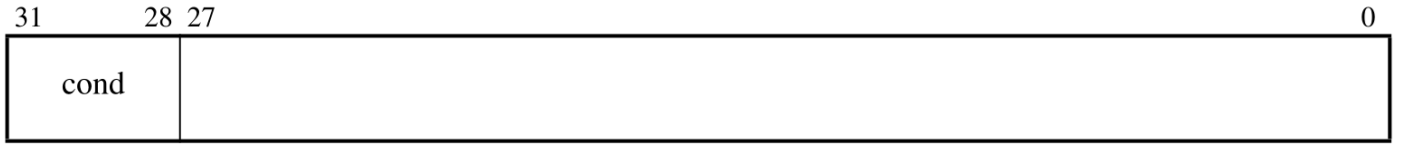
- Instruktionen können nur unter bestimmten Bedingungen ausgeführt werden

Bedingungen



- Instruktionen können nur unter bestimmten Bedingungen ausgeführt werden
- Höchste 4 Bits reserviert für Kodierung der Bedingung

Bedingungen

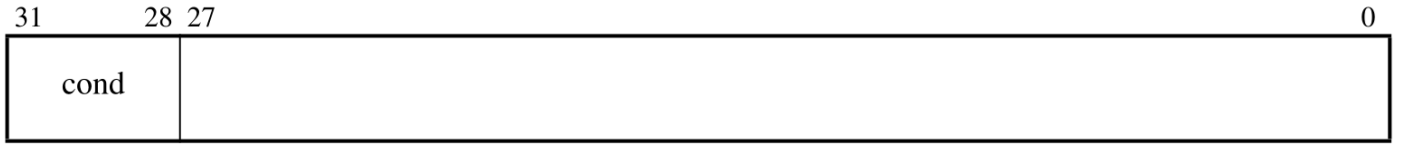


- Instruktionen können nur unter bestimmten Bedingungen ausgeführt werden
- Höchste 4 Bits reserviert für Kodierung der Bedingung
- Geknüpft an die Status-Flags NZCV, werden vor Ausführung der Bedingung überprüft

Kodierung [31:28]	Mnemonik	Bedeutung	Status–Flags
0000	EQ	Gleichheit (Equal)	Z=1
0001	NE	Ungleichheit (Unequal)	Z=0
0010	CS/HS	Carry-Bit gesetzt (Carry set)/ Vorzeichenlos größer oder gleich (unsigned higher or same)	C=1
0011	CC/LO	Carry-Bit nicht gesetzt (Carry clear)/ Vorzeichenlos kleiner (unsigned lower)	C=0
0100	MI	Negativ (Minus)	N=1
0101	PL	Positiv (Plus)	N=0
0110	VS	Überlauf (Overflow/V set)	V=1
0111	VC	Kein Überlauf (No Overflow/V clear)	V=0
1000	HI	Vorzeichenlos größer (Unsigned higher)	C=1, Z=0
1001	LS	Vorzeichenlos kleiner oder gleich (Unsigned lower or same)	C=0, Z=1

Abbildungen: Beispiele für Bedingungen und Kodierung [2]

Bedingungen



- Instruktionen können nur unter bestimmten Bedingungen ausgeführt werden
- Höchste 4 Bits reserviert für Kodierung der Bedingung
- Geknüpft an die Status-Flags NZCV, werden vor Ausführung der Bedingung überprüft
- Mnemonik an Instruktion im Code anhängen

Kodierung [31:28]	Mnemonik	Bedeutung	Status–Flags
0000	EQ	Gleichheit (Equal)	Z=1
0001	NE	Ungleichheit (Unequal)	Z=0
0010	CS/HS	Carry-Bit gesetzt (Carry set)/ Vorzeichenlos größer oder gleich (unsigned higher or same)	C=1
0011	CC/LO	Carry-Bit nicht gesetzt (Carry clear)/ Vorzeichenlos kleiner (unsigned lower)	C=0
0100	MI	Negativ (Minus)	N=1
0101	PL	Positiv (Plus)	N=0
0110	VS	Überlauf (Overflow/V set)	V=1
0111	VC	Kein Überlauf (No Overflow/V clear)	V=0
1000	HI	Vorzeichenlos größer (Unsigned higher)	C=1, Z=0
1001	LS	Vorzeichenlos kleiner oder gleich (Unsigned lower or same)	C=0, Z=1

Abbildungen: Beispiele für Bedingungen und Kodierung [2]

Datenverarbeitende Instruktionen

- Arithmetische- und Vergleichsoperationen

Datenverarbeitende Instruktionen

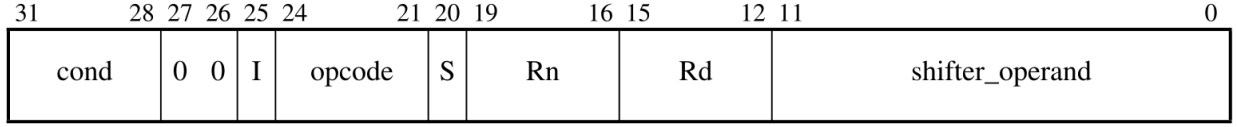
- Arithmetische- und Vergleichsoperationen
- Instruktionen über Befehlscode unterschieden

Befehlscode [24:21]	**Mnemonik**	**Operation**	**Aktion**
0000	AND	Logisches Und	$Rd := Rn$ AND $shift\_op$
0001	EOR	Logisches exklusives Oder	$Rd := Rn$ EOR $shift\_op$
0010	SUB	Subtraktion	$Rd := Rn - shift\_op$
0011	RSB	Umgekehrte Subtraktion	$Rd := shift\_op - Rn$
0100	ADD	Addition	$Rd := Rn + shift\_op$
0101	ADC	Addition mit Carry	$Rd := Rn + shift\_op + C$
0110	SBC	Subtraktion mit Carry	$Rd := Rn - shift\_op -$ NOT(C)
0111	RSC	Umgekehrte Subtraktion mit Carry	$Rd := shift\_op - Rn -$ NOT(C)
1000	TST	Test	Rn AND $shift\_op$ und aktualisiere Flags
1001	TEQ	Äquivalenztest	Rn EOR $shift\_op$ und aktualisiere Flags

Abbildungen: Datenverarbeitende Instruktionen [2]

Datenverarbeitende Instruktionen

- Arithmetische- und Vergleichsoperationen
- Instruktionen über Befehlscode unterschieden
- Register-Operanden *Rn* und *Rd*

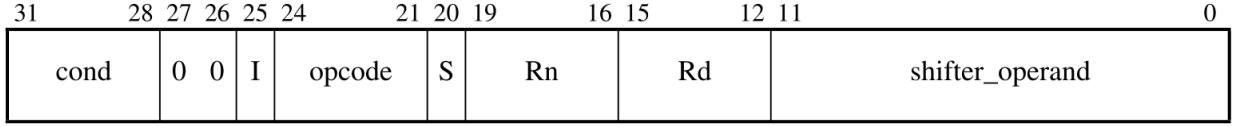


Befehlscode [24:21]	**Mnemonik**	**Operation**	**Aktion**
0000	AND	Logisches Und	$Rd := Rn$ AND $shift\_op$
0001	EOR	Logisches exklusives Oder	$Rd := Rn$ EOR $shift\_op$
0010	SUB	Subtraktion	$Rd := Rn - shift\_op$
0011	RSB	Umgekehrte Subtraktion	$Rd := shift\_op - Rn$
0100	ADD	Addition	$Rd := Rn + shift\_op$
0101	ADC	Addition mit Carry	$Rd := Rn + shift\_op + C$
0110	SBC	Subtraktion mit Carry	$Rd := Rn - shift\_op -$ NOT(C)
0111	RSC	Umgekehrte Subtraktion mit Carry	$Rd := shift\_op - Rn -$ NOT(C)
1000	TST	Test	Rn AND $shift\_op$ und aktualisiere Flags
1001	TEQ	Äquivalenztest	Rn EOR $shift\_op$ und aktualisiere Flags

Abbildungen: Datenverarbeitende Instruktionen [2]

Datenverarbeitende Instruktionen

- Arithmetische- und Vergleichsoperationen
- Instruktionen über Befehlscode unterschieden
- Register-Operanden *Rn* und *Rd*
- Flexibler dritter Operand, der Zugriff auf Barrel-Shifter hat

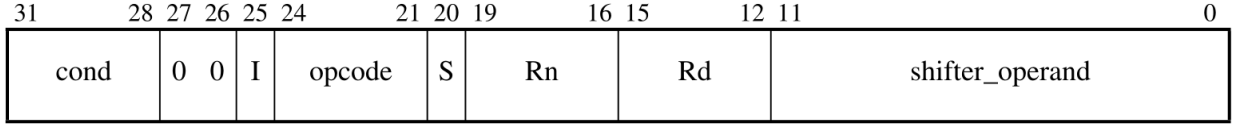


Befehlscode [24:21]	Mnemonik	Operation	Aktion
0000	AND	Logisches Und	$Rd := Rn$ AND $shift\_op$
0001	EOR	Logisches exklusives Oder	$Rd := Rn$ EOR $shift\_op$
0010	SUB	Subtraktion	$Rd := Rn - shift\_op$
0011	RSB	Umgekehrte Subtraktion	$Rd := shift\_op - Rn$
0100	ADD	Addition	$Rd := Rn + shift\_op$
0101	ADC	Addition mit Carry	$Rd := Rn + shift\_op + C$
0110	SBC	Subtraktion mit Carry	$Rd := Rn - shift\_op -$ NOT(C)
0111	RSC	Umgekehrte Subtraktion mit Carry	$Rd := shift\_op - Rn -$ NOT(C)
1000	TST	Test	Rn AND $shift\_op$ und aktualisiere Flags
1001	TEQ	Äquivalenztest	Rn EOR $shift\_op$ und aktualisiere Flags

Abbildungen: Datenverarbeitende Instruktionen [2]

Datenverarbeitende Instruktionen

- Arithmetische- und Vergleichsoperationen
- Instruktionen über Befehlscode unterschieden
- Register-Operanden *Rn* und *Rd*
- Flexibler dritter Operand, der Zugriff auf Barrel-Shifter hat
- S-Bit aktualisiert die Status-Flags



Befehlscode [24:21]	Mnemonik	Operation	Aktion
0000	AND	Logisches Und	$Rd := Rn$ AND $shift\_op$
0001	EOR	Logisches exklusives Oder	$Rd := Rn$ EOR $shift\_op$
0010	SUB	Subtraktion	$Rd := Rn - shift\_op$
0011	RSB	Umgekehrte Subtraktion	$Rd := shift\_op - Rn$
0100	ADD	Addition	$Rd := Rn + shift\_op$
0101	ADC	Addition mit Carry	$Rd := Rn + shift\_op + C$
0110	SBC	Subtraktion mit Carry	$Rd := Rn - shift\_op -$ NOT(C)
0111	RSC	Umgekehrte Subtraktion mit Carry	$Rd := shift\_op - Rn -$ NOT(C)
1000	TST	Test	Rn AND $shift\_op$ und aktualisiere Flags
1001	TEQ	Äquivalenztest	Rn EOR $shift\_op$ und aktualisiere Flags

Abbildungen: Datenverarbeitende Instruktionen [2]

Shifter-Operand

- Immediate-Wert (8 Bit + Rotation)

Shifter-Operand

- Immediate-Wert (8 Bit + Rotation)
- Register

Shifter-Operand

- Immediate-Wert (8 Bit + Rotation)
- Register
- Verschiebeoperation

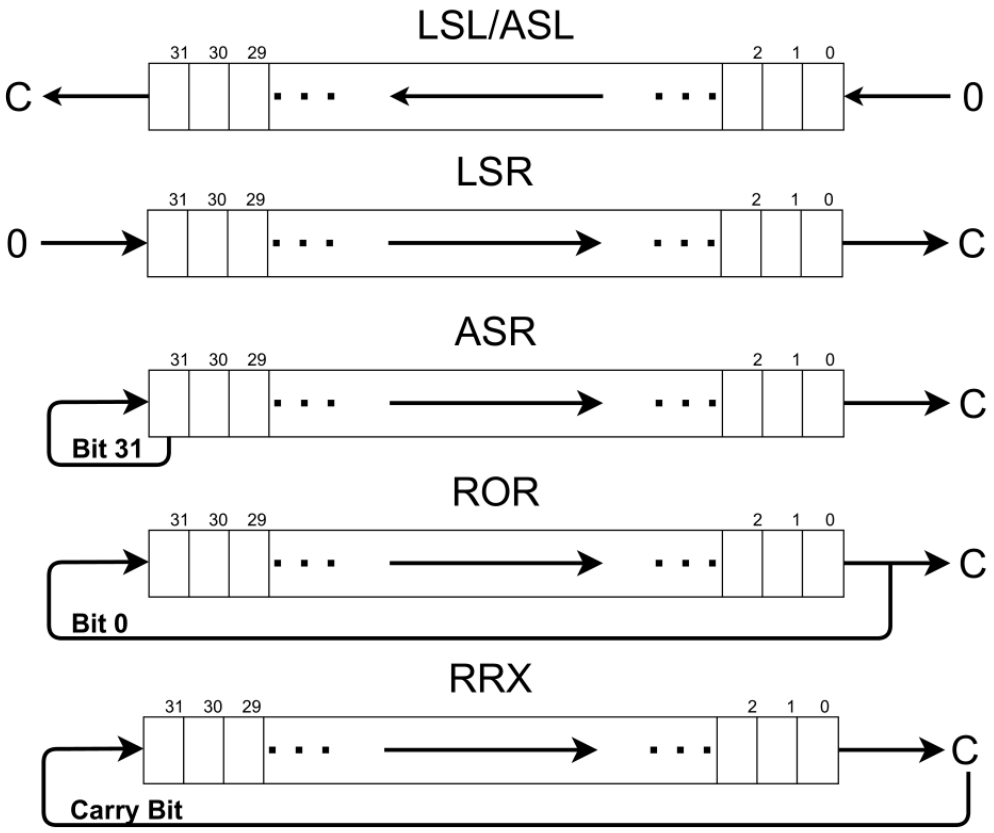


Abbildung: Verschiebeoperationen [8]

Shifter-Operand

- Immediate-Wert (8 Bit + Rotation)
- Register
- Verschiebeoperation
  - Logische Linksverschiebung

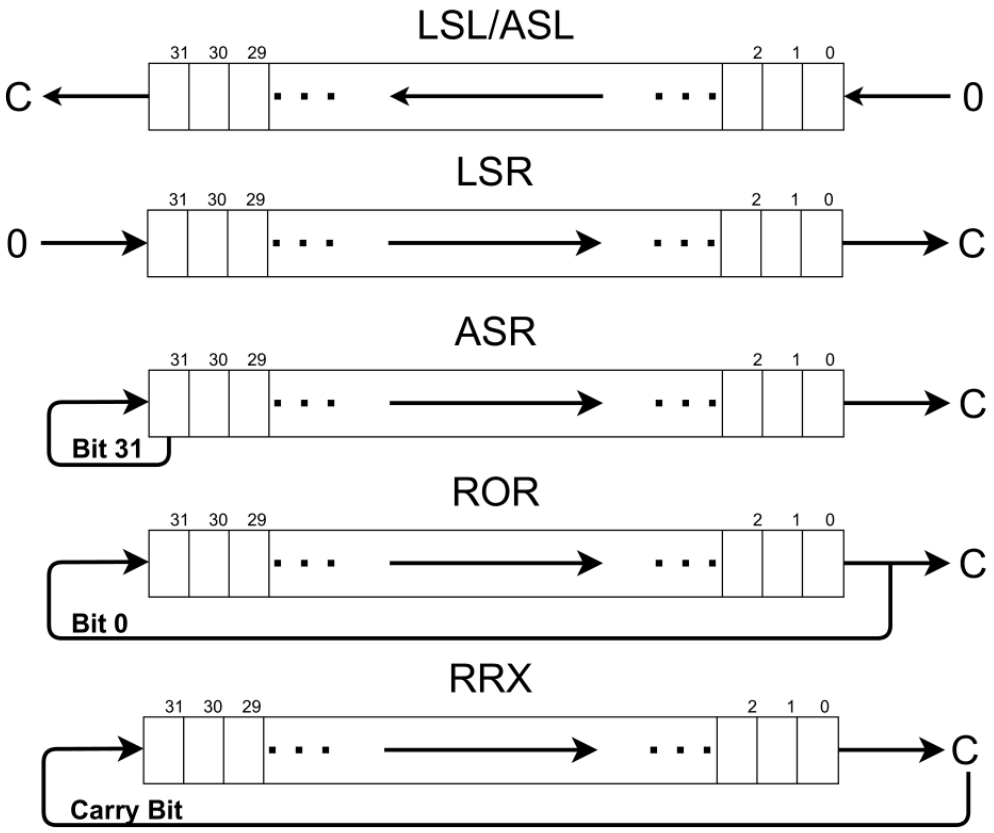


Abbildung: Verschiebeoperationen [8]

Shifter-Operand

- Immediate-Wert (8 Bit + Rotation)
- Register
- Verschiebeoperation
  - Logische Linksverschiebung
  - Arithmetische/Logische Rechtsverschiebung

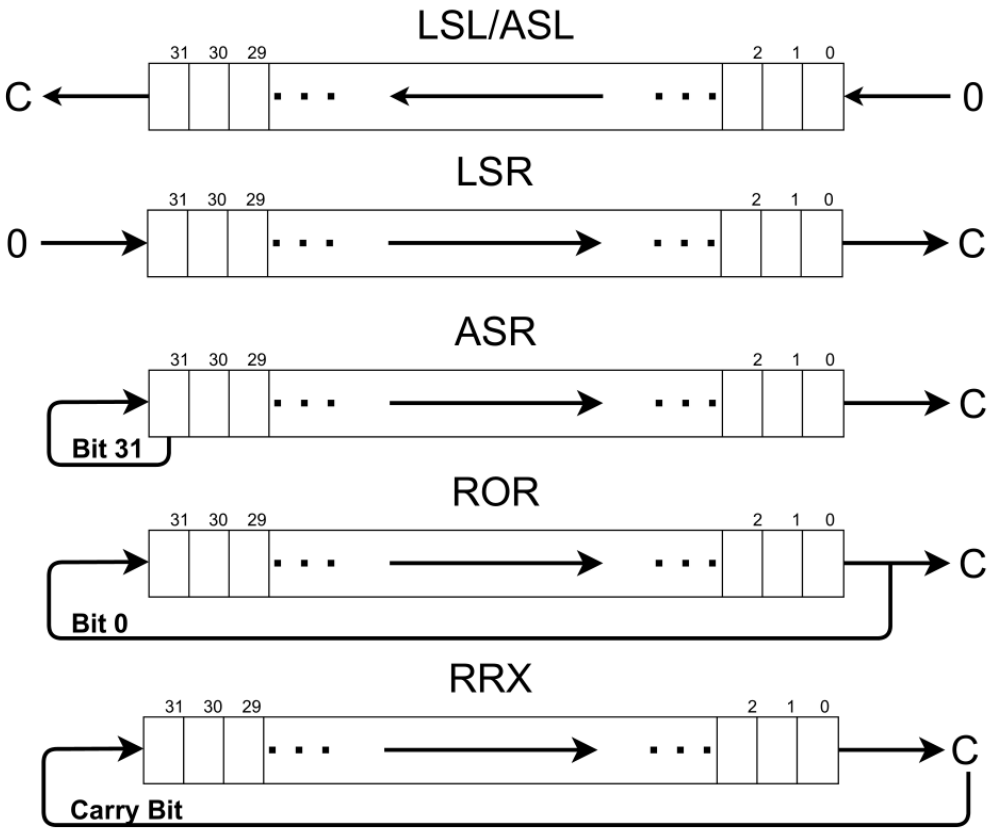


Abbildung: Verschiebeoperationen [8]

Shifter-Operand

- Immediate-Wert (8 Bit + Rotation)
- Register
- Verschiebeoperation
  - Logische Linksverschiebung
  - Arithmetische/Logische Rechtsverschiebung
  - Rechtsrotation

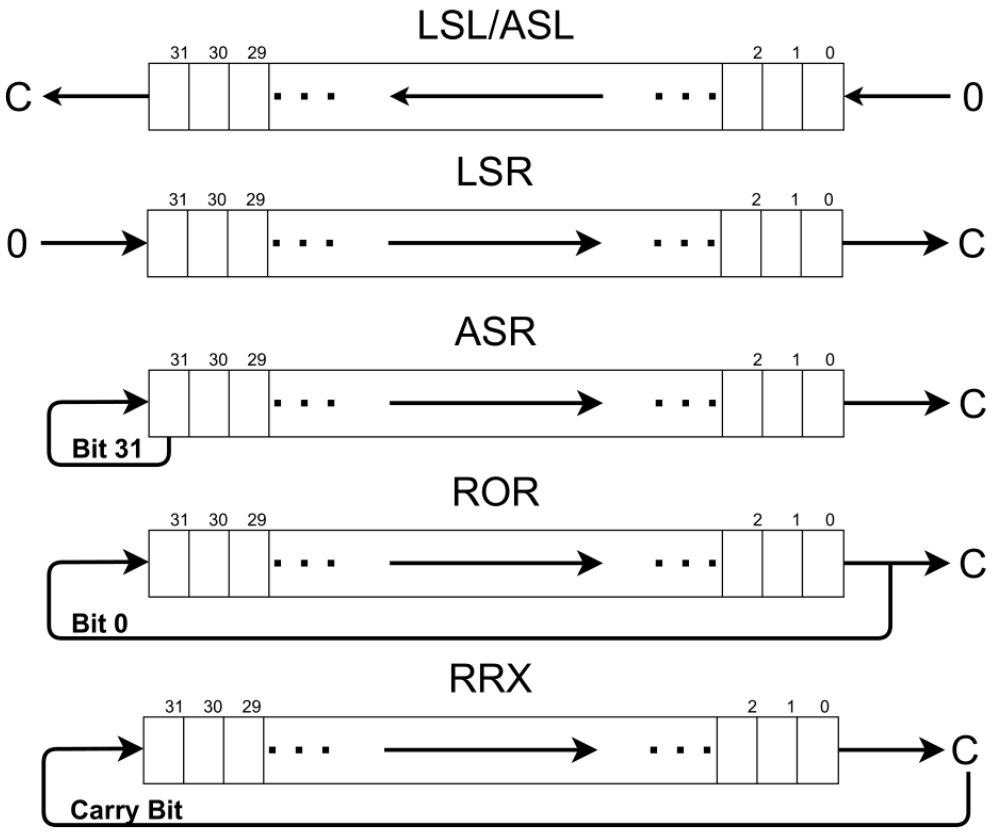


Abbildung: Verschiebeoperationen [8]

Shifter-Operand

- Immediate-Wert (8 Bit + Rotation)
- Register
- Verschiebeoperation
  - Logische Linksverschiebung
  - Arithmetische/Logische Rechtsverschiebung
  - Rechtsrotation
  - Erweiterte Rechtsrotation

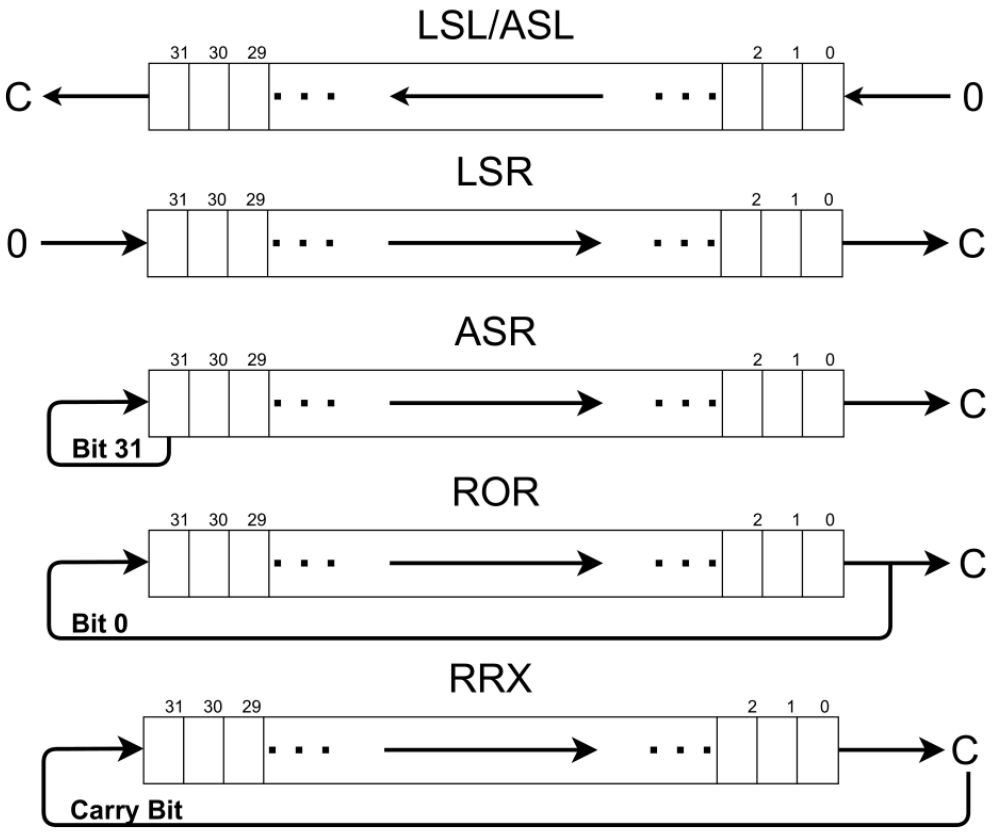


Abbildung: Verschiebeoperationen [8]

Sprunginstruktionen

- Adressen im Hauptspeicher können Labels zugewiesen werden

Sprunginstruktionen

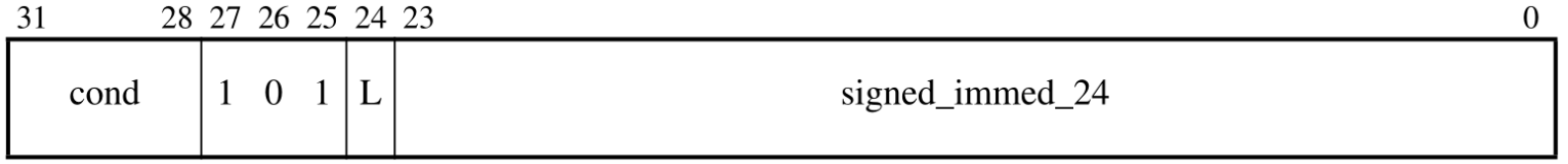


Abbildung: Kodierung Sprungoperation [2]

