

Wrocław, dn. 10 kwietnia 2016r

Jakub Pomykała, 209897  
PN-P-8

prowadzący: prof. Janusz Biernat

Laboratorium Architektury Komputerów  
*Podstawy uruchamiania programów assemblerowych  
na platformie Linux/x86*

## 1 Treść ćwiczenia

### Zakres ćwiczenia:

- Zapoznanie się z podstawami pisania, linkowania i uruchamiania programów w języku assembler.

## 2 Przebieg ćwiczenia

W celu uruchomienia powyższego programu należało skompilować, a następnie skonsolidować przy pomocy poniższych instrukcji.

```
#kompilacja  
as lab.s -o lab.o
```

```
#konsolidacja  
ld lab.o -o lab
```

```
#uruchomienie  
./lab
```

### 2.1 Wczytywanie znaków z klawiatury do bufora

Program miał zapisać znaki wprowadzone przez użytkownika do bufora o nazwie bufor.

```
%kod
```

### 2.2 Operacje na znakach w buforze

Powyższy fragment programu przygotowuje do wykonania i wykonuje funkcję systemową `SYSWRITE`. Użyta na początku dyrektywa `.global` ma za zadanie zdefiniowanie punktu wejścia do programu, czyli poinformowanie systemowego programu ładującego o punkcie, w którym powinno zacząć się jego wykonanie.

Jak widać z powyższego fragmentu kodu, wywołanie funkcji systemowych na platformie *Linux/x86* polega na

1. umieszczeniu numeru funkcji w rejestrze procesora `eax`,
2. umieszczeniu jej argumentów w rejestrach `ebx`, `ecx`, ...,
3. wykonanie przerwania programowego `0x80`.

Należy tu zauważyć, że składnia poszczególnych instrukcji i trybów adresowania jest odmienna od tej stosowanej na platformie *DOS/WINDOWS* – np. kolejność argumentów instrukcji jest odwrotna (od lewej do prawej).

Ostatni krok, który należy wykonać aby dokończyć program, to wywołanie funkcji systemowej `SYSEXIT` z argumentem `EXIT_SUCCESS`. Operacja ta jest przygotowywana analogicznie.

```
mov $SYSEXIT, %eax      ; funkcja do wywołania - SYSEXIT
mov $EXIT_SUCCESS, %ebx ; 1 arg. -- kod wyjścia z programu
int $0x80               ; wywołanie przerwania programowego -
                        ; wykonanie funkcji systemowej.
```

## 2.3 Uruchomienie programu pod kontrolą gdb

Dzięki przygotowanemu plikowi `Makefile` proces kompilacji programu sprowadzał się do wydania powłocie polecenia `make`.

```
[kberezow@localhost src]$ make
as -o hello.o hello.s
ld -o hello hello.o
[kberezow@localhost src]$
```

Uruchomienie go pod kontrolą programu `gdb` następuje poprzez wydanie polecenia `gdb hello`. Przykład kompletnej sesji z programem `gdb` uwidoczniony jest na rys. 1.

Pod kontrolą programu `gdb` można uruchomić program w trybie pracy krokowej bądź ciągłej oraz podejrzeć zarówno tekst programu jak i wartości zawarte w pamięci i rejestrach procesora. W przypadku naszego pierwszego programu możemy uzyskać jedynie wtórnie rozkodowany tekst programu, gdyż nie umieściliśmy żadnej informacji uruchomieniowej w jego treści.

```
(gdb) disassemble _start
Dump of assembler code for function _start:
0x804808e <_start>:      mov     $0x4,%eax
0x8048093 <_start+5>:    mov     $0x1,%ebx
0x8048098 <_start+10>:   mov     $0x8048080,%ecx
0x804809d <_start+15>:   mov     $0xe,%edx
0x80480a2 <_start+20>:   int     $0x80
0x80480a4 <_start+22>:   mov     $0x1,%eax
0x80480a9 <_start+27>:   mov     $0x0,%ebx
0x80480ae <_start+32>:   int     $0x80
End of assembler dump.
(gdb)
```

---

```
[kberezow@localhost src]$ gdb hello
GNU gdb 5.3-22mdk (Mandrake Linux)
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i586-mandrake-linux-gnu"...
(gdb) run
Starting program: /home/kberezow/projects/AK/src/hello
Hello, world!

Program exited normally.
(gdb) quit
[kberezow@localhost src]$
```

---

Rysunek 1: Przebieg najkrótszej sesji uruchomieniowej dla programu `hello` w środowisku `gdb`.

Jak widać wyświetlony program odpowiada (z dokładnością do wyliczonych wartości etykiet) zawartości pliku źródłowego.

