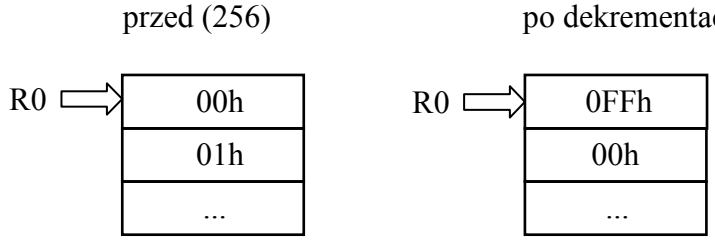
Zadanie 1

Dekrementacja liczby dwubajtowej w pamięci wewnętrznej (IRAM)

Wejście: R0 - adres młodszego bajtu (Lo) liczby

dec_iram:

Liczba dwubajtowa umieszczona jest w pamięci wewnętrznej IRAM, pierwszy bajt mniej znaczący. Procedura powinna zmniejszyć wartość liczby o 1, na przykład:



Można wykorzystać rozkaz odejmowania (*subb*) lub dekrementacji (*dec*) pamiętając, że dekrementacja nie ustawia CY (trzeba wykryć zmianę młodszego bajtu 00h → 0FFh).

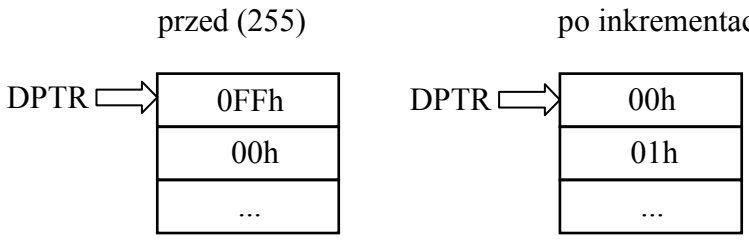
Zadanie 2

Inkrementacja liczby dwubajtowej w pamięci zewnętrznej (XRAM)

Wejście: DPTR - adres młodszego bajtu (Lo) liczby

inc_xram:

Liczba dwubajtowa umieszczona jest w pamięci zewnętrznej XRAM, pierwszy bajt mniej znaczący. Procedura powinna zwiększyć wartość liczby o 1, na przykład:



Można wykorzystać rozkazy dodawania (*add*/*addc*) lub inkrementacji (*inc*) pamiętając, że inkrementacja nie ustawia CY (trzeba wykryć zmianę młodszego bajtu 0FFh → 00h).

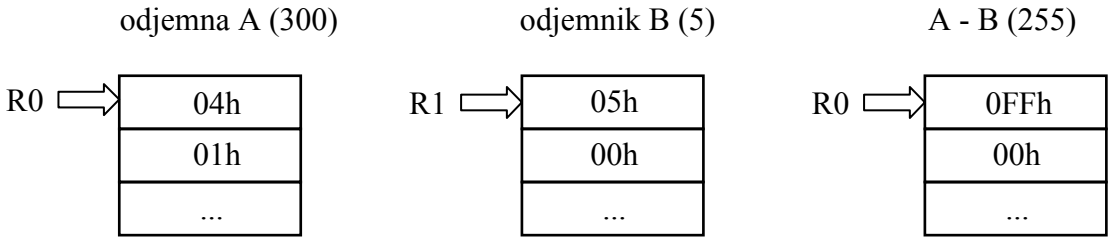
Zadanie 3

Odjęcie liczb dwubajtowych w pamięci wewnętrznej (IRAM)

Wejście: R0 - adres młodszego bajtu (Lo) odjemnej A oraz różnicy (A ← A - B)
R1 - adres młodszego bajtu (Lo) odjemnika B

sub_iram:

Odjemna i odjemnik (liczby dwubajtowe) umieszczone są w różnych obszarach pamięci wewnętrznej IRAM, pierwszy bajt mniej znaczący. Procedura powinna wykonać odejmowanie i umieścić wynik w miejscu odjemnej, na przykład:



Trzeba pamiętać, że (w przeciwieństwie do dodawania) dysponujemy tylko rozkazem *subb*, wykonującym odejmowanie z uwzględnieniem pożyczki (przechowywanej w CY).

