STUDIA INFORMATICA 2010
Volume 31 Number 1 (88)

Dariusz CABAN
Politechnika Śląska, Instytut Informatyki

OPTYMALNE KONSTRUKCJE JĘZYKA C W PROGRAMACH DLA 8-BITOWYCH MIKROSTEROWNIKÓW PIC

Streszczenie. Zdaniem niektórych osób programy dla mikrosterowników z wewnętrzną pamięcią programu o małej pojemności należy pisać tylko w języku asemblera. Kod wynikowy programu w języku C z reguły zajmuje więcej pamięci. Nie musi być to dużo więcej. Wiele zależy od sposobu zapisania programu źródłowego.

Słowa kluczowe: mikrosterownik, język C, kod źródłowy, kod wynikowy

OPTIMAL C CONSTRUCTS FOR 8-BIT PIC MICROCONTROLLERS

Summary. Some person think that software for microcotrollers with small internal program memory should be written in assembly language only. The object code of C program usually occupy more memory space. It is not having to be far more. The way of writing the source code is playing a significant role.

Keywords: microcontroller, C language, source code, object code

1. Wprowadzenie

Ośmiobitowe mikrosterowniki PIC zawierają pamięć programu typu Flash, o pojemności od 256 słów do 64 kilosłów [2, 5, 7]. Najczęściej nie można do nich przyłączać zewnętrznej pamięci programu. Istnieje pogląd, że kod wynikowy programów w języku C zajmuje zbyt dużo pamięci [4], przez co programy dla takich mikrosterowników należy pisać tylko w języku asemblera. Współczesne kompilatory języka C generują wydajny kod wynikowy, przy czym wpływ na jego rozmiar ma także sposób zakodowania programu źródłowego.

W artykule przedstawiono kilka wskazówek, jakie konstrukcje języka C należy stosować, aby zmniejszyć rozmiar kodu wynikowego programu. Programy były tłumaczone przy

użyciu kompilatorów firmy CCS w wersji 3 (obecnie w sprzedaży jest wersja 4) [3] Są to niedrogie narzędzia – zakup pakietu, w którego skład wchodzą kompilatory dla wszystkich serii 8-bitowych mikrosterowników PIC (Baseline, Midrange, PIC18 i najnowszej Enhanced Midrange, oczywiście nieobsługiwanej przez kompilator w wersji 3) oraz zintegrowane środowisko uruchomieniowe to wydatek zaledwie 500 dolarów. Bardzo często podawane są zakresy rozmiarów kodu wynikowego prezentowanych konstrukcji, a nie pojedyncze wartości. Wynika to m. in. z tego, że mikrosterowniki serii PIC18 posiadają o wiele bogatszą listę rozkazów niż pozostałe. Nie występowała konieczność przełączania banków pamięci danych ani odpowiedniego ustawiania bitów wykorzystywanych przy adresowaniu pośrednim.

2. Czy komunikat Out of ROM zawsze informuje o przekroczeniu pojemności pamięci programu?

W mikrosterownikach serii Midrange oraz niektórych serii Baseline pamięć programu jest podzielona na strony. Rozmiar strony zależy od liczby bitów adresu zawartych w rozkazach: skoku bezwarunkowego i wywołania podprogramu, numer strony jest zapisany w rejestrze PCLATH lub w rejestrze statusu. Pamięci może wystarczyć na przechowanie całego programu, ale gdy kod wynikowy którejś z funkcji w całości nie zmieści się na stronie, zostanie wyświetlony komunikat **Out of ROM**.

Na rozmiar kodu wynikowego funkcji wpływ ma m. in. to, jak są tłumaczone zawarte w niej wywołania innych funkcji. Mikrosterowniki ze stronicowaną pamięcią programu mają niewielki stos sprzętowy na przechowanie adresów powrotu. Z tego powodu kompilator może powielać kod wynikowy wywoływanej funkcji w kodzie funkcji wywołującej. Powielaniu kodu zapobiega się przez poprzedzenie funkcji dyrektywą `#separate`. Jeżeli jednak wywoływanie innej funkcji przy użyciu rozkazu `CALL` jest niewskazane, trzeba przerobić funkcję wywołującą, np. podzielić ją na kilka mniejszych.

