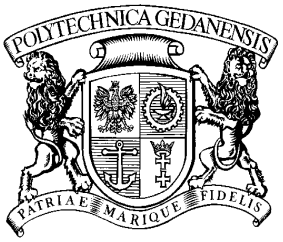
POLITECHNIKA GDAŃSKA
WYDZIAŁ ELEKTRONIKI, TELEKOMUNIKACJI i INFORMATYKI
KATEDRA ARCHITEKTURY SYSTEMÓW KOMPUTEROWYCH



Architektura komputerów

Opracowanie pytań z egzaminów:
zebrane i poukładane
by Conek

Gdańsk 2008

1. *W jakim celu w modyfikacjach adresowych stosuje się modyfikatory *2 *4 *8*

Stosowane w trybie 32-bitowym, przy drugim rejestrze modyfikacji (index). Zawartość rejestru mnożona będzie przez współczynnik skali i dodawana podczas obliczania adresu efektywnego. Zawartość drugiego rejestru nazywamy rejestrem modyfikacyjnym index`owym (pomocne przy dostawaniu się do danej komórki tablicy). Rozmiar pojedynczej komórki mnożony jest wtedy przez współczynnik skali.

2. *Jak można usunąć ze stosu 3 liczby 16-bitowe, nie używając POP`a*

Np.: można użyć instrukcji *add* esp, 6
a 32-bitowe, add esp, 12

3. *Jaka rolę pełni licznik lokacji w trakcie asemblacji programu*

Jest to rejestr programowy definiowany przez asembler. Określa lokację w pamięci operacyjnej, do której zostanie przesłany aktualnie tłumaczony rozkaz lub dana. Po załadowaniu rozkazu lub danej, licznik lokacji zostaje zwiększony o liczbę bajtów zajmowanych przez ten rozkaz lub daną. Licznik nie wskazuje adresu fizycznego ładowanej lokacji, lecz jedynie jej położenie względem segmentu. W trakcie tłumaczenia pierwszego wiersza licznik lokacji = 0. Aktualna wartość licznika lokacji reprezentowana jest znakiem $.

4. *Techniki porównywania liczb stałoprzecinkowych w procesorach Pentium*

a) CMP – instrukcja porównująca ustawia znaczniki ZF i CF, ale nigdzie nie zapisuje wyniku operacji (odejmowanie)
Za pomocą instrukcji odejmowania ustawia znaczniki ZF i CF i zapisuje wynik operacji.
Jeżeli ZF=1 i CF=0 to są np. bx=cx
Jeżeli ZF=0 i CF=0 to są bx>cx
Jeżeli ZF=0 i CF=1 to są bx<cx
Dla liczb ze znakiem sprawdza się zawartość znaczników: ZF, OF, SF
Dla liczb bez znaku: ZF, CF
Za pomocą instrukcji bitowych. Te instrukcje ustawiają znaczniki: ZF, SF, PF (CF i OF są zerowane).

b) SUB – instrukcja odejmowania, która również ustawi znaczniki ZF i CF, ale wadą może być konieczność zapisania wyniku.

5. *Jakie argumenty przemawiają za stosowaniem architektury segmentowej*

- wskazanie położenia danej, lub rozkazu w pamięci wymaga niewielkiej liczby bitów
- programy są krótsze i zajmują mniej miejsca w pamięci, wykonują się szybciej
- stałe części adresów przechowywane w rejestrach segmentowych
- procesor generuje adresy, które mają skłonność do skupiania się (zasada lokalności)

6. *Arytmetyka nasycenia w operacjach MMX*

Arytmetyka przewinięcia	Arytmetyka nasycenia	
Liczby ze znakiem i bez znaku	liczby bez znaku	liczby ze znak.
Dodawanie: 9000H + A010H = 3010H	Dodawanie: 9000H + A010H = FFFFH (*nasycenie do górnej granicy*)	dodawanie: 9000H +A010H = 8000H (*nasycenie do dolnej granicy*)
Odejmowanie: 9000H – A010H = EFF0H	Odejmowanie: 9000H – A010H = 0H (*nasycenie do dolnej granicy*)	odejmowanie: 9000H – A010H = EFF0H (*nie ma nasycenia*)

9	0	0	0	H
1001	0000	0000	0000	b

A	0	1	0	H
1010	0000	0001	0000	b

Arytmetyka liczb bez znaku, jest zrozumiała ☺.

Arytmetyka liczb ze znakiem:

a) **dodawanie**:

1001	0000	0000	0000	b (9000H)
1010	0000	0001	0000	b (A010H)
(1)0011	0000	0001	0000	b
1000	0000	0000	0000	b
8	0	0	0	H

b) **odejmowanie** (bez nasycenia):

zamieniam A010H na kod w U2 (wstawiam zera zaczynając od prawej strony do napotkania pierwszej jedynki przepisując ją, a później negujemy następne bity)

1010	0000	0001	0000	b (A010H)
0101	1111	111**1**	**0000**	b (5FF0H)

1001	0000	0000	0000	b (9000H)
0101	1111	1111	0000	b (5FF0H)
1110	1111	1111	0000	b
E	F	F	0	H

7. *Jakie wnioski praktyczne wynikają z prawa Amdahla*

Uogólnienie przedstawionego problemu nosi nazwę *skalowalności systemu*; chodzi tu o to czy zwiększenie liczby procesorów, czy innych zasobów systemu prowadzi do proporcjonalnego

zwiększenia wydajności; dyskusja wynikająca z prawa Amdahla wskazuje, że uzyskiwane rezultaty przy dużej liczbie procesorów są dość umiarkowane. Wynika to z faktu, że często algorytmy nie są dostosowane (lub nie moga być dostosowane) do pracy równoległej; znaczna część obliczeń musi być prowadzona sekwencyjnie.

8. *Niedomiar stopniowany w koprocesorze arytmetycznym*

Jest to koncepcja kodowania bardzo małych liczb w taki sposób, aby były one zrozumiale dla koprocesora. Wykładnik zawierać musi same zera (traktowany będzie jakby zawierał liczbę 00...001). Najbardziej znaczący bit mantysy (niejawny) wynosi 0, zamiast 1. Poprzez zmniejszanie dokładności mantysy rozszerzamy wykładnik w kierunku ujemnych wartości. Oznacza to rezygnacje z warunku normalizacji (liczby znienormalizowane). Operacje na operandach nieznormalizowanych generują wyjątki koprocesora.

9. *W jakim celu niektóre rozkazy poprzedza się przedrostkiem zmiany rozmiary argumentu 66H*

W trybie rzeczywistym (i w trybie V86) standardowo używane są operandy 8- i 16-bitowe; jeśli jednak przed instrukcją wykonującą działanie na operandzie 16-bitowym zostanie umieszczony dodatkowy bajt 66H, to działanie zostanie wykonane na operandzie 32-bitowym; ten dodatkowy bajt nosi nazwę *przedrostka rozmiaru operandu*.

10. *Problem zapewnienia spójności zawartości pamięci operacyjnej i pamięci podręcznej*

Istotnym problemem jest zapewnienie spójności zawartości pamięci operacyjnej (głównej) i pamięci podręcznej; problem ten nie występuje jeśli pamięć operacyjna używana jest tylko do przechowywania rozkazów; stosowane są dwie podstawowe metody:

- metoda *zapis przez* (ang. write-through) wykonuje zapis do pamięci głównej po każdej operacji zapisu w pamięci podręcznej; wadą tej metody jest często występujące oczekiwanie na wykonanie zapisu w pamięci głównej, co podważa celowość stosowania pamięci podręcznej;
- metoda *zapis z opóźnieniem* (ang. write-back), polega na tym, że zamiast natychmiastowego zapisu bloku do pamięci głównej, zmienia się tylko bit stanu, oznaczający, że wiersz bufora został zmodyfikowany; zmodyfikowany blok jest kopiowany do pamięci głównej, dopiero, gdy trzeba go zastąpić innym; jednak w systemach wieloprocesorowych czy też przypadku używania transmisji DMA zapewnienie spójności tą metodą może być problemem.

11. *Bufor okrężny, zasady działania (np.: w klawiaturze)*

Bufor klawiatury w systemie BIOS ma organizację FIFO i jest zorganizowany jako bufor okrężny. Zajmuje 32 bajty i jest przeznaczony do przechowywania maksymalnie 15 elementów (dwubajtowych) dwa bajty pozostają niewykorzystane ale są niezbędne do poprawnej pracy procesora. Procedura obsługi przerwania sprzętowego z klawiatury rejestruje w kolejnych bajtach buforu kody naciśniętych klawiszy, natomiast procedura usługowa wywoływana przez program użytkownika pobiera znaki z bufora i przekazuje je do programu. Program obsługi korzysta z dwóch wskaźników 16-bitowych umieszczonych w obszarze zmiennych BIOSU – adresu kolejnego bajtu do pobrania (head) i pierwszego wolnego bajtu w buforze (tail). Zasadą działania bufora okrężnego jest przesuwanie jedynie tych wskaźników, zamiast przesuwania wartości bufora.

12. *W jaki sposób zasada lokalności, kształtuje rozwój podzespołów we współczesnych procesorach*

Dzięki lokalności czasowej i przestrzennej, uzasadnionym stało się stosowanie małej pojemnościowo, ale bardzo szybkiej pamięci statycznej. Pomimo prostej budowy (min. 6 tranzystorów) jest bardzo wydajna i nie wymaga dokonywania „odświeżania" danych. Służy ona jako pamięć podręczna, przechowane są w niej dane o największej ilości odwołań.