- Adressen im Hauptspeicher können Labels zugewiesen werden
- Operationen können Sprünge zu diesen Labels durchführen

Sprunginstruktionen

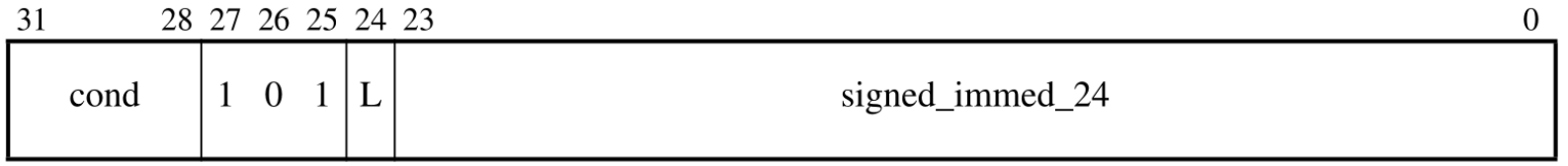


Abbildung: Kodierung Sprungoperation [2]

- Adressen im Hauptspeicher können Labels zugewiesen werden
- Operationen können Sprünge zu diesen Labels durchführen
  - Adresse des Labels wird in Register für Befehlszähler kopieren

Sprunginstruktionen

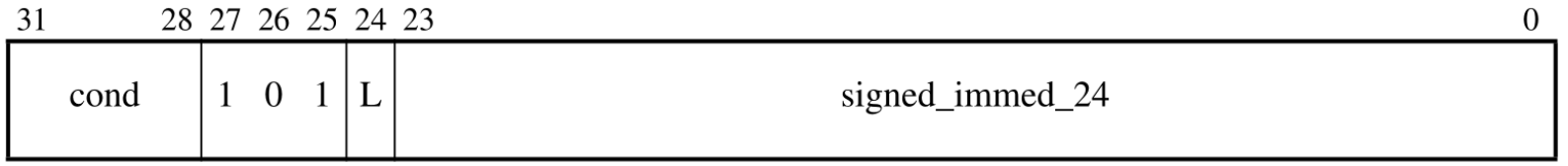


Abbildung: Kodierung Sprungoperation [2]

- Adressen im Hauptspeicher können Labels zugewiesen werden
- Operationen können Sprünge zu diesen Labels durchführen
  - Adresse des Labels wird in Register für Befehlszähler kopieren
  - Abstand zu aktueller Adresse berechnet und in *signed_immed_24* gespeichert

Sprunginstruktionen

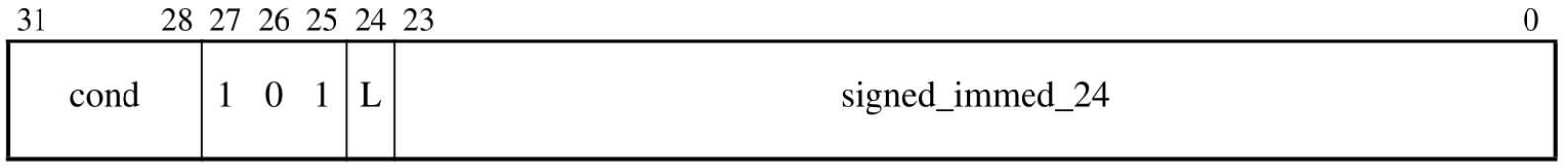


Abbildung: Kodierung Sprungoperation [2]

- Adressen im Hauptspeicher können Labels zugewiesen werden
- Operationen können Sprünge zu diesen Labels durchführen
  - Adresse des Labels wird in Register für Befehlszähler kopieren
  - Abstand zu aktueller Adresse berechnet und in *signed_immed_24* gespeichert
  - L-Bit zum Hinterlegen der Rücksprungadresse im Link-Register

Lade- und Speicherinstruktionen

- Lädt Inhalt von Adresse im Hauptspeicher in Zielregister oder speichert Inhalt eines Registers in den Hauptspeicher

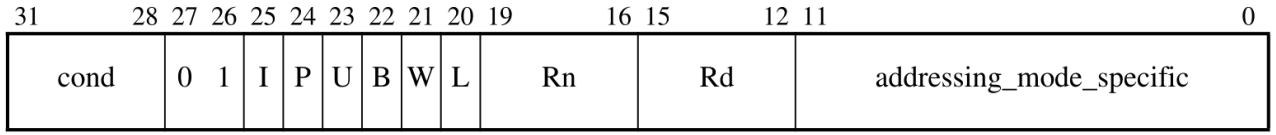


Abbildung: Kodierung Lade- und Speicherinstruktionen [2]

Lade- und Speicherinstruktionen

- Lädt Inhalt von Adresse im Hauptspeicher in Zielregister oder speichert Inhalt eines Registers in den Hauptspeicher
- Auch Laden/Speichern von Halbwörtern und Bytes möglich

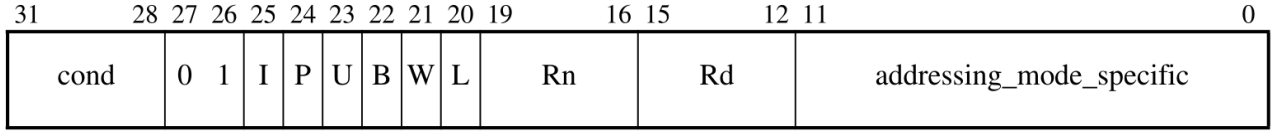


Abbildung: Kodierung Lade- und Speicherinstruktionen [2]

Lade- und Speicherinstruktionen

- Lädt Inhalt von Adresse im Hauptspeicher in Zielregister oder speichert Inhalt eines Registers in den Hauptspeicher
- Auch Laden/Speichern von Halbwörtern und Bytes möglich
- Adressierungsart:

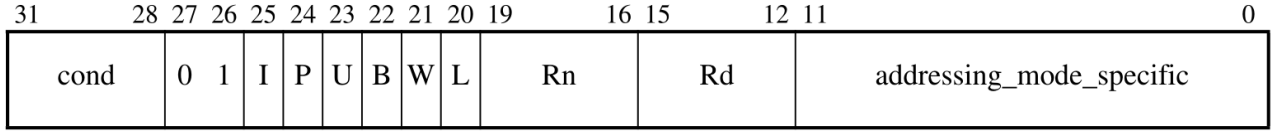


Abbildung: Kodierung Lade- und Speicherinstruktionen [2]

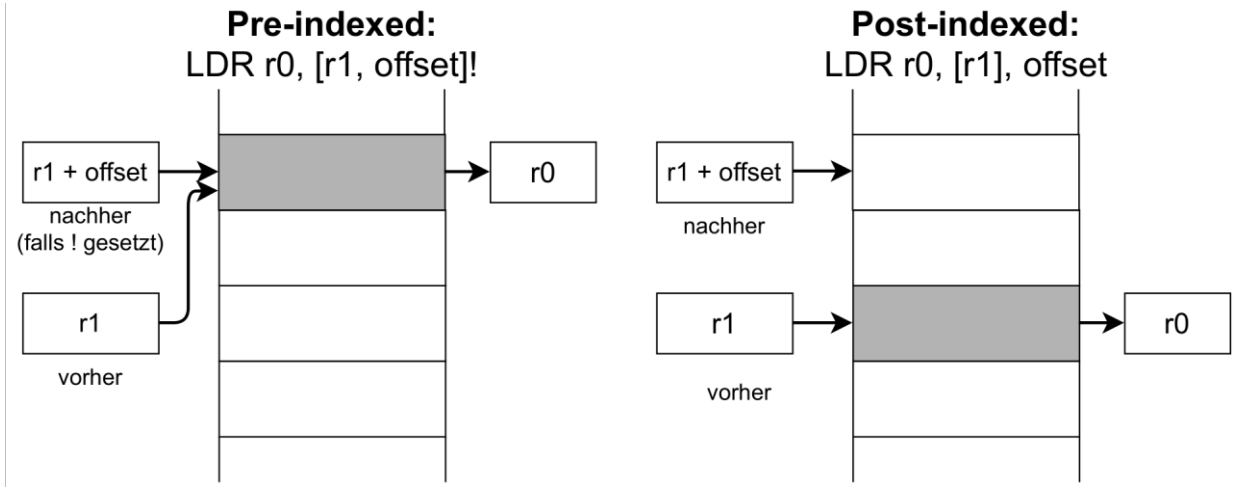


Abbildung: Adressierungsarten [8]

Lade- und Speicherinstruktionen

- Lädt Inhalt von Adresse im Hauptspeicher in Zielregister oder speichert Inhalt eines Registers in den Hauptspeicher
- Auch Laden/Speichern von Halbwörtern und Bytes möglich
- Adressierungsart:
  - Offset

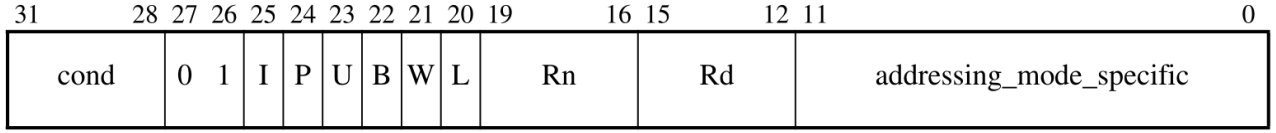


Abbildung: Kodierung Lade- und Speicherinstruktionen [2]

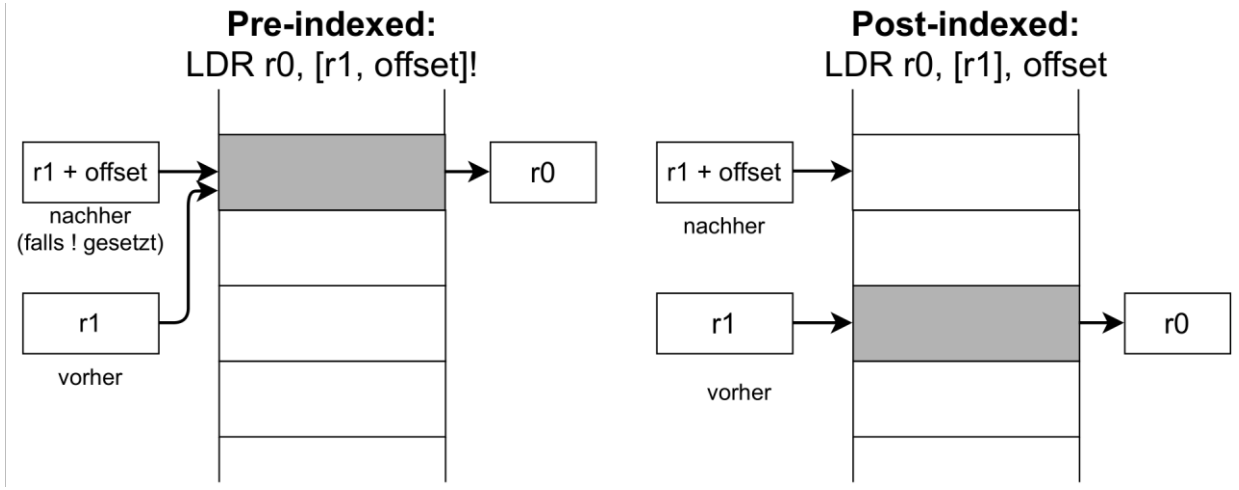


Abbildung: Adressierungsarten [8]

Lade- und Speicherinstruktionen

- Lädt Inhalt von Adresse im Hauptspeicher in Zielregister oder speichert Inhalt eines Registers in den Hauptspeicher
- Auch Laden/Speichern von Halbwörtern und Bytes möglich
- Adressierungsart:
  - Offset
  - Pre- oder Post-indexed

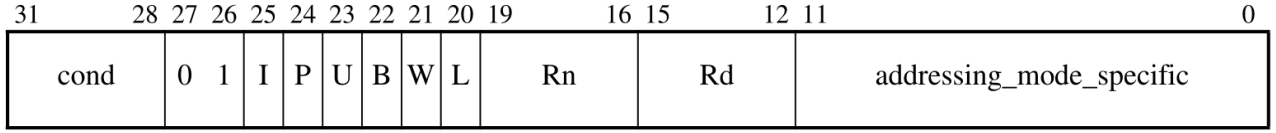


Abbildung: Kodierung Lade- und Speicherinstruktionen [2]

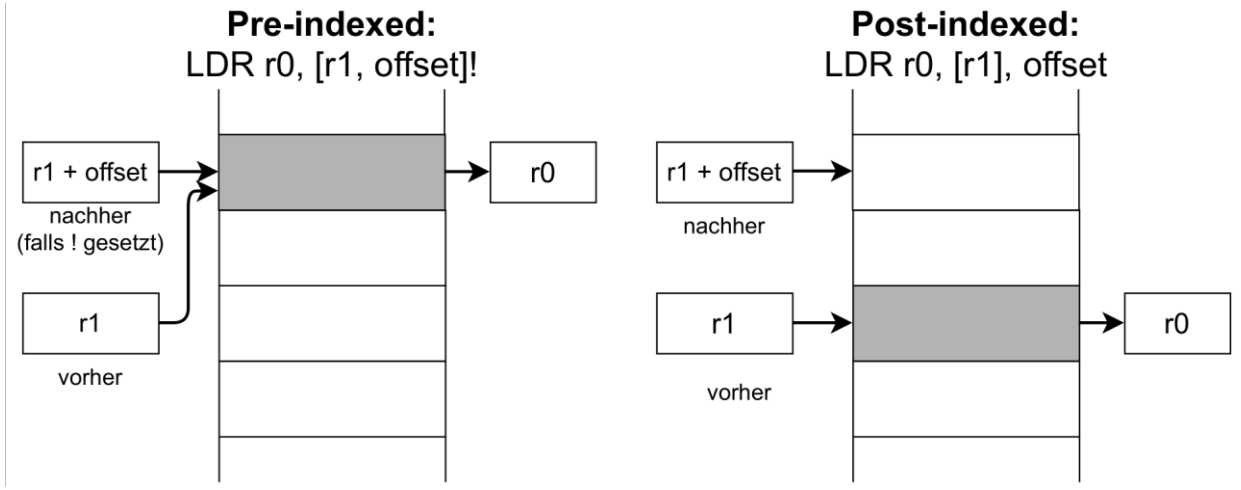


Abbildung: Adressierungsarten [8]

Lade- und Speicherinstruktionen

- Lädt Inhalt von Adresse im Hauptspeicher in Zielregister oder speichert Inhalt eines Registers in den Hauptspeicher
- Auch Laden/Speichern von Halbwörtern und Bytes möglich
- Adressierungsart:
  - Offset
  - Pre- oder Post-indexed
  - Auto-Inkrement

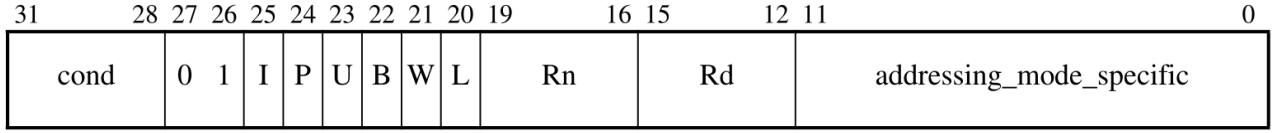


Abbildung: Kodierung Lade- und Speicherinstruktionen [2]

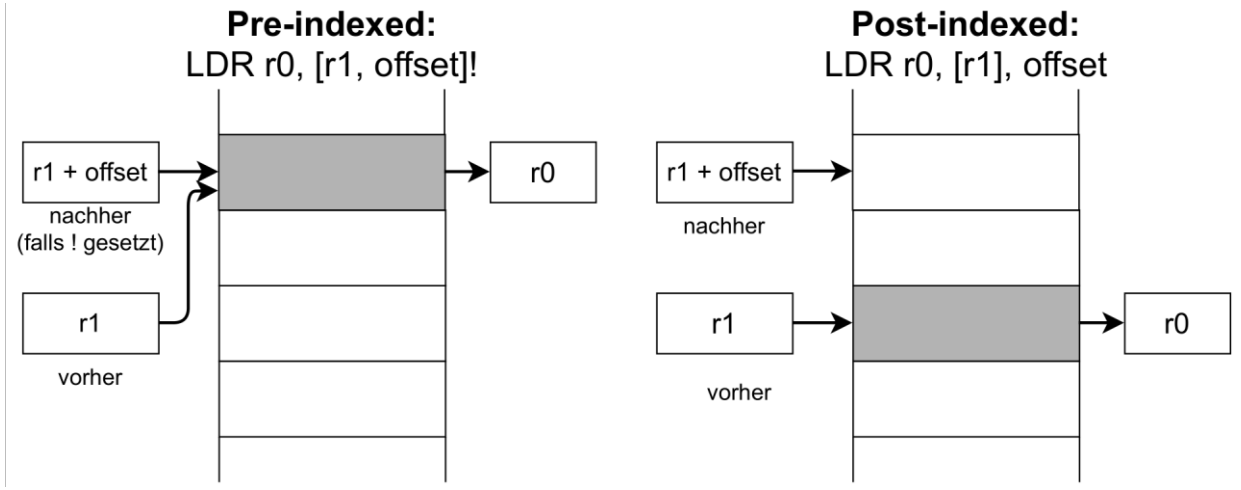


Abbildung: Adressierungsarten [8]

Lade- und Speicherinstruktionen für mehrere Register

- Register werden als Liste angegeben und dann ausgehend vom Basisregister *Rn* geladen oder geschrieben

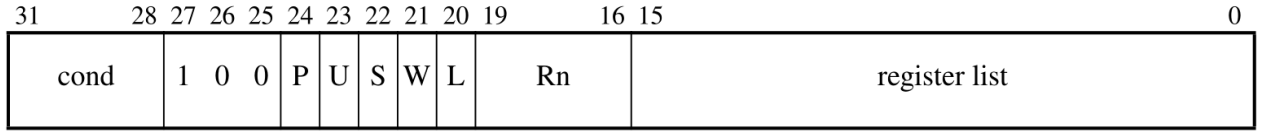


Abbildung: Kodierung Lade- und Speicherinstruktionen für mehrere Register [2]

Lade- und Speicherinstruktionen für mehrere Register

- Register werden als Liste angegeben und dann ausgehend vom Basisregister *Rn* geladen oder geschrieben
- Adressierungsarten **IA/IB/DA/DB**:

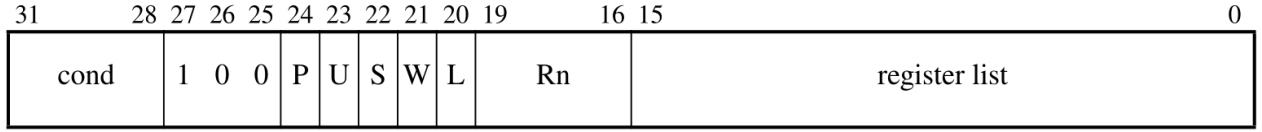


Abbildung: Kodierung Lade- und Speicherinstruktionen für mehrere Register [2]

Lade- und Speicherinstruktionen für mehrere Register

- Register werden als Liste angegeben und dann ausgehend vom Basisregister *Rn* geladen oder geschrieben
- Adressierungsarten **IA/IB/DA/DB**:
  - **I**ncrement/**D**ecrement: Basisregister wird erhöht/verringert

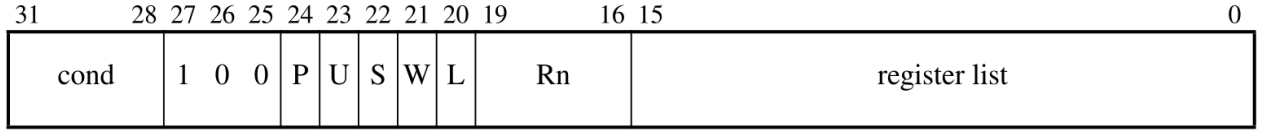


Abbildung: Kodierung Lade- und Speicherinstruktionen für mehrere Register [2]

Lade- und Speicherinstruktionen für mehrere Register

- Register werden als Liste angegeben und dann ausgehend vom Basisregister *Rn* geladen oder geschrieben
- Adressierungsarten **IA/IB/DA/DB**:
  - **I**ncrement/**D**ecrement: Basisregister wird erhöht/verringert
  - **A**fter/**B**efore: Aktualisierung des Basisregister vor/nach Laden oder Speichern

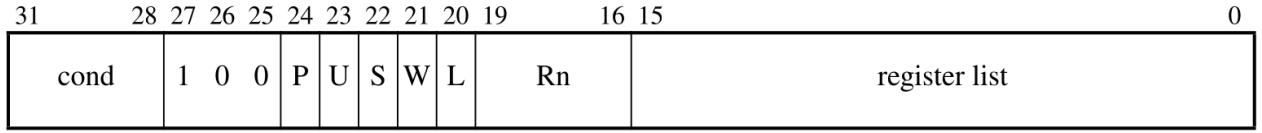


Abbildung: Kodierung Lade- und Speicherinstruktionen für mehrere Register [2]

Lade- und Speicherinstruktionen für mehrere Register

- Register werden als Liste angegeben und dann ausgehend vom Basisregister *Rn* geladen oder geschrieben
- Adressierungsarten **IA/IB/DA/DB**:
  - **I**ncrement/**D**ecrement: Basisregister wird erhöht/verringert
  - **A**fter/**B**efore: Aktualisierung des Basisregister vor/nach Laden oder Speichern
- Alternative Stack-Adressierungsarten **FA/FD/EA/ED**:

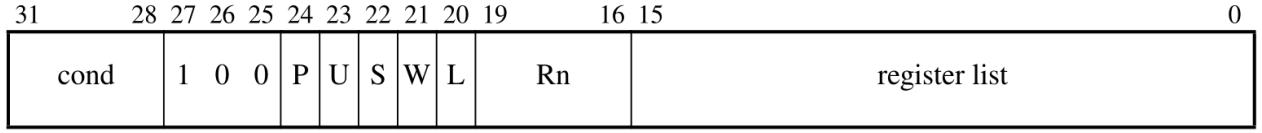


Abbildung: Kodierung Lade- und Speicherinstruktionen für mehrere Register [2]

Lade- und Speicherinstruktionen für mehrere Register

- Register werden als Liste angegeben und dann ausgehend vom Basisregister *Rn* geladen oder geschrieben
- Adressierungsarten **IA/IB/DA/DB**:
  - **I**ncrement/**D**ecrement: Basisregister wird erhöht/verringert
  - **A**fter/**B**efore: Aktualisierung des Basisregister vor/nach Laden oder Speichern
- Alternative Stack-Adressierungsarten **FA/FD/EA/ED**:
  - **A**scending/**D**escending: Basisregister wird erhöht/verringert

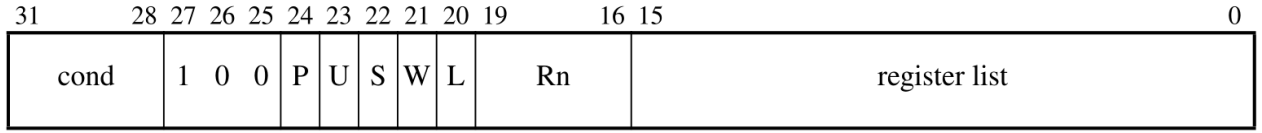


Abbildung: Kodierung Lade- und Speicherinstruktionen für mehrere Register [2]

Lade- und Speicherinstruktionen für mehrere Register

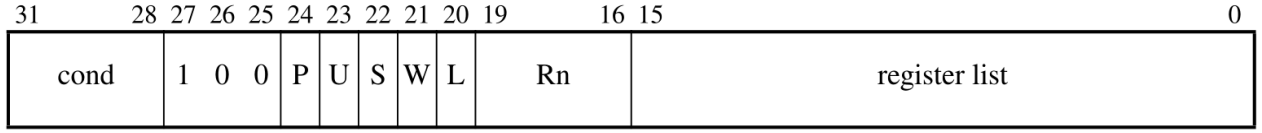


Abbildung: Kodierung Lade- und Speicherinstruktionen für mehrere Register [2]

- Register werden als Liste angegeben und dann ausgehend vom Basisregister *Rn* geladen oder geschrieben
- Adressierungsarten **IA/IB/DA/DB**:
  - **I**ncrement/**D**ecrement: Basisregister wird erhöht/verringert
  - **A**fter/**B**efore: Aktualisierung des Basisregister vor/nach Laden oder Speichern
- Alternative Stack-Adressierungsarten **FA/FD/EA/ED**:
  - **A**scending/**D**escending: Basisregister wird erhöht/verringert
  - **F**ull/**E**mpty: Stapelzeiger weist auf gefüllte/leere Adresse

Parsing Expression Grammatik

- Ungeordneter Alternativen-Operator | bei kontext-freien Grammatiken

CFG	PEG
1. A --> a | a b	1. A <-- a / a b
2. A --> a b | a	2. A <-- a b / a

Abbildung: Vergleich kontextfreie Grammatik und Parsing Expression Grammatik [7]

Parsing Expression Grammatik

CFG	PEG
1. A --> a | a b	1. A <-- a / a b
2. A --> a b | a	2. A <-- a b / a

Abbildung: Vergleich kontextfreie Grammatik und Parsing Expression Grammatik [7]

- Ungeordneter Alternativen-Operator | bei kontext-freien Grammatiken
  - Ungeordnet – Definition 1 und 2 gleichwertig

Parsing Expression Grammatik

CFG	PEG
1. A --> a | a b	1. A <-- a / a b
2. A --> a b | a	2. A <-- a b / a

Abbildung: Vergleich kontextfreie Grammatik und Parsing Expression Grammatik [7]

- Ungeordneter Alternativen-Operator | bei kontext-freien Grammatiken
  - Ungeordnet – Definition 1 und 2 gleichwertig
  - Mehrdeutigkeiten – Parser muss alle Alternativen betrachten

Parsing Expression Grammatik

CFG	PEG
1. A --> a | a b	1. A <-- a / a b
2. A --> a b | a	2. A <-- a b / a

Abbildung: Vergleich kontextfreie Grammatik und Parsing Expression Grammatik [7]

- Ungeordneter Alternativen-Operator | bei kontext-freien Grammatiken
  - Ungeordnet – Definition 1 und 2 gleichwertig
  - Mehrdeutigkeiten – Parser muss alle Alternativen betrachten

