

OAK - ZAGADNIENIA EGZAMINACYJNE: (10 zadań na egzaminie; 50 pkt max.) 09.06.2011
Pamiętać o kalkulatorze!

[punkty bonusowe obowiązują OAK
tylko na piątkowy terminie;
min. 25 pkt do rozliczenia]

Najaz zagadnień (najbardziej prawdopodobne)

* Podstawowe pojęcia: CISC i RISC

* Wydajność komputerów: Definicje CPI, wykorzystanie CPI, współczynnik speedup (obliczanie z użyciem CPI)

* Prawo Amdahla!

* Artymatyka komputerów: konwersja binarów, systemy little-endian i big-endian.
- zapis binarów FP w formatach IEEE 754. (NIE BĘDZIE NA EGZAMINIE!)

* Budowa procesora: rejestr wskazników i jego zastosowanie; stos i jego funkcje.

* Pamięć cache:

- metody odwzorowania i porównanie ich efektywności
- obliczenie tajennej pojemności pamięci cache
- obliczenie miss penalty i wydajności cache (AMAT)

$$\text{miss rate} + \text{hit rate} = 1$$

* Potok i architektury superskalarme:

- obliczenie latency i throughput o różnych konfiguracjach.
- hazardy i ich rodzaje

$$\begin{array}{|c|}\hline 1s = 10^{12} ps \\ \hline 1G = 10^9 \\ \hline \end{array}$$

* Architektura IA-32

- segmentacja i jej warianty (IA32 oraz x86 - tryb rzeczywisty)

* Architektura IA-64

- konsepcja EPIC (VLIW), format instrukcji. (pytanie typu quizowego)

* Architektura listy instrukcji:

- tryby adresowania (7)
- adresowanie w instrukcjach skoków

* Elementy Assemblera Pentium:

- definiowanie danych i ich inicjalizacja
- tryby adresowania argumentów

- podstawowe rodzaje instrukcji: przekier., operacje arytmetyczne i logiczne, skoki.
- analiza wykonyania instrukcji (dane jest zawartość rejestrów i komórek pamięci)
przed wykonyaniem instrukcji, podać te zawartości po wykonyaniu instrukcji)

Organizacja i Architektura Komputerów (OAK)

Podsumowanie materiału egzaminacyjnego:

Zagadnienia egzaminacyjne:

Bit, bajt, słowa

- Bity są grupowane w większe zespoły o długości będącej potągią liczby 2:
 - tetrada: 4 bity (*nibble*)
 - bajt: 8 bitów
 - słowo: 16, 32, 64 lub 128 bitów
 - podwójne słowo (*doubleword*)
 - półsłowo (*halfword*)
- Architektura RISC zapewnia lepsze wykorzystanie możliwości przetwarzania potokowego niż architektura CISC. W architekturze RISC:
 - instrukcje mają taką samą długość
 - większość operacji dotyczy rejestrów
 - odwołania do pamięci, które mogą ewentualnie powodować kolizje są rzadsze i występują tylko w przypadku specjalnie do tego celu używanych instrukcji *load* i *store*
- W architekturze CISC instrukcje mają różną długość, występuje bogactwo sposobów adresowania danych w pamięci

1s = 1000 ms (milisekund)

1 ms = 1000 μs (mikrosekund)

1 μs = 1000 ns (nanosekund)

1 ns = 1000 ps (pikosekund)

- **Częstotliwość zegara CPU** (1 Hz = 1 cykl / s):

$$\text{Czas cyklu} = \frac{1}{\text{Częstotliwość zegara}}$$

CPI – średnia liczba cykli procesora potrzebna do wykonania instrukcji.
 Jeżeli dwa komputery mają taką samą architekturę ISA, to będą miały identyczne:
CPI oraz liczbę instrukcji (IC)

- **CPI:**

$$CPI = \frac{\sum_{i=1}^n CPI_i \times IC_i}{IC} = \sum_{i=1}^n CPI_i \times \left(\frac{IC_i}{IC} \right) = \sum_{i=1}^n CPI_i \times F_i$$

gdzie:

- CPI_i – liczba cykli potrzebnych do wykonania instrukcji typu i
- IC_i – liczba instrukcji typu i w programie
- IC – ogólna liczba instrukcji w programie
- F_i - względna częstość występowania instrukcji typu i w programie

[Czyli suma iloczynów: CPI operacji x częstość występowania operacji]

- **Speedup enhanced:**

$$Speedup_{enh} = \frac{CPI_{org}}{CPI_{new}}$$

- **Fraction enhanced:**

$$Frac_{enh} = \frac{(CPI \text{ instrukcji} \cdot \text{częstość występowania instrukcji})}{CPI_{org}}$$

- **Prawo Amdahla (Speedup overall):**

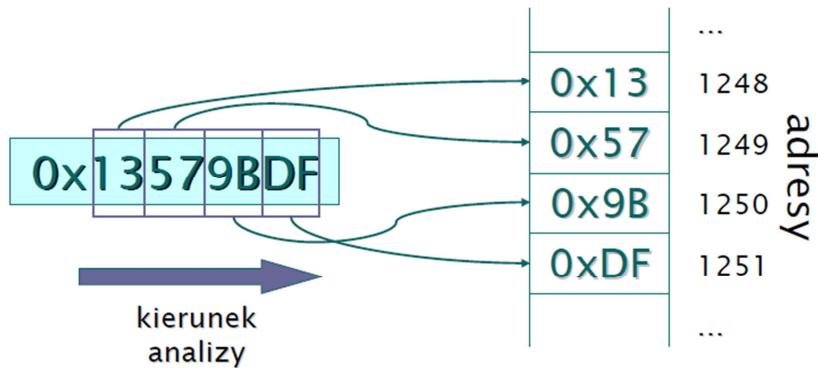
$$Speedup_{overall} = \frac{CPUtime_{old}}{CPUtime_{new}} = \frac{1}{(1 - Frac_{enh}) + \frac{Frac_{enh}}{Speedup_{enh}}}$$

Używane kody:

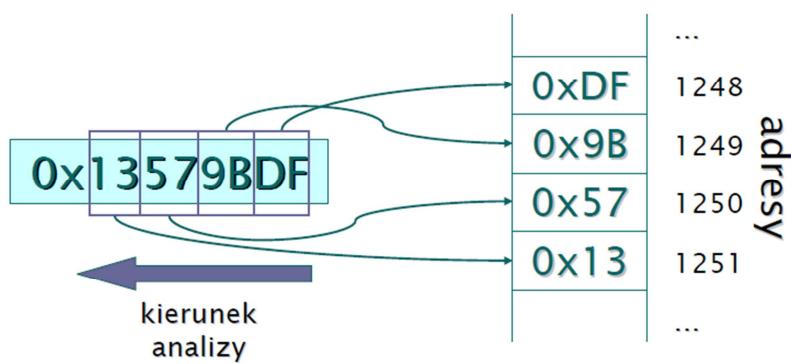
- Dziesiętny (RADIX)
- Szesnastkowy (HEX)
- Naturalny kod binarny (NKB)

- BCD [każda cyfra dziesiętna zapisana binarnie na 4 bitach]
- U2 [gdy liczba w NKB dodatnia, wtedy identyczna; w przeciwnym wypadku, w liczbie w NKB zamienić 1 ->0 i 0->1 i dodać 1]

Big endian:



Little endian:



Rejestry

- Niewielka pamięć robocza CPU do przechowywania tymczasowych wyników obliczeń
- Liczba rejestrów i ich funkcje różnią się dla różnych procesorów
- Rejestry mogą pełnić rozmaite funkcje:
 - Rejestry ogólnego przeznaczenia (GP – *general purpose*)
 - Rejestry danych (np. akumulator)
 - Rejestry adresowe
 - Rejestr wskaźników (stanu, warunków, flag)

Rejestr wskaźników i jego zastosowania:

Wskaźniki (inaczej flagi) są przechowywane jako zero-jedynkowe zmienne bitowe. Służą one przy operacjach arytmetycznych do:

- przeniesienia – gdy wynik przekroczył maksymalną wartość możliwą do przechowania w danym typie danych,
- przeniesienia pomocniczego – przeniesienie na starszą tetradę, lub na młodszą (przy odejmowaniu).
- nadmiaru – jest ustawiany w kodzie U2, gdyby nastąpiło przeniesienie na bit znaku (ten najstarszy) to bit zero ustawiany jest gdy otrzymano zero, a bit znaku, gdy liczę ujemną.