Teraz pozostaje problem, jak określić te dane, które będą najczęściej używane w przód, pamiętając przy tym o bardzo ograniczonej pojemności pamięci. O ile przy zwykłych procesorach stanowi to pewien problem, a co dopiero przy wielordzeniowych, gdzie ładuje się kilka potoków danych naraz (np.: jak mamy skok warunkowy, jak przewidzieć czy skoczy ?). Stosuje się do tego tzw. bity historii, które uwzględniając przeszłe skoki, aby dokonać decyzji, a w nowych procesorach IA-64 wprowadzono mechanizm predykatowego wykonywania instrukcji, eliminujące rozgałęzienia w programie (wymaga innej techniki programowania).

13. *FLD i FST w koprocesorze arytmetycznym*

FLD – ładowanie na wierzchołek stosu koprocesora liczby zmiennoprzecinkowej pobranej z lokacji pamięci lub ze stosu koprocesora.

FST – przesyłanie zawartości wierzchołka stosu do lokacji pamięci lub do innego rejestru stosu koprocesora.

14. *Omówić podstawowe koncepcje potokowego wykonywania rozkazów przez procesor*

Jest to pewna technika wykonywania rozkazów etapami, przyspieszająca prace procesora, można ją przyrównać do linii montażowej (na jednym końcu potoku przyjmowane są nowe elementy wejściowe, zanim jeszcze elementy poprzednio przyjęte ukażą się na wyjściu).

Każdy etap zajmuje jeden cykl zegarowy, po czym przyjmuje nowy rozkaz do potoku, a obecny przechodzi dalej. To nie skraca czasu wykonywania programu, ale zwiększa całkowitą przepustowość, powodując zakończenie jednego rozkazu po każdym cyklu zegara. Można wyróżnić cztery etapy:

- *pobranie rozkazu (ang. instruction fetch)*
- *dekodowanie rozkazu (ang. instruction decode)* – w procesorach klasy CISC, dekodowanie rozkazu jest bardziej złożone:
  - *właściwe dekodowanie rozkazu* – określenie kodu operacji i specyfikatorów argumentów
  - *obliczanie argumentów* – obliczanie adresów efektywnych argumentów
  - *pobranie argumentów*
- *wykonanie rozkazu (ang. execution)*
- *zapisanie wyników (ang. write-back)*

Cykl	Etapy Pobranie rozkazu	Dekodo-wanie rozkazu	Wykonanie rozkazu	Zapisanie wyników rozkazu
1	*Rozkaz 1*			
2	*Rozkaz 2*	*Rozkaz 1*		
3	*Rozkaz 3*	*Rozkaz 2*	*Rozkaz 1*	
4	*Rozkaz 4*	*Rozkaz 3*	*Rozkaz 2*	*Rozkaz 1*
5	*Rozkaz 5*	*Rozkaz 4*	*Rozkaz 3*	*Rozkaz 2*

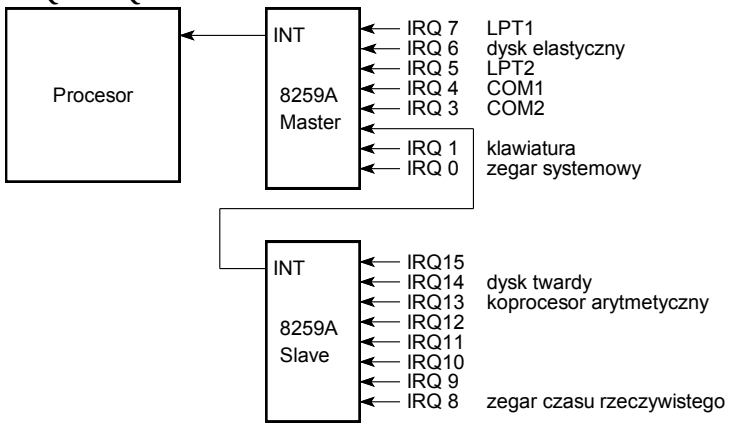
Czynniki powodujące zmniejszenie przyspieszenia (***hazardy***):

- realizacja niektórych etapów może powodować konflikty dostępu do pamięci;

- jeśli czasy trwania poszczególnych etapów mogą być niejednakowe, to na różnych etapach wystąpi pewne oczekiwanie;
- w programie występują skoki (rozgałęzienia) warunkowe, które mogą zmienić kolejność wykonywania instrukcji, a tym samym unieważnić kilka pobranych rozkazów — muszą one być usunięte z potoku, a potok musi być zapełniony nowym strumieniem rozkazów;
- wystąpienie przerwania sprzętowego lub wyjątku procesora stanowi zdarzenie nieprzewidywalne i również pogarsza przetwarzanie potokowe;
- niektóre rozkazy wymagają dodatkowych cykli (np. do ładowania danych);
- czasami rozkazy muszą oczekiwać z powodu zależności od nie zakończonych poprzednich rozkazów — system musi zawierać rozwiązania zapobiegające tego rodzaju konfliktom;

15. *Omówić podstawowe elementy systemu przerwań w PC ?*

System przerwań obsługiwany jest przez dwa układy 8259 (sekretarka procesora). Sygnały przerwań kierowane są z poszczególnych urządzeń do układów 8259 poprzez linie IRQ1-IRQ15



16. *Koncepcja kodowania programów bez użycia skoków*

Metoda predykatowa, wspomagana sprzętowo w najnowszych procesorach (np. 64bit itanium), polega na odejściu (lub znacznym ograniczeniu) skoków warunkowych w kodzie programu. Wykonuje się to przy pomocy manipulowania operacjami bitowymi (OR i AND) w taki sposób, aby jednoznacznie stwierdzić (tutaj dla przykładu), czy dana liczba jest większa od innej.

17. *W jakim celu zdefiniowano nieliczby Nan, w koprocesorze arytmetycznym*

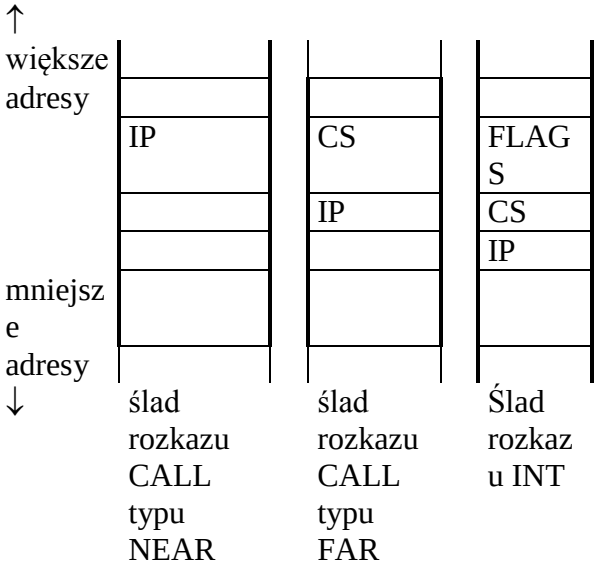
Dowolna wartość zmiennoprzecinkowa z polem wykładnika zawierającym same jedynki i mantysą różną od zera, jest traktowana jako nieliczba (NaN); używane są do reprezentacji specjalnych liczb, których arytmetyka definiowana jest przez oprogramowanie (każde wystąpienie takiej liczby generuje wyjątek); generowana są przy niedozwolonych operacjach, np. pierwiastkowaniu liczb ujemnych.

19. *Porównać działanie CALL i INT*

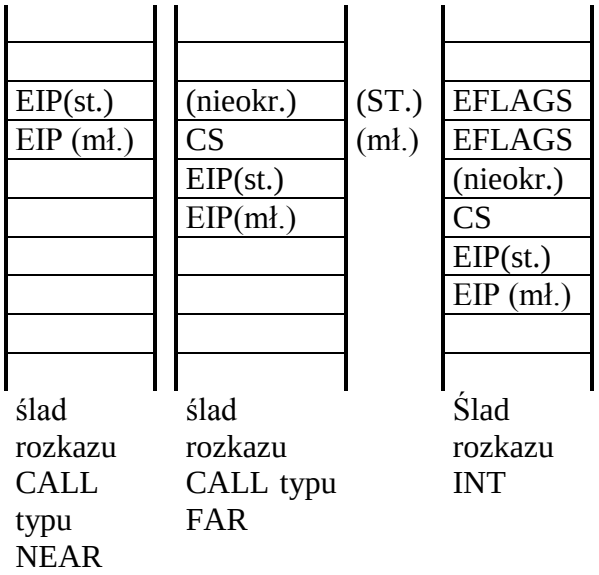
CALL – wywołuje procedurę i zostawia na stosie odpowiedni ślad, aby umożliwić powrót (przekazanie sterowania w odpowiednie miejsce) z tej procedury do programu głównego.