3. Odpowiedni dobór typów danych

Jednostka centralna w 8-bitowych mikrosterownikach PIC wykonuje operacje nie tylko na bajtach, ale i na pojedynczych bitach. Dowolny bit pamięci danych może być ustawiany i zerowany, od jego stanu można uzależnić wykonanie skoku. Dlatego tam, gdzie to możliwe, należy używać zmiennych zajmujących jeden bajt lub bit pamięci [4]. Kod wynikowy instrukcji pętli

```
for(i = 0; i < N; i++)
```

gdzie `N` jest stałą, zajmuje 6÷9 słów przy zmiennej sterującej typu `int8` (liczba całkowita 8-bitowa, bez znaku), a 11÷15 słów przy zmiennej typu `int16`. Tej części instrukcji warunkowej

```
if (flag1 && flag2)
```

przy zmiennych bitowych i mikrosterowniku docelowym serii Baseline lub Midrange, odpowiada ciąg 3÷4 rozkazów. Przy zmiennych typu `int8` jest to ciąg 5÷6 rozkazów. Jeżeli docelowy jest mikrosterownik serii PIC18, kod wynikowy w obu rozpatrywanych przypadkach liczy 4÷6 rozkazów. Wynik tłumaczenia zależy tutaj także od rozmiaru kodu instrukcji wykonywanych przy spełnieniu warunku.

Nie należy bez potrzeby używać liczb ze znakiem. Rozmiar kodu wynikowego powyższej pętli przy zmiennej sterującej typu `signed int8` wzrasta o 2 słowa.

4. Modyfikacja elementu tablicy

Do zmiany o 1 wartości elementu tablicy stosuje się z reguły operator ++ lub --. W innych przypadkach najlepiej używać operatorów przypisania `op=`, gdzie `op` jest jednym z operatorów arytmetycznych. Kod wynikowy instrukcji

```
array[i] += b;
```

gdy elementy tablicy, indeks oraz zmienna *b* są typu `int8`, zajmuje 6÷10 słów, natomiast instrukcji

```
array[i] = array[i] + b;
```

13÷24 słów.

5. Arytmetyka modulo

Obliczenie wartości indeksu kolejnej komórki bufora okrężnego do odczytu lub zapisu wymaga wykonania m. in. dzielenia modulo rozmiar bufora. Zazwyczaj używa się do tego operatora `%`. Jeżeli dzielnik jest stałą równą potędze dwójki, obliczenie wartości wyniku polega na wyzerowaniu części bitów dzielnej. Przy indeksie *i* typu `int8` kod wynikowy instrukcji

```
i = ++i % 32;
```

zajmuje zaledwie 3 słowa pamięci. Dzielenie modulo przez liczbę, która nie jest potęgą dwójki, wykonuje odpowiedni podprogram. Rozmiar kodu wynikowego tej instrukcji

```
i = ++i % 60;
```

wynosi 29÷30 słów, z czego 21÷22 słów przypada na kod podprogramu dzielenia. Lepiej ją zastąpić instrukcją warunkową

```
if (++i == 60)
  i = 0;
```

której kod wynikowy zajmuje 5 słów.

6. Operacje wejścia-wyjścia

Linie wejścia-wyjścia są pogrupowane w porty. Z każdym portem związane są dwa, a w mikrosterownikach serii PIC18 trzy rejestry specjalnego przeznaczenia, w tym rejestr konfiguracji. Do realizacji operacji wejścia-wyjścia, jak i sterowania wewnętrznymi układami peryferyjnymi producent kompilatora zaleca użycie, o ile to możliwe, funkcji bibliotecznych zamiast bezpośrednich odwołań do rejestrów [1]. Kod takiej funkcji jest powielany w kodzie funkcji wywołującej.

