ARKO – laboratorium

Materiały pomocnicze do zajęć dla grupy z wtorku, 10.15..12.00, semestr letni 2015

Grzegorz Mazur, IIPW, p. 310

konsultacje: wtorek 12.15, ew. środa 12.15

Lista grup (z numerami zadań „dużych” projektów)

1	7	Bogucki	Michał Jan	258919	B2I2	Wt_NP_10
2	14	Czerwiński	Bartłomiej	262732	B2I2	Wt_NP_10
3	19	Filipiuk	Mateusz Aleksander	253084	B2I2	Wt_NP_10
4	28	Jeleński	Borys	265749	B2I2	Wt_NP_10
5	33	Krakowski	Tomasz	272039	B2I2	Wt_NP_10
6	37	Kuczyński	Hubert	265754	B2I2	Wt_NP_10
7	38	Lenard	Maciej	265755	B2I2	Wt_NP_10
8	66	Rewerska	Patrycja	238405	Z2SID	Wt_NP_10
9	81	Tomaszewski	Mateusz Maciej	246456	U1SID	Wt_NP_10
10	92	Zabłocki	Aleksander Andrzej	241774	Z2SID	Wt_NP_10
11	20	Giemza	Michał	259908	C2I1	Wt_P_10
12	26	Janeczko	Marcin Hubert	265748	B2I2	Wt_P_10
13	46	Majewski	Rafał Bogdan	265758	B2I2	Wt_P_10
14	47	Malanowska	Agnieszka Halina	265759	B2I2	Wt_P_10
15	49	Mazurek	Jakub Mateusz	265761	B2I2	Wt_P_10
16	54	Musialik	Daniel Robert	265762	B2I2	Wt_P_10
17	58	Niedźwiedź	Łukasz	265825	B2I2	Wt_P_10
18	62	Piękos	Piotr	265767	B2I2	Wt_P_10
19	63	Postępski	Jakub	257947	B2I2	Wt_P_10
20	72	Skupińska	Anna	265771	B2I2	Wt_P_10
21	76	Szumski	Jakub	265776	B2I2	Wt_P_10
22	84	Tym	Aleksander Dariusz	265782	B2I2	Wt_P_10
23	86	Wasak	Jan Paweł	265783		Wt_P_10
24	90	Wojtkowski	Mariusz Krzysztof	265785	B2I2	Wt_P_10

Uwagi ogólne

Argumenty programów nie mogą być zapisane na sztywno w kodzie; w programach hybrydowych x86 powinny one być pobierane z linii polecenia, a w programach asemblerowych MIPS – z linii polecenia lub interakcyjnie z konsoli.

Pliki asemblerowe powinny być sformatowane zgodnie z typową konwencją: nazwy i etykiety od pierwszej kolumny tekstu, dyrektywy i instrukcje po tabulacji, argumenty po następnej tabulacji, oddzielane przecinkiem i spacją.

Należy unikać oczywistych nieoptymalności, w tym zwłaszcza sekwencji skoków – skok bezwarunkowy bezpośrednio po warunkowym omijającym go.

Stałe powinny być zapisane w postaci zgodnej z intuicją programisty – asembler akceptuje zapisy stałych zgodnych ze składnią języka C, w tym szesnastkowych i znakowych. Do reprezentacji znaków widocznych nie należy w programie używać odczytanych z tabeli dziesiętnych wartości kodów ASCII.

W kodzie asemblerowym preferowane jest użycie wskaźników zamiast indeksowania. O ile jest to możliwe (gwarantowany min. jedna iteracja), pętle należy zapisywać w postaci ze skokiem warunkowym zamykającym pętlę, umieszczonym na jej końcu.

Ćwiczenie 1 – Prosty projekt programu asemblerowego w środowisku symulatora SPIM/MARS

Ćwiczenie polega na napisaniu i uruchomieniu programu realizującego zadaną funkcjonalność – przetwarzanie łańcuchów tekstowych. Program musi być uruchomiony w laboratorium w czasie 2h. Ocena maksymalna – 2p.