INT – wywołuje podprogram systemowy za pomocą tablicy przerwań, na stosie zostawia ślad

Ślad w trybie 16-bitowym



Ślad w trybie 32-bitowym

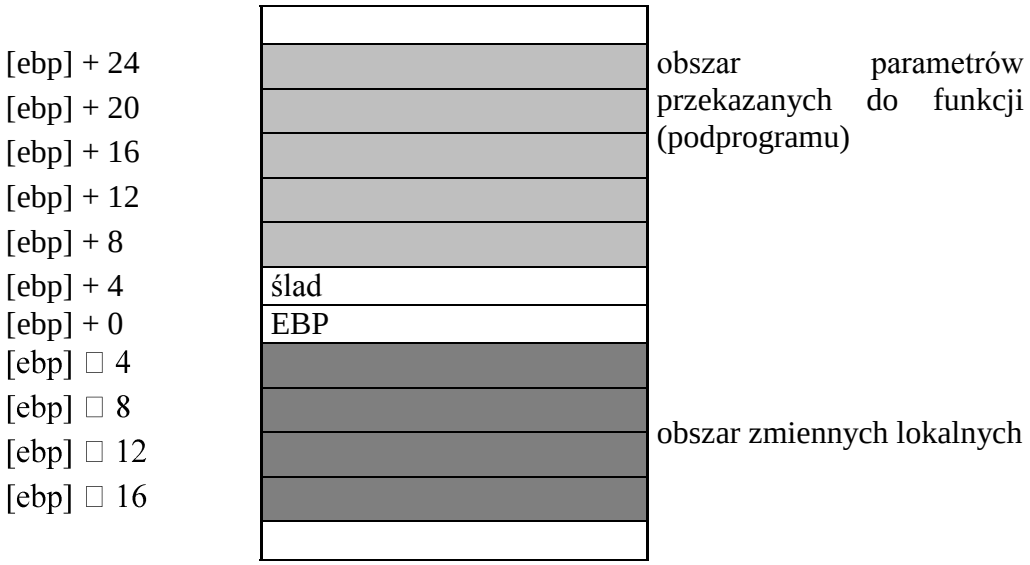


20. *W jakim celu kod instrukcji poprzedza się przedrostkiem chwilowej zmiany segmentu*

Procesory wykonują operacje adresowania pamięci korzystając z domyślnych rejestrów segmentowych (CS – wskazuje początek segmentu kodu zawierającego instrukcje , DS – dane, SS – stos), ale niekiedy dane umieszczone są w segmencie , którego początek wskazuje rejestr inny niż domyślny np. ES. W takim przypadku konieczne jest poinformowanie procesora, by adres fizyczny był dalej obliczany z użyciem innego rejestru segmentowego. W tym celu instrukcje poprzedza się dodatkowym bajtem, nazywanym przedrostkiem chwilowej zmiany segmentu. Np. mov bh, es:tablica[di] .

21. *Dostęp do zmiennych dynamicznych umieszczonych na stosie za pomocą modykacji, przy pomocy wskaźnika EBP*

Miejsce na zmienne dynamiczne rezerwuje się zazwyczaj na początku podprogramu. Standardowo robi się to przez skopiowanie ESP do EBP i przesunięcie wierzchołka stosu o potrzebną ilość bajtów na zmienne dynamiczne. W procedurze operuje się tymi zmiennymi odwołując do [ebp-x] (EBP wskazuje cały czas początek obszaru zmiennych dynamicznych, gdzieś pod wierzchołkiem stosu, x to odległość danej zmiennej w bajtach). Jednocześnie wygodnie można operować na zmiennych przekazanych do podprogramu przez stos, pamiętając o znajdującym się pod adresem [ebp+4] śladzie. Na zakończenie podprogramu trzeba zwolnić zmienne dynamiczne, przywracając ESP wartość EBP.



22. *Zasady działania pamięci statycznych i dynamicznych*

Statyczna: podstawowy element, to przerzutnik dwustanowy (min 6 tranzystorów). Ustawienie przerzutnika nie zależy od czasu i trwa aż do zmiany informacji. Wyłączenie zasilania powoduje utratę informacji. Przerzutnik musi znajdować się w stanie przewodzenia, lub nie (duża energiożerność). Cechy (SRAM): krótki czas dostępu, duża odporność na szumy i zakłócenia, prostota rozwiązania, nie wymagają dodatkowych operacji mających podtrzymywać informacje, mały stopień scalenia, wysokie koszty.

Dynamiczna: Zbudowana z tranzystora i mikrokondensatora. Tranzystor traktowany jest jako, włączający/wyłączający kondensator. Działanie pamięci oparte jest na magazynowaniu ładunku (nienaładowany = 0). Stan 1 jest nietrwały w czasie, w wyniku nieuniknionych upływności => konieczny proces odświeżania (z reguły co kilka ms). Cechy: duży stopień scalenia, mała odporność na szumy i zakłócenia, potrzebne operacje podtrzymujące informacje, małe koszty.

23. *Przyczyny utrudniajgce działanie przetwarzania potokowego w procesorach*

- potrzeba równoczesnego dostępu do tego samego obszaru pamięci przez różne podzespoły, powodująca zamrożenie potoku do czasu rozwiązania konfliku, co prowadzi do zwiększenia liczby cykli potrzebnych do realizacji rozkazu,
- zależności danych powodujące konieczność oczekiwania rozkazów na zakończenie działania rozkazu poprzedniego; w znacznej mierze mogą być wyeliminowane przez odpowiednie szeregowanie rozkazów:
    - odczyt po zapisie – odczyt danych może być zrealizowany dopiero w momencie zakończenia operacji zapisu danego rejestru
    - zapis po odczycie – zapis danych może być zrealizowany dopiero w momencie zakończenia czytania rejestru

    - zapis po zapisie – operacje zapisu muszą zostać wykonane w odpowiedniej kolejności
- rozkazy skoków warunkowych mogą zmienić kolejność wykonywania instrukcji, unieważniając rozkazy, które zostały już pobrane i rozpoczęło się ich przetwarzanie, a potok musi być zapełniony nowym strumieniem rozkazów. W celu uniknięcia takich sytuacji, wprowadza się predykcję skoków, dodatkowe bufory potoków lub programowanie bez użycia skoków.
- przerwania i wyjątki przerywają wykonanie programu, zmuszając procesor do wyczyszczenia potoku na takiej samej zasadzie jak skoki warunkowe.

24. *Główne różnice pomiędzy systemami MIMD z pamięcią wspólną i pamięcią rozproszoną*

Z pamięcią wspólną: procesory komunikują się poprzez sieć połączeń, czytając i zapisując dane zawarte w pamięci wspólnej. Prowadzi to do konfliktów dostępowych i w przypadku zastosowania magistrali z podziałem czasu (procesor sprawdza, czy magistrala jest wolna, jeśli nie to czeka) do nieświeżości danych w pamięciach podręcznych procesorów. Wymaga zastosowania specjalnych protokołów (np. MESI stosowany w procesorach Pentium), umożliwiających informowanie innych procesorów o zmianach w pamięci. Otrzymanie sygnału o zmianie w bloku pamięci powoduje w pamięciach podręcznych innych procesorów oznaczenie tego bloku jako „nieświeży".

Z pamięcią rozproszoną: Oferują liniowa skalowalność, każdy procesor ma pamięć lokalną, a procesory nie współdzielą zmiennych — zamiast tego procesory wymieniają dane, przesyłając między sobą komunikaty za pośrednictwem specyficznej sieci komunikacyjnej; każdy węzeł zawiera procesor, pamięć lokalną i kilka łączy komunikacyjnych do wymiany komunikatów; znaczne przyspieszenie można uzyskać poprzez wprowadzenie specjalnych procesorów przesyłających komunikaty

25. *Zasady wykonywania operacji arytmetycznych na liczbach wielokrotnej długości*

Dodawanie – realizowane przy użyciu instrukcji ADD lub ADC. ADD dodaje do siebie dwa argumenty, a wynik umieszcza w pierwszym z nich. Jeżeli wynik operacji nie mieści się w argumencie docelowym ustawiony zostaje znacznik przeniesienia CF. ADC służy do dodawania liczb wielobajtowych. Pozwala w trakcie dodawania uwzględnić przeniesienie powstałe w przypadku dodawania młodszych bajtów.

Odejmowanie – polega na dodaniu do pierwszego argumentu drugiego w postaci U2. SBB to odejmowanie dwóch liczb z pożyczką (odejmowanie bardziej znaczących słów liczby wielobajtowej). Ustawiane są te same znaczniki co w przypadku dodawania.

Mnożenie – przy korzystaniu z instrukcji mnożenia MUL (liczby bez z znaku) i IMUL (liczby ze znakiem) należy pamiętać, że wynik umieszczany jest w dwóch rejestrach. (algorytm mnożenia ze szkoły podstawowej, tzw. „słupki").

Dzielenie – instrukcje DIV i IDV należy pamiętać, że rozmiar dzielnej powinien być 2 razy większy, niż dzielnik (32-bit dzielimy przez 16-bit).

26. *Rola tablicy wektorów podczas wywoływania podprogramów systemowych*

W wielu systemach zastosowano wywoływanie podprogramów systemowych w sposób pośredni, za pośrednictwem tablicy adresowej, zawierającej adresy podprogramów systemowych; w takim przypadku programista nie podaje adresu podprogramu, ale indeks w tablicy adresowej; W kolejnych wersjach systemu adresy zawarte w tablicy ulegają zmianom, ale odwołania do konkretnych podprogramów realizowane są poprzez te same indeksy w tablicy adresowej.

27. *Jak postępuje się w przypadku, gdy w treści makrodefinicji zawierającej kod asemblerowy występuję etykieta*

Złożone makrodefinicje zawierają zazwyczaj instrukcje skoku do innych instrukcji, które poprzedzone są etykietami – jeśli makrodefinicja wywoływana jest więcej niż jeden raz, to w programie wystąpią wielokrotne definicje tych samych etykiet, co jest uważane za błąd formalny; dawniej zalecano by w takich przypadkach nie używać etykiet, a adresy instrukcji sterujących kodować jako adresy względne.