Przy domyślnej obsłudze wejścia-wyjścia w funkcji bibliotecznej ustawiana jest najpierw odpowiednia konfiguracja portu, co wymaga 1÷5 rozkazów. Gdy przeznaczenie linii portu jest niezmienne w trakcie wykonywania programu lub zmiana konfiguracji następuje rzadko, warto wymusić tzw. szybką obsługę wejść-wyjść. Służy do tego dyrektywa `#use fast_io(x)`, gdzie `x` to symbol portu (`A`...`G`). Konfigurację portu ustawia się wówczas przy użyciu funkcji `set_tris_x()`, której kod wynikowy składa się z 2 rozkazów.

7. Składanie bajtów w słowa

Słowo 16-bitowe z bajtów tworzy się z reguły za pomocą instrukcji:

```
word = ((int16)highbyte << 8) + lowbyte;
```

lub

```
word = ((int16)highbyte << 8) | lowbyte;
```

Jawne przekształcenie typu zmiennej *highbyte* musi wystąpić, jeżeli używamy kompilatora CCS, bez tego starszy bajt wyniku miałby wartość 0. Rozmiar kodu wynikowego powyższych instrukcji wynosi odpowiednio 8÷9 oraz 7 słów. Rzutowanie jest niepotrzebne, gdy operacje: przesunięcia i dodawania zapisze się w osobnych instrukcjach. Ich kod

wynikowy zajmuje odpowiednio 7 i 5 słów. Warto skorzystać z funkcji bibliotecznej `make16()` – jej kod wynikowy, powielany w kodzie funkcji wywołującej, zajmuje 4 słowa, są to wyłącznie rozkazy przesłań.

8. Testowanie wybranego bitu zmiennej

Kod wynikowy części decyzyjnej instrukcji warunkowych

```
if (status & 0x80) ...
if (bit_test(status, 7)) ...
if (busy) ...
```

gdzie: *status* – zmienna bajtowa, *busy* – zmienna bitowa o takim samym adresie co adres najstarszego bitu zmiennej *status* zajmuje 1÷3 słów. Przy tak zapisanym wyrażeniu warunkowym:

```
(status & 0x80) == 0x80
```

kod ten zajmuje 4÷7 słów.

9. Negacja zmiennej bitowej

Można to wykonać kilkoma sposobami:

```
bitvar ^= 1;
bitvar = !bitvar;
bitvar = ~bitvar;
```

Kod wynikowy pierwszej instrukcji zajmuje aż 10 słów, pozostałych 1÷2 słów.

10. Inicjowanie zmiennych

Z nieinicjowanych zmiennych tylko zmienne statyczne domyślnie otrzymują wartość 0 przed rozpoczęciem wykonywania kodu funkcji `main()`. Na zmienną przypada *n* rozkazów zerowania bajtu pamięci, gdzie *n* to jej rozmiar w bajtach.

Nadanie wartości początkowej zmiennej inicjowanej wymaga od *n* do 2**n* rozkazów (zapis wartości niezerowej do bajtu pamięci to dwa rozkazy). Do zainicjowania tablicy lub struktury czasem lepiej użyć bibliotecznej funkcji `memset()`. Kod tej funkcji jest powielany w kodzie funkcji wywołującej i liczy 8÷10 rozkazów.

Można wymusić wypełnienie zerami całej wykorzystywanej przez program pamięci danych[1] (dyrektywa `#zero_ram`). Gdy duża liczba zmiennych, niebędących zmiennymi statycznymi, ma mieć wartość początkową równą 0, rozwiązanie takie może być opłacalne. Kod wykonujący taką operację zajmuje 11÷54 słów, zależnie od serii mikrosterownika oraz liczby wypełnianych banków pamięci. Nieinicjowane zmienne statyczne będą niestety zerowane ponownie.

