Zadanie 1

Sumowanie bloku danych w pamięci wewnętrznej (IRAM)

Wejście: R0 - adres początkowy bloku danych
R2 - długość bloku danych

Wyjście: R7|R6 - 16-bit suma elementów bloku (Hi | Lo)

---

sum_iram:

Na początku procedury suma, która ma być zwrócona w parze rejestrów R7|R6, powinna być wyzerowana, nie może to być zrzucone na kod wywołujący.

Pobieramy kolejne bajty danych z pamięci IRAM i dodajemy do młodszego bajtu sumy. Jeśli przy dodawaniu wystąpiło przeniesienie, musi być zwiększony starszy bajt sumy.

Przykłady:

IRAM: 01 02 03 04 wynik: R7|R6 = 00|0Ah (10)
01 02 FF FF R7|R6 = 02|01h (513)

Zadanie 2

Kopiowanie bloku danych w pamięci wewnętrznej (IRAM) z odwróceniem

Wejście: R0 - adres początkowy obszaru źródłowego
R1 - adres początkowy obszaru docelowego
R2 - długość kopiowanego obszaru

---

copy_iram_iram_inv:

Na początku procedury trzeba zmodyfikować R1 tak aby wskazywał na ostatni bajt obszaru docelowego. Po skopiowaniu bieżącego bajtu adres obszaru źródłowego powinien być zwiększany a adres obszaru docelowego zmniejszany.

Przykład:

IRAM: 01 02 03 04 obszar docelowy: 04 03 02 01

Zadanie 3

Kopiowanie bloku danych z pamięci zewnętrznej (XRAM) do wewnętrznej (IRAM)
Przy kopiowaniu powinny być pominięte elementy zerowe

Wejście: DPTR - adres początkowy obszaru źródłowego
R0 - adres początkowy obszaru docelowego
R2 - długość kopiowanego obszaru

---

copy_xram_iram_z:

Pobieramy kolejne bajty z obszaru źródłowego i sprawdzamy czy są różne od zera:

- jeśli tak to kopiujemy i zwiększamy adres obszaru docelowego
- jeśli nie, idziemy dalej
- zawsze zwiększamy adres źródłowy

Przykłady:

XRAM:	01 02 03 04	01 00 00 04
IRAM:	01 02 03 04	01 04 ?? ??

Zadanie 4

Kopiowanie bloku danych w pamięci zewnętrznej (XRAM → XRAM)

Wejście:	DPTR	- adres początkowy obszaru źródłowego
	R1|R0	- adres początkowy obszaru docelowego
	R2	- długość kopiowanego obszaru

---

copy_xram_xram:

Pobieramy kolejne bajty z obszaru źródłowego i zapisujemy do obszaru docelowego.

Wskazówki:

Rozkazy umożliwiające dostęp do pamięci XRAM:

movx	A, @R0/1	movx	@R0/1, A
movx	A, @DPTR	movx	@DPTR, A

Wariant z adresowaniem 8-bit przez rejestry R0/R1 pozwala na zapis/odczyt tylko w ramach bloku pamięci XRAM o długości 256 bajtów. Starszy adres musi być wtedy ustawiany ręcznie przez port P2. Inkrementacja adresu wiązałaby się wtedy z wykrywaniem przepełnienia młodszej części i ewentualną inkrementacją starszego adresu na P2. Dlatego do kopiowania w tym zadaniu nadaje się praktycznie tylko wariant z 16-bit adresem w DPTR.

Przy kopiowaniu XRAM → XRAM w DPTR powinien na przemian znajdować się adres źródła lub celu. Można to osiagnac przez zamianę DPTR ↔ R1|R0 (oczywiście w taki sposób, aby żaden z tych adresów nie został zniszczony). Inkrementację adresu docelowego w R1|R0 najlepiej wykonać wtedy, gdy znajduje się on w DPTR.

Zadanie 5

Zliczanie w bloku danych w pamięci wewnętrznej (IRAM) liczb parzystych > 10

Wejscie:	R0	- adres początkowy bloku danych
	R2	- długość bloku danych
Wyjscie:	A	- liczba elementów spełniających warunek

---

count_even_gt10:

Zerowanie licznika musi być wykonane wewnątrz procedury a nie w kodzie wywołującym.

Pobieramy kolejne bajty z pamięci i sprawdzamy czy wartość >10:

- jeśli nie, idziemy dalej
- jeśli tak, sprawdzamy czy liczba jest parzysta
  - jeśli nie, idziemy dalej
  - jeśli tak, zwiększamy licznik

W rezultacie licznik powinien być zwiększany w kodzie tylko w jednym miejscu (które będzie po prostu omijane jeśli testowana liczba nie spełnia zadanego warunku).

Wskazówki:

Liczby parzyste mają najmłodszy bit równy 0. Jeśli testowana dana będzie umieszczona w A, to bit ten można przetestować bezpośrednio w A (*jb*/*jnb*) lub przesuwając go do CY (*jc*/*jnc*). Testowanie parzystości jako reszty z dzielenia przez 2 nie jest efektywnym rozwiązaniem.

Do sprawdzenia, czy liczba jest >10, można wykorzystać rozkaz *cjne* albo *subb* (trzeba pamiętać, że przy odejmowaniu uwzględniana jest również pożyczka w CY).

Przykłady:

Dane w pamięci w formacie hex.

IRAM:	01 02 03 04	wynik:	A = 00h (0)	wszystkie ≤10
	01 02 13 14		A = 01h (1)	dwie >10, jedna z nich parzysta
	11 12 13 14		A = 02h (2)	wszystkie >10, dwie parzyste
	10 12 14 16		A = 04h (4)	wszystkie >10 i parzyste
	11 13 15 17		A = 00h (0)	wszystkie >10 ale nieparzyste

Uwaga

We wszystkich procedurach jednym z parametrów jest długość obszaru danych. W pierwszym, roboczym wariancie można przyjąć, że będzie ona zawsze niezerowa. Wskazane byłoby jednak dodanie później dodanie odpowiedniego zabezpieczenia dla przypadku zerowej długości.

Wykorzystanie szablonu kodu

W pliku *ptm_lab_2n.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. Ponieważ każdy fragment kodu kończy się zapętleniem (*sjmp* $), wykonany zostanie tylko pierwszy z nich.

Chcąc testować tylko jedną, wybraną procedurę wystarczy przenieść jej kod testowy na początek programu. Można również przy jednym uruchomieniu programu testować po kolei kilka procedur usuwając (lub komentując) odpowiednie pętle po wywołaniach procedur.

Testując kod trzeba pamiętać, aby uwzględnić różne, charakterystyczne przypadki danych. Na przykład dla zliczania liczb parzystych > 10 trzeba zweryfikować działanie dla liczb < 10, = 10 oraz > 10 (parzystych i nieparzystych).

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_2n.asm