Podany problem rozwiązuje dyrektywa LOCAL – etykiety podane na liście tej dyrektywy są zastępowane, w trakcie rozwijania makro, przez unikatowe symbole, tzn. ta sama etykieta przy każdym rozwinięciu makro jest automatycznie zastępowana innym symbolem (??0000, ??0001, ??0002, ...); dyrektywa LOCAL musi poprzedzać inne wiersze makrodefinicji.

28. *Wartości specjalne w kontekście koprocesora*

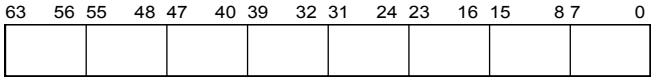
Są to wartości, które nie wynikają z definicji formatu zmiennoprzecinkowego, ale można je reprezentować w koprocesorze. Wszystkie liczby, których pole wykładnika zawiera same zera, lub same jedynki traktowane są jako wartości specjalne, na dodatek mogą być argumentami obliczeń dla zwykłych liczb. Są to:

- zero dodatnie i ujemne
- nieskończoność dodatnia i ujemna
- NaNy (wyniki niedozwolonych operacji)
- liczby zdenormalizowane, tzn. na tyle bliskie zeru, że wykładnik musiałby być ujemny; w takim przypadku rezygnuje się z warunku normalizacji mantysy i przyjmuje, że jej część całkowita wynosi 0

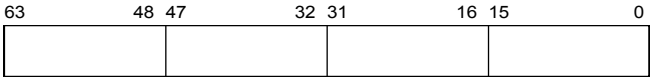
29. *Formaty danych stosowane przez rozkazy* **MMX** *i* **SSE**

Rozkazy grup **MMX** i **SSE** należą do instrukcji typu **SIMD (single instruction, multiple data)**, zostały zaprojektowane do szybkiego, równoległego operowania na kilku liczbach niewielkich rozmiarów. **MMX** operuje na danych całkowitoliczbowych, **SSE** na zmiennoprzecinkowych, obie grupy posiadają specjalne rejestry przeznaczone do swoich obliczeń (MMX 64-bitowe, SSE 128-bitowe). Obliczenia tego typu stosuje się głównie w operacjach multimedialnych. Do formatów danych MMX należą upakowany bajt (operacje na 8 liczbach jednobajtowych), upakowane słowo (4 liczby dwubajtowe), upakowane podwójne słowo (2 liczby czterobajtowe), poczwórne słowo (liczba ośmiobajtowa). Nie ma możliwości dostępu do pojedynczych danych w rejestrach. Formatem danych w SSE są 4 upakowane liczby zmiennoprzecinkowe 32-bitowe lub liczba skalarna (32-bitowa liczba zmiennoprzecinkowa na najmłodszych bitach rejestru SSE; działania takie są analogiczne do działań w koprocesorze).

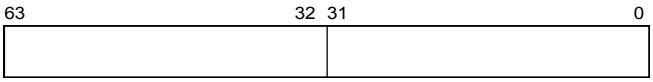
MMX: Upakowane: bajt, słowo, podwójne słowo i nieupakowane poczwórne słowo (w rejestrze 64-bit). Działa na liczbach całkowitych. Przechowuje dane wg. Sposoby „mniejsze niżej”.



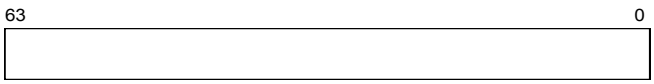
upakowany bajt (ang. packed byte)



upakowane słowo (ang. packed word)



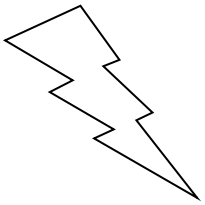
upakowane podwójne słowo (ang. packed double word)



poczwórne słowo (ang. quadword)

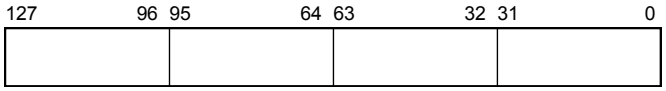
WAŻNE: *Instrukcje **MMX**, nie zmieniają rejestru znaczników. Prawie wszystkie instrukcje zaczynają się od przedrostka P (np.: PADD). Możliwe przyrost (ważne)i:*

S – arytmetyka nasycenia
U – liczby bez znaku
L – młodsza część
H – starsza część
B – bajt (8-bit)
W – słowo (16-bit)
D – podwójne słowo (32-bit)
Q – poczwórne słowo (64-bit)



Np.: **PADDUSB** – dodawanie bajtowo liczb bez znaku, z arytmetyką nasycenia

SSE: 8 rejestrów, 128-bit. W każdym cztery liczby zmiennoprzecinkowe 32-bit.



30. *Koncepcja pamięci podręcznej dwukanałowej*

Ety-kieta	**Dane**	Ety-kieta	**Dane**	
				00
				01
				10
				11

Przy takiej organizacji pamięci, numer linii w pamięci podręcznej wskazywany jest przez dwa ostatnie bity adresu w pamięci głównej; po wybraniu odpowiedniej linii trzeba jeszcze sprawdzić, czy najstarsze bity adresu są identyczne z polem jednej, albo drugiej etykiety — jeśli tak, to wystąpiło *trafienie.*

Podany schemat wymaga wprawdzie wykonania dwóch porównań zamiast jednego, ale pozwala na bardziej elastyczne wypełnianie pamięci podręcznej, co w rezultacie zmniejsza liczbę odwołań chybionych.

32. *Omówić typowe miary wydajności systemów komputerowych*

MIPS	liczba rozkazów wykonywana w ciągu sekundy, wyrażona w milionach
MFLOPS	liczba rozkazów zmiennoprzecinkowych wykonywana w ciągu sekundy, wyrażona w milionach
dhrystones/s	wartość uzyskiwana poprzez wykonanie charakterystycznego programu zawierającego rozkazy stałoprzecinkowe
whetstones/s	wartość uzyskiwana poprzez wykonanie charakterystycznego programu zawierającego rozkazy zmiennoprzecinkowe

33. *Zapisać znak C4H 98H w postaci 16-bitowego Unicode 110xxxxx 10xxxxxx*

C	4	H
1100	0100	b

9	8	H
1001	1000	b

110x xxxx 10xx xxxx
110**0 0100** (C4h) 10**01 1000** (98h)

Odp: 0000 0001 0001 1000 = 01 18 = U+0118

34. *Na podstawie przykładu wyjaśnić zastosowanie $ - licznika lokacji*

$ reprezentuje w wyrażeniach języka assembler bieżącą lokację wewnątrz aktualnie tłumaczonego segmentu, np.: jmp $+2 , powoduje przejście do następnego rozkazu. Tego typu rozkazy stosuje się czasami w celu wprowadzenia dodatkowego opóźnienia w programie. Kolejny przykład pokazuje, jak można wyznaczyć liczbę znaków łańcucha z użyciem symbolu $:

```
lancuch db 'Przykładowy łańcuch znaków.'
dlugosc = $ - lancuch
mov cx, długosc
```

Długość łańcucha obliczana jest w trakcie asemblacji programu. Tego rodzaju wyrażenia są nazywane *wyrażeniami arytmetycznymi czasu translacji* . Natomiast zmienna *długosc* jest *zmienną czasu translacji* , tzn. nie jest dla niej rezerwowany obszar pamięci w programie wynikowym.

35. *Opisać rozkazy POP EBP i POP [EBP]*

Przy POP EBP, wartość z wierzchołka stosu, jest skopiowana do rejestru EBP, a następnie dodane 4 bajty do rejestru ESP.

Przy POP [EBP], wartość z podanego adresu z wierzchołka stosu, jest skopiowana do rejestru EBP, a następnie dodane 4 bajty do rejestru ESP.

36. *Opisać komunikacje komputera z klawiaturą*

Klawiatura posiada własny sterownik – mikrokontroler klawiatury. Każdy przycisk na klawiaturze ma przypisany 8-bitowy *kod pozycji* generowany przez mikrokontroler jako *kod*

naciśnięcia, lub *kod zwolnienia* (poprzedzony bajtem F0H), który przesyłany jest szeregowo do układów na płycie głównej komputera. Komunikacja realizowana jest wg reguł dwukierunkowego synchronicznego protokołu szeregowego (istnieje możliwość przesłania danych zarówno z klawiatury do komputera jak i odwrotnie). Przesyłane informacje mają postać bajtów, które przesyłane są w postaci *ramek danych* (11 bitów). Sygnał przerwania klawiatury to **IRQ 1**.

37. *Zapisać w postaci zmiennoprzecinkowej double liczbę +1.0 (mantysa 52 bity)*

Liczba jest formatu 64-bitowego. Wykładnik ma 11 bitów.

Liczba = (**mantysa** + 1) * $2^{(\textbf{wykładnik} - 1023)}$ = (**0** + 1) * $2^{(\textbf{1023} - 1023)}$ = 1

S	wykładnik	mantysa
0	011 1111 1111	0 ... 0

Gdybyśmy mieli przedstawić w formacie float, sprawa wyglądałby podobnie (mantysa 23 bity).

Liczba jest formatu 32-bitowego. Wykładnik ma 8 bitów.

