Kod pośredni i jego interpreter

Opis procesora wirtualnego

Procesor wirtualny posiada 3 rejestry:

- Licznik programu PC
- Wskaźnik stosu SP, wskazujący na ostatni zajęty bajt stosu, stos rośnie w kierunku malejących adresów
- Wskaźnik ramki BP

Procesor wirtualny ma architekturę typu pamięć-pamięć, tj. wszystkie instrukcje operują bezpośrednio na operandach znajdujących się w pamięci. Przestrzeń danych jest rozdzielna z przestrzenią programu.

Pojedynczy rozkaz ma następującą postać:

etykieta: mnemonik.s arg1, arg2, arg3; komentarz

Etykieta jest opcjonalna. Typ argumentów s oraz argumenty arg1, arg2 i arg3 nie występują we wszystkich instrukcjach. Tekst od znaku średnika do końca linii jest ignorowany jako komentarz.

Występują trzy tryby adresowania:

- Adresowanie natychmiastowe, oznaczane przez # jako pierwszy znak adresu
- Adresowanie bezpośrednie, charakteryzujące się brakiem specjalnego znaku przed wyrażeniem adresowym
- Adresowanie pośrednie, oznaczane przez * jako pierwszy znak adresu

Wyrażenie adresowe może mieć następującą postać

- etykieta
- liczba_całkowita
- liczba_rzeczywista (jedynie w trybie natychmiastowym)
- bp + liczba_całkowita
- bp - liczba_całkowita
- bp

Znacznik typu argumentów s może przybrać jedną z dwóch wartości:

- i dla typu całkowitego
- r dla typu rzeczywistego

W przypadku skoków warunkowych ostatni argument (adres) jest zawsze typu całkowitego.
Etykieta jest dowolnym ciągiem liter i cyfr zaczynającym się od litery.

Przykładowo, instrukcja

add.r #3.1415926, bp+18, *bp-32

dodaje wartość stałej 3.1415926 do liczby rzeczywistej zawartej pod adresem BP+18 i umieszcza wynik pod adresem odczytanym z komórki pamięci, której adres jest zawarty pod adresem bp-32.

Rozpoznawane są następujące instrukcje:

mov.s arg1, arg2

Kopiuje daną typu s z arg1 do arg2

add.s arg1, arg2, arg3

Dodaje arg1 do arg2 i umieszcza wynik w arg3

call arg1

Umieszcza na stosie adres następnej instrukcji i przypisuje PC wartość arg1

enter arg1

Ustawia wartość wskaźnika ramki BP i rezerwuje pamięć na stosie na zmienne lokalne. Odpowiada następującej sekwencji operacji:

push BP
BP:=SP
SP:=SP-arg1

leave

Odtwarza stan stosu sprzed wykonania instrukcji enter. Odpowiada sekwencji operacji:
SP := BP
pop BP

return

Podnosi ze stosu liczbę całkowitą i umieszcza ją w liczniku programu PC

write.s arg1

Wypisuje na ekran wartość arg1

push.s arg1

Odkłada na stosie wartość arg1

inscp arg1

Zwiększa wskaźnik stosu SP o arg1

jump arg1

Umieszcza arg1 w liczniku programu PC

je.s arg1, arg2, arg3

Porównuje arg1 z arg2, jeżeli liczby te są równe, zapisuje w liczniku rozkazów PC wartość arg3, w przeciwnym wypadku zwiększa PC o 1

jge.s arg1, arg2, arg3

Porównuje arg1 z arg2 jako liczby ze znakiem, jeżeli arg1 jest większy lub równy arg2, zapisuje w liczniku rozkazów PC wartość arg3, w przeciwnym wypadku zwiększa PC o 1

exit

Kończy wykonywanie programu

Struktura programu

Program składa się z dwóch zasadniczych części: asemblera i emulatora procesora.
Asembler przetwarza plik o nazwie podanej jako argument programu. Asemblacja odbywa się w dwóch przebiegach, w pierwszym obliczane są adresy poszczególnych etykiet i umieszczane w tablicy symboli **symtab**, w drugim instrukcje umieszczane są w wektorze **instructions**. Numer przebiegu (0 lub 1) zawarty jest w zmiennej globalnej **pass**. Za przetwarzanie poszczególnych linii programu odpowiedzialna jest funkcja **analyze**. Funkcja ta przyjmuje jako argument wstępnie przetworzoną linię programu, z której usunięty został komentarz i której wszystkie wielkie litery zostały zamienione na małe. W analizie programu wykorzystana jest tablica asocjacyjna **opcode_table**, zawierająca nazwy i kody poszczególnych instrukcji, wektor **default_argtype_table**, określający domyślny typ argumentów dla danej instrukcji oraz wektor **syntax_table**, zawierający adresy procedur sprawdzających liczbę i typy argumentów dla poszczególnych instrukcji. Powyższe struktury danych wypełniane są przez funkcję **setup_opcodes**, wywoływaną na początku funkcji **main**, na podstawie zawartości tablicy **opcodes**. Funkcja **extract_label** jest odpowiedzialna za analizę etykiet, **extract_instr** za analizę mnemoników instrukcji, a **extract_address** za analizę wyrażenia określającego pojedynczy argument. Powyższe funkcje, wraz z kilkoma funkcjami pomocniczymi, są zawarte w pliku analyzer.cpp. Błędy składniowe są sygnalizowane przez zgłoszenie wyjątku **syntax_error**, którego konstruktor przyjmuje jako parametr łańcuch tekstowy opisujący błąd bardziej szczegółowo.

Emulator procesora jest zawarty w pliku machine.cpp. Funkcja **execute** odczytuje kolejne instrukcje z tablicy **instructions** i wywołuje funkcje odpowiedzialne za ich emulację za pośrednictwem tablicy **dispatch_table**, wypełnianej w funkcji **setup_opcodes**. Wykonanie funkcji kończy się po napotkaniu instrukcji **exit**. Nazwy wszystkich funkcji odpowiadających za emulację poszczególnych instrukcji zaczynają się od przedrostka **h_**. Funkcje te intensywnie korzystają z funkcji pomocniczych **get_int_operand** oraz **get_real_operand** zwracających wartości źródłowych argumentów instrukcji oraz **get_int_operand_ref** i **get_real_operand_ref**, zwracających referencję do argumentu docelowego. W przypadku próby dostępu do nieistniejącej komórki pamięci lub wykonania niepoprawnej instrukcji zgłaszany jest wyjątek **segmentation_fault**. Plik zawiera również definicje szeregu funkcji o przedrostku **s_**, odpowiedzialnych za kontrolę liczby i typów argumentów instrukcji podczas asemblacji. Adres tablicy emulującej pamięć danych jest zawarty w zmiennej globalnej **memory**, rozmiar pamięci danych w zmiennej globalnej **memorysize**. Przed rozpoczęciem emulacji wartość licznika rozkazów PC jest ustawiana na 0, a wskaźnika stosu SP na **memorysize**.
Plik vm.h zawiera m.in. deklaracje różnych typów. Kod instrukcji jest określony przez typ wyliczeniowy **opcode**, wartość OC_FINAL musi zawsze występować na końcu listy i określa liczbę instrukcji. Typ wyliczeniowy **argtype** określa typ argumentu operacji, **addrmode** określa natychmiastowy, bezpośredni lub pośredni tryb adresowania, **basereg** określa, czy do adresowania jest użyty rejestr ramki stosu BP. Typy INT, UINT oraz REAL określają rozmiary danych użytych w emulatorze.