Wrocław, 07.05.2018

Adam Karczewski, 235039
WT-N-7

prowadzący: Aleksandra Postawka

Laboratorium Architektury Komputerów

(4) Łączenie różnych języków programowania w jednym projekcie

4.1 Treść ćwiczenia

Zakres i program ćwiczenia:
Celem ćwiczenia była nauka łączenia języka assemblera wraz z językiem C.

Zrealizowanie zadania:
Pierwsze zadanie polegało na napisaniu programu w języku assemblera, który wczyta liczbę typu integer, float, double, wywoła funkcję w języku C, która obliczy sumę argumentów i zwróci wynik. Zarówno do wczytania jaki i wyświetlania wykorzystane miały zostać funkcje standardowej biblioteki języka C.
Drugim zadaniem było napisanie programu w języku C, który dokona konwersji liczby do łańcucha znaków w reprezentacji ósemkowej (konwersja z wykorzystaniem wstawek asemblerowych).
Ostatni program miał zostać napisany w języku C, jego zadaniem było wywołanie funkcji napisanej w języku assemblera, która rozkłada wczytaną liczbę na czynniki pierwsze. Na koniec miały zostać wyświetlone wyniki rozkładu.

4.2 Budowa kodu źródłowego

4.2.1 Dodanie liczb

Pierwszym etapem jest wywołanie funkcji scanf, celem wczytania argumentów. W tym celu w rejestrze `rax` umieszczamy ilość argumentów zmiennoprzecinkowych- 0. Do rejestru rdi zapisujemy pierwszy argument stałoprzecinkowy- adres łańcucha znakowego "`%d %lf %f`", łańcuch ten określa w jakiej kolejności będą podawane dane: integer, double, float. Kolejno do rejestrów `rsi,rdx,rcx` przekazujemy adresy buforów, do których mają zostać zapisane dane. Bufor int i float są 4B, natomiast double 8B. Po umieszczeniu wszystkich argumentów w odpowiednich rejestrach wywołujemy funkcję poleceniem `call`.

```
movq $0, %rax
movq $format_scanf, %rdi
movq $int, %rsi
movq $double, %rdx
```

```
movq $float, %rcx
call scanf
```

Aby wywołać funkcję napisaną w C `double f(int,double,float)`, umieszczamy w rejestrze `rax` liczbę argumentów zmiennoprzecinkowych- 2, kopiujemy do rejestru `rdi` zawartość bufora int, a do rejestrów xmm0 i xmm1 zawartość buforów double i float. Aby skopiować dane do rejestru `xmm` należy użyć mnemonika `movss` dla liczby pojedynczej precyzji i `movsd` dla podwójnej precyzji.

```
movq $0, %rdi
movq $0, %rcx
movq $2, %rax
movq int(,%rcx, 4), %rdi
movsd double, %xmm0
movss float, %xmm1
call f  # wynik w xmm0
```

Funkcja w języku C jest bardzo prosta- dodaje argumenty i zwraca wynik.

```
double f(int x, double y, float z)
{
  double wynik=x+y+z;
  return  wynik;
}
```

Ostatnim etapem jest wyświetlenie wyniku na ekranie za pomocą funkcji printf. Jest to zadanie analogiczne do pobrania danych. Przesyłamy jeden argument zmiennoprzecinkowy, który został zwrócony przez funkcję f i znajduje się w rejestrze `xmm0`. Łańcuch `format_printf` wygląda następująco `"%f"`. Aby zapobiec zmianie ostatniej wartości na stosie przez funkcję printf należy wykonać tzw. workaround- tymczasowo zmienić szczyt stosu.

```
movq $1, %rax
movq $format_printf, %rdi
sub $8, %rsp
call printf
add $8, %rsp
```

Program kompilujemy poleceniem:
`gcc kod_asm.s kod_c.c -o plik_wykonywalny -g`

4.2.2 Konwersja liczby

Program pobiera od użytkownika liczbę i zapisuje ją do zmiennej globalnej.

```
scanf("%d", &liczba);
```