Liczba = (**mantysa** + 1) * $2^{(\textbf{wykładnik} - 127)}$ = (**0** + 1) * $2^{(\textbf{127} - 127)}$ = 1

S	wykładnik	mantysa
0	0111 1111	0 ... 0

38. *Wyjaśnić pojęcie skalowalności i napisać jaka, wg klasyfikacji Flynna architektura jest najlepsza pod tym względem i dlaczego*

Skalowalność jest to cecha systemów komputerowych, polegająca na zdolności do dalszej rozbudowy, ale także miniaturyzacji systemu. Najlepsze są systemy MIMD z pamięcią rozproszoną, bo oferują liniową skalowalność, a lepszej niż liniowej nie można osiągnąć.

39. *Napisać, ile ma blok pamięci (w bajtach), w pamięci podręcznej cztero-kanałowej, gdzie cała pamięć 512 KB z dołączonymi 12 liniami*

[pamięć] / [kanałowość] / $2^{[\text{linie}]}$ = wielkość bloku pamięci [B]

512KB / 4 / 2^{12} = 512KB / 2^{14} = 32B

40. *Jaką rolę w pracy komputera pełni system BIOS*

BIOS — Basic Input Output System — znajduje się w pamięci ROM komputera. Jest to mini system operacyjny, który uruchamiany jest bezpośrednio po uruchomieniu komputera. Odpowiada za wprowadzenie do pamięci właściwego systemu operacyjnego (bootstrapping). Ponadto odpowiada za inicjalizację, wstępne przetestowanie i konfigurację zainstalowanego sprzętu (programy testujące POST). Wstępne testowanie sprzętu przed uruchomieniem OS, zapobiega przed dalszymi powikłaniami, które mogą doprowadzić do uszkodzenia oprogramowania i danych.

Dla innych programów udostępniane są funkcje usługowe tworzące **BIOS API.** Oferuje on zestaw operacji, niezależny od konstrukcji konkretnego komputera („wyrównanie" różnic konstrukcji sprzętu). Jednak ponieważ BIOS pracuje w trybie 16-bitowym i nie jest

przystosowany do zaawansowanych metod obsługi sprzętu (np. wirtualizacja urządzeń), jest niechętnie wykorzystywany przez współczesne systemy operacyjne. Był natomiast szeroko wykorzystywany przez system DOS.

41. *Opisać technikę przydzielania pamięci i dostępu do zmiennych dynamicznych ulokowanych na stosie*

Zmienne lokalne (dynamiczne) – definiowane wewnątrz procedury potrzebne są tylko w trakcie jej wykonywania, zatem powinny zajmować obszar pamięci, tylko na czas wykonywania podprogramu. Dokonuje się tego odpowiednio zmniejszając wskaźnik stosu ESP. Dostęp do tych zmiennych realizowany jest za pomocą EBP, pomocniczego wskaźnika stosu [EBP – x] (x podzielne przez 4). Przydzielony obszar pamięci zwalniany jest poprzez przypisanie ESP <- -- EBP.

42. *Pewien znak reprezentowany jest w formacie UTF-8, przez dwa bajty C4H i 99H. Podać kod tego znaku w standardzie Unicode w postaci 16-bitowej. Wskazówka: znaki Unicode z przedziału 80H ÷ 7FFH kodowane są w UTF-8 w postaci 110xxxxx 10xxxxxx*

C4 99 = 110**0 0100** 10**01 1001**

Bierzemy bity zaznaczone pogrubioną czcionką i dopełniamy je z lewej zerami:

0000 0**001 0001 1001** = 01 19 = U+0119

43. *Wyjaśnić dlaczego w operacjach porównywania rozkaz CMP można ewentualnie zastąpić rozkazem SUB, chociaż może to być niewygodne*

Ponieważ obydwa rozkazy wykonują to samo działanie — odejmowanie. W obydwu przypadkach ustawiane są znaczniki CF, OF, ZF, SF i PF, dlatego można rozkaz CMP zastąpić rozkazem SUB. Różnica polega na tym, że w przypadku SUB wynik operacji zapisywany jest do pierwszego operandu, co w przypadku porównania rzadko jest pożądane.

44. *Opisać format kodowania rozkazów dla procesorów o architekturze VLIW*

Długość słowa dla tej architektury wynosi od 256 do 1024 bitów — każde pole używane jest do kodowania operacji dla konkretnej jednostki wykonującej. Używa się statycznych technik szeregowania (skąd brak skalowalności), więc potencjalnie są dużo szybsze niż procesory superskalarne.
Rozkazy szereguje się na etapie kompilacji przyporządkowując je do odpowiednich modułów wykonawczych, np.:

ALU1	ALU2	ALU3	MOV	FPU

Wady: powolność i duży koszt kompilatorów; kod maszynowy nie może być przenoszony na procesory o podobnej architekturze, ale innej liczbie jednostek wykonujących.

45. *Po co używać PTR*

PTR jest operatorem używanym, gdy wyrażenie adresowe nie pozwala na jednoznaczne przetłumaczenie rozkazu. Dzięki niemu unikamy *odwołań anonimowych* np.

mov [eax], 5, gdzie może być problem z określeniem rozmiaru operandu, ponieważ nie określa ilu bajtowa jest liczba 5. Dlatego informujemy o rozmiarze operandu, poprzez podanie 'byte' (8 bitów), 'word' (16 bitów), 'dword' (32 bity), 'fword' (48 bitów), 'qword' (64 bity), 'tword' (80 bitów). Przykładowa konstrukcja, np: **mov dword PTR [eax], 5** . Stosuje się go również przy określaniu adresów rozkazów sterujących (skoków).

46. *Czemu EBP jest lepszy od ESP*

ESP – wskaźnik stosu, EBP – pomocniczy wskaźnik stosu. Otóż zawartość ESP, może się zmieniać w trakcie wykonywania programu, wrzucając coś na stos, lub zdejmując. W tej sytuacji, byłoby problematycznym używać argumentów przekazanych przez stos do np.: podprogramu, ponieważ zmieniałoby się ich położenie na stosie względem rejestru ESP. Dlatego opłaca się przekopiować tuż po rozpoczęciu podprogramu, zawartość rejestru ESP do EBP, dzięki temu mamy stałe położenie przekazanych argumentów względem EBP (np.: **[ebp + x]**, gdzie x wartość dodatnia podzielna przez 4, wskazuje argumenty, a **[ebp – x]** położenie zmiennych lokalnych na stosie).

47. *Jak zatrzymać przyjmowanie przerwań*

Jeśli przerwania mogą być blokowane poprzez wyzerowanie znacznika **IF** (komendą **CLI**), zaliczane są do klasy *przerwań maskowalnych*. Procesor Pentium może też przyjmować *przerwania niemaskowalne*, które nie mogą być blokowane. *Przerwania niemaskowalne* (ang. **NMI — non-maskable interrupt**) stosuje do sygnalizacji zdarzeń wymagających natychmiastowej obsługi niezależnie od stanu systemu. W komputerach PC *przerwanie niemaskowalne*, generowane jest w przypadku zidentyfikowania błędu pamięci RAM.

.

48. *Jak rejestrowany jest nadmiar w dodawaniu, odejmowaniu, mnożeniu i dzieleniu w liczbach stałoprzecinkowych ze znakiem i bez*

W przypadku dodawania i odejmowania ustawiana jest jedna z flag procesora – w przypadku operacji na liczbach ze znakiem (w kodzie U2) jest to **overflow flag (OF)**, w przypadku liczby bez znaku **carry flag (CF)**. Przy mnożeniu liczb całkowitych nadmiar nie powstaje (bo wynik zapisywany jest w dwóch rejestrach), a przy dzieleniu zgłaszany jest wyjątek procesora. Nadmiar sygnalizowany jest wpisaniem 1 w odpowiednią flagę.

49. *Załóżmy, że pętla wykona się 16 razy. Jaka jest dokładność przewidywania dla tej pętli, zakładając mechanizm predykcji oparty o bit historii*

Jeden bit historii mówi czy skok się wykonał, czy nie. Jeżeli skos się wykonał to na podstawie predykcji wiemy, że wykona się ponownie. Zatem przekłamanie mogło nastąpić jedynie przy pierwszym i ostatnim skoku, na pierwszym ponieważ bit historii wynosił 0 (wcześniej nie sprawdzał tego warunku, wartość domyślna), na ostatnim ponieważ bit historii wynosi 1 (zakłada, że się wykona, a jest zakończenie pętli).
(16-2) / 16 = 14 / 16 = 87,5%

50. *Wstaw wartość tak, aby dokonał się skok, jeśli w EDI jest 0 lub 16*
TEST EDI, ….
JZ skok

TEST EDI, 0FFFFFFEFH ; 1111 1111 1111 1111 1111 1111 1110 1111

;czyli zero na 5 miejscu - waga = 16

Inny przykład (w edx, 0 lub 1)

TEST EDX, 0FFFFFFFEH ; 1111 1111 1111 1111 1111 1111 1111 1110

51. *Czemu EPIC, jest tak rzadko używany*

Jest to nowa architektura, obejmująca procesy wielordzeniowe. W technice EPIC za równoległe przetwarzanie instrukcji i optymalne wykorzystanie jednostek wykonawczych, np. modułów arytmetyczno-logicznych, odpowiada kompilator programu. Jest to odwrócenie sposobu działania klasycznych, 32-bitowych procesorów, takich jak Pentium 4 czy Athlon XP, gdzie funkcje te realizuje wewnętrzna logika układu. Zbyt duże skomplikowanie kompilatorów, które umożliwiają realizację tej architektury - stad ich mała popularność.

