Laboratorium PTM 2

Informacje wstępne:

1. Środowisko *Keil uVision* można w wersji ewaluacyjnej pobrać za darmo ze strony producenta: https://www.keil.com/demo/eval/c51.htm
   Przy pobraniu trzeba podać kilka informacji o sobie – ja się nie obawiam ich podania – pobrałem, rekomenduję by Państwo też sobie to środowisko ściągnęli na swoje komputery. W jednej z rubryk trzeba podać sprzęt, na którym chcemy pracować – tam proszę podać: *80C537*. Ściągną Państwo plik *c51v960a.exe*. To gotowy do zainstalowania produkt. Instalacja przebiega bez kłopotu i sprawnie – też trzeba podać kilka – tych samych – co przy pobieraniu detali o sobie. Ta wersja środowiska jest nowsza niż dostępna w laboratorium – bo na numer 5, w laboratorium jest 3.

2. Chociaż wersja jest nowsza, to opisy, które Państwo znajdą w *Instrukcji Laboratoryjnej* dostępnej na stronie poświęconej *Laboratorium PTM* zasadniczo „pasują", zatem wszelkie czynności konfiguracyjne proszę przeprowadzić według instrukcji – uwaga – potrzebna jest precyzja działań – „jazda na skróty – bo pewnie jest dobrze defaultowo" – prowadzi „w pole i generuje kłopoty". Drobne uwagi:
   - plik projektowy ma rozszerzenie *.uvproj*, a nie *.uv3*,
   - nie dołączamy do projektu pliku *STARTUP.A51* – system o to zapyta,
   - inaczej niż powiedziano w Instrukcji – wybieramy opcję *Use Simulator* w zakładce *Debug*,
   - plik źródłowy musi mieć koniecznie rozszerzenie *.a51* lub *.asm* – ono nie jest dostępne z poziomu rozwijanego menu – trzeba je wpisać „ręką",
   - proszę wszystkie tworzone pliki nazywać „jedno-wyrazowo" i bez polskich znaków,
   - każdy etap tworzenia programu trzeba koniecznie zakończyć „zasejwowaniem" plików – jeśli tego nie zrobimy – kolejny etap jest „spalony" – zrobi się coś przypadkowego zamiast następnego kroku w celu przygotowania programu,
   - każda zmiana w pliku źródłowym wymaga – po „zasejwowaniu" – wykonania kompilacji i linkowania – te czynności „nie robią się same",
   - wynik kompilacji i linkowania musi zakończyć się „bilansem 0:0" w kwestii „errorów i warningów" – komunikaty po zakończeniu kompilacji i linkowania są dostępne w oknie u dołu ekranu – proszę je przeczytać i – w przypadku wskazanych błędów i ostrzeżeń – proszę zrobić stosowne korekty w kodzie źródłowym – a potem „sejwowanie" i ponowna kompilacja z linkowaniem,
   - jak bilans kompilacji i linkowania jest „zero-zero" program jest gotowy do uruchomienia – o czym za chwilę.

3. Szkielet programu – na trywialnym przykładzie – wygląda następująco:

```
ljmp start ; od tej linii zawsze zaczynamy program – chodzi o przeniesienie sterowania do miejsca, gdzie będzie ulokowany nasz program, start to etykieta znakująca początek naszego kodu
org 0100h; dyrektywa lokująca nasz program od adresu 0100h w pamięci kodu programu – miejsce jest dobre – pozwala bezpiecznie rozminąć się z systemem przerwań ulokowanym poniżej 0100h
    start: mov a, #01H; nasz program zaczyna się od adresu 0100h – tu pokazuje etykieta start, do akumulatora ładujemy wartość 01h
    mov r0, #02H; do rejestru r0 ładujemy wartość 02h
    add a, r0; dodajemy zawartość akumulatora i rejestru r0 – wynik jest w akumulatorze
    nop; tu mamy 3 operacje „nic nie rób" – będą się nam przydawały nieco później, ale nabierajmy wprawy w ich wpisywaniu na końcu naszego programu
    nop;
    nop;
    jmp $; to skok "do samego siebie" – etykieta $ będzie powodowała takie zapętlenie skoku – znów niezbędne w kolejnych ćwiczeniach, ale nabieramy przyzwyczajenia
    end start; dyrektywa mówiąca, że kończymy program, który zaczynał się od etykiety start
```

4. Korzystając ze szkieletu programu pokazanego w punkcie 3 w czasie zajęć przygotujemy: napiszemy, skompilujemy-zlinkujemy i uruchomimy proste programy:
    - operacje arytmetyczne: dodawanie, odejmowanie, mnożenie i dzielenie na argumentach 8-bitowych umieszczonych w zasobach rejestrowych procesora: rejestry: *A, B, R0 … R7*, rozkazy – ładowania wartości rejestrów *MOV* i operacji: *ADD, SUBB, MUL, DIV,*
    - operacje arytmetyczne: dodawanie i odejmowanie na argumentach 16-bitowych umieszczonych w zasobach rejestrowych procesora – rejestry są 8-bitowe, zatem operacje trzeba wykonać „na dwa razy" – etapami, a argumenty oraz wyniki gromadzić w parach rejestrów, rozkazy *MOV* oraz *ADD, ADDC, SUBB,*
    - operacje logiczne *and*, *or*, *xor*, *not* na danych 8-bitowych umieszczonych w rejestrach procesora, rozkazy: *MOV, ORL, XRL, ANL, CPL*,
    - ładowanie danych z rejestrów do pamięci danych oraz z pamięci danych do rejestrów procesora – operacja *MOVX*, przy wykonywaniu programu proszę wykonywać podgląd zawartości komórek pamięci, proszę pamiętać o rejestrowym pośrednim trybie adresowania poprzez 16-bitowy rejestr *DPTR*,
    - ładowanie danych z pamięci kodu programu do rejestrów procesora – operacja *MOVC*, przy wykonywaniu programu proszę wykonywać podgląd zawartości komórek pamięci, proszę pamiętać o bazowo-indeksowym trybie adresowania poprzez 16-bitowy rejestr *DPTR* i akumulator *A,*
    - szukanie wartości minimalnej i maksymalnej lub sortowanie bąbelkowe fragmentu pamięci zewnętrznej pamięci danych traktowanej jako tablicy 1-wymiarowej o znanej i założonej liczbie elementów, to zadanie łączy w sobie operacje arytmetyczne, logiczne i dostęp do komórek pamięci.

5. Przyjmujemy zasadę, że w jednym projekcie mamy tylko jeden plik źródłowy – do jednego pliku *.uvproj* podpinamy jeden plik *.a51*. Zatem zadania opisane w punkcie 3 albo robimy pisząc jeden taki program, który w poszczególnych swych etapach realizuje kolejne funkcje i się nie przejmujemy zupełnie, że nie ma między tymi etapami związku logicznego, albo robimy kolejne programy – każdy dedykowany do kolejnego zadania jako odrębny projekt. Pisząc programy posiłkujemy się listą rozkazów dostępną na stronie poświęconej laboratorium – ze zrozumieniem co w danym rozkazie się dzieje. Innymi słowy nie wymyślamy swojej składni i swej koncepcji rozkazu – to absolutnie nie prowadzi do sukcesu.

6. Program prawidłowo napisany, prawidłowo i bezbłędnie skompilowany i zlinkowany uruchamiamy. W tym celu wybieramy opcję *Debug*, a w niej włączamy *Start/Stop Debug Session*. Potwierdzamy następnie komunikat, że jest to wersja ewaluacyjna i mamy program gotowy do dalszej akcji. Oczywiście *Instrukcja Laboratoryjna* opisuje te kroki w detalu – proszę do niej zaglądnąć. Po lewej stronie ekranu widzimy podgląd rejestrów procesora, w zakładce *View* możemy na bieżąco wybierać – włączać/wyłączać inne podglądy innych zasobów, które są nam potrzebne. Realizując zadania ćwiczenia pierwszego będziemy uruchamiali te proste programy tylko w trybie pracy krokowej. Zatem po kiedy już wykonaliśmy *Start Debug* kolejne rozkazy tworzące kod naszego programu uruchamiamy krok-po-kroku – klawisz *F11* jest odpowiedzialny za pracę krokową. Zadanie polega na obserwowaniu w podglądzie rejestrów i komórek pamięci co się tam dzieje przy wykonaniu każdej instrukcji i weryfikowanie tym samym czy widoczne efekty są zgodne z tym co stać się powinno. Gdy nasz program dotrze do miejsca, gdzie mamy operacje *NOP* – to znaczy, że merytoryczna jego część została wykonana i jest to dobry moment by posługując się *Start/Stop Debug Session* program wyłączyć.