

AKO OPRACOWANIE PYTAŃ Z E-NAUCZANIA

A1. Omówić podstawowe zasady wykonywania programu przez procesor

Trochę niejasne pytanie, ale chyba chodzi o cykl rozkazowy CPU

Ładowanie Rozkazu z pamięci -> Dekodowanie rozkazu -> ustawienie EIP, żeby wskazywało na następny rozkaz -> obliczenie adresu argumentu -> wykonanie rozkazu -> zapisanie wyniku do rej. lub pamięci



Tak wygląda uproszczony diagram z początku wykładu później po dekodowaniu rozkazu wspomniano o fazach:

- **Ustawiania EIP na kolejny rozkaz**
- **Wyznaczania adresu argumentu**

Dla wielu rozkazów nie potrzeba wszystkich faz, np przesłanie `mov eax, [ebp]` nie wymaga ALU

A2. Ile linii adresowych potrzebnych jest do adresowania pamięci 64 GB?

$$2^{32} = 4\text{GB}$$

$$4 \cdot 16 = 64 \rightarrow 2^{32} \cdot 2^4 = 2^{36}$$

Potrzeba 36 linii adresowych

A3. Porównać własności różnych rodzajów pamięci stosowanych w komputerach.

1. **Rejestry** - najszybsza pamięć w komputerze, jest najdroższa i jest jej najmniej, znajduje się na procesorze, służą do tymczasowego trzymania wyników obliczeń, adresów. Stanowią najwyższy szczebel w hierarchii pamięci.
2. **Cache** - Inaczej pamięć kieszeniowa, najczęściej typu **SRAM** dzieli się na on-chip i off-chip

Pamięci dynamiczne są zbyt wolne dla współczesnych CPU, przy dostępie występują stany oczekiwania. Pamięć cache zawiera pewną ilość obszarów z danymi (tzw linii wierszy), które służą do kopiowania bloków **pamięci operacyjnej**.

Może przechowywać rozkazy i dane.

Typowy blok zawiera 4-64 bajty.

W trakcie wykonywania rozkazów CPU szuka najpierw rozkazów i danych w pamięci podręcznej:

- hit
- miss (granica opłacalności to 20%, aktualnie dąży się do 5%)

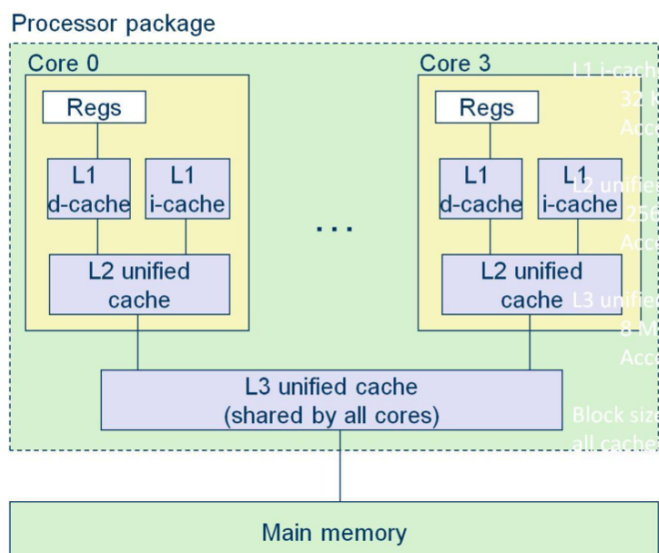
W i7 wygląda to tak:

(L0 Rejestry)

L1. Cache on chip, osobno dla każdego rdzenia (zintegrowana z rdzeniem) z podziałem osobno na dane i adresy

L2. Cache on chip, osobno dla każdego rdzenia (zintegrowana z rdzeniem) dla rozkazów i danych razem

L3. Cache off chip, wspólny dla wszystkich rdzeni



Zapewnienie spójności cache i ram:

1. Zapis przez (**write-through**) - zapis do ram po każdym zapisie w cache (wolne)
2. Zapis z opóźnieniem (**write back**) - zamiast zapisu do ramu zmienia się tylko **bit stanu**, trzeba aktualizować jeśli więcej wyjątków korzysta z tej pamięci

Pamięć SRAM (Przerzutnik dwustanowy)

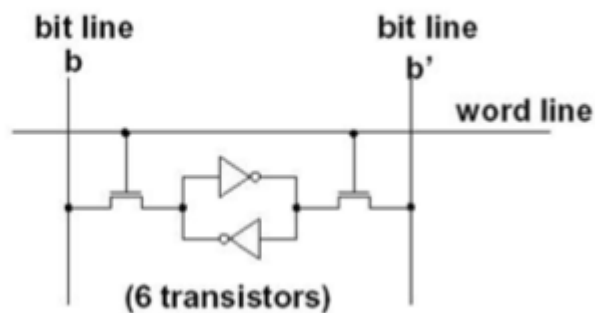
Zalety:

- Szybki dostęp do danych
- krótszy czas cyklu odczytu (nie trzeba odświeżać)
- wysoka odporność na zakłócenia

Wady:

- Koszt
- Niski stopień scalenia
- Duży pobór mocy

Pamięć statyczna SRAM



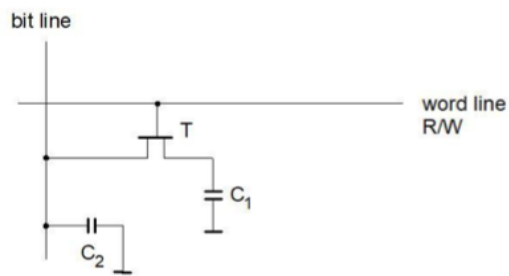
3. **Pamięć operacyjna** zazwyczaj DRAM (tranzystor i kondensator). Przechowywane są w niej aktualnie wykonywane programy i ich dane, wyniki. Ponieważ jest to pamięć dynamiczna to przed kolejną operacją odczytu trzeba ją odświeżać

Zalety:

- niski koszt
- względna energooszczędność
- duży stopień scalenia

Wady:

- podatność na zakłócenia
- czas dostępu
- czas cyklu odczytu
- konieczność odświeżania



FPMRAM -> SDR -> DDR -> DDR2

Jest bardzo wrażliwa na zakłucenia:

- dodatkowy bit (parity bit)
- w komp powyżej 1gb ram pełniących odpowiedzialne funkcję stosuje się pamięć ECC (Error checking and correction)

4. **Pamięć masowa** - np dysk twardy, bez dostępu do konkretnych bajtów

5. **ROM** -

- pamięć nieulotna
 - program inicjalizujący pracę komputera
1. ROM - zawartość w trakcie produkcji
 2. PROM (programmable ROM) jednokrotnie przez usera
 3. EPROM (erasable PROM) możliwość usuwania przy pomocy UV
 4. EEPROM (electrically EPROM)
 5. FLASH - pamięci błyskowe, możliwe jest programowanie całych bloków
 6. NVRAM połączenie SRAM z EEPROM

A4. W jaki sposób zmienia się zawartość wskaźnika instrukcji EIP w procesorach rodziny x86 w trakcie wykonywania różnych typów rozkazów?

