TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Technische Informatik, Professur für VLSI-Entwurfssysteme, Diagnostik und Architektur

Verteidigung des INF-PM-FP-ANW: Entwicklung einer MIPS-CPU

Dresden, 5. Mai 2014

Gliederung

- Aufgabenstellung und Arbeitsumgebung
- Entwurf des Prozessors
- Ausarbeitung: Compiler-Backends

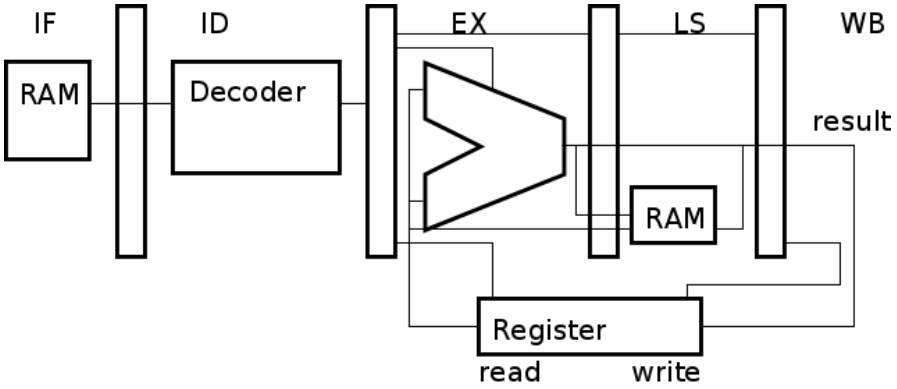
Arbeitsumgebung

- Spartan 3 XC3S200-4FT256
- 50 MHz (20 ns)
- Bootram
- UART
- ISE 13.4

Aufgabenstellung

- CPU mit MIPS Instruction Set
- 5 stufige Pipeline
- Getrennter Befehls- und Datenbus
- Wishbone-Protokoll für Befehle und Daten
- Instruction Cache, der die Burst-Logik des Wishbone-Protokolls nutzt
- Load- und Store-Befehle mit natürlich ausgerichteten Wörtern, Halbwörtern und Bytes
- Eigene ALU

