# Piszę w języku C, bo...

Języki programowania pisane dla „dużych" komputerów posiadają naleciałości środowiska, dla którego są przeznaczone w związku z tym, że pracują pod kontrolą konkretnego systemu operacyjnego, z wykorzystaniem konkretnych bibliotek. W „małych" mikrokontrolerach jest inaczej. Brak systemu operacyjnego (w większości prostych aplikacji programiści tworzą namiastkę własnego OS) jest powodem, dla którego wszystkie kompilatory, bez względu na język programowania, są w stanie wyprodukować prawie taki sam kod wynikowy. Może inaczej – gdyby pisała je ta sama firma, lub osoba, to kod wynikowy powstający po kompilacji przy pomocy języków Pascal, Basic i C będzie bardzo podobny. Wynika z tego bardzo dobra informacja: nie ma znaczenia, czy program powstaje w Basicu, C, czy Pascalu – liczą się umiejętności! Owszem, przy pomocy asemblera można wykonać najmniejszy i działający najbardziej sprawnie kod wynikowy, ale czy każdy jest w stanie to zrobić? Czy nie jest nadmiernym brakiem skromności twierdzenie, że jest się lepszym od całego zespołu programistów pracujących nad kompilatorem języka C? Albo z innej strony. Osobiście dla większości aplikacji wybieram język C, ale nie stronię od różnych Basic'ów i Visual Composer'ów, jeśli aplikacja jest prosta i chcę ją zrobić szybko. Moim zdaniem C ma najwięcej zalet i daje największą swobodę programiście. Nie zawsze jest to cecha dobra, ponieważ częstokroć nadmiar swobody, idący w parze z brakiem doświadczenia lub po prostu niewiedzą, jest niebezpieczeństwem dla aplikacji. Niestety, jeszcze większą swobodę programowania daje każdemu asembler.

Zaczynałem swoją praktykę programowania od asemblera. We wczesnych latach porównywałem kod napisany w asemblerze z tym tworzonym przez kompilator. Często uśmiechałem się przy tym sam do siebie, utwierdzając się w przekonaniu, że jestem lepszy. Tak, to prawda – kod wynikowy tworzony przeze mnie był mniejszy, ale czas potrzebny na jego napisanie i uruchomienie był znacznie dłuższy, niż dla programów napisanych w języku wysokiego poziomu. Wówczas jednak nie miałem jeszcze tej świadomości. To fakt, że większość kompilatorów była wtedy typu „*command line*", że kosztowały koszmarnie wielkie kwoty pieniędzy, a do tego były niezbyt wygodne w użyciu, że nie mieliśmy do dyspozycji pamięci większych, niż 2 kB, ale czasy się zmieniły, a wraz z nimi odszedł tamten sposób myślenia. Teraz tak naprawdę liczy się to, ile jest się w stanie wyprodukować i w jakim czasie. Decydującym kryterium jest tzw. *time to market.* I obojętnie co będą twierdzić zwolennicy programowania w asemblerze, to właśnie kompilatory języków wysokiego poziomu pozwalają na maksymalne skrócenie czasu od pomysłu do realizacji.

Dziś większość programów piszę w języku C, czasami robię wstawki w asemblerze, i to tylko dla krytycznych czasowo procedur obsługi, i tylko wówczas, gdy naprawdę muszę. Z reguły unikam jednak nawet tego, ponieważ niepotrzebne używanie asemblera robi program nieczytelnym. Jednym zdaniem: staram się tak pisać program, aby w całości był w C. Krótki czas potrzebny na napisanie aplikacji jest moim zdaniem znacznie ważniejszym czynnikiem, aniżeli niewielki kod wynikowy. Przy dzisiejszych cenach mikrokontrolerów i układów pamięci kto dba o to, czy program zajmuje 4, czy 16 kB? Dopóki mieści się w pamięci procesora i dobrze wykonuje swoje zadanie, to wielkość zajmowanej przezeń pamięci nie ma absolutnie żadnego znaczenia. Ponadto uważam, że początkujący programista nie mający zbyt dużego doświadczenia w programowaniu, np. o półrocznym stażu, niekoniecznie napisze program w asemblerze, którego kod wynikowy będzie mniejszy niż ten utworzony przez kompilator języka wysokiego poziomu. Moim zdaniem, o czym już wspomniałem wcześniej, używanie jako jedynego kryterium ilości zajmowanej pamięci, jest w dzisiejszych czasach trywializmem i zupełnie nie ma sensu. **Kod programu nie powinien być jak najmniejszy, a efektywnie realizować powierzone mu zadania i powinien powstać w jak najkrótszym czasie.**