11. Pętle o zadanej liczbie obiegów

Każdą z pętli w języku C można zapisać tak, aby pewne operacje zostały wykonane zadaną liczbę razy. W tabeli 1 podano rozmiary kodu wynikowego kilku pętli, o liczbie obiegów `N` znanej na etapie tworzenia programu. Zmienna sterująca *i* jest typu `int8`. Zmiennej sterującej pętli 3 i 4 należy jeszcze przypisać wartość początkową, co zajmuje odpowiednio 1 i 2 słowa.

Tabela 1
Rozmiary kodu wynikowego pętli o liczbie obiegów określonej wartością stałą

pętla	rozmiar kodu wynikowego [w słowach]
`for(i = 0; i < N; i++)`	6÷9
`for(i = N; i; i--)`	6÷7
`do while(++i < N)`	4÷7
`do while(--i)`	2

Tabela 2 zawiera rozmiary kodu wynikowego kilku pętli, dla których liczba obiegów jest obliczana w trakcie wykonywania programu i zapisana w zmiennej *Count*. Zmienne: *i* oraz *Count* są typu `int8`. Zmienną `i` przed wejściem w pętlę `do-while` trzeba jeszcze wyzerować, co wykonuje 1 rozkaz.

Tabela 2
Rozmiary kodu wynikowego pętli o liczbie obiegów zapisanej w zmiennej

pętla	rozmiar kodu wynikowego [w słowach]
`for(i = 0; i < Count; i++)`	6÷7
`for( ; Count; Count--)`	4÷5
`do while(++i < Count)`	4÷5
`do while(--Count)`	2

Podane pętle `do-while` mają tak krótki kod wynikowy, dlatego że do ich organizacji wykorzystuje się rozkaz `DECFSZ`. Jego działanie jest następujące: zawartość wskazanego bajtu pamięci jest zmniejszana o 1 i jeśli wynik będzie równy 0, kolejny rozkaz zostanie pominięty. Pomijanym rozkazem jest w tym przypadku rozkaz skoku na początek pętli.

[1] W programach dla mikrosterowników serii Baseline i Midrange, w których pamięć danych jest

12. Rozwijanie pętli

Jeżeli liczba obiegów pętli jest mała, czasem warto zrezygnować z jej użycia i powielić instrukcje. Taką modyfikację wartości elementów tablicy

```
for(i = 0; i < 3; i++)
  temp_disp[i] += '0';
```

realizuje kod złożony 13÷16 rozkazów, gdy elementy tablicy i zmienna sterująca pętli są typu `int8`. Kod wynikowy ciągu instrukcji przypisania

```
temp_disp[0] += '0';
temp_disp[1] += '0';
temp_disp[2] += '0';
```

zajmuje 4 słowa. Gdy indeks elementu tablicy jest stałą, jego adres jest obliczany podczas tłumaczenia programu i dostęp do niego odbywa się w trybie adresowania bezpośredniego.

13. Instrukcja `switch`

Jeżeli wartość wyrażenia jest 8-bitową liczbą bez znaku, liczba przypadków n wynosi co najmniej 4, brak przypadku `default`, a mikrosterownik docelowy należy do serii Midrange lub PIC18, to wybór przypadku i przekazanie sterowania może być niekiedy realizowane za pośrednictwem tablicy konwersji. Wartość wyrażenia, w razie potrzeby przeskalowana do zakresu $0...V_{max} - V_{min}$, gdzie V_{max} i V_{min}, to największa i najmniejsza wartość wybierająca, jest indeksem komórki z rozkazem lub adresem skoku w odpowiednie miejsce. Tablica zawiera rozkazy skoków wtedy, gdy niemożliwy jest programowy odczyt zawartości pamięci programu.