Stos – zastosowania:

- przy wywołaniu podprogramu zapamiętuje adres powrotu do programu głównego
- przy przejściu do programu obsługi przerwania zapamiętuje adres powrotu do przerwanego programu
- służy do chwilowego przechowywania zawartości rejestrów w celu uwolnienia ich do innych zadań
- może być użyty do przekazywania parametrów do podprogramów

Pamięć cache:

Terminologia:

cache hit – trafienie, czyli informacja znajduje się w pamięci cache,

cache miss – chybienie, czyli informacja nie znajduje się w pamięci cache,

hit rate – współczynnik trafień, stosunek liczby trafień do liczby wszystkich odwołań,

miss rate – współczynnik chybień: $1 - \text{hit rate}$,

hit time – czas dostępu do żądanej informacji w sytuacji, gdy wystąpiło trafienie,

miss penalty – kara za chybienie, czyli czas potrzebny na wymianę wybranej linijki.

$$\text{miss rate} + \text{hit rate} = 1$$

Poprawa wydajności pamięci cache polega na skróceniu czasu AMAT poprzez: zmniejszenie miss rate, zwiększenie hit rate, zmniejszenie miss penalty, zmniejszenie hit time.

Odwzorowanie bezpośrednie – każda linijka pamięci głównej jest przyporządkowana tylko jednej linijce w pamięci cache. Bity znacznika: 1

Odwzorowanie skojarzeniowe – zezwalamy na lokalizację linijk w dowolnym miejscu pamięci cache. Bity znacznika: cała wartość

Zadanie z pamięcią cache:

Adresy x bitowe -> 2^x B dla pamięci operacyjnej.

y KB pamięci cache -> $y = 2^{\log_2 y} \cdot 2^{10} B = 2^t$

z B dla linijki -> $z = 2^u B$

Ilość linijek w pamięci operacyjnej: $\frac{2^x}{2^u} = 2^{x-u}$

Ilość linijek w pamięci cache: $\frac{2^t}{2^u} = 2^{t-u}$

Dla odwzorowania bezpośredniego:

$\frac{2^{x-u}}{2^{t-u}} = 2^{x-t}$ ilość linijek przypadających na jedną linijkę pamięci cache.

Znacznik: x-t bitów.

$z \cdot 8 + (x-t) + 1 = \text{pojemność linijki brutto}$

Dla odwzorowania skojarzeniowego (w pełni asocjacyjnego):

Znacznik: x-u bitów.

$z \cdot 8 + (x-u) + 1 = \text{pojemność linijki brutto}$

Dla odwzorowania 4-drożnego: (2^2)

Liczba podzbiorów: $\frac{2^{t-u}}{2^2} = 2^{t-u-2}$ podzbiór w pamięci cache

$\frac{2^{x-u}}{2^{t-u-2}} = 2^{x-u-t+u+2} = 2^{x-t+2}$ ilość linijek przypadających na jeden podzbiór

Znacznik: x-t+2.

$z \cdot 8 + (x-t+2) + 1 = \text{pojemność linijki brutto}$

AMAT = hit rate \cdot hit time + miss rate \cdot miss penalty

w uproszczeniu: AMAT = hit time + miss rate \cdot miss penalty

AMAT dla wspólnego i rozdzielonego cache:

AMAT = %instr x (instr hit time + instr miss rate \cdot instr miss penalty) +
%data x (data hit time + data miss rate \cdot data miss penalty)

[im mniejsza wartość tym lepiej!]