Ćwiczenie 2 – Projekt programu asemblerowego w środowisku symulatora SPIM/MARS

Projekt wykonywany w laboratorium lub w domu, oddawany w czasie sesji laboratoryjnych.
Napisać program w języku asemblera MIPS działający w symulatorze MARS lub podobnym, zgodnym ze SPIM. Wykonanie programu nie może być uzależnione od wartości niezainicjowanych rejestrów; program musi działać prawidłowo po restarcie, bez ponownego ładowania.
Należy unikać sekwencji skoków, zwłaszcza skoków bezwarunkowych następujących po skokach warunkowych.
Programy przetwarzające pliki tekstowe powinny zawierać funkcje getc i putc zapewniające buforowanie dostępów do plików – dostępy do plików powinny być realizowane poprzez bufory o pojemności np. 512 B.
Programy z arytmetyką zmiennopozycyjną korzystają z funkcji systemowych wprowadzania i wyświetlania liczb zmiennopozycyjnych, wykonując obliczenia na jednostce stałopozycyjnej (ze sprawdzeniem wyniku na jednostce zmiennopozycyjnej).
Programy wyświetlające obrazy graficzne korzystają z funkcji terminala graficznego dostępnych w nowych wersjach symulatora Mars.
Przy przetwarzaniu plików .BMP należy zwrócić uwagę na brak wyrównania naturalnego pól nagłówka – mogą być tu pożyteczne instrukcje z grupy unaligned load.
Ocena maksymalna – 10 punktów.
Przyporządkowanie projektów zgodnie z pozycjami na załączonej liście studentów.

1. Program sumujący wprowadzane z konsoli dziesiętne liczby stałopozycyjne zawierające do 200 cyfr.

2. Wyświetlanie obrazu z pliku .BMP w formacie 24 bpp z odbiciem lustrzanym względem pionowej osi symetrii.

3. Rotacja obrazu zapisanego w pliku .BMP w formacie 1 bpp o 90 stopni.

4. Kalkulator działający w notacji polskiej odwrotnej operujący na liczbach całkowitych zapisanych w systemie siódemkowym.

5. Obliczanie pierwiastka całkowitego liczby całkowitej algorytmem nierestytucyjnym.

6. Program czytający program asemblerowy MIPS i generujący statystykę liczb wystąpień poszczególnych instrukcji i dyrektyw asemblera.

7. Program przetwarzający tekst poprawnego programu w języku C poprzez zastępowanie w stałych znakowych (pojedynczych znakach i łańcuchach) znaków spoza zakresu ASCII odpowiednimi sekwencjami \xnn.

8. Wejściowy plik tekstowy zawiera tekst naturalny z datami zapisanymi w dowolnym typowym formacie numerycznym (dd.mm.yy, mm/dd/yy, yyyy-mm-dd, z 2- lub 4-cyfrową reprezentacją roku). Program przetwarza tekst poprzez zmianę postaci wszystkich dat na jednolitą – wybraną przez użytkownika.

9. Program dopisujący na końcu pliku tekstowego wiersz tekstu zawierający kolejną liczbę dziesiętną bez znaku po ostatniej znalezionej w pliku. Jeśli plik nie istnieje lub nie zawiera liczb – w ostatnim wierszu powinna pojawić się liczba 1.

10. Program czyta plik .BMP zawierający obraz zapisany w formacie 1, 2 lub 4 bpp (jeden format do wyboru) i wyświetla górny lewy róg obrazu (max. 64×24 piksele) na konsoli używając do reprezentacji poszczególnych pikseli pojedynczych znaków ASCII. Program musi poprawnej obsługiwać pliki zawierające obrazy o dowolnej rozdzielczości, również mniejszej od 64×24.

11. Dzielenie liczb zmiennpozycyjnych IEEE binary64 przez IEEE binary32 bez użycia jednostki zmiennopozycyjnej.

12. Generowanie obrazu o zadanej wysokości/szerokości z umieszczonym centralnie kołem o zadanej średnicy. Użyć algorytmu Bresenhama do równoczesnego kreślenia 8 punktów okręgu oraz dowolnego prostego algorytmu wypełniania koła.

13. Mnożenie liczby zmiennopozycyjnej IEEE binary64 przez 32-bitową liczbę całkowitą ze znakiem, z wynikiem binary64.

14. Wyświetlanie obrazu z pliku .BMP 24 bpp z proporcjonalnym skalowaniem w dół w pionie i w poziomie tak, aby wysokość wyświetlanego obraz nie przekraczała np. 600 pikseli.

15. Program zamieniający stałe binarne w programie źródłowym w języku C na zgodne ze standardem ANSI stałe szesnastkowe o możliwie najkrótszej reprezentacji (0b100001 → 0x21).