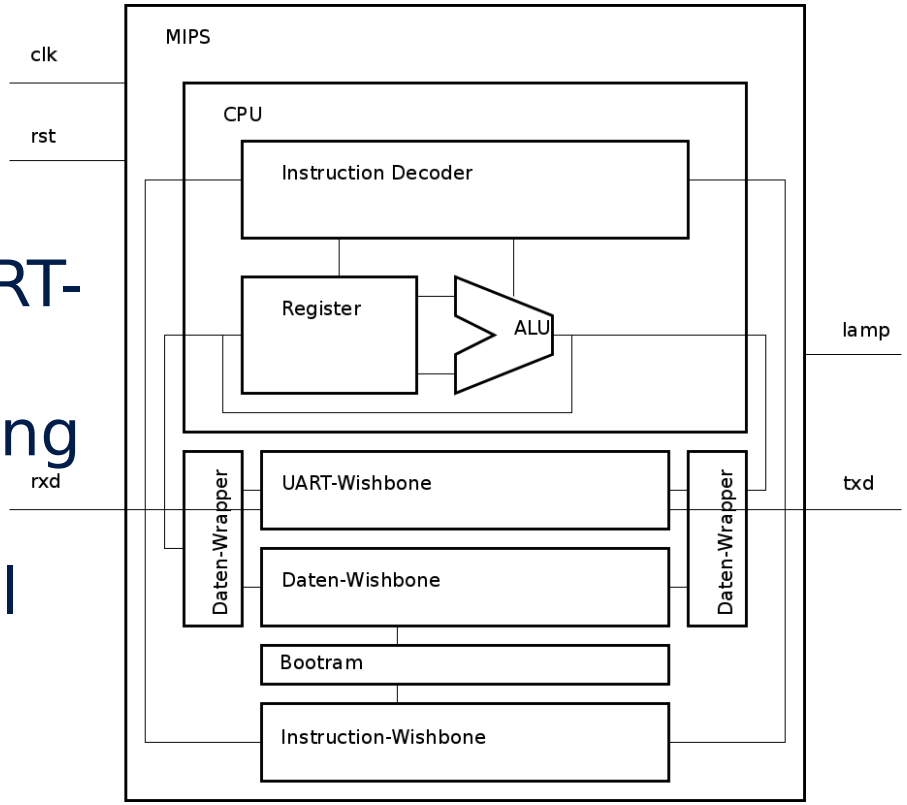
Aufbau der Pipeline



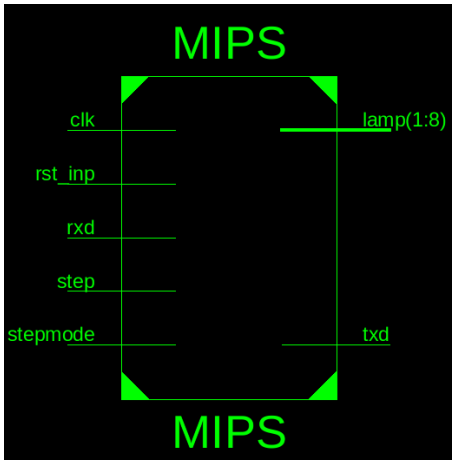
- IF aus dem Instruction Cache
- ID-Ergebnis kommt in eine Datenstruktur
- EX nutzt Registerbank-Werte und Immediate-Werte aus dem ID
- LS nutzt Adressberechnung aus der ALU
- LS-Ergebnis und ALU-Ergebnis werden in WB in die Registerbank zurückgeführt

Unterteilung in Module

- MIPS: physischer UART-Zugriff, Bootram mit Wishbone-Ansteuerung
- CPU: MIPS-Prozessor mit Speicherprotokoll
- Instruction Decoder
- Registerbank
- ALU



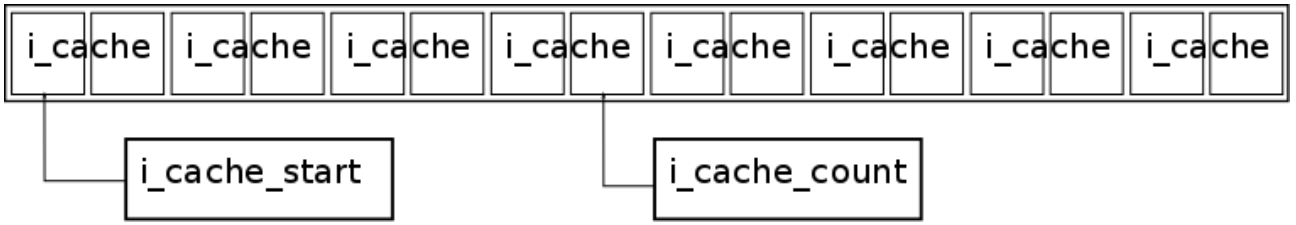
MIPS-Modul



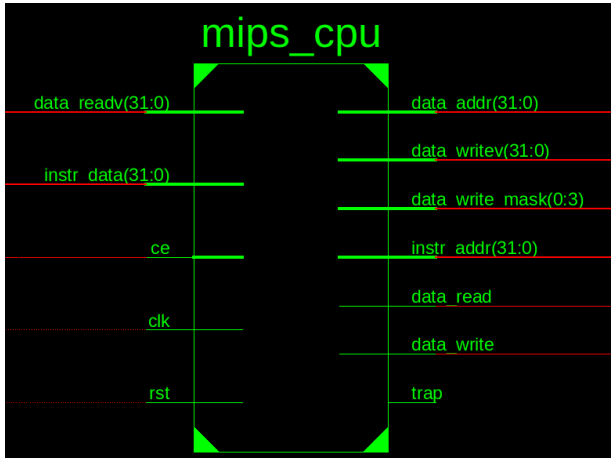
- Äußere Pins des FPGA: clk, rst, rxd, txd
- Instanziiert eine CPU
- Instanziiert Bootram
- Versorgt CPU mit Instruktion und Daten
- Implementiert Instruction Cache, der den Instruction Wishbone nutzt
- Wrapt UART-Wishbone für IO und Bootram-Wishbone für Daten für die Daten-Versorgung der CPU
- Steuert CE der CPU, um auf Speicher warten zu können

Instruction Cache

- 3 Register: i_cache_start, i_cache_count und i_cache
- Lesen innerhalb der Grenzen von i_cache_start und i_cache_count in einem Takt
- Lesen außerhalb dieser Grenzen sorgt für neuen Lesebefehl
- Burst-Logik des Wishbone wird angestoßen und füllt in jedem Takt ein Datenwort in den Cache
- Burst-Logik kann vorzeitig unterbrochen werden



