Wrocław, dn. 4 czerwca 2016r

Jakub Pomykała, 209897
PN-P-8

prowadzący: prof. Janusz Biernat

Laboratorium Architektury Komputerów
(5) Zapoznanie się z jednostkami wektorowymi rodziny x86

1 Treść ćwiczenia

Zakres ćwiczenia:

- Zapoznanie się z formatem plików **.bmp*
- Zapoznanie się z jednostkami wektorowymi *mmx*

Zrealizowane ćwiczenia:

- Obrót obrazu w formacie **.bmp*
- Negacja obrazu **.bmp*
- Negacja obrazu **.bmp* przy użyciu *mmx*

2 Wstęp teoretyczny

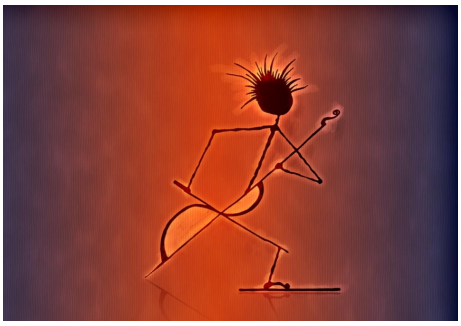
Negatyw obrazu to odwrócenie wartości kolorów skladowych każdego piksela. Piksele w pliku są zapisywyane po kolei od lewego dolnego rogu. Na jeden piksel składają się 24-bity w formacie BGR. Na jeden kolor przypada zatem 1 bajt. Na końcu każdego wiersza dopisywana jest odpowiednia ilość (od 1 do 3) bajtów zerowych tak, aby iloczyn szerokości i ilości bajtów przypadających na piksel był podzielny przez 4. Wartość bajtów dopełniających nie jest istotna, ponieważ większość programów przeważnie je ignoruje. W przypadku obrazów 8-bitowych, każdy piksel jest oznaczony numerem koloru z opcjonalnej palety kolorów, która znajduje się między danymi obrazu, a nagłówkiem.

3 Przebieg ćwiczenia

3.1 Obrót obrazu

Funkcja obrotu obrazu została napisana w języku C, przy użyciu funkcji `malloc`. Przyjmuje ona 3 argumenty: dane z informacjami o pikselach, wysokość oraz szerokość. Pętla for realizuje funkcję `swap`, czyli zamieniane są miejscami kolejne wiersze z pikselami w pliku *.bmp. Informacje o szerokości i wysokości zostały pobrane z nagłówka, którego odczyt został napisany w języku `assembler`.

```
void rotate(char *data, int width, int height) {
   //alokujemy pamiec na jeden wiersz
   char *tmpBuffer = (char*) malloc(3 * width);

   // długość wiersza do skopiowania z uwzględnieniem bitów dopełnienia
   int lenghtToCopy = 3 * (width) + width % 4;
   int i, offset_new, offset_old;
   for(i = 0; i < height/2; i++) {
       // offset określający wiersz do skopiowania
      offset_old = (lenghtToCopy * (i + 1));
      //stary wiersz do tmpBuffer
      memcpy(tmpBuffer, data + offset_old, lenghtToCopy);

      // offset określający wiersz, w którym ma pojawić się nowy ciag pikseli
      offset_new = ((height - i) * lenghtToCopy);
      //zastąpienie "dolnego wiersza" "górnym wierszem"
      memcpy(data + offset_old, data + offset_new, lenghtToCopy);

      //skopiowanie wiersza z bufora tymczasowwego do nowego miejsca
      memcpy(data + offset_new, tmpBuffer, lenghtToCopy);
    }
}
```



(a) Orygnalny obrazek

(b) Po wywołaniu funkcji

Rysunek 1: Efekt działania funkcji `rotate()`

3.2 Negatyw obrazu

Negatyw wykonano w dwóch wersjach. Jedna z użyciem rozszerzenia MMX oraz druga z wykorzystaniem rejestrów ogólnego przeznaczenia.

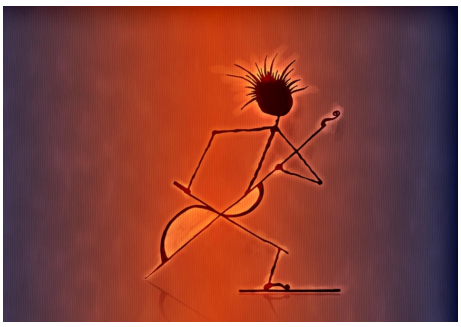
Wersja MMX

```
movl 8(%ebp), %ebx # adres gdzie zaczyna sie ciag pikseli pliku bmp
movl $ZERO, %esi
movl 12(%ebp), %edi # długość pliku

#odłożenie na stos ciagu 64 jedynek, które posłużą do kolejnych operacji
pushl $-1
pushl $-1

PETLA:
#pobranie 64 bitów zawierającego piksele do mm0
movq (%ebx), %mm0

#pobranie ciągu jedynek ze stosu do mm1
movq (%esp), %mm1

#odjecie ciągu jedynek od wartości kanałów pikseli
psubq %mm0, %mm1

#nadpisanie informacji o wartościach kanałów
movq %mm1, (%ebx)

#zwiększnie licznika
addl $8, %esi

#następne 8 bajtów tablicy bajtów
addl $8, %ebx

#czy koniec danych
cmpl %edi, %esi
jb PETLA
```

