TECHNISCHE
UNIVERSITÄT
DARMSTADT

CODEERZEUGUNG

Embedded Systems & Applications

Übersicht

Übersicht

TECHNISCHE UNIVERSITÄT DARMSTADT

- Verfügbare **Targets** im Backend: x86, ARM, PowerPC, **RISC-V**, …

Übersicht

TECHNISCHE UNIVERSITÄT DARMSTADT

- Verfügbare **Targets** im Backend: x86, ARM, PowerPC, **RISC-V**, …
- Alle nicht-exotischen Prozessor-Befehlssätze ähneln sich, daher sind in den entsprechenden Targets ähnliche Aufgaben zu erledigen

Übersicht

TECHNISCHE UNIVERSITÄT DARMSTADT

- Verfügbare **Targets** im Backend: x86, ARM, PowerPC, **RISC-V**, …
- Alle nicht-exotischen Prozessor-Befehlssätze ähneln sich, daher sind in den entsprechenden Targets ähnliche Aufgaben zu erledigen
- Exotische Backends (C++, Verilog, …) sind natürlich auch möglich

Übersicht

TECHNISCHE UNIVERSITÄT DARMSTADT

- Verfügbare **Targets** im Backend: x86, ARM, PowerPC, **RISC-V**, …
- Alle nicht-exotischen Prozessor-Befehlssätze ähneln sich, daher sind in den entsprechenden Targets ähnliche Aufgaben zu erledigen
- Exotische Backends (C++, Verilog, …) sind natürlich auch möglich

- **The LLVM Target-Independent Code Generator**

Übersicht

TECHNISCHE UNIVERSITÄT DARMSTADT

- Verfügbare **Targets** im Backend: x86, ARM, PowerPC, **RISC-V**, …
- Alle nicht-exotischen Prozessor-Befehlssätze ähneln sich, daher sind in den entsprechenden Targets ähnliche Aufgaben zu erledigen
- Exotische Backends (C++, Verilog, …) sind natürlich auch möglich

- **The LLVM Target-Independent Code Generator**
  - Algorithmen und Zwischendarstellungen architekturneutral implementiert

Übersicht

TECHNISCHE UNIVERSITÄT DARMSTADT

- Verfügbare **Targets** im Backend: x86, ARM, PowerPC, **RISC-V**, …
- Alle nicht-exotischen Prozessor-Befehlssätze ähneln sich, daher sind in den entsprechenden Targets ähnliche Aufgaben zu erledigen
- Exotische Backends (C++, Verilog, …) sind natürlich auch möglich

- **The LLVM Target-Independent Code Generator**
  - Algorithmen und Zwischendarstellungen architekturneutral implementiert
  - Targets enthalten viel Code zur **Beschreibung** ihrer Eigenschaften

Übersicht

TECHNISCHE UNIVERSITÄT DARMSTADT

- Verfügbare **Targets** im Backend: x86, ARM, PowerPC, **RISC-V**, …
- Alle nicht-exotischen Prozessor-Befehlssätze ähneln sich, daher sind in den entsprechenden Targets ähnliche Aufgaben zu erledigen
- Exotische Backends (C++, Verilog, …) sind natürlich auch möglich

- **The LLVM Target-Independent Code Generator**
  - Algorithmen und Zwischendarstellungen architekturneutral implementiert
  - Targets enthalten viel Code zur **Beschreibung** ihrer Eigenschaften
    - TableGen-Format (*.td)

Übersicht

TECHNISCHE UNIVERSITÄT DARMSTADT

- Verfügbare **Targets** im Backend: x86, ARM, PowerPC, **RISC-V**, …
- Alle nicht-exotischen Prozessor-Befehlssätze ähneln sich, daher sind in den entsprechenden Targets ähnliche Aufgaben zu erledigen
- Exotische Backends (C++, Verilog, …) sind natürlich auch möglich

- **The LLVM Target-Independent Code Generator**
  - Algorithmen und Zwischendarstellungen architekturneutral implementiert
  - Targets enthalten viel Code zur **Beschreibung** ihrer Eigenschaften
    - TableGen-Format (*.td)
  - Targets können außerdem Hooks implementieren, wenn besondere Anforderungen bestehen

- Plan: Weg einer Division durch das RISC-V-Target

TECHNISCHE UNIVERSITÄT DARMSTADT

- Plan: Weg einer Division durch das RISC-V-Target

- *RISC-V ???*

Embedded Systems & Applications

- Plan: Weg einer Division durch das RISC-V-Target

- *RISC-V ???*
- Moderner, offener/freier Befehlssatz gedacht für alles zwischen Embedded … HPC

RISC-V

TECHNISCHE UNIVERSITÄT DARMSTADT

- Plan: Weg einer Division durch das RISC-V-Target

- *RISC-V ???*
- Moderner, offener/freier Befehlssatz gedacht für alles zwischen Embedded … HPC
- Modular durch Befehlssatzerweiterungen

Base	Version	Frozen?
RV32I	2.0	Y
RV32E	1.9	N
RV64I	2.0	Y
RV128I	1.7	N
Extension	**Version**	**Frozen?**
M	2.0	Y
A	2.0	Y
F	2.0	Y
D	2.0	Y
Q	2.0	Y
L	0.0	N
C	2.0	Y
B	0.0	N
J	0.0	N
T	0.0	N
P	0.1	N
V	0.2	N
N	1.1	N

RISC-V

TECHNISCHE UNIVERSITÄT DARMSTADT

- Plan: Weg einer Division durch das RISC-V-Target

- *RISC-V ???*
- Moderner, offener/freier Befehlssatz gedacht für alles zwischen Embedded … HPC
- Modular durch Befehlssatzerweiterungen

- Sauber implementiertes LLVM-Target ohne Altlasten

Base	Version	Frozen?
RV32I	2.0	Y
RV32E	1.9	N
RV64I	2.0	Y
RV128I	1.7	N
Extension	**Version**	**Frozen?**
M	2.0	Y
A	2.0	Y
F	2.0	Y
D	2.0	Y
Q	2.0	Y
L	0.0	N
C	2.0	Y
B	0.0	N
J	0.0	N
T	0.0	N
P	0.1	N
V	0.2	N
N	1.1	N

