Ćwiczenie 13: Minimalizacja rozmiaru programu

Instrukcja laboratorium

Mariusz Chilmon <mariusz.chilmon@ctm.gdynia.pl>

2024-04-13

> One of my most productive days was throwing away 1000 lines of code.
>
> — *Kenneth Lane Thompson*

Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z:

- poziomami optymalizacji kompilatora GCC,
- kosztem obliczeń zmiennoprzecinkowych,
- kosztem pełnej implementacji funkcji z rodziny `printf()`.

Uruchomienie programu wyjściowego

1. Podłącz płytkę *LCD Keypad Shield* do *Arduino Uno*.
2. Podłącz termometr LM35DZ do linii *A5*.
3. Na wyświetlaczu widoczny jest odczyt wartości z ADC, mierzącego napięcie z termometru scalonego LM35DZ, i obliczona na tej podstawie temperatura.
4. **Zapisz w sprawozdaniu wyjściowy rozmiar programu (wersja zmiennoprzecinkowa).**

Rysunek 1: Wyjściowy stan wyświetlacza

Zadanie podstawowe

Celem zadania podstawowego jest zmniejszenie rozmiaru programu przez zastąpienie obliczeń zmiennoprzecinkowych obliczeniami na liczbach całkowitych.

Wymagania funkcjonalne

1. Funkcjowanie urządzenia nie zmienia się[1].
2. Następuje zmniejszenie rozmiaru programu o około 400 B…500 B.

[1] Wynik pomiaru może różnić się o 0,01°C ze względu na brak zaokrąglania przy obliczeniach całkowitych.

Modyfikacja programu

Opis rozwiązania wyjściowego (operacji zmiennoprzecinkowych)

Zależność między odczytem ADC z 10-bitowego przetwornika analogowo-cyfrowego a temperaturą T, wyznaczona w oparciu o dokumentację ADC oraz termometru scalonego LM35DZ, wyraża się wzorem:

$$T = \frac{V_{REF} \cdot ADC}{2^{10}} \cdot 100 \frac{\text{°C}}{\text{V}} \tag{1}$$

co przy napięciu referencyjnym 1,1 V daje:

$$T = \frac{1{,}1 \cdot ADC}{1024} \cdot 100\text{°C} \tag{2}$$

z czego wynika:

```
double temperature = 1.1 * measurement / 1024 * 100;
```

co z kolei można zamienić na tekst w buforze `char buf[]` za pomocą funkcji `snprintf()`:

```
snprintf(buf, sizeof(buf), "%5.2f", temperature);
```

która wartość zmiennoprzecinkową `temperature` formatuje do postaci tekstowej za pomocą parametru `"%5.2f"`:

% Prefiks oznaczający pole formatujące.

5 Szerokość pola (tu: 5 znaków, włącznie z kropką dziesiętną).

.2 Precyzja — liczba cyfr części ułamkowej (tu: 2 miejsca po przecinku).

f Formatowanie liczby zmiennoprzecinkowej do formatu ułamka dziesiętnego.

Zamiana obliczeń zmiennoprzecinkowych na całkowite

Zauważ, że w ramach określonej precyzji wyniku można zastąpić obliczenia zmiennoprzecinkowe obliczeniami na liczbach całkowitych, np. obliczenia na długościach wyrażonych w ułamkach metra można zastąpić obliczeniami na tych samych długościach wyrażonych w milimetrach — jeżeli nie dochodzi do utraty dokładności, jest to tylko zmiana sposobu zapisu liczb.

Weźmy na przykład obliczenia:

```
double result = 3.14 * 1.42;
printf("%5.3f\n", result);
// Wynik: "4.459"
```

Możemy przeskalować wartości ułamkowe do wartości całkowitych, a wynik wyświetlić jako dwie liczby całkowite oddzielone kropką. Przed kropką mamy część całkowitą uzyskaną za pomocą dzielenia całkowitego `fixed / 1000` zaś za kropką mamy część ułamkową, która jest resztą z tego dzielenia `fixed % 1000`.