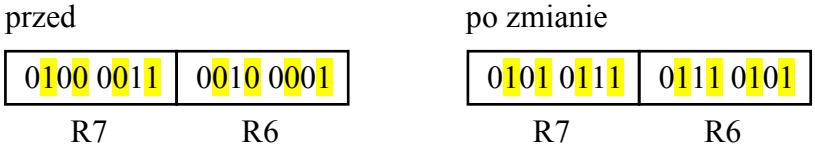
Zadanie 4

Ustawienie bitów parzystych (0, 2, ..., 14) w liczbie dwubajtowej

Wejście: R7|R6 - liczba dwubajtowa
Wyjście: R7|R6 - liczba po modyfikacji

set_bits:

W przekazanej w parze rejestrów R7|R6 dwubajtowej liczbie powinny zostać ustawione wszystkie bity parzyste (0, 2, ..., 14). Inne bity nie powinny być zmieniane, na przykład:



Wykorzystać rozkaz *orl*.

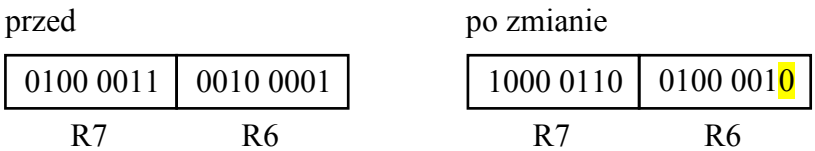
Zadanie 5

Przesunięcie w lewo liczby dwubajtowej (mnożenie przez 2)

Wejście: R7|R6 - liczba dwubajtowa
Wyjście: R7|R6 - liczba po modyfikacji

shift_left:

Przekazana w parze rejestrów R7|R6 dwubajtowa liczba powinna zostać przesunięta o jeden bit w lewo, bit 0 powinien być wyzerowany co odpowiada mnożeniu przez 2, na przykład:



Przesunięcie można wykonać wykorzystując rozkaz *rlc*, zaczynając od młodszego bajtu i zerując wcześniej CY.

Zadanie 6

Pobranie liczby dwubajtowej z pamięci kodu

Wejście: DPTR - adres młodszego bajtu (Lo) liczby w pamięci kodu
Wyjście: R7|R6 - liczba dwubajtowa

get_code_const:

Procedura powinna umożliwić pobranie dwubajtowej liczby z pamięci kodu (stałe dane) do pary rejestrów R7|R6. Wykorzystać rozkaz *movc*.

Dane w pamięci kodu można umieścić następująco (na końcu programu):

```
code_const:
        db      LO(1234h)
        db      HIGH(1234h)
```

Dyrektywy asemblera *LO* i *HIGH* pozwalają wydzielić z wartości 16-bitowej młodszą i starszą część. Równoważny byłby zapis:

```
code_const:
        db      34h
        db      12h
```

Etykieta *code_const* jest 16-bitowym adresem w pamięci kodu. Przed wywołaniem procedury powinna być ona użyta do ustawienia wartości parametru w DPTR:

```
        mov     DPTR, #code_const
```

Zadanie 7

Zamiana wartości rejestrów DPTR i R7|R6
Nie niszczy żadnych rejestrów

swap_regs:

Procedura powinna zamienić ze sobą dwie liczby dwubajtowe z rejestrów DPTR i R7|R6, nie powinny być niszczone żadne rejestry.

Rejestr DPTR składa się z dwóch 8-bitowych części DPH|DPL, do których jest osobny dostęp. Do zamiany trzeba wykorzystać rozkaz *xch*, ponieważ musi on używać akumulatora, trzeba go przechować na stosie (zakładamy w tym przypadku, że procedura nie niszczy rejestrów).

W rozkazach, w których używany jest symbol A, oznacza on akumulator jako specjalny, uprzywilejowany rejestr. Symbol ACC reprezentuje natomiast adres akumulatora w obszarze SFR. Ponieważ argumentem rozkazów *push/pop* może być tylko adres (*direct*), musimy użyć symbolu ACC.

