Intermediate Representation Code Generation

Lorenzo Ceragioli

November 20, 2024

IMT Lucca

MiniRISC

Language for our Next Steps

MiniRISC: A simplified RISC Assembly for programming our abstract machine.

- Simple operations over integers and registers
- commands for reading and writing values on the memory

We assume that everything is an integer:

- integers
- boolean values
- memory addresses

We also assume an infinite amount of registers (for the language)

The Role of MiniRISC for our Project

We use MiniRISC as

- Our **target language**
- Our **intermediate representation** via its control-flow graph

The other difference between the intermediate representation and the target language is the run-time environment:

- In the IR we will assume an infinite amount of registers
- For the target code, they will be limited

A RISC Assembly

MiniRISC program: labelled blocks (lists of instructions)

MiniRISC instructions:

$$
\begin{aligned}
comm &\coloneqq \texttt{nop} \mid brop\ r\ r\ \texttt{=>}\ r \mid biop\ r\ n\ \texttt{=>}\ r \mid urop\ r\ \texttt{=>}\ r \\
&\quad \mid \texttt{load}\ r\ \texttt{=>}\ r \mid \texttt{loadI}\ n\ \texttt{=>}\ r \mid \texttt{store}\ r\ \texttt{=>}\ r \\
&\quad \mid \texttt{jump}\ l \mid \texttt{cjump}\ r\ l\ l \\
brop &\coloneqq \texttt{add} \mid \texttt{sub} \mid \texttt{mult} \mid \texttt{and} \mid \texttt{less} \\
biop &\coloneqq \texttt{addI} \mid \texttt{subI} \mid \texttt{multI} \mid \texttt{andI} \\
urop &\coloneqq \texttt{not} \mid \texttt{copy}
\end{aligned}
$$

where l is a label, r is a register, n is an integer

RISC Architecture

We have two memories

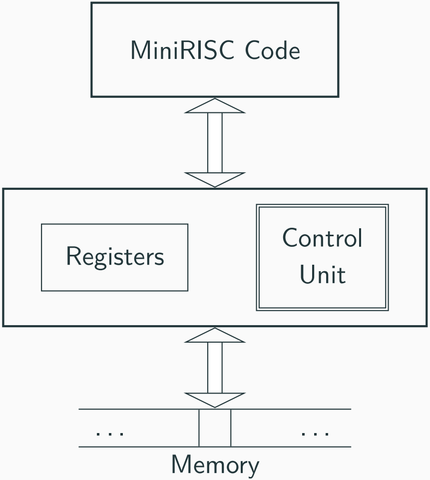
- Registers — $\sigma_R : R \longrightarrow \mathbb{Z}$
- RAM — $\sigma_M : \mathbb{Z} \longrightarrow \mathbb{Z}$

(R are registers)

We also assume a function

- Code — $\xi : L \longrightarrow C^*$
- Special label: `main` (where the computation starts)
- Special registers
  - `in` – for user input
  - `out` – for user output

(L are labels, C are commands)



Semantics

Program

$$\frac{\langle \xi, \xi(\mathit{main}), \sigma_R[\mathit{in} \mapsto \mathit{input}], \sigma_M \rangle \longrightarrow^* \langle \xi, \epsilon, \sigma'_R, \sigma'_M \rangle}{\langle \xi, \sigma_R, \sigma_M, \mathit{input} \rangle \longrightarrow \sigma'_R(\mathit{out})}$$

Small-Step Semantics of Commands

$$\frac{}{\langle \xi, \texttt{nop} \cdot b, \sigma_R, \sigma_M \rangle \longrightarrow \langle \xi, b, \sigma_R, \sigma_M \rangle}\ \textit{nop}$$

$$\frac{n = \sigma_R(r_1)\ op\ \sigma_R(r_2)}{\langle \xi, (\textit{brop}\ \ r_1\ \ r_2\ \ \texttt{=>}\ \ r_3) \cdot b, \sigma_R, \sigma_M \rangle \longrightarrow \langle \xi, b, \sigma_R[r_3 \mapsto n], \sigma_M \rangle}\ \textit{brop}$$

Where *op* is the operator corresponding to *brop*

- $\texttt{add}, \texttt{sub}, \texttt{mult}$ are as expected
- `and` and `less` require encoding boolean values as integers
  - 0 for `false`
  - 1 for `true`

Small-Step Semantics of Commands

$$\frac{n' = \sigma_R(r_1)\ op\ n}{\langle \xi, (biop\ r_1\ n\ \texttt{=>}\ r_2) \cdot b, \sigma_R, \sigma_M \rangle \longrightarrow \langle \xi, b, \sigma_R[r_2 \mapsto n'], \sigma_M \rangle}\ biop$$

$$\frac{n = \sigma_R(r_1)}{\langle \xi, (\texttt{copy}\ r_1\ \texttt{=>}\ r_2) \cdot b, \sigma_R, \sigma_M \rangle \longrightarrow \langle \xi, b, \sigma_R[r_2 \mapsto n], \sigma_M \rangle}\ copy$$

$$\frac{n = not(\sigma_R(r_1))}{\langle \xi, (\texttt{not}\ r_1\ \texttt{=>}\ r_2) \cdot b, \sigma_R, \sigma_M \rangle \longrightarrow \langle \xi, b, \sigma_R[r_2 \mapsto n], \sigma_M \rangle}\ not$$