16. Wyświetlanie obrazu gładko cieniowanego trójkąta o zadanych współrzędnych (x, y) i kolorach wierzchołków (rgb).

17. Wyświetlanie obrazu z pliku .BMP (do wyboru – odcienie szarości 8 BPP lub kolorowy 24 BPP z zamianą na skalę szarości) w postaci czarno-białej (dwa kolory pikseli) przy użyciu techniki drżenia (dithering). Błąd odwzorowania powinien propagować do kolejnego piksela w wierszu. Błąd odwzorowania koloru ostatniego piksela w wierszu propaguje do piksela położonego pod lub nad nim w kolejnym wierszu – w kolejnych wierszach następuje zmiana kierunku propagacji błędu i kolejności określania kolorów pikseli.

18. Mnożenie dziesiętnych liczb stałopozycyjnych bez znaku zawierających do 50 cyfr znaczących.

19. Dzielenie liczb całkowitych ze znakiem z wynikiem zmiennopozycyjnym IEEE binary64, bez użycia jednostki zmiennopozycyjnej ani dzielenia stałopozycyjnego.

20. Program przetwarzający plik źródłowy w języka asemblerowym, zastępujący stałe szesnastkowe w tradycyjnej notacji asemblerowej (np. 0abch, 123h) stałymi w notacji zgodnej ze składnią C (0xabc, 0x123). Program nie powinien zmieniać postaci stałych innych, niż szesnastkowe.

21. Wyświetlanie obrazu z pliku .BMP 24 bpp z rotacją o 90 stopni w prawo.

22. Wyświetlanie obrazu z pliku .BMP 24 bpp z redukcją nasycenia barw w stosunku x/256, x zadawane jako parametr.

23. Wyświetlanie symulowanego obrazu poziomego widma barw widzialnych (od czerwieni do fioletu) o zadanej przez użytkownika wysokości (>= 1) i szerokości (>= 3).

24. Skanowanie z tekstu źródłowego liczb całkowitych bez znaku w dowolnym zapisie zgodnym ze składnią języka C (ósemkowych, dziesiętnych, szesnastkowych) i wyświetlanie każdej z nich w trzech postaciach: ósemkowej, dziesiętnej i szesnastkowej. Wymagana poprawna obsługa zakresu $0..2^{32}-1$.

Ćwiczenie 3 – Projekt prostego programu hybrydowego – C + assembler x86

Ćwiczenie 3 polega na napisaniu i uruchomieniu programu hybrydowego, z modułem głównym w języku C i wywoływaną z niego funkcją napisaną w asemblerze x86 w składni NASM. Program musi zostać napisany i uruchomiony w czasie zajęć laboratoryjnych (2h). Ocena maksymalna – 2p. Zadania mają stopień złożoności zbliżony do przykładów z poniższej listy.

`char *removerng(char *s, char a, char b);`
usuwanie z łańcucha znaków o kodach z zakresu od a do b, a < b

`char *remnth(char *s, int n);`
usuwanie co n-tego znaku z łańcucha

`char *leavelastndig(char *s, int n);`
Usuwanie z łańcucha znaków poza ostatnimi n cyframi

`char *remrep(char *s);`
usuwanie powtórzeń znaków

`char *leavelongestnum(char *s, int n);`
Usuwanie z łańcucha znaków poza najdłuższą liczbą dziesiętną

`char *leaverng(char *s, char a, char b);`
pozostawienie w łańcuch znaków o kodach z zakresu od a do b

`char *remlastnum(char *s);`
usuwanie ostatniego ciągu cyfr dziesiętnych z łańcucha

`unsigned int getdec(char *s);`
wczytanie pierwszej liczby dziesiętnej z łańcucha

`unsigned int gethex(char *s);`
wczytanie pierwszej liczby szesnastkowej z łańcucha

`char *reversedig(char *s);`
odwracanie kolejności cyfr w łańcuchu z zachowaniem pozycji pozostałych znaków

`char *reverselet(char *s);`
odwracanie kolejności liter w łańcuchu z zachowaniem pozycji pozostałych znaków

`char *reversepairs(char *s);`
odwracanie kolejności znaków w parach; poprawna obsługa łańcuchów o nieparzystej długości

`char *replnum(char *s, char a);`
zastępowanie ciągów cyfr dziesiętnych pojedyczym znakiem

`char *capwords(char *s);`
Zmiana pierwszej litery każdego słowa w łańcuchu na wielką.