Następnie we wstawce assemblerowej dokonywana jest konwersja. Podana liczba w pętli dzielona jest przez 8, reszta z dzielenia kodowana jest w ascii i zapisywana do tablicy znaków str- zapis od końca. Pętla wykonuje się dopóki wynik z dzielenia nie jest zerem. Przyjęto założenie, że tablica str ma rozmiar 11 i jest wyzerowana.

```
asm(
    "movq %1, %%rax \n"
    "movq $10, %%r8 \n"
    "movq $8, %%r9 \n"
    "petla: \n"
    "movq $0, %%rdx \n"
    "div %%r9 \n"
    "addb $48,%%dl \n"
    "movb %%dl, (%0,%%r8,1) \n"
    "dec %%r8 \n"
    "cmp $0, %%rax \n"
    "jne petla \n"
```

Należy pamiętać, że każda linijka kodu języka assemblera powinna być zapisana jako osobny ciąg znaków zakończonych znakiem końca linii, a odwołania do rejestrów powinny być poprzedzone dodatkowym znakiem %.

Wstawka zakończona jest informacją o zwracanych argumentach- brak, przyjmowanych argumentach: `liczba, str`. Odwołanie do przyjętych argumentów wygląda następująco `%numer`, gdzie numer to numer argumentu. Należy także zapisać, z których rejestrów korzystała wstawka, gdyż ich wartości mogą zostać przez nią zmienione.

```
    :
    : "r"(&str), "m"(liczba)
    :"%rax", "%r8", "%r9", "%rdx", "%rcx"
);
```

Na koniec program wyświetla wynik.

```
printf("%s /n", str );
```

4.2.3 Znajdowanie czynników pierwszych

Program wczytuje liczbę i wywołuję funkcję w języku asemblera, aby było to możliwe należy ją wcześniej zadeklarować.

```
extern int funkcja(int liczba,char* pierwsze);
```

Wywołanie funkcji jest identyczne jak w przypadku funkcji napisanej w języku C.

Funkcja, aby zostać wywołana w języku C musi być napisana w sekcji text i odpowiednio zadeklarowana.

```
.text
.global funkcja
.type funkcja, @function

funkcja:

push %rbp
movq %rsp, %rbp

movq %rdi, %r8
movq $2, %r9
movq $0, %r10
```

Pierwszy argument- liczba do rozkładu kopiowana jest z rejestru `rdi`, przez który została przekazana do rejestru `r8`, `r9` zawiera pierwszą liczbę pierwszą, a `r10` liczbę czynników. Rozkład dokonuje się w zagnieżdżonej pętli.

Pętla wewnętrzna wykonuję się do momentu, dopóki liczba jest podzielna przez czynnik pierwszy. Wynik dzielenia modulo zostaje zakodowany w ascii i zapisany do pamięci (program zakłada, że czynniki pierwsze są cyframi), a liczba do rozkładu na czynnik jest zastępowana wynikiem dzielenia przez czynnik.

```
petla_zewnetrzna:

petla_wewnetrzna:
movq $0, %rdx
movq %r8, %rax
div %r9
cmp $0, %rdx
jne koniec_wewnetrznej

movq %r9, %rdx
add $48, %dl
movb %dl, (%rsi,%r10,1)
inc %r10


movq $0, %rdx
movq %r8, %rax
div %r9
movq %rax, %r8
```

```
jmp petla_wewnetrzna
```

W pętli zewnętrznej wykonywana jest inkrementacja dzielnika, pętla ta wykonuje się dopóki liczba do rozkładu jest większa od jeden.

```
koniec_wewnetrznej:
inc %r9;

cmp $1, %r8
jg petla_zewnetrzna
```

Ostatnim etapem jest zwrócenie ilości czynników po przez rejestr rax i prawidłowe zakończenie funkcji.

Po wywołaniu funkcji następuje wyświetlenie czynników pierwszych na ekranie konsoli.

```
ilosc=funkcja(liczba,pierwsze);
for(int i=0; i<ilosc;i++)
{
  printf("%c \n", pierwsze[i]);
 }
```

4.3 Wnioski

Na zajęciach nauczyłem się łączyć na różne sposoby kod języków C i assemblera.