Zwischendarstellungen

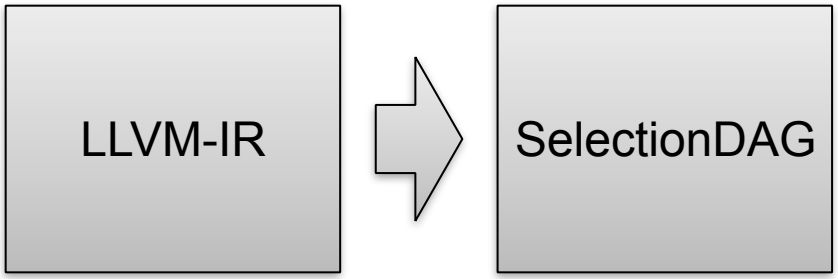
TECHNISCHE UNIVERSITÄT DARMSTADT

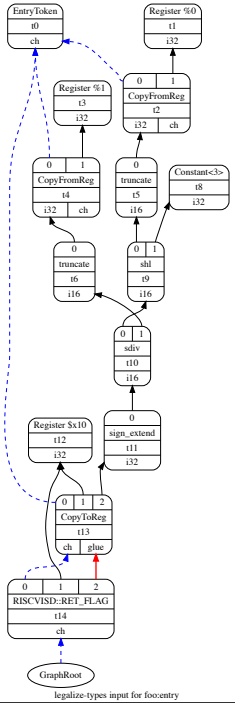
LLVM-IR

Embedded Systems & Applications

Zwischendarstellungen

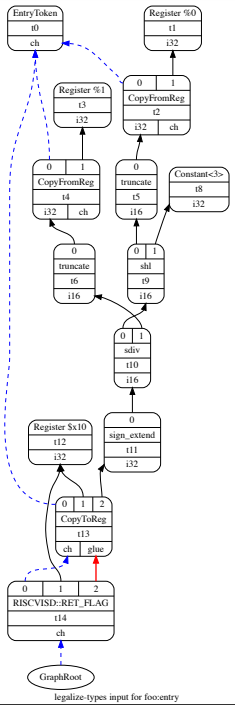
TECHNISCHE UNIVERSITÄT DARMSTADT





Embedded Systems & Applications

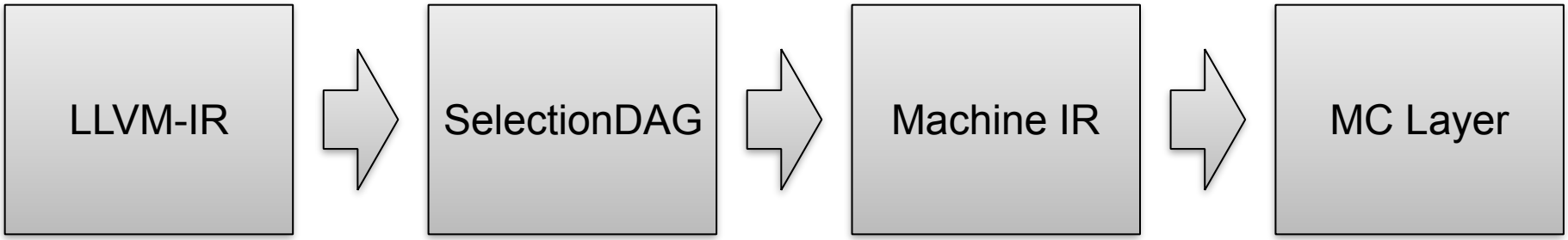
Zwischendarstellungen

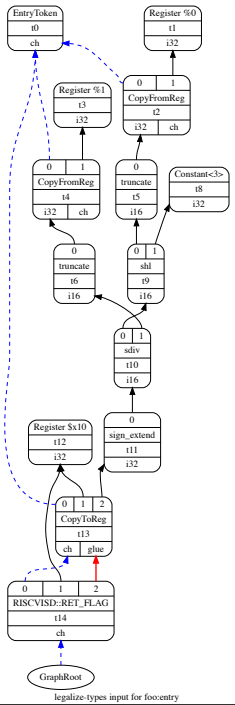



```
# Machine code for function foo: IsSSA, TracksLiveness
Function Live Ins: $x10 in %0, $x11 in %1

bb.0.entry:
  liveins: $x10, $x11
  %1:gpr = COPY $x11
  %0:gpr = COPY $x10
  %2:gpr = SLLI %1:gpr, 16
  %3:gpr = SRAI killed %2:gpr, 16
  %4:gpr = SLLI %0:gpr, 19
  %5:gpr = SRAI killed %4:gpr, 16
  %6:gpr = DIV killed %5:gpr, killed %3:gpr
  %7:gpr = SLLI killed %6:gpr, 16
  %8:gpr = SRAI killed %7:gpr, 16
  $x10 = COPY %8:gpr
  PseudoRET implicit $x10

# End machine code for function foo.
```

Zwischendarstellungen

TECHNISCHE UNIVERSITÄT DARMSTADT






```
# Machine code for function foo: IsSSA, TracksLiveness
Function Live Ins: $x10 in %0, $x11 in %1

bb.0.entry:
  liveins: $x10, $x11
  %1:gpr = COPY $x11
  %0:gpr = COPY $x10
  %2:gpr = SLLI %1:gpr, 16
  %3:gpr = SRAI killed %2:gpr, 16
  %4:gpr = SLLI %0:gpr, 19
  %5:gpr = SRAI killed %4:gpr, 16
  %6:gpr = DIV killed %5:gpr, killed %3:gpr
  %7:gpr = SLLI killed %6:gpr, 16
  %8:gpr = SRAI killed %7:gpr, 16
  $x10 = COPY %8:gpr
  PseudoRET implicit $x10

# End machine code for function foo.
```

```
        .text
        .file       "test.c"
        .globl      foo
        .p2align    2
        .type       foo,@function
foo:
        addi        sp, sp, -16
        sw          ra, 12(sp)
        sw          s0, 8(sp)
        addi        s0, sp, 16
        slli        a1, a1, 16
        srai        a1, a1, 16
        slli        a0, a0, 19
        srai        a0, a0, 16
        div         a0, a0, a1
        slli        a0, a0, 16
        srai        a0, a0, 16
        lw          s0, 8(sp)
        lw          ra, 12(sp)
        addi        sp, sp, 16
        ret
```

LLVM-IR —> SelectionDAG

TECHNISCHE UNIVERSITÄT DARMSTADT

Embedded Systems & Applications

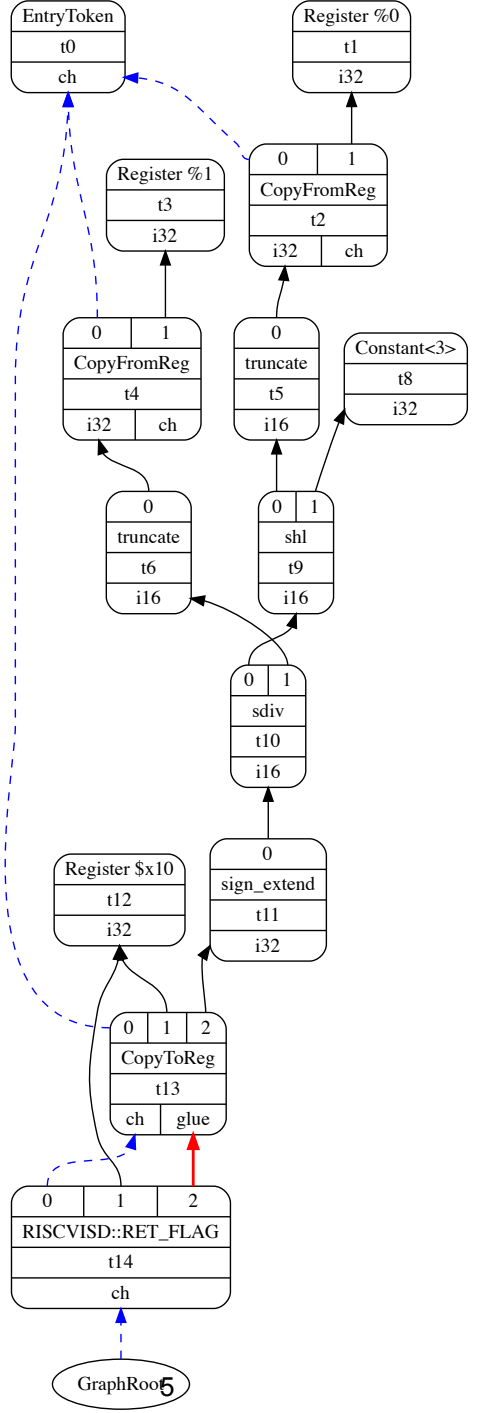
LLVM-IR —> SelectionDAG

TECHNISCHE UNIVERSITÄT DARMSTADT

```
define i32 @foo(i32 %a, i32 %b) #0 {
entry:
  %a16 = trunc i32 %a to i16
  %b16 = trunc i32 %b to i16
  %shl = shl i16 %a16, 3
  %div = sdiv i16 %shl, %b16
  %div32 = sext i16 %div to i32
  ret i32 %div32
}
```

LLVM-IR —> SelectionDAG

```
define i32 @foo(i32 %a, i32 %b) #0 {
entry:
  %a16 = trunc i32 %a to i16
  %b16 = trunc i32 %b to i16
  %shl = shl i16 %a16, 3
  %div = sdiv i16 %shl, %b16
  %div32 = sext i16 %div to i32
  ret i32 %div32
}
```

