Cele dydaktyczne

Po przestudiowaniu tego rozdziału powinieneś być w stanie:

- Przedstawić przegląd podstawowych właściwości **rozkazów maszynowych**.
- Opisać typy argumentów używanych w typowych listach rozkazów maszynowych.
- Przedstawić przegląd typów danych w architekturze x86 i ARM.
- Opisać typy argumentów obsługiwanych przez typowe listy rozkazów maszynowych.
- Przedstawić przegląd typów operacji w architekturze x86 i ARM.
- Zrozumieć różnice pomiędzy **grubokońcowością** (ang. big endian), **cienkokońcowością** (ang. little endian) i **dwukońcowością** (ang. bi-endian).

Licznych komponentów opisanych w tej książce nie widzi użytkownik lub programista komputera. Jeśli programista używa języka wysokiego poziomu, takiego jak Pascal lub Ada, to nie jest w stanie zauważyć zbyt wielu szczegółów architektury wykorzystywanego komputera.

Jednym z obszarów, w których zarówno projektant komputera, jak i jego programista mają wspólny obraz maszyny, jest lista rozkazów maszynowych. Z punktu widzenia projektanta lista rozkazów maszynowych określa wymagania funkcjonalne w stosunku do procesora: implementacja procesora jest zadaniem, które w znacznej mierze polega na wdrożeniu listy rozkazów maszynowych. Jedynie użytkownik, który wybiera programowanie w języku maszynowym (właściwie w języku asemblera, patrz rozdział 15), jest świadomy struktury rejestrów i pamięci, rodzaju danych bezpośrednio obsługiwanych przez maszynę oraz funkcjonowanie ALU.

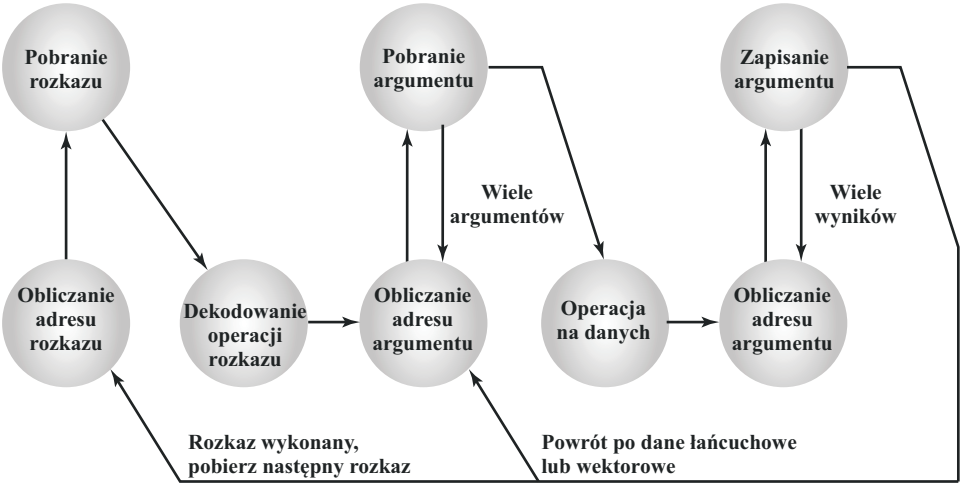
Opis listy rozkazów maszynowych komputera oznacza duży krok naprzód w wyjaśnianiu działania procesora. W związku z tym koncentrujemy się na instrukcjach maszynowych w tym i następnym rozdziale.

13.1. WŁASNOŚCI ROZKAZÓW MASZYNOWYCH

Działanie procesora określają wykonywane przez niego rozkazy, zwane *rozkazami maszynowymi* lub *rozkazami komputerowymi*. Zbiór różnych rozkazów, które procesor może wykonać, jest określany jako *lista rozkazów* procesora.

Elementy rozkazu maszynowego

Każdy rozkaz musi zawierać informacje wymagane przez procesor do jego wykonania. Rysunek 13.1, który jest powtórzeniem rysunku 3.6 z tomu I, przedstawia etapy związane z wykonaniem rozkazów. W rezultacie etapy te definiują elementy rozkazu maszynowego. Są to następujące elementy:



Rysunek 13.1. Schemat cyklu rozkazu

- **Kod operacji:** Określa operację do wykonania (np. ADD, we-wy). Operacja jest określona przez kod binarny, znany jako kod operacji lub **opcode**.
- **Odniesienie do argumentów źródłowych:** Operacja może obejmować jeden lub więcej argumentów źródłowych. Są to argumenty, które są danymi wejściowymi operacji.
- **Odniesienie do wyniku:** Operacja może prowadzić do powstania wyniku.
- **Odniesienie do następnego rozkazu:** Precyzuje procesorowi, skąd ma pobrać następny rozkaz po zakończeniu wykonywania bieżącego rozkazu.

