Piszę w asemblerze, bo...

Jestem pewnie jednym z nielicznych programistów (jeśli można tak dumnie nazwać twórców oprogramowania dla mikrokontrolerów 8-bitowych), którzy nie używają języków wysokiego poziomu, w tym absolutnie obowiązującego każdego prawdziwego programistę języka C. Mimo drwiących uśmieszków kolegów z sąsiednich biurek, ja z uporem maniaka pozostaję na poziomie zero-jedynkowym, ale czemuż miałbym to zmieniać, skoro jak dotąd nie natknąłem się na zagadnienie, którego bym nie rozwiązał używając asemblera. Programowanie na niskim poziomie, w tym przypadku oczywiście nie oznaczające programowania kiepskiego, pozwala mi wycisnąć „siódme poty" z każdego mikrokontrolera, panować nad każdą mikrosekundą wykonywanego kodu. Moje czoło, wbrew podejrzeniom programistów wysokopoziomowych, pozostaje przy tym suche, co świadczy o umiarkowanym co najwyżej wysiłku wkładanym w pisanie programów.

Przeciwnicy asemblera wielokrotnie wyrażali swoją wątpliwość, czy jako autor programu, już w tydzień po jego napisaniu wiem, co dany jego fragment robi. Ja jednak jestem gotów założyć się, że więcej będę wiedział o swoim programie asemblerowym, niż oni o swoich programach napisanych w C. Wynika to przede wszystkim z konsekwentnie umieszczanych przeze mnie komentarzy, nawet we fragmentach o pozornie oczywistym przeznaczeniu. Piszący w języku C, wierząc w przejrzystość tego języka nie zawsze tak robią, działając na swoją zgubę. Próba „rozgryzienia" programu bez komentarzy kończy się często koniecznością długiej jego analizy. Nie chodzi tu nawet o zastosowany algorytm, już same deklaracje zmiennych wymagają niekiedy głębokiego zastanowienia. Jak na ironię przyczyną jest jeden z najmocniejszych mechanizmów języka C, jakim są nawiasy i klamry. Dzięki nim można w jednej linii programu zapisać nawet bardzo skomplikowaną operację, ale można też nieźle się pogubić. Ciekawe ilu C-programistów poradzi sobie bez zaglądania do książek, co na przykład zadeklarowano w linii:

```
int (*(*zmienna)[5][10])(unsigned char);
```

Nie wspomnę już o tym, ilu z nich potrafi podać, jakie wartości osiągną wszystkie zmienne po wykonaniu poniższego fragmentu programu (nie, nie, proszę nie uruchamiać debugera, najpierw analizujemy „na piechotę"):

```
int x=1,y=2,z;
z=(x,x-=y,y=x+=1-y,x--);
```

Powyższe przykłady nie należą do najbardziej wymyślnych, mogę nawet powiedzieć, że to najprostsze zadania, jakie przyszły mi na myśl. Oczywiście zdaję sobie sprawę z tego, że podobną piłeczkę mogą odbić programiści wysokopoziomowi, ale przepychanka nie ma tu większego sensu.

Asembler to swojego rodzaju informatyczny swahili – język (jeśli w ogóle zasługuje na taką nazwę) bardzo prymitywny, ale jak na ironię, to właśnie ta cecha decyduje o jego potędze. Istnieją zresztą elementy upodabniające asembler do języka wysokiego poziomu, takie na przykład jak makra. Dzisiaj asemblery w najprostszej postaci praktycznie nie występują, a **każdy programista używa makroasemblerów**. Pewną niewygodą – tu muszę się zgodzić z moimi adwersarzami – jest różnorodność mnemoników dla poszczególnych mikrokontrolerów/mikroprocesorów. Do dziś wspominam łatwość pisania programów dla mikroprocesora Z80, wyśmiewana już trochę w dobie AVR-ów i ARM-ów '51-ka też jest miła w obsłudze. Gdy zabieram się do programowania nowego mikrokontrolera, muszę najpierw dokładnie przestudiować jego listę rozkazów, tryby adresowania itd., itd. Obowiązek ten – nie zawsze najmilszy, bo lektura datasheet'ów nie jest specjalnie porywająca – daje w efekcie wszechstronną znajomość układu, nad którym będę pracował. Programiści używający języków wysokiego poziomu często z tego etapu sami zwalniają się (czy dobrze robią?), wszak krecią robotę wykona za nich kompilator. Znam przypadki, w których autorzy programów nie mają zielonego pojęcia o tym, jak działa użyty przez nich mikrokontroler, co to jest system przerwań, jak pracuje UART. Hmm, liczy się efekt końcowy. To fakt, ale osobiście miałbym moralnego kaca, gdyby przyszło mi tak pisać programy.

A propos efektu końcowego: dyskusja na temat tego, czy dysponując współczesnymi mikrokontrolerami warto pozostawać przy asemblerze jest jednak trochę jałowa. Z grubsza ten sam rezultat można bowiem osiągnąć zarówno pisząc w asemblerze, jak i w języku C. Może to wymagać w jednym przypadku nieco dłuższego czasu opracowania projektu, w drugim zaś może się wiązać z koniecznością sięgnięcia po bardziej wydajny mikrokontroler. Skutki finansowe są trudne do przewidzenia przy podejmowaniu tematu. Można je oszacować po zakończeniu projektu i wdrożeniu go do produkcji. Broniąc swojej pozycji uważam, że być może mój kod wynikowy, mniejszy o kilka bajtów od kodu wytworzonego przez kompilator języka wysokiego poziomu, pozwoli na zastosowanie układu z mniejszą pamięcią. Zaoszczędzone w ten sposób centy przełożone na tysiące wyprodukowanych egzemplarzy urządzenia dadzą wymierną korzyść w postaci tysięcy dolarów. Chyba warto. Jestem jednak osobą ugodową, skłonną na kompromisy i dlatego uważam, że nie ma wcale wojny między asemblerowcami i C-programistami. Współczesne urządzenia są często tworzone nie przez jednego autora, ale przez całe zespoły. W takim zespole na pewno znajdzie się miejsce dla kogoś, kto będzie odpowiedzialny za krytyczne czasowo funkcje programu, albo zajdzie konieczność optymalizacji programu pod względem wielkości kodu wynikowego (tu człowiek jest nadal ponad najlepszymi kompilatorami). Myślę, że używając tylko asemblera o miejsce pracy nie muszę się martwić.

assemblerowiec