Target Code Generation

Lorenzo Ceragioli

December 4, 2024

IMT Lucca

Recall: MiniRISC

MiniRISC program: labelled blocks (lists of instructions)

MiniRISC instructions:

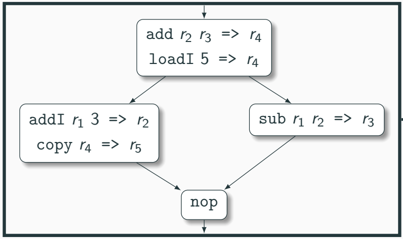
$$
\begin{aligned}
\textit{comm} &\coloneqq \texttt{nop} \mid \textit{brop}\ r\ r\ \texttt{=>}\ r \mid \textit{biop}\ r\ n\ \texttt{=>}\ r \mid \textit{urop}\ r\ \texttt{=>}\ r \\
&\quad \mid \texttt{load}\ r\ \texttt{=>}\ r \mid \texttt{loadI}\ n\ \texttt{=>}\ r \mid \texttt{store}\ r\ \texttt{=>}\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

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

Generating MiniRISC Code

MiniRISC CFG

MiniRISC Code
with limited registers



```
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
```

Pipeline

Assume $n \geq 4$ is the amount of registers in the target machine

1) MiniRISC CFG with m registers

   $\longrightarrow$ reduce registers by using memory $\longrightarrow$

2) MiniRISC CFG with n registers

   $\longrightarrow$ map control-flow edges to jumps $\longrightarrow$

3) MiniRISC with n registers

Reducing Registers – a simple approach

Note: we need at least 2 free registers for temporary values: r_A, r_B

Start with m registers, reduce to $n - 2$ registers:

- Choose $n - 2$ registers to keep

  (You can choose some heuristic, how to deal with *in* and *out*?)
- Create a mapping *addr* from register names to memory addresses

  (for the remaining $m - n + 2$ registers)
- For each operation on registers that are stored in memory add load and store instructions

Depending on your translation from MiniImp to MiniRISC you may have to consider some specific cases.

Using Memory to Store Register Values

Example1: `add` r_1 r_2 `=>` r_3 where only r_1 is in memory

`loadI` $addr(r_1)$ `=>` r_A ← we put the address for r_1 in r_A

`load` r_A `=>` r_A ← we put the value of r_1 in r_A

`add` r_A r_2 `=>` r_3 ← we compute the result

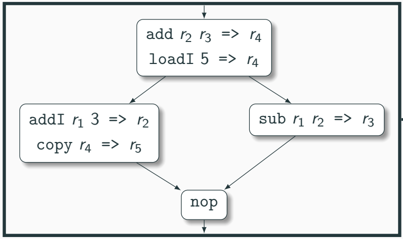
Using Memory to Store Register Values

Example2: `add` r_1 r_2 `=>` r_3 where all registers are in memory

`loadI` $addr(r_1)$ `=>` r_A	$\leftarrow$ we put the address for r_1 in r_A
`load` r_A `=>` r_A	$\leftarrow$ we put the value of r_1 in r_A
`loadI` $addr(r_2)$ `=>` r_B	$\leftarrow$ we do the same for r_2 with r_B
`load` r_B `=>` r_B	$\leftarrow$ ...
`add` r_A r_B `=>` r_B	$\leftarrow$ we compute the result
`loadI` $addr(r_3)$ `=>` r_A	$\leftarrow$ we put the address for r_3 in r_A
`store` r_B `=>` r_A	$\leftarrow$ we put the result in the address for r_3

Final Step: Just Insert Jump Instructions

MiniRISC CFG
with limited registers



MiniRISC Code
with limited registers

```
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
```

Optimization via Live Registers

Assume $n \geq 4$ is the amount of registers in the target machine

1) MiniRISC CFG with m registers

$\longrightarrow$ reduce registers by using memory $\longrightarrow$

2) MiniRISC CFG with n registers

$\longrightarrow$ map control-flow edges to jumps $\longrightarrow$

3) MiniRISC with n registers

Optimization via Live Registers

Assume $n \geq 4$ is the amount of registers in the target machine

1) MiniRISC CFG with m registers

   $\longrightarrow$ reduce registers by merging them $\longrightarrow$

1.b) MiniRISC CFG with $m' \leq m$ registers

   $\longrightarrow$ reduce registers by using memory $\longrightarrow$

2) MiniRISC CFG with n registers

   $\longrightarrow$ map control-flow edges to jumps $\longrightarrow$

3) MiniRISC with n registers

Merging Registers that are Never Live Together

Recall: live analysis stores for each block

- the live variables when entering the block
- the live variables when leaving the block
- for each register, we can derive its *live range*!