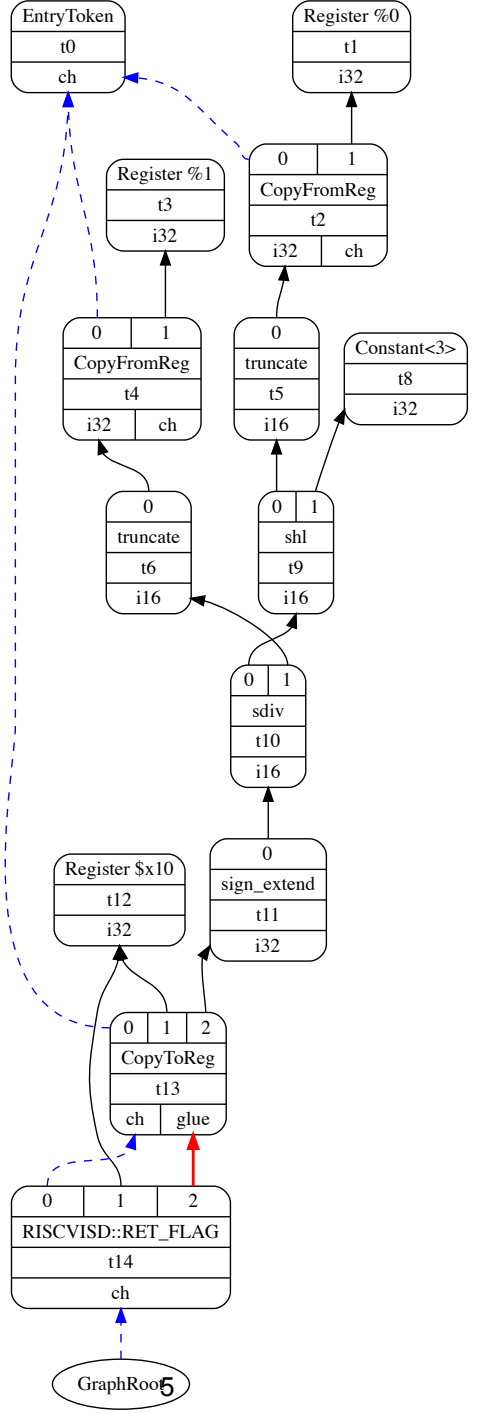


legalize-types input for foo:entry

LLVM-IR —> SelectionDAG

```
define i32 @foo(i32 %a, i32 %b) #0 {
entry:
  %a16 = trunc i32 %a to i16
  %b16 = trunc i32 %b to i16
  %shl = shl i16 %a16, 3
  %div = sdiv i16 %shl, %b16
  %div32 = sext i16 %div to i32
  ret i32 %div32
}
```

```
void SelectionDAGBuilder::visitSDiv(const User &I) {
  SDValue Op1 = getValue(I.getOperand(0));
  SDValue Op2 = getValue(I.getOperand(1));

  SDNodeFlags Flags;
  Flags.setExact(isa<PossiblyExactOperator>(&I) &&
                 cast<PossiblyExactOperator>(&I)->isExact());
  setValue(&I, DAG.getNode(ISD::SDIV, getCurSDLoc(),
        Op1.getValueType(), Op1, Op2, Flags));
}
```



legalize-types input for foo:entry

SelectionDAG

SelectionDAG

TECHNISCHE UNIVERSITÄT DARMSTADT

- SelectionDAG ist ein gerichteter, azyklischer Graph pro Basicblock

SelectionDAG

TECHNISCHE UNIVERSITÄT DARMSTADT

- SelectionDAG ist ein gerichteter, azyklischer Graph pro Basicblock
  - Knotentypen: viele (siehe rechts)

```
…
ADD, SUB, MUL, SDIV, UDIV, SREM, UREM,
SMUL_LOHI, UMUL_LOHI,
SDIVREM, UDIVREM,
ADDC, SUBC,
ADDE, SUBE,
ADDCARRY, SUBCARRY,
SADDO, UADDO,
SSUBO, USUBO,
SMULO, UMULO,
FADD, FSUB, FMUL, FDIV, FREM,
FMA, FMAD, FCOPYSIGN,
FGETSIGN, FCANONICALIZE,
BUILD_VECTOR, INSERT_VECTOR_ELT,
EXTRACT_VECTOR_ELT, CONCAT_VECTORS,
INSERT_SUBVECTOR, EXTRACT_SUBVECTOR,
VECTOR_SHUFFLE,
SCALAR_TO_VECTOR,
MULHU, MULHS,
SMIN, SMAX, UMIN, UMAX,
AND, OR, XOR,
ABS,
SHL, SRA, SRL, ROTL, ROTR,
BSWAP, CTTZ, CTLZ, CTPOP, BITREVERSE,
CTTZ_ZERO_UNDEF, CTLZ_ZERO_UNDEF,
SELECT,
…
```

SelectionDAG

TECHNISCHE UNIVERSITÄT DARMSTADT

- SelectionDAG ist ein gerichteter, azyklischer Graph pro Basicblock
  - Knotentypen: viele (siehe rechts)
    - Target kann weitere hinzufügen

```
…
ADD, SUB, MUL, SDIV, UDIV, SREM, UREM,
SMUL_LOHI, UMUL_LOHI,
SDIVREM, UDIVREM,
ADDC, SUBC,
ADDE, SUBE,
ADDCARRY, SUBCARRY,
SADDO, UADDO,
SSUBO, USUBO,
SMULO, UMULO,
FADD, FSUB, FMUL, FDIV, FREM,
FMA, FMAD, FCOPYSIGN,
FGETSIGN, FCANONICALIZE,
BUILD_VECTOR, INSERT_VECTOR_ELT,
EXTRACT_VECTOR_ELT, CONCAT_VECTORS,
INSERT_SUBVECTOR, EXTRACT_SUBVECTOR,
VECTOR_SHUFFLE,
SCALAR_TO_VECTOR,
MULHU, MULHS,
SMIN, SMAX, UMIN, UMAX,
AND, OR, XOR,
ABS,
SHL, SRA, SRL, ROTL, ROTR,
BSWAP, CTTZ, CTLZ, CTPOP, BITREVERSE,
CTTZ_ZERO_UNDEF, CTLZ_ZERO_UNDEF,
SELECT,
…
```

SelectionDAG

TECHNISCHE UNIVERSITÄT DARMSTADT

- SelectionDAG ist ein gerichteter, azyklischer Graph pro Basicblock
  - Knotentypen: viele (siehe rechts)
    - Target kann weitere hinzufügen
  - Kanten: Daten und „Chain“ zur Erhaltung der Ordnung

```
…
ADD, SUB, MUL, SDIV, UDIV, SREM, UREM,
SMUL_LOHI, UMUL_LOHI,
SDIVREM, UDIVREM,
ADDC, SUBC,
ADDE, SUBE,
ADDCARRY, SUBCARRY,
SADDO, UADDO,
SSUBO, USUBO,
SMULO, UMULO,
FADD, FSUB, FMUL, FDIV, FREM,
FMA, FMAD, FCOPYSIGN,
FGETSIGN, FCANONICALIZE,
BUILD_VECTOR, INSERT_VECTOR_ELT,
EXTRACT_VECTOR_ELT, CONCAT_VECTORS,
INSERT_SUBVECTOR, EXTRACT_SUBVECTOR,
VECTOR_SHUFFLE,
SCALAR_TO_VECTOR,
MULHU, MULHS,
SMIN, SMAX, UMIN, UMAX,
AND, OR, XOR,
ABS,
SHL, SRA, SRL, ROTL, ROTR,
BSWAP, CTTZ, CTLZ, CTPOP, BITREVERSE,
CTTZ_ZERO_UNDEF, CTLZ_ZERO_UNDEF,
SELECT,
…
```

SelectionDAG

TECHNISCHE UNIVERSITÄT DARMSTADT

- SelectionDAG ist ein gerichteter, azyklischer Graph pro Basicblock
  - Knotentypen: viele (siehe rechts)
    - Target kann weitere hinzufügen
  - Kanten: Daten und „Chain“ zur Erhaltung der Ordnung
  - Kein Target unterstützt alle Operationen/Datentypen

```
…
ADD, SUB, MUL, SDIV, UDIV, SREM, UREM,
SMUL_LOHI, UMUL_LOHI,
SDIVREM, UDIVREM,
ADDC, SUBC,
ADDE, SUBE,
ADDCARRY, SUBCARRY,
SADDO, UADDO,
SSUBO, USUBO,
SMULO, UMULO,
FADD, FSUB, FMUL, FDIV, FREM,
FMA, FMAD, FCOPYSIGN,
FGETSIGN, FCANONICALIZE,
BUILD_VECTOR, INSERT_VECTOR_ELT,
EXTRACT_VECTOR_ELT, CONCAT_VECTORS,
INSERT_SUBVECTOR, EXTRACT_SUBVECTOR,
VECTOR_SHUFFLE,
SCALAR_TO_VECTOR,
MULHU, MULHS,
SMIN, SMAX, UMIN, UMAX,
AND, OR, XOR,
ABS,
SHL, SRA, SRL, ROTL, ROTR,
BSWAP, CTTZ, CTLZ, CTPOP, BITREVERSE,
CTTZ_ZERO_UNDEF, CTLZ_ZERO_UNDEF,
SELECT,
…
```