Ćwiczenie 4 – Projekt programu hybrydowego – C + assembler x86

Projekt wykonywany w domu, oddawany w czasie sesji laboratoryjnych.
Napisać program działający pod kontrolą systemu Linux, składający się z głównego modułu w języku ANSI C, zapewniającego wejście i wyjście oraz modułu asemblerowego realizującego przetwarzanie danych. Treść zadania zawiera deklarację funkcji asemblerowej widzianą na poziomie języka C. Do asemblacji używamy asemblera NASM (nasm.sf.net). Kompilacja i konsolidacja następuje przy użyciu drivera kompilatora CC. Argumenty dla programu powinny być zadawane w linii polecenia, w celu ułatwienia testowania. Moduł asemblerowy nie powinien wywoływać funkcji z biblioteki języka C. Argumenty dla procedur operujących na bitach powinny być zadawane przez użytkownika w postaci szesnastkowej. Procedury przetwarzające obrazy w formacie .BMP mogą otrzymywać jako argument wskaźnik na bufor zawierający cały plik albo wskaźnik na samą strukturę obrazu i informacje o jego rozmiarach. Jeśli nie zaznaczono inaczej, procedura powinna poprawnie przetwarzać obraz o dowolnych rozmiarach.
Należy unikać sekwencji skoków, zwłaszcza skoków bezwarunkowych następujących bezpośrednio po skokach warunkowych. Próba oddania projektu jawnie niezgodnego konwencję wołania powoduje stratę jednego punktu.
Przyporządkowanie projektów zgodnie z pozycjami na załączonej wyżej liście studentów.

Zadania:

1. `void flipdiag bmp24(void *img, int width);`
   Odbicie lustrzane kwadratowego obrazu .BMP w formacie 24bpp względem przekątnej.

2. `void rotbmp24(void *img, int width);`
   Rotacja kwadratowego obrazu .BMP w formacie 24bpp o 90 stopni w prawo.

3. `void *checkerboard24(void *img, unsigned int sqsize, unsigned int color1, unsigned int color2);`
   Generowanie pliku .BMP 24 bpp zawierającego obraz dwukolorowej szachownicy o zadanym boku kwadratu, wysokości I szerokości calego obrazu. Wysokość i szerokość obrazu nie muszą być wielokrotnościami boku kwadratu. Kolory zadane w postaci dwóch liczb (efektywnie 24-bitowych), czytanych przez program główny z linii polecenia w postaci szesnastkowej.

4. `void flipdiag bmp1(void *img, int width);`
   Odbicie lustrzane kwadratowego obrazu .BMP w formacie 1bpp względem przekątnej.

5. `void bwdither(void *img, int width, int height);`
   Kwantyzacja obrazu .BMP w odcieniach szarości do dwóch kolorów – czarnego i białego, z użyciem drżenia (dithering). Błąd kwantyzacji powinien propagować wzdłuż każdego wiersza, wpływając na kolory kolejnych pikseli; kierunek propagacji błęDu zmienia się na końcu każdego wiersza – błąd z ostatniego piksela pierwszego wiersza propaguje do ostatniego piksela drugiego wiersza.

6. `void shrinkbmp24(void *img, unsigned int scale_num, unsigned int scale_den);`
   Pomniejszenie obrazu .BMP 24 bpp w stosunku podanym jako ułamek scale_num/scale_den (< 1). Można zmodyfikować listę argumentów stosownie do wymagań implementacji. Rozważyć możliwość użycia jednostki wektorowej.

7. `void xpandbmp24(void *img, unsigned int scale_num, unsigned int scale_den, ...);`
   Powiększenie obrazu .BMP 24 bpp w stosunku podanym jako ułamek scale_num/scale_den (> 1). Należy uzupełnić lub zmodyfikować listę argumentów stosownie do wymagań implementacji. Rozważyć możliwość użycia jednostki wektorowej.

8. `void mirrorbmp1 (void *img, int width, int height, ... );`
   Odbicie lustrzane obrazu .BMP w formacie 1 bpp względem osi pionowej.

9. `char * smul(char *d, char *s1, char *s2);`
   Mnożenie długich całkowitych liczb dziesiętnych bez znaku, zapisanych jako łańcuchy cyfr, z wynikiem w takiej samej postaci- Program główny powinien zaalokować bufor na łańcuch wynikowy. Wskazane zastosowanie instrukcji AAM.

10. `void fadetop(void *img, int width, int height, int dist);`
    Rozjaśnienie obrazu od górnej krawędzi. Kolor każdego piksel powinien zostać zinterpolowany liniowo pomiędzy kolorem oryginalnym i bielą proporcjonalnie do odległości piksela od górnej krawędzi obrazu. Piksele położone dalej od krawędzi niż wartość dist nie podlegają modyfikacji.

11. `void *spectrum(void *img, int w, int h);`
    Wygenerować obraz .BMP 24bpp zawierający przybliżony widok poziomego widma światła białego – od czerwonego po lewej stronie do niebieskiego po prawej, o zadanej szerokości i wysokości. Zastosować prostą interpolację liniową z uwzględnieniem skalowania dla dowolnej szerokości >= 3 pikseli.