Użycie tablicy konwersji skraca kod wynikowy instrukcji `switch` w programie dla mikrosterownika serii Midrange dla każdego $n > 4$. Jeśli program jest przeznaczony dla mikrosterownika serii PIC18, nie jest to rozwiązanie korzystne przy małym n. Przy wyborze za pomocą tablicy rozmiar kodu wynikowego wynosi $n + 19$ słów (+ 2 słowa, jeśli trzeba przeskalować wynik wyrażenia), z czego 13 słów przypada na odczyt adresu skoku i jego załadowanie do licznika rozkazów, w przeciwnym razie $2*n + 2$ słowa. W takiej sytuacji warto dodać pusty przypadek `default`.

---

podzielona na 2 lub 4 banki, domyślnie wykorzystywana jest pamięć tylko z połowy banków.

14. Przekazywanie argumentów funkcji

Argumenty przekazywane przez wartość funkcja otrzymuje w wydzielonym bloku pamięci. Tak samo argumenty przekazywane przez referencje, gdy kod źródłowy funkcji jest poprzedzony dyrektywą `#separate`, są to wtedy adresy argumentów. Domyślnie kod funkcji z argumentami przekazywanymi przez referencje jest powielany w kodzie funkcji wywołującej. W takim przypadku funkcja odwołuje się bezpośrednio do zmiennych będących argumentami, co skraca jej kod wynikowy oraz eliminuje konieczność kopiowania wartości zmiennych lub ich adresów. Jeżeli funkcja jest wywoływana wielokrotnie, rozwiązanie to nie musi okazać się korzystniejsze.

15. Zakończenie

Już tak prosty zabieg jak stosowanie, tam gdzie są wystarczające, zmiennych bajtowych i bitowych przyczynia się do zmniejszenia rozmiaru kodu wynikowego programu i to niezależnie od użytego kompilatora. Rozwinięcia asemblerowe niektórych z przedstawionych w artykule konstrukcji mogą jednak zależeć od kompilatora.

BIBLIOGRAFIA

1. Borowik B., Borowik W., Borowik B.: Meandry języka C/C++. Wydawnictwo Naukowe PWN, Warszawa 2006.
2. Borowik B., Borowik B., Kurytnik I. P.: Mikrokontroler PIC w zastosowaniach. Wydawnictwo Pomiary Automatyka Kontrola, Gliwice 2009.
3. CCS: C Compiler Reference Manual. July 2003.
4. Jones N.: Efficient C Code for Eight-Bit MCUs. Embedded Systems Programming, November 1998.
5. Karyś S.: Wprowadzenie do programowania mikrokontrolerów serii 18F firmy Microchip. Wydawnictwo Politechniki Świętokrzyskiej, Kielce 2008.
6. Pełka R.: Mikrokontrolery. Architektura, programowanie, zastosowanie. Wydawnictwa Komunikacji i Łączności, Warszawa 1999.
7. Pietraszek S.: Mikrokontrolery PIC12Fxxx w praktyce. Wydawnictwo BTC, Warszawa 2005.

Recenzent: Dr inż. Bohdan Borowik

Wpłynęło do Redakcji 5 października 2009 r.

Abstract

8-bit PIC microcontrollers have from 256 words to 64 kwords of Flash memory to store program code. Most often external program memory can not be used. Some person think that software for such microcontrollers should not be written in C language. The object code of C program is usually bigger than object code of assembly program. It is not having to be much bigger. Size of object code can be lowered by using of appropriate constructs of high-level language. This article presents such constructs. Test programs were translated by CCS compilers (version 3, object code for Enhanced Midrange PICs can not be generated). They are not expensive tools.

CPU of 8-bit PICs can process byte or bit. Already applying byte and bit variables, where they are sufficient, results in reducing of size of object code, independently of the compiler. But some presented constructs can be translated differently, when compiler of other manufacturer is used.

Adres

Dariusz CABAN: Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16,
44-100 Gliwice, Polska, Dariusz.Caban@polsl.pl.