52. *Obliczyć przyspieszenie*

Dane:
cykl - 10ns
ALU 30% 2 cykle
PAM 40% 4 cykle
sterujące 30% 6 cykli
narzut na potok: 3ns

(<cykl> * (<cykle_ALU> * <ALU> + <cykle_PAM> * <PAM> + <cykle_sterujące> * <sterujące>) / (<cykl> + <cykl_narzut_na_pokot>) = przyspieszenie

[10ns * (2*0,3 + 4*0,4 + 6*0,3)] / [10ns + 3ns] = 40ns / 13 ns = 307%

53. *Dzielenie przez zero w procesorze i koprocesorze*

KOPROCESOR:
Za dzielenie przez zero jest odpowiedzialny znacznik ZE w rejestrze stanu koprocesora. Wyjątek dzielenie przez zero powstaje gdy występuje dzielenie liczby różnej od zera przez zero. W przypadku zamaskowania tego wyjątku (ZM = 1) wynikiem operacji jest wartość wyjątkowa "nieskończoność": pole wykładnika zawiera same jedynki, pole mantysy zawiera same 0.
W podanych tu rozważaniach termin nieskończoność (□) należy rozumieć jako wartość specjalną używaną przez koprocesor arytmetyczny; w szczególności dla pewnej liczby x>0 zachodzi (x nie jest □)

$\frac{x}{+0} = +\infty \quad \frac{-x}{+0} = -\infty \quad \frac{x}{-0} = -\infty \quad \frac{-x}{-0} = +\infty$

PROCESOR:
Dzielenie przez zero powoduje powstanie nadmiaru. Nadmiar przy dzieleniu powoduje, wyjątek procesora, co zazwyczaj powoduje natychmiastowe zakończenie wykonywania programu.

54. *Jakie adresowanie użyć, jeśli adres danej ustalony jest podczas wykonywania programu*

Adresowanie za pomocą modyfikacji adresowych, np. mov ecx, **tablica[ebx][eax*4+12]**

55. *Dlaczego niektóre wyjątki koprocesora są zazwyczaj maskowane w typowych programach generowanych przez kompilatory języków programowania*

Obliczane wyniki pośrednie z nadmiarem bądź niedomiarem w złożonych obliczeniach często mają niewielki wpływ na wynik końcowy. Powstałe w wyniku wyjątków wartości specjalne mogą być argumentami do dalszych obliczeń a wynik działania na wartości specjalnej może być **poprawny,** albo ponownie wartością specjalną. Takie „łagodne wyjątki" to: *dzielenie przez zero, niedomiar, niedokładny wynik (Np: 1/3)*.

56. *Podać przykład rozkazu, w którym adres efektywny jest mniejszy od zawartości pola adresowego*

mov ecx, 20000
mov bh, byte PTR [0FFFFFD01h + ecx]

Nastąpi tu przewinięcie (bo w ecx mamy większą wartość niż ta, którą możemy dodać), dlatego adres efektywny będzie mniejszy od zawartości pola adresowego.

57. *Porównanie modelu von Neumanna z architekturą harwardzką, oraz zastosowania architektury harwardzkiej w procesorach typu Pentium 4 Core 2 Duo*

System komputerowy von Neumanna nie posiada oddzielnych pamięci do przechowywania danych i instrukcji. Instrukcje jak i dane są zakodowane w postaci liczb. Bez analizy programu trudno jest określić czy dany obszar pamięci zawiera dane czy instrukcje. Wykonywany program może się sam modyfikować traktując obszar instrukcji jako dane, a po przetworzeniu tych instrukcji - danych - zacząć je wykonywać.

Architektura harwardzka - rodzaj architektury komputera. W odróżnieniu od architektury von Neumanna, pamięć danych programu jest oddzielona od pamięci rozkazów.
Podstawowa architektura komputerów zerowej generacji i początkowa komputerów pierwszej generacji.

Prostsza (w stosunku do architektury von Neumanna) budowa przekłada się na większą szybkość działania - dlatego ten typ architektury jest często wykorzystywany w procesorach sygnałowych oraz przy dostępie procesora do pamięci cache.
Separacja pamięci danych od pamięci rozkazów sprawia, że architektura harwardzka jest obecnie powszechnie stosowana w mikrokomputerach jednoukładowych, w których dane programu są najczęściej zapisane w nieulotnej pamięci ROM (EPROM/EEPROM), natomiast dla danych tymczasowych wykorzystana jest pamięć RAM (wewnętrzna lub zewnętrzna).

58. *Słownik symboli, do czego służy*

- asemblacja realizowana jest dwuprzebiegowo: w każdym przebiegu czytany jest cały plik źródłowy (ściśle: moduł) od początku do końca;
- w pierwszym przebiegu asembler stara się wyznaczyć ilości bajtów zajmowane przez poszczególne rozkazy i dane; jednocześnie asembler rejestruje w *słowniku symboli* wszystkie pojawiające się definicje symboli (zmiennych i etykiet);
- w drugim przebiegu asembler tworzy kompletną wersję przetłumaczonego programu określając adresy wszystkich rozkazów w oparciu o informacje zawarte w słowniku symboli;
- jeśli asembler napotka wiersz zawierający definicję symbolu, to rejestruje go w *słowniku symboli*, jednocześnie przypisując temu symbolowi wartość równą aktualnej zawartości licznika lokacji; ponadto zapisywane są także atrybuty symbolu, jak np. far, byte, itp;

59. *Jak wpisać liczbę, do wskaźnika instrukcji*

CALL – wywołuje procedurę i zostawia na stosie odpowiedni ślad, aby umożliwić powrót (przekazanie sterowania w odpowiednie miejsce) z tej procedury do programu głównego.
INT – wywołuje podprogram systemowy za pomocą tablicy przerwań, na stosie zostawia ślad
RET – zdejmuje z wierzchołka stosu ślad zostawiony przez CALL
IRET– zdejmuje z wierzchołka stosu ślad zostawiony przez INT

Ślad zapisany na stosie wskazuje miejsce w programie, dokąd należy przekazać sterowanie po wykonaniu podprogramu; innymi słowy: zawartość wierzchołka stosu powinna zostać przepisana do rejestru EIP — czynności te realizuje rozkaz RET; jeśli podprogram wywoływany jest za pomocą rozkazu INT (rozkaz omawiany dalej), to stosuje się rozkaz IRET;

Jeżeli chcemy zmodyfikować EIP to wywołujemy wcześniej przygotowana procedurę za pomocą instrukcji call. W tej procedurze, adres stosu odkładany jest do rejestru ebp, modyfikuje odpowiedni adres stosu (standardowo [ebp+4]) wstawiając tam nowa wartość EIP, ponieważ podczas wywoływania instrukcji call odkładany jest tam adres następnej instrukcji do wywołania. Po wyjściu z procedury za pomocą instrukcji ret, program rozpocznie działanie w miejscu wskazanym przez adres, który zmodyfikowaliśmy.

60. *Do jakiego rodzaju systemów (klasyfikacja Flynna) należy klaster. Wady i zalety*

Klasyfikujemy jako MIMD (**M**ultiple **I**nstruction, **M**ultiple **D**ata) - równolegle wykonywanych jest wiele programów, z których każdy przetwarza własne strumienie danych - przykładem mogą być komputery wieloprocesorowe, a także klastry i gridy.

Zalety:
- powiększenie mocy obliczeniowej
- zwiększona niezawodność - zespół komputerów dublujących nawzajem swoje funkcje. W razie awarii jednego z węzłów, następuje automatyczne przejęcie jego funkcji przez inne węzły.
- Sprzęt jest dostępny od wielu dystrybutorów, co zapewnia niskie ceny i łatwą konserwację
- Brak zależności od jednego dostawcy
- Duża wydajność i duża przepustowość.
- Duża niezależność i dyspozycyjność – Przy wielu połączonych maszynach, strata jednej nie wpłynie zbytnio na działanie klastra. Najważniejsze komponenty systemu można powielać na wielu maszynach, tak, aby system mógł działać po awarii którejś z maszyn.
- Ekonomia – Klastry mają największy stosunek wydajności do ceny.
- Duża rozszerzalność i skalowalność – Kiedy wzrasta zapotrzebowanie na moc obliczeniową, dodatkowa stacja może być dołączona do sieci, a oprogramowanie „skalowane" lub skopiowane z innego komputera.
- Technologia – Dzisiejsze Klastry mogą być zbudowane z tysięcy maszyn i osiągają prędkości niedostępną dla systemów zcentralizowanych.