- Alternativen-Operator mit Priorität / bei Parsing Expression Grammatiken

Parsing Expression Grammatik

CFG	PEG
1. A --> a | a b	1. A <-- a / a b
2. A --> a b | a	2. A <-- a b / a

Abbildung: Vergleich kontextfreie Grammatik und Parsing Expression Grammatik [7]

- Ungeordneter Alternativen-Operator | bei kontext-freien Grammatiken
  - Ungeordnet – Definition 1 und 2 gleichwertig
  - Mehrdeutigkeiten – Parser muss alle Alternativen betrachten

- Alternativen-Operator mit Priorität / bei Parsing Expression Grammatiken
  - Alternativen mit absteigender Priorität geordnet

Parsing Expression Grammatik

CFG	PEG
1. A --> a | a b	1. A <-- a / a b
2. A --> a b | a	2. A <-- a b / a

Abbildung: Vergleich kontextfreie Grammatik und Parsing Expression Grammatik [7]

- Ungeordneter Alternativen-Operator | bei kontext-freien Grammatiken
  - Ungeordnet – Definition 1 und 2 gleichwertig
  - Mehrdeutigkeiten – Parser muss alle Alternativen betrachten

- Alternativen-Operator mit Priorität / bei Parsing Expression Grammatiken
  - Alternativen mit absteigender Priorität geordnet
  - Effizienter – Parser kann nach gefundenem Match stoppen

tsPEG

- tsPEG [5] ist ein Parser-Generator für TypeScript [10]

tsPEG

```
start := helloChoice

helloChoice := hello planet='Planet[0-9]' | helloWorld
helloWorld := hello planet='World'
hello := 'Hello '
```

Abbildung: Hello World Beispiel für tsPEG [5]

- tsPEG [5] ist ein Parser-Generator für TypeScript [10]
  - Definiere Regeln von PEGs mit :=

tsPEG

```
start := helloChoice

helloChoice := hello planet='Planet[0-9]' | helloWorld
helloWorld := hello planet='World'
hello := 'Hello '
```

Abbildung: Hello World Beispiel für tsPEG [5]

- tsPEG [5] ist ein Parser-Generator für TypeScript [10]
  - Definiere Regeln von PEGs mit :=
  - Zuweisen von Variablen mit =

tsPEG

```
start := helloChoice

helloChoice := hello planet='Planet[0-9]' | helloWorld
helloWorld := hello planet='World'
hello := 'Hello '
```

Abbildung: Hello World Beispiel für tsPEG [5]

- tsPEG [5] ist ein Parser-Generator für TypeScript [10]
  - Definiere Regeln von PEGs mit :=
  - Zuweisen von Variablen mit =
  - Reguläre Ausdrücke oder Strings zum Matchen

tsPEG

```
start := helloChoice

helloChoice := hello planet='Planet[0-9]' | helloWorld
helloWorld := hello planet='World'
hello := 'Hello '
```

Abbildung: Hello World Beispiel für tsPEG [5]

- tsPEG [5] ist ein Parser-Generator für TypeScript [10]
  - Definiere Regeln von PEGs mit :=
  - Zuweisen von Variablen mit =
  - Reguläre Ausdrücke oder Strings zum Matchen

- Erfolgreiches Parsen:

tsPEG

```
start := helloChoice

helloChoice := hello planet='Planet[0-9]' | helloWorld
helloWorld := hello planet='World'
hello := 'Hello '
```

Abbildung: Hello World Beispiel für tsPEG [5]

- tsPEG [5] ist ein Parser-Generator für TypeScript [10]
  - Definiere Regeln von PEGs mit :=
  - Zuweisen von Variablen mit =
  - Reguläre Ausdrücke oder Strings zum Matchen

- Erfolgreiches Parsen:
  - Generiert abstrakten Syntax-Baum aus zugewiesenen Variablen in TypeScript

Aufbau

- Klassen für Operanden

Aufbau

- Klassen für Operanden
  - Instruktionen aufgebaut aus Operanden

Aufbau

- Klassen für Operanden
  - Instruktionen aufgebaut aus Operanden

- Einteilung:

Aufbau

- Klassen für Operanden
  - Instruktionen aufgebaut aus Operanden

- Einteilung:
  - CPU

Aufbau

- Klassen für Operanden
  - Instruktionen aufgebaut aus Operanden

- Einteilung:
  - CPU
  - Parser

Aufbau

- Klassen für Operanden
  - Instruktionen aufgebaut aus Operanden

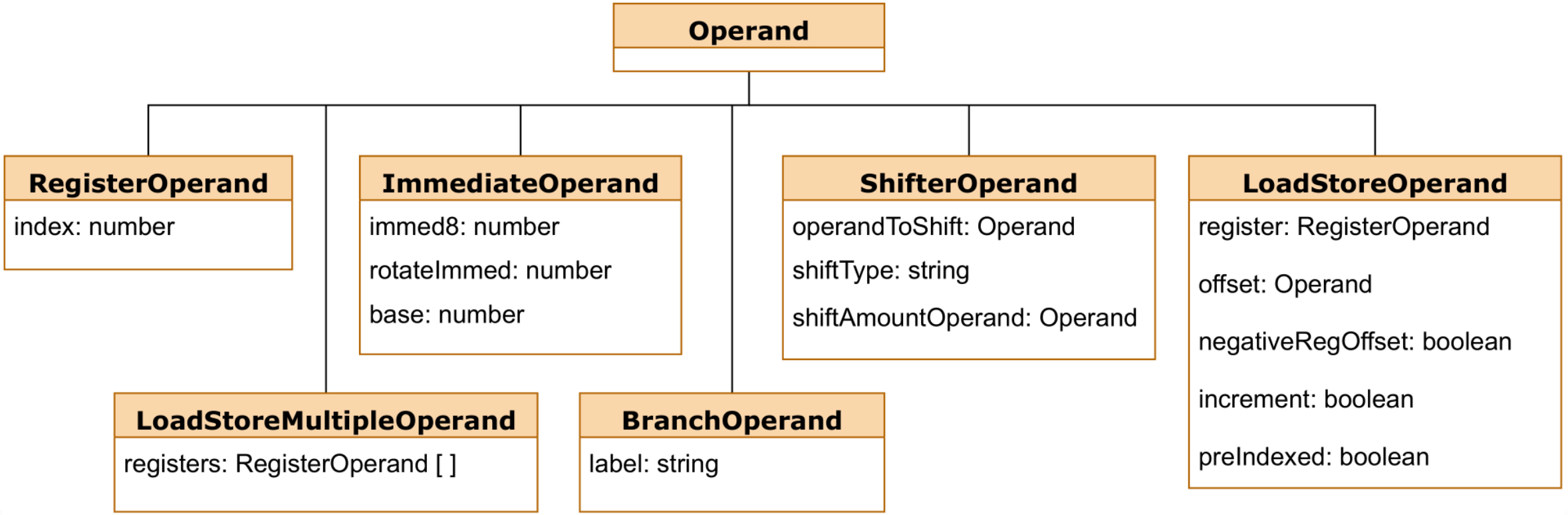
- Einteilung:
  - CPU
  - Parser
  - Hauptspeicher

Aufbau

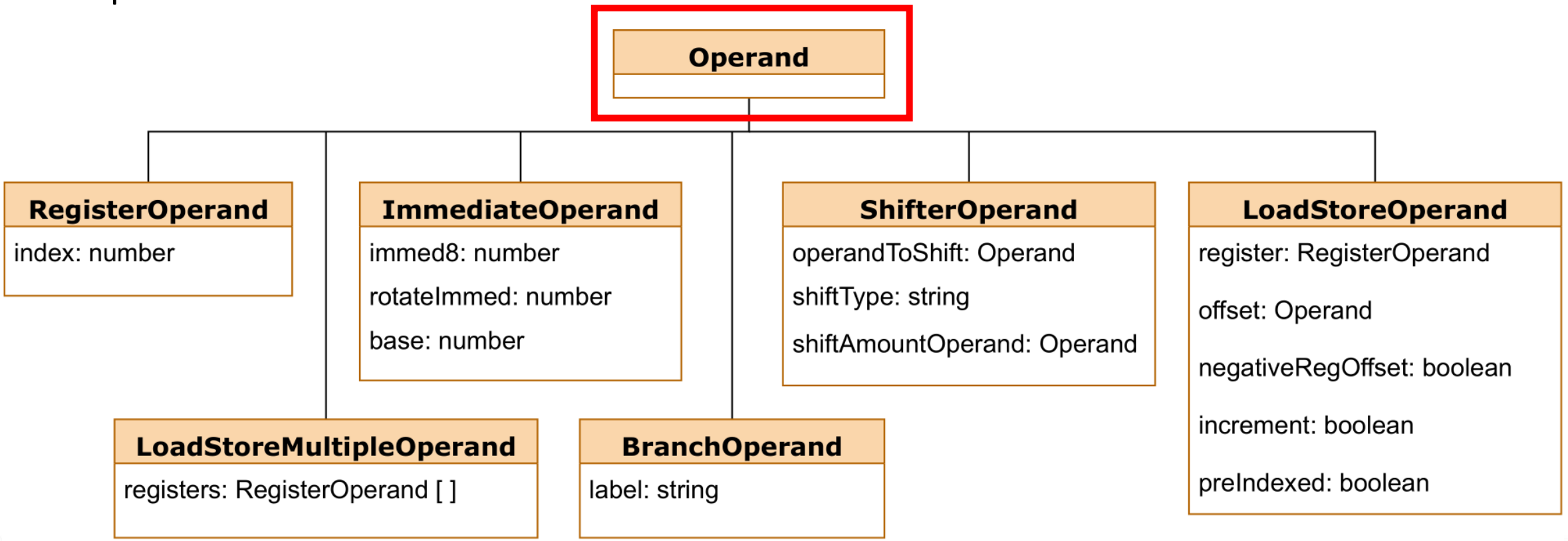
- Klassen für Operanden
  - Instruktionen aufgebaut aus Operanden

- Einteilung:
  - CPU
  - Parser
  - Hauptspeicher
  - Code-Ausführung

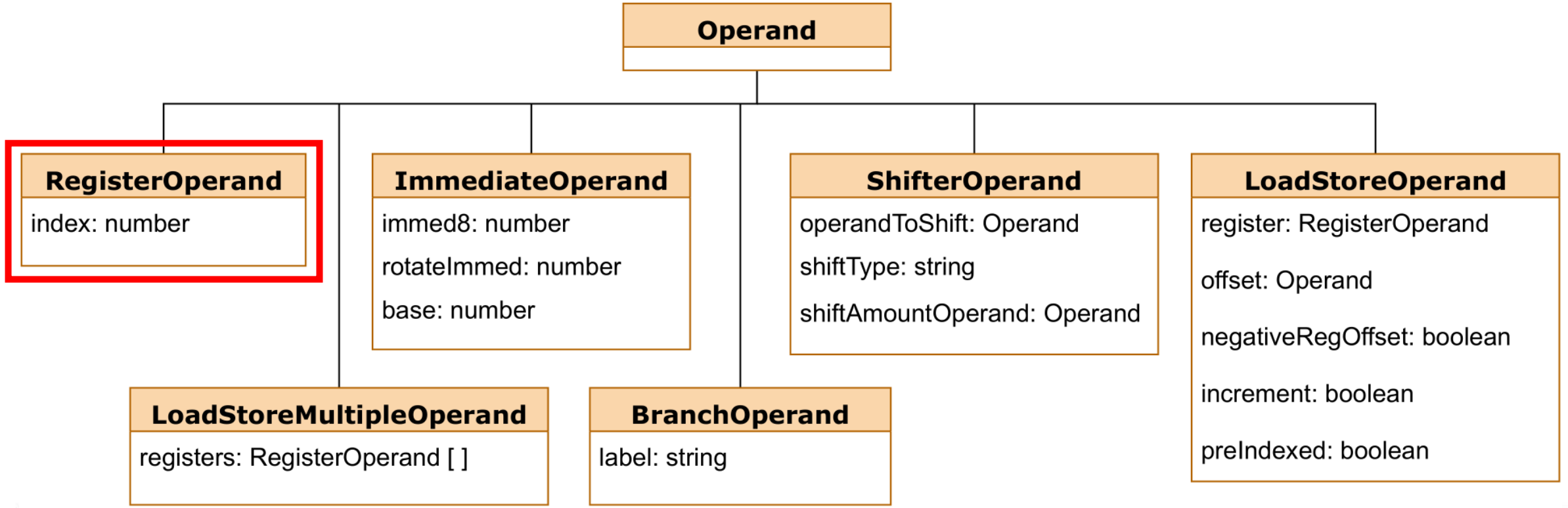
Operanden


Operand
RegisterOperand
index: number
ImmediateOperand
immed8: number
rotateImmed: number
base: number
ShifterOperand
operandToShift: Operand
shiftType: string
shiftAmountOperand: Operand
LoadStoreOperand
register: RegisterOperand
offset: Operand
negativeRegOffset: boolean
increment: boolean
preIndexed: boolean
LoadStoreMultipleOperand
registers: RegisterOperand [ ]
BranchOperand
label: string

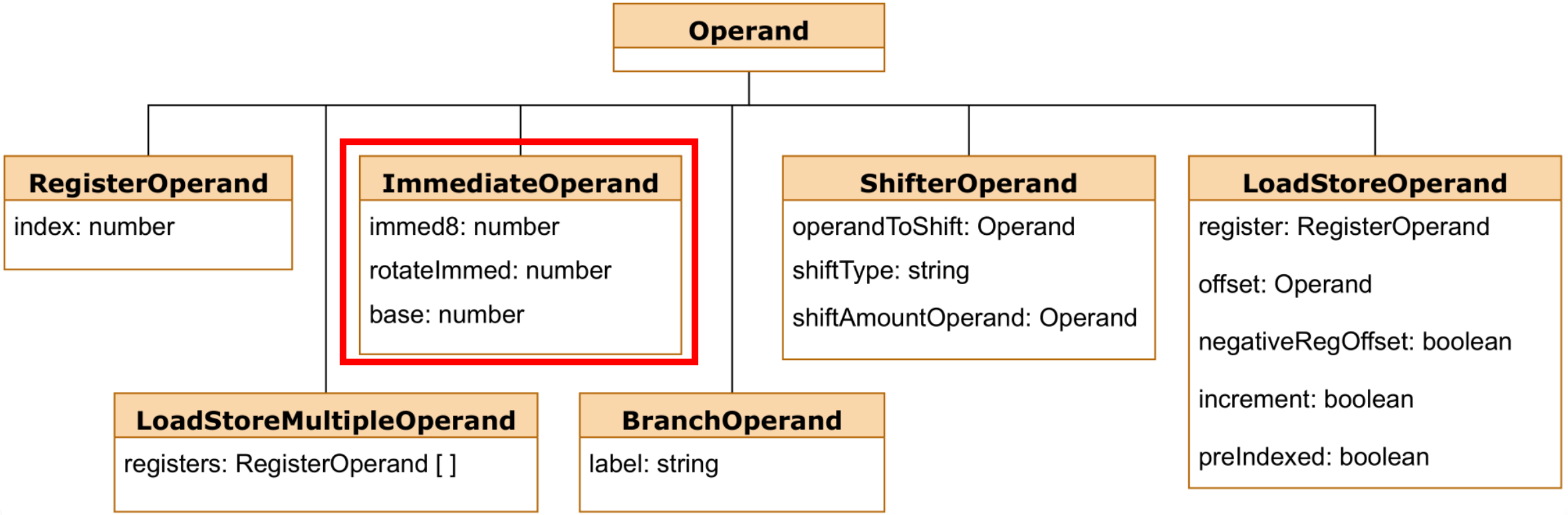
Operanden


Operand
RegisterOperand
index: number
ImmediateOperand
immed8: number
rotateImmed: number
base: number
ShifterOperand
operandToShift: Operand
shiftType: string
shiftAmountOperand: Operand
LoadStoreOperand
register: RegisterOperand
offset: Operand
negativeRegOffset: boolean
increment: boolean
preIndexed: boolean
LoadStoreMultipleOperand
registers: RegisterOperand [ ]
BranchOperand
label: string

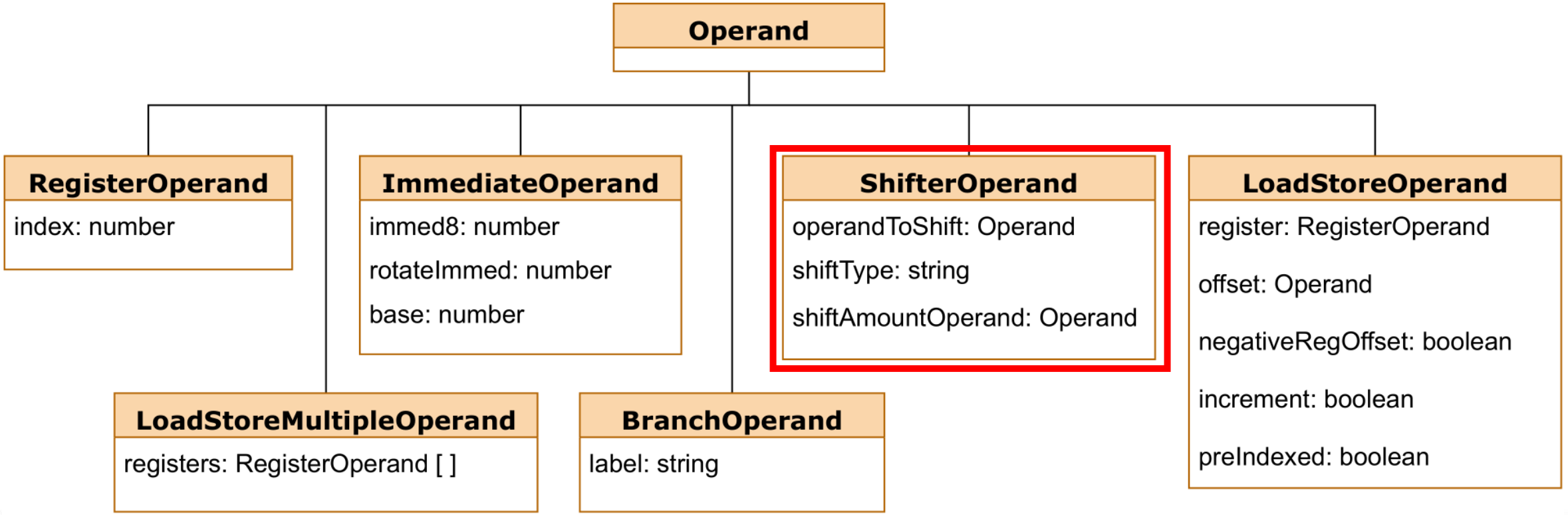
Operanden


Operand
RegisterOperand
index: number
ImmediateOperand
immed8: number
rotateImmed: number
base: number
ShifterOperand
operandToShift: Operand
shiftType: string
shiftAmountOperand: Operand
LoadStoreOperand
register: RegisterOperand
offset: Operand
negativeRegOffset: boolean
increment: boolean
preIndexed: boolean
LoadStoreMultipleOperand
registers: RegisterOperand [ ]
BranchOperand
label: string

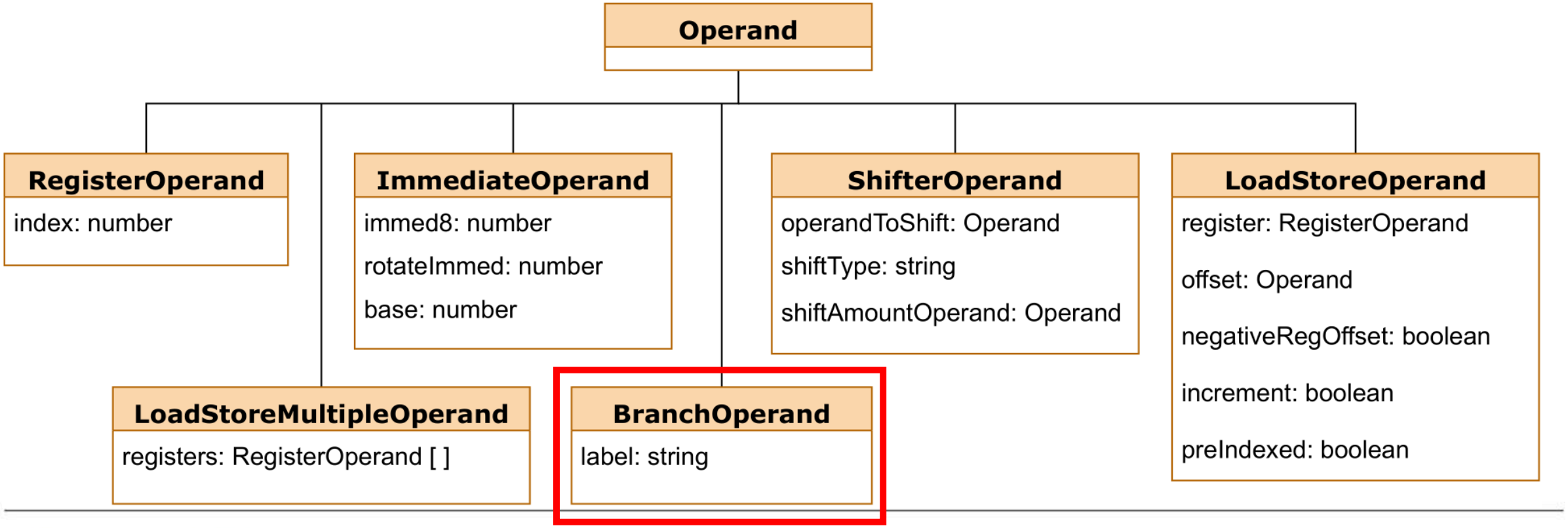
Operanden


Operand
RegisterOperand
index: number
ImmediateOperand
immed8: number
rotateImmed: number
base: number
ShifterOperand
operandToShift: Operand
shiftType: string
shiftAmountOperand: Operand
LoadStoreOperand
register: RegisterOperand
offset: Operand
negativeRegOffset: boolean
increment: boolean
preIndexed: boolean
LoadStoreMultipleOperand
registers: RegisterOperand [ ]
BranchOperand
label: string

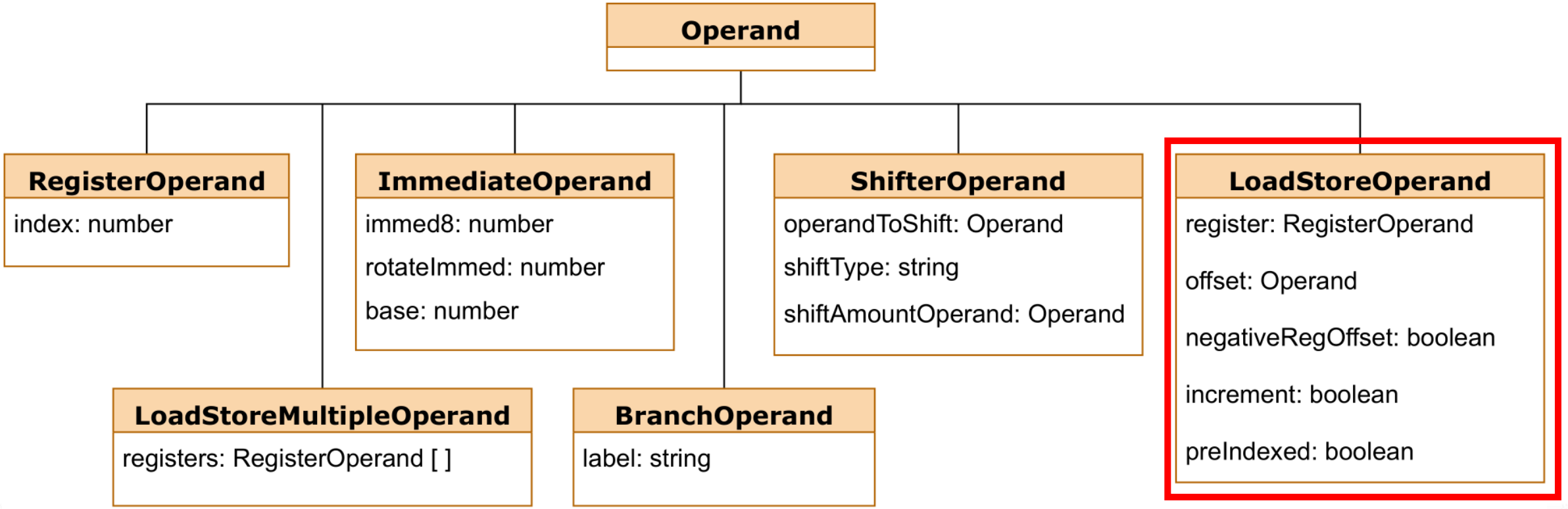
Operanden


Operand
RegisterOperand
index: number
ImmediateOperand
immed8: number
rotateImmed: number
base: number
ShifterOperand
operandToShift: Operand
shiftType: string
shiftAmountOperand: Operand
LoadStoreOperand
register: RegisterOperand
offset: Operand
negativeRegOffset: boolean
increment: boolean
preIndexed: boolean
LoadStoreMultipleOperand
registers: RegisterOperand [ ]
BranchOperand
label: string

