PODSTAWY TECHNIKI MIKROPROCESOROWEJ

Laboratorium 2

Autorzy:	Dawid Waligórski Adrian Kotula
Termin zajęć:	Środa, TP, godz. 13.15

W celu poprawy czytelności umieszczonych w sprawozdaniu listingów usunięto z nich komentarze. W pełni okomentowany kod, napisany na potrzeby laboratorium, zamieszczono w załączniku do tego sprawozdania.

1 Zadanie 1:

Celem zadania 1 było napisanie procedury sumującej blok danych w pamięci zewnętrznej (XRAM). W momencie wywołania procedury w rejestrze DPTR miał znajdować się adres początkowy sumowanego bloku danych. W rejestrze R2 zawarta miała być długość tego bloku.

Napisana procedura w pierwszej kolejności sprawdzała, czy zadany do niej blok danych nie miał zerowej długości. W wypadku, gdy długość bufora była zerowa, nastąpić miał powrót z procedury.

Następnym krokiem było wyczyszczenie rejestrów R6 oraz R7, które miały przechowywać kolejno młodszy i starszy bajt wyliczanej sumy. Następnie dodawano do młodszego bajtu sumy kolejne bajty pobrane z zadanego bufora i adresowane przez DPTR. Następnie do starszego bajtu dodawano przeniesienie wynikające z wykonanego dodawania.

Listing 1: Kod z zadania 1

```
sum_xram:
        MOV A, R2
        JZ reached_sum_buffers_end

        MOV R6, #0
        MOV R7, #0

add_next:
        MOVX A, @DPTR
        ADD A, R6
        MOV R6, A

        MOV A, R7
        ADDC A, #0
        MOV R7, A

        INC DPTR
        DJNZ R2, add_next

reached_sum_buffers_end:
        RET
```

2 Zadanie 2:

Celem zadania 2 było napisanie procedury kopiującej zadany blok danych z pamięci zewnętrznej (XRAM) do pamięci wewnętrznej (IRAM). Elementy miał zostać umieszczone w pamięci IRAM w kolejności odwrotnej niż ta w pamięci XRAM. W momencie wywołania procedury w rejestrze DPTR miał znajdować się adres początkowy kopiowanego bloku danych. W rejestrze R2 zawarta miała być długość tego bloku. W rejestrze R0 miał znajdować się adres, od którego miał zaczynać się skopiowany blok danych.

Napisana procedura w pierwszej kolejności sprawdzała, czy zadany do niej blok danych z XRAM nie miał zerowej długości. W wypadku, gdy długość bufora była zerowa, nastąpić miał powrót z procedury.

Następnie następowało przesunięcie wskaźnika obszaru docelowego o tyle bajtów, ile wynosiła długość kopiowanego bufora. Dzięki temu można było skopiować bufor zgodnie z założeniami zadania. Wartość w DPTR adresująca w XRAM zwiększała się, kopiując zawartość bufora do komórek adresowanych przez wartości w R0, które stale się zmieszały. Sytuacja odwrotna, ze zmniejszającym się DPTR i zwiększającym się R0 byłaby znacznie trudniejsza w implementacji ze względu na brak rozkazu dekrementującego zawartość DPTR.

Listing 2: Kod z zadania 2

```
copy_xram_iram_inv:
        MOV A, R2
        JZ reached_cpy_xram_buffers_end

        ADD A, R0
        DEC A
        MOV R0, A

copy_next_xram:
        MOVX A, @DPTR
        MOV @R0, A

        DEC R0
        INC DPTR
        DJNZ R2, copy_next_xram

reached_cpy_xram_buffers_end:
        RET
```

3 Zadanie 3:

Celem zadania 3 było napisanie procedury kopiującej zawartość zadanego bufora z pamięci wewnętrznej (IRAM) do pamięci zewnętrznej (XRAM). Elementy zerowe miały zostać pominięte. W momencie wywoływania procedury w rejestrze R0 miał znajdować się adres początkowy kopiowanego bloku danych. W rejestrze R2 zawarta miała być długość tego bloku. W rejestrze DPTR miał znajdować się adres, od którego miał zaczynać się skopiowany blok danych.

Napisana procedura w pierwszej kolejności sprawdzała, czy zadany do niej blok danych z IRAM nie miał zerowej długości. W wypadku, gdy długość bufora była zerowa, nastąpić miał powrót z procedury.

Następnie kolejne bajty bufora z IRAM kopiowano do rozpoczynającego się od wskazanego adresu obszaru w XRAM. Sprawdzano, czy kopiowana wartość była niezerowa. W wypadku niespełnienia tego warunku nie kopiowano jej zawartości do XRAM. W przeciwnym wypadku wykonywano kopię

Listing 3: Kod z zadania 3

```
copy_iram_xram_z:
        MOV A, R2
        JZ reached_cpy_iram_buffers_end

copy_next_iram:
        MOV A, @R0
        JZ iram_value_is_zero

        MOVX @DPTR, A
        INC DPTR

iram_value_is_zero:
        INC R0
        DJNZ R2, copy_next_iram

reached_cpy_iram_buffers_end:
        RET
```

4 Zadanie 4:

Celem zadania 4 było napisanie procedury kopiującej zawartość zadanego bufora z pamięci XRAM do innego obszaru pamięci wewnętrznej. Elementy niezerowe miały zostać skopiowane dwukrotnie. W momencie wywoływania procedury w rejestrze DPTR miał znajdować się adres początkowy kopiowanego bloku. W rejestrze R2 zawarta miała być długość tego bloku. W rejestrach R1 i R0 miały znajdować się kolejno starsza i młodsza część adresu początkowego obszaru XRAM, do którego skopiowany miał zostać bufor.

