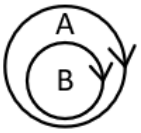
Program wielowątkowy z wywłaszczaniem
Instrukcja do ćwiczeń

2019.8.9

Wstęp

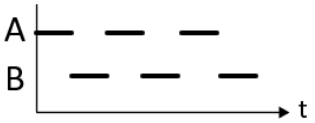
Celem ćwiczenia jest wyjaśnienie podstawowej zasady równoległego wykonywania dwóch (wątków) na procesorach z jedną jednostką wykonawczą. Wątki będą miały postać pętli nieskończonych.

W programach jednowątkowych program zawsze w danym momencie wykonuje jedną pętlę, która ewentualnie może być pętlą zagnieżdżoną, czyli wykonanie pętli wew. zależy od pętli zew. i vice versa.



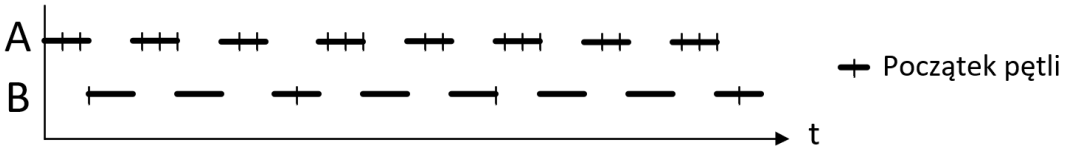
W poniższym przykładzie pętle będą się wykonywać równolegle.

Zostanie to osiągnięte przez wywłaszczanie, tzn., że co pewien czas procesor będzie przełączał się z wykonywania jednej pętli na wykonywanie drugiej pętli.



Przełączanie będzie odbywać się ze stałą częstotliwością a co za tym idzie w sposób asynchroniczny względem pętli. Określenie asynchroniczny oznacza, że moment przełączenia w żaden sposób nie zależy od fazy wykonania pętli, np. że przełączenie między pętlami odbywa się zawsze na końcu pętli.

W zależności od częstotliwości przełączania oraz okresu obiegu poszczególnych pętli przełączenie może odbywać się kilka razy w czasie jednego obiegu pętli albo co kilka obiegów pętli. W tym drugim przypadku jest b. mało prawdopodobne, że będzie to dokładna wielokrotność, np. przełączenie co 5 obiegów pętli.



Jedynym sposobem na przerwanie wykonywania pętli (wywłaszczenie) jest mechanizm przerwania. W poprzednich ćwiczeniach po wykonaniu przerwania program wracał do wykonywania pętli dokładnie w to samo miejsce w którym wystąpiło przerwanie (a dokładnie do następnej instrukcji). Działo się tak dzięki zapamiętaniu na stosie adresu powrotu.

Mechanizm ten można wykorzystać do przełączania między pętlami. W tym celu, w procedurze obsługi przerwania, należy podmienić adres powrotu znajdujący się na stosie.

Czyli jeżeli przerwanie nastąpiło w trakcie wykonywania pętli A to należy:

1. Zdjąć ze stosu i zapamiętać (np. zapisać do rejestru) adres powrotu (do pętli A)
2. Wstawić na stos adres powrotu do pętli B

Jeżeli natomiast przerwanie nastąpiło w trakcie wykonywania pętli B to należy:

1. Zdjąć ze stosu i zapamiętać (np. zapisać do rejestru) adres powrotu (do pętli B)
2. Wstawić na stos adres powrotu do pętli A

Ćwiczenia

Ćwiczenie 1.
Korzystając z programu z AVR_1 napisać program, który będzie się składał pętli głównej oraz procedury obsługi przerwania uruchamianej co 100 cykli zegara głównego (preskaler równy 1). Pętla główna powinna składać się z pięciu instrukcji „nop". Procedura obsługi przerwania powinna składać się z jednej instrukcji „nop". Etykieta pętli głównej powinna wyglądać nast.: „ThradA".

UWAGA: Program powinien zawierać tylko i wyłącznie kod niezbędny do realizacji

Działanie programu sprawdzić symulatorem sprawdzając co ile cykli następuje wejście do procedury obsługi przerwania. (powinno być ok. 100 cykli)

Ćwiczenie 2.
Wstawić na początek kodu programu zmienną rejestrową CurrentThread w sposób nast. „`.def CurrentThread=R20`"
i dodać jej zerowanie do inicjalizacji programu.
Wstawić na początek procedury obsługi przerwania kod, który będzie negował TYLKO najmłodszy bit zmiennej `CurrentThread`. W tym celu użyć instrukcji inkrementacji oraz iloczynu bitowego.

Działanie programu sprawdzić symulatorem.

Ćwiczenie 3.

a) Wstawić na początek kodu programu zmienne rejestrowe `ThreadA_LSB i ThreadA_MSB` oraz dodać do programu inicjalizację dodanych zmiennych adresem pętli ThreadA. W tym celu użyć wbudowanych funkcji assemblera „LOW" i „HIGH".

Sprawdzić poprawności przypisania adresu do zmiennych używając symulatora oraz przeglądając zawartość plików „.map" i „.lss" z podkatalogu „Debug" katalogu projektu.

b) Sprawdzić czy przerwanie działa asynchronicznie względem wątku ThreadA. W tym celu sprawdzić, w które miejsce wątku następuje powrót z procedury obsługi programu. W tym celu ustawić brakepointa na instrukcji powrotu z przerwania. Po zatrzymaniu na brakepoincie wykonać jeden krok. Zwolnić program. Poczekać na zatrzymanie na brakepincie, Itd.. Zwrócić uwagę czy powrót do wątku następuje w przypadkowych miejscach.

c) Wstawić do procedury obsługi przerwania fragment kodu, który będzie ustawiał adres powrotu zawsze na adres początku wątku ThradA. W tym celu należy użyć instrukcji do operacji na stosie oraz zmiennych z punktu a).

Test przeprowadzić analogicznie jak w punkcie b)

Ćwiczenie 4.

a) Dodać do programu wątek ThreadB dokładnie taki sam jak ThreadA. Ponadto dodać do programu zmienne rejestrowe przechowujące adres ThreadB oraz ich inicjalizacje (nazwy analogiczne do ThreadA)

Sprawdzić poprawność inicjalizacji.

b) Zmodyfikować procedurę obsługi przerwania tak aby zawsze wracała na początek ThreadB.

Sprawdzić analogicznie jak w 3b).

c) Zmodyfikować procedurę obsługi przerwania tak aby realizowała scenariusz przełączania wątków przedstawiony pod koniec wstępu. Należy wykorzystać najmłodszy bit zmiennej „`CurrentThread`" (patrz ćw. 2). W celu łatwiejszej weryfikacji równoległej pracy wątków zastąpić instrukcje „nop" instrukcjami „inc CtrA" i „inc CtrB" (utworzyć odpowiednie zmienne rejestrowe).

Weryfikację przeprowadzić wstawiając brakepointa na początek procedury obsługi przerwania i krokując program do wyjścia do momentu wyjścia procedury. (sprawdzić gdzie wraca)

Weryfikację przeprowadzić zatrzymując co pewien czas program uruchomiony bez brakepointów (nie „stop debugging") i obserwując wartości zmiennych CtrA i CtrB.

Ćwiczenie 5.

Zmodyfikować program tak aby wątki ThradA i ThradB powodowały miganie cyframi odpowiednio zerową i pierwszą. Należy użyć dwóch funkcji delay pracujących na różnych rejestrach. (obecna wersja programu nie zapewnia ochrony rejestrów wątków).