SelectionDAG

TECHNISCHE UNIVERSITÄT DARMSTADT

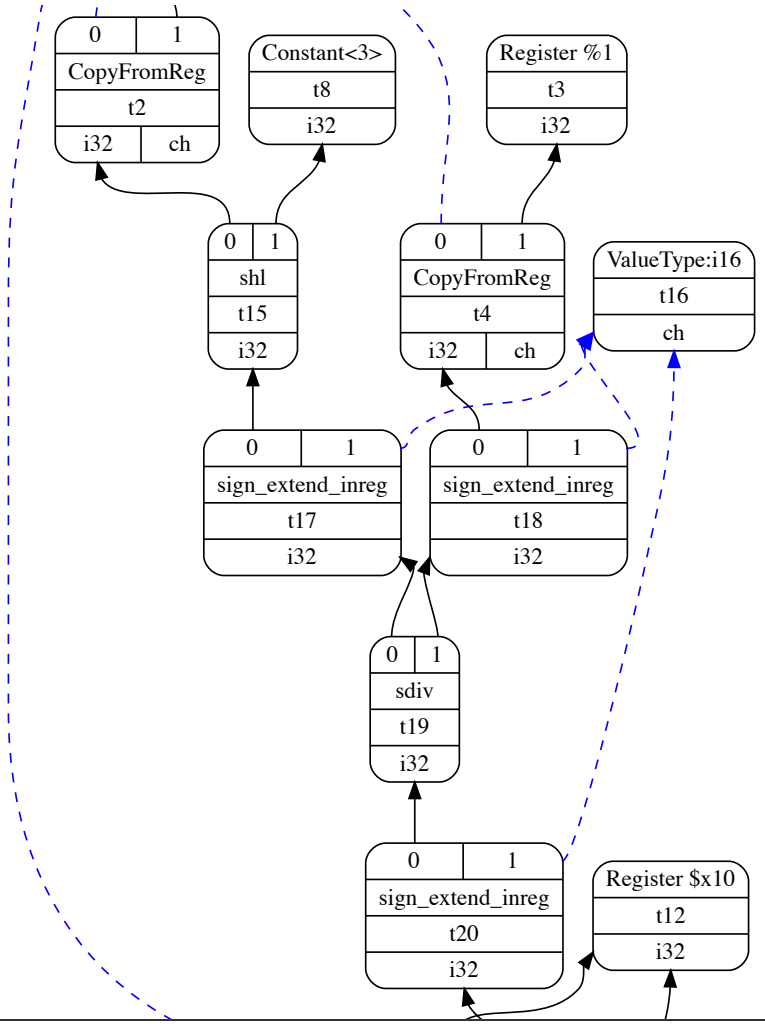
- SelectionDAG ist ein gerichteter, azyklischer Graph pro Basicblock
  - Knotentypen: viele (siehe rechts)
    - Target kann weitere hinzufügen
  - Kanten: Daten und „Chain“ zur Erhaltung der Ordnung
  - Kein Target unterstützt alle Operationen/Datentypen
    - Konzept des **legalen/illegalen** SelectionDAGs

```
…
ADD, SUB, MUL, SDIV, UDIV, SREM, UREM,
SMUL_LOHI, UMUL_LOHI,
SDIVREM, UDIVREM,
ADDC, SUBC,
ADDE, SUBE,
ADDCARRY, SUBCARRY,
SADDO, UADDO,
SSUBO, USUBO,
SMULO, UMULO,
FADD, FSUB, FMUL, FDIV, FREM,
FMA, FMAD, FCOPYSIGN,
FGETSIGN, FCANONICALIZE,
BUILD_VECTOR, INSERT_VECTOR_ELT,
EXTRACT_VECTOR_ELT, CONCAT_VECTORS,
INSERT_SUBVECTOR, EXTRACT_SUBVECTOR,
VECTOR_SHUFFLE,
SCALAR_TO_VECTOR,
MULHU, MULHS,
SMIN, SMAX, UMIN, UMAX,
AND, OR, XOR,
ABS,
SHL, SRA, SRL, ROTL, ROTR,
BSWAP, CTTZ, CTLZ, CTPOP, BITREVERSE,
CTTZ_ZERO_UNDEF, CTLZ_ZERO_UNDEF,
SELECT,
…
```

SelectionDAG - Typ-Legalisierung

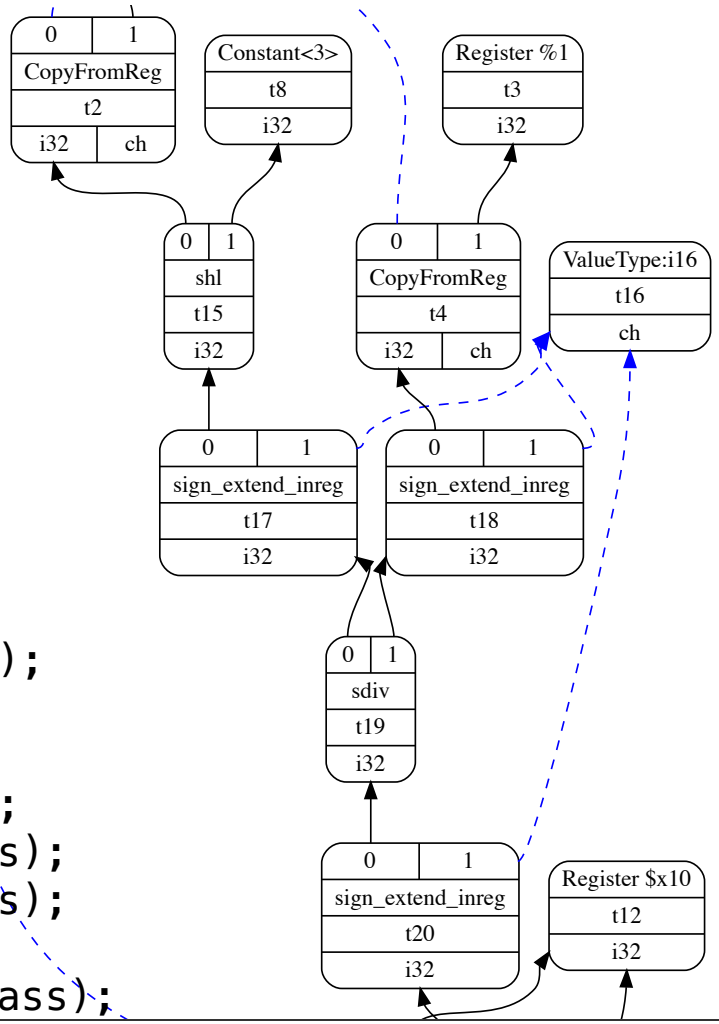
SelectionDAG - Typ-Legalisierung

- Typ-Legalisierung stellt sicher, dass nur noch vom Target unterstützte Datentypen vorkommen
  - Hier: nur 32-bit



SelectionDAG - Typ-Legalisierung

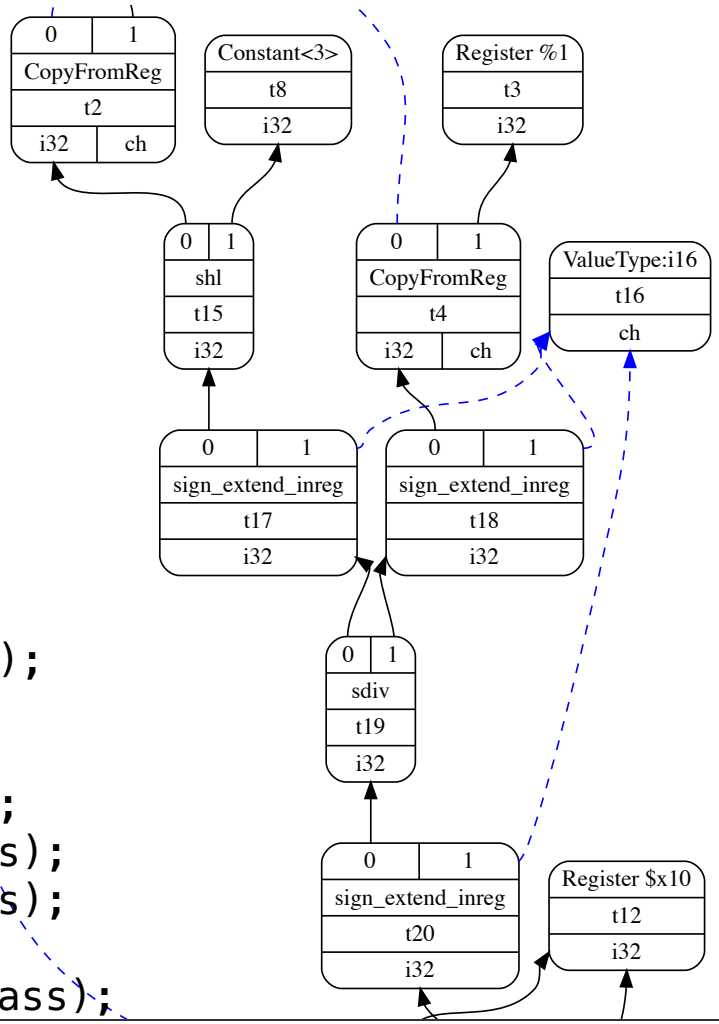
- Typ-Legalisierung stellt sicher, dass nur noch vom Target unterstützte Datentypen vorkommen
  - Hier: nur 32-bit




```
// RISCVISelLowering.cpp
addRegisterClass(XLenVT, &RISCV::GPRRegClass);

// zum Vergleich: X86ISelLowering.cpp
addRegisterClass(MVT::i8, &X86::GR8RegClass);
addRegisterClass(MVT::i16, &X86::GR16RegClass);
addRegisterClass(MVT::i32, &X86::GR32RegClass);
if (Subtarget.is64Bit())
  addRegisterClass(MVT::i64, &X86::GR64RegClass);
```