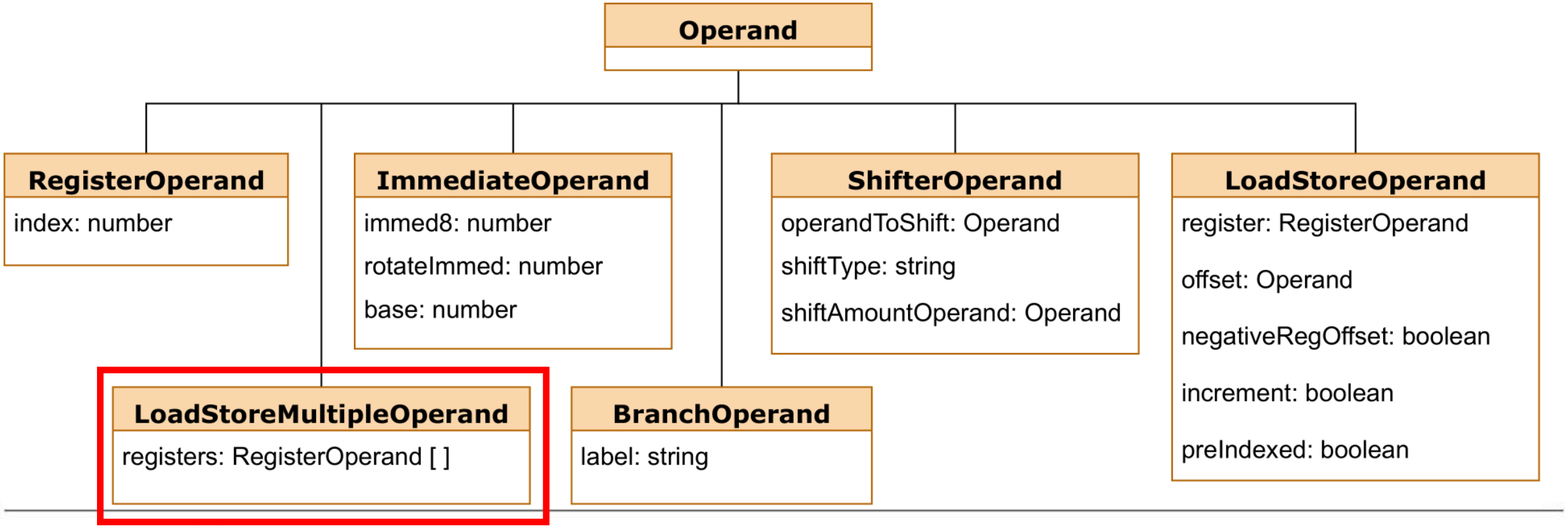
Operanden


Operand
RegisterOperand
index: number
ImmediateOperand
immed8: number
rotateImmed: number
base: number
ShifterOperand
operandToShift: Operand
shiftType: string
shiftAmountOperand: Operand
LoadStoreOperand
register: RegisterOperand
offset: Operand
negativeRegOffset: boolean
increment: boolean
preIndexed: boolean
LoadStoreMultipleOperand
registers: RegisterOperand [ ]
BranchOperand
label: string

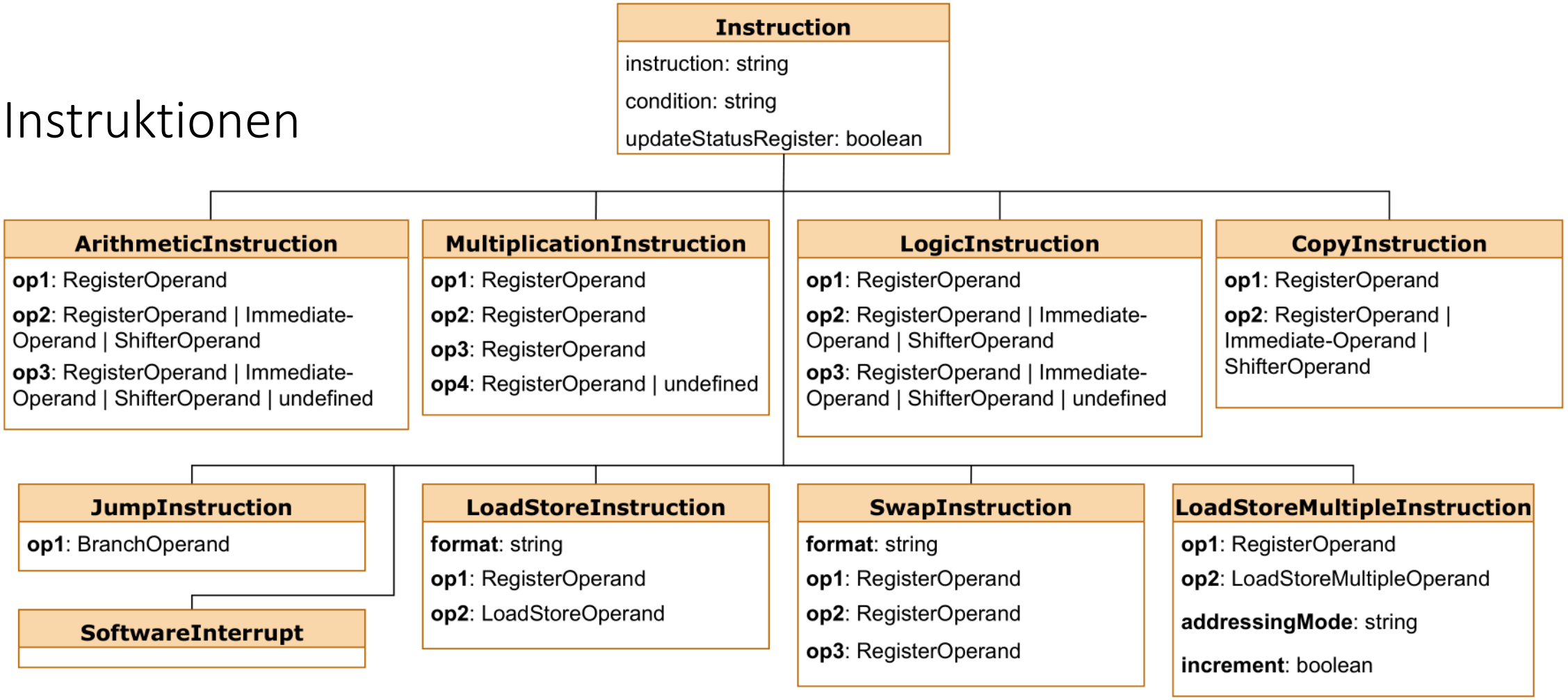
Operanden


Operand
RegisterOperand
index: number
ImmediateOperand
immed8: number
rotateImmed: number
base: number
ShifterOperand
operandToShift: Operand
shiftType: string
shiftAmountOperand: Operand
LoadStoreOperand
register: RegisterOperand
offset: Operand
negativeRegOffset: boolean
increment: boolean
preIndexed: boolean
LoadStoreMultipleOperand
registers: RegisterOperand [ ]
BranchOperand
label: string

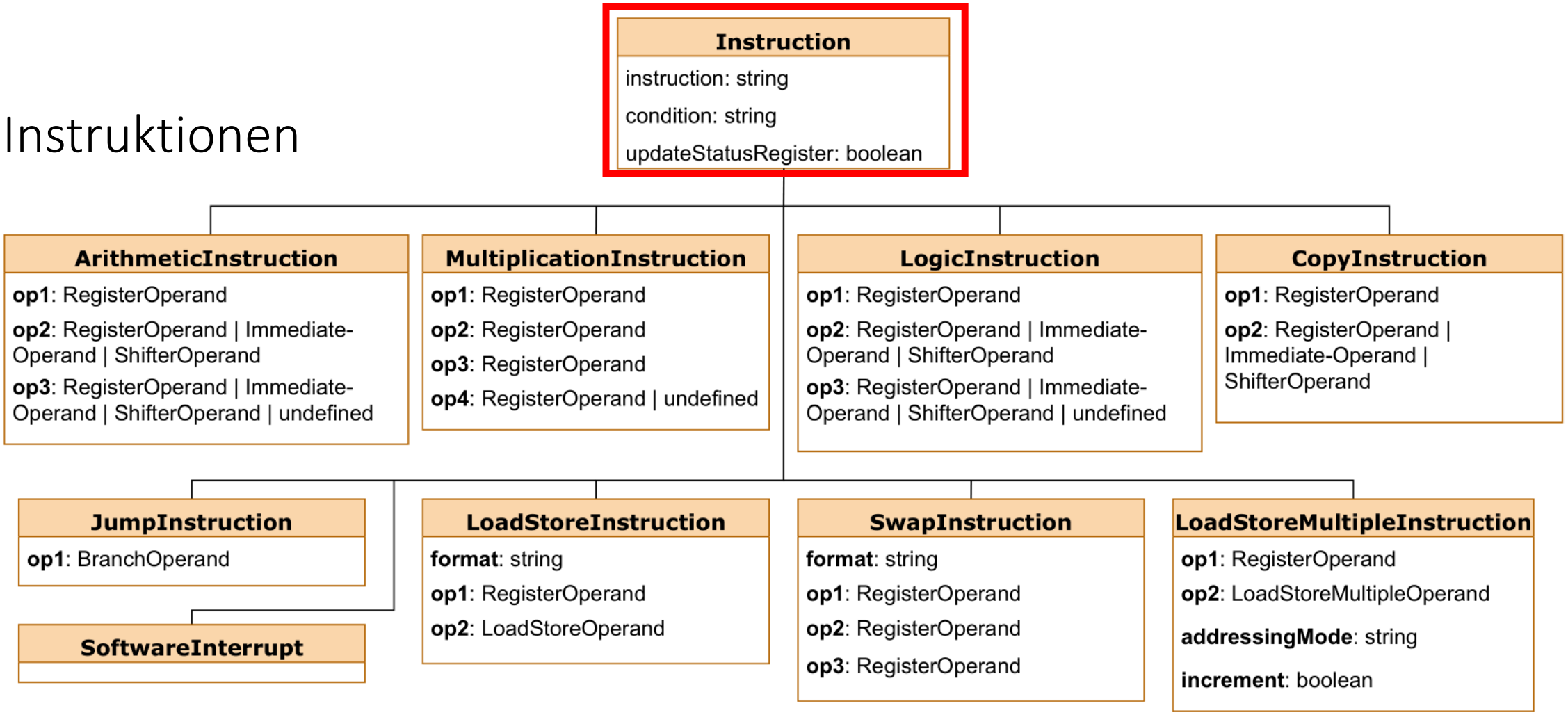
Operanden


Operand
RegisterOperand
index: number
ImmediateOperand
immed8: number
rotateImmed: number
base: number
ShifterOperand
operandToShift: Operand
shiftType: string
shiftAmountOperand: Operand
LoadStoreOperand
register: RegisterOperand
offset: Operand
negativeRegOffset: boolean
increment: boolean
preIndexed: boolean
LoadStoreMultipleOperand
registers: RegisterOperand [ ]
BranchOperand
label: string