Wady:
- Ograniczona ilość oprogramowania – Obecnie istnieje niewielka ilość oprogramowania wykorzystująca Klastry, a pisanie takich aplikacji od podstaw może być kosztowne i pracochłonne.
- Sieć łącząca stacje może zawodzić – Sieć może być przeciążona, gubić komunikaty lub może mieć za mała przepustowość aby efektywnie obsłużyć odpowiednią ilość maszyn.
- Problemy bezpieczeństwa – W przypadku **klastrów** zwiększa się możliwość dostępu do systemu, a także ataku osób trzecich lub włamania do systemu.
- Koszty administracji,
- Połączenia za pomocą I/O, a nie szyny pamięci,
- Klaster potrzebuje N niezależnych pamięci i N kopii OS

61. *Rozkaz negujący bity parzyste w EBX, a nieparzyste bez zmian*

xor ebx, AAAAAAAAh ;1010 1010 1010 1010 1010 1010 1010 1010

62. *Podać podstawowe zasady tworzenia procedur obsługi przerwań sprzętowych*

Tworząc własne procedury obsługi przerwań, należy zapamiętać poprzednią wartość wektora przerwań, zablokować nadchodzenie przerwań rozkazem **CLI**, zainstalować nową procedurę wpisując jej adres do wektora przerwań, odblokować nadchodzenie przerwań rozkazem **STI**. Po deinstalacji należy przywrócić poprzednią zawartość wektora przerwań. Procedura obsługi przerwania powinna wywoływać także procedurę domyślną, jeśli nie spełnia jej funkcji. Zazwyczaj procedury takie działają przy wyzerowanym znaczniku **IF**, aby zapobiec przerywaniu ich innymi przerwaniami.

63. *Do czego służy rozkaz NOP*

NOP - no operation - rozkaz pusty. W przypadku skoku do przodu, położenie etykiety nie jest znane – assembler nie jest w stanie określić, jaki skos będzie wykonany (dwu- czy sześciobajtowy), zatem zakłada wykonanie 6-bajtowego skoku. Jeżeli podczas asemblacji okaże się, że można zastosować skok 2-bajtowy, to do niezajętych bajtów wpisywane są rozkazy **NOP (90H)**.

64. *Porównać UTF-8 i Unicode ?*

Znaki o kodach U+0000 do U+007F kodowane są jako pojedynczy bajt 0x00 – 0x7F. Znaki o kodach większych niż U+007F kodowane są jako sekwencja kilku bajtów, z których każdy ma ustawiony najstarszy bajt na 1. Pierwszy bajt jest liczbą z przedziału 0xC0 – 0xFD i określa ile bajtów występuje po nim. Pozostałe zawierają liczby z przedziały 0x80 – 0xBF.

65. *Opisać składniki z których zbudowany jest procesor superskalarny (np Pentium)*

Procesory superskalarne (np. Pentium) umożliwiają równoległe wykonywanie wielu rozkazów — wymaga to umieszczenia wielu modułów wykonawczych. Procesor Pentium ma dwa moduły wykonujące działania na liczbach całkowitych, oznaczone U i V, przy czym kanał V może wykonywać tylko tzw. proste instrukcje (m.in. ADD, SUB, CMP, INC, AND, OR, JMP, skoki warunkowe).

W sytuacji, gdy dwie proste instrukcje występują zaraz po sobie w kolejce rozkazów, i spełnione są pewne dodatkowe założenia (np. nie może być zależności odczyt po zapisie), procesor dobiera instrukcje w pary i wykonuje równocześnie; dobieranie w pary nazywane jest skalowaniem, a architektura takiego procesora nazywana jest architekturą skalowalną (superskalarną);

Procesor PowerPC 620 zawiera 6 niezależnych jednostek wykonawczych: jednostkę rozkazów, trzy jednostki całkowitoliczbowe, jednostkę ładowania/zapisu, jednostkę zmiennopozycyjną. Procesor zawiera układy przewidywania rozgałęzień, bufory przemianowania rejestrów oraz stacje rezerwowe wewnątrz jednostek wykonawczych. Przewidywanie skoków oparte jest na tablicy historii zawierającej 2048 zapisów. Skuteczność przewidywań: 90%.

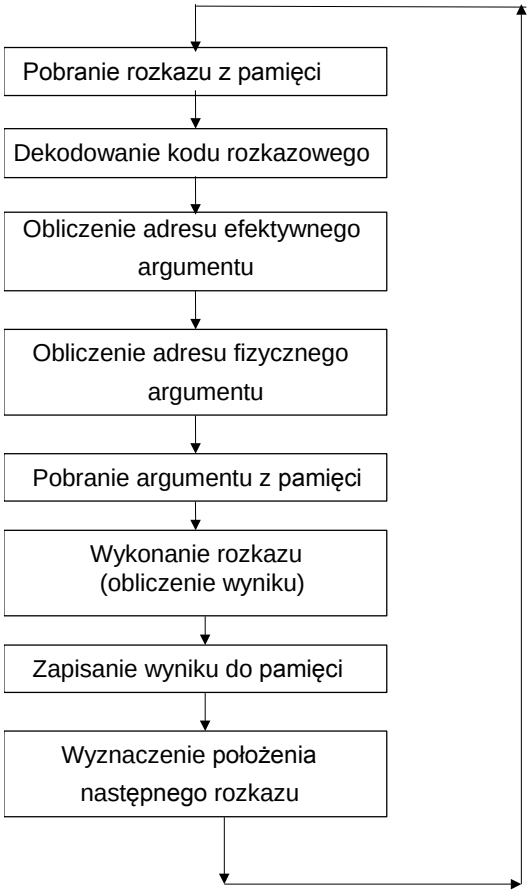
66. *Wyjaśnić zadania układu APIC*

Układ obsługujący system przerwań, umożliwiający dołączenie 24 linii przerwań. Jest swego rodzaju „sekretarką" procesora. Rejestruje nadchodzące sygnały przerwań i wysyła je do jednostki lokalnej (magistralą ICC), która odbiera i zgłasza przerwanie do procesora. Układ APID wspomaga także pracę wieloprocesorową.

A. Podstawowe zasady działania komputera

67. *Omówić podstawowe zasady wykonywania programu przez procesor.*

Procesor wykonuje instrukcje programu po kolei, zgodnie z ustalonym schematem, zwanym cyklem rozkazowym. Na cykl rozkazowy składa się kolejno:



Współczesne procesory stosują technikę zwaną przetwarzaniem potokowym, polegającą na jednoczesnym wykonywaniu kolejnych etapów cyklu dla sąsiednich rozkazów przez różne bloki funkcjonalne, każdy z rozkazów jest o jeden krok wykonania dalej niż następny. Umożliwia to przyspieszenie pracy procesora.

68. *Porównać własności różnych rodzajów pamięci stosowanych w komputerach.*

- **pamięci SRAM (static RAM)**: droga, mały stopień scalenia, wydziela dużo mocy, za to jest prosta, szybka, odporna na zakłócenia, nie trzeba jej odświeżać, używana w pamięci podręcznej procesora (poziomy L1, L2)
- **pamięci DRAM (dynamic RAM)**: tańsze, większy stopień scalenia, nie wydzielają wiele mocy, za to wolniejsze, mniej odporne na zakłócenia, wymagają odświeżania zawartości (bo zbudowana z kondensatorów), używana w pamięci operacyjnej (poziom L3); występuje w różnych rodzajach, np. FPM RAM (fast page mode) przyspieszający uzyskiwanie danych z jednego wiersza (wymaga tylko jednego sygnału RAS), EDO DRAM - jak poprzednie, z ulepszoną synchronizacją zegarową, SDRAM z synchronicznym sterowaniem impulsami zegarowymi (odświeża pamięć wykorzystując

moment, gdy nie jest używana), DDR - SDRAM wykorzystujący oba zbocza sygnału zegarowego
- ponadto ROM-y (pamięci stałe read only), wśród nich MROM (dane zaprogramowane przy produkcji, niezmienialne), PROM (jednokrotnie programowalne), EPROM, EEPROM, Flash (kasowalne impulsami świetlnymi bądź elektrycznymi), NVRAM (SRAM z możliwością zapisywania części danych na EEPROM w celu zachowania po wyłączeniu zasilania)
- najtańsze pamięci dyskowe, wolne ale tanie, zachowujące dane po wyłączeniu zasilania

69. *Omówić podstawowe tryby pracy procesorów zgodnych z architekturą IA–32*

- **tryb rzeczywisty** – występuje segmentacja pamieci, segmenty stałego rozmiaru 64KB, przestrzeń adresowa ograniczona do 1MB, w praktyce do okolo 640KB - jest to tryb domyślny, w tym trybie procesor pracuje po włączeniu komputera; adresowanie liniowe poprzez parę segment:offset, adres fizyczny obliczany wg wzoru: segment*16+offset
- **tryb chroniony** - tryb inicjalizowany i w znacznej mierze kontrolowany przez system operacyjny, pamięć może być zorganizowana w segmenty dowolnej wielkości, adresowanie zależy od systemu operacyjnego, może być model płaski (bez segmentacji), z segmentacją jak w trybie rzeczywistym, lub (zazwyczaj) adresowanie nieliniowe (tzw. logiczne) za pomocą pary selektor:offset, adres fizyczny jest zależny od wpisu w globalnej/lokalnej tablicy deskryptorów, na który wskazuje selektor; w tym trybie procesor wspiera wielozadaniowość, chroni przed nieupoważnionym dostępem do urządzeń wejścia/wyjścia; fizyczna przestrzeń adresowa ograniczona do 64 GB, liniowa przestrzeń adresowa do 4 GB; posiada także odmianę nazywaną „virtual86”, który jest symulacją trybu rzeczywistego w trybie chronionym, służy np. do uruchamiania programów MS-DOS
- **tryb SMM (System Management Mode)** - jest to tryb specjalnie stworzony dla systemów operacyjnych, by ułatwić zarządzanie sprzętem Tryb ten jest podobny to trybu rzeczywistego, w trybie tym można zaadresować max 4GB pamięci, nie ma w nim poziomów uprzywilejowania (w przeciwieństwie do trybu chronionego), ani mapowania adresów, możliwe jest wykonywanie dowolnych instrukcji wejścia/wyjścia, oraz wszystkich instrukcji systemowych

70. *W jaki sposób zmienia się zawartość wskaźnika instrukcji EIP w procesorach klasy IA–32 w trakcie wykonywania różnych typów rozkazów.*

Przy instrukcjach zwykłych (niesterujących) powiększa się wartość rejestru EIP o ilość bajtów aktualnie wykonywanego rozkazu, tak żeby wskazywał adres następnej instrukcji. Przy instrukcjach sterujących (skokach, call, ret itp.), jeśli wykonywany będzie skok, EIP zwiększa się dodatkowo o zawartość pola przesunięcia rozkazu. Przesunięcie może być ujemne, wtedy jego wartość zakodowana jest zgodnie z kodem U2 (w praktyce procesor dodaje wartość przesunięcia i stosuje arytmetykę przewinięcia, co daje ten sam efekt).