Program uruchomieniowy możemy również wykorzystać do podejrzenia zawartości pamięci, np. używając polecenia `p[rint]`<sup>1</sup>.

```
(gdb) print {char}0x8048080
$6 = 72 'H'
(gdb) print {char}(0x8048080+1)
$8 = 101 'e'
(gdb) print {char}(0x8048080+2)
$9 = 108 'l'
(gdb) print {char}(0x8048080+3)
$10 = 108 'l'
(gdb) print {char}(0x8048080+4)
$11 = 111 'o'
(gdb) print {char}(0x8048080+5)
$12 = 44 ', '
(gdb) print {char}(0x8048080+6)
$13 = 32 ' '
(gdb) print {char}(0x8048080+7)
$14 = 119 'w'
(gdb) print {char}(0x8048080+8)
$15 = 111 'o'
(gdb) print {char}(0x8048080+9)
```

---

<sup>1</sup>Program `gdb` umożliwia skracanie nazw poleceń, jeżeli powstały w wyniku tego procesu skrót jest jednoznaczny w obrębie zbioru jego poleceń. Aby odzwierciedlić tę własność, zachowując równocześnie przejrzystość treści sprawozdania, będziemy wyróżniać część polecenia, którą można bezpiecznie odrzucić stosując nawiasy kwadratowe `[]`.

```

$16 = 114 'r'
(gdb) print {char}(0x8048080+10)
$18 = 108 'l'
(gdb) print {char}(0x8048080+11)
$19 = 100 'd'
(gdb) print {char}(0x8048080+12)
$20 = 33 '!'
(gdb) print {char}(0x8048080+13)
$21 = 10 '\n'
(gdb)

```

Ze względu na to, że w pakiecie kompilatorów GNU gcc informację pomocniczą dla programu uruchomieniowego kompilatory umieszczają na poziomie kodu asemblerowego, aby przeprowadzić bardziej zaawansowane operacje śledzenia przebiegu wykonania programu, należy skompilować nasz program kompilatorem języka C, w trybie z uzupełnianiem informacji uruchomieniowej (opcja kompilatora `-g`). Oznacza to, że aby precyzyjnie śledzić przebieg programów asemblerowych musieliśmy zmienić nasz plik sterujący Makefile np. do następującej postaci:

```

# reguła kompilacji i linkowania
hello: hello.s
    gcc -g -o hello hello.s

```

a punkt wejścia do programu zmienić z `.global _start` do `.global main` tak, aby zapewnić poprawność łączenia z programem startowym języka C.

Dzięki takim zmianom mogliśmy zastosować krokowe śledzenie przebiegu naszego programu, korzystając z następujących poleceń programu gdb [6]:

- `b[reak]` – ustawienie pułapki,
- `r[un]` – uruchomienie programu,
- `s[tep]`, `n[ext]` – wykonanie kroku w pracy krokowej (omówienie różnic pomiędzy poleceniami `s` i `n` można znaleźć się w [6]),
- `inf[o]` – wyświetlenie informacji o wykonywanym programie (np. `inf[o] reg[isters]`).

Przebieg typowej sesji pracy krokowej, wykazujący nabyte umiejętności pracy z programem uruchomieniowym, widoczny jest poniżej.

```

(gdb) b main
Breakpoint 1 at 0x804832e: file hello.s, line 20.
(gdb) run
Starting program: /home/kberezow/projects/AK/src/hello
1
Breakpoint 1, main () at hello.s:20
20      mov $SYSWRITE, %eax
Current language:  auto; currently asm

```

```

(gdb) list
15     msg_hello_len = . - msg_hello
16
17     .global main
18
19     main:
20         mov $SYSWRITE, %eax
21         mov $STDOUT, %ebx
22         mov $msg_hello, %ecx
23         mov $msg_hello_len, %edx
24         int $0x80
(gdb) info registers
eax             0x1             1
ecx             0x401517c8      1075124168
edx             0x0             0
ebx             0x40153f50      1075134288
esp             0xbffff50c     0xbffff50c
ebp             0xbffff528     0xbffff528
esi             0x40012780      1073817472
edi             0xbffff554     -1073744556
eip             0x804832e       0x804832e
eflags          0x246          582
cs              0x23           35
ss              0x2b           43
ds              0x2b           43
es              0x2b           43
fs              0x0            0
gs              0x0            0
fctrl           0x37f          895
fstat           0x0            0
ftag            0xffff         65535
fiseg           0x0            0
fioff           0x0            0
foseg           0x0            0
fooff           0x0            0
fop             0x0            0
mxcsr           0x0            0
orig_eax        0xffffffff     -1
(gdb) s
21         mov $STDOUT, %ebx
(gdb)
22         mov $msg_hello, %ecx
(gdb)
23         mov $msg_hello_len, %edx
(gdb)
24         int $0x80
(gdb)
Hello, world!

```

```
27      mov $EXIT_SUCCESS, %ebx
(gdb)
```

### 3 Podsumowanie i wnioski

Wykonane ćwiczenie pozwoliło nam opanować umiejętności tworzenia i uruchamiania programów assemblerowych na platformie *Linux/x86*. Jego wykonanie uświadomiło nam, iż pomiędzy platformą *Linux/x86*, a znaną nam wcześniej platformą *DOS/WINDOWS* istnieją zarówno podobieństwa jak i różnice.

Pierwszą zaobserwowaną i dotkliwą dla nas różnicą jest odmienna składnia assemblera, niezgodna ze składnią propagowaną przez firmę *INTEL*. Dla programisty, który nabył już pewnych doświadczeń w tworzeniu programów assemblerowych dla systemów *DOS/WINDOWS*, szczególnie niebezpieczna może być odwrotna kolejność argumentów instrukcji assemblera w składni AT&T stosowanej w assemblerze *GNU as*, gdyż może prowadzić do generowania trudnych do wykrycia błędów semantycznych. Drugą istotną różnicą, jest brak możliwości wygenerowania informacji dla programu uruchomieniowego w procesie asemblacji programów, która zmusza do uruchamiania programów assemblerowych w środowisku kompilatora języka C bądź C++. To wszystko powoduje, iż zmiana platformy z *DOS/WINDOWS* na *Linux/x86* raczej nie będzie intuicyjna dla programisty assemblera.

Pomimo występowania różnic, na bazie stworzonego programu, możemy również zauważyć, że występują istotne podobieństwa w sposobie komunikacji pomiędzy programem użytkowym a systemem. Zarówno w systemie *DOS* jak i *Linux* stosuje się instrukcję przerwania programowego. Zwraca uwagę uporządkowany sposób przekazywania argumentów do funkcji systemowych, który, w przeciwieństwie do systemu *DOS*, jest bardzo intuicyjny.

Odmienny, w stosunku do znanego z systemu *DOS* programu *Turbo Debugger* (*td.exe*), jest również sposób pracy z programem uruchomieniowym – choć zestaw dostępnych poleceń służących do analizy programów jest w dużej mierze podobny, a być może nawet szerszy, to praca z tym narzędziem wymaga od programisty większej wiedzy o nim samym. Przede wszystkim, w programie *gdb*, programista jest zmuszony do tego, aby wszystkie informacje związane z przebiegiem uruchamianego programu uzyskiwać wydając właściwe polecenia – w zasadzie żadna informacja, poza całkowicie podstawową, nie jest dostarczana przez program samoczynnie. Jest to związane z tym, że program *gdb* jest przystosowany do pracy z różnymi interfejsami użytkownika, a także może być osadzany w środowisku innych programów [6]. Przykładem mogą być tu programy dostarczające rozbudowany interfejs graficzny dla różnych programów uruchomieniowych, jak *Data Display Debugger* (*ddd*), czy możliwość uruchamiania i analizy programów bezpośrednio w środowisku edytora *EMACS* zademonstrowana na rys. ??.

### Literatura

- [1] J. Biernat, *Profesjonalne przygotowanie publikacji*, materiały konferencyjne X Krajowej Konferencji KOWBAN, str. 401–408, Wyd. WTN, Wrocław, 2003.

- [2] M. Węgrzyn, *Zastosowanie pakietu MS Word do przygotowania publikacji naukowych*, materiały konferencyjne X Krajowej Konferencji KOWBAN, str. 409–414, Wyd. WTN, Wrocław, 2003.
- [3] E. Polański i inni, *Nowy słownik ortograficzny PWN z zasadami pisowni i interpunkcji*, PWN, Warszawa, 1997
- [4] `/usr/include/asm/unistd.h`, plik nagłówkowy kompilatora `gcc` z listą kodów funkcji systemowych systemu *Linux*.
- [5] `info gas`, dokumentacja `info` asemblera GNU `as`.
- [6] `info gdb`, dokumentacja `info` programu GNU `gdb`.
- [7] `info ld`, dokumentacja `info` programu GNU `ld`.
- [8] `info make`, dokumentacja `info` programu GNU `make`.
- [9] <http://www.linuxassembly.org>, witryna internetowa z informacjami dla programistów asemblera dla platformy *Linux/x86*.

