Tiny x86 - Architecture Simulator for Educational Purposes

Author: Ing. Ivo Strejc
Supervisor: Ing. Petr Máj
Faculty of Information Technology, CTU in Prague

FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Introduction

This thesis was created to aid compiler course taught at FIT CTU. Such compiler course faces a decision of what technologies to use and how to tackle problems associated with either:

- Compiling to a single existing architecture.
  - ⇒ Such architectures are usually complex and take a lot of time studying just to be able to compile simple programs.
  - ⇒ Students must confine to its limitations right away, like having a limited number of registers.
  - ⇒ There isn‘t a lot of chance to practice different approaches for architecture designs.

or

- Extending already existing compiler.
  - ⇒ Students must study the compilers design
  - ⇒ Most of the simpler, yet interesting topics are already implemented in the compiler, leaving only complex modifications available.

To solve these kind of problems, **Tiny x86** was created.

```
int main() {
  const char* str = "Hello world\n";
  for (int i = 0; str[i] != '\0'; ++i)
    putchar(str[i]);
}
```

Example input source file

Tiny x86 features

- It is designed to be quicky adopted by the students and allow them to build out their compiler gradually.
- Teacher has the option of setting different scenarios by altering configuration.
- Code produced by compiler to Tiny x86 can be run and benchmarked on multiple platforms.

Tiny x86 composition

Tiny86 consists of:

- **Extensible and modifiable instruction set** accommodating features from different architectural styles while resembling wide-spread x86 architecture. Instructions present in the current design range from simple ones, typical for RISC, to more complex, as found in CISC, and can be easily extended with more instructions and addressing modes.
- **Configurable virtual machine** (VM) implementing interesting features of modern processors from the compiler's point of view, like pipelining and out-of-order execution. This VM enables students to start simple, as they can set the VM to more beginner friendly configuration, and later try out advanced techniques like using a limited number of registers or minimizing pipeline stalls.
- **Reporting system** provides information that can be used for benchmarking the code generated by compilers and to help identify their weak spots for given VM configuration, such as too many memory accesses etc.

```
using namespace tiny::t86;

Program HelloWorld() {
  ProgramBuilder pb;
  DataLabel str = pb.addData("Hello world\n");
    pb.add(MOV{Reg(0), str });
    pb.add(MOV{Reg(1), 0});
  Label loop =
    pb.add(MOV{Reg(2), Mem(Reg(0) + Reg (1))});
    pb.add(CMP{Reg(2), ’\0’});
  Label jumpToBody =
    pb.add(JNE{Label::empty ()});
    pb.add(HALT {});
  Label body = pb.add(PUTCHAR{Reg(2)});
    pb.add(INC{Reg(1)});
    pb.add(JMP{loop});
    pb.patch(jumpToBody, body);
  return pb.program();
}
```

Possible translation of given example

```
Total ticks: 246
Total instructions executed: 78
Throughput: 0.317073 instructions per tick
Average instruction latency: 3.15385 ticks
Global averages:
        Times executed: 78
        Average instruction lifetime: 17.5513 ticks
        Average fetch stalls: 2.84615 ticks
        Average decode stalls: 2.84615 ticks
        Average operand fetching stalls: 4.97436 ticks
                Average register fetch stalls: 4.14103 ticks
                Average memory read stalls: 0.833333 ticks
        Average waiting for ALU: 0 ticks
        Average executing: 2.97436 ticks
        Average waiting for retirement: 1.07692 ticks
        Average retirement: 1 ticks
```

Output of reporting system for given translation

Conclusion

The **Tiny x86** was already used in 2021 summer semester as a target for students’ semestral projects, showcasing its readiness to be used for educational purposes. It might serve as a stepping stone for creation of a hypothetical tiny-verse, an unified ecosystem deployable in a whole Programming Languages and Systems Programming stack of courses.