71. *Scharakteryzować grupę instrukcji procesora określanych jako operacje bitowe.*

- instrukcje operacji logicznych **AND, OR, XOR, NOT** - kolejno suma, iloczyn, suma modulo 2, negacja bitowa – instrukcje te wynik zapisują w pierwszym operandzie, ustawiają flagi procesora
- **TEST** – instrukcja identyczna z **AND**, ale tylko ustawia znaczniki (nie zapisuje nigdzie wyniku)
- przesunięcia logiczne **SHL, SHR** – odpowiadają mnożeniu/dzieleniu przez 2
- przesunięcia arytmetyczne **SAL, SAR** - przesunięcie w lewo działa identycznie jak **SHL**, przesunięcie w prawo powiela najstarszy bit, np. 1010 po przesunięciu będzie miało

wartość 1101

- obroty **ROL, ROR** – przy obrocie w lewo najstarszy bit trafia na miejsce najmłodszego i dodatkowo jest kopiowany do **carry flag (CF)**; w prawo odwrotnie
- obroty przez carry **RCL, RCR** – przy obrocie w lewo najstarszy bit trafia tylko do **CF**, a poprzednia zawartość CF w miejsce najmłodszego; w prawo odwrotnie
- ustawianie, zerowanie, negowanie pojedynczych bitów: **BTS, BTR, BTC**
- przeszukiwanie bitowe (bit search) od frontu **BSF** lub od końca (rear) **BSR**

Operacje bitowe są wydajną i efektywną metodą na uproszczenie wielu obliczeń. Stosuje się je m.in. do obliczeń na liczbach wielokrotnej długości, szybkiego porównywania, zerowania itd.

B. Kodowanie danych i instrukcji

72. *Omówić różne rodzaje kodowania liczb binarnych w komputerze.*

- liczby całkowite bez znaku - kolejne bity (od prawej do lewej) odpowiadają kolejnym potęgom dwójki
- liczby całkowite ze znakiem – najstarszy bit jest bitem znaku, ma wartość 1 dla liczb ujemnych; wartości zgodne są z kodem U2, moduł liczby uzyskuje się dodając 1 i negując wszystkie bity
- liczby BCD – binarny kod dziesiętny, w którym każdej cyfrze dziesiętnej odpowiadają 4 bity; stosuje się odmianę upakowaną (2 cyfry/bajt) i nieupakowaną (1 cyfra na bajt, pozostałe 4 bity zerowe)
- liczby zmiennoprzecinkowe w normalizowanym formacie z wykładnikiem i mantysą; pierwszy bit jest bitem znaku, następnie znajduje się wykładnik przesunięty o pewną wartość tak, aby zawsze był dodatni, następnie mantysa z przedziału [1;2), z reguły części całkowitej mantysy nie zapisuje się;

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

W little endian mniej znaczące bajty liczb przechowywane są w niższych adresach, w big endian mniej znaczące bajty przechowywane są w wyższych adresach. W little endian liczba FFEEDDCCH (załóżmy, że adresy rosną do prawej), to CC DD EE FF; w big endian naturalnie, tzn. FF EE DD CC.

74. *Wyznaczyć wartość dziesiętną 32-bitowej liczby zmiennoprzecinkowej 0100 0000 1111 1000 0000 0000 0000 0000*

Bit znaku = 0
Wykładnik = 100 0000 1 = 129
Mantysa = 111 1000 0000 0000 0000 0000 = .5 + .25 + .125 + .0625 = .9375
Uwzględniając przesunięcie wykładnika o 127 i niejawną jedynkę w mantysie, daje to 1.9375 * 2^2 = 7.75

75. *Jakie rodzaje zaokrąglenia stosuje się w koprocesorze arytmetycznym*

- zaokrąglenie w kierunku liczby najbliższej [**RC = 00**]
- zaokrąglenie w kierunku zera (obcięcie dodatkowych bitów) [**RC = 11**]
- zaokrąglenie w górę (do +niesk) [**RC = 10**]
- zaokrąglenie w dół (do -niesk) [**RC = 01**]

O rodzaju zaokrąglenia decyduje wartość bitów RC w rejestrze sterującym koprocesora

76. *Czym różnią się rozkazy koprocesora arytmetycznego:* **FLD** *i* **FST**

FLD ładuje liczbę z pamięci na wierzchołek stosu koprocesora lub kopiuje liczbę z głębi stosu na jego wierzchołek;

FST zrzuca (kopiuje) liczbę z wierzchołka stosu koprocesora do pamięci lub wgłąb stosu.

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

Dowolna wartość zmiennoprzecinkowa z polem wykładnika zawierającym same jedynki i mantysą różną od zera jest traktowana jako ***nieliczba (NaN)***. Wartości takie wprowadzono w celu reprezentowania wyników niedozwolonych operacji, rozróżniając do tego ***QNaN*** (*quiet*, niesygnalizujące wyjątku) od ***SNaN*** (*signalling*, generujące wyjątki koprocesora).

Arytmetyka takich liczb definiowana może być przez oprogramowanie (każde wystąpienie takiej liczby generuje wyjątek).

78. *Omówić zasady wykonywania operacji arytmetycznych na liczbach wielokrotnej długości*

W przypadku dodawania i odejmowania stosuje się algorytm podobny do działań „w słupkach" uwzględniając pożyczki i przeniesienia, znajdujące się w **carry flag (CF)**. Służą do tego rozkazy **ADC** i **SBB**. W przypadku mnożenia można rozłożyć liczbę na sumę potęg dwójki, obliczyć potęgi za pomocą przesunięć (**SHL**) i obrotów bitowych przez **carry flag** (rozkaz **SHL**) i następnie sumując obliczone elementy. W dzieleniu zastosowanie ma traktowanie reszty z dzielenia starszej części liczby jako najstarsze bity części młodszej.

79. *Omówić podstawowe zasady kodowania rozkazów procesora*

Rozkazy koduje się w postaci ciągów zerojedynkowych. W architekturze IA-32 rozkazy są zmiennej długości, od 1 do 7 bajtów. Kod powinien zawierać rozkaz, położenie w pamięci argumentów lub same argumenty, miejsce docelowe wyniku, w niektórych architekturach ponadto stosuje się adres następnej instrukcji. W celu ograniczenia długości rozkazu w IA-32 przyjęto, że adres następnej instrukcji zawiera rejestr EIP, miejsce docelowe równe jest zazwyczaj położeniu pierwszego argumentu i tylko jeden z argumentów może odwoływać się do pamięci. Standardowo, rozkazy przyjmują następujący format:

- 1 bajt: 6 bitów kodu operacji; bit d (wskazuje, w którym operandzie zapisać wynik, jeśli d=1 zapisuje w operandzie A, jeśli 0 to w B); bit w (rozmiar danych, w=0 dla operacji na danych 8-bitowych, w=1 dla 16/32-bitowych)
- 2 bajt: 2-bitowe pole mod (podaje liczbę bajtów pola przesunięcie - 00 jeśli brak przesunięcia, 01 jeśli 1 bajt przesunięcia, 10 jeśli 4 bajty, 11 jeśli rozkaz w ogóle nie korzysta z pamięci), 3-bitowe pole reg (operand A, zazwyczaj kod rejestru), 3-bitowe pole r/m (operand B, może być kod rejestru jeśli mod=11, lub kod modyfikacji adresowej)
- 3 bajt (tylko w przypadku modyfikacji adresowej r/m = 100) - bajt SIB: dodatkowe parametry modyfikacji adresowej; 2-bitowe pole ss (współczynnik skali = 1,2,4 lub 8), 3-bitowe pole index (kod rejestru modyfikacji mnożonego przez wsp. skali), 3-bitowe pole base (kod rejestru modyfikacji dodawanego do reszty). Adres efektywny w takim przypadku oblicza się ze wzoru base + ss*index + pole przesunięcia
- kolejne 4 bajty to pole przesunięcia, czyli przesunięcie danych w stosunku do początku segmentu

80. *W jakim celu kod rozkazu poprzedza się przedrostkiem zmiany rodzaju operandu*

W trybie rzeczywistym (i w trybie V86) standardowo używane są operandy 8- i 16-bitowe; jeśli jednak przed instrukcją wykonującą działanie na operandzie 16-bitowym

zostanie umieszczony dodatkowy bajt **66H**, to działanie zostanie wykonane na operandzie 32-bitowym; (np. na rejestrze ECX). Ten sam bajt umieszczony przed instrukