# bfc64 - dokumentacja

### Grzegorz Koperwas

### 17 grudnia 2020

## 1. Kompilowanie

### Wymagania do kompilacji:

- gcc kompilator
- make
- python generowanie pliku template.cpp (patrz kompilator)
- xelatex dokumentacja

Kompilujemy poleceniem make.

Instalujemy poleceniem sudo make install

Dokumentacje kompilujemy poleceniem make docs.

Przykładowy test uruchamiamy poleceniem make test, powinien uruchomić emulator Vice z programem.

#### Wymagania do używania

- kickassembler<sup>1</sup> bfc64 generuje pliki .asm dla tego assemblera
- System Linux, testowane wyłącznie na Arch'u

Program uruchamiamy poleceniem bfc64 <ścieżka do pliku źródłowego>, utworzy on plik a.asm gotowy do przetworzenia kickassemblerem za pomocą polecenia kickassa.asm. Utworzy on nam plik wykonywalny a.prg dla commodore 64 lub emulatora.

Emulator Vice<sup>2</sup> uruchomi automatycznie programy poleceniem:

Grzegorz Koperwas 1

 $<sup>1 + \</sup>frac{1}{\text{http:}} / \frac{\text{http:}}{\text{theweb.dk/KickAssembler/Main.html\#frontpage}}$ 

<sup>&</sup>lt;sup>2</sup>https://vice-emu.sourceforge.io/

## 2. Działanie kompilatora bfc64

Kompilator składa się z trzech głównych części:

- Parsera zamieniającego pliki tekstowe na listę symboli
- Optymalizatora zamieniającego sąsiednie ++++-- na operacje dodawania lub odejmowania.
- Kompilatora zamieniającego listę symboli na kod assemblera korzystając z funkcji z przestrzeni arch

#### Parser

Parser jest funkcją parseSourceFile, która przyjmuje jako argument plik z kodem źródłowym. Zamienia ona wewnętrznie znaki języka brainfuck na odpowiadające im symbole według tablicy 2.

Symbol	Wartość
+	inc
-	$\operatorname{dec}$
<	left
>	right
[	loopBegin
]	loopEnd
,	in
	out

Tablica 1: Symbole oraz odpowiadające im wartości z Symbol Type

Dodatkowo parser wyświetla ostrzeżenia w przypadku jeśli w trakcie pętli za każdą iteracją jest porównywalna inna komórka pamięci, na przykład dla pętli [>><] zostanie wyświetlone ostrzeżenie wraz z numerem lini początkowym i końcowym.

Jeśli parser napotka ] bez poprzedniego [ czy nie znajdzie w pliku końca pętli przed jego końcem zgłosi on błąd użytkownikowi i nie skompiluje programu.

Parser traktuje wszystkie inne znaki jako komentarz.

### Optymalizator

W celu ograniczenia pamięci potrzebnej na załadowanie programu optymalizator zamienia sąsiednie symbole + oraz - na symbole specjalne add oraz subtract. W planach jest dodanie zamieniania > i < na symbole specjalne jmpLeft oraz jmpRight

## Kompilator

Kompilator tworzy stringa na podstawie listy symboli z parsera za pomocą wzorców generowanych podczas kompilacji z plików w folderze processor/arch/c64. Dodatkowo na początek dołącza arch::begin a na koniec arch::end

Wzorce, z których korzysta kompilator są generowane automatycznie skryptem pythona templateGen.py. Generuje on plik źródłowy template.cpp z pomocą pliku-wzorca

template.cpp.template, do którego podstawia za placeholdery<sup>3</sup> odpowiednie sumy stringów oraz parametrów funkcji według plików-wzorców assemblera, w których podmienia label() na argument label itd. Opis wzorców w sekcji 3..

Wygenerowany string program zapisuje do pliku wyjściowego.

## 3. Opis działania skompilowanego programu

### Wyzwania architektury 6502/6510/commodore 64

Procesory MOS 6502/6510 posiadają 8 bitową szynę danych oraz 16 bitową szynę adresową. Z tego powodu "pointery" muszą się znajdować w pierwszych 256 bajtach pamięci, tak zwanej "zeropage". Kompilator umieszcza w pamięci o adresie \$00FB adres \$7300, który jest adresem początku taśmy.

Procesor posiada opcje indeksowania pamięci rejestrem Y, taki odpowiednik (pointer + rejestr $_Y$ )\* w języku C++. Jednak jako iż rejestr Y jest 8 bitowy, a chcemy taśmę o długości większej od 256 to ruch po taśmie w lewo wygląda tak:

```
cpy #$00  // compare Y to 0
bne label()  // if Y != 0 goto label
dec $00fb + 1 // decrement tape pointer
label():
dey  // decrement Y
```

#### Taśma

Definicja taśmy, w pliku end.asm, wygląda tak:

```
*=$7300 "Tape" // at address $C000
. fill 1024, 0 // place 1024 zeros
```

Generuje ona taśmę o długości 1024 bajtów, jednak jako iż kompilator nie zabezpiecza nas przed "wyjściem" z jej przestrzeni, zatem jeśli komuś chce się pisać dużo > i wiedząc iż adres pierwszej komórki to \$73FF/4 możemy zmieniać kolory tła, tekstu, odtwarzać muzykę oraz nadpisywać program.

## Wypisywanie znaków na ekran i ich odczyt

Kernal commodore 64 posiada funkcje, które realizują te zadania.

Pod adresem \$FFCF jest funkcja, która umieszcza w rejestrze A wartość wprowadzoną przez użytkownika.

Pod adresem \$FFD2 jest funkcja, która wartość w rejestrze A wypisuje na ekran.

Grzegorz Koperwas

 $<sup>^3\</sup>mathrm{W}$  formie nazwa

<sup>&</sup>lt;sup>4</sup>Początek taśmy \$7300 + początkowa wartość Y równa \$ff

Wadą tych funkcji jest to iż nie używają zestawu znaków ASCII, tylko własnego PETSCII $^5$ . Z tego powodu przy wypisywaniu na ekran za pomocą . trzeba ustawiać wartość komórki na wartości PETSCIII

Grzegorz Koperwas 4

 $<sup>^5 \</sup>mathrm{https://www.c64-wiki.com/wiki/PETSCII}$