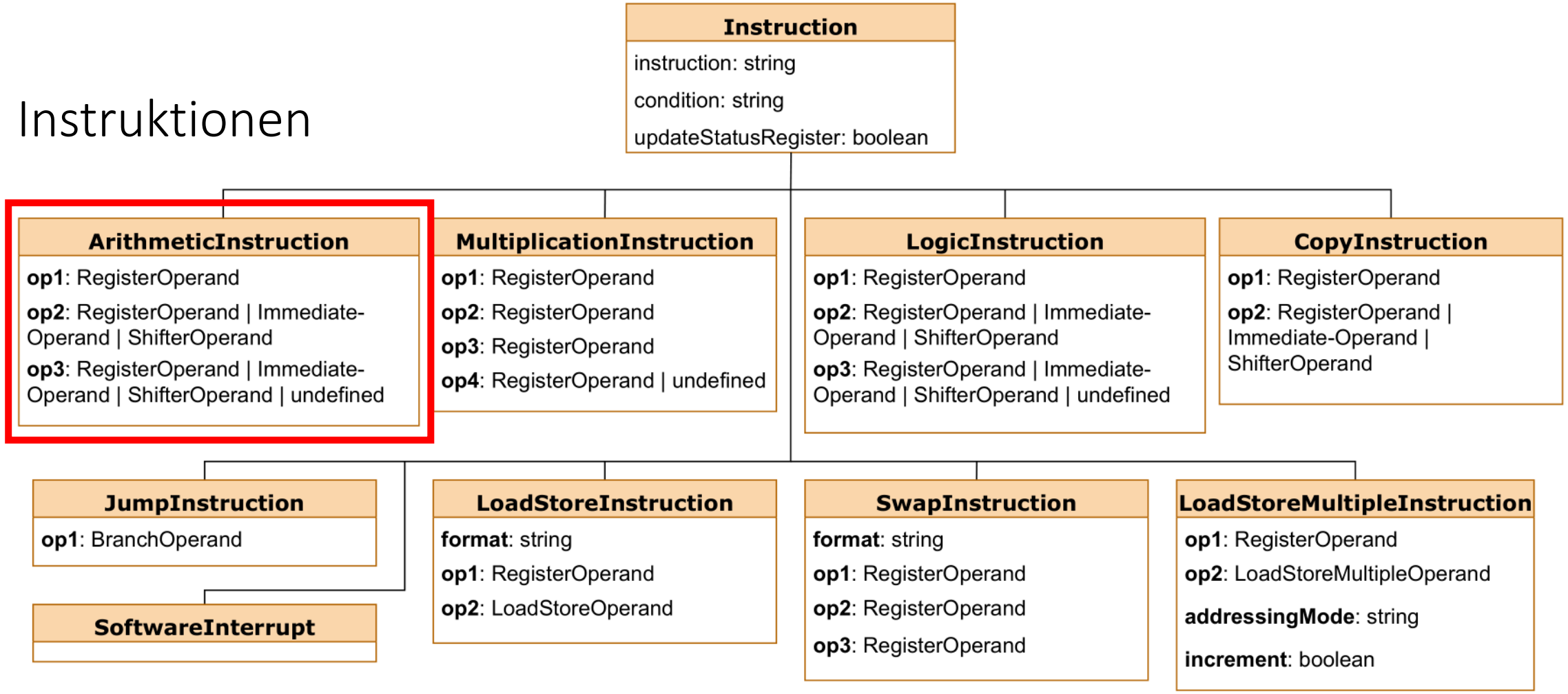
Instruktionen



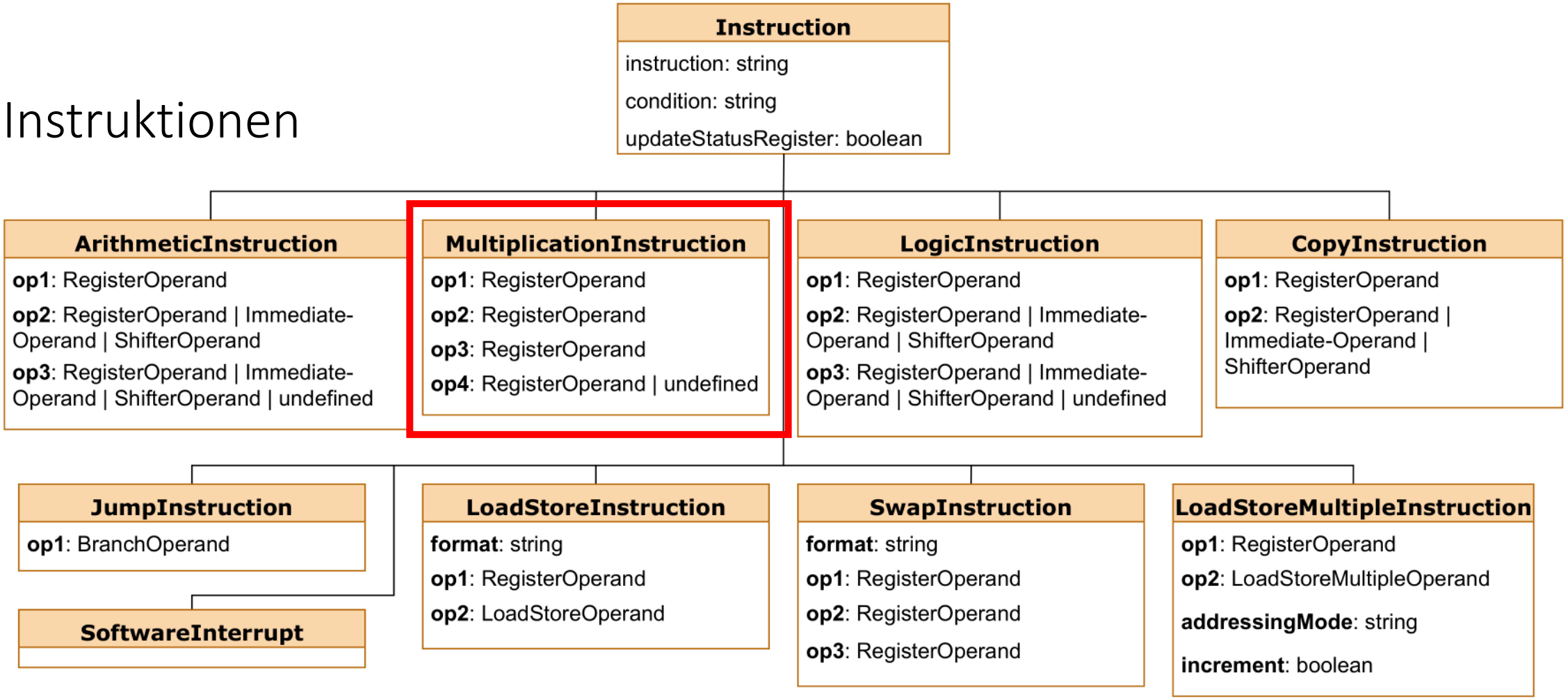
Instruktionen



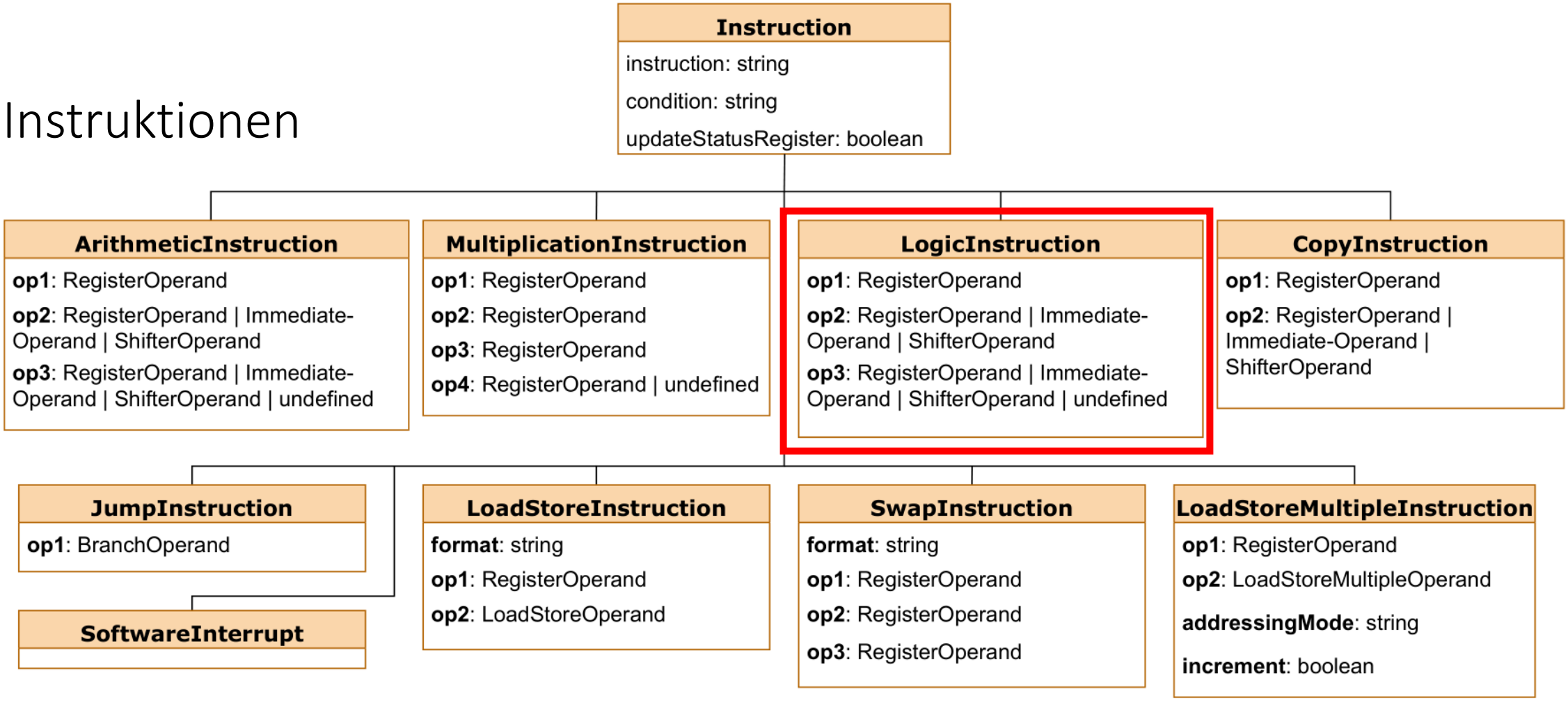
Instruktionen



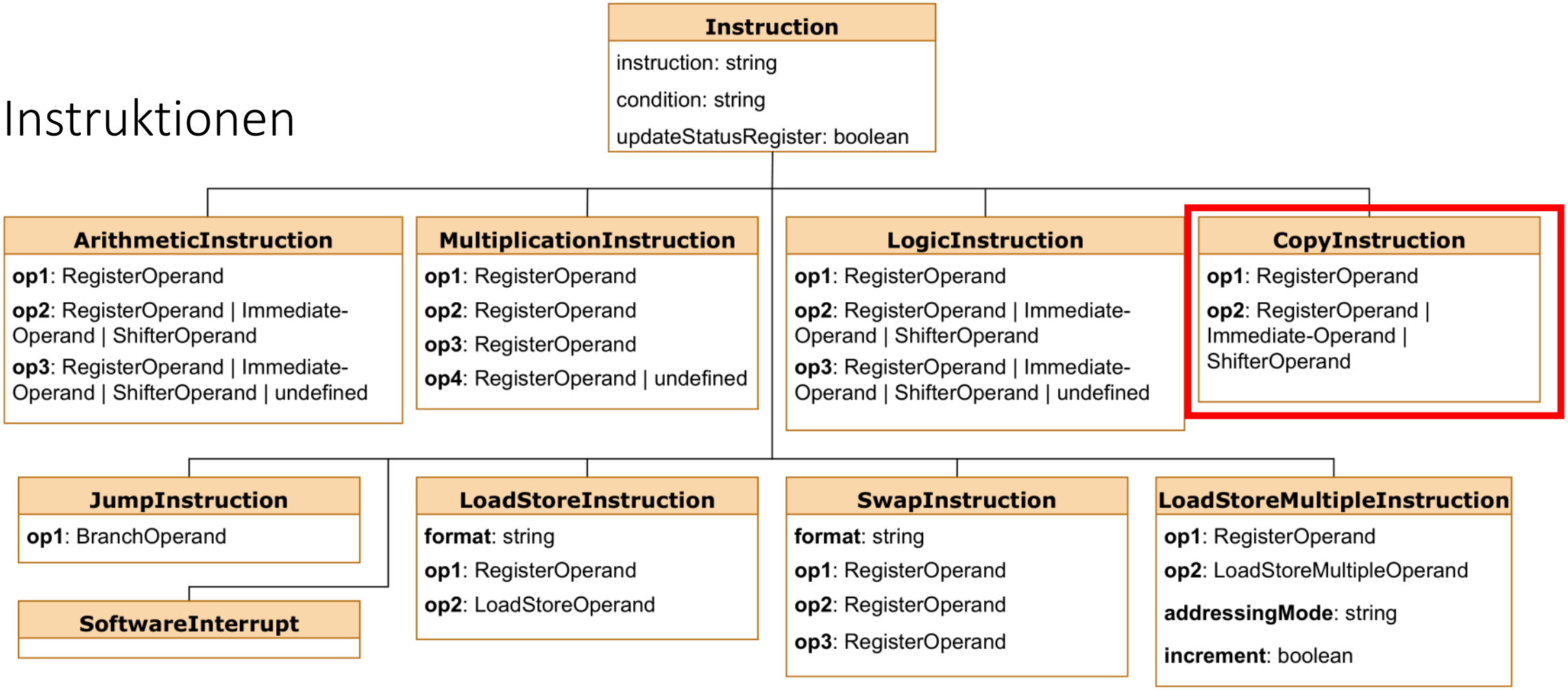
Instruktionen



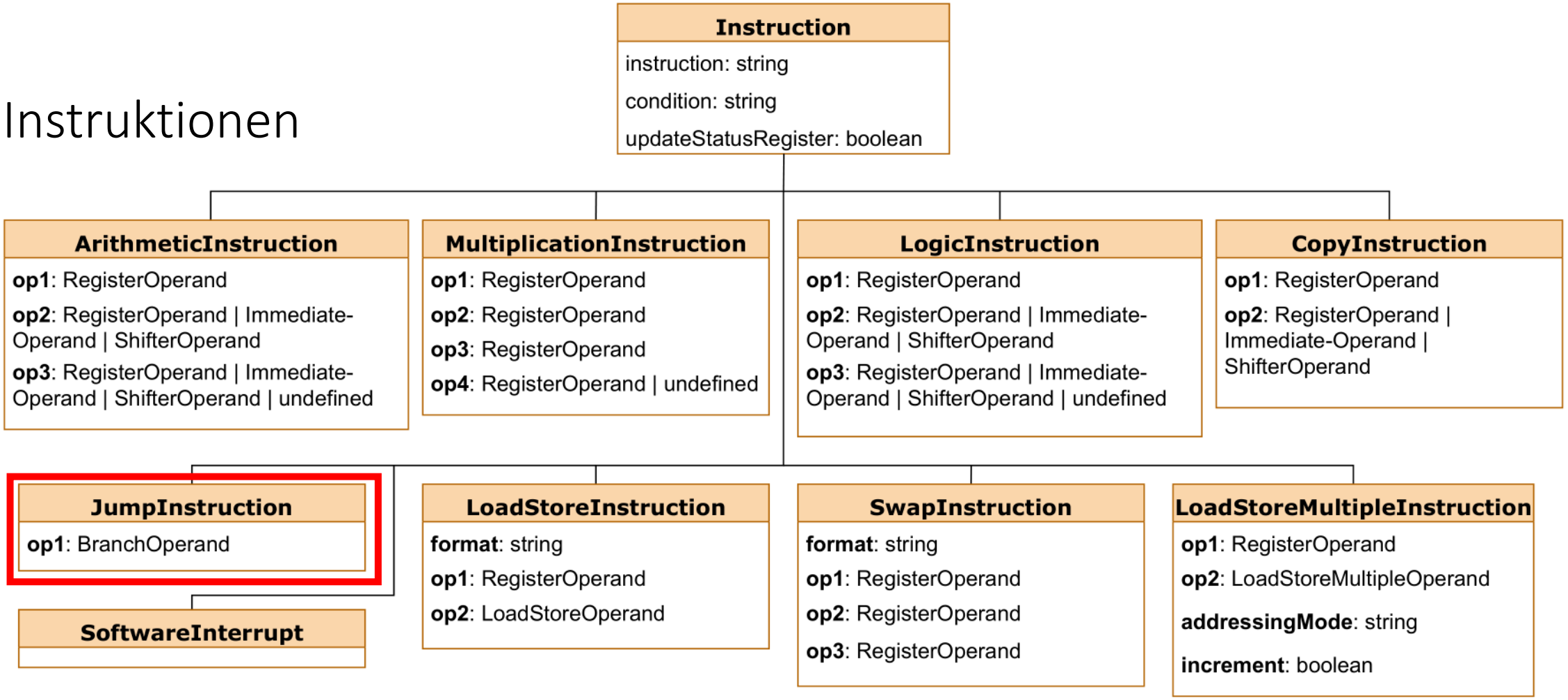
Instruktionen



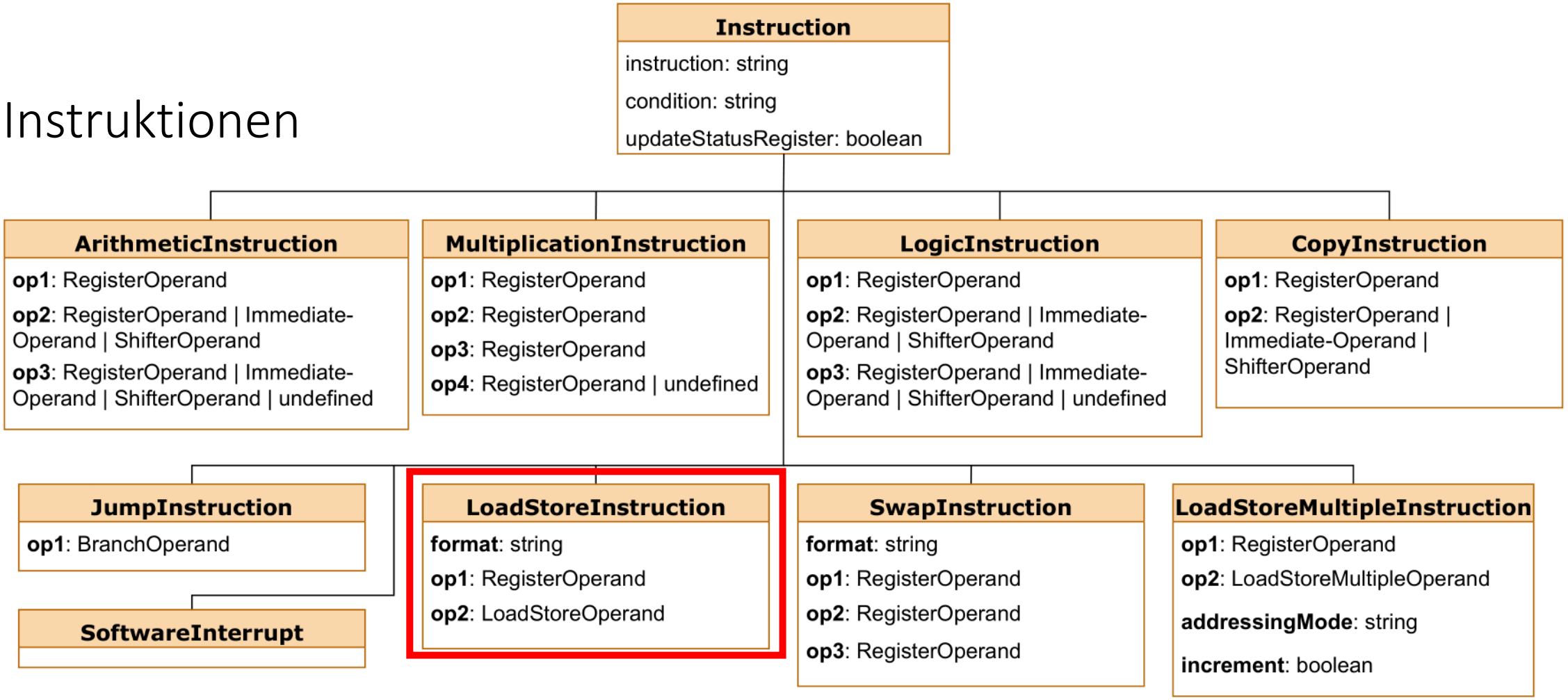
Instruktionen



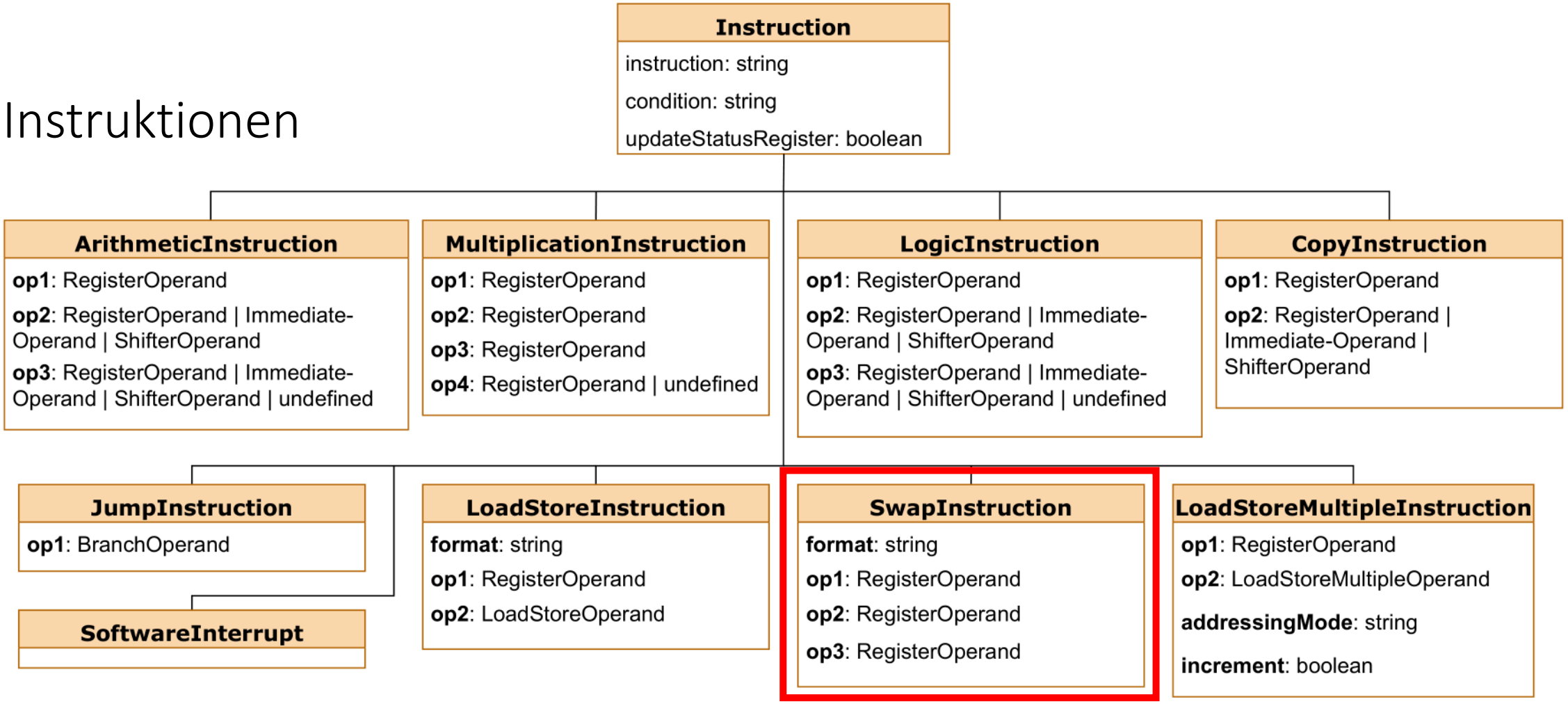
Instruktionen



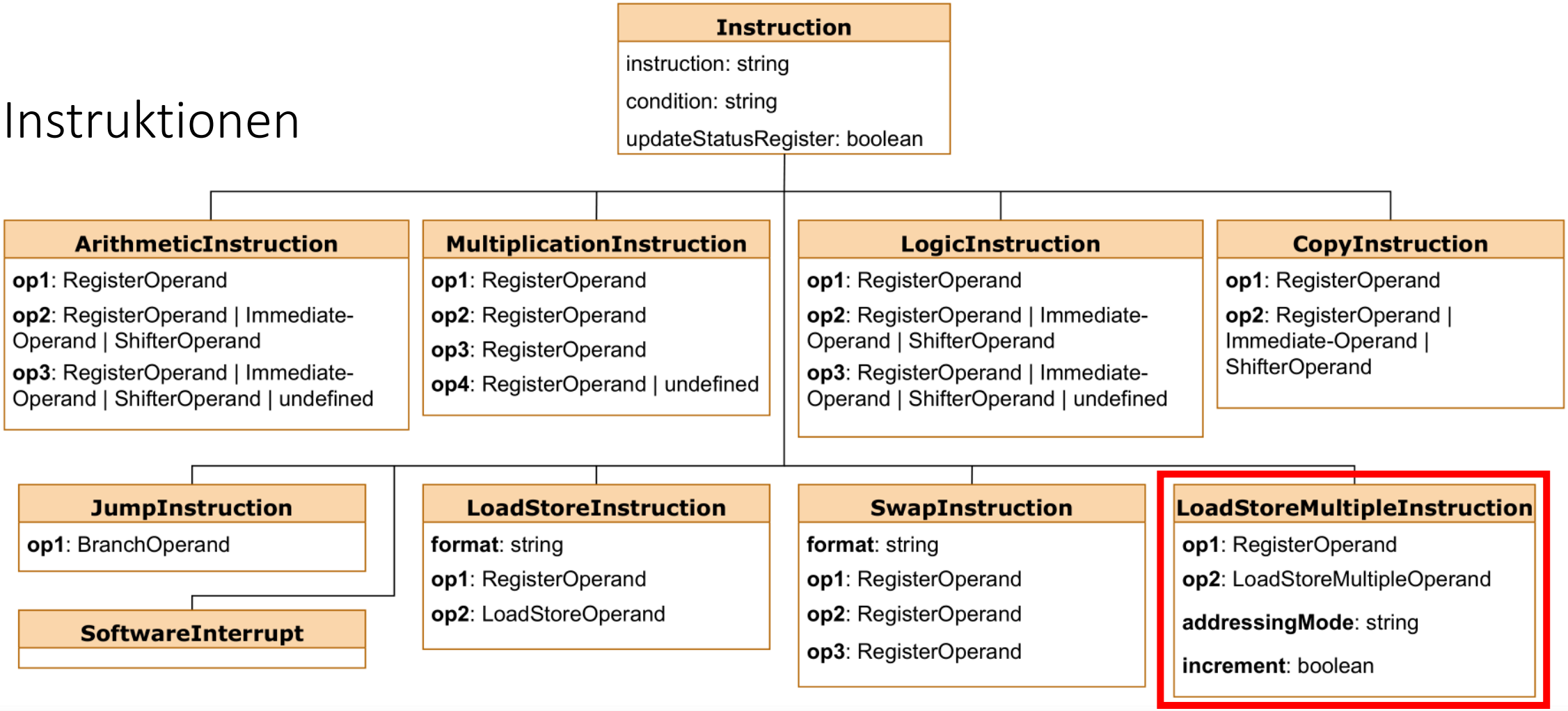
Instruktionen



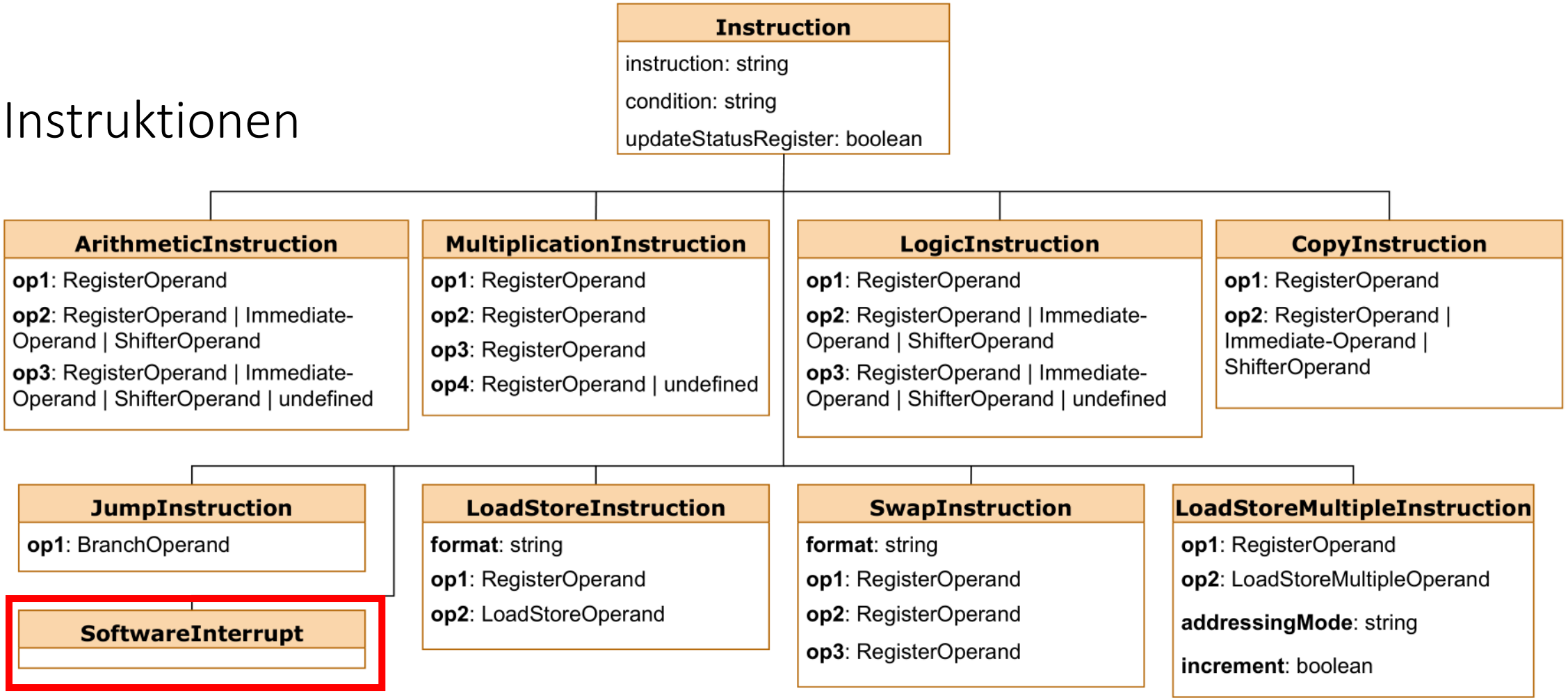
Instruktionen



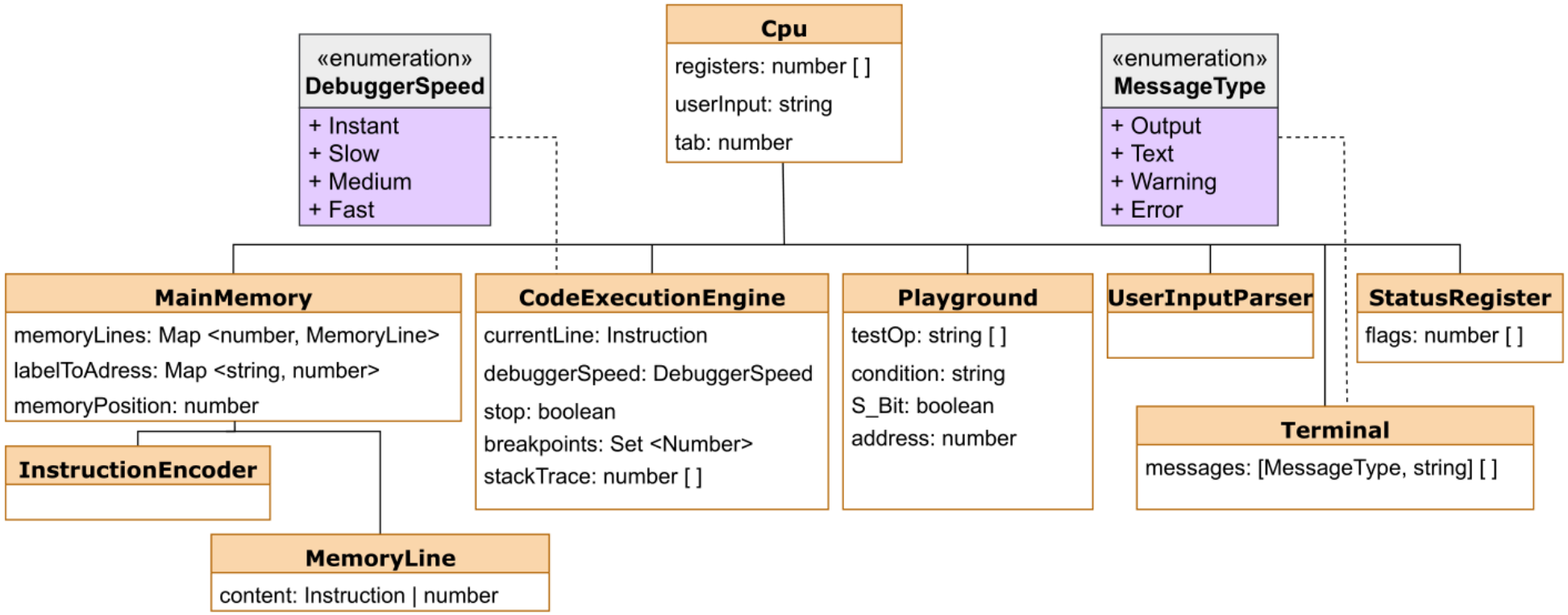
Instruktionen



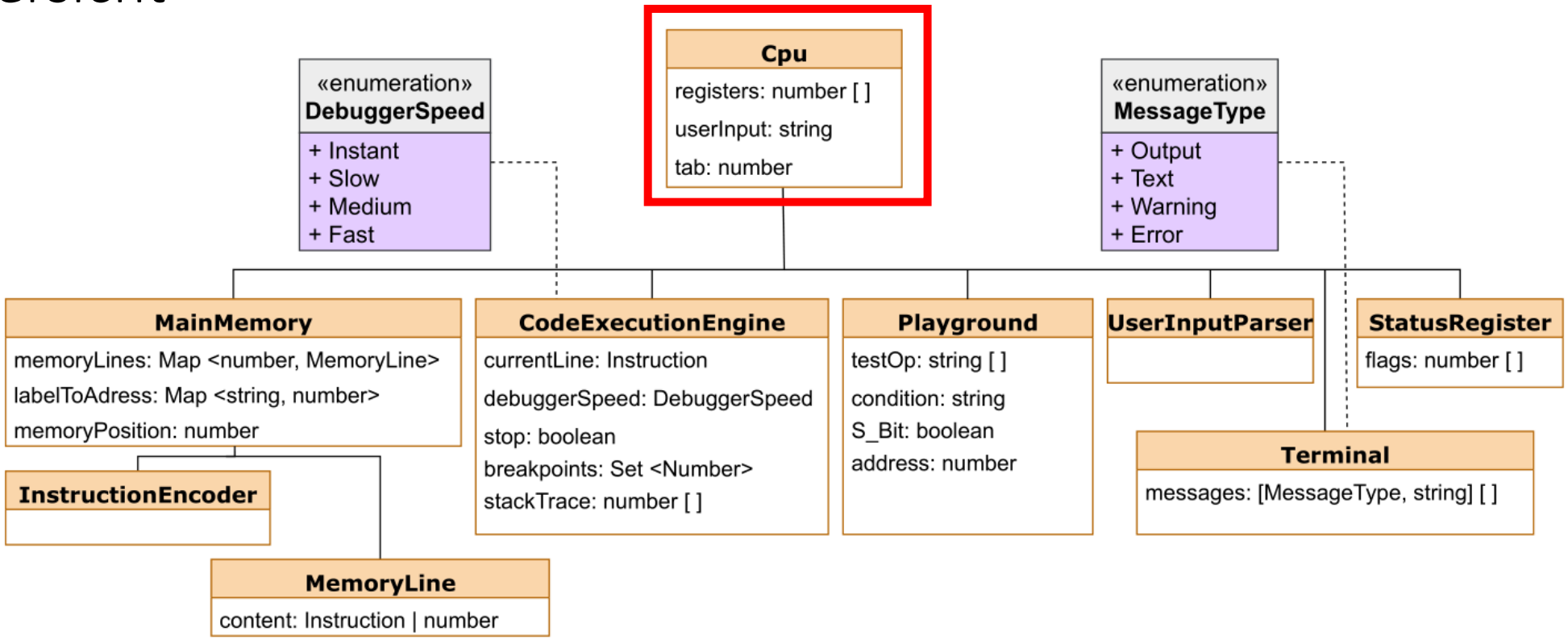
Instruktionen



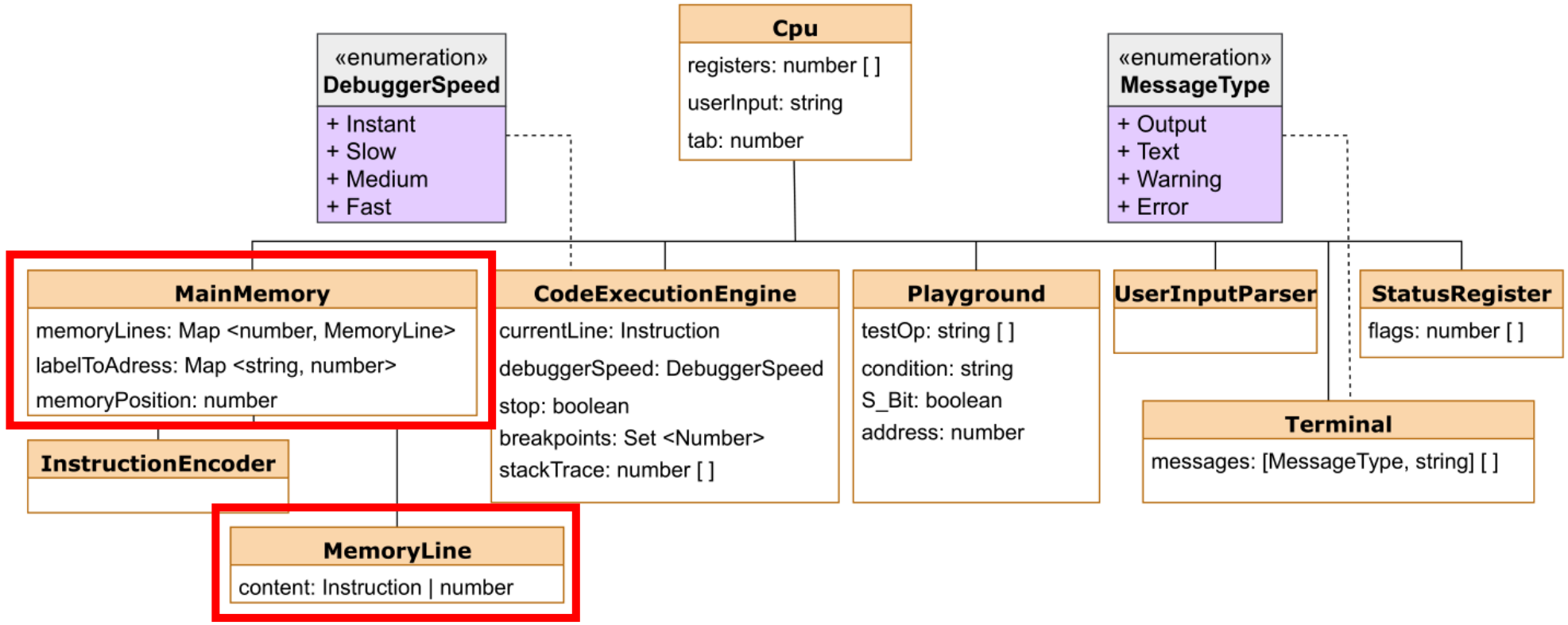
Übersicht


«enumeration»
DebuggerSpeed
+ Instant
+ Slow
+ Medium
+ Fast
Cpu
registers: number [ ]
userInput: string
tab: number
«enumeration»
MessageType
+ Output
+ Text
+ Warning
+ Error
MainMemory
memoryLines: Map <number, MemoryLine>
labelToAdress: Map <string, number>
memoryPosition: number
CodeExecutionEngine
currentLine: Instruction
debuggerSpeed: DebuggerSpeed
stop: boolean
breakpoints: Set <Number>
stackTrace: number [ ]
Playground
testOp: string [ ]
condition: string
S_Bit: boolean
address: number
UserInputParser
StatusRegister
flags: number [ ]
InstructionEncoder
Terminal
messages: [MessageType, string] [ ]
MemoryLine
content: Instruction | number

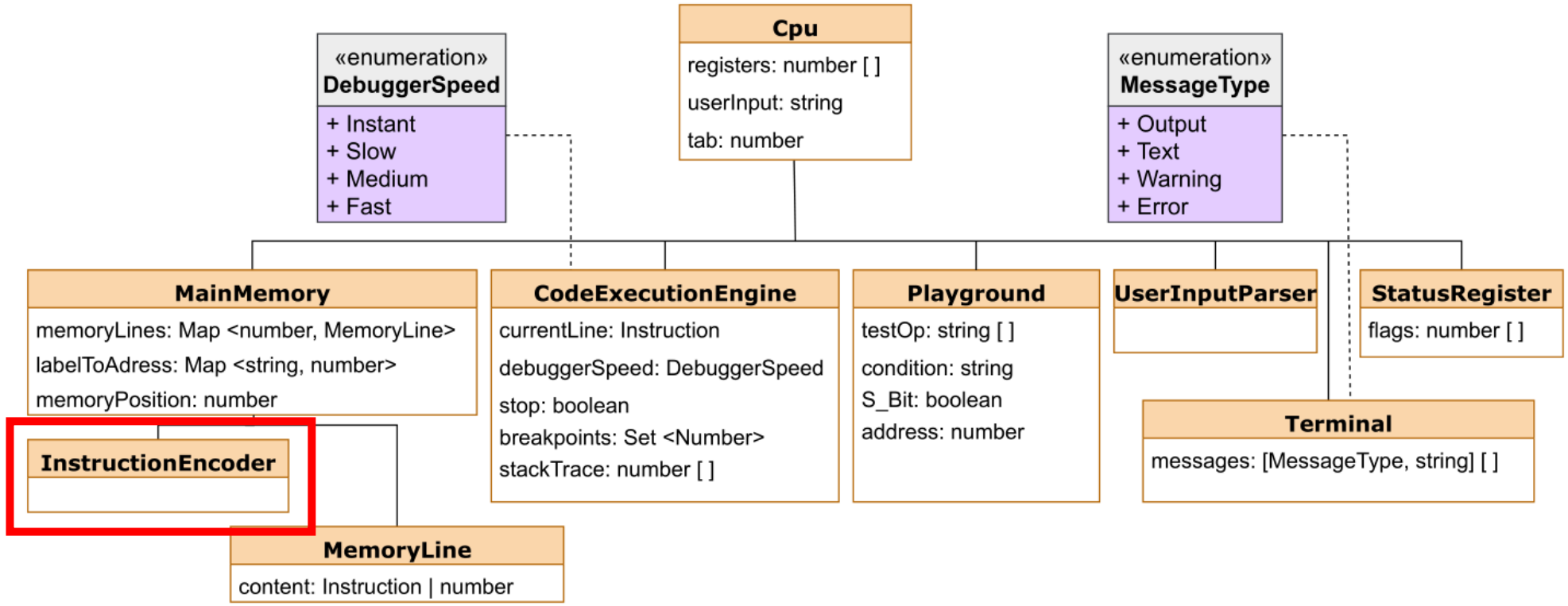
Übersicht


«enumeration»
DebuggerSpeed
+ Instant
+ Slow
+ Medium
+ Fast
Cpu
registers: number [ ]
userInput: string
tab: number
«enumeration»
MessageType
+ Output
+ Text
+ Warning
+ Error
MainMemory
memoryLines: Map <number, MemoryLine>
labelToAdress: Map <string, number>
memoryPosition: number
CodeExecutionEngine
currentLine: Instruction
debuggerSpeed: DebuggerSpeed
stop: boolean
breakpoints: Set <Number>
stackTrace: number [ ]
Playground
testOp: string [ ]
condition: string
S_Bit: boolean
address: number
UserInputParser
StatusRegister
flags: number [ ]
InstructionEncoder
Terminal
messages: [MessageType, string] [ ]
MemoryLine
content: Instruction | number