Należy tylko zauważyć, że pomnożyliśmy obie stałe przez 100, więc wynik jest $100 \cdot 100 = 10000$ razy większy od oczekiwanego. Ponieważ na potrzeby wyświetlania dzielimy go tylko przez 1000, należy go jeszcze przeskalować o rząd wielkości dzieląc przez dziesięć.

```
uint32_t fixed = 314 * 142 / 10;
printf("%lu.%03u\n", fixed / 1000, fixed % 1000);
// Wynik: "4.458"
```

Wynik różni się na najmniej znaczącej pozycji ze względu na to, że dzielenie całkowite odrzuca część ułamkową (ignoruje resztę z dzielenia), nie dokonując zaokrąglenia. Akceptujemy ten błąd na potrzeby ćwiczenia, można go jednak obsłużyć, samodzielnie implementując zaokrąglenie.

Przeprowadź analogiczną redukcję obliczeń pomiaru temperatury w funkcji `printMeasurement()`, zmieniając wartość zmiennoprzecinkową `1.1` na wartość całkowitą.

W razie problemów pamiętaj, że:

1. Dzielenie całkowite odrzuca część ułamkową, obniżając precyzję obliczeń, więc najlepiej wykonywać je na końcu.
2. Nawet jeżeli ostateczny wynik obliczeń mieści się w zakresie np. typu `uint16_t`, to pośrednie obliczenia mogą ten typ przepełnić.
3. Kompilator dobiera typ stosowany do obliczeń na podstawie pierwszej wartości w ciągu obliczeń.
4. Dla kompilatora AVR-GCC typem domyślnym jest 16-bitowy **`int`** ze znakiem (`int16_t`), którego zakres jest znacznie mniejszy od 32-bitowego czy 64-bitowego stosowanego w komputerach PC.
5. Aby nastąpiła redukcja rozmiar programu, nie może on w żadnym miejscu wykonywać operacji zmiennoprzecinkowych.

Można wymusić na kompilatorze stosowanie szerszego typu. Wyrażenie `1000 * 1000 / 1000` da nieprawidłowy wynik 16, gdyż liczba 1 000 000, będąca wynikiem mnożenia, nie mieści się w zakresie `int16_t`. Prawidłowy wynik można uzyskać, każąc zacząć obliczenia na typie `uint32_t` za pomocą wyrażenia `UINT32_C(1000)* 1000 / 1000`.

Zadanie rozszerzone

Celem zadania rozszerzonego jest zmniejszenie rozmiaru programu przez użycie odpowiednich flag kompilatora i linkera.

Wymagania funkcjonalne

1. Funkcjowanie urządzenia nie zmienia się.
2. Następuje zmniejszenie rozmiaru programu.

Modyfikacja programu

Dobranie poziomu optymalizacji kompilatora

Znajdź w pliku `Makefile` zmienną `OPTIMIZATION` i sprawdź jej wpływ na rozmiar programu, korzystając z tabeli w sprawozdaniu.

Plik `Makefile` odpowiedzialny jest za śledzenie zmian w kodzie źródłowym i nie wykrywa on zmian w samym sobie. Oznacza to, że przed każdym zbudowaniem programu należy wywołać zadanie `clean`, żeby kompilacja przebiegła od początku.

Użycie uproszczonej implementacji funkcji rodziny `printf()`

Po zmianie obliczeń ze zmiennoprzecinkowych na stałoprzecinkowe możliwe jest użycie biblioteki z uproszczonymi funkcjami rodziny `printf()`. W tym celu z pliku `Makefile` należy usunąć linię z flagami linkera, które nakazują mu użycie biblioteki wspierającej operacje zmiennoprzecinkowe zamiast domyślnej, uproszczonej:

```
1 LDFLAGS += -Wl,-u,vfprintf -lprintf_flt
```

Pełna implementacja funkcji `printf()` jest bardzo wymagająca, gdyż osbługuje wiele formatów. Szczególnie wymagające jest formatowanie typów zmiennoprzecinkowych ze względu na możliwości standardu IEEE 754 oraz złożoność obliczeń potrzebnych do obsługi różnych sposobów reprezentacji ułamków. Z tego powodu w systemach wbudowanych często stosuje się zredukowane biblioteki standardowe.