9.2 Lista rozkazów i sposoby adresowania

1. Formaty rozkazów

Istnieje kilka typowych *formatów instrukcji* (por. rys. 1). Każdy z formatów ma część operacyjną (ang. *operational code*) precyzującą, do jakiej kategorii rozkaz należy i ewentualnie dalsze części zawierające dane lub adresy. W maszynie von Neumanna zakładaliśmy, że instrukcja reprezentowana jest przez słowo binarne o stałej długości *n*. W praktyce nie jest to wygodne i na ogół jeden procesor ma kilka różnych formatów instrukcji.

a)

Część operacyjna

b)

Część operacyjna	*Dane*

c)

Część operacyjna	*Adres*

d)

Część operacyjna	*Adres 1*	*Adres 2*

Rys.1. Kilka podstawowych formatów instrukcji; w polu *Adres* znajduje się adres komórki pamięci, gdzie umieszczony jest argument, lub adres komórki, gdzie należy przesłać wynik działania instrukcji; W polu *Dane* mamy argument instrukcji

Poznając konkretny procesor czy mikroprocesor dowiadujemy się jakie ma on formaty instrukcji i na jakich podstawowych *formatach danych* operuje (formaty danych poznaliśmy w rozdziale 1 „Kody i kodowanie w systemach cyfrowych").

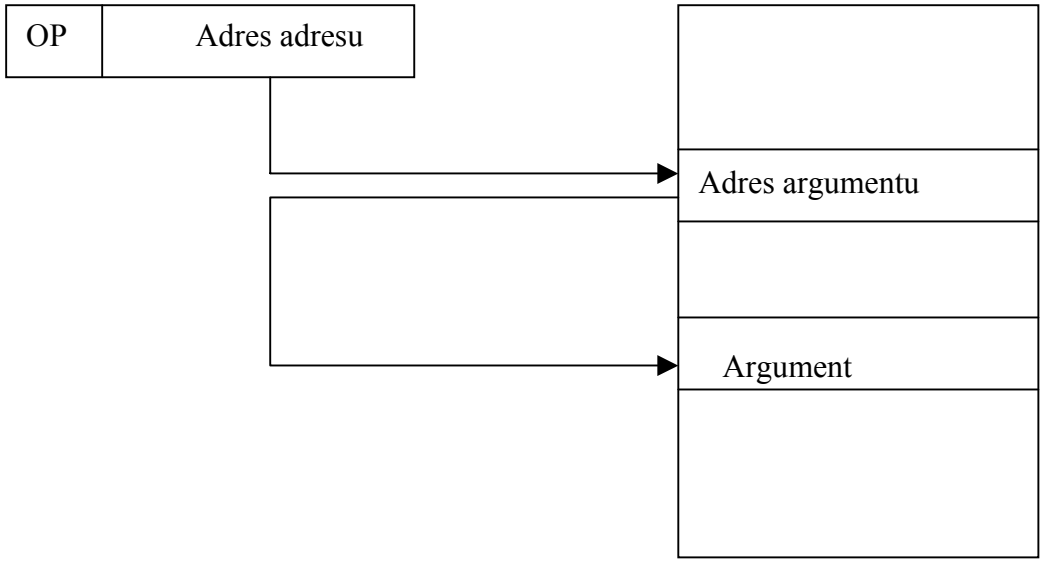
2. Sposoby adresowania

Sposoby adresowania to inaczej *tryby adresowania* (ang. *addressing modes*). Tryb adresowania jest sposobem czyli algorytmem określenia tzw. *adresu efektywnego* EA na podstawie zawartości pola adresowego A w rozkazie oraz zawartości pewnych komórek pamięci lub rejestrów procesora. Adres efektywny zamieniany jest na tzw. *adres fizyczny*, tzn. adres pojawiający się na tzw. *szynie adresowej* procesora, ale w sposób w zasadzie ukryty dla programisty.

Po kolei omówimy najważniejsze tryby adresowania. Będziemy przy tym używać następującej notacji: **`EA`** to będzie tzw. adres efektywny, a **`A`** pole adresowe rozkazu.

- *Adresowanie natychmiastowe* (ang. *immediate addressing*) jest to sposób adresowania, w którym argument dla instrukcji jest bezpośrednio podany w tej instrukcji.
- *Adresowanie bezpośrednie* (ang. *direct addressing*) to sposób adresowania w pewnym sensie najnaturalniejszy. Adres argumentu lub wyniku jest podany w części adresowej rozkazu.

- *Adresowanie rejestrowe* (ang. *register addressing*) jest w gruncie rzeczy adresowaniem bezpośrednim, z tym że pole adresu (zwykle bardzo krótkie) adresuje tylko adresy wewnętrzne procesora.
- *Adresowanie pośrednie* (ang. *indirect addressing*) to podawanie adresu komórki pamięci, w której znajduje się adres argumentu lub wyniku działania instrukcji. Jeśli pośrednich odwołań jest więcej, to taki sposób adresowania nazywamy adresowaniem pośrednim wielopoziomowym.



