Compiler Code Size Density

Nidal Faour
Toolchain Engineer
Western Digital
Agenda

• Code size on RISCV
  • why there is a problem?

• Research
  • Addressing the problem

• Solutions
  • GCC patch example

• Going forward
  • Research
  • Techniques
  • Test-cases: Embench

• Summary
Code size on RISC-V
Code size on RISCV

• RAM is the most expensive resource in the world of embedded devices.

• RISC-V ISA does not provide a multi operations in a single instruction form.

• RISC-V GCC & LLVM are 10-20% behind other ISA
Research
Research

• Trying out RISC-V on a real project

• Inspect the assembly for both targets, RISC-V and WD legacy vendor

• Finding weak points: flows where the compiler should have made a better decisions.

• ISA limitations will not be addressed

✓ • Compiler immaturity can be improved
Solutions
Solution – Issue & fix A

- save/restore patch:
  Unneeded prologue/epilogue functions. The GCC compiler had injected these calls where it should not.

Example: On a tail call the return to the **caller** can be done from the **callee**.

```c
1 void test()
2 {
3 ... 
4 foo();
5 ... 
6 }
```

```c
1 int foo()
2 {
3 ... 
4 return bar();
5 }
```

```c
1 int bar()
2 {
3 ... 
4 return val;
5 }
```
Solution – Issue & fix A

**Caller**

<test>:

0: \( x = y + 4 \)
1: \( ra = 3 \)
2: call foo
3: \( t = x + 1 \)
4: ...

**Current**

<foo>:

12: save ra -> “3”
13: \( v = 19 \)
14: \( ra = 16 \)
15: call bar
16: restore ra <- “3”
17: return to “3”

**Callee**

<bar>:

25: \( w = 7 \)
26: \( z = w + m \)
27: return to “16”

#RISCVSUMMIT | tmt.knect365.com/risc-v-summit/
Solution – Issue & fix B

Common st/ld base address:

• In a stream of memory access, the GCC compiler was not compress aware.

• The access was made using the address with a large offset which left us with non compressed 32 bits instructions

• Using a register to rebase the address then step in small offsets, gave the opportunity to use compressed 16 bits instruction

• Using the compressed instruction gave the same stream of instruction a reduction of 10% in code size.
## Solution – Issue & fix B

### Memory access

<table>
<thead>
<tr>
<th>Instruction</th>
<th>32 bits stream</th>
<th>Target compressed form</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 lui</td>
<td>a4,0xf0080</td>
<td>a4,0xf0080</td>
</tr>
<tr>
<td>4 lw</td>
<td>a3,248(a4)</td>
<td></td>
</tr>
<tr>
<td>4 lui</td>
<td>a5,0xf0019</td>
<td>a5,0xf0019</td>
</tr>
<tr>
<td>4 addi</td>
<td>a5,a5,-256</td>
<td></td>
</tr>
<tr>
<td>4 lw</td>
<td>a3,248(a4)</td>
<td>a4,a4,248</td>
</tr>
<tr>
<td>4 sw</td>
<td>4 lui</td>
<td>a4, 0xf0080 + 248</td>
</tr>
<tr>
<td>4 lw</td>
<td>2 c.lw</td>
<td>a3,0(a4)</td>
</tr>
<tr>
<td>4 sw</td>
<td>a3,-236(a5)</td>
<td>2 c.lw a3,4(a4)</td>
</tr>
<tr>
<td>4 lw</td>
<td>a3,256(a4)</td>
<td>2 c.sw a3,0(a5)</td>
</tr>
<tr>
<td>4 sw</td>
<td>a3,-236(a5)</td>
<td>2 c.lw a3,8(a4)</td>
</tr>
</tbody>
</table>

### Result and penalty

- **Load instruction**: = 4 bytes
- **Load instruction**: = 4 bytes
- **Rebase instruction**: = 4 bytes
- **Store/load**: = 4 bytes
- **C. Store/load**: = 2 bytes

