# Zadanie 2

# Emulator procesora SO

Zaimplementuj w asemblerze x86-64 wołaną z języka C funkcję emulującą działanie procesora SO.

# Architektura procesora SO

Procesor SO ma cztery ośmiobitowe rejestry danych o nazwach: `A`, `D`, `X`, `Y`, ośmiobitowy licznik rozkazów `PC`, jednobitowy znacznik `C` przeniesienia-pożyczki w operacjach arytmetycznych oraz jednobitowy znacznik `Z` ustawiany, gdy wynik operacji arytmetyczno-logicznej jest zerem, a zerowany w przeciwnym przypadku. Po uruchomieniu procesora wartości wszystkich rejestrów i znaczników są wyzerowane.

Adresy w procesorze SO są 8-bitowe. Procesor ma osobne przestrzenie adresowe danych i programu. Pamięć danych zawiera 256 bajtów. Pamięć programu zawiera 256 słów 16-bitowych.

Wszystkie operacje na danych i adresach wykonuje się modulo 256. W trakcie wykonywania instrukcji licznik rozkazów zwiększa się o jeden i wskazuje na kolejną instrukcję do wykonania, chyba że instrukcja wykonuje skok, wtedy oprócz tego zwiększenia do wartości licznika rozkazów dodaje się wartość stałej podanej w kodzie instrukcji. Wszystkie skoki są skokami względnymi.

Lista instrukcji procesora SO podana jest poniżej.

Procesor SO może być jednordzeniowy lub wielordzeniowy.

# Wersja jednordzeniowa

Zaimplementuj w asemblerze x86-64 następującą funkcję wołaną z języka C:

```
so_state_t so_emul(uint16_t const *code, uint8_t *data, size_t steps);
```

Parametr `code` wskazuje na pamięć programu. Parametr `data` wskazuje na pamięć danych. Parametr `steps` mówi, ile kroków, kolejnych instrukcji, ma wykonać emulator. Wynikiem tej funkcji jest instancja struktury reprezentującej aktualny stan procesora, czyli wartości jego rejestrów i znaczników. Stan procesora powinien być pamiętany między kolejnymi wywołaniami tej funkcji. Definicja struktury jest następująca:

```
typedef struct __attribute__((packed)) {
  uint8_t A, D, X, Y, PC;
  uint8_t unused; // Wypełniacz, aby struktura zajmowała 8 bajtów.
  bool    C, Z;
} so_state_t;
```

# Wersja wielordzeniowa

Zaimplementuj w asemblerze x86-64 następującą funkcję wołaną z języka C, gdzie dodatkowy parametr `core` zawiera numer rdzenia. Rdzenie są numerowane do `0` do `CORES - 1`, gdzie `CORES` jest stałą kompilacji.

```
so_state_t so_emul(uint16_t const *code, uint8_t *data, size_t steps, size_t core);
```

Wynikiem tej funkcji jest instancja struktury reprezentującej aktualny stan rdzenia. Każdy rdzeń ma własny zestaw rejestrów i znaczników. Dla każdego rdzenia funkcję `so_emul` uruchamia się w osobnym wątku.

# Instrukcje procesora jednordzeniowego

W poniższych opisach słowo „kod” oznacza kod maszynowy, czyli binarną reprezentację instrukcji lub parametru instrukcji. Instrukcja może mieć parametry oznaczane jako `arg1`, `arg2` lub `imm8`. Parametr `imm8` to 8-bitowa stała. Parametry `arg1`, `arg2` mogą mieć następującą postać:

- `A` – wartość rejestru `A`, kod `0`;
- `D` – wartość rejestru `D`, kod `1`;
- `X` – wartość rejestru `X`, kod `2`;
- `Y` – wartość rejestru `Y`, kod `3`;
- `[X]` – wartość komórki pamięci danych o adresie w rejestrze `X`, kod `4`;
- `[Y]` – wartość komórki pamięci danych o adresie w rejestrze `Y`, kod `5`;
- `[X + D]` – wartość komórki pamięci danych o adresie będącym sumą wartości rejestrów `X` i `D`, kod `6`;
- `[Y + D]` – wartość komórki pamięci danych o adresie będącym sumą wartości rejestrów `Y` i `D`, kod `7`.

Lista instrukcji:

- `MOV arg1, arg2`

  kod `0x0000 + 0x100 * arg1 + 0x0800 * arg2`

  Przepisuje wartość `arg2` do `arg1`. Nie modyfikuje znaczników.

- `OR arg1, arg2`

  kod `0x0002 + 0x100 * arg1 + 0x0800 * arg2`

  Wykonuje bitową alternatywę `arg1` i `arg2`. Umieszcza wynik w `arg1`. Ustawia znacznik `Z` zgodnie z wynikiem operacji. Nie modyfikuje znacznika `C`.

- `ADD arg1, arg2`

  kod `0x0004 + 0x100 * arg1 + 0x0800 * arg2`

  Dodaje wartość `arg2` do `arg1`. Ustawia znacznik `Z` zgodnie z wynikiem operacji. Nie modyfikuje znacznika `C`.

- `SUB arg1, arg2`

  kod `0x0005 + 0x100 * arg1 + 0x0800 * arg2`

  Odejmuje wartość `arg2` od `arg1`. Ustawia znacznik `Z` zgodnie z wynikiem operacji. Nie modyfikuje znacznika `C`.

- `ADC arg1, arg2`

  kod `0x0006 + 0x100 * arg1 + 0x0800 * arg2`

  Dodaje wartości `arg2` oraz `C` do `arg1`. Ustawia znaczniki `C` i `Z` zgodnie z wynikiem operacji.

- `SBB arg1, arg2`

  kod `0x0007 + 0x100 * arg1 + 0x0800 * arg2`

  Odejmuje wartości `arg2` oraz `C` od `arg1`. Ustawia znaczniki `C` i `Z` zgodnie z wynikiem operacji.

- `MOVI arg1, imm8`

  kod `0x4000 + 0x100 * arg1 + imm8`

  Przepisuje wartość `imm8` do `arg1`. Nie modyfikuje znaczników.

- `XORI arg1, imm8`

  kod `0x5800 + 0x100 * arg1 + imm8`

  Wykonuje bitową rozłączną alternatywę `arg1` i `imm8`. Umieszcza wynik w `arg1`. Ustawia znacznik `Z` zgodnie z wynikiem operacji. Nie modyfikuje znacznika `C`.

- `ADDI arg1, imm8`

  kod `0x6000 + 0x100 * arg1 + imm8`

  Dodaje wartość `imm8` do `arg1`. Ustawia znacznik `Z` zgodnie z wynikiem operacji. Nie modyfikuje znacznika `C`.

- `CMPI arg1, imm8`

  kod `0x6800 + 0x100 * arg1 + imm8`

  Odejmuje wartość `imm8` od `arg1`, ale nie zapisuje wyniku. Ustawia znaczniki `C` i `Z` zgodnie z wynikiem operacji.

- `RCR arg1`

  kod `0x7001 + 0x100 * arg1`

  Rotuje zawartość `arg` o jeden bit w prawo poprzez znacznik `C`. Nie modyfikuje znacznika `Z`.

- `CLC`

  kod `0x8000`

  Zeruje znacznik `C`. Nie modyfikuje znacznika `Z`.

- `STC`

  kod `0x8100`

  Ustawia znacznik `C`. Nie modyfikuje znacznika `Z`.

- `JMP imm8`

  kod `0xC000 + imm8`

  Wykonuje skok bezwarunkowy względny. Nie modyfikuje znaczników.

- `JNC imm8`

  kod `0xC200 + imm8`

  Wykonuje skok względny, gdy znacznik `C` nie jest ustawiony. Nie modyfikuje znaczników.

- `JC imm8`

  kod `0xC300 + imm8`

  Wykonuje skok względny, gdy znacznik `C` jest ustawiony. Nie modyfikuje znaczników.

- `JNZ imm8`

  kod `0xC400 + imm8`

  Wykonuje skok względny, gdy znacznik `Z` nie jest ustawiony. Nie modyfikuje znaczników.

- `JZ imm8`

  kod `0xC500 + imm8`

  Wykonuje skok względny, gdy znacznik `Z` jest ustawiony. Nie modyfikuje znaczników.

- `BRK`

  kod `0xFFFF`

  Pułapka przerywająca wykonywanie programu procesora SO. Nie modyfikuje rejestrów ani znaczników stanu.

# Instrukcje procesora wielordzeniowego

Procesor wielordzeniowy obsługuje wszystkie instrukcje procesora jednordzeniowego i dodatkowo następującą instrukcję:

- `XCHG arg1, arg2`

  kod `0x0008 + 0x100 * arg1 + 0x0800 * arg2`

  Zamienia miejscami wartości `arg1` i `arg2`. Nie modyfikuje znaczników. Jeśli `arg1` wskazuje na pamięć, a `arg2` jest rejestrem, to instrukcja jest atomowa. Jeśli `arg2` wskazuje na pamięć, to instrukcja nie jest atomowa.

# Dodatkowe ustalenia

Zachowanie procesora dla nieprawidłowych kodów instrukcji jest niezdefiniowane, ale najlepiej będzie ignorować takie kody. Wolno założyć, że funkcje są zawsze wywoływane z poprawnymi parametrami. Dla zachowania kompatybilności rozwiązanie jednordzeniowe po prostu ignoruje parametr `core` i stałą `CORES`. Emulator po napotkaniu instrukcji `BRK` wykonuje ją i kończy działanie funkcji `so_emul`. W procesorze wielordzeniowym instrukcja `BRK` kończy tylko wykonywanie funkcji `so_emul` dla tego rdzenia, który napotkał tę instrukcję.

# Oddawanie rozwiązania

Jako rozwiązanie należy wstawić w Moodle plik o nazwie `so_emulator.asm`.

# Kompilowanie rozwiązania

Rozwiązanie będzie kompilowane poleceniem

```
nasm -DCORES=$N -f elf64 -w+all -w+error -o so_emulator.o so_emulator.asm
```

gdzie `$N` jest wartością stałej `CORES`.

# Przykłady użycia

Przykładowe testy znajdują się w załączonym pliku `so_emulator_example.c`. Całość kompiluje się poleceniami:

```
gcc -DCORES=4 -c -Wall -Wextra -std=c17 -O2 -o so_emulator_example.o so_emulator_example.c
nasm -DCORES=4 -f elf64 -w+all -w+error -o so_emulator.o so_emulator.asm
gcc -pthread -o so_emulator_example so_emulator_example.o so_emulator.o
```

Przykładowe uruchomienie testów:

```
./so_emulator
./so_emulator 61 18
./so_emulator 10240
```

Plik `so_emulator_example.out` zawiera wyniki wypisywane na terminal przez powyższe polecenia.

# Ocenianie

Oceniane będą poprawność i szybkość działania programu, zajętość pamięci (rozmiary poszczególnych sekcji), styl kodowania, komentarze. Wystawienie oceny może też być uzależnione od osobistego wyjaśnienia szczegółów działania programu prowadzącemu zajęcia.