Napisana procedura w pierwszej kolejności sprawdzała, czy zadany do niej blok danych z XRAM nie miał zerowej długości. W wypadku, gdy długość bufora była zerowa, nastąpić miał powrót z procedury.

Następnie kolejne bajty z bufora wejściowego kopiowano do obszaru docelowego w pamięci zewnętrznej. Po wykonaniu pierwszej kopii sprawdzano, czy kopiowana wartość była niezerowa. Jeżeli była, zapisywano jej drugą kopię w kolejnej komórce obszaru docelowego.

Istotnym problemem w tym zadaniu była również inkrementacja adresu w

obszarze docelowym, który był przechowywany w parze rejestrów R1|R0. Początkowo zaimplementowano rozwiązanie polegające na wczytaniu tego adresu do rejestru DPTR, który można było już łatwo zinkrementować. Oczywiście wymagało to tymczasowego zapamiętania zapisanego tam uprzednio adresuz obszaru źródłowego na stosie (inaczej zostałby on bezpowrotnie utracony). To rozwiązanie umieszczono na listingu 4.

Z pomocą prowadzącego laboratorium udało się jednak znaleźć bardziej optymalne rozwiązanie problemu inkrementacji adresu docelowego. Zamiast przenoszenia pary R1|R0 do DPTR, napisano prosty zestaw instrukcji inkrementujących liczbę dwubajtową zawartą w parze P2|R0. To rozwiązanie umieszczono na listingu 6.

Listing 4: Kod z zadania 4 - inkrementujemy DPTR

```
copy_xram_xram_2:
        MOV A, R2
        JZ reached_xram_end_2

copy_next_xram_to_xram:
        MOVX A, @DPTR
        MOV P2, R1
        MOVX @R0, A

        LCALL increment_r1r0_pointer

        JZ check_cpy_next
        MOV P2, R1
        MOVX @R0, A
        LCALL increment_r1r0_pointer

check_cpy_next:
        INC DPTR
        DJNZ R2, copy_next_xram_to_xram

reached_xram_end_2:
        RET
```

```
increment_r1r0_pointer:
        PUSH DPH
        PUSH DPL

        MOV DPH, R1
        MOV DPL, R0

        INC DPTR

        MOV R1, DPH
        MOV R0, DPL

        POP DPL
        POP DPH

        RET
```

Listing 5: Kod z zadania 4 - tylko inkrementujemy parę P2|R0

```
copy_xram_xram_2:
        MOV A, R2
        JZ reached_xram_end_2

        MOV P2, R1

copy_next_xram_to_xram:
        MOVX A, @DPTR
        MOVX @R0, A

        LCALL increment_r1r0_pointer

        JZ check_cpy_next
        MOVX @R0, A
        LCALL increment_r1r0_pointer

check_cpy_next:
        INC DPTR
        DJNZ R2, copy_next_xram_to_xram

reached_xram_end_2:
        RET
```

```
increment_r1r0_pointer:
        INC R0
        CJNE R0, #0, end_increment
        INC P2

        RET
```

5 Zadanie 5:

Celem zadania 5 było napisanie procedury zliczającej liczby z przedziału [10;100] w zadanym buforze w pamięci wewnętrznej (IRAM). Rolę licznika pełnić miał akumulator. W momencie wywołania procedury w rejestrze R0 miał znajdować się adres początkowy bufora. W rejestrze R2 zawarta miała być jego długość.

Napisana procedura w pierwszej kolejności sprawdzała, czy zadany do niej blok danych z IRAM nie miał zerowej długości. W wypadku, gdy długość bufora była zerowa, nastąpić miał powrót z procedury.

Następnie zerowany był licznik. Później dla każdego bajtu w buforze sprawdzano czy nie narusza kolejno dolnej oraz górnej granicy dozwolonego przedziału. W wpadku naruszenia jednego z tych ograniczeń następowało bezpośrednie przejście do kolejnej wartości z bufora. W przeciwnym wypadku przed przejściem do kolejnej wartości wykonywano inkrementacje licznika.

Do porównywania zawartości komórek bufora z granicami przedziału wykorzystano zmienianą przez rozkaz CJNE flagę carry. Pozwalała ona określić, czy wartość z bufora była mniejsza od granicy (CY=1) lub większa od granicy (CY=0).

Listing 6: Kod z zadania 5

```
count_range:
        MOV A, R2
        JZ reached_count_end

        CLR A

analyze_next:
        CJNE @R0, #10, compared_to_10

compared_to_10:
        JC check_loop_condition
        CJNE @R0, #101, compared_to_100

compared_to_100:
        JNC check_loop_condition
        INC A

check_loop_condition:
        INC R0
        DJNZ R2, analyze_next

reached_count_end:
        RET
```