Architektury systemów komputerowych

Lista zadań nr 8

Na zajęcia 23 i 24 kwietnia 2018

Wymogi formalne

Zadania należy napisać w asemblerze x86-64 ze składnią AT&T – składnia Intel jest niedozwolona. Działanie procedur należy sprawdzić wołając je z programu napisanego w języku C. Testowanie polega na uruchamianiu programu z przykładowymi wartościami wczytywanymi z linii poleceń. Sekcja .symtab plików relokowalnych musi być prawidłowo wypełniona, tj. każdemu zdefiniowanemu symbolowi należy przypisać rozmiar i typ.

Podglądanie kodu generowanego przez kompilator celem implementacji rozwiązań jest niedozwolone!

Rozwiązania zadań wysyła się z użyciem systemu SKOS. Należy dostarczyć plik w formacie «tar.gz» o nazwie «indeks_imie_nazwisko.tar.gz». Po rozpakowaniu archiwum struktura katalogów ma prezentować się następująco:

```
999999_jan_nowak/
  zad1/
    clz.s
    zad1.c
    Makefile
  zad2/
    lcm_gcd.s
    zad2.c
    Makefile
  ...
```

Oceniający zadania używa komputera z zainstalowanym systemem Debian GNU/Linux 9 dla architektury x86-64. Ściąga z systemu SKOS archiwum dostarczone przez studenta, po czym sprawdza poprawność struktury katalogów. Dla każdego sprawdzanego zadania powtarza poniższą procedurę:

1. wchodzi do jednego z podkatalogów, np.: zad5,
2. wykonuje polecenie «make» celem zbudowania pliku wykonywalnego o nazwie zad5,
3. testuje program wywołując go z linii poleceń z zestawem wybranych argumentów, np.: «./zad5 10.7542 0.031415»,
4. czyta treść rozwiązania celem znalezienia usterek i plagiatów,
5. wywołuje polecenie «make clean», aby usunąć z katalogu wszystkie pliki poza źródłowymi.

Jeśli którykolwiek z powyższych kroków zakończy się niepowodzeniem, sprawdzający ma prawo przerwać ocenianie i nie zaliczyć rozwiązania.

Termin oddawania zadań z użyciem systemu SKOS wyznaczono na koniec dnia 4 maja 2018.

Prezentacja rozwiązań

Przed zajęciami student składa deklarację (kuponik) wyrażającą chęć prezentacji kodu źródłowego swoich rozwiązań i umiejętności posługiwania się narzędziami `objdump`, `readelf` i `gdb`. Punkty z kuponików liczą się osobno w stosunku do zdobytych za rozwiązania zamieszczone w systemie SKOS.

W trakcie zajęć dostępny będzie rzutnik lub telewizor z wejściem HDMI. Student wezwany do prezentacji rozwiązania podłącza laptop z uruchomionym systemem `Linux`. Prezentacja ma odbywać się w konsoli z ustawioną czytelną czcionką i paletą kolorów o dobrym kontraście – w trybie pełnoekranowym terminal ma mieścić około 30 linii, tło powinno być jasne, a czcionka rysowana ciemnym kolorem. W przypadku zbędnej zwłoki wynikającej z problemów technicznych student może nie otrzymać punków za zadanie.

Należy uruchomić swój program pod kontrolą debuggera `gdb`. Będziecie proszeni o wyświetlenie zawartości rejestrów, zmiennych, komórek pamięci i ramek stosu; ustawienie punktów wstrzymań i obserwacji. Trzeba się do tego odpowiednio przygotować czytając RMS's gdb Debugger Tutorial[1] i GDB Quick Reference[2].

Prowadzący zajęcia będzie zadawał pytania odnośnie używanych instrukcji procesora, dyrektyw asemblera, konwencji wołania procedur, organizacji stosu oraz struktury plików relokowalnych i wykonywalnych `ELF`.

Studentowi może zostać przydzielony punkt uznaniowy, jeśli ten przygotował starannie swoje wystąpienie i wykazał się bardzo dobrą znajomością wyżej wymienionych narzędzi.

Porady i wskazówki

Zapoznaj się z przykładami ze strony GNU Assembler Examples[3]. Należy wykorzystywać dyrektywy opisane w GNU as: Assembler Directives[4]. Zadbaj o czytelność kodu! Jeśli używasz wielu rejestrów do przechowywania zmiennych lokalnych to przypisz im nazwy, np.: `epsilon = %xmm6`, `arg0 = %rdi`. Magiczne stałe należy sensownie nazwać, np.: `MIN_INT = 0x7fffffff`, `exit = 60`.

Zadanie 1. Napisz procedurę, która dla danego słowa maszynowego wyznacza długość prefiksu składającego się z samych zer. Rozwiązanie ma działać w $O(\log n)$, gdzie n jest długością słowa.

```
int clz(long); /* count leading zeros */
```

Dla każdego modułu translacji wyświetl poleceniem `nm` tablicę symboli, a `objdump` relokacje w sekcji `.text`.

Zadanie 2. Zaimplementuj procedurę wyznaczającą największy wspólny dzielnik oraz najmniejszą wspólną wielokrotność dwóch liczb naturalnych metodą iteracyjną. Wynik mieści się w rejestrach `%rdx` i `%rax`.

```
typedef struct {
  unsigned long lcm, gcd;
} result_t;

result_t lcm_gcd(unsigned long, unsigned long);
```

W trakcie prezentacji zadania użyj `gdb`, aby zatrzymać się na początku każdej iteracji i wyświetlić wartość zmiennych lokalnych przechowywanych w rejestrach.