Throughput =
$$\frac{1s}{\text{czas naj wolniejszego stopnia} + \text{ew. opóźnienie zegara}}$$

Lattency = czas pracy najwolniejszego stopnia · ilość stopni potoku

Zjawisko hazardu

- Równoległe przetwarzanie instrukcji w potoku prowadzi często do niekorzystnych zjawisk nazywanych hazardem. Hazard polega na braku możliwości wykonania instrukcji w przewidzianym dla niej cyklu. Wyróżnia się trzy rodzaje hazardu:
 1. Hazard zasobów (*structural hazard*) – kiedy dwie instrukcje odwołują się do tych samych zasobów
 2. Hazard danych (*data hazard*) – kiedy wykonanie instrukcji zależy od wyniku wcześniejszej, nie zakończonej jeszcze instrukcji znajdującej się w potoku
 3. Hazard sterowania (*control hazard*) – przy przetwarzaniu instrukcji skoków i innych instrukcji zmieniających stan licznika rozkazów (np. wywołania podprogramów)
- Hazard sterowania został omówiony już wcześniej – teraz zajmiemy się hazardem zasobów i danych

Rodzaje hazardów:

- Hazard RAW (*read after write*) występuje gdy pojawi się żądanie odczytu danych przed zakończeniem ich zapisu (po przekątnej w prawo)
- Przykład (taki sam jak na poprzednim slajdzie)

```
add ax, bx  
mov cx, ax
```

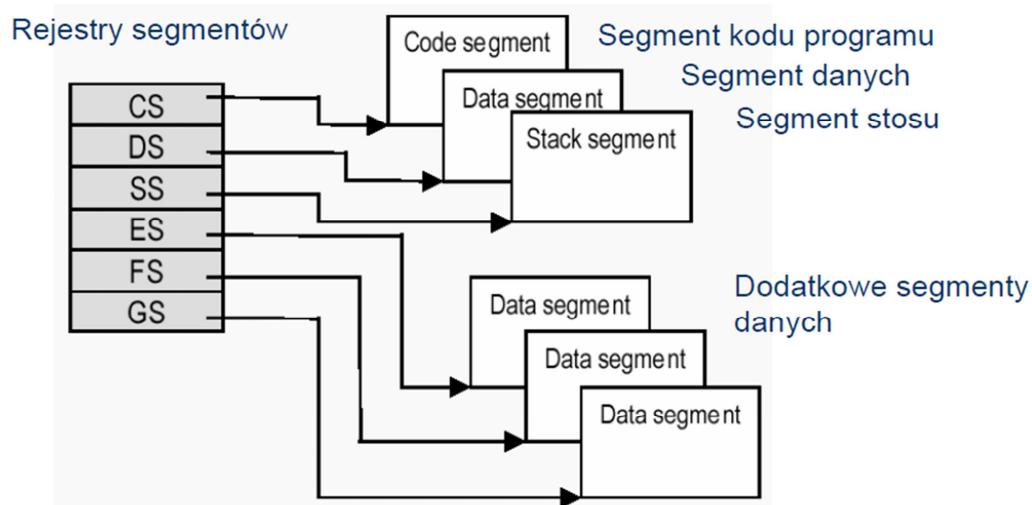
- Hazard WAR (*write after read*) występuje gdy pojawi się żądanie zapisu danych przed zakończeniem ich odczytu (po przekątnej w lewo)
- Przykład:

```
mov bx, ax  
add ax, cx
```

- Hazard WAW (*write after write*) występuje gdy pojawi się żądanie zapisu danych przed zakończeniem wcześniejszej operacji zapisu (jedno pod drugim po lewej stronie)
- Przykład:

```
mov ax, [mem]  
add ax, bx
```

Segmentacja pamięci



Tryby adresowania

Real-address mode:

Selector S = x Offset = y

Adres efektywny: $x \cdot 16 + y$

Segmented-address mode:

Selector S wskazuje deskryptor, gdzie dany jest adres bazowy x

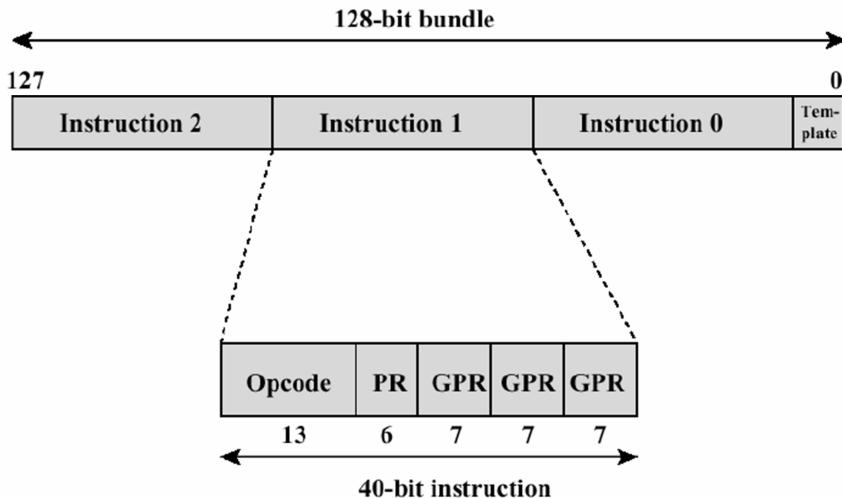
Offset = y

Adres efektywny: x+y

Koncepcja EPIC (Explicitly Parallel Instruction Computing):

Jest to wariant VLIW (Very Long Instruction Word) i polega na takiej komplikacji kodu, aby pewne jego części mogły być wykonywane w programie równolegle.

Format instrukcji IA-64



PR = Predicate register

GPR = General-purpose register (integer or floating-point)

Tryby adresowania

- Sposoby wskazywania na argumenty instrukcji
- Podstawowe tryby adresowania używane we współczesnych procesorach:
 - Register – rejestrowy, wewnętrzny
 - Immediate – natychmiastowy
 - Direct – bezpośredni
 - Register indirect – rejestrowy pośredni
 - Base + Index – bazowo-indeksowy
 - Register relative – względny
 - Stack addressing – adresowanie stosu

Adresowanie rejestrowe

- Argumenty operacji znajdują się w rejestrach procesora
- Przykład:

```
mov al,bl    ;prześlij bl do al
inc bx      ;zwiększ bx o 1
dec al      ;zmniejsz al o 1
sub dx,cx   ;odejmij cx od dx
```

Adresowanie natychmiastowe

- Argument operacji znajduje się w postaci jawniej w instrukcji, zaraz po kodzie operacji
- Przykłady:

```
mov bl,44 ;prześlij 44 (dziesiątne) do bl  
mov ah,44h ;prześlij 44 (hex) do ah  
mov di,2ab4h ;prześlij 2ab4 (hex) do di  
add ax,867 ;dodaj 867 (dec) do ax
```

Adresowanie bezpośrednie cd.

- Argument jest umieszczony w komórce pamięci; adres tej komórki jest podany w instrukcji
- Przykłady:

```
mov al,DANA ;prześlij bajt z komórki  
;pamięci o adresie DS:DANA  
;do rejestru al.  
mov ax,news ;prześlij ax do adresu DS:NEWS
```

- W procesorach x86 do adresowania rejestrowego pośredniego można wykorzystywać rejesty: **bx**, **bp**, **si**, **di**
- Należy pamiętać, że rejesty **bx**, **si**, **di** domyślnie współpracują z rejestrem segmentu **ds**, natomiast rejestr **bp** współpracuje domyślnie z rejestrem segmentu **ss**

Adresowanie pośrednie rejestrowe

- Przykłady:

mov cx, [bx]

- 16-bitowe słowo z komórki adresowanej przez ofset w rejestrze **bx** w segmencie danych (**ds**) jest kopiowane do rejestrów **cx**

mov [bp], dl

- bajt z **dl** jest kopiowany do segmentu stosu pod adres podany w rejestrze **bp**

mov [di], [bx]

- przesyłania typu pamięć-pamięć nie są dozwolone (z wyjątkiem szczególnych instrukcji operujących na łańcuchach znaków – stringach)