1. Rozkazy nie sterujące
 $EIP = EIP + \text{długość rozkazu}$
2. Rozkazy sterujące
 - Skoki warunkowe np jmpe, jmpb
 - skoki bezwarunkowe np jmp etykieta
 - call, int
 - pośrednie/bezpośrednie np jmp dword ptr zmienna
 - zaliczyłbym tutaj też loop

A5. Scharakteryzować grupę instrukcji procesora określanych jako operacje bitowe.

ROL, ROR
 SHL, SHR

SAR, SAL
RCL, RCR
BT, BTR, BTS
AND, OR, XOR, MOT

A6. Wyjaśnić zasady działania układów DMA w komputerze.

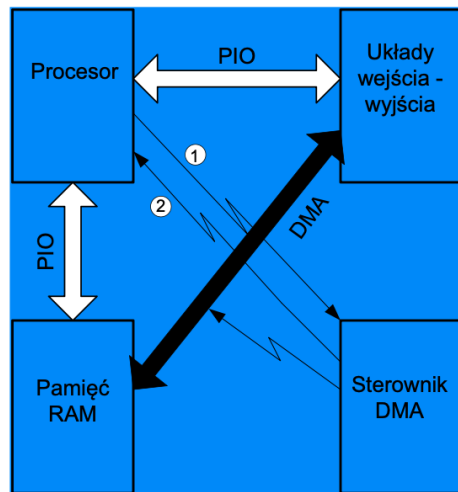
DMA (Direct Memory Access) - Technika przesyłania danych z pamięci głównej do urządzeń (lub odwrotnie) z pominięciem CPU.

Trzeba jedynie odpowiednio zainicjalizować układ DMA.

Po przesłaniu wszystkich bajtów DMA wywołuje przerwanie sprzętowe sygnalizując koniec przesyłania.

Moduł DMA potrzebuje takich informacji:

- rodzaj operacji (zapis, odczyt)
- adres urządzenia I/O
- adres obszaru RAM przewidzianego do odczytania/zapisania
- liczba bajtów



DMA używane są powszechnie do szybkich urządzeń np dyski, sterowniki usb
Istotne trudności w korzystaniu z DMA pojawiają się gdy PC ma pamięć podręczną.

B1. Omówić różne rodzaje kodowania liczb binarnych w komputerze.

1. NKB - zwykły dwójkowy
2. U2 (ze znakiem), najstarszy bit to liczba ujemna
Fajny sposób na konwersję NKB -> U2 (gdzie liczba nkb jest naszą liczbą, ale bez minusa)
Lecimy od tyłu (od najmłodszego bitu) aż pojawi się jedynka, zostawiamy ją w spokoju, a wszystkie bity za nią negujemy
np:
 $4 = 0100$
 $-4 = 1100$
3. ZM najstarszy bit jest znakiem
4. BCD - 4 bity kodują cyfry

5. Zmiennoprzecinkowe:
- np **float** 32 bity. niejawną jedynką
 - Z | 8 bitów wykładnika | 23 bity mantysy
 - <https://www.h-schmidt.net/FloatConverter/IEEE754.html>
 - double** 64 bity, niejawną jedynką
 - Z | 11 | 52
 - Format 80-bitowy** - jawna jedynka, wykorzystywany przez koprocessor w x86
 - Z | 15 | 64

B2. W jaki sposób w procesorach zgodnych z architekturą x86 sygnalizowane jest wystąpienie nadmiaru w operacjach dodawania, odejmowania, mnożenia i dzielenia na liczbach stałoprzecinkowych?

1. Dodawanie i odejmowanie liczb w NKB
Odejmowanie jest realizowane przez CPU jak dodawanie liczby przeciwnej.
CF zostaje ustawiony na 1, jeśli wystąpił nadmiar.
CF jest wyznaczone jako wartość przeniesienia przy dodawaniu dwóch ostatnich bitów
2. Dodawanie i odejmowanie liczb w NKB
Ustawiana jest OF
OF jest wyznaczana jako XOR CF i przeniesienia na ostatni bit
3. Mnożenie
Nadmiar nigdy nie wystąpi
4. Dzielenie
Nadmiar występuje gdy wyniki, lub reszta nie mieści się w przeznaczonych na nie rejestrach, albo gdy dzielimy przez 0. Zgłaszany jest wtedy wyjątek procesora (FAULT), więc można obsłużyć

B3. Na czym polegają różnice w sposobie przechowywania liczb w pamięci znane jako mniejsze niżej (ang. little endian) i mniejsze wyżej (ang. big endian)?

LE - mniejsze części liczby na niższych adresach, zazwyczaj stosowane
BE - na odwrót, większe części liczby na mniejszych adresach

B4. Omówić technikę porównywania liczb stałoprzecinkowych stosowaną w architekturze x86.

CMP <1>, <2> - jest to instrukcja porównująca (nie wpisująca nigdzie wyniku odejmowania), która ustawia odpowiednio znaczniki ZF, OF, CF
OF dodatkowo tylko w liczbach ze znakiem

SUB robi dokładnie to samo, ale zapisuje wynik
Liczby można porównywać również przy użyciu instrukcji bitowych np. and

B5. Dlaczego procesor w trakcie wykonywania rozkazu dodawania ADD ustawia jednocześnie dwa znaczniki nadmiaru CF i OF ?

Dlatego, że instrukcja add działa tak samo dla liczb w U2 i NKB, ale inaczej powstaje nadmiar i te dwie opcje rozróżnia OF i CF.

Procesor nie wie na jakim formacie wykonuje rozkaz, więc rolą programisty jest interpretacja flag, cpu wyznacza dwie

B6. Czym różni się rozkaz CALL od rozkazu INT?

CALL - służy do wywołania procedur, zapisuje na stosie ślad - adres wskazujący na rozkaz, który ma być wykonany zaraz po procedurze

Funkcje kończą się rozkazem RET, wpisuje on do EIP (wskaźnika instrukcji) ślad

INT - stosujemy przy wywołaniu podprogramu systemowego przy pomocy tablicy wektorów przerwań. Zapisuje on ślad w postaci IP, CS, i FLAGS (IP w połączeniu z cs daje adres kolejnej instrukcji). Powrót sygnalizują się rozkazem IRET

B7. Zadaniem poniższej sekwencji rozkazów jest zwiększenie o 1 liczby całkowitej znajdującej się rejestrach RDX:RAX. Uzupełnić brakujący argument drugiego rozkazu

```
add rax, 1  
adc rdx, 0
```

Przy odejmowaniu jest taki rozkaz **SBB**

B8. W jakich przypadkach (w architekturze x86) zamiast rozkazów mnożenia i dzielenia można zastosować rozkazy SAL i SAR ?\

SAL - Gdy mnożymy przez potęgę 2. Trzeba uważać, czy CF = 1, bo to oznacza, że nasza liczba może się zmniejszyć zamiast zwiększyć

SAR - Dzielenie przez potęgę 2, powiela on bit znaku, więc rozkaz nadaje się też do działań na liczbach w U2

Rozkazy te są znacznie szybsze niż standardowe dzielenie/mnożenie

B9. W jakiej sytuacji, w trakcie wykonywania rozkazu sterującego (skoku), procesor ignoruje zawartość pola adresowego tego rozkazu?

Gdy warunek nie zostanie spełniony. Wtedy $EIP = EIP + d\ell \text{ rozkazu}$

B10. Wyznaczyć wartość dziesiętną 32-bitowej liczby zmiennoprzecinkowej (format float)

Z | 8 bitów wykładnika | 23 b mantysy

0 100 0000 1 111 1000 0000 0000 0000 0000

Wykładnik = $129 - 127 = 2$

Trzeba pamiętać o niejawniej jedynce

$1.1111 \cdot 2^2 = 111.11 = 7.75$ (w dziesiętnym)

B11. Co oznacza termin wartości specjalne używany w kontekście koprocatora arytmetycznego?

Są to wartości, które nie wynikają z definicji formatu zmiennoprzecinkowego, ale da się je zapisać na koprocesorze. Należą do nich:

1. +/- 0 - Wykładnik i mantysa to same zera
2. +/- nieskończoność - Wykładnik same jedynki, mantysa same zera
3. Nan (Not a number) - Wynik niedozwolonej operacji np pierw z -1. Wykładnik same 1, mantysa nie same 1
4. Liczby z niedomiarem, Wykładnik same 0, Mantysa nie same 0

B12. Czym różnią się rozkazy koprocatora arytmetycznego: FLD i FST?

Rozkaz **FLD** ładuje liczbę zmiennoprzecinkową na wierzchołek stosu koprocatora z pamięci, lub ze stosu koprocatora

Rozkaz **FST** przesyła z wierzchołka stosu koprocatora liczbę zmiennoprzecinkową do pamięci, lub na inny rejestr koprocatora

B13. Zadaniem poniższej sekwencji rozkazów jest wpisanie liczby 27.0 na wierzchołek stosu rejestrów koprocatora arytmetycznego. Uzupełnić brakujący rozkaz

push dword ptr 27

FLD DWORD PTR [ESP] ; bo całkowita

add esp, 4

B14. Wyjaśnić w jakim celu zdefiniowano nieliczby (NaN) w koprocesorze arytmetycznym.

Złożone działania numeryczne trwają czasami bardzo długo, godziny, czasem dni.

Wystąpienie nadmiaru, albo niedomiaru nie powinno powodować załamania programu. Praktyka pokazuje, że wyniki pośrednie z nadmiarem/niedomiarem mają często mały wpływ na wyniki końcowe.

B15. Do jakiej klasy wartości specjalnych liczb zmiennoprzecinkowych należy zaliczyć niżej podaną wartość typu float?

1	11111111	000000000000000000000000
---	----------	--------------------------

Jest to minus nieskończoność - znak to minus, wykładnik to same jedynki, a mantysa same zera

B16. Poniżej podano reprezentację binarną dwóch 32-bitowych liczb binarnych zmiennoprzecinkowych (format float) x i y. Ile wynosi iloraz x/y tych liczb ? Wynik podać w postaci liczby dziesiętnej (trzy cyfry po kropce)

x	0	01111111	000000000000000000000000
y	1	10000000	100000000000000000000000

x:

wykładnik: $127 - 127 = 0$

trzeba pamiętać o niejawnej "1"

+ 1.0 = 1 (dziesiętnie)

y:

znak to minus

wykładnik: $128 - 127 = 1$

11.0 = 3 (dziesiętnie)

-3

-0.333

B17. Omówić podstawowe zasady kodowania rozkazów procesora.

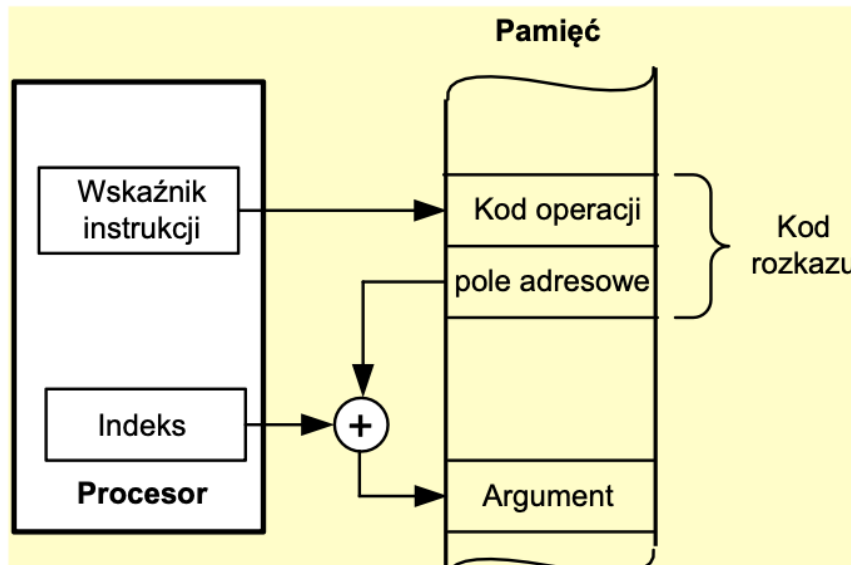
Producenci przygotowują instrukcję, przyporządkowują im konkretne ciągi zer i jedynek
Ciąg operacji dwuargumentowych powinien zawierać:

1. kod operacji
2. położenie pierwszego operandu
3. położenie drugiego operandu
4. informację dokąd przesłać wynik
5. Gdzie znajduje się kolejny wynik (Jeśli nie używa się wskaźnika instrukcji)

Konstruktorzy zwracają szczególną uwagę na długość instrukcji dlatego często wprowadza się ograniczenia takie jak np:

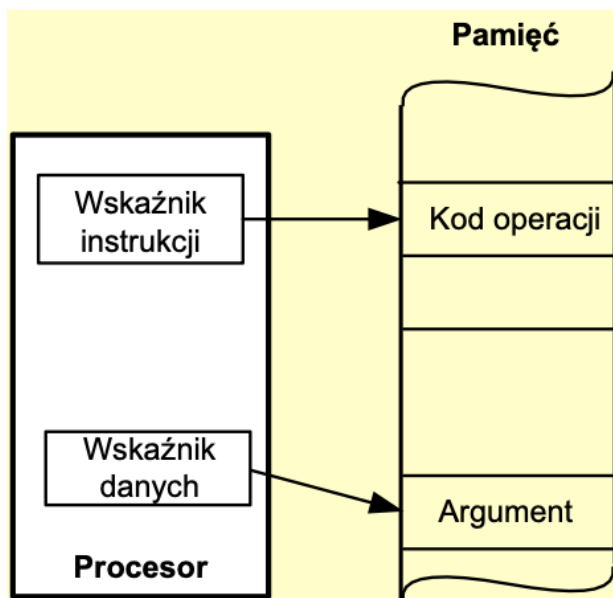
- wynik operacji wpisywany jest w miejsce pierwszego operandu
- Co najwyżej jeden argument może wskazywać na pamięć

C1. Omówić podstawowe koncepcje adresowania indeksowego.



np `add eax, [ebx+36]`

Szczególny przypadek adresowania indeksowego to **adresowanie za pomocą wskaźników**, wtedy sam rejestr wskazuje adres



np `add eax, [ebx]`

C2. Porównać wyznaczanie adresu efektywnego za pomocą instrukcji LEA i operatora OFFSET.

Oba te rozkazy obliczają przesunięcie elementu względem początku obszaru danych:

Rozkaz LEA wyznacza adres efektywny **w trakcie wykonywania programu**,
Wartość operatora OFFSET obliczana jest **w trakcie translacji (aseblacji) programu**.

C3. W jakim celu stosowany jest współczynnik skali w wyrażeniach adresowych?

Wsp skali może być tylko 1, 2, 4, 8

Często jest używany, by ułatwić uwzględnienie wielkości zmiennych np w tablicy, albo do pomijania niektórych części. Często używa się go w przypadku iterowania po jakiejś tablicy, gdy rejestr to nasz index, który dodajemy do adresu bazowego, tutaj np pierwszy element tablicy.

C4. W jakim celu niektóre rozkazy poprzedza się przedrostkiem zmiany rozmiaru argumentu (66H)?

W celu identyfikacji tego, że chcemy wykonać operację na rejestrach o rozmiarze 16 bitów. Jeśli bit w = 1 to standardowo wykonują się na 32 bitach, żeby działać na 16 trzeba dołożyć przedrostek w postaci bajtu o wartości 66h.

D1. Jak należy rozumieć termin licznik lokacji w kontekście programu asemblerowego?

- Licznik lokacji = rejestr programowy
- Określa adres kom. pam. do której zostanie przesłany aktualnie tłumaczony rozkaz lub dana
- Po załadowaniu rozkazu lub danej licznik lokacji jest zwiększany o jej wielkość
- Przed rozpoczęciem tłumaczenia pierwszego wiersza licznik lokacji =0

Jeśli asembler napotka wiersz zawierający definicję symbolu to rejestruje go w **słowniku symboli** przypisując mu wartość licznika lokacji. Zapisywane są też jego atrybuty takie jak np BYTE, WORD itp

Licznikiem lokacji można posługiwać się w programie za pomocą symbolu "\$".
Reprezentują bieżącą lokację wewnątrz tłumaczonego kodu

Czasami używa się **jmp \$+2** w celu wprowadzenia dodatkowego opóźnienia

Można zmieniać wartość licznika lokacji przy pomocy **ORG**

ORG 100h - wpisanie do licznika lokacji 100h

ORG \$ + 7 - zwiększenie licznika lokacji o 7\

Możemy zostawić sobie dzięki temu pewien niezaalokowany obszar, który możemy później wykorzystać.

D2. Wyjaśnić znaczenie terminu: zmienne czasu translacji.

Przekształcenie kodu źródłowego w assemblerze na ciągi zerojedynkowe zrozumiałe przez CPU realizowane jest w kilku etapach, z których pierwszym jest asemlacja. Polega ona na przekształceniu wierszy źródłowych na ciągi zerojedynkowe, ale pozostawia pewną elastyczność, która pozwala na późniejsze dołączenie podprogramów bibliotecznych i innych modułów programowych. Kod wygenerowany przez assembler jest kodem w języku pośrednim.

To właśnie w trakcie asemlacji obliczana jest zmienna rozmiar z przykładu:

```
blad_par db 'Podano błędne parametry'
rozmiar = $ — blad_par
```

Rozmiar jest zmienną czasu translacji tzn:

- nie jest dla niej zarezerwowany obszar pamięci w programie wynikowym
- funkcjonuje ona tylko w trakcie translacji
- Nigdzie nie znajduje się jej deklaracja
- jest to charakterystyczne dla języków wysokiego poziomu
-

E1. W jaki sposób rozkazy PUSH i POP wpływają na stan wskaźnika stosu ESP?

PUSH - zmniejsza wartość ESP o rozmiar argumentu (stos rośnie w kierunku malejących adresów), (2, albo 4 bajtowe - NIE DA SIĘ 1)

POP - zwiększa wartość ESP o rozmiar argumentu (2, albo 4)

E2. W jaki sposób można usunąć ze stosu trzy liczby 32- bitowe nie używając instrukcji POP?

add esp, 12 ; Bo 3 razy 4 bajty

E3. Omówić sposób dostępu zmiennych dynamicznych umieszczonych na stosie za pomocą indeksowania z użyciem pomocniczego wskaźnika stosu EBP.

Żeby wygodnie używać zmienne dynamiczne należy wykonać standardowy prolog w postaci:

```
push ebp
mov ebp, esp
```

Aby zarezerwować zmienne dynamiczne należy odjąć od esp ilość potrzebnych bajtów np rezerwacja 12 na 3 zmienne 4 bajtowe wygląda tak:

```
sub esp, 12
```

Adresy tych zmiennych to:
[ebp-4], [ebp-8], [ebp-12]

na koniec wykonywania programu należy pamiętać o zwolnieniu pamięci wykonując rozkaz

```
add esp, 12
```

E4. Porównać typowe do techniki podprogramu przekazywania stosowane w parametrów procesorach CISC i RISC.

W procesorach CISC typowe jest przekazywanie przez stos

Standardy	Kolejność ładowania na stos	Obowiązek zdjęcia parametrów
Pascal	L->P	Podprogram wywoływany
C	P->L	Podprogram wywołujący
StdCall	P->L	Podprogram wywoływany

FastCall - przez rejestry, nie ma standardów, kompilatory różnie to realizują

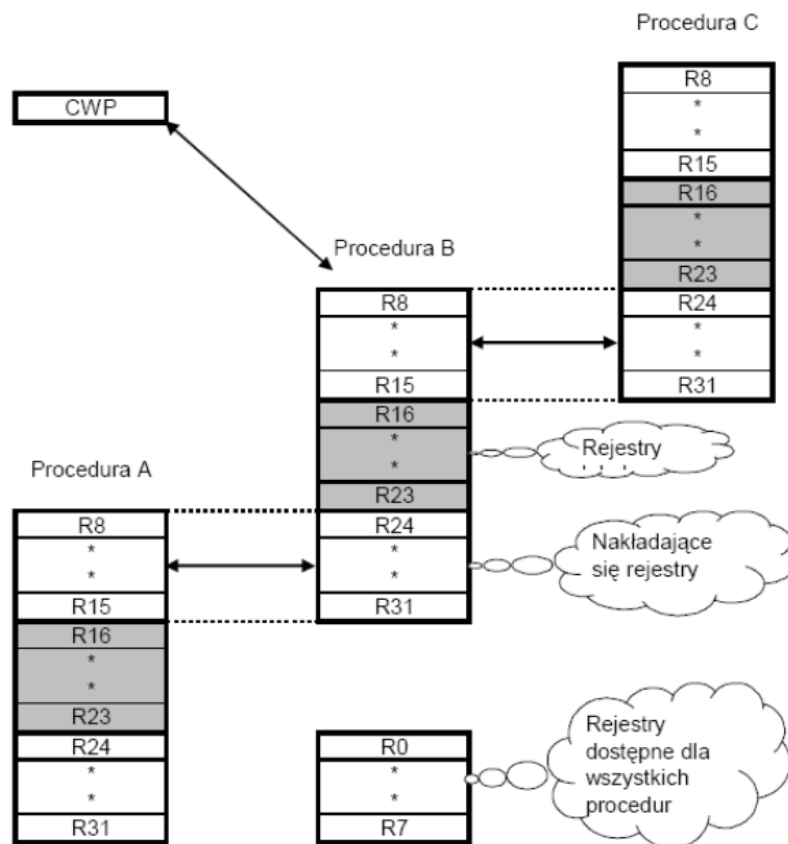
Przekazywanie przez ślad - Parametry są podane bezpośrednio w kodzie tuż za call

W procesorach RISC działa na:

- parametrach przekazanych podczas wywołania
- zmiennych lokalnych, w których zapisywane są wyniki pośrednie
- zmiennych globalnych, dostępnych także dla innych funkcji

W procesorach RISC wszystkie te zmienne i parametry przechowywane są w rejestrach zorganizowanych w postaci tzw okien; w typowych rozwiązaniach okno zawiera 32 rejestry, a liczba okien to np 16. Każda procedura korzysta z własnego okna rejestrów, przy czym okna sąsiednich procedur (w sensie wywoływania) częściowo się na siebie nakładają, co ułatwia przekazywanie parametrów. Wywołanie procedury powoduje automatyczne przełączenie się procesora do użycia innego okna rejestrów

Takie rozwiązanie musi posiadać też licznik wywołań.



E5. Dlaczego wiele programów generowanych przez kompilatory języków wysokiego poziomu używa stosu do przechowywania wartości zmiennych?

Gdyby kompilatory stosowały zmienne zdefiniowane w segmencie .data to byłyby to zmienne globalne, dostępne dla wszystkich funkcji, co w wielu przypadkach byłoby niepożądane. Funkcje mogłyby zmieniać zawartość zmiennych potrzebnych w innych funkcjach.

Deklarowanie zmiennych zwiększa objętość kodu i czas jego wykonania
Korzystanie ze stosu umożliwia szybkie alokowanie i zwalnianie pamięci

E6.W jaki sposób, na poziomie kodu asemblerowego, zdefiniowaną w funkcję można wywołać Win32 API?

Win32API - Standard StdCall

na początku programu trzeba dodać "extern _funkcja@x: PROC"

gdzie x to liczba bitów przekazywanych argumentów

Przed rozkazem call należy w załadować na stos argumenty w kolejności od prawej do lewej.

Po rozkazie call nie musimy zdejmować nic ze stosu (StdCall)

F1. Omówić bitowe operacje logiczne wykonywane przez instrukcje procesora.

[Link](#) <- było wcześniej

G1. Omówić podstawowe koncepcje komunikacji komputera z urządzeniami zewnętrznymi

CPU nie komunikuje się bezpośrednio z urządzeniami zewnętrznymi, są one zazwyczaj połączone przez układy pośredniczące, adaptery, które dopasowują sygnały CPU i płyty głównej do specyficznych wymagań poszczególnych urządzeń.

