Architektury systemów komputerowych 2016

Lista zadań nr 7

Na zajęcia 13, 14, 18 i 19 kwietnia 2016

UWAGA! Zaimplementowane procedury należy przetestować wołając je z poziomu języka `C` z przykładowymi wartościami (np. przeczytanymi z linii poleceń). Rozwiązania bez testów nie będą akceptowane! Sekcja `.symtab` musi być prawidłowo wypełniona – tj. każdemu zdefiniowanemu symbolowi należy przypisać rozmiar i typ. Należy być przygotowanym do wytłumaczenia używanych instrukcji i dyrektyw. Pliki źródłowe należy napisać samodzielnie bez patrzenia na kod generowany przez `gcc`!

Zapoznaj się ze przykładami ze strony GNU Assembler Examples. Należy wykorzystywać dyrektywy opisane w GNU as: Assembler Directives. Magicznym stałym należy przypisać nazwy, np.: `MIN_INT = 0x7fffffff`, `exit = 60`. Jeśli używasz dużej ilości rejestrów do przechowywania zmiennych lokalnych, to dla czytelności kodu przypisz im nazwy, np.: `epsilon = %xmm6`, `arg0 = %rdi`.

W trakcie prezentacji zadania będziecie proszeni o uruchomienie swojego programu pod kontrolą debuggera `gdb`. Będziecie nadzorować wykonanie programu, tj. wyświetlać zawartość rejestrów, stosu, pamięci; ustawiać punkty wstrzymań i obserwacji. Należy się do tego odpowiednio przygotować czytając RMS's gdb Debugger Tutorial i GDB Quick Reference.

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);
```

Zadanie 2. Zaimplementuj procedurę wyznaczającą największy wspólny dzielnik oraz najmniejszą wspólną wielokrotność dwóch liczb naturalnych metodą iteracyjną.

```
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`.

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 (funkcja `main`), który wczyta z linii poleceń listę ciągów znaków reprezentujących liczby całkowite, skonwertuje je do typu `long` (funkcja `atol`) i umieści w tablicy ulokowanej na stosie. Następnie należy w tej tablicy odnaleźć element minimalny i maksymalny, po czym wydrukować je na standardowe wyjście (funkcja `printf`).

Zadanie 7. Napisz samodzielny program (opcja `-nostdlib`), 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 danej liczby za pomocą wzoru $x_{n+1} = \frac{x_n + a/x_n}{2}$. 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);
```

W trakcie prezentacji zadania użyj gdb, aby zatrzymać się na początku każdej iteracji i wyświetlić aproksymowaną wartość.

Zadanie 9. Poniższą funkcję zaimplementuj jako wstawkę asemblerową dla kompilatora `gcc`.

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

Zwróć uwagę na następujące przypadki wymagające specjalnego potraktowania:

- 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.