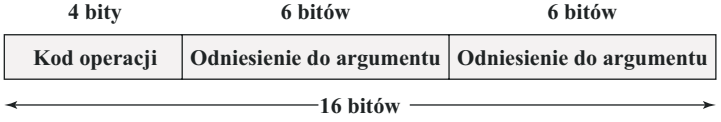
Adres kolejnego rozkazu do pobrania może być adresem rzeczywistym lub adresem wirtualnym, w zależności od architektury systemu. Ogólnie rzecz biorąc, rozróżnienie jest przejrzyste dla architektury listy rozkazów. W większości przypadków następny rozkaz przewidziany do pobrania następuje bezpośrednio po bieżącym rozkazie. W takich przypadkach nie istnieje jawne odniesienie do kolejnego rozkazu. Gdy odwołanie takie jest wymagane, musi być dostarczony adres w pamięci głównej lub w pamięci wirtualnej. Formę, w której jest dostarczany adres, omówimy w rozdziale 14.

Argumenty źródłowe i wyniki mogą się znajdować w jednym z czterech obszarów:

- **Pamięć główna lub wirtualna:** Podobnie jak w przypadku odwołań do następnego rozkazu, konieczne jest dostarczenie adresu pamięci głównej lub wirtualnej.
- **Rejestr procesora:** Z rzadkimi wyjątkami procesor zawiera jeden lub więcej rejestrów, do których mogą się odwoływać rozkazy maszynowe. Jeśli istnieje tylko jeden rejestr, to odwołanie do niego może być domyślne. Jeżeli istnieje więcej niż jeden rejestr, to każdemu przypisywana jest unikatowa nazwa lub numer, a rozkaz musi zawierać numer żądanego rejestru.
- **Natychmiastowy:** Wartość argumentu jest przechowywana w polu wykonywanego rozkazu.
- **Urządzenie we-wy:** Rozkaz musi określać moduł we-wy i urządzenie używane podczas operacji. Jeśli używane jest we-wy odwzorowane w pamięci, jest to po prostu kolejny adres w pamięci głównej lub wirtualnej.

Reprezentacja rozkazu

W systemach komputerowych każdy rozkaz jest reprezentowany za pomocą ciągu bitów. Rozkaz jest dzielony na pola odpowiadające elementom składowym rozkazu. Prosty przykład formatu rozkazu pokazano na rysunku 13.2. Innym przykładem jest format rozkazu IAS z rysunku 1.7 (tom I). Na większości list rozkazów używany jest więcej niż jeden format. Podczas wykonywania rozkazu jest on wczytywany do rejestru rozkazów (IR) w procesorze. Procesor musi być w stanie wyodrębnić dane z różnych pól rozkazu tak, by wykonać wymaganą operację.



Rysunek 13.2. Prosty format rozkazu

Zarówno programiście, jak i czytelnikowi trudno jest posługiwać się binarną reprezentacją rozkazów maszynowych. Z tego powodu powszechną praktyką stało się używanie *symbolicznej reprezentacji* rozkazów maszynowych. Przykładem tego jest lista rozkazów IAS z tabeli 1.1 (tom I).

Kody operacji są reprezentowane przez skróty, zwane *mnemonikami*, które określają operację. Typowe przykłady to:

ADD	Dodaj
SUB	Odejmij
MUL	Pomnóż
DIV	Podziel
LOAD	Ładuj dane z pamięci
STOR	Zapisz dane w pamięci

Argumenty są również reprezentowane symbolicznie. Na przykład rozkaz

ADD R, Y

może oznaczać: dodaj wartość zawartą w pozycji danych Y do zawartości rejestru R. W tym przykładzie Y odnosi się do adresu komórki pamięci, natomiast R określa dany rejestr. Zauważmy, że operacja jest przeprowadzana na zawartościach tych lokacji, a nie na adresach.

Jest więc możliwe napisanie programu w języku maszynowym w postaci symbolicznej. Każdy symboliczny kod operacji ma ustaloną reprezentację binarną, a programista określa lokalizację każdego symbolicznego argumentu. Programista może na przykład rozpocząć od listy definicji:

X = 513
Y = 514

i tak dalej. Prosty program zaakceptowałby te symboliczne dane wejściowe, przekonwertował kody operacji i odwołania do argumentów do postaci binarnej i zbudował binarne instrukcje maszynowe.