CS340400 Compiler Design
Homework 3 Bonus

PL NTHU LAB

Outline

- Introduction
  - Andes DSP Extension
  - Intrinsic Function
  - Enabling DSP Extension in Arduino IDE
  - Testing Sample Project
- HW3 Bonus Specification

Introduction

- For the HW3 bonus, you need to add support for the intrinsic function of the Andes P-extension in your compiler.
- Demo of HW3 bonus can only use the F1/T1 board.
- Worth 5 points towards the total semester grade.

Andes DSP Extension

- Andes DSP extension, or P-extension (Packed SIMD Extension), enhances processors with specialized instructions for Digital Signal Processing (DSP).
- For example, to perform addition on four unsigned 8-bit integers:

RV32

```
uint8_t a[4] = {10, 20, 30, 40};
uint8_t b[4] = {40, 30, 20, 10};
uint8_t result[4];

for (int i = 0; i < 4; i++) {
      result[i] = a[i] + b[i];
}
```

RV32 with P-ext

```
// 0x0A = 10, 0x14 = 20, 0x1E = 30, 0x28 = 40
uint8x4_t a = 0x0A141E28;
uint8x4_t b = 0x281E140A
uint8x4_t result;

result = __rv_v_uadd8(a, b);
```

- riscv-p-spec

Intrinsic Function

- An intrinsic function in C is a function provided by the compiler that is not part of the standard C, but is built into the compiler itself.
- These functions are directly mapped to machine instructions, often as specialized extensions, enabling more efficient execution.
- Andes toolchain supports p-extension intrinsics, by including the <nds_intrinsic.h> header file.

```
extern "C" {
#include <nds_intrinsic.h>
}
```

```
uint32_t c = __rv__ukadd8(a, b);
```

```
lw    a5, -24(s0)       # Load a1 (saved value) into a5
ukadd8   a5, a4, a5     # UKADD8 a4 and a5, store the result in a5
mv    a0, a5            # Move the result to a0 (return value)
```

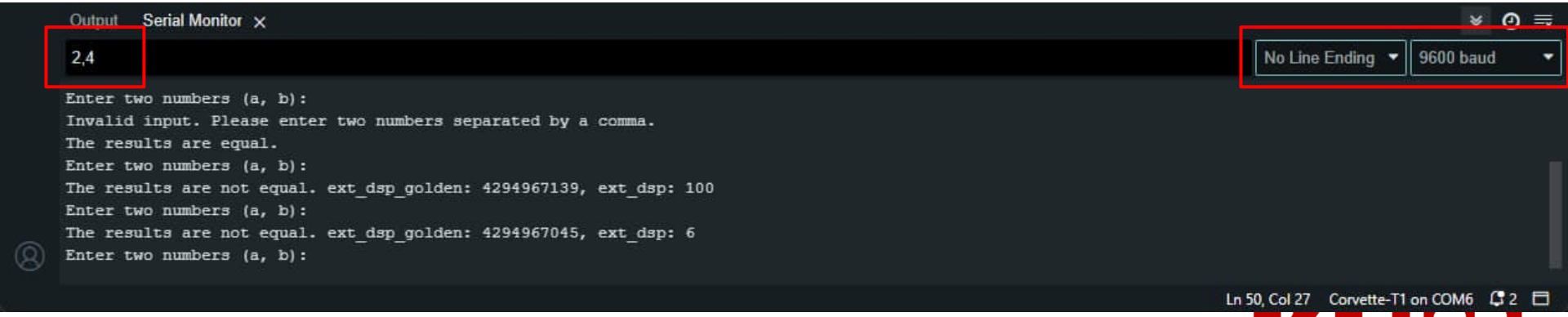
Enabling P Extension in Arduino IDE

- To enable the p-extension of the toolchain, edit the compilation flags in the platform.txt file located at:
  "C:\Users\(youraccount)\AppData\Local\Arduino15\packages\Corvette\hardware\ Corvette-(F1-N25 or T1)\1.8.13\platform.txt" .
- C flags:
  - `- compiler.c.flags=-g3 -Os -c …`
  - `+ compiler.c.flags=-g3 -Os -mext-dsp -c …`
- Assembly flags:
  - `- compiler.S.flags=-g3 -Os -c …`
  - `+ compiler.S.flags=-g3 -Os -mext-dsp -c …`
- C++ flags:
  - `- compiler.cpp.flags=-g3 -Os -c …`
  - `+ compiler.cpp.flags=-g3 -Os -mext-dsp -c …`
- Reopen the Arduino IDE to apply the changes.

NTHU PL LAB

Testing Sample Project

- Open HW3_Bonus.ino.
- Navigate to Tools > Board/Port, and select the correct board and serial port.
- Upload the sketch.
- Open Tools > Serial Monitor, configure it to use “No Line Ending” and “9600 baud”.
- When connected, type `"int, int"` to test the program.
  - e.g. `“65280, 65283”, “100, 200”, …`



HW3 Bonus Specification

Support Four SIMD 8-bit Intrinsics

- UKADD8 (__rv__ukadd8)
  - Syntax: ukadd8 rd, rs1, rs2
- UKSUB8 (__rv__uksub8)
  - Syntax: uksub8 rd, rs1, rs2
- CMPEQ8 (__rv__cmpeq8)
  - Syntax: cmpeq8 rd, rs1, rs2
- UCMPLT8 (__rv__ucmplt8)
  - Syntax: ucmplt8 rd, rs1, rs2

- riscv-p-spec

Testcase

- In the .ino file, the loop() function waits for your input values (int, int).
- Once it receives the two integers, it passes them to ext_dsp_golden() and ext_dsp_codegen() to test if their output are equal.
- During demo, TA will provide you a .ino file and the testcase.c.
- Your compiler should generate the codegen.S for the testcase.c file.

```
// Calculate the outputs of the functions
uint32_t result_golden = ext_dsp_golden(a, b);
uint32_t result = ext_dsp_codegen(a, b);

// Compare the results and print the output
if (result_golden == result) {
```

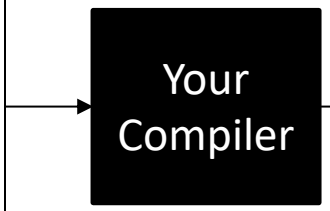
NTHU PL LAB

Testcase (cont.)

- The function body of ext_dsp_golden() in the .ino is identical to the function body of the ext_dsp_codegen() in testcase.c.

```
// .ino
uint32_t ext_dsp_golden(uint32_t a, uint32_t b) {
        uint32_t c = __rv_ukadd8(a, b);
        return c;
}
```

```
// testcase.c
uint32_t exp_dsp_codegen(uint32_t a, uint32_t b);
uint32_t ext_dsp_codegen(uint32_t a, uint32_t b) {
        uint32_t c = __rv_ukadd8(a, b);
        return c;
}
```



Grading Policies

- Total 5 bonus points.
- 2 – successfully run the sample project
- 4 – pass the released testcase (testcase.c)
- 5 – pass the hidden testcase