Kiedyś: dużo kart rozszerzeniowych

Teraz: Funkcje tych kart realizuje chipset

Metody sterowania pracą urządzeń zewnętrznych:

Metoda aktywnego oczekiwania (odpytywania)

- pytanie cały czas, czy dana dostępna
- jeśli wynik pozytywny to coś robimy
- nieefektywna, zabiera czas cpu
- zdarzenie może nigdy nie nastąpić
- występują przerwy w obsłudze urządzeń, co jest czasami niedopuszczalne np nieodczytany bajt jest zamieniany przez kolejny, a przy częstym odpytywaniu marnujemy czas cpu
- dobra dla prostych urządzeń, gdy cpu nie jest obciążony np urządzenia domowe, sterowniki oświetlenia
- nie wymaga rozbudowanego sprzętu
- jak jest bardzo dużo wolnych urządzeń to git
- w starych komputerach

Metoda przerw

- Cpu po otrzymaniu sygnału przerwania przerywa wykonywanie aktualnego programu i zaczyna obsługę przerwania (zazwyczaj SO)
- Po wykonaniu CPU wysyła sygnał do urządzenia i kontynuuje działanie

Przerwanie - zdarzenie asynchroniczne

Istnieje możliwość przesyłania danych z urządzeń zewnętrznych do pamięci operacyjnej (lub na odwrót) przy pomocy układu DMA. Wówczas sterownik wysyła sygnał przerwania na początek transferu danych i drugie oznaczające koniec transferu.

Przed użyciem trzeba zainicjować moduł DMA, potrzeba do tego:

- adresu urządzenia I/O
- rodzaj działania (zapis/odczyt)

- adres pamięci z której mają być wysłane/ gdzie mają trafić dane
- rozmiar pamięci

DMA używa się powszechnie do szybkich urządzeń np sterowniki USB
Stosowanie DMA w przypadku CPU z pamięcią cache jest skomplikowane

G2. Omówić podstawowe elementy systemu przerwań stosowanego w komputerach PC.

Przerwanie - zdarzenie asynchroniczne

CPU po otrzymaniu sygnału przerwania przerywa wykonywanie aktualnego programu i zaczyna obsługę przerwania (Zazwyczaj SO).

Zapisuje ślad przerwania, czyli IP, CS i znaczniki, na podstawie IP i CS da się wyznaczyć adres następnego rozkazu

W chwili przyjęcia przerwania zerowana jest flaga IF(interrupt enable flag, STI, CLI)

Przerwanie nie może zmienić żadnych rejestrów

Program obsługi przerwania kończy IRET, który na podstawie śladu przerwania na stosie odtwarza IP,CS i znacznik.

Po wykonaniu CPU wysyła sygnał do urządzenia i kontynuuje pracę

Obsługa przerwań jest ściśle związana z tablicą wektorów(tryb rzeczywisty)/deskryptorów przerwań, które zawierają 256 4 bajtowych adresów w postaci segment:offset

Układ APIC (w trybie rzeczywistym kaskadowo 2*8259) to sekretarka procesora Sygnały przerwań są prowadzone do nich przez linie IRQ, które są bezpośrednio powiązane z tablicą wektorów przerwań

G3. Jaką rolę podczas wywoływania podprogramów systemowych pełni tablica wektorów (deskryptorów) przerwań?

Zawiera 256 adresów 4 bajtowych w postaci segment:offset, pod którymi znajdują się podprogramy systemowe, programista nie podaje adresu podprogramu, ale adres indeks tablicy adresowej. Adresy mogą się zmieniać, ale ich miejsce w tablicy zostaje takie same.

Skojarzenie wierszy i numerów linii wykonywane jest przez SO w trakcie inicjalizacji

G4. Wyjaśnić co oznacza termin "przestrzeń adresowa portów"

Oznacza izolowaną przestrzeń adresową dla urządzeń I/O. Dostęp do rejestrów I/O działa przy użyciu rozkazów in/out. Inaczej izolowane wejście/wyjście

G5. Jaką rolę w komunikacji z urządzeniami zewnętrznymi pełni obszar współadresowalny pamięci?

Jest to jedna z metod dostępu do rejestrów urządzeń I/O. Rejestry są udostępnione jako zwykłe komórki pamięci w przestrzeni adresowej pamięci. Można je odczytywać i działać na nich tak samo jak na zwykłych komórkach np. rozkazami mov, sub.

W ten sposób działa np. pamięć ekranu w trybie tekstowym - znaki wyświetlane na ekranie mają swoje odwzorowanie w przestrzeni adresowej procesora pod adresem fizycznym B8000h, ale fizycznie są zainstalowane gdzieś indziej.

G6. Dlaczego metoda przeglądania (odpytywania) stosowana jest głównie w systemach, w których występuje niewielkie obciążenie procesora?

Operacja wysyłania i odbierania musi być poprzedzona sprawdzeniem, czy dana jest dostępna. Jest ona niewydajna i zabiera cenny czas pracy procesora.

G7. Omówić sposób odwzorowania zawartości pamięci ekranu (w trybie tekstowym) na postać tekstu wyświetlanego na ekranie.

Tryb tekstowy:

Rozdzielczość: 80*25

Parzysty bajt: ASCII

Nieparzysty: Kolor tła, kolor tekstu, ew. miganie

Całkowita pamięć to $80 \times 25 \times 2 = 4000$ bajtów

Adres: B8000H

Tryb graficzny:

Rozdzielczość: 320*200px

256 kolorów na px tzn. 1 bajt

Całkowita pamięć to: $320 \times 200 = 64000$ B

Adres: A0000H, nr **13H**

G8. Jakie czynności podejmuje procesor po nadejściu sygnału przerwania?

Dokończy aktualny rozkaz

Zapisuje ślad przerwania IP, CS i znaczniki

Odczytuje z tablicy deskryptorów/wektorów adres i wpisuje go do EIP

Zaczyna operacje obsługi przerwania, tak jak zwykły podprogram

Zakończenie IRET, który przywraca IP, CS, znaczniki

Przy rozpoczęciu przerwania znacznik IF jest ustawiany na 0, co oznacza, że reszta przerwania musi czekać

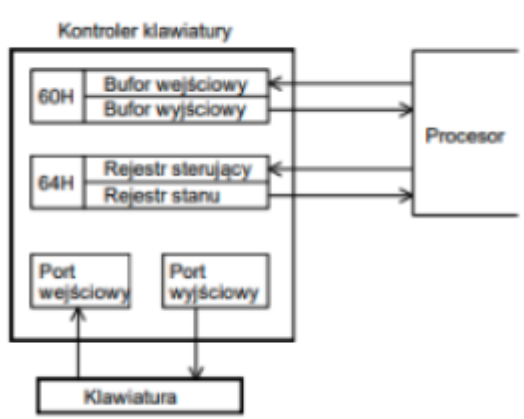
G9. Omówić sposób kodowania informacji przesyłanych między komputerem a klawiaturą.

Każdy z klawiszy ma swój 8 bitowy kod przycisku (scan code).