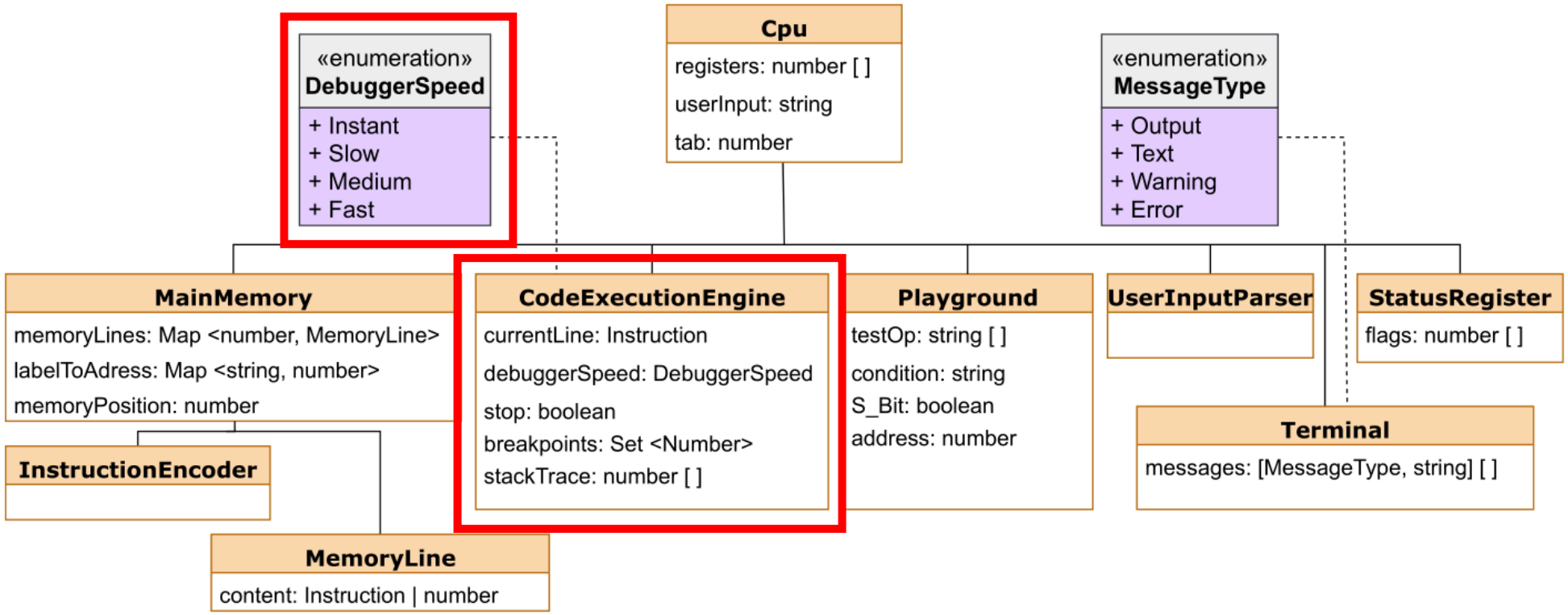
Übersicht


«enumeration»
DebuggerSpeed
+ Instant
+ Slow
+ Medium
+ Fast
Cpu
registers: number [ ]
userInput: string
tab: number
«enumeration»
MessageType
+ Output
+ Text
+ Warning
+ Error
MainMemory
memoryLines: Map <number, MemoryLine>
labelToAdress: Map <string, number>
memoryPosition: number
CodeExecutionEngine
currentLine: Instruction
debuggerSpeed: DebuggerSpeed
stop: boolean
breakpoints: Set <Number>
stackTrace: number [ ]
Playground
testOp: string [ ]
condition: string
S_Bit: boolean
address: number
UserInputParser
StatusRegister
flags: number [ ]
Terminal
messages: [MessageType, string] [ ]
InstructionEncoder
MemoryLine
content: Instruction | number

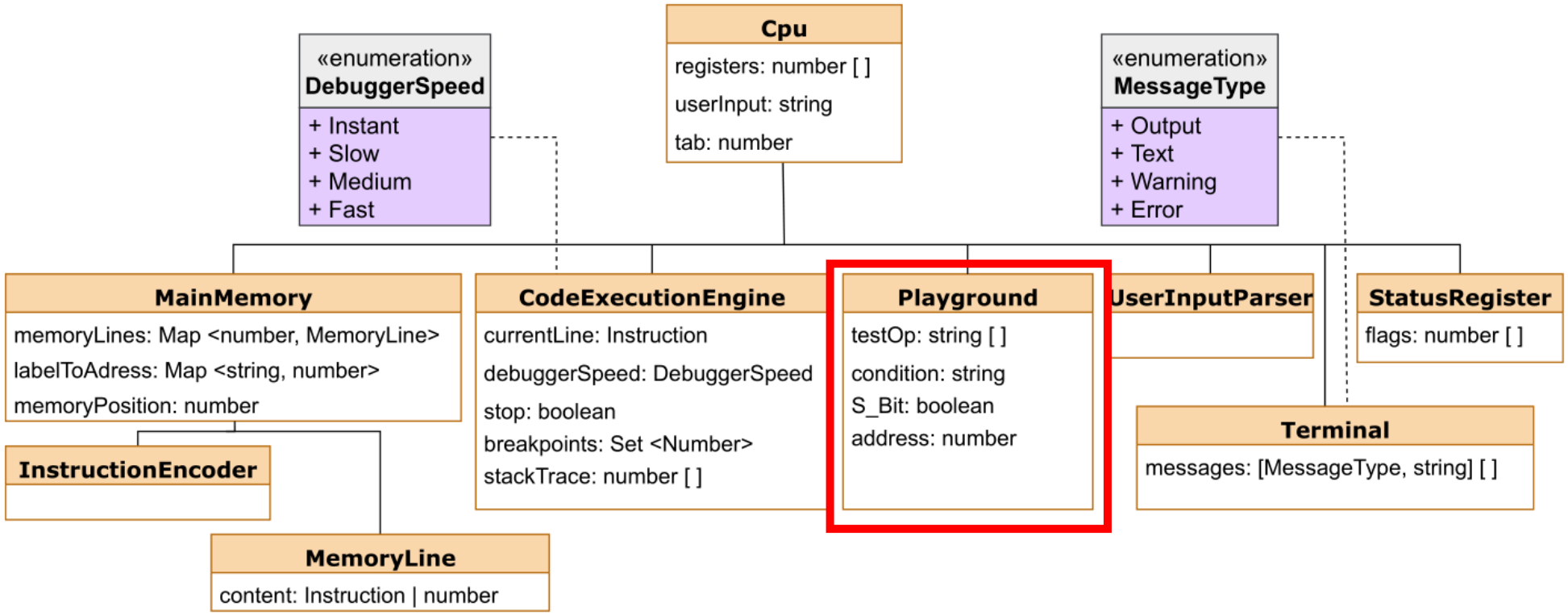
Übersicht


«enumeration»
DebuggerSpeed
+ Instant
+ Slow
+ Medium
+ Fast
Cpu
registers: number [ ]
userInput: string
tab: number
«enumeration»
MessageType
+ Output
+ Text
+ Warning
+ Error
MainMemory
memoryLines: Map <number, MemoryLine>
labelToAdress: Map <string, number>
memoryPosition: number
CodeExecutionEngine
currentLine: Instruction
debuggerSpeed: DebuggerSpeed
stop: boolean
breakpoints: Set <Number>
stackTrace: number [ ]
Playground
testOp: string [ ]
condition: string
S_Bit: boolean
address: number
UserInputParser
StatusRegister
flags: number [ ]
InstructionEncoder
Terminal
messages: [MessageType, string] [ ]
MemoryLine
content: Instruction | number

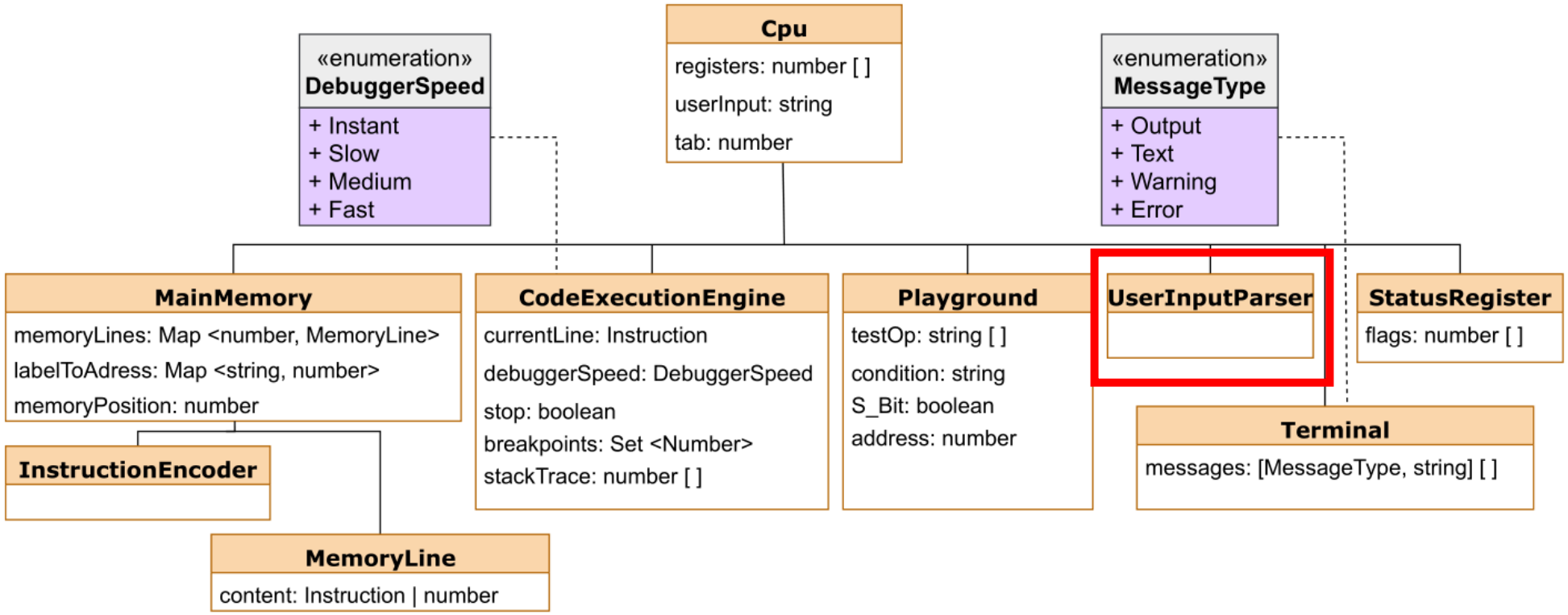
Übersicht


«enumeration»
DebuggerSpeed
+ Instant
+ Slow
+ Medium
+ Fast
Cpu
registers: number [ ]
userInput: string
tab: number
«enumeration»
MessageType
+ Output
+ Text
+ Warning
+ Error
MainMemory
memoryLines: Map <number, MemoryLine>
labelToAdress: Map <string, number>
memoryPosition: number
CodeExecutionEngine
currentLine: Instruction
debuggerSpeed: DebuggerSpeed
stop: boolean
breakpoints: Set <Number>
stackTrace: number [ ]
Playground
testOp: string [ ]
condition: string
S_Bit: boolean
address: number
UserInputParser
StatusRegister
flags: number [ ]
InstructionEncoder
Terminal
messages: [MessageType, string] [ ]
MemoryLine
content: Instruction | number

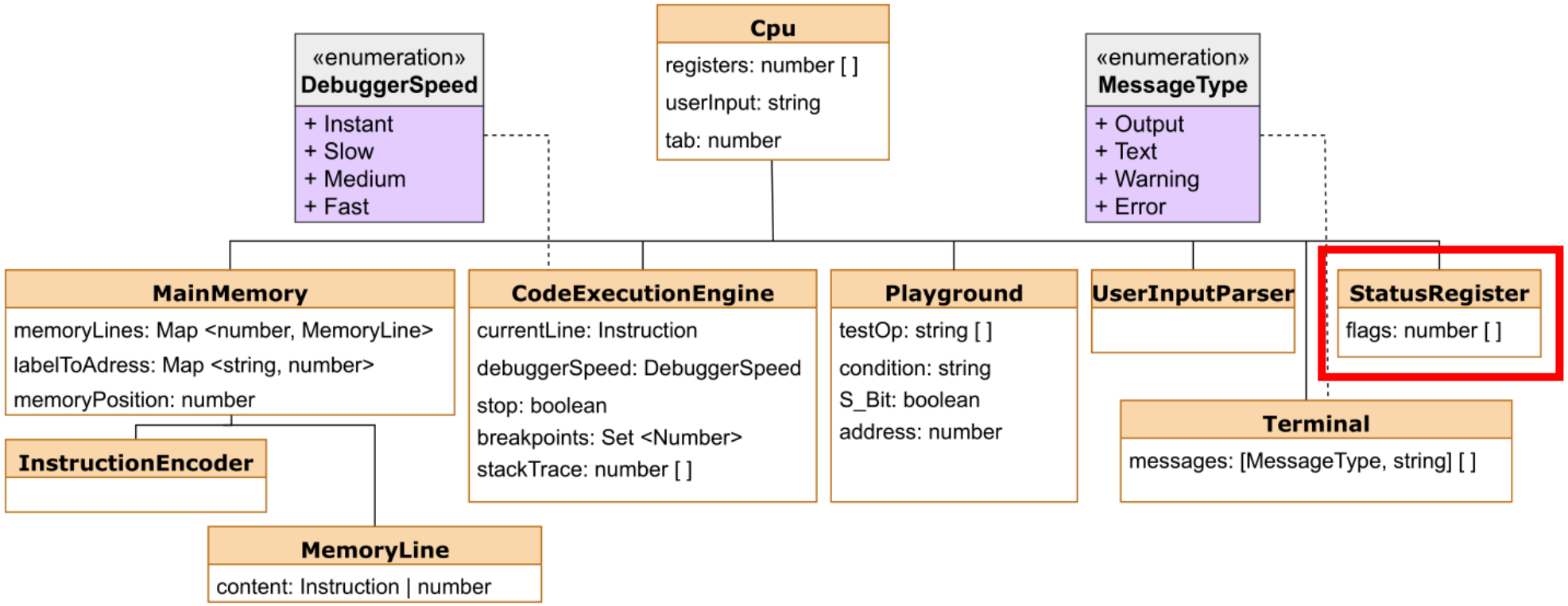
Übersicht


«enumeration»
DebuggerSpeed
+ Instant
+ Slow
+ Medium
+ Fast
Cpu
registers: number [ ]
userInput: string
tab: number
«enumeration»
MessageType
+ Output
+ Text
+ Warning
+ Error
MainMemory
memoryLines: Map <number, MemoryLine>
labelToAdress: Map <string, number>
memoryPosition: number
CodeExecutionEngine
currentLine: Instruction
debuggerSpeed: DebuggerSpeed
stop: boolean
breakpoints: Set <Number>
stackTrace: number [ ]
Playground
testOp: string [ ]
condition: string
S_Bit: boolean
address: number
UserInputParser
StatusRegister
flags: number [ ]
InstructionEncoder
MemoryLine
content: Instruction | number
Terminal
messages: [MessageType, string] [ ]

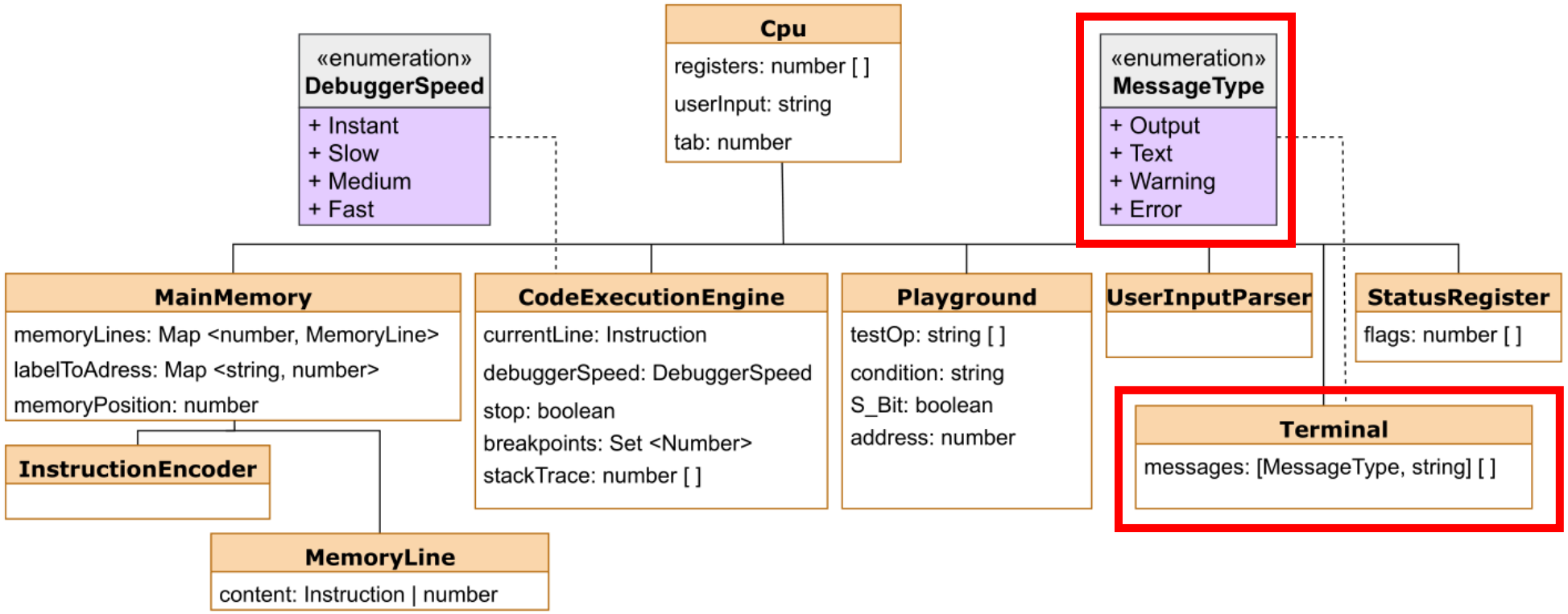
Übersicht


«enumeration»
DebuggerSpeed
+ Instant
+ Slow
+ Medium
+ Fast
Cpu
registers: number [ ]
userInput: string
tab: number
«enumeration»
MessageType
+ Output
+ Text
+ Warning
+ Error
MainMemory
memoryLines: Map <number, MemoryLine>
labelToAdress: Map <string, number>
memoryPosition: number
CodeExecutionEngine
currentLine: Instruction
debuggerSpeed: DebuggerSpeed
stop: boolean
breakpoints: Set <Number>
stackTrace: number [ ]
Playground
testOp: string [ ]
condition: string
S_Bit: boolean
address: number
UserInputParser
StatusRegister
flags: number [ ]
Terminal
messages: [MessageType, string] [ ]
InstructionEncoder
MemoryLine
content: Instruction | number

Übersicht


«enumeration»
DebuggerSpeed
+ Instant
+ Slow
+ Medium
+ Fast
Cpu
registers: number [ ]
userInput: string
tab: number
«enumeration»
MessageType
+ Output
+ Text
+ Warning
+ Error
MainMemory
memoryLines: Map <number, MemoryLine>
labelToAdress: Map <string, number>
memoryPosition: number
CodeExecutionEngine
currentLine: Instruction
debuggerSpeed: DebuggerSpeed
stop: boolean
breakpoints: Set <Number>
stackTrace: number [ ]
Playground
testOp: string [ ]
condition: string
S_Bit: boolean
address: number
UserInputParser
StatusRegister
flags: number [ ]
InstructionEncoder
Terminal
messages: [MessageType, string] [ ]
MemoryLine
content: Instruction | number

Übersicht


«enumeration»
DebuggerSpeed
+ Instant
+ Slow
+ Medium
+ Fast
Cpu
registers: number [ ]
userInput: string
tab: number
«enumeration»
MessageType
+ Output
+ Text
+ Warning
+ Error
MainMemory
memoryLines: Map <number, MemoryLine>
labelToAdress: Map <string, number>
memoryPosition: number
CodeExecutionEngine
currentLine: Instruction
debuggerSpeed: DebuggerSpeed
stop: boolean
breakpoints: Set <Number>
stackTrace: number [ ]
Playground
testOp: string [ ]
condition: string
S_Bit: boolean
address: number
UserInputParser
StatusRegister
flags: number [ ]
InstructionEncoder
MemoryLine
content: Instruction | number
Terminal
messages: [MessageType, string] [ ]

Parser

```
start := start=line

line := label? directive comment? nextLine |
        label? instruction comment? nextLine |
        label? commentLine? nextLine |
        $
```

- Einteilung der Grammatik in Zeilen:

Parser

```
start := start=line

line := label? directive comment? nextLine |
        label? instruction comment? nextLine |
        label? commentLine? nextLine |
        $
```

- Einteilung der Grammatik in Zeilen:
  - Direktiven

Parser

```
start := start=line

line := label? directive comment? nextLine |
        label? instruction comment? nextLine |
        label? commentLine? nextLine |
        $
```

- Einteilung der Grammatik in Zeilen:
  - Direktiven
  - Instruktionen

Parser

```
start := start=line

line := label? directive comment? nextLine |
        label? instruction comment? nextLine |
        label? commentLine? nextLine |
        $
```

- Einteilung der Grammatik in Zeilen:
  - Direktiven
  - Instruktionen
  - Kommentare

Parser

```
start := start=line

line := label? directive comment? nextLine |
        label? instruction comment? nextLine |
        label? commentLine? nextLine |
        $
```

- Einteilung der Grammatik in Zeilen:
  - Direktiven
  - Instruktionen
  - Kommentare
- Optionales Label und Kommentar

Parser

```
start := start=line

line := label? directive comment? nextLine |
        label? instruction comment? nextLine |
        label? commentLine? nextLine |
        $
```

- Einteilung der Grammatik in Zeilen:
  - Direktiven
  - Instruktionen
  - Kommentare
- Optionales Label und Kommentar
- Jede Zeile besitzt weitere Zeile, außer das Ende des Codes **$** ist erreicht

Parser

- Weitere Einteilung je nach Art der Zeile

```
instruction := art | log | copyJump | loadStore | loadStoreMultiple |
               softwareInterrupt

art := inst=artInst cond=condition operands=artOp |
       inst='mul' cond=condition operands=artMulOp
       inst='mla' cond=condition operands=artMlaOp

artInst := 'add' | 'adc' | 'sub' | 'sbc' | 'rsb' | 'rsc'

artOp := artOp3 | artOp2
artOp2 := op1=regOp ',' op2=op
artOp3 := op1=regOp ',' op2=regOp ',' op3=op

artMulOp := op1=regOp ',' op2=regOp ',' op3=regOp
artMlaOp := op1=regOp ',' op2=regOp ',' op3=regOp ',' op4=regOp
```

Parser

- Weitere Einteilung je nach Art der Zeile
- Instruktionen aufgeteilt in Typen

```
instruction := art | log | copyJump | loadStore | loadStoreMultiple |
               softwareInterrupt

art := inst=artInst cond=condition operands=artOp |
       inst='mul' cond=condition operands=artMulOp
       inst='mla' cond=condition operands=artMlaOp

artInst := 'add' | 'adc' | 'sub' | 'sbc' | 'rsb' | 'rsc'

artOp := artOp3 | artOp2
artOp2 := op1=regOp ',' op2=op
artOp3 := op1=regOp ',' op2=regOp ',' op3=op

artMulOp := op1=regOp ',' op2=regOp ',' op3=regOp
artMlaOp := op1=regOp ',' op2=regOp ',' op3=regOp ',' op4=regOp
```

Parser

- Weitere Einteilung je nach Art der Zeile
- Instruktionen aufgeteilt in Typen
- Instruktion:

```
instruction := art | log | copyJump | loadStore | loadStoreMultiple |
               softwareInterrupt

art := inst=artInst cond=condition operands=artOp |
       inst='mul' cond=condition operands=artMulOp
       inst='mla' cond=condition operands=artMlaOp

artInst := 'add' | 'adc' | 'sub' | 'sbc' | 'rsb' | 'rsc'

artOp := artOp3 | artOp2
artOp2 := op1=regOp ',' op2=op
artOp3 := op1=regOp ',' op2=regOp ',' op3=op

artMulOp := op1=regOp ',' op2=regOp ',' op3=regOp
artMlaOp := op1=regOp ',' op2=regOp ',' op3=regOp ',' op4=regOp
```