Niektóre osoby twierdzą, że pisząc w asemblerze czują się jak „Neo chwytający kule", co też moim zdaniem nie jest tak do końca prawdą. Łatwo jest mówić w ten sposób, gdy pisze się dla jednego typu procesora (np. z rdzeniem 8051), ale co zrobić, gdy ten sam algorytm trzeba przenieść na inny typ mikrokontrolera, z innym rdzeniem? Wówczas pisząc w asemblerze trzeba usiąść i na nowo napisać praktycznie cały program, bo nowy procesor, to nowe tryby adresowania, inne rozkazy, często inne nazwy rejestrów i struktura logiczna CPU. Tak naprawdę liczy się efektywny algorytm, a nie „chwytanie kul"! Asembler to tylko narzędzie i to do tego narzędzie bardzo trudne w użyciu, mało elastyczne i przez to wymagające mnóstwa czasu. Kiepski programista będzie miał tak problemy w asemblerze, jak i w każdym innym języku programowania.

Owszem, wielu programistów w świecie mikrokontrolerów może czuć się jak Neo, jednak liczba osób, które pisząc programy może konkurować z producentami kompilatorów zmniejsza się wraz z kolejną generacją mikrokontrolerów pojawiających się na rynku. A kto wie, co wydarzy się za 10 lat? Prawdopodobnie w systemach embedded pojawią się rdzenie mikrokontrolerów o zmiennej liczbie cykli maszynowych potrzebnych na realizację instrukcji, zależnie od tego, w jakiej lokalizacji pamięci umieszczone są dane oraz jaka jest historia ich użycia (np. dla danych umieszczonych w pamięci *cache*). Wówczas bardzo trudno będzie wyznaczyć np. liczbę cykli w obsłudze przerwania, przewidzieć czas realizacji procedury i panować nad tym, co robi CPU procesora. Jednocześnie wzrośnie szybkość pracy procesorów i to, czy obsługa przerwania będzie napisana w asemblerze, czy w języku wysokiego poziomu, przestanie mieć znaczenie. Już dziś można kupić popularną 51-kę taktowaną zegarem 100 MHz (sic!) o cyklu maszynowym równym cyklowi zegarowemu. A przecież nie można założyć, że na tym postęp w dziedzinie mikrokontrolerów się zakończy. A co z procesorami o zmiennych listach instrukcji? W takim procesorze można wgrywać zestawy instrukcji najlepiej dopasowane do realizacji konkretnego zadania. Jak wówczas poradzi sobie programista piszący w asemblerze? Kompilator języka wysokiego poziomu sam może wgrać odpowiednie zestawy instrukcji, wybrać je optymalnie. Oczywiście może to też zrobić programista, ale znowuż pojawia się zagadnienie jego umiejętności.

Jeśli program będzie spełniał stawiane przed nim wymagania, to czy ma to znaczenie w jakim języku jest napisany? A skoro na napisanie i uruchomienie programu w języku C potrzeba znacznie mniej czasu, niż na równoważny w asemblerze, to czy wysiłek ma sens?

Moim zdaniem asembler trzeba poznać, ponieważ jest to język programowania sprzętu, na którym będzie się pracować. Nic lepiej nie uczy zasad działania, niż asembler, ale pisanie w nim całych programów nie ma sensu. Dla mnie na dziś dzień jest to język „wstawek", konieczny do użycia tylko w skrajnych przypadkach. Jest też językiem dobrym (być może) do pisania niewielkich, amatorskich programów. To, że program napisany w języku C będzie potrzebował np. o 300% więcej pamięci, nie jest już w dzisiejszych czasach problemem. Za to czas od pomysłu do uruchomienia i co najważniejsze – sprzedaży, będzie o wiele krótszy. Na dodatek łatwo jest przekazać program komuś innemu, wytłumaczyć jak działa. Struktura jest czytelna i zrozumiała. Osobiście bardzo cenię też programistów, którzy mają wystarczającą wiedzę i wybierają najlepsze narzędzie do rozwiązania konkretnego problemu, nie kierując się uprzedzeniami typu „Basic jest zły, a asembler to najlepszy język programowania", a nie piszą programy o najmniejszym kodzie wynikowym.

**Wysokopoziomowiec**