- **Memory access**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>32 Bytes total</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 lui</td>
<td>a4,0xf0080</td>
</tr>
<tr>
<td>4 addi</td>
<td>a4,a4,248</td>
</tr>
<tr>
<td>2 c.lw</td>
<td>a3,0(a4)</td>
</tr>
</tbody>
</table>

- **Load instruction**: = 4 bytes
- **Load instruction**: = 4 bytes
- **Rebase instruction**: = 4 bytes
- **Store/load**: = 4 bytes
- **C. Store/load**: = 2 bytes

- **Memory access**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>32 Bytes total</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 lui</td>
<td>a4,0xf0080</td>
</tr>
<tr>
<td>4 addi</td>
<td>a4,a4,248</td>
</tr>
<tr>
<td>2 c.lw</td>
<td>a3,0(a4)</td>
</tr>
</tbody>
</table>

- **Load instruction**: = 4 bytes
- **Load instruction**: = 4 bytes
- **Rebase instruction**: = 4 bytes
- **Store/load**: = 4 bytes
- **C. Store/load**: = 2 bytes

- **Memory access**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>12 Bytes total</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 lui</td>
<td>a4,0xf0080</td>
</tr>
<tr>
<td>4 addi</td>
<td>a4,a4,248</td>
</tr>
<tr>
<td>2 c.lw</td>
<td>a3,0(a4)</td>
</tr>
</tbody>
</table>

- **Load instruction**: = 4 bytes
- **Load instruction**: = 4 bytes
- **Rebase instruction**: = 4 bytes
- **Store/load**: = 4 bytes
- **C. Store/load**: = 2 bytes

- **Memory access**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>16 Bytes total</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 lui</td>
<td>a4,0xf0080</td>
</tr>
<tr>
<td>4 addi</td>
<td>a4,a4,248</td>
</tr>
<tr>
<td>2 c.lw</td>
<td>a3,0(a4)</td>
</tr>
</tbody>
</table>

- **Load instruction**: = 4 bytes
- **Load instruction**: = 4 bytes
- **Rebase instruction**: = 4 bytes
- **Store/load**: = 4 bytes
- **C. Store/load**: = 2 bytes

- **Memory access**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>14 Bytes total</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 lui</td>
<td>a4,0xf0080</td>
</tr>
<tr>
<td>4 addi</td>
<td>a4,a4,248</td>
</tr>
<tr>
<td>2 c.lw</td>
<td>a3,0(a4)</td>
</tr>
</tbody>
</table>

- **Load instruction**: = 4 bytes
- **Load instruction**: = 4 bytes
- **Rebase instruction**: = 4 bytes
- **Store/load**: = 4 bytes
- **C. Store/load**: = 2 bytes

### Note:

- st/ld does not have to be a continuous series

### Equation:

\[ f(x) = C + \frac{x}{2} \cong \frac{x}{2} \rightarrow 50\% \text{ max} \]
Going forward
Going forward

• Same research is on going over LLVM

• Outlining will be provided for the LLVM compiler

• More density test cases will be added to the Embench
Going forward - outlining

**Code example**

```c
void foo() {
    ....
    a = a + b;
    b = a - b;
    a = a - b;
    ....
}

void bar() {
    ....
    t = t + z;
    z = t - z;
    t = t - z;
    ....
}
```

**Outlining**

```c
void foo() {
    ....
    #1_354298(a, b);
    ....
}

void bar() {
    ....
    #1_354298(t, z);
    ....
}

#1_354298(x,y){
    x = x + y;
    y = x - y;
    x = x - y;
}
```

**Compiler generated instruction block**

Similar code sequence
Going forward - Embench

- Code size measurement
- Lack of code size open benchmarks
- Embench – benchmark suite for embedded systems
- Dhryson – cases for code size
- CoreMark

https://embench.org/
https://github.com/westerndigitalcorporation/riscv32-Code-density-test-bench
Summary

- Current effort results – 5% improvement to code size
Thank you