

Projekt 5 – część pierwsza: generowanie kodu

1. WSTĘP.

Celem niniejszego projektu jest generowanie kodu dla pewnego fragmentu języka C– opisanego poniżej. Wygenerowany kod powinien być zgodny z operacyjną semantyką języka C– opisaną tutaj:

<http://www.math.us.edu.pl/~pgladki/teaching/2011-2012/tk-cminusminus.html#opsem>

1.1. Fragment języka, który należy uwzględnić. Na potrzeby niniejszego projektu Wasz program powinien obsługiwać programy napisane w języku C– zawierające (lub, odpowiednio, nie zawierające) wypisane poniżej elementy:

Zmienne:

Globalne: zmienne typu *int*, *char*, *array of int*, *array of char*;

Lokalne: co najwyżej jeden parametr formalny, który nie jest typu *int*, *char*, *array of int*, *array of char* (ale nie dodatkowe zmienne lokalne).

Zdania: zdanie zawierające wywołania zera lub więcej zadań lub procedurę wywołuje (tj. brak zdań *if*, *for*, *while*), wraz z następującymi dodatkowymi ograniczeniami:

- zadania są “proste” w następującym sensie: lewa strona jest zmienną skalarną (tj. nie odwołującą się do żadnych tablic), a prawa strona jest zmienną skalarną lub stałą. W szczególności oznacza to, że w tym projekcie nie będziemy zajmować się odwołaniami do tablic: jedyny powód, dla którego zajmujemy się tu tablicami (w sekcji **Zmienne** powyżej) jest umożliwienie odwoływania się do *print_string*(“\n”), co ułatwia wydruk wartości.
- wywołanie procedury zawiera co najwyżej jeden argument, który jest typu *int*, *char*, *array of int*, *array of char*.

Wyrażenia: zmienne skalarne, stałe typu *int*, *char* oraz ciągi stałych (tj. bez odwołań do operatorów, tablic, czy wywołań funkcji)

1.2. Docelowa architektura. Docelową architekturą dla tego projektu będzie procesor MIPS R2000. Wasz kompilator wygeneruje kod dla tego procesora; kod będzie wykonywany za pomocą symulatora SPIM

<http://spimsimulator.sourceforge.net/>

dostępnego w wersji na Windows, Maca i Linuksa.

1.3. Operacje wejścia i wyjścia. Programy testujące będą wykorzystywały następujące dwie funkcje celem wydrukowania rezultatów obliczeń:

```
extern void print_int(int x);
```

```
extern void print_string(char x[]);
```

Na przykład, program testujący może wyglądać tak:

```
extern void print_int(int x);
```

```
extern void print_string(char x[]);
```

```
void main( void ) { print_int(12345); print_string("\n"); }
```

Poza wygenerowanym kodem powinniście też wygenerować sekwencję instrukcji dla funkcji *print_int()*, *print_string()* tak, jak jest to opisane w notatkach o tłumaczeniu kodu na assembler MIPSa:

<http://www.math.us.edu.pl/~pgladki/teaching/2011-2012/tk-3addr2spim.pdf>

1.4. Testowanie programów. Możecie zobaczyć “spodziewane wyjście” dla Waszych programów testujących najpierw kompilując je w *gcc* z definicjami dla *print_int()*, *print_string()* dostarczonymi w osobnym pliku i podlinkowanymi w oczywisty sposób.

1.5. Dodatkowa dokumentacja. Następujące dokumenty mogą okazać się przydatne w pracy:

- (1) Instrukcja dla kodu pośredniego w C-:
<http://www.math.us.edu.pl/~pgladki/teaching/2011-2012/tk-intcode.html>
- (2) notatki o tłumaczeniu kodu na assembler MIPSa:
<http://www.math.us.edu.pl/~pgladki/teaching/2011-2012/tk-3addr2spim.pdf>
- (3) dokumentacja do SIMPa:
<http://spimsimulator.sourceforge.net/>

2. WYWOŁYWANIE PROGRAMU.

Plik wykonywalny powinien się nazywać *compile* i powinien czytać z pliku *stdin* i zapisywać do pliku *stdout*. Komunikaty o błędach powinny być zapisane do pliku *stderr*. Razem z zadaniem dostarczyć wszystkie źródłówki.