SelectionDAG - Typ-Legalisierung

- Typ-Legalisierung stellt sicher, dass nur noch vom Target unterstützte Datentypen vorkommen
  - Hier: nur 32-bit
- Nebenher laufen lokale Optimierungen auf SD (`shl` „vor“ `sign_extend`)




```
// RISCVISelLowering.cpp
addRegisterClass(XLenVT, &RISCV::GPRRegClass);

// zum Vergleich: X86ISelLowering.cpp
addRegisterClass(MVT::i8, &X86::GR8RegClass);
addRegisterClass(MVT::i16, &X86::GR16RegClass);
addRegisterClass(MVT::i32, &X86::GR32RegClass);
if (Subtarget.is64Bit())
  addRegisterClass(MVT::i64, &X86::GR64RegClass);
```

SelectionDAG - Legalisierung der Operationen

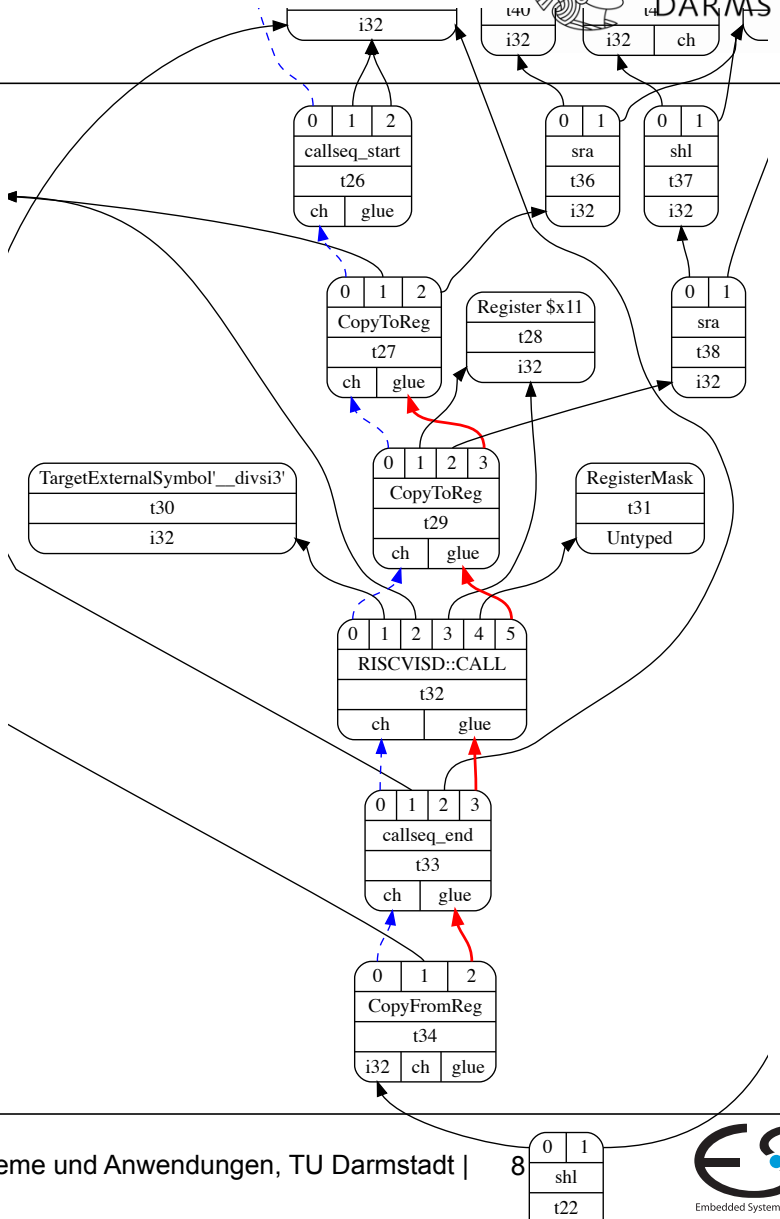
SelectionDAG - Legalisierung der Operationen

TECHNISCHE UNIVERSITÄT DARMSTADT

- RISC-V-Basis-ISA hat keine Multiplikation/Division
  - Legalisierung des SD durch Aufruf von Library-Funktion

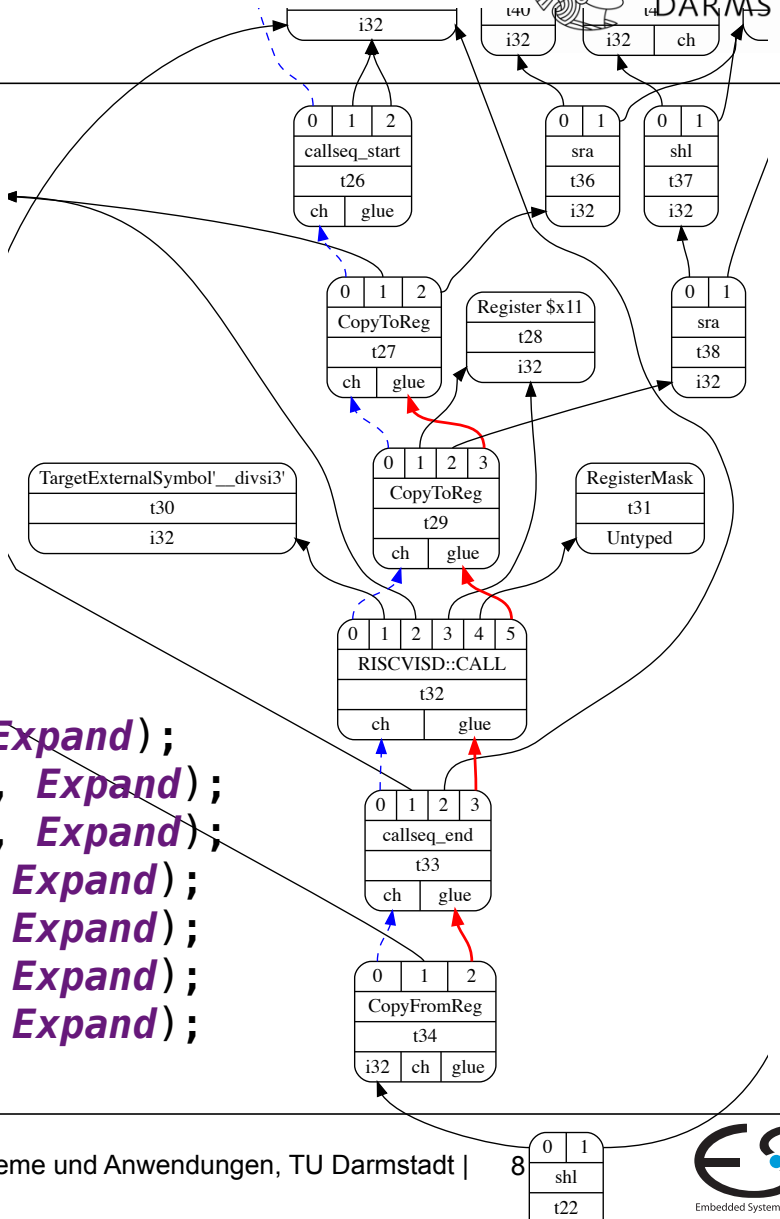
SelectionDAG - Legalisierung der Operationen

- RISC-V-Basis-ISA hat keine Multiplikation/Division
  - Legalisierung des SD durch Aufruf von Library-Funktion



SelectionDAG - Legalisierung der Operationen

- RISC-V-Basis-ISA hat keine Multiplikation/Division
  - Legalisierung des SD durch Aufruf von Library-Funktion