Parser

- Weitere Einteilung je nach Art der Zeile
- Instruktionen aufgeteilt in Typen
- Instruktion:
  - Name der Instruktion

```
instruction := art | log | copyJump | loadStore | loadStoreMultiple |
               softwareInterrupt

art := inst=artInst cond=condition operands=artOp |
       inst='mul' cond=condition operands=artMulOp
       inst='mla' cond=condition operands=artMlaOp

artInst := 'add' | 'adc' | 'sub' | 'sbc' | 'rsb' | 'rsc'

artOp := artOp3 | artOp2
artOp2 := op1=regOp ',' op2=op
artOp3 := op1=regOp ',' op2=regOp ',' op3=op

artMulOp := op1=regOp ',' op2=regOp ',' op3=regOp
artMlaOp := op1=regOp ',' op2=regOp ',' op3=regOp ',' op4=regOp
```

Parser

- Weitere Einteilung je nach Art der Zeile
- Instruktionen aufgeteilt in Typen
- Instruktion:
  - Name der Instruktion
  - Bedingung

```
instruction := art | log | copyJump | loadStore | loadStoreMultiple |
               softwareInterrupt

art := inst=artInst cond=condition operands=artOp |
       inst='mul' cond=condition operands=artMulOp
       inst='mla' cond=condition operands=artMlaOp

artInst := 'add' | 'adc' | 'sub' | 'sbc' | 'rsb' | 'rsc'

artOp := artOp3 | artOp2
artOp2 := op1=regOp ',' op2=op
artOp3 := op1=regOp ',' op2=regOp ',' op3=op

artMulOp := op1=regOp ',' op2=regOp ',' op3=regOp
artMlaOp := op1=regOp ',' op2=regOp ',' op3=regOp ',' op4=regOp
```

Parser

- Weitere Einteilung je nach Art der Zeile
- Instruktionen aufgeteilt in Typen
- Instruktion:
  - Name der Instruktion
  - Bedingung
  - Operanden

```
instruction := art | log | copyJump | loadStore | loadStoreMultiple |
               softwareInterrupt

art := inst=artInst cond=condition operands=artOp |
       inst='mul' cond=condition operands=artMulOp
       inst='mla' cond=condition operands=artMlaOp

artInst := 'add' | 'adc' | 'sub' | 'sbc' | 'rsb' | 'rsc'

artOp := artOp3 | artOp2
artOp2 := op1=regOp ',' op2=op
artOp3 := op1=regOp ',' op2=regOp ',' op3=op

artMulOp := op1=regOp ',' op2=regOp ',' op3=regOp
artMlaOp := op1=regOp ',' op2=regOp ',' op3=regOp ',' op4=regOp
```

Parser

- Abarbeiten des abstrakten Syntax Baums Zeile für Zeile

```
let line = ast.start;

while (line.kind !== ASTKinds.line_5) {
    let currentLine = line.currentLine;

    switch (currentLine.kind) {
        case ASTKinds.instruction_1: this.parseArithmeticInstruction(currentLine.instruction); break;
        case ASTKinds.instruction_2: this.parseLogicInstruction(currentLine.instruction); break;
        ...
        case ASTKinds.directive_1: this.addASCIIData(currentLine.directive.data); break;
        case ASTKinds.directive_2: this.addData(currentLine.directive.size, "0"); break;
        ...
    }

    line = line.nextLine;
}
```

Parser

- Abarbeiten des abstrakten Syntax Baums Zeile für Zeile

- Zuweisen der aktuellen Zeile gefolgt von While-Loop:

```
let line = ast.start;

while (line.kind !== ASTKinds.line_5) {
    let currentLine = line.currentLine;

    switch (currentLine.kind) {
        case ASTKinds.instruction_1: this.parseArithmeticInstruction(currentLine.instruction); break;
        case ASTKinds.instruction_2: this.parseLogicInstruction(currentLine.instruction); break;
        ...
        case ASTKinds.directive_1: this.addASCIIData(currentLine.directive.data); break;
        case ASTKinds.directive_2: this.addData(currentLine.directive.size, "0"); break;
        ...
    }

    line = line.nextLine;
}
```

Parser

- Abarbeiten des abstrakten Syntax Baums Zeile für Zeile

- Zuweisen der aktuellen Zeile gefolgt von While-Loop:
  - Aufrufen der korrekten Funktion

```
let line = ast.start;

while (line.kind !== ASTKinds.line_5) {
    let currentLine = line.currentLine;

    switch (currentLine.kind) {
        case ASTKinds.instruction_1: this.parseArithmeticInstruction(currentLine.instruction); break;
        case ASTKinds.instruction_2: this.parseLogicInstruction(currentLine.instruction); break;
        ...
        case ASTKinds.directive_1: this.addASCIIData(currentLine.directive.data); break;
        case ASTKinds.directive_2: this.addData(currentLine.directive.size, "0"); break;
        ...
    }

    line = line.nextLine;
}
```

Parser

- Abarbeiten des abstrakten Syntax Baums Zeile für Zeile

- Zuweisen der aktuellen Zeile gefolgt von While-Loop:
  - Aufrufen der korrekten Funktion
  - Zuweisen der nächsten Zeile

```
let line = ast.start;

while (line.kind !== ASTKinds.line_5) {
    let currentLine = line.currentLine;

    switch (currentLine.kind) {
        case ASTKinds.instruction_1: this.parseArithmeticInstruction(currentLine.instruction); break;
        case ASTKinds.instruction_2: this.parseLogicInstruction(currentLine.instruction); break;
        ...
        case ASTKinds.directive_1: this.addASCIIData(currentLine.directive.data); break;
        case ASTKinds.directive_2: this.addData(currentLine.directive.size, "0"); break;
        ...
    }

    line = line.nextLine;
}
```

Hauptspeicher

- Klasse mit Funktionen zum Hinzufügen von Instruktionen, Daten und Labels mit Überprüfung auf Korrektheit

Hauptspeicher

- Klasse mit Funktionen zum Hinzufügen von Instruktionen, Daten und Labels mit Überprüfung auf Korrektheit

- Anzeige des Hauptspeichers:

	Address	Encoding	Instruction
		msg:	
	00000000	6c6c6548	
	00000004	6e49206f	
	00000008	7262736e	
	0000000c	216b6375	
	00000010	0000000a	
		_start:	
	00000014	e3a00001	mov r0, #1
	00000018	e0000000	ldr r1, =msg
●	0000001c	e0000000	ldr r2, =len
	00000020	e3a07004	mov r7, #4
●	00000024	ef000000	swi #0
	00000028	e3a00000	mov r0, #0
	0000002c	e3a07001	mov r7, #1
	00000030	ef000000	swi #0

Hauptspeicher

- Klasse mit Funktionen zum Hinzufügen von Instruktionen, Daten und Labels mit Überprüfung auf Korrektheit

- Anzeige des Hauptspeichers:
  - Breakpoints

	Address	Encoding	Instruction
		msg:	
	00000000	6c6c6548	
	00000004	6e49206f	
	00000008	7262736e	
	0000000c	216b6375	
	00000010	0000000a	
		_start:	
	00000014	e3a00001	mov r0, #1
	00000018	e0000000	ldr r1, =msg
●	0000001c	e0000000	ldr r2, =len
	00000020	e3a07004	mov r7, #4
●	00000024	ef000000	swi #0
	00000028	e3a00000	mov r0, #0
	0000002c	e3a07001	mov r7, #1
	00000030	ef000000	swi #0

Hauptspeicher

- Klasse mit Funktionen zum Hinzufügen von Instruktionen, Daten und Labels mit Überprüfung auf Korrektheit

- Anzeige des Hauptspeichers:
  - Breakpoints
  - Adresse

	Address	Encoding	Instruction
		msg:	
	00000000	6c6c6548	
	00000004	6e49206f	
	00000008	7262736e	
	0000000c	216b6375	
	00000010	0000000a	
		_start:	
	00000014	e3a00001	mov r0, #1
	00000018	e0000000	ldr r1, =msg
●	0000001c	e0000000	ldr r2, =len
	00000020	e3a07004	mov r7, #4
●	00000024	ef000000	swi #0
	00000028	e3a00000	mov r0, #0
	0000002c	e3a07001	mov r7, #1
	00000030	ef000000	swi #0

Hauptspeicher

- Klasse mit Funktionen zum Hinzufügen von Instruktionen, Daten und Labels mit Überprüfung auf Korrektheit

- Anzeige des Hauptspeichers:
  - Breakpoints
  - Adresse
  - Kodierung bzw. Daten

	Address	Encoding	Instruction
		msg:	
	00000000	6c6c6548	
	00000004	6e49206f	
	00000008	7262736e	
	0000000c	216b6375	
	00000010	0000000a	
		_start:	
	00000014	e3a00001	mov r0, #1
	00000018	e0000000	ldr r1, =msg
●	0000001c	e0000000	ldr r2, =len
	00000020	e3a07004	mov r7, #4
●	00000024	ef000000	swi #0
	00000028	e3a00000	mov r0, #0
	0000002c	e3a07001	mov r7, #1
	00000030	ef000000	swi #0

Hauptspeicher

- Klasse mit Funktionen zum Hinzufügen von Instruktionen, Daten und Labels mit Überprüfung auf Korrektheit

- Anzeige des Hauptspeichers:
  - Breakpoints
  - Adresse
  - Kodierung bzw. Daten
  - Instruktion

	Address	Encoding	Instruction
		msg:	
	00000000	6c6c6548	
	00000004	6e49206f	
	00000008	7262736e	
	0000000c	216b6375	
	00000010	0000000a	
		_start:	
	00000014	e3a00001	mov r0, #1
	00000018	e0000000	ldr r1, =msg
●	0000001c	e0000000	ldr r2, =len
	00000020	e3a07004	mov r7, #4
●	00000024	ef000000	swi #0
	00000028	e3a00000	mov r0, #0
	0000002c	e3a07001	mov r7, #1
	00000030	ef000000	swi #0

Hauptspeicher

- Klasse mit Funktionen zum Hinzufügen von Instruktionen, Daten und Labels mit Überprüfung auf Korrektheit

- Anzeige des Hauptspeichers:
  - Breakpoints
  - Adresse
  - Kodierung bzw. Daten
  - Instruktion
- Hervorheben der aktuellen Instruktion

	Address	Encoding	Instruction
		msg:	
	00000000	6c6c6548	
	00000004	6e49206f	
	00000008	7262736e	
	0000000c	216b6375	
	00000010	0000000a	
		_start:	
	00000014	e3a00001	mov r0, #1
	00000018	e0000000	ldr r1, =msg
●	0000001c	e0000000	ldr r2, =len
	00000020	e3a07004	mov r7, #4
●	00000024	ef000000	swi #0
	00000028	e3a00000	mov r0, #0
	0000002c	e3a07001	mov r7, #1
	00000030	ef000000	swi #0

Code Execution Engine

- Klasse zur Ausführung der Instruktionen im Hauptspeicher

Code Execution Engine

- Klasse zur Ausführung der Instruktionen im Hauptspeicher
- Aufgeteilt in 3 Hauptfunktionen:

Code Execution Engine

- Klasse zur Ausführung der Instruktionen im Hauptspeicher
- Aufgeteilt in 3 Hauptfunktionen:

- 1. *continue()*

Code Execution Engine

- Klasse zur Ausführung der Instruktionen im Hauptspeicher
- Aufgeteilt in 3 Hauptfunktionen:

- 1. *continue()*
  - Asynchrone Funktion mit unterschiedlichen Debugger-Geschwindigkeiten

Code Execution Engine

- Klasse zur Ausführung der Instruktionen im Hauptspeicher
- Aufgeteilt in 3 Hauptfunktionen:

- 1. *continue()*
  - Asynchrone Funktion mit unterschiedlichen Debugger-Geschwindigkeiten
  - Überprüft Abbruchbedingungen (Breakpoints, Ende einer Subroutine)

Code Execution Engine

- Klasse zur Ausführung der Instruktionen im Hauptspeicher
- Aufgeteilt in 3 Hauptfunktionen:

- 1. *continue()*
  - Asynchrone Funktion mit unterschiedlichen Debugger-Geschwindigkeiten
  - Überprüft Abbruchbedingungen (Breakpoints, Ende einer Subroutine)
  - Ruft Funktion zum Ausführen der nächsten Instruktion auf

Code Execution Engine

- Klasse zur Ausführung der Instruktionen im Hauptspeicher
- Aufgeteilt in 3 Hauptfunktionen:

- 1. *continue()*
  - Asynchrone Funktion mit unterschiedlichen Debugger-Geschwindigkeiten
  - Überprüft Abbruchbedingungen (Breakpoints, Ende einer Subroutine)
  - Ruft Funktion zum Ausführen der nächsten Instruktion auf
  - Aktualisierung der Benutzeroberfläche

Code Execution Engine

- 2. *executeNextInstruction()*

Code Execution Engine

- 2. *executeNextInstruction()*
  - Überprüft korrekt ausgerichtete Adresse

Code Execution Engine

- 2. *executeNextInstruction()*
  - Überprüft korrekt ausgerichtete Adresse
  - Holt nächste Instruktion aus Hauptspeicher

Code Execution Engine

- 2. *executeNextInstruction()*
  - Überprüft korrekt ausgerichtete Adresse
  - Holt nächste Instruktion aus Hauptspeicher
  - Aktualisiert Stacktrace

Code Execution Engine

- 2. *executeNextInstruction()*
  - Überprüft korrekt ausgerichtete Adresse
  - Holt nächste Instruktion aus Hauptspeicher
  - Aktualisiert Stacktrace

- 3. *executeInstruction()*

Code Execution Engine

- 2. *executeNextInstruction()*
  - Überprüft korrekt ausgerichtete Adresse
  - Holt nächste Instruktion aus Hauptspeicher
  - Aktualisiert Stacktrace

- 3. *executeInstruction()*
  - Überprüft Ausführungsbedingung

Code Execution Engine

- 2. *executeNextInstruction()*
  - Überprüft korrekt ausgerichtete Adresse
  - Holt nächste Instruktion aus Hauptspeicher
  - Aktualisiert Stacktrace

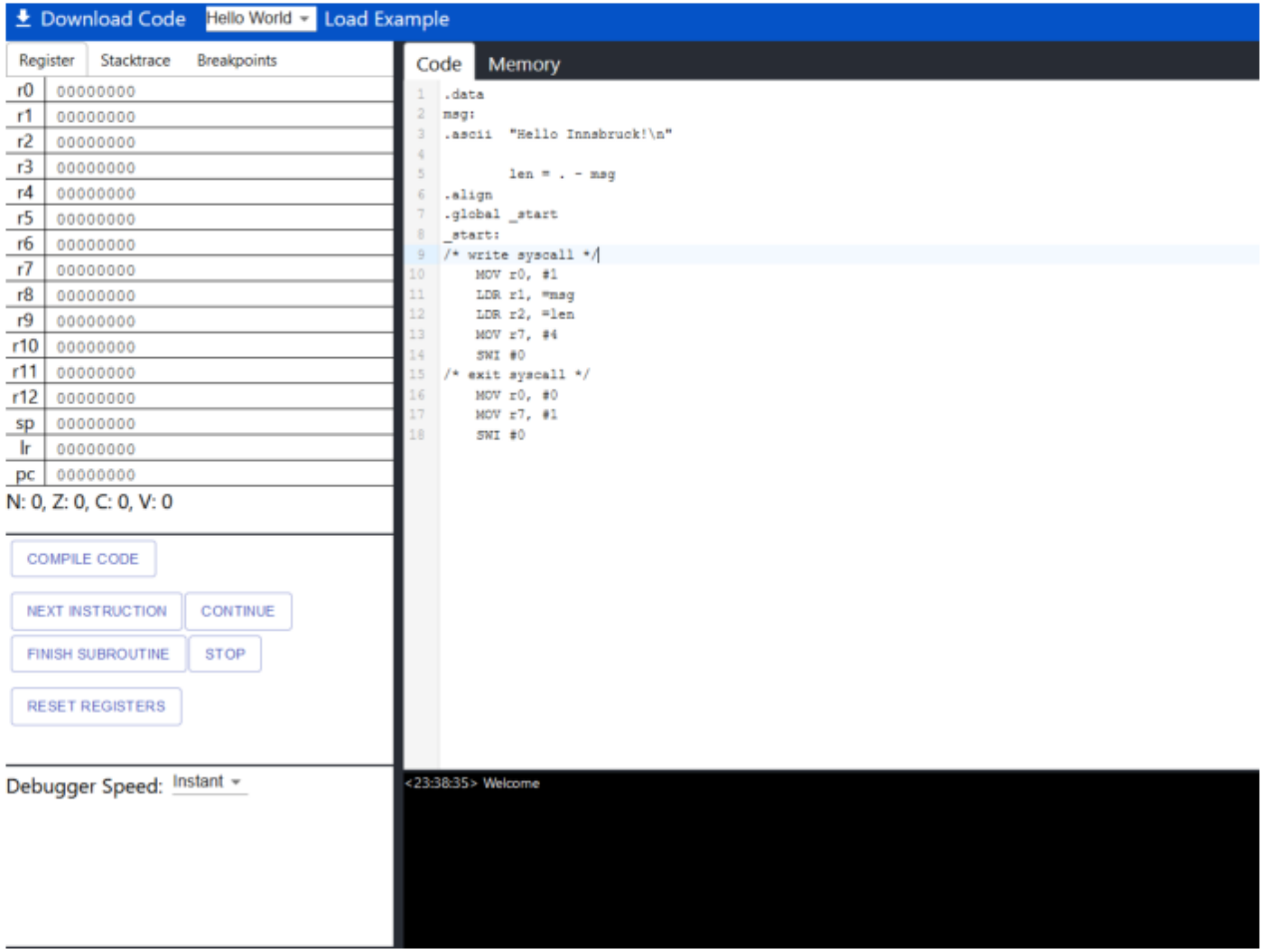
- 3. *executeInstruction()*
  - Überprüft Ausführungsbedingung
  - Ruft korrekte Funktion, je nach Typ der Instruktion auf

Code Execution Engine

- 2. *executeNextInstruction()*
  - Überprüft korrekt ausgerichtete Adresse
  - Holt nächste Instruktion aus Hauptspeicher
  - Aktualisiert Stacktrace

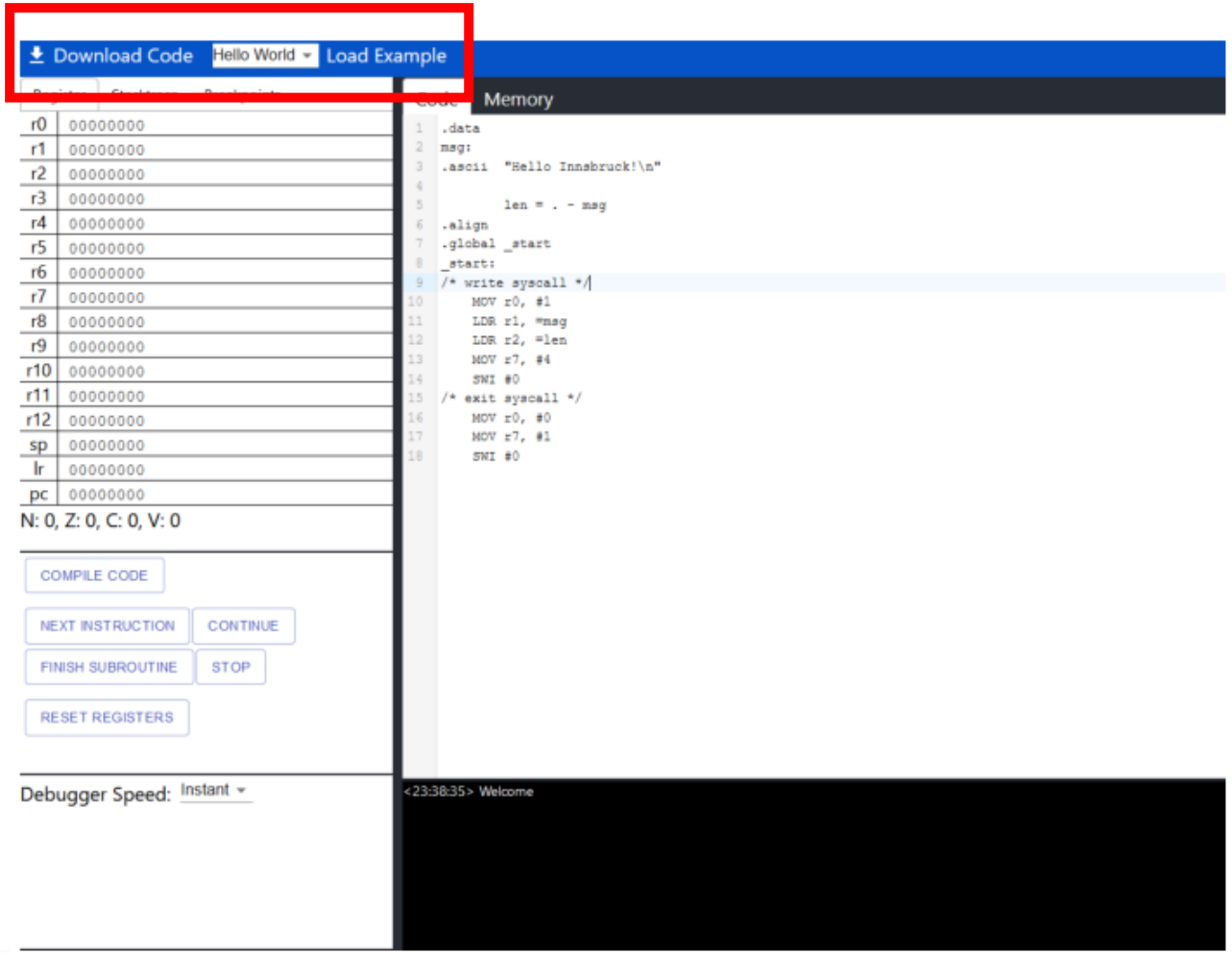
- 3. *executeInstruction()*
  - Überprüft Ausführungsbedingung
  - Ruft korrekte Funktion, je nach Typ der Instruktion auf
  - Aktualisiert Register, Hauptspeicher

Benutzeroberfläche



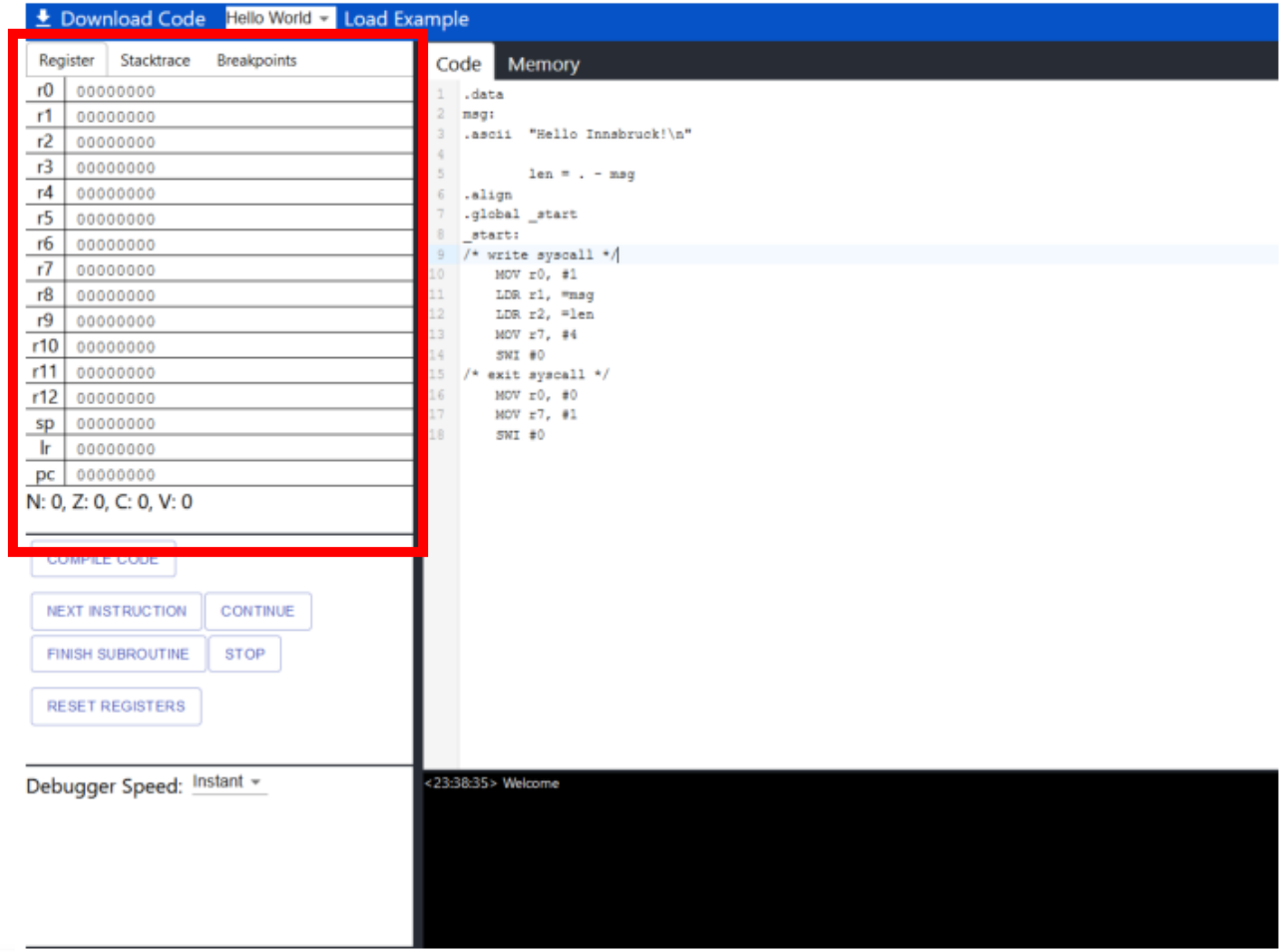
Benutzeroberfläche

- Header mit Download-Button und Dropdown-Menü zum Laden von Beispielen



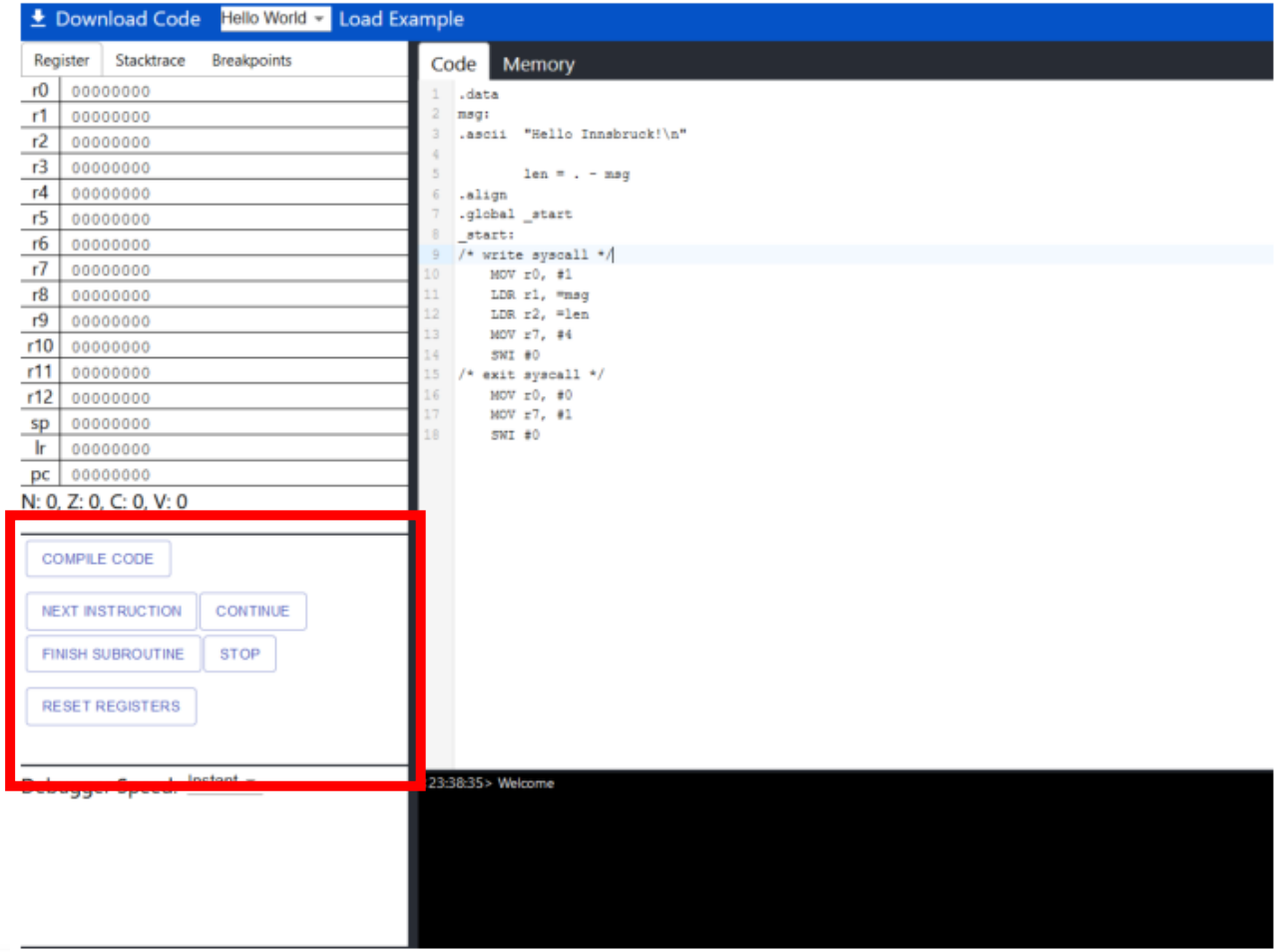
Benutzeroberfläche

- Header mit Download-Button und Dropdown-Menü zum Laden von Beispielen
- Aktuelle Register, Stacktrace und Breakpoints



