ARKO – laboratorium
programowanie asemblerowe i hybrydowe

materiały pomocnicze do zajęć

aktualizacja 20.03.2023

Grzegorz Mazur
Instytut Informatyki

Wymagania 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 RISC-V – 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 (gwarantowana min. jedna iteracja), pętle należy zapisywać w postaci ze skokiem warunkowym zamykającym pętlę, umieszczonym na jej końcu.

Należy unikać sekwencji instrukcji złożonych ze skoku warunkowego i następującego bezpośrednio o nim skoku bezwarunkowego – w większości przypadków świadczy to o odwróceniu warunku skoku lub niewłaściwej kolejności fragmentów kodu połączonych skokami.

Ćwiczenie 1 – Wprowadzenie do programowania asemblerowego w środowisku symulatora RARS

Ćwiczenie polega na napisaniu i uruchomieniu programu realizującego zadaną funkcjonalność – przetwarzanie łańcuchów tekstowych wprowadzanych z konsoli z wyświetlaniem wyniku przetwarzania na konsoli. Program musi być uruchomiony w laboratorium w czasie 2h. Ocena maksymalna – 3p. Przykładowe zadania:

1. Zastępowanie cyfr ich dopełnieniem do 9 ($0 \rightarrow 9$, $1 \rightarrow 8$, $2 \rightarrow 7$ itd.).
2. Wypisywanie najdłuższego ciągu cyfr znalezionego w łańcuchu wejściowym.
3. Odwrócenie kolejności cyfr w łańcuchu tekstowym zawierającym również inne znaki.
4. Skanowanie (konwersja) i wyświetlenie największej liczby dziesiętnej z łańcucha.
5. Zliczanie liczb dziesiętnych w łańcuchu.
6. Usunięcie z łańcucha znaków z określonego podzbioru (cyfr, małych liter itp.)
7. Usunięcie z łańcucha sekwencji znaków o pozycji i długości zadanej w postaci liczb, z właściwą reakcją na nieprawidłowe wartości argumentów.
8. Zastępowanie co trzeciej małej litery przez odpowiadającą jej wielką.
9. Usuwanie z każdej ciągłej sekwencji wielkich liter wszystkich oprócz pierwszej litery.

Ćwiczenie 2 – Projekt programu asemblerowego w środowisku symulatora RARS

Projekt wykonywany w laboratorium lub w domu, oddawany w czasie sesji laboratoryjnych.

Napisać program w języku asemblera RISC-V działający w symulatorze RARS. 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 RARS.
Przy przetwarzaniu plików .BMP należy zwrócić uwagę na brak wyrównania naturalnego pól nagłówka – mogą być tu pożyteczne instrukcje z grupy unaligned load.
Ocena maksymalna – 6 punktów.

Przykładowe zadania:

1. Program wyświetlający zbiór Mandelbrota w oknie graficznym symulatora RARS. Obliczenia należy wykonać na liczbach stałopozycyjnych o odpowiednio dobranym formacie. Kolor lub jasność piksela zależy od liczby iteracji potrzebnych do stwierdzenia rozbieżności przekształcenia. Obliczenia dla każdego piksela wykonuje się do stwierdzenia rozbieżności lub osiągnięcia maksymalnej założonej liczby iteracji (np. 200).

2. Program sumujący wprowadzane z konsoli dziesiętne liczby stałopozycyjne zawierające do 200 cyfr. Liczby mogą zawierać tylko część całkowitą, tylko część ułamkową lub obie części.

3. Kalkulator działający w notacji polskiej odwrotnej operujący na liczbach całkowitych zapisanych w systemie ósemkowym.

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

5. Program zamieniający stałe binarne w programie źródłowym w języku C na zgodne ze standardem ANSI stałe szesnastkowe o możliwie najkrótszej reprezentacji (0b100001 → 0x21). Program nie może modyfikować łańcuchów tekstowych ani komentarzy.

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

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 – 3p. 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/x86-64

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.us). Kompilacja i konsolidacja następuje przy użyciu drivera kompilatora CC.
Projekt powinien być przygotowany w dwóch wersjach – 32- i 64-bitowej. Z uwagi na dostępność większej liczby rejestrów wersja 64-bitowa powinna korzystać z mniejszej liczby zmiennych na stosie w pamięci. Ocena za jedną (dowolną) wersję – 6 punktów, za drugą – 2 punkty. Dane 32-bitowe powinny być w wersji 64-bitowej traktowane jako 32-bitowe – jedynie operacje na adresach są 64 bitowe.
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 z konwencją wołania (niestandardowa kolejność akcji w prologu, niezachowanie rejestrów grupy saved itp.) powoduje stratę jednego punktu.

Przykładowe zadania:

1. `void flipdiagbmp24(void *img, int width);`
   Odbicie lustrzane kwadratowego obrazu .BMP w formacie 24bpp względem przekątnej.

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

3. `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.

4. `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.

5. `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.

6. `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.