Adresowanie bazowo-indeksowe

- Podobne do adresowania rejestrowego pośredniego
- Różnica polega na tym, że offset jest obliczany jako suma zawartości dwóch rejestrów
- Przykład: `mov dx, [bx+di]`

Adresowanie rejestrowe względne

- Podobne do adresowania bazowo-indeksowego
- Offset jest obliczany jako suma zawartości rejestru bazowego lub indeksowego (`bp`, `bx`, `di` lub `si`) oraz liczby określającej bazę
- Przykład: `mov ax, [di+100h]`
powyższa instrukcja przesyła do `ax` słowo z komórki pamięci w segmencie `ds` o adresie równym sumie zawartości rejestru `di` i liczby `100h`

Adr. względne bazowo-indeksowe

- Adresowanie względne bazowo-indeksowe jest kombinacją dwóch trybów:
 - bazowo-indeksowego
 - rejestrowego względnego
- Przykłady:
`mov dh, [bx+di+20h]`
rejestr `dh` jest ładowany z segmentu danych, z komórki o adresie równym sumie `bx+di+20h`
`mov ax, file[bx+di]`
rejestr `ax` jest ładowany z segmentu danych, z komórki o adresie równym sumie adresu początkowego tablicy `file`, rejestru `bx` oraz rejestru `di`
`mov list[bp+di], cl`
rejestr `cl` jest zapamiętyany w segmencie stosu, w komórce o adresie równym sumie adresu początkowego tablicy `list`, rejestru `bp` oraz rejestru `di`

Adresowanie stosu

- Jest używane przy dostępie do stosu
- Przykłady

<code>popf</code>	;pobierz słowo wskaźników (flagi) ze stosu
<code>pushf</code>	;wyślij słowo wskaźników na stos
<code>push ax</code>	;wyślij <code>ax</code> na stos
<code>pop bx</code>	;odczytaj <code>bx</code> ze stosu
<code>push ds</code>	;wyślij rejestr segmentu <code>ds</code> na stos
<code>pop cs</code>	;odczytaj rejestr segmentu ze stosu
<code>push [bx]</code>	;zapisz komórkę pamięci o adresie podanym ;w <code>bx</code> na stos

Adresowanie w instrukcjach skoku

- Adres w instrukcjach skoków może mieć różną postać:
 - krótką (short)
 - bliską (near)
 - daleką (far)
- Adresy mogą być podane w różny sposób:
 - Wprost w części adresowej instrukcji – tryb bezpośredni (direct)
 - W rejestrze
 - W komórce pamięci adresowanej przez rejestr

Skoki krótkie (short)

- Adres jest jednobajtową liczbą w kodzie U2
- Skok polega na dodaniu tej liczby do rejestrów IP
- Krótki skok ma zakres od -128 do +127 bajtów w stosunku do aktualnej wartości IP i odbywa się w obrębie tego samego segmentu wskazanego przez `cs`

Skoki bliskie (near)

- Skok może być wykonany do dowolnej instrukcji w obrębie segmentu wskazanego przez aktualną wartość w rejestrze `cs`
- W trybie adresów rzeczywistych adres skoku jest 16-bitowy, natomiast segment ma rozmiar 64 kB
- W trybie segmentowym IA-32 adres jest 32-bitowy

Skoki dalekie (far)

- Skok może być wykonany do dowolnej instrukcji w obrębie dowolnego innego segmentu
- W rozkazie skoku należy podać nie tylko adres skoku tak jak w przypadku skoków bliskich, ale ponadto również nową wartość rejestru segmentu `cs`

Asembler Pentium:

Alokacja danych cd.