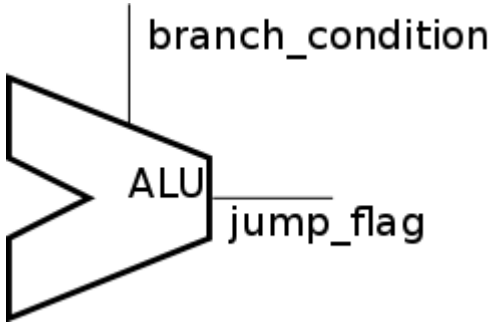
CPU-Modul



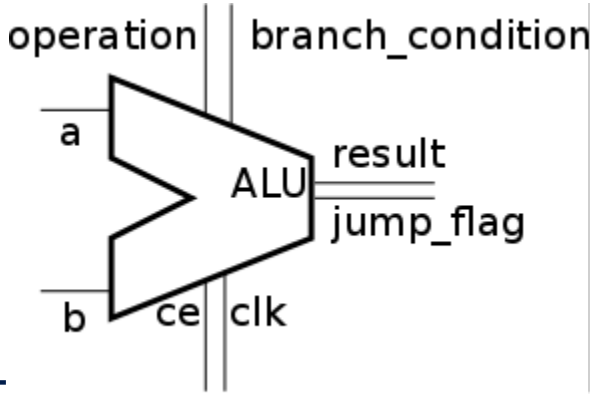
- CE-Flag kann die CPU einfrieren
- Pins für Instruktionen: instr_addr, instr_data
- Pins für Daten: data_addr, data_read, data_readv, data_write, data_write_mask, data_writev
- Zählt Instruktionszeiger (oder verzweigt, wenn die ALU dies vorschreibt zu der vom ID vorgegebenen Adresse)

Sprünge

- idresult_ls.branch_condition<>never?
- Warte auf jump_flag der ALU (diese wertet die Bedingung aus)
- Wenn jump_flag='1', ersetze Befehl aus dem ID durch NOP (nur 1 Branch delay slot)
- Sprungadress-Berechnung
  - Register (2. Slot)
  - Immediate-Relativ
  - Immediate-Absolut



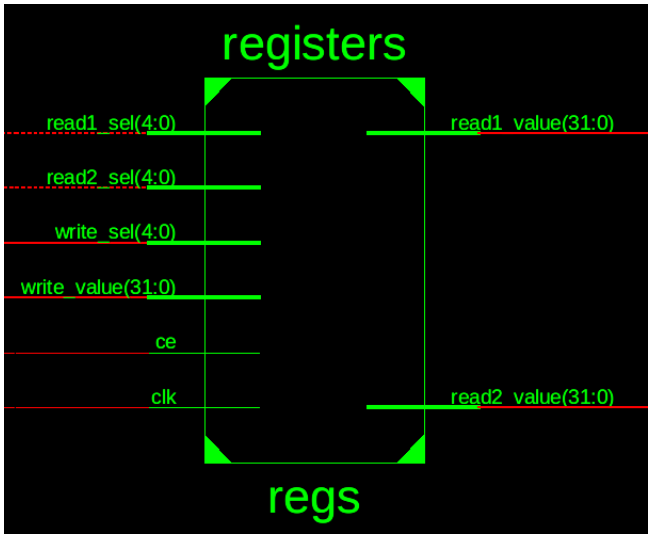
ALU

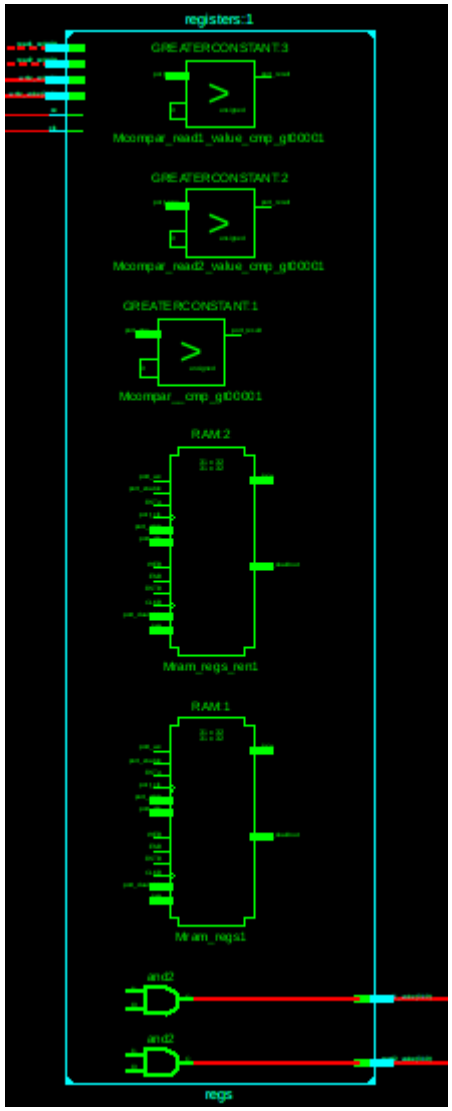


- Eingang a, b, shift_ammount
- Ausgang result, jump_flag
- Arithmetik/Logik (mit selbst implementierten Barrel Shifter)
- Auswertung einer Sprungbedingung
- Logisch gesehen: Schaltwerk
- Physisch aber ein Schaltnetz, da Pipelineregister in der ALU sind → in VHDL-Syntax leichter auszudrücken wegen dem CASE-Statement

Registerbank

- 2 Lese-, ein Schreibslot
- In Hardware: 2 geklonte MRAM-Blöcke





CPU-Testbench

- Ersatz für das MIPS-Modul
- Instanziiert ebenfalls CPU
- Getrennten Instruktions- und Datenspeicher
- Testfälle im Instruktionsspeicher
- Datenspeicher und Instruktionszeiger werden genutzt, um die CPU zu evaluieren/testen
- Keine IO und keinen I-Cache

Ergebnisse

- Hello World klappt (und andere Modifikationen auch)

Folgende Auslastung des FPGA wurde erreicht:

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops	636	3,840	16%	
Number of 4 input LUTs	3,321	3,840	86%	
Number of occupied Slices	1,817	1,920	94%	
Number of Slices containing only related logic	1,817	1,817	100%	
Number of Slices containing unrelated logic	0	1,817	0%	
Total Number of 4 input LUTs	3,352	3,840	87%	
Number used as logic	2,969			
Number used as a route-thru	31			
Number used for Dual Port RAMs	352			
Number of bonded IOBs	14	173	8%	
Number of RAMB16s	4	12	33%	
Number of BUFGMUXs	1	8	12%	
Average Fanout of Non-Clock Nets	4.20			

Ausarbeitung: Compiler-Backends

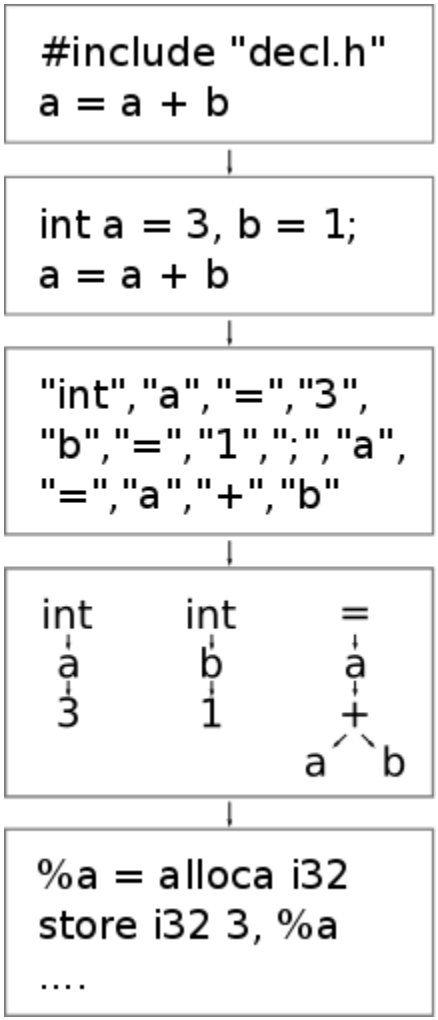
- Besonderheiten MIPS-Befehlssatz
- Compilerbau allgemein
- Application Binary Interface
- Codegenerator des GCC
- Codegenerator der LLVM

MIPS-Befehlssatz

- Wenige Befehlstypen (nicht viele Peephole-Optimierungen möglich)
- 3-Adress-Maschine (vereinfacht den Registerallokator)
- Je Takt nur ein Speicherbefehl (keine komplexen ADD [MEM], imm wie bei x86)
- Branch Delay Slots (mehr Arbeit für den Compiler)
- 32 GPR (weit mehr als x86)
- Keine expliziten Stack-Operationen