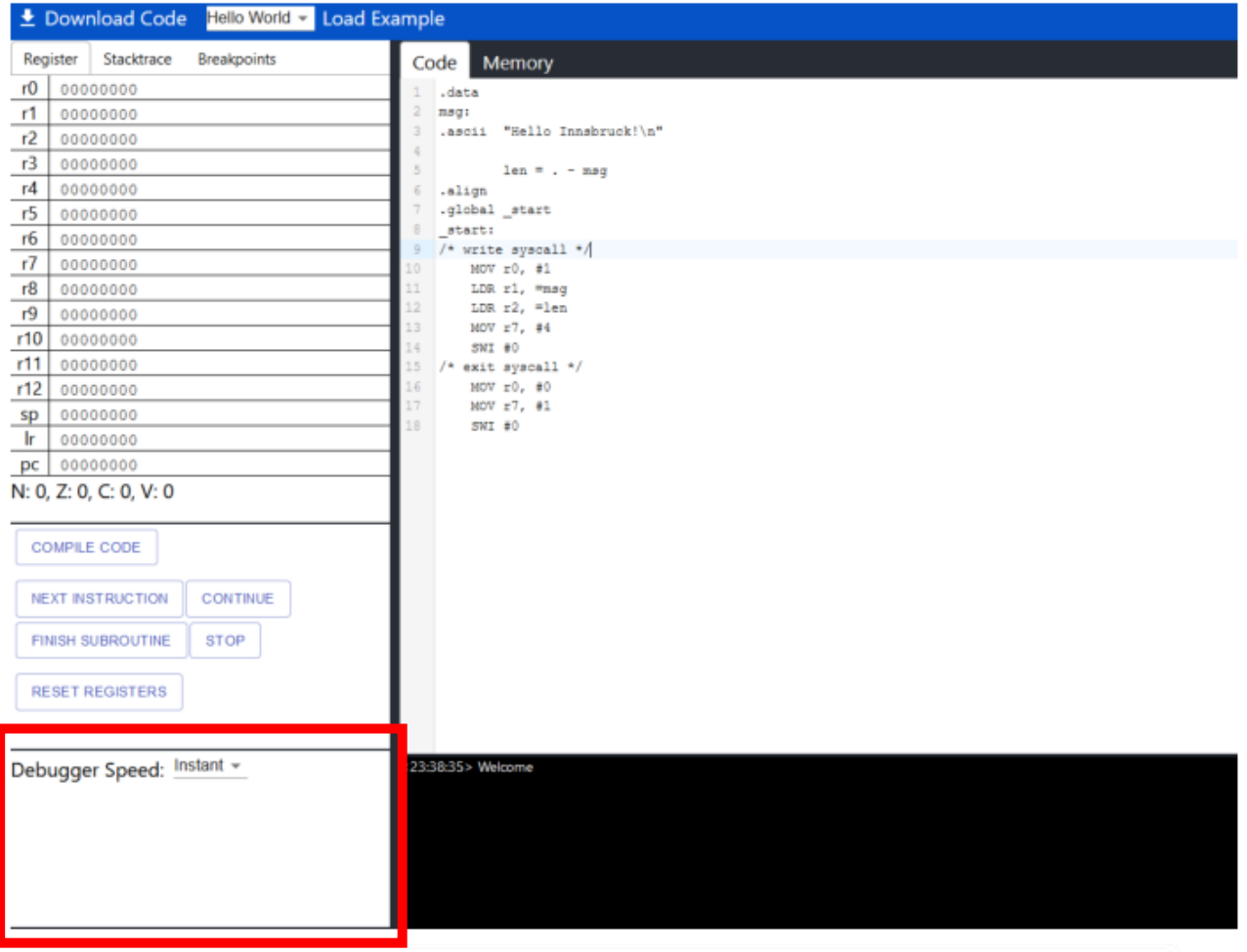
Benutzeroberfläche

- Header mit Download-Button und Dropdown-Menü zum Laden von Beispielen
- Aktuelle Register, Stacktrace und Breakpoints
- Debugger



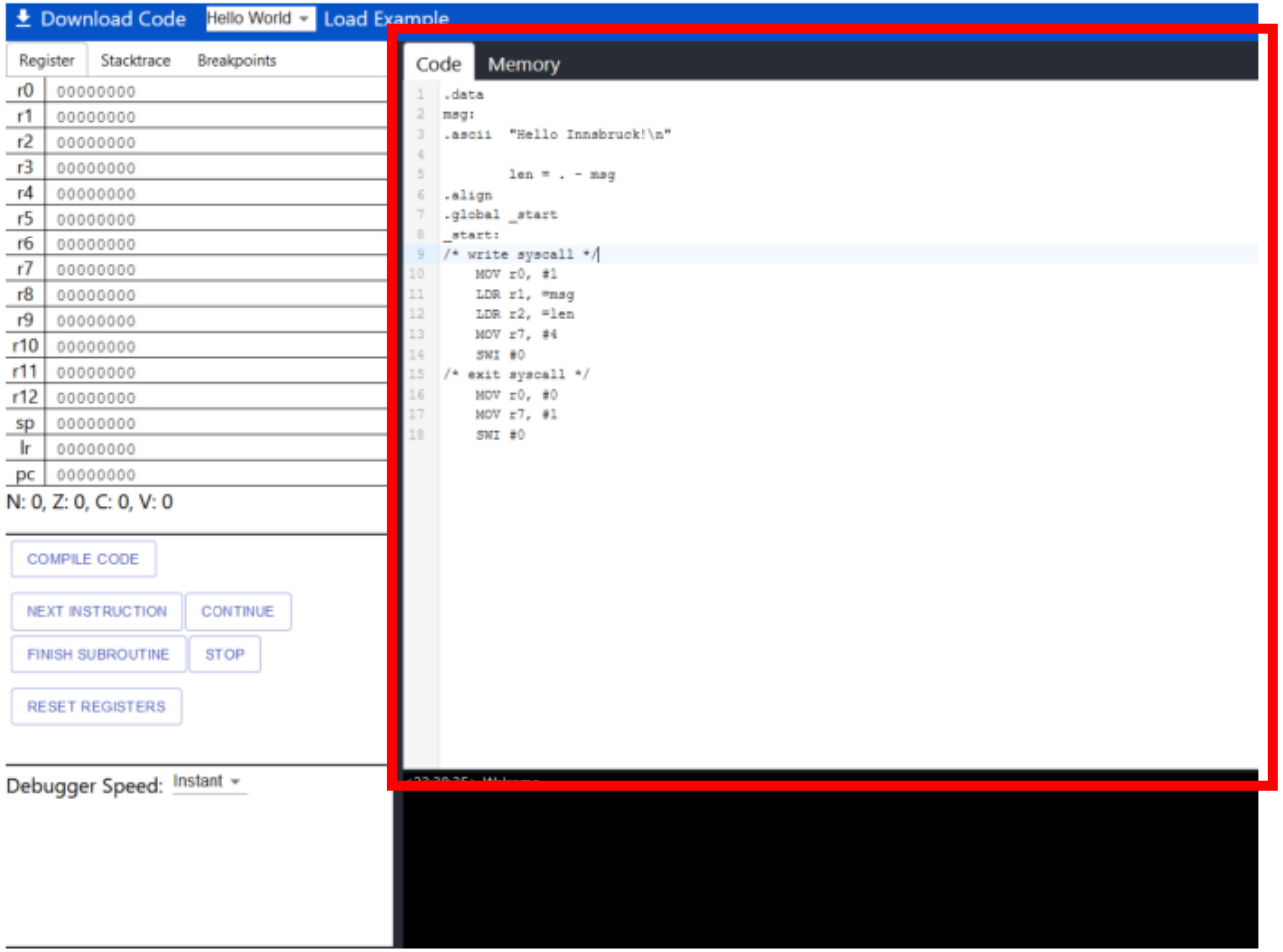
Benutzeroberfläche

- Header mit Download-Button und Dropdown-Menü zum Laden von Beispielen
- Aktuelle Register, Stacktrace und Breakpoints
- Debugger
- Optionen



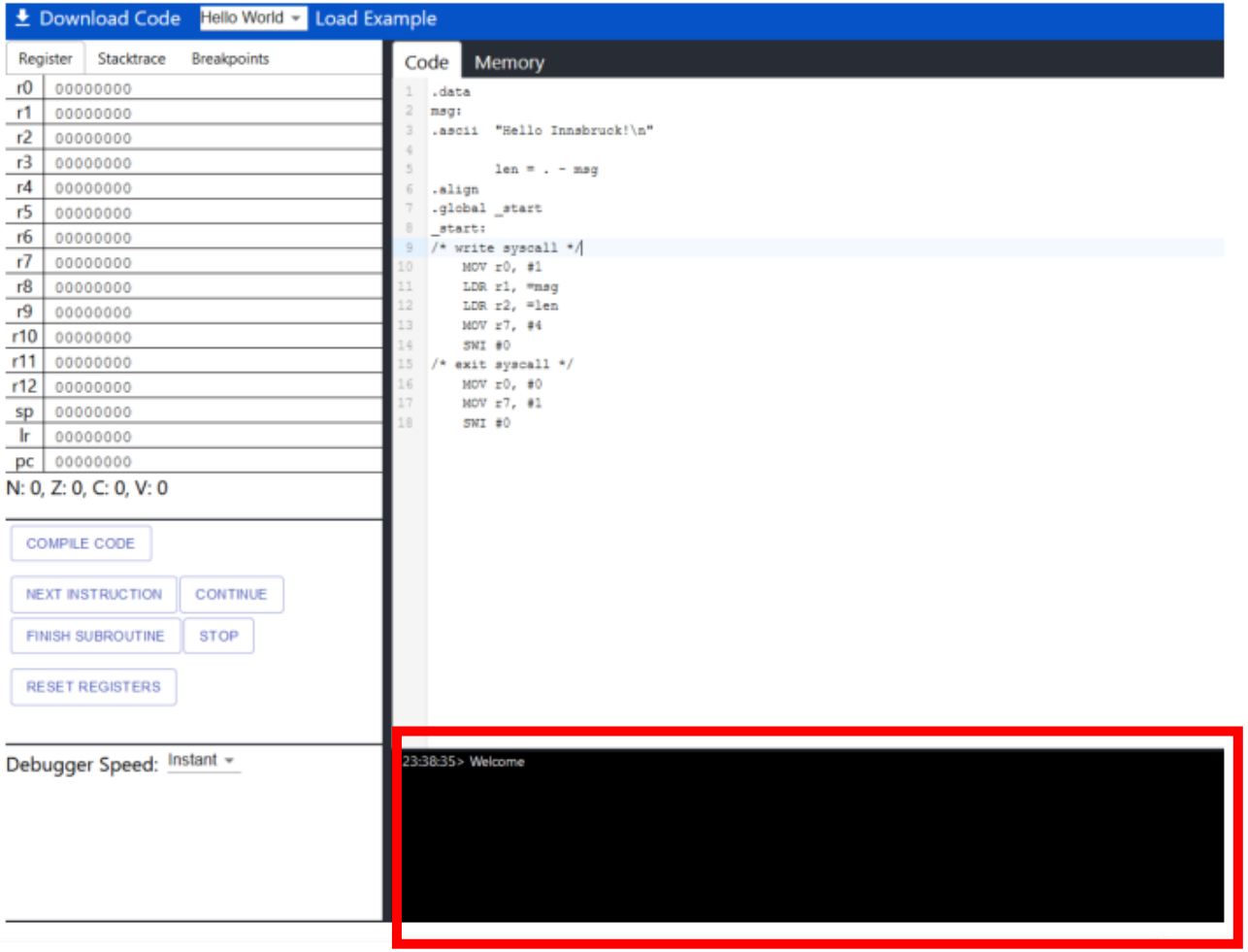
Benutzeroberfläche

- Header mit Download-Button und Dropdown-Menü zum Laden von Beispielen
- Aktuelle Register, Stacktrace und Breakpoints
- Debugger
- Optionen
- Feld für Benutzereingabe



Benutzeroberfläche

- Header mit Download-Button und Dropdown-Menü zum Laden von Beispielen
- Aktuelle Register, Stacktrace und Breakpoints
- Debugger
- Optionen
- Feld für Benutzereingabe
- Terminal



Live Demo

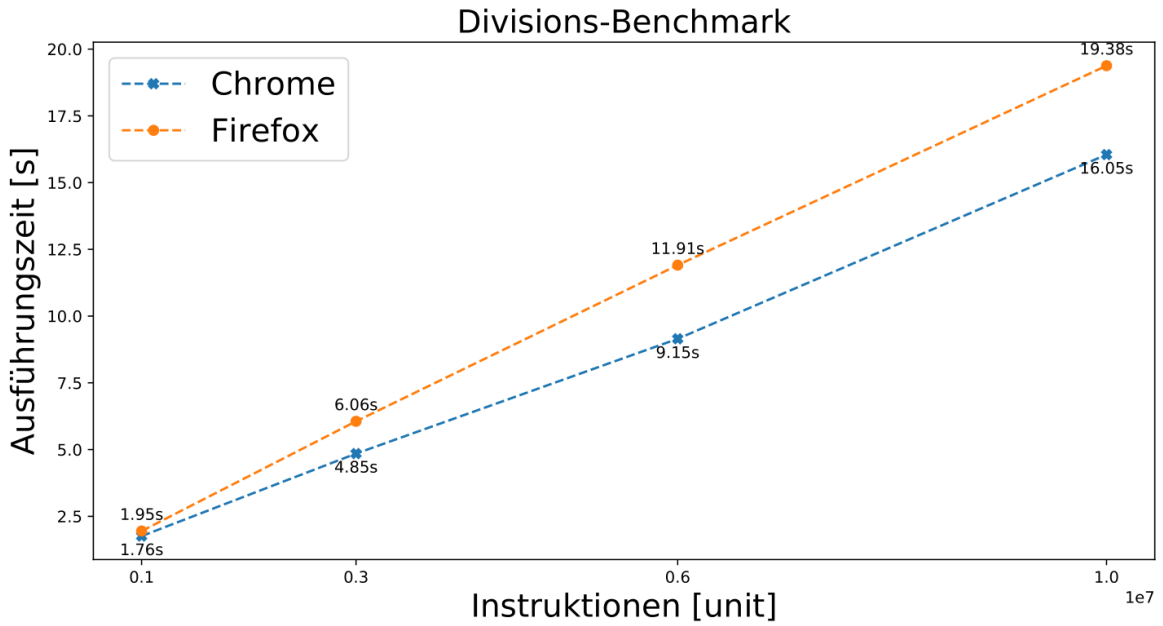
Evaluation

- Korrektheit getestet mit Beispielen aus Vorlesung/ Proseminar – Verfügbar über Dropdown-Menü in der Webanwendung

Evaluation

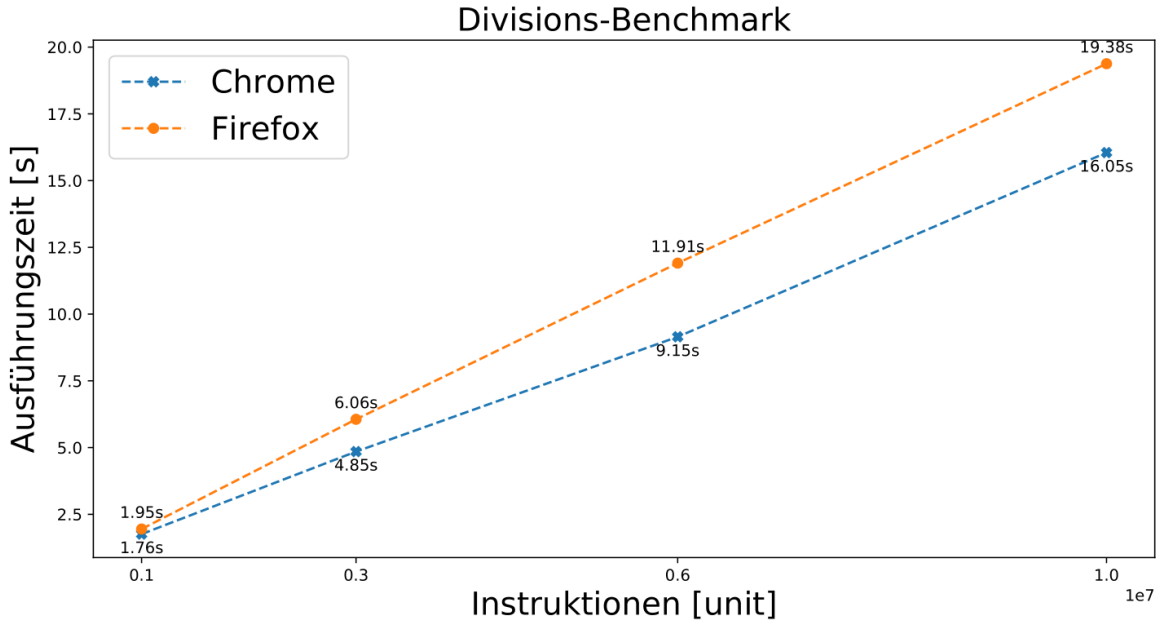
- Korrektheit getestet mit Beispielen aus Vorlesung/ Proseminar – Verfügbar über Dropdown-Menü in der Webanwendung

- Benchmark für Ausführungszeit (i5-4450 @ 3.20 GHz):



Evaluation

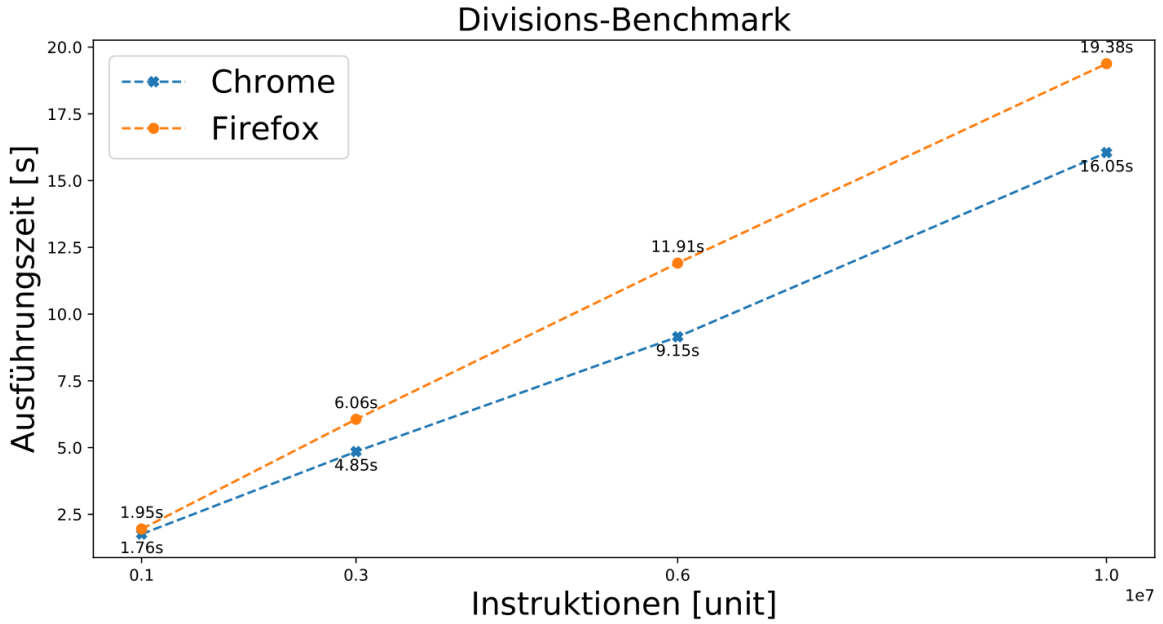
- Korrektheit getestet mit Beispielen aus Vorlesung/ Proseminar – Verfügbar über Dropdown-Menü in der Webanwendung



- Benchmark für Ausführungszeit (i5-4450 @ 3.20 GHz):
  - Arithmetische Instruktionen (Divisions-Beispiel) – 507000 Inst/s (Firefox) bzw. 616000 Inst/s (Chrome)

Evaluation

- Korrektheit getestet mit Beispielen aus Vorlesung/ Proseminar – Verfügbar über Dropdown-Menü in der Webanwendung



- Benchmark für Ausführungszeit (i5-4450 @ 3.20 GHz):
  - Arithmetische Instruktionen (Divisions-Beispiel) – 507000 Inst/s (Firefox) bzw. 616000 Inst/s (Chrome)
  - Lade- und Speicherinstruktionen (Pascal-Beispiel) – 374000 Inst/s (Firefox) bzw. 506000 Inst/s (Chrome)

Zusammenfassung

- Simulator mit allen nötigen Teilen einer ARMv5 Entwicklungsumgebung um Assembler Programme schreiben, debuggen und analysieren zu können

Zusammenfassung

- Simulator mit allen nötigen Teilen einer ARMv5 Entwicklungsumgebung um Assembler Programme schreiben, debuggen und analysieren zu können
- Ausreichende Performance für die kleinen PS-Programme

Zusammenfassung

- Simulator mit allen nötigen Teilen einer ARMv5 Entwicklungsumgebung um Assembler Programme schreiben, debuggen und analysieren zu können
- Ausreichende Performance für die kleinen PS-Programme
- Aufteilung in Operanden, Instruktionen und Teile einer CPU um Erweiterung durch zusätzliche Funktionen zu erleichtern

Referenzen

[1] ARM Limited. GNU Toolchain for ARM processors. Zugegriffen am: 29.09.2021. https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain.

[2] ARM Limited. ARMv5 Architecture Reference Manual - Issue I, 2005.

[3] G. Bierman, M. Abadi, and M. Torgersen. Understanding TypeScript. In ECOOP 2014 – Object-Oriented Programming, pages 257–281, 2014.

[4] E. Blem, J. Menon, and K. Sankaralingam. Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures. In 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA), 2013.

[5] E. Davey. tsPEG: A PEG Parser Generator for TypeScript. Zugegriffen am: 29.09.2021. https://github.com/EoinDavey/tsPEG.

[6] Facebook. React. Zugegriffen am: 29.09.2021. https://reactjs.org/.

[7] B. Ford. Parsing Expression Grammars: A Recognition-Based Syntactic Foundation. SIGPLAN Not., 39(1):111–122, January 2004.

[8] P. Knaggs. ARM Assembly Language Programming, 2016.

[9] L . Lee. Fast context-free grammar parsing requires fast boolean matrix multiplication. J. ACM, 49(1), January 2002.

[10] Microsoft. TypeScript. Zugegriffen am: 29.09.2021 . https://www.typescriptlang.org/.

[11] Microsoft. Windows Subsystem for Linux. Zugegriffen am: 29.09.2021. https://docs.microsoft.com/en-us/windows/wsl/install-win10.

[12] The GNU Project. GDB: The GNU Project Debugger. Zugegriffen am: 29.09.2021. https://www.gnu.org/software/gdb/.

[13] The QEMU Project Developers. QEMU User Mode Emulation. Zugegriffen am: 29.09.2021. https://qemu.readthedocs.io/en/latest/user/index.html.

[14] H. Wong. CPUlator: A CPU and I/O device simulator. Zugegriffen am: 29.09.2021. https://cpulator.01xz.net/?sys=arm.

[15] J. Mossberg. Use GDB on an ARM assembly program. Zugegriffen am: 04.03.2021. https://jacobmossberg.se/posts/2017/01/17/use-gdb-on-arm-assembly-program.html

www.uibk.ac.at