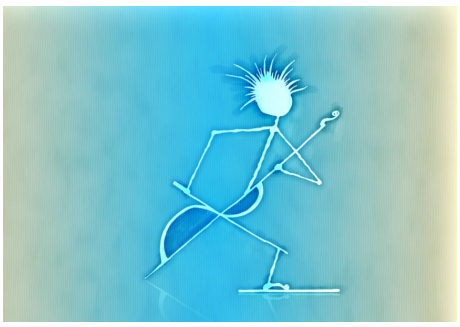
Wersja standardowa

Wersja standardowa funkcji `negateImage` działa w analogiczny sposób. Jedynymi zmianami są optymalizacje takie jak nieprzesyłanie ciagu jedynek przez stos, tylko bezpośrednie wpisywanie potrzebnej wartości do rejestru. Ze względu na mniejszy rozmiar rejestrów ogólnego przeznaczenia w architekturze 32 bitowej, pętla wykonuje 2 krotnie więcej iteracji, a pobierane ciągi bajtów są 32 bitowe.

```
PETLA2:
movl (%ebx), %eax # pobranie 32 bitów z ciągu bajtów z pikselami do %eax
movl $-1, %edx #same jedynki
subl %eax, %edx #odjęcie od samych jedynek wartości kanałów w %eax
movl %edx, (%ebx) # zastąpienie danych
addl $4, %esi #zwiększenie licznika
addl $4, %ebx #następne 32 bity danych
cmpl %edi, %esi #czy koniec danych
jb PETLA2
```



(a) Orygnalny obrazek



(b) Po wywołaniu funkcji

Rysunek 2: Efekt działania funkcji `negateImage()` lub `negateImageMMX()`

3.3 Uruchomienie programu

Programy były kompilowane i uruchamiane przy użyciu poniższych instrukcji

```
gcc image.c image.s -o test -m32
./test
```

3.4 Test wydajności

Do testów został użyty program `Gprof`. Obie funkcje negacji obrazu zostały wykonane po 10 000 razy. Przy użyciu tego samego pliku `bmp` o wadze 21mb.

```
void testMMX(char *data, int fileLenght){
     int i;
     for(i = 0; i < 10000; i++){
        negateImageMMX(data, fileLenght);
    }
}
```

```
void testWithoutMMX(char *data, int fileLenght){
     int i;
    for(i = 0; i < 10000; i++){
        negateImage(data, fileLenght);
    }
}
```

Program uruchomiono z dodatkowymi flagami `-Wall` oraz `-pg`. Służącymi do generowania dodatkowych informacji do pliku `gmon.out`.

```
gcc -Wall -pg image.c image.s -o test -m32
```

Następnie przy pomocy narzędzia `gprof` wyekstraktowano z danych binarnych znajdujacych się w wygenerowanym pliku `gmon.out` dane do analizy do pliku tekstowego.

```
gprof test gmon.out > analiza.txt
```

Czas wykonania funkcji bez użycia MMX wyniósł 249,97 sekund, natomiast czas przy użyciu MMX wyniósł 167,97 sekund. Z analizy wynika, że funkcja wykorzystująca jednostę wektorową jest szybsza. Prawdopodobnie wynika to z faktu iż funkcja, która wykorzystywała jednostkę wektorową, wykonała 2 krotnie mniej iteracji w pętli.

4 Podsumowanie i wnioski

Zadanie wymagało pełnej wiedzy o tym jak działa jednostka wektorowa, oraz w jaki sposób wygląda format `bmp`, gdyż trudnością okazało się debugowanie programu i monitorowanie wartości jakie aktualnie znajdują się w rejestrach. Jednostka `MMX` jest to szczególne przydatne narzędzie w przypadku przetwarzania obrazów lub dźwięku, ale też wszędzie tam gdzie przetwarzane są duże ilości danych. Należy jednak pamiętać, że rejestry jednostki wektorowej MMX korzystają z tych samych rejestrów co koprocesor x87, dlatego też po końcu wykonywania operacji przy użyciu MMX należy wywołać instrukcję `emms`, która wyczyści dane w rejestrach MMX/FPU.

Literatura

[1] https://en.wikipedia.org/wiki/BMP_file_format, informacje na temat formatu bmp.

[2] http://zak.ict.pwr.wroc.pl/materials/architektura/laboratorium/Dokumentacja/Intel%20Penium%20IV/IA-32%20Intel%20Architecture%20Software%20Developers%20Manual%20vol.%201%20-%20Basic%20Architecture.pdf, informacje na temat jednostki wektorowej.

[3] https://hakin9.org, czasopismo hakin9, Nr 3/2008, informacje o formacie bmp.

[4] http://www.thegeekstuff.com/2012/08/gprof-tutorial/ informacje o liczeniu czasu przy użyciu gprof.