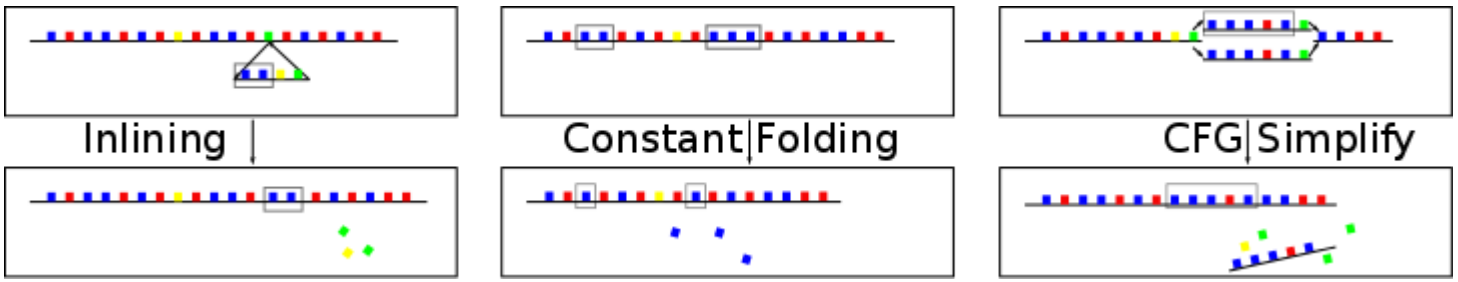
Compilerbau: Ablauf der Übersetzung

- Präprozessor
- Lexer: Tokenisierung des Codes
- Parser: Aufbau des Syntaxbaums
- Evtl. Übersetzung des Syntaxbaums in eine andere Zwischensprache
- Optimierungen auf diesem Code
- Codegenerierung
- (Linken)



Optimierungen

- Passes:
  - Inlining (kleine Funktionen)
  - Konstantenfaltung, Vereinfachung von Berechnungen
  - Vereinfachung und Sortierung von Verzweigungen
  - Schleifennormalisierung
  - Extraktion von Schleifeninvarianten
  - Entrollung von Schleifen

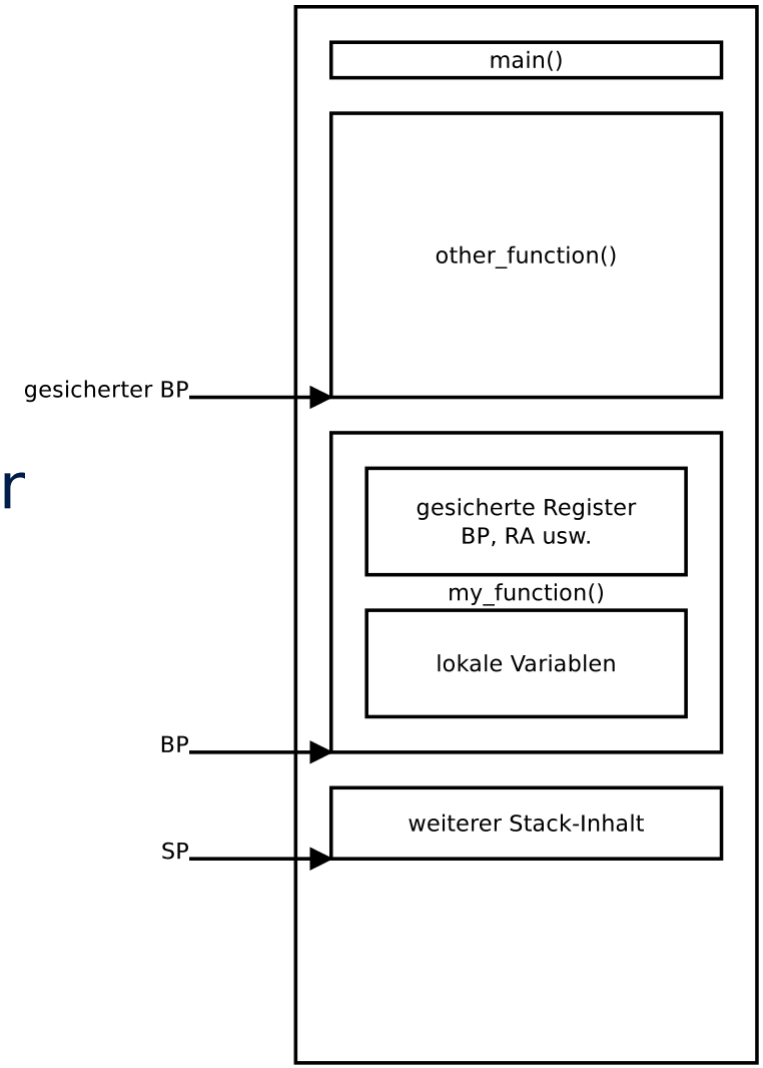


Application Binary Interface

- Spezifikation des Ablaufs von Funktionsaufrufen
- Konvention zur Benutzung von Registern
- Layout des Stack-Frames
- Parameterübergabe (evtl. mehrere Konventionen)
- Speicherort von lokalen Variablen