Zadanie 3. Napisz procedurę, która sortuje tablicę liczb całkowitych metodą *insertion sort*. Tablica jest zadana przez dwa wskaźniki – na element początkowy `first` i końcowy `last`.

```
void insert_sort(long *first, long *last);
```

W trakcie prezentacji zadania użyj `gdb`, aby zatrzymać się na początku każdej iteracji i wyświetlić zawartość sortowanej tablicy na podstawie zawartości rejestrów `%rdi` i `%rsi`.

[1] `http://www.unknownroad.com/rtfm/gdbtut/`
[2] `http://www.cs.berkeley.edu/~mavam/teaching/cs161-sp11/gdb-refcard.pdf`
[3] `http://cs.lmu.edu/~ray/notes/gasexamples/`
[4] `https://sourceware.org/binutils/docs-2.26/as/Pseudo-Ops.html`

Zadanie 4. Napisz procedurę, wyznaczającą metodą rekurencyjną n-ty wyrazu ciągu Fibonacciego.

```
unsigned long fibonacci(unsigned long n);
```

W trakcie prezentacji zadania użyj gdb, aby zatrzymać się w momencie osiągnięcia przez zmienną `n` wartości podanej przez osobę prowadzącą zajęcia. W tym celu użyj punktu obserwacyjnego z wyrażeniem warunkowym. Następnie pokaż wszystkie ramki stosu. Pamiętaj o odpowiednim ustawieniu rejestru `%rbp` przechowującego wskaźnik na bieżącą ramkę stosu.

Zadanie 5. Zaimplementuj procedurę, która przyjmuje binarną reprezentację wartości zmiennopozycyjnych zapisanych w standardzie IEEE-754 pojedynczej precyzji i zwraca ich iloczyn. Obliczenia należy przeprowadzić bez używania instrukcji jednostki zmiennopozycyjnej. Nie trzeba obsługiwać przypadków brzegowych (liczby zdenormalizowane, wartość *NaN*, nieskończoności).

```
unsigned mulf(unsigned a, unsigned b);
```

Zadanie 6. Napisz program, tj. procedurę `main`, który wczyta z linii poleceń listę ciągów znaków reprezentujących liczby całkowite, skonwertuje je do typu `long` (funkcją `atol`) i umieści w tablicy ulokowanej na stosie. Następnie należy odnaleźć element minimalny i maksymalny tablicy, po czym obydwie wartości wydrukować na standardowe wyjście (funkcją `printf`).

Wskazówka: Dla procedur o zmiennej liczbie parametrów `%eax` określa liczbę parametrów przekazywanych przez rejestry `%xmm`.

Zadanie 7. Napisz samodzielny program[5], który będzie wczytywał ze standardowego wejścia znak po znaku, zamieniał duże litery na małe, a małe na duże, po czym drukował zmodyfikowany znak na standardowe wyjście. Zakładamy, że w napisie będą tylko znaki standardu `ASCII`. Do wczytywania i wypisywania znaków użyj wywołań systemowych `read` i `write`. Program ma zakończyć działanie, kiedy wywołanie `read` zwróci `0`, co oznacza napotkanie końca strumienia. Bufor na znaki należy trzymać w sekcji `.bss`.

Zadanie 8. Napisz procedurę wyznaczającą iteracyjnie pierwiastek kwadratowy liczby `x` za pomocą wzoru $x_{n+1} = \frac{1}{2}(x_n + a/x_n)$. Obliczenia należy zakończyć, gdy $|x_{n+1} - x_n| < \epsilon$. Pamiętaj, że instrukcja dzielenia wykonuje się dłużej niż instrukcja mnożenia. Stałe zmiennopozycyjne należy trzymać w sekcji `.rodata`.

```
double approx_sqrt(double x, double epsilon);
```

Prezentując rozwiązanie użyj gdb, aby wyświetlać aproksymowaną wartość na początku każdej iteracji.

Zadanie 9 (bonus). Poniższą funkcję zaimplementuj jako **wstawkę asemblerową** kompilatora gcc. Składnię objaśniono w rozdziale „Inline Assembly Code" książki Advanced Linux Programming[6].

$$\texttt{long}\ adds(\texttt{long}\ x, \texttt{long}\ y) = \begin{cases} \texttt{LONG\_MIN} & \text{dla } x+y \leq \texttt{LONG\_MIN} \\ \texttt{LONG\_MAX} & \text{dla } x+y \geq \texttt{LONG\_MAX} \\ x+y & \text{w p.p.} \end{cases}$$

Zwróć uwagę na następujące przypadki wymagające specjalnego potraktowania:

- wstawka musi nadawać się do wielokrotnego użycia w obrębie tej samej funkcji,
- jeśli korzystasz z instrukcji skoków, to należy używać etykiet lokalnych – inaczej wielokrotne użycie wstawki wygeneruje błąd asemblera,
- jeśli modyfikujesz rejestry będące argumentami wstawki (`input operands`), to należy użyć odpowiednich modyfikatorów więzów (`constraint modifiers`),
- jeśli używasz dodatkowych rejestrów, to należy je wpisać na listę `clobbers`.

Szczegółowe informacje dot. wstawek asemblerowych można znaleźć na stronie gcc: Extended Asm.

[5] Nie można korzystać z procedur bibliotecznych. Pliki należy skonsolidować z opcją `-nostdlib`.
[6] `http://richard.esplins.org/static/downloads/linux_book.pdf`