- W języku asemblera do alokacji danych używa się dyrektywy `define`
 - Rezerwuje miejsce w pamięci
 - Etykietuje miejsce w pamięci
 - Inicjalizuje daną
- Mamy 5 rodzajów dyrektywy `define`:
 - `db` definiuj bajt (alokacja 1 bajtu)
 - `dw` definiuj słowo (alokacja 2 bajtów)
 - `dd` definiuj podwójne słowo (alokacja 4 bajtów)
 - `dq` definiuj poczwórne słowo (alokacja 8 bajtów)
 - `dt` definiuj 10 bajtów (alokacja 10 bajtów)

Adresowanie argumentów

- Adresowanie rejestrowe (register)
 - Argumenty znajdują się w wewnętrznych rejestrach CPU
 - Najbardziej efektywny tryb adresowania
- Przykłady

`mov eax, ebx`

`mov bx, cx`

– Instrukcja `mov`

`mov destination, source`

– Kopiuje dane z `source` do `destination`

Adresowanie argumentów cd.

- Adresowanie natychmiastowe (immediate)
 - Argument jest zapisany w instrukcji jako jej część
 - Tryb jest efektywny, ponieważ nie jest potrzebny osobny cykl ładowania argumentu
 - Używany w przypadku danych o stałej wartości
- Przykład
 - `mov al, 75`
 - Liczba 75 (dziesiętna) jest ładowana do rejestru `al` (8-bitowego)
 - Powyższa instrukcja używa trybu rejestrów do wskazania argumentu docelowego oraz trybu natychmiastowego do wskazania argumentu źródłowego

Adresowanie argumentów cd.

- Adresowanie bezpośrednie
- Przykład
 - `mov al, response`
 - Asembler zastępuje symbol `response` przez jego liczbową wartość (offset) z tablicy symboli
 - `mov table1, 56`
 - Założymy, że obiekt `table1` jest zadeklarowany jako `table1 dw 20 DUP (0)`
 - Instrukcja `mov` odnosi się w tym przypadku do pierwszego elementu wektora `table1` (elementu o numerze 0)
 - W języku C równoważny zapis jest następujący:
`table1[0] = 56`

Adresowanie argumentów cd.

- Adresowanie pośrednie (indirect)
 - Offset jest określony pośrednio, z wykorzystaniem rejestru
 - Tryb ten czasem jest nazywany pośrednim rejestrówym (register indirect)
 - Przy adresowaniu 16-bitowym offset znajduje się w jednym z rejestrów: `bx`, `si` lub `di`
 - Przy adresowaniu 32-bitowym offset może znajdować się w dowolnym z 32-bitowych rejestrów GP procesora
- Przykład
 - `mov ax, [bx]`
 - Nawiązy [] oznaczają, że rejestr `bx` zawiera offset, a nie argument

Instrukcje przesyłań danych

- Najważniejsze instrukcje tego typu w asemblerze Pentium to:
`mov` (move) – kopij daną
`xchg` (exchange) – zamień ze sobą dwa argumenty
`xlat` (translate) – transluluje bajt używając tablicy translacji

Instrukcje arytmetyczne

- Instrukcje `inc` oraz `dec`
- Format:
`inc destination`
`dec destination`
- Instrukcja dodawania `add`
- Instrukcja odejmowania `sub`
- Instrukcja porównania `cmp`

Instrukcje skoków

- Skok bezwarunkowy `jmp`
- Skoki warunkowe
- Format
`j<cond> label`
- Instrukcja pętli `loop`
- Format
`loop target`

Instrukcje logiczne

- Format

```
and destination,source  
or destination,source  
not destination
```

- Działanie

- Wykonuje odpowiednią operację logiczną na bitach
 - Wynik jest umieszczany w `destination`

- Instrukcja `test` jest odpowiednikiem instrukcji `and`, ale nie niszczy `destination` (podobnie jak `cmp`) tylko testuje argumenty

```
test destination,source
```

Instrukcje przesunięć logicznych

- Format

- Przesunięcia w lewo
 - `shl destination,count`
 - `shl destination,cl`
 - Przesunięcia w prawo
 - `shr destination,count`
 - `shr destination,cl`

Instrukcje przesunięć cyklicznych

- Dwa typy instrukcji rotate

- Przesunięcie cykliczne przez bit przeniesienia
 - `rcl` (rotate through carry left)
 - `rcr` (rotate through carry right)
 - Przesunięcie cykliczne bez bitu przeniesienia
 - `rol` (rotate left)
 - `ror` (rotate right)