Stackframe

- Layout des Stacks
- Verschachtelte Funktionsaufrufe
- Welche Register müsser wie gesichert werden?
- In der Funktion: Prolog, Body und Epilog



Registerallokator

- Zuweisung von virtuellen Registern und Variablen zu physischen Registern
- Bei Registerdruck: Welche Werte müssen auf den Stack, ohne viel Geschwindigkeit einzubüßen? Mehrfachbelegung?

Nummer	Nutzung	Muss bei Aufruf gesichert werden
0	0-Konstante	-
1	Temorär	nein
2-3	Rückgabewerte und Zwischenergebnisse	nein
4-11*	Aufrufparameter	nein
12-15*	Temporär	nein
16-23	gesicherte Temporäre	ja
24-25	Temporär	nein
26-27	reserviert fürs Betriebssystem	-
28	Global-Zeiger	ja
29	Kellerzeiger	ja
30	Zeiger auf das Stack Frame (Basiszeiger)	ja
31	Rucksprungadresse	ja

GCC

- In C implementiert, viele Altlasten
- 2 Zwischensprachen: GENERIC und GIMPLE
- GIMPLE ist 3-Adress SSA
- Unterteilung in HIGH GIMPLE und LOW GIMPLE: HIGH GIMPLE hat noch TRY, BIND, CATCH usw.
- Optimierungen laufen auf GIMPLE
- Optimierungs-Passes haben Kostenmodell; solange sich Code verbessert, optimiere

Codegenerierung im GCC

- Beschreibung des Prozessors in .md-Datei
  - Tiefe der Pipeline
  - Verfügbare Anweisungen
  - Taktgenaues Kostenmodell
  - Meherere .md-Dateien für verschiedene MIPS
- Selektion einer .md per -march=
- Durchlauf durch alle GIMPLE-Instruktion
- Je Instruktionstyp ein Callback im Codegenerator (mips_assign, mips_goto etc.)
- Jeder Callback schreibt Assembler-Instruktionen in einen Ausgabetreiber (AS)

LLVM

- Compiler-Tools: C-Compiler, Code-Generatoren, Optimierer
- Zentraler Bestandteil: Zwischensprache LLVM-IR (plattformagnostisch)
- Aufbau:
  - Frontend generiert aus C-Code LLVM-IR
  - Analyse-Passes analysieren Code
  - Optimierungs-Passes manipulieren LLVM-IR und können dazu Ergebnisse aus Analyse-Passes nutzen
  - Backend (Codegenerator) erzeugt aus LLVM-IR Maschinencode für eine konkrete CPU

LLVM-IR

- Module:
  - Deklaration vs Definition
  - Globale Variablen, Funktionen und Konstanten
- Funktionen
  - Basic-Blocks: Entry Block, Branches, Return
  - SSA
  - PHI-Knoten
  - Alloca: Alloziiere einen Platz auf dem Stack

Codegenerierung in LLVM

- Erzeugung einer MCInst-Liste aus dem LLVM-IR
  - Register alloziieren
  - LLVM-Instruktion → Maschineninstruktion
  - Registerverschiebungen (z.B. bei Rücksprüngen)
- Peephole-Pass über MCInst-Liste (Vergleiche und Sprünge zu BXX-Instruktionen zusammenfassen)
- Drucken der MCInst-Liste in Objektdatei oder direkt in RAM

Vergleich GCC vs LLVM

Kriterium	GCC	LLVM
Implementiert in	C	C++
Zwischensprache	GIMPLE	LLVM-IR
Codegenerierung	Callbacks, print-asm-Instruktion, AS	Linked List von MachineInstr
Optimierungs-Passes	Taktgenaues Kostenmodell	Nur Optimierungen, die auf allen (MIPS-)Plattformen Vorteile bringen
Gemeinsamkeiten	Zwischensprache, Optimierungs-Passes, SSA, Register-Allokator	

»Wissen schafft Brücken.«