Po naciśnięciu przycisku mikrokontroler klawiatury na jego podstawie tworzy kod naciśnięcia (make code), albo zwolnienia (break code , poprzedzony bajtem F0h) - kod ten jest przesyłany do mikrokontrolera na płycie głównej. Układ na płycie głównej odtwarza scan code, który jest dostępny w porcie 60h in al, 60h

Make code i break code nie są dostępne na poziomie programowania

Scan code != make code



H1. Porównać charakterystyczne elementy architektury procesorów CISC i RISC.

CISC (Complex instruction set computer)

- nie ma konkretnej, ścisłej definicji
- duża ilość instrukcji
- duża ilość trybów adresowania
- instrukcje wykonują wyspecjalizowane czynności np MOVS
- mikroprogramowalna jednostka sterująca
- parametry przez stos
- różna długość rozkazów
- adresy mogą być niewyrównane

RISC (Reduced instruction set computer)

- nie ma konkretnej, ścisłej definicji
- mała ilość instrukcji
- mała ilość trybów adresowania
- odwołania do pamięci tylko za pomocą rozkazów load/store

- układowa instrukcja sterująca
- przetwarzanie potokowe
- parametry przez rejestry
- jednakowa długość rozkazów
- tylko wyrównane adresy

H2. Opisać zasady działania pamięci statycznych i dynamicznych.

Pamięci dynamiczne (DRAM) Tranzystor i kondensator

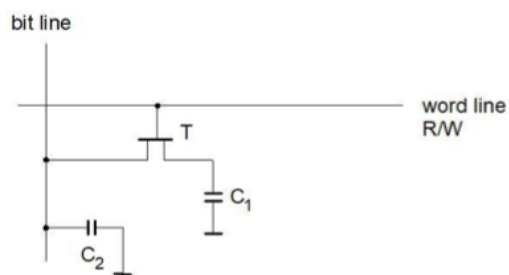
Zazwyczaj używana do pamięci operacyjnej.

Zalety:

- Względna energooszczędność
- niska cena
- duży stopień scalenie

Wady:

- niska odporność na zakłócenia
- wymaga odświeżania
- długi czas dostępu
- dłuższy czas cyklu odczytu



Pamięci statyczne (DRAM) Przerzutnik dwustanowy

Zazwyczaj używana w budowie pamięci kieszeniowej (cache).

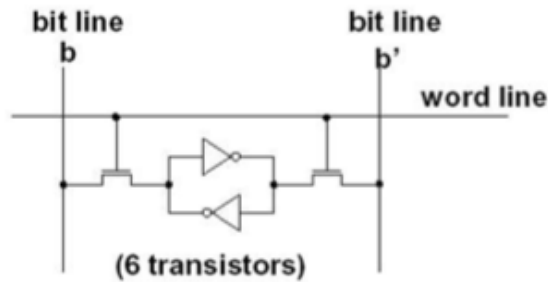
Zalety:

- szybki czas dostępu
- szybszy czas cyklu odczytu
- nie trzeba odświeżać
- odporna na zakłócenia

Wady:

- wysoka cena
- niski stopień scalenia

Pamięć statyczna SRAM



H3. Opisać technikę asocjacyjnego dostępu do pamięci podręcznej.

Adres pamięci składa się z etykiety i słowa. Żeby znaleźć dane w pamięci trzeba porównać etykiety, jeśli znajdziemy je w cache to następuje trafienie (hit), jeśli nie to miss i trzeba pobrać daną z pamięci operacyjnej.

Wyszukiwanie jest przeprowadzane układowo, każda linia ma swój komparator, dlatego też tworzenie takich dużych pamięci jest drogie i bardzo skomplikowane



H4. Omówić algorytmy dostępu i aktualizacji zawartości pamięci podręcznej.

Trochę nie wiadomo o co chodzi z algorytmami dostępu, ale chyba ma być coś takiego:

Algorytmu wyboru bloku do wymiany:

- Losowy - przesunięcie rejestrów, w praktyce jest bardzo często stosowany
- LRU (Least Recently Used) - usuwamy to, co było najdawniej używane
- FIFO
- LFO (Least Frequently Used)

Pod aktualizację możemy podciągnąć też metody zapewnienia spójności CACHE I RAM

- zapisze przez (write through) zapis do ram po każdym zapisie w cache, jest to wolna metoda
- Zapis z opóźnieniem (write back). Zamiast zapisu do ramu zmienia się tylko **bit stanu**. Trzeba aktualizować jeśli więcej wątków korzysta z tej pamięci. Pozwala to ograniczyć liczbę "uaktualnień"

H5. W jaki sposób zasada lokalności wiąże się z pamięcią podręczną?

Zasada lokalności pamięci - w ciągu krótkiego czasu CPU odwołuje się do wąskiego zakresu komórek. Dzielimy ją na:

- lokalność czasowa - wielokrotne odwołanie się do tych samych komórek pamięci, które niekoniecznie mają bliskie adresy (np. pętla)
- lokalność pamięciowa - wielokrotne odwołanie się do komórek pamięci o zbliżonych adresach, w sąsiedztwie (np do kolejnych elementów w tablicy)

W praktyce często odwołujemy się do elementów o zbliżonych adresach, więc pobranie od razu całego bloku tak jak dzieje się to w pamięci cache pozwala oszczędzić sporo czasu.

H6. Na czym polega problem zapewnienia spójności pamięci operacyjnej pamięci i zawartości podręcznej?

Pamięć dynamiczna jest zbyt wolna dla współczesnych cpu, więc stosuje się pamięć cache, która jest szybsza i przechowuje się w niej dane i rozkazy na których aktualnie działamy z pamięci operacyjnej. Po jej zmianie trzeba przenieść te wyniki również do pamięci operacyjnej.

- zapisze przez (write through) zapis do ram po każdym zapisie w cache, jest to wolna metoda
- Zapis z opóźnieniem (write back). Zamiast zapisu do ramu zmienia się tylko **bit stanu**. Trzeba aktualizować jeśli więcej wątków korzysta z tej pamięci. Pozwala to ograniczyć liczbę "uaktualnień"

H7. Wyjaśnić koncepcję pamięci podręcznej dwukanałowej.