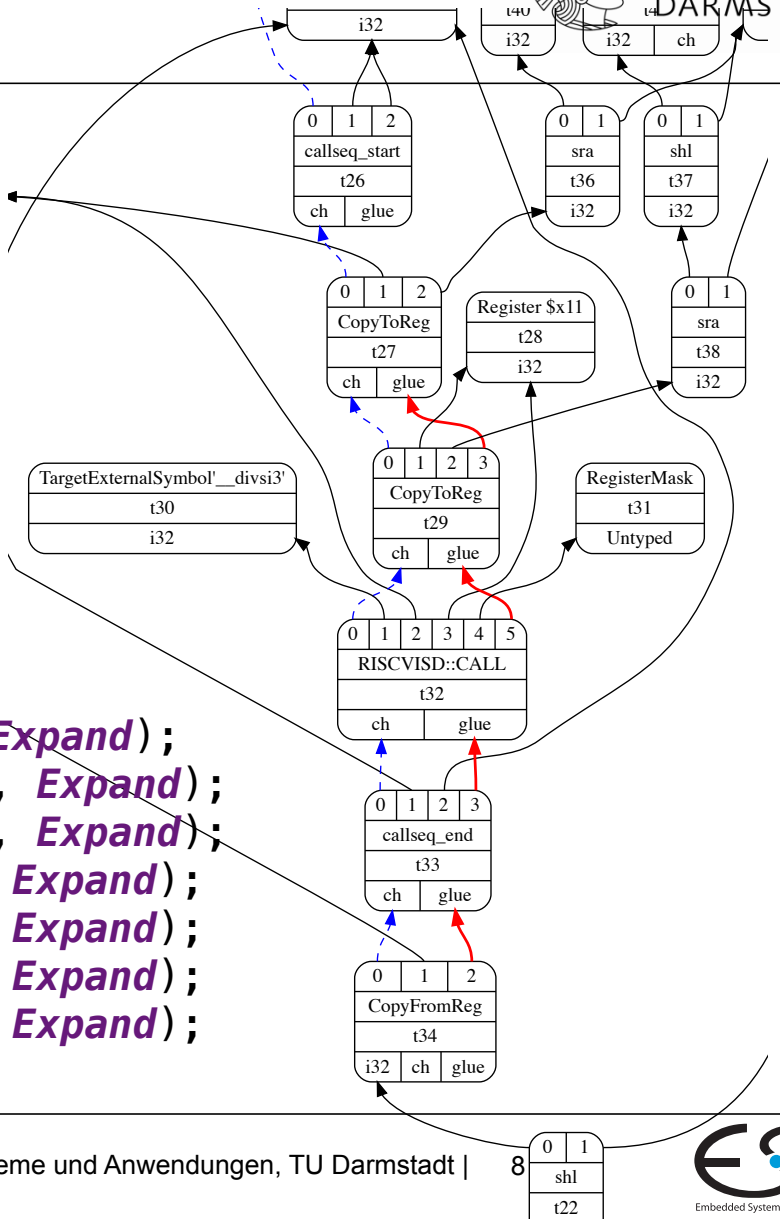



```
// RISCVISelLowering.cpp
if (!Subtarget.hasStdExtM()) {
  setOperationAction(ISD::MUL, XLenVT, Expand);
  setOperationAction(ISD::MULHS, XLenVT, Expand);
  setOperationAction(ISD::MULHU, XLenVT, Expand);
  setOperationAction(ISD::SDIV, XLenVT, Expand);
  setOperationAction(ISD::UDIV, XLenVT, Expand);
  setOperationAction(ISD::SREM, XLenVT, Expand);
  setOperationAction(ISD::UREM, XLenVT, Expand);
}
```

SelectionDAG - Legalisierung der Operationen

- RISC-V-Basis-ISA hat keine Multiplikation/Division
  - Legalisierung des SD durch Aufruf von Library-Funktion
- Custom-Lowering (durch C++-Code) möglich

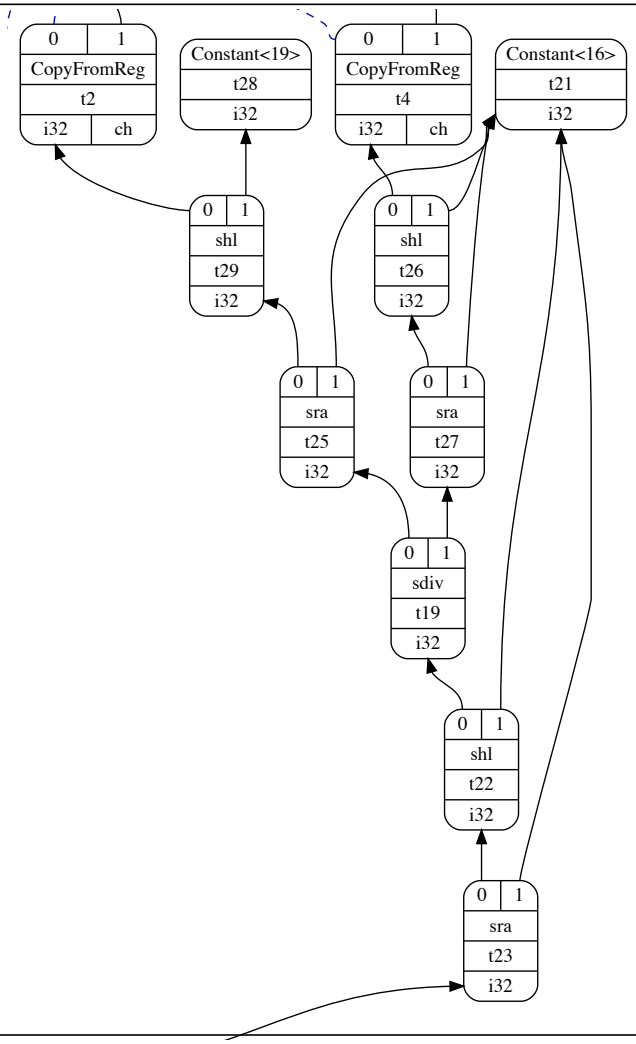
```
// RISCVISelLowering.cpp
if (!Subtarget.hasStdExtM()) {
  setOperationAction(ISD::MUL, XLenVT, Expand);
  setOperationAction(ISD::MULHS, XLenVT, Expand);
  setOperationAction(ISD::MULHU, XLenVT, Expand);
  setOperationAction(ISD::SDIV, XLenVT, Expand);
  setOperationAction(ISD::UDIV, XLenVT, Expand);
  setOperationAction(ISD::SREM, XLenVT, Expand);
  setOperationAction(ISD::UREM, XLenVT, Expand);
}
```



SelectionDAG - Legalisierung der Operationen

SelectionDAG - Legalisierung der Operationen

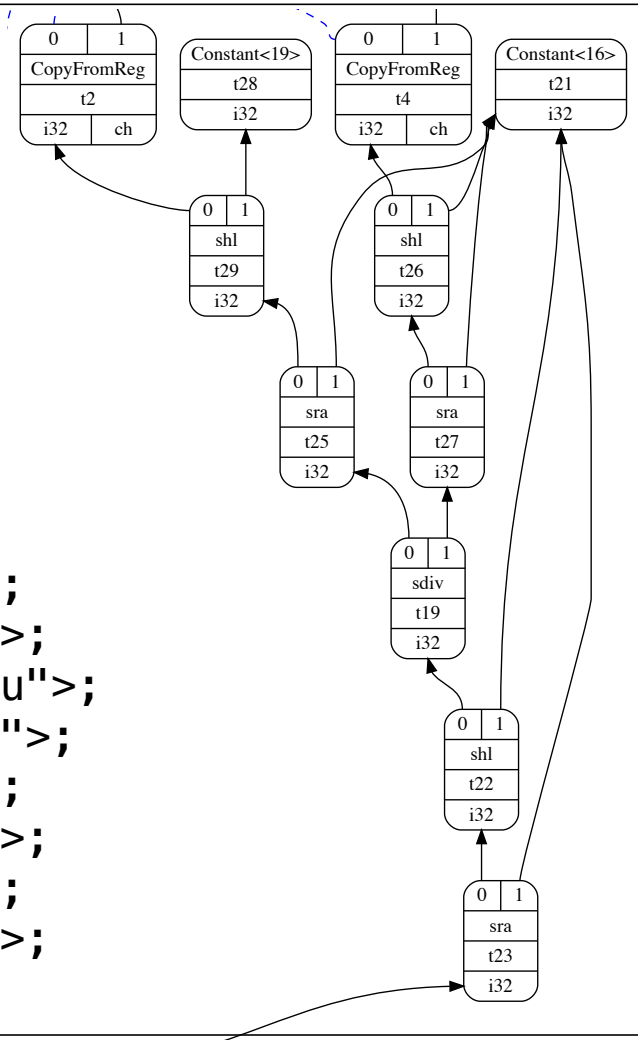
- RISC-V-ISA-Erweiterung „m“ stellt Multiplikation/Division bereit
- `sign_extend_inreg` wurde zu `shl`/`sra`-Paar



SelectionDAG - Legalisierung der Operationen

- RISC-V-ISA-Erweiterung „m“ stellt Multiplikation/Division bereit
- `sign_extend_inreg` wurde zu `shl`/`sra`-Paar

```
// RISCVInstrInfoM.td

let Predicates = [HasStdExtM] in {
def MUL     : ALU_rr<0b0000001, 0b000, "mul">;
def MULH    : ALU_rr<0b0000001, 0b001, "mulh">;
def MULHSU  : ALU_rr<0b0000001, 0b010, "mulhsu">;
def MULHU   : ALU_rr<0b0000001, 0b011, "mulhu">;
def DIV     : ALU_rr<0b0000001, 0b100, "div">;
def DIVU    : ALU_rr<0b0000001, 0b101, "divu">;
def REM     : ALU_rr<0b0000001, 0b110, "rem">;
def REMU    : ALU_rr<0b0000001, 0b111, "remu">;
} // Predicates = [HasStdExtM]
```