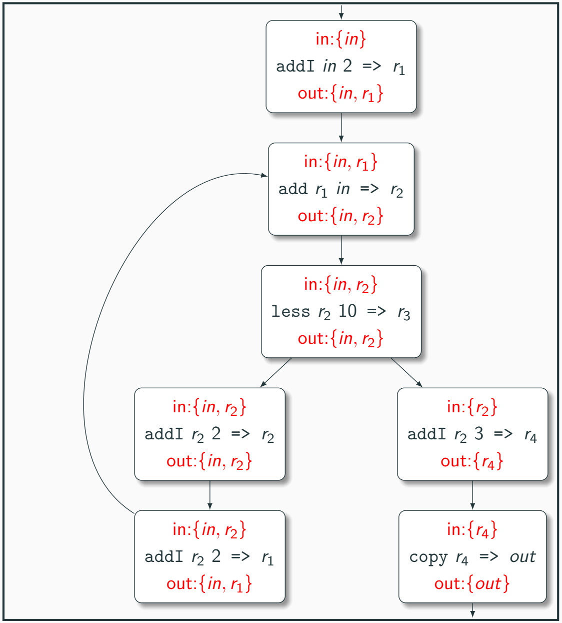
Definitions:

- r is live in the edge $l \longrightarrow l'$ if it is live when entering l'
- The live range of r is the set of edges in which r is live

Idea: We can merge registers for whom live ranges have empty intersection!

- Compute set of registers to merge
- Choose a name
- Update occurrences in the code

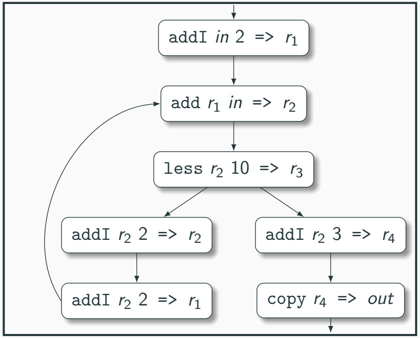
Live Variables



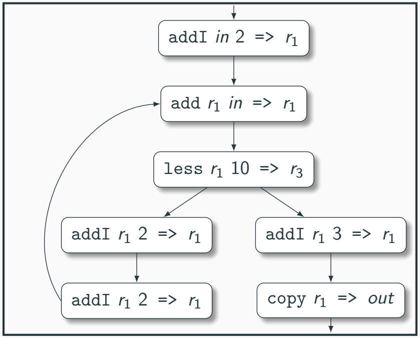
- r_1, r_2, r_4 can be merged!
- also *out*, but take care of the output
- r_3 depends on your choice for encoding conditions (ignore here)

Merging Registers

From:



To:



Project Fragment

- Implement a translation from MiniRISC CFG to MiniRISC for a target architecture: the number of registers must be an integer parameter (must work for $n \geq 4$)
- Implement an optimization procedure from MiniRISC CFG to MiniRISC CFG that tries to reduce the number of registers exploiting the liveness analysis
- Detail your translations and analysis in the report

Project Fragment

- Wrap everything in a program (using the needed modules) that defines a MiniImp to MiniRISC compiler: the compiler takes three inputs
    - the number of registers available in the target machine
    - the MiniImp program file
    - the path where to write the MiniRISC code
- and has two options (also input or optional input)
    - an option to activate/deactivate the check for undefined variables
      (if the variable is really undefined the compiler can fail)
    - an option to activate/deactivate the optimization
- Detail your implementation and how to use the program in the report

Project Recap!

- The final objective is to produce:
  - A MiniFun interpreter (parser included for this or the next one)
  - A MiniTyFun interpreter (parser included for this or the previous one)
  - A MiniImp interpreter (parser included)
  - A MiniImp compiler for machines with any number of registers (≥ 4), enriched with
    - a static analysis for checking that variables are initialized
    - an optimizer that reduce the number of used registers
- You will need to implement the modules seen so far
- Recall to explain the structure of your code in the report

Submission

- When: anytime from now
    - recall that a complete well written project is already 30L
    - ... if something is missing it is not the end of the world
    - code efficiency and further optimizations for extra points, if done in a reasonable way (and well explained in the report)
- What: archive and pdf
    - create an archive for the code
    - create a pdf for the report
- How: via email
    - put your name in the email
    - attach the archive and pdf
    - send email at lorenzo.ceragioli@imtlucca.it with Subject ”Project Submission”
    - I will confirm the reception, if I do not, write me!
    - Give me all the instructions for compiling and running your code

Next Lessons

We missed two lectures, they will be recovered:

- **Friday 13: 9-11** the room will be communicated in my website (not Thursday 12)
- **Thursday 19: 12-14** at lab-M (not 11-13)

The course is complete, you can come and work to the project, I will be there to answer your questions :)