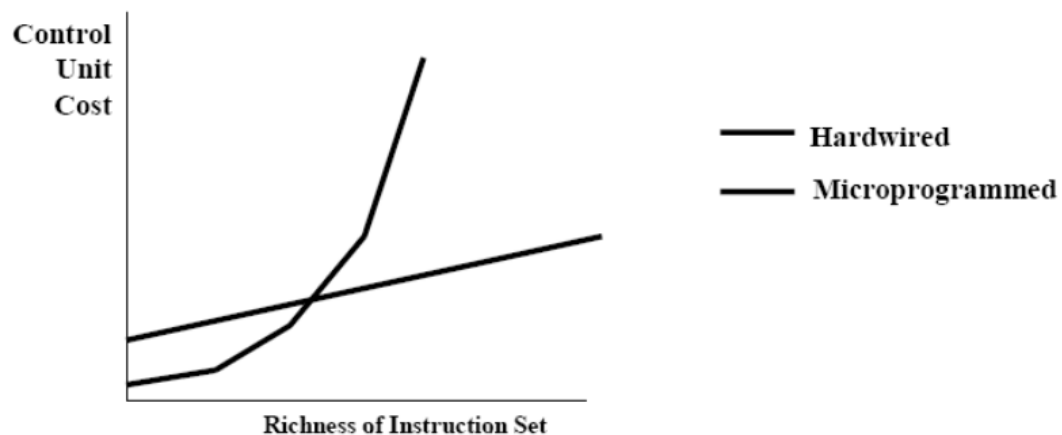
Jeśli stosujemy pamięć bezpośrednią dwukanałową to oprócz sprawdzenia numeru linii trzeba sprawdzić dwie etykiety. Pozwala to na bardziej elastyczne wypełnianie pamięci podręcznej, co w rezultacie zmniejsza liczbę chybień.

H8. Na czym polega różnica w sterowaniu mikroprogramowym i układowym procesora?

Jednostki sterujące dzielą się na układowe i mikroprogramowe. Te pierwsze stosowane są w komputerach typu RISC, a drugie w typu CISC. Jednostka sterująca na podstawie kodu rozkazu przesyła sygnały do podzespołów procesora w taki sposób, by wykonały swoje działania.

Układowe jednostki sterujące charakteryzują się tym, że generowanie mikroimpulsów do podzespołów cpu jest realizowane układowo przez duży, skomplikowany układ. Zaletą jest szybkość działania, a wadą to, że dla dużej ilości rozkazów zaprojektowanie takiego układu jest bardzo trudne i drogie. Wadą jest również to, że trudno dokonywać zmian, uaktualnień

Układy mikroprogramowe również generują mikro sygnały do podzespołów, ale robią to przy pomocy własnego mikroprocesora z własnym wskaźnikiem instrukcji wykonującym mikroprogram. Rozwiązanie to jest tańsze dla dużej ilości rozkazów. Pozwala na aktualizowanie listy rozkazów bez konieczności zwiększania rozmiaru układu oraz pozwala na względnie łatwe usuwanie błędów



H9. Omówić przyczyny utrudniające realizację przetwarzania potokowego w procesorach.

Zakłócenia w przetwarzaniu potokowym to **hazardy**

1. Skoki - każdy skok czyści potok
2. Konflikty z pamięcią
3. różnica czasowa wykonywanych rozkazów
4. przerwania sprzętowe
5. niektóre rozkazy wymagają dodatkowych cykli (np. ładowanie danych)
6. czasami z powodu zależności danych rozkazy muszą czekać **pipeline stall**

H10. W jaki sposób we współczesnych procesorach przewiduje się zachowanie instrukcji rozgałęzień?

Z problemem rozgałęzień można walczyć w następujący sposób:

- zwielokrotnienie strumieni
- bufor pętli
- kodowanie bez skoków

- **przewidywanie statyczne** (75%) - nie zależy od historii skoków, przyjmuje np że skok zawsze nastąpi, albo nie, lub, że dla pewnych rodzajów rozkazów zawsze nastąpi, a dla innych nie, to ostatnie daje 75 % szans na sukces
- **przewidywanie dynamiczne** - oparte na historii wykonanych skoków. Rejestruje się wykonanie, lub niewykonanie skoku za pomocą **bitu skoku**. Jeśli skok wystąpił wcześniej zakłada się, że następnym razem też wystąpi, w przypadku pętli daje to tylko dwie pomyłki. Aby uzyskać lepsze rezultaty stosuje się złożone tablice historii skoków, na podstawie których wybiera się najbardziej prawdopodobną ścieżkę wykonania programu (np **Branch Target Buffer**)

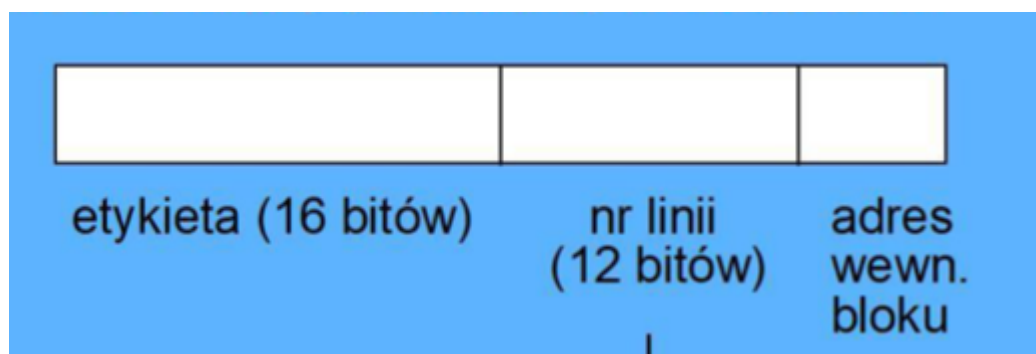
H11. W pewnym systemie zastosowano adresowanie 16- bitowe. Pamięć podręczna ma rozmiar 4096 bajtów, jest 2 kanałowa, a jej bloki są 32 bajtowe. Przedstaw obliczenia i podaj wartość etykiety, numer linii i offset w linii dla komórki o adresie 0123h.

$$4096 = 2^{10} \cdot 2^2 = 2^{12}$$

$$32 = 2^5$$

$$2^{12} / 2^5 = 2^7 \text{ - tyle by było linii dla jednokanałowej}$$

$$2^7 / 2 = 2^6 \text{ - tyle linii}$$



tak wygląda struktura adresu (pomijamy wielkości, chodzi o to, że najpierw etykieta - nr linii i offset)

na **offset** musimy poświęcić 5 bitów

na **nr linii** 6 bitów

czyli na etykietę zostaje 5 bitów

0000 00**1** 00**1**0 00**1**1

etykieta = 0

nr linii - 00**1** 00**1** = 9

offset = 3