TableGen

TableGen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Targets müssen viele domänenspezifische Informationen für den Codegenerator zur Verfügung stellen
  - Instruktionen, Registersatz, …

TableGen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Targets müssen viele domänenspezifische Informationen für den Codegenerator zur Verfügung stellen
  - Instruktionen, Registersatz, …
- TableGen dient zur Beschreibung solcher **Records**

TableGen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Targets müssen viele domänenspezifische Informationen für den Codegenerator zur Verfügung stellen
  - Instruktionen, Registersatz, …
- TableGen dient zur Beschreibung solcher **Records**
- Ziel: Möglichst viele Gemeinsamkeiten ausfaktorisieren, um Beschreibung übersichtlich zu halten

TableGen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Targets müssen viele domänenspezifische Informationen für den Codegenerator zur Verfügung stellen
  - Instruktionen, Registersatz, …
- TableGen dient zur Beschreibung solcher **Records**
- Ziel: Möglichst viele Gemeinsamkeiten ausfaktorisieren, um Beschreibung übersichtlich zu halten
- Flow: TableGen-Tool liest .td-Datei, instanziiert Records, und gibt sie an domänenspezifisches Backend weiter
  - … häufig werden C++-Definitionen erzeugt, die anderswo inkludiert werden

TableGen: Beispiel

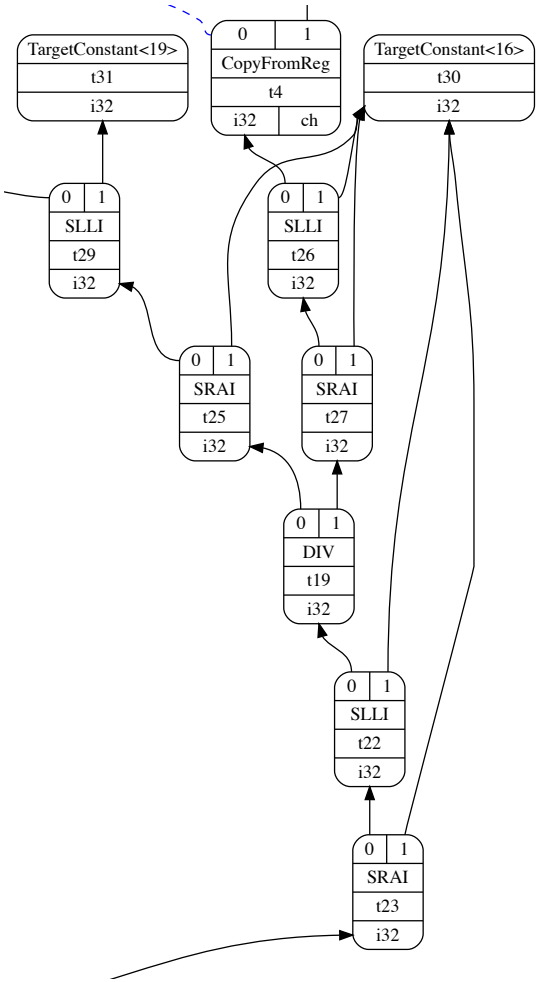
```
class klass<string Arg1, string Arg2> {
  string message =
    "hello" # Arg1 # " -> " # Arg2;
}
class klass2<int n> {
  int N = n;
}

def foo : klass<"world", „kthxbai">;

let message = "overriding..." in {
  def bar : klass<"doesnt", "matter">;
}
def baz :
  klass<"number", "see below“>,
  klass2<42>;

def blub {
  string here = "you go";
}
```

```
------------ Classes ----------------
class klass<string klass:Arg1 = ?, string klass
  string message = !strconcat("hello", !strconc
klass:Arg2)));
}
class klass2<int klass2:n = ?> {
  int N = klass2:n;
}
------------ Defs -----------------
def bar {// klass
  string message = "overriding...";
}
def baz {// klass klass2
  string message = "hellonumber -> see below";
  int N = 42;
}
def blub {
  string here = "you go";
}
def foo {// klass
  string message = "helloworld -> kthxbai";
}
```

SelectionDAG: Instruction Selection

SelectionDAG: Instruction Selection

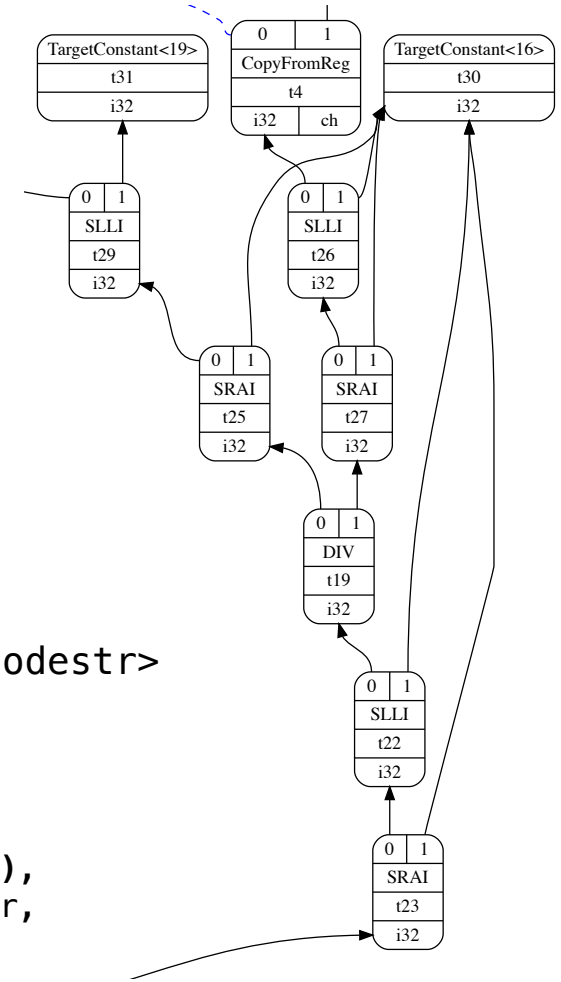
- Mapping von SDNodes auf Opcodes des Targets



SelectionDAG: Instruction Selection

TECHNISCHE UNIVERSITÄT DARMSTADT

- Mapping von SDNodes auf Opcodes des Targets
- Pattern Matching wird automatisch aus TableGen-Beschreibung des Instruktionssatzes generiert




```
def SLLI : Shift_ri<0, 0b001, "slli">;
def SRAI : Shift_ri<1, 0b101, "srai">;

def DIV     : ALU_rr<0b0000001, 0b100, "div">;

class ALU_rr<bits<7> funct7, bits<3> funct3, string opcodestr>
    : RVInstR<funct7, funct3, OPC_OP,
              (outs GPR:$rd), (ins GPR:$rs1, GPR:$rs2),
               opcodestr, "$rd, $rs1, $rs2“>;
class Shift_ri<bit arithshift, bits<3> funct3, string opcodestr>
    : RVInstIShift<arithshift, funct3, OPC_OP_IMM, (outs GPR:$rd),
                   (ins GPR:$rs1, uimmlog2xlen:$shamt), opcodestr,
                   "$rd, $rs1, $shamt">;
```


Embedded Systems & Applications

SelectionDAG —> Machine IR

TECHNISCHE UNIVERSITÄT DARMSTADT

- Machine IR besteht wieder aus (Machine-)Functions, BasicBlock, Instructions
- Scheduler berechnet Ordnung der MachineInstructions
- Code verwendet virtuelle Register, ist weiterhin in SSA-Form

```
# Machine code for function foo: IsSSA,
# TracksLiveness
Function Live Ins: $x10 in %0, $x11 in %1

bb.0.entry:
  liveins: $x10, $x11
  %1:gpr = COPY $x11
  %0:gpr = COPY $x10
  %2:gpr = SLLI %1:gpr, 16
  %3:gpr = SRAI killed %2:gpr, 16
  %4:gpr = SLLI %0:gpr, 19
  %5:gpr = SRAI killed %4:gpr, 16
  %6:gpr = DIV killed %5:gpr, killed %3:gpr
  %7:gpr = SLLI killed %6:gpr, 16
  %8:gpr = SRAI killed %7:gpr, 16
  $x10 = COPY %8:gpr
  PseudoRET implicit $x10

# End machine code for function foo.
```