Where

- `addI`, `subI`, `multI` are as expected
- `andI` requires encoding boolean values as integers

Small-Step Semantics of Commands

$$\frac{n = \sigma_M(\sigma_R(r_1))}{\langle \xi, (\texttt{load}\ r_1\ \texttt{=>}\ r_2) \cdot b, \sigma_R, \sigma_M \rangle \longrightarrow \langle \xi, b, \sigma_R[r_2 \mapsto n], \sigma_M \rangle}\ \textit{load}$$

$$\frac{}{\langle \xi, (\texttt{loadI}\ n\ \texttt{=>}\ r) \cdot b, \sigma_R, \sigma_M \rangle \longrightarrow \langle \xi, b, \sigma_R[r \mapsto n], \sigma_M \rangle}\ \textit{loadI}$$

$$\frac{n = \sigma_R(r_1) \qquad n' = \sigma_R(r_2)}{\langle \xi, (\texttt{store}\ r_1\ \texttt{=>}\ r_2) \cdot b, \sigma_R, \sigma_M \rangle \longrightarrow \langle \xi, b, \sigma_R, \sigma_M[n' \mapsto n] \rangle}\ \textit{store}$$

Small-Step Semantics of Commands

$$\frac{}{\langle \xi, (\texttt{jump}\ l) \cdot b, \sigma_R, \sigma_M \rangle \longrightarrow \langle \xi, \xi(l), \sigma_R, \sigma_M \rangle}\ \textit{jump}$$

$$\frac{\sigma_R(r) = 1}{\langle \xi, (\texttt{cjump}\ r\ l\ l') \cdot b, \sigma_R, \sigma_M \rangle \longrightarrow \langle \xi, \xi(l), \sigma_R, \sigma_M \rangle}\ \textit{cjumpt}$$

$$\frac{\sigma_R(r) = 0}{\langle \xi, (\texttt{cjump}\ r\ l\ l') \cdot b, \sigma_R, \sigma_M \rangle \longrightarrow \langle \xi, \xi(l'), \sigma_R, \sigma_M \rangle}\ \textit{cjumpf}$$

Generating Intermediate Code

Intermediate Representation

MiniRISC simple statements

$$
\begin{aligned}
scomm &\coloneqq \texttt{nop} \mid brop\ r\ r\ \texttt{=>}\ r \mid biop\ r\ n\ \texttt{=>}\ r \mid urop\ r\ \texttt{=>}\ r \\
&\quad \mid \texttt{load}\ r\ \texttt{=>}\ r \mid \texttt{loadI}\ n\ \texttt{=>}\ r \mid \texttt{store}\ r\ \texttt{=>}\ r \\
brop &\coloneqq \texttt{add} \mid \texttt{sub} \mid \texttt{mult} \mid \texttt{and} \mid \texttt{less} \\
biop &\coloneqq \texttt{addI} \mid \texttt{subI} \mid \texttt{multI} \mid \texttt{andI} \\
urop &\coloneqq \texttt{not} \mid \texttt{copy}
\end{aligned}
$$

Recall

- IR is the control-flow graph of our RISC Assembly
- blocks are lists of MiniRISC simple statements
- of course labels are associated to blocks!

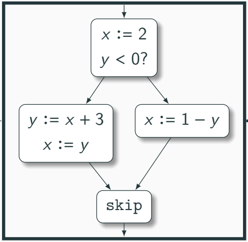
Recap: Compiling MiniImp

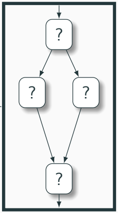
MiniImp Code

MiniImp CFG
(input y, output x)

MiniRISC CFG

```
def main with input y output x as
  x := 2;
  if y < 0 then (
    y := x + 3;
    x := y
  )
  else
    x := 1 - y
```



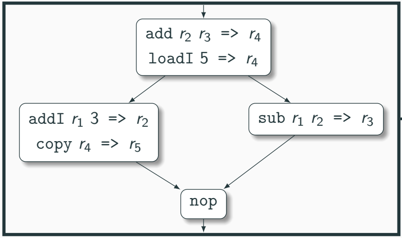


Generating MiniRISC Simple Statements

- Map variables to registers
- (mind the input and output registers and variables)
- We need other additional registers
  - intermediate values for computing arithmetical expressions
  - result of boolean guards *b*? compilation
- Compile a MiniImp simple statement into a list of MiniRISC simple statements
  - `skip` $\mapsto$ [`nop`]
  - *x* := *aexp* $\mapsto$ [compute *aexp* and write it in the register for x]

Generating MiniRISC Code

MiniRISC CFG



MiniRISC Code

```
main:  add r2 r3 => r4
       loadI 5 => r4
       cjump r4 l1 l2
  l1:  addI r1 3 => r2
       copy r4 => r5
       jump l3
  l2:  sub r1 r2 => r3
       jump l3
  l3:  nop
```

Generating MiniRISC Code

Idea:

- Associate a label to each block
- Transform transitions into jumps

Note:

- We will need the CFG for static analysis
- The target language is MiniRISC, but we will have constraints on the architecture

Project Fragment

- Write a module for MiniRISC (syntax and simple statements, the semantics is not required)
- Implement a translation from MiniImp CFG to MiniRISC CFG
- Implement a translation from MiniRISC CFG to MiniRISC
- Detail your translations in the report