# A Wytyczne i uwagi dotyczące dokumentu sprawozdania

## A.1 Uwagi ogólne

Zgodnie z ogłoszonymi wcześniej założeniami dotyczącymi sprawozdawczości, sprawozdania nie zawierają wstępów teoretycznych czy omówienia użytych narzędzi w zakresie szerszym niż to wynika z tematu ćwiczenia (*vide*: brak omówienia programu `make` i formatu plików `Makefile` w niniejszym dokumencie).

Sprawozdanie powinno być strony formy wypowiedzi zbliżone do rozprawki. Tym bardziej uczulam Państwa na treści zawarte w podrozdziale 3. Zawarty tam tekst powinien między innymi w syntetyczny sposób posumować nabytą wiedzę. Poza tym, jest tam miejsce na Państwa przemyślenia i obserwacje. Ostrzegam, że ta część sprawozdania – niewłaściwie napisana czy niestarannie przemyślana – może Państwa kosztować najwięcej punktów. Nie oznacza to jednak, że brak sformułowania tez (celu ćwiczenia) we wprowadzeniu ujdzie Państwu na sucho!

## A.2 Wymagania składu

Układ strony i akapitów powinien zachowywać następujące własności:

- krój czcionki podstawowej – *Times*, 12pt (*Computer Modern*, 12pt),
- interlinia – 1.5,
- marginesy – lewy, prawy, górny, dolny – 2,5 cm,
- nagłówki numerowane (preferowane cyfry arabskie),
- strony numerowane (preferowane cyfry arabskie) od strony tytułowej, jednakże bez umieszczania numeru na stronie tytułowej,
- wyrównanie tekstu obustronne, dzielenie wyrazów opcjonalne,
- rysunki i tabele opatrzone numerem i tytułem, umieszczane na górze bądź na dole strony (nie w ciągu akapitu),
- odniesienia do rysunków/tabel w tekście poprzez podanie numeru rysunku,
- cytaty z kodu źródłowego umieszczane w ciągu akapitu lub jako rysunki,
- strona tytułowa zgodna z dostarczonym wzorcem.

Proszę o wprowadzanie i stosowanie konwencji typograficznych związanych z przytaczaniem nazw własnych, symbolicznych, cytowaniem kodu czy wyjścia programów. Konwencje takie przyjmują Państwo i pielęgnują na własną rękę, jednakże konsekwencja ich przestrzegania podlega ocenie.

Szczęśliwi użytkownicy systemu  $\text{\LaTeX} 2_{\epsilon}$  mogą posłużyć się plikiem źródłowym niniejszego dokumentu jako wzorcem do konstrukcji własnych sprawozdań i zapomnieć o znakomitej większości problemów składu. Użytkownicy edytora *MS Word* powinni na własną rękę przygotować właściwy szablon dokumentu. Tym ostatnim polecam lekturę artykułów [1, 2], których wersje elektroniczne być może posiada prof. J. Biernat, a do wglądu i odbicia dostępne są również u mnie.



## A.3 Język

Przypominam, że językiem urzędowym w naszym kraju jest język polski co oznacza, że należy przestrzegać właściwych mu zasad [3]:

- interpunkcji (!),
- stylu (!!),
- gramatyki (!!!) oraz
- ortografii (!!!!).

Zwracam więc uwagę na to, że z powyższej listy zasad, narzędzia automatycznej korekty zadowolająco radzą sobie jedynie z ortografią.

**UWAGA:** pozostawianie jednoliterowych spójników (a, i, w, z, ...) na końcu wiersza, choć dopuszczone przez Radę Języka Polskiego, wciąż powszechnie uważa się za brak profesjonalizmu w przygotowywaniu dokumentów.

## A.4 Informacje o autorach

Kod zespołu (umieszczany w lewym górnym rogu strony tytułowej) składa się z czterech elementów rozdzielonych znakiem ‘/’. Kolejne pola oznaczają:

1. XX – dzień tygodnia (wartości: PN, WT, SR, CZ, PT,
2. Y – „parzystość” tygodnia (wartości: P, N),
3. ZZ – godzina rozpoczęcia zajęć (wartości przykładowe: 08, 11, ...)
4. T – numer zespołu: wartości przykładowe: 1, 2, ...

Dwuelementowa lista autorów umieszczana pod kodem zespołu powinna być rosnąco posortowana ze względu na nazwiska (oczywiście stosujemy sortowanie leksykograficzne).

## A.5 Podsumowanie

Przygotowanie całości tego dokumentu, wraz ze stworzeniem i uruchomieniem programu „Hello, world!” oraz przygotowaniem niniejszego komentarza, zajęło mi około czterech godzin, więc nie jest to dla Państwa wysiłek ponad miarę!