Zadanie 8

Dodanie 10 do danych w obszarze pamięci zewnętrznej (XRAM)

```
Wejście:    DPTR  - adres początku obszaru
            R2    - długość obszaru
```

add_xram:

Procedura powinna zwiększyć o 10 wszystkie komórki pamięci XRAM o początku wskazywanym przez DPTR i długości podanej w R2. Przy dodawaniu dopuszczalne jest przepełnienie (dodanie 10 do 255 da wynik 9).

W pierwszej wersji można przyjąć, że długość będzie zawsze niezerowa, następnie wprowadzić odpowiednie zabezpieczenie.

Wykorzystanie szablonu kodu

Każde z zadań powinno być wykonane w postaci procedury, większość z procedur będzie używała parametrów, które powinny być ustalone przed wywołaniem. W pierwszej części kodu powinny być zawarte testowe wywołania procedur.

W pliku *ptm_lab_1n.asm* znajduje się szablon kodu, zawierający puste procedury (ramka komentarza, etykieta z nazwą procedury oraz instrukcja *ret*). Trzeba wypełnić te procedury kodem, realizującym zadania zgodne z opisem procedury.

Do testów procedur można użyć własnego kodu lub skorzystać z przykładowych wywołań, dowolnie modyfikując parametry. Procedura może być wywołana jednokrotnie, potem zatrzymanie programu w pętli, na przykład:

```
        mov   R0, #30h      ; liczba w komórkach pamięci 30h i 31h
        lcall dec_iram      ; wywołanie procedury
        sjmp  $             ; pętla bez końca
```

Jeśli chcemy w wygodny sposób przetestować procedurę dla różnych danych można powtarzać ją w pętli i przed każdym wywołaniem (w pracy krokowej lub z wykorzystaniem pułapki) ustawiać w pamięci (lub rejestrach, zależnie od procedury) inne dane, na przykład:

```
loop:   mov   R0, #30h
        lcall dec_iram
        sjmp  loop          ; powtarzamy
```

Można też wykonywać testy kilku procedur, jedna po drugiej. Wszystkie testowe wywołania mogą pozostawać w kodzie. Przy testowaniu jednej, wybranej procedury inne mogą być umieszczone w komentarzach, lub (co jest wygodniejsze) można użyć skoku aby je ominąć. Na przykład procedura *dec_iram* została już sprawdzona a obecnie testujemy *inc_xram*:

```
        ljmp  test

        mov   R0, #30h      ; liczba w komórkach IRAM 30h i 31h
        lcall dec_iram      ; wywołanie procedury
        sjmp  $             ; zapętlenie w bieżącym rozkazie

test:   mov   DPTR, #8000h  ; liczba w komórkach XRAM 8000h i 8001h
        lcall inc_xram      ; wywołanie procedury
        sjmp  $             ; zapętlenie w bieżącym rozkazie

        ...                 ; inne procedury, do których nie dojdziemy
```

Testując kod trzeba pamiętać, aby uwzględnić różne, charakterystyczne przypadki danych.

Ważne uwagi

1. Nie zmieniamy założeń co do działania procedur i układu parametrów w rejestrach,
2. Ramki komentarzy opisujących procedurę pozostawiamy bez zmian,
3. Używamy podanych nazw (etykiet) procedur, nie zmieniamy ich.

Formatowanie kodu

1. Etykiety bez wcięcia (zawsze od pierwszej kolumny),
2. Kod zawsze z wcięciem, preferowane jednolite wcięcie dla całego kodu,

3. Między kodem rozkazu a ewentualnymi parametrami lepiej wstawić tabulację a nie spację, kod jest wtedy bardziej przejrzysty,
4. Komentarze z prawej strony rozkazów powinny być wyrównane,
5. Zalecany rozmiar tabulacji to 8 znaków (jest to domyślne ustawienie Keil µVision dla plików asemblerowych).

Link do pliku z szablonem kodu:

http://staff.iiar.pwr.wroc.pl/antoni.sterna/ptm/zadania/ptm_lab_1n.asm