Zespół: **XX/Y/ZZ/T** Wrocław, dn. 12 października 2004

Imie i nazwisko 1, nr indeksu 1
Imie i nazwisko 2, nr indeksu 2

Ocena:
Oddano:

Podstawy uruchamiania programów asemblerowych *na platformie* Linux/x86

sprawozdanie z laboratorium przedmiotu „Architektura Komputerów”

Rok akad. 2003/2004, kierunek: INF, specjalność: Ixx

PROWADZĄCY:
dr inż. Krzystof Berezowski

Spis treści

1 Cel ćwiczenia

Celem ćwiczenia było zapoznanie się z technikami tworzenia i uruchamiania programów napisanych w języku asemblera na platformie *Linux/x86*. Zgodnie z przekazanymi wymaganiami w ramach ćwiczenia mieliśmy zapoznać się z następującymi zagadnieniami:

- tworzeniem poprawnych składniowo i semantycznie plików źródłowych dla asemblera `GNU as`,
- właściwym korzystaniem z wybranych funkcji systemowych,
- uruchamianiem i analizą programów asemblerowych pod kontrolą programu uruchomieniowego `GNU gdb`.

Cel ćwiczenia, zgodnie z wymaganiami prowadzącego, został zrealizowany poprzez stworzenie i uruchomienie programu programu typu `,,Hello, world!''` wykorzystującego funkcje systemowe nr `1 (SYSEXIT)` oraz nr `3 (SYSWITE)` przerwania programowego `0x80`.

2 Przebieg ćwiczenia

Wypełniając wymagania przygotowaliśmy plik źródłowy `hello.s` oraz plik sterujący `Makefile` dla programu sterującego procesem kompilacji `make` [8]. Jego zastosowanie ułatwiło nam pracę ze środowiskiem asemblera `GNU as` [5] oraz programu łączącego `GNU ld` [7]. Ze względu na niewielką objętość przygotowanych plików zostaną one w całości zacytowane w tekście sprawozdania.

Zadaniem programu `hello.s` było wyprowadzenie napisu na standardowy strumień wyjściowy `STDOUT` oraz poprawne zakończenie się. W celu uproszczenia procesu kompilacji przygotowaliśmy pomocniczy plik sterujący `Makefile` o następującej treści [5, 7, 8]:

```
# reguła linkowania
hello: hello.o
```

```
  ld -o hello hello.o

# reguła kompilacji
hello.o: hello.s
  as -o hello.o hello.s
```

Dzięki jego zastosowaniu, zastąpiliśmy ciąg kolejnych wywołań programów `as` i `ld` jednym wywołaniem programu `make`, upraszczając i przyspieszając prowadzone eksperymenty.

2.1 Konstrukcja pliku źródłowego

Przygotowanie pliku źródłowego rozpoczęliśmy od zadeklarowania nazw symbolicznych dla wywoływanych funkcji systemowych oraz wybranych ich argumentów. Numery potrzebnych nam funkcji systemowych odtworzyliśmy na podstawie pliku nagłówkowego `asm/unistd.h` [4].

```
SYSEXIT      = 1
SYSREAD      = 3
SYSWRITE     = 4
STDOUT       = 1
EXIT_SUCCESS = 0
```

Wprowadzone nazwy symboliczne, choć nie stanowią niezbędnego elementu pliku źródłowego, zdecydowanie zwiększają jego czytelność.

Kolejnym krokiem było zdefiniowanie sekcji kodu programu oraz zdefiniowanie w niej stałej tekstowej zawierającej napis `,,Hello, world!''`, która zostanie wyprowadzona na strumień wyjściowy. Ze względu na to, że napis ten nie będzie podlegał modyfikacjom podczas działania programu, mogliśmy umieścić go w sekcji tekstu, a cały nasz program w ogóle nie posiada deklaracji sekcji danych.

```
.align 32
.text

msg_hello: .ascii "Hello, world!\n"
msg_hello_len = . - msg_hello
```

Użyta w preambule dyrektywa `.align 32` zmusza kompilator do wyrównania kodu programu do granicy słowa maszynowego, co ma wpływ na zwiększenie szybkości wymiany danych między procesorem a pamięcią operacyjną.

Ponieważ napis wskazywany przez etykietę `msg_hello` ma zostać wyprowadzony na strumień wyjściowy za pomocą funkcji `SYSWRITE` musimy również ustalić jego długość, gdyż jest to jeden z wymaganych argumentów wywołania tej funkcji. Aby uniknąć wyznaczania tej wielkości ręcznie, co może prowadzić do powstania trudnych do wykrycia i usunięcia błędów czasu wykonania, skorzystaliśmy z pomocy kompilatora, który jest zdolny do wykonywania obliczeń arytmetycznych w trakcie kompilacji. Długość łańcucha została więc wyznaczona jako różnica bieżącej pozycji w sekcji `.text`, reprezentowanej przez operator '`.`', oraz wartości etykiety `msg_hello`. Dzięki temu nazwa symboliczna `msg_hello_len` będzie zawsze rozwijana do poprawnej długości napisu (niezależnie od jego długości) pod warunkiem, że obliczenie to będzie następowało bezpośrednio po deklaracji napisu.

Kolejnym krokiem jest przygotowanie wywołania funcji systemowej `SYSWRITE`.

```
.global _start              ; wskazanie punktu wejścia do programu

_start:
  mov $SYSWRITE, %eax       ; funkcja do wywołania - SYSWRITE
  mov $STDOUT, %ebx         ; 1 arg. - syst. deskryptor stdout
  mov $msg_hello, %ecx      ; 2 arg. - adres początkowy napisu
  mov $msg_hello_len, %edx  ; 3 arg. - długość łańcucha
```

```
int $0x80                ; wywołanie przerwania programowego -
                         ; wykonanie funcji systemowej.
```

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 funcji systemowej.
```

2.2 Uruchomienie programu pod kontrolą `gdb`

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

```
[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`.

```
[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)
```

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]`[1].

```
(gdb) print {char}0x8048080
$6 = 72 'H'
```

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

```
(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)
$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 aseblerowych 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]:

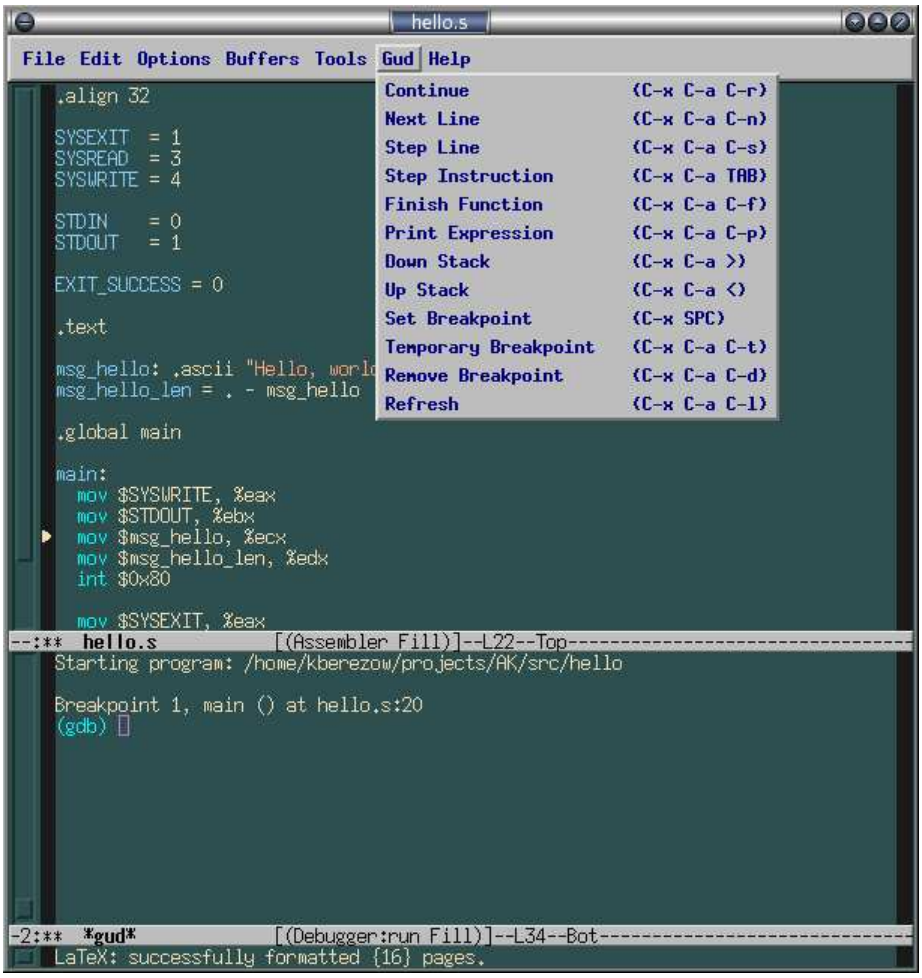
- `b[reak]` – ustawienie pułapki,
- `r[un]` – uruchomienie programu,
- `s[tep], n[ext]` – wykonanie kroku w pracy krokowej (omówienie różnic pomiędzy poleceniami `s` i `n` można znaleźć się w [6]),
- `inf[o]` – wyświetlenie informacji o wykonywanym programie (np. `inf[o] reg[isters]`).

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
l
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)
```



Rysunek 2: Sesja uruchomieniowa programu `hello` wewnątrz środowiska edytora *EMACS* (wykorzystanie pakietu *GUD*).

3 Podsumowanie i wnioski

Wykonane ćwiczenie pozwoliło nam opanować umiejętności tworzenia i uruchamiania programów asemblerowych 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 asemblera, niezgodna ze składnią propagowaną przez firmę *INTEL*. Dla programisty, który nabył już pewnych doświadczeń w tworzeniu programów asemblerowych dla systemów

DOS/WINDOWS, szczególnie niebezpieczna może być odwrotna kolejność argumentów instrukcji asemblera w składni AT&T stosowanej w asemblerze `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 asemblerowych 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 asemblera.

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. 2.

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 tekscie 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 LaTeX 2_ε 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 zadowalająco 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!