12. `void desat(void *img, int level);`
    Redukcja nasycenia barw obrazu . BMP 24bpp poprzez interpolację koloru piksel pomiędzy kolorem oryginału i kolorem szarym w stosunku level/64 (0 – kolor oryginalny, 64 – cały obraz jednolicie szary).

13. `char * sdiv(unsigned int base, char *d, char *s1, char *s2);`
    Dzielenie dwóch liczb całkowitych bez znaku przekazanych w postaci łańcuchów cyfr w systemie liczbowym o zadanej bazie, z wynikiem w postaci dwóch łańcuchów znaków. Reprezentujących iloraz i resztę. Program główny powinien alokować bufor na wynik d o właściwym rozmiarze. Reszta z dzielenia jest zwracana w buforze s1.

14. `void sunfade(void *img, int width, int height, int dist, int x, int y);`
    Rozjaśnienie obrazu od zadanego punktu. Kolor każdego piksel powinien zostać zinterpolowany liniowo pomiędzy kolorem oryginalnym i bielą proporcjonalnie do kwadratu odległości piksela od piksela o współrzędnych x,y. Piksele położone dalej od krawędzi niż wartość dist nie podlegają modyfikacji.

15. `float myfsqrt(float f);`
    Obliczenie pierwiastka kwadratowego liczby zmiennopozycyjnej bez użycia jednostki zmiennopozycyjnej ani wektorowej. Można użyć np. szeregu Newtona-Raphsona.

16. `void rotbmp1(void *img, int width, int height, ...);`
    Rotacja kwadratowego obrazu BMP w formacie 1 bpp o 90 stopni w prawo.

17. `int mandel(int re, int im);`
    Generowanie obrazu zbioru Mandelbrota o zadanej szerokości (w pikselach) I dobranej do niej wysokości. Obliczenia należy prowadzić w reprezentacji stałopozycyjnej (16.16 lub 8.24). Funkcja asemblerowa sprawdza, czy punkt o zadanych współrzędnych (stałopozycyjnych) należy do zbioru Mandelbrota – funkcja zwraa liczbę iteracji niezbędną do stwierdzenia rozbieżności przekształcenia, ograniczoną do 255. Program w języku C generuje obraz, zamieniając wartość funkcji np. na odcień szarości lub kolor piksela.

18. `void circle(void *image, int width, int height, int xc, int yc, int radius, unsigned int color)`
    Rysowanie okręgu o zadanych parametrach. Program w C zapisuje plik .BMP, procedura asemblerowa rysuje okrąg .Okrąg powinien być kreślony przy użyciu algorytmu minimalizującego złożoność obliczeniową zadania, np. Bresenhama.

19. `void shaderect(int height, int width, unsigned int color[4]);`
    Generowanie obrazu .BMP zawierającego gładko cieniowany prostokąt o zadanych kolorach czterech wierzchołków. Obliczenia kolorów pikseli powinny być prowadzone na liczbach stałopozycyjnych w formacie 16.16.

20. `void enhance_contrast(void *img);`
    Wzmocnienie kontrastu obrazu .BMP 24 bpp poprzez przeskalowanie zakresu składowych przez jeden wspólny współczynnik, w taki sposób, by najmniejsza wartość składowej wynosiła 0,a największa – 255. Obliczenia stałopozycyjne, wskazane użycie jednostki wektorowej.

21. `void sepia(void *img, int width, int height);`
    Zamiana obrazu kolorowego .BMP image na skalę sepii.

22. `void uquantize(void *img, int levels);`
    Równomierna kwantyzacja składowych obrazu .BMP 24 bpp .BMP do zadanej liczby poziomów. Np. przy levels = 4, zakresy 0..63, 64..127, 128..191 i 192..255 powinny być odpowiednio reprezentowane przez wartości 32, 96, 160 i 224.

23. `void sudoku(char mtx[9][9]);`
    Rozwiązywanie sudoku. Program w C powinien czytać znane cyfry ze strumienia wejściowego, zawierającego 9 wierszy po 9 znaków (cyfry i spacje lub inne symbole) i wypisywać rozwiązanie do strumienia wyjściowego.

24. `void reduce_contrast(void *img, unsigned int rfactor);`
    Osłabienie kontrastu obrazu .BMP 24 bpp w stosunku rfactor/128 poprzez przeskalowanie składowych w kierunku środka zakresu (128). Obliczenia stałopozycyjne, wskazane użycie jednostki wektorowej.