Machine IR

Machine IR

TECHNISCHE UNIVERSITÄT DARMSTADT

- Machine IR wird nochmal ausgiebig optimiert:

```
Expand ISel Pseudo-instructions:
Early Tail Duplication:
Optimize machine instruction PHIs:
Merge disjoint stack slots:
Local Stack Slot Allocation:
Remove dead machine instructions:
Early Machine Loop Invariant Code Motion:
Machine Common Subexpression Elimination:
Machine code sinking:
Peephole Optimizations:
Remove dead machine instructions:
Detect Dead Lanes:
Process Implicit Definitions:
Live Variable Analysis:
Machine Natural Loop Construction:
Eliminate PHI nodes for register allocation:
Two-Address instruction pass:
Simple Register Coalescing:
Rename Disconnected Subregister Components:
Machine Instruction Scheduler:
Greedy Register Allocator:
Virtual Register Rewriter:
Stack Slot Coloring:
Machine Copy Propagation Pass:
Machine Loop Invariant Code Motion:
PostRA Machine Sink:
Shrink Wrapping analysis:
Prologue/Epilogue Insertion & Frame Finalization:
Control Flow Optimizer:
Tail Duplication:
Machine Copy Propagation Pass:
Post-RA pseudo instruction expansion pass:
Post RA top-down list latency scheduler:
Analyze Machine Code For Garbage Collection:
Branch Probability Basic Block Placement:
Branch relaxation pass:
Contiguously Lay Out Funclets:
StackMap Liveness Analysis:
Live DEBUG_VALUE analysis:
Insert fentry calls:
Insert XRay ops:
Implement the 'patchable-function' attribute:
```

Machine IR

TECHNISCHE UNIVERSITÄT DARMSTADT

- Machine IR wird nochmal ausgiebig optimiert:
- Routinenprotokoll, Registerallokation, …

```
# Machine code for function foo: NoPHIs, TracksLiveness, NoVRegs
Frame Objects:
  fi#0: size=4, align=4, at location [SP-4]
  fi#1: size=4, align=4, at location [SP-8]
Function Live Ins: $x10, $x11

bb.0.entry:
  liveins: $x10, $x11
  $x2 = frame-setup ADDI $x2, -16
  SW killed $x1, $x2, 12
  SW killed $x8, $x2, 8
  $x8 = frame-setup ADDI $x2, 16
  renamable $x11 = SLLI killed renamable $x11, 16
  renamable $x11 = SRAI killed renamable $x11, 16
  renamable $x10 = SLLI killed renamable $x10, 19
  renamable $x10 = SRAI killed renamable $x10, 16
  renamable $x10 = DIV killed renamable $x10, killed renamable $x11
  renamable $x10 = SLLI killed renamable $x10, 16
  renamable $x10 = SRAI killed renamable $x10, 16
  $x8 = LW $x2, 8
  $x1 = LW $x2, 12
  $x2 = frame-destroy ADDI $x2, 16
  PseudoRET implicit $x10
```

```
Expand ISel Pseudo-instructions:
Early Tail Duplication:
Optimize machine instruction PHIs:
Merge disjoint stack slots:
Local Stack Slot Allocation:
Remove dead machine instructions:
Early Machine Loop Invariant Code Motion:
Machine Common Subexpression Elimination:
Machine code sinking:
Peephole Optimizations:
Remove dead machine instructions:
Detect Dead Lanes:
Process Implicit Definitions:
Live Variable Analysis:
Machine Natural Loop Construction:
Eliminate PHI nodes for register allocation:
Two-Address instruction pass:
Simple Register Coalescing:
Rename Disconnected Subregister Components:
Machine Instruction Scheduler:
Greedy Register Allocator:
Virtual Register Rewriter:
Stack Slot Coloring:
Machine Copy Propagation Pass:
Machine Loop Invariant Code Motion:
PostRA Machine Sink:
Shrink Wrapping analysis:
Prologue/Epilogue Insertion & Frame Finalization:
Control Flow Optimizer:
Tail Duplication:
Machine Copy Propagation Pass:
Post-RA pseudo instruction expansion pass:
Post RA top-down list latency scheduler:
Analyze Machine Code For Garbage Collection:
Branch Probability Basic Block Placement:
Branch relaxation pass:
Contiguously Lay Out Funclets:
StackMap Liveness Analysis:
Live DEBUG_VALUE analysis:
Insert fentry calls:
Insert XRay ops:
Implement the 'patchable-function' attribute:
```

Machine IR —> MC Layer

Machine IR —> MC Layer

TECHNISCHE UNIVERSITÄT DARMSTADT

- Abstraktionsebene: Sections, Labels, Instructions, …

Machine IR —> MC Layer

TECHNISCHE UNIVERSITÄT DARMSTADT

- Abstraktionsebene: Sections, Labels, Instructions, …
- Ausgabe entweder als Assemblercode oder binär

```
        .text
        .file   "test.c"
        .globl  foo
        .p2align 2
        .type   foo,@function
foo:
        addi    sp, sp, -16             # encoding: [0x13,0x01,0x01,0xff]
        sw      ra, 12(sp)              # encoding: [0x23,0x26,0x11,0x00]
        sw      s0, 8(sp)               # encoding: [0x23,0x24,0x81,0x00]
        addi    s0, sp, 16              # encoding: [0x13,0x04,0x01,0x01]
        slli    a1, a1, 16              # encoding: [0x93,0x95,0x05,0x01]
        srai    a1, a1, 16              # encoding: [0x93,0xd5,0x05,0x41]
        slli    a0, a0, 19              # encoding: [0x13,0x15,0x35,0x01]
        srai    a0, a0, 16              # encoding: [0x13,0x55,0x05,0x41]
        div     a0, a0, a1              # encoding: [0x33,0x45,0xb5,0x02]
        slli    a0, a0, 16              # encoding: [0x13,0x15,0x05,0x01]
        srai    a0, a0, 16              # encoding: [0x13,0x55,0x05,0x41]
        lw      s0, 8(sp)               # encoding: [0x03,0x24,0x81,0x00]
        lw      ra, 12(sp)              # encoding: [0x83,0x20,0xc1,0x00]
        addi    sp, sp, 16              # encoding: [0x13,0x01,0x01,0x01]
        ret                            # encoding: [0x67,0x80,0x00,0x00]
```

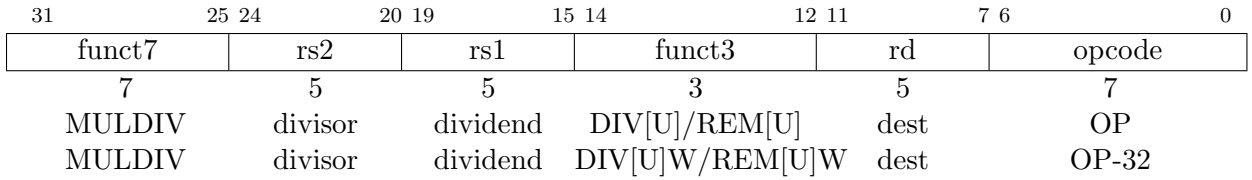

Embedded Systems & Applications

Machine IR —> MC Layer

TECHNISCHE UNIVERSITÄT DARMSTADT

- Details des Instruktionsformats sind als TableGen-Records kodiert
- Assembler-Darstellung ebenfalls

```
def DIV     : ALU_rr<0b0000001, 0b100, "div">;

class ALU_rr<bits<7> funct7, bits<3> funct3, string opcodestr>
    : RVInstR<funct7, funct3, OPC_OP,
              (outs GPR:$rd), (ins GPR:$rs1, GPR:$rs2),
              opcodestr, "$rd, $rs1, $rs2“>;

class RVInstR<bits<7> funct7, bits<3> funct3, RISCVOpcode opcode, dag outs,
              dag ins, string opcodestr, string argstr>
    : RVInst<outs, ins, opcodestr, argstr, [], InstFormatR> {
  bits<5> rs2;
  bits<5> rs1;
  bits<5> rd;

  let Inst{31-25} = funct7;
  let Inst{24-20} = rs2;
  let Inst{19-15} = rs1;
  let Inst{14-12} = funct3;
  let Inst{11-7} = rd;
  let Opcode = opcode.Value;
}
```



31–25	24–20	19–15	14–12	11–7	6–0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
MULDIV	divisor	dividend	DIV[U]/REM[U]	dest	OP
MULDIV	divisor	dividend	DIV[U]W/REM[U]W	dest	OP-32

Embedded Systems & Applications

Quellen / weiterführende Infos

TECHNISCHE UNIVERSITÄT DARMSTADT

- https://www.llvm.org/docs/CodeGenerator.html
- https://www.llvm.org/docs/WritingAnLLVMBackend.html
- https://riscv.org/specifications/
- https://eli.thegreenplace.net/2012/11/24/life-of-an-instruction-in-llvm/
- https://content.riscv.org/wp-content/uploads/2018/05/asb_llvm_2018_barcelona_riscv_workshop_slides.pdf
- https://git.llvm.org/git/llvm.git