Rys. 2. Adresowanie pośrednie

- *Adresowanie względne* polega na tym, że adres efektywny EA otrzymujemy przez dodanie wartości pola adresu A do bieżącej zawartości licznika rozkazów. Zwykle przy tym sposobie adresowania pole adresowe jest traktowane jako liczba w kodzie uzupełnień do dwóch.
- *Adresowanie pośrednie rejestrowe* (ang. *indirect addressing*) jest to w gruncie rzeczy adresowanie pośrednie ale adres argumentu lub wyniku znajduje się w rejestrze.
- *Adresowanie indeksowe* (ang. *indexed addressing*) polega na dodaniu adresu podanego w rozkazie do tzw. rejestru indeksowego. Pole adresowe A rozkazu zawiera przy tym sposobie adresowania więcej bitów niż w przypadku adresowania bazowego.
- *Adresowanie bazowe* (ang. *base register addressing*) polega na dodaniu do adresu podanego w rozkazie do tzw. rejestru bazowego.
- *Adresowanie bazowo-indeksowe* polega na dodaniu do siebie zawartości rejestru indeksowego, rejestru bazowego i wartości pola adresowego rozkazu.
- *Adresowanie stosowe***.** Praktycznie w każdym procesorze organizuje się bardzo użyteczną strukturę danych jaką jest stos. Do stosu odwołujemy się za pomocą specjalnego rejestru tzw. wskaźnika stosu oznaczanego z reguły symbolem SP (od ang. *stack pointer*). Możemy przesłać np. słowo na stos znajdzie się ono wtedy na wierzchołku stosu. Ze stosu możemy również odczytać słowo ale tylko znajdujące się na wierzchołku stosu. Znika ono wtedy ze stosu a wskaźnik stosu SP jest odpowiednio modyfikowany.

Dobrym wyobrażeniem stosu jest stosik kartek, z którego kartkę możemy zdejmować tylko od góry i kłaść kartki w naturalny sposób na wierzch stosu kartek.

Uwaga: Adresowanie względne, adresowanie bazowe, adresowanie indeksowe i indeksowo-bazowe nazywa się *adresowaniem z przesunięciem* (ang. *displacement addressing*). Ogólny schemat obliczania adresu efektywnego jest tu taki

$$EA=A+(R)$$

lub

$$EA=A+(R1)+(R2)$$

gdzie R, R1, R2 są rejestrami procesora.

Czasami używamy określenia *adresowanie niejawne*. Określenie to oznacza, że dany rozkaz odwołuje się do pewnych rejestrów czy komórek pamięci w sposób określony w opisie rozkazu, nie korzystając w tym trybie adresowania z pola adresowego A

3. Lista rozkazów

Lista rozkazów to inaczej *lista instrukcji* (ang *instruction set*). Typowa lista instrukcji współczesnego mikroprocesora zawiera 100 do 200 instrukcji.

Lista instrukcji jest podstawą do podzielenia mikroprocesorów na mikroprocesory typu *RISC* (ang. *Reduced Instruction Set Computer*) i typu *CISC* (ang. *Complex Instruction Set Computer*). Mikroprocesory typu RISC mają relatywnie krótką listę instrukcji i instrukcje są proste. Mikroprocesory typu CISC mają bardzo rozbudowaną listę instrukcji.

Rozkazy można podzielić na kilka kategorii. Przykłady podane w ramach danej kategorii dotyczą asemblera dla rodziny mikroprocesorów Intel x86.

- *Rozkazy przesłań* (rozkazy transmisji danych)

  Przykłady:

```
MOV AX,BX ;prześlij zawartość rejestru BX do rejestru akumulatora (rejestru AX)
          ; również instrukcje operujące na stosie są instrukcjami przesłań
PUSH AX   ; prześlij zawartość rejestru AX na szczyt stosu
POP AX    ; prześlij 16-bitowe słowo znajdujące się na szczycie stosu
      ; do rejestru AX
```

- *Rozkazy arytmetyczne* (dodawanie ADD, odejmowanie SUB, mnożenie MUL, dzielenie DIV)

  Przykład:

```
ADD AX,BX ; dodaj zawartość rejestru AX do zawartości rejestru BX
          ; i wynik umieść w rejestrze AX
```

- *Rozkazy logiczne* (NOT, OR, AND, XOR)

  Przykład:

```
XOR AL,AL ; instrukcja logiczna zerująca rejestr AL
```

- *Przesunięcia i rotacje*

  Przykład:

```
RCL   AX,5 ; przesuń w lewo cyklicznie akumulator o 5 pozycji w lewo
```

- *Skoki i wywołania podprogramów*

  Przykłady:

```
INT 21      ;  wywołanie programu obsługi przerwania 21
JMP et      ; skocz do etykiety et
```

- *Instrukcje we/wy* (IN, OUT)

Przykład.

```
IN AL, 5   ; wprowadzenie danej z portu o numerze 5 do rejestru AL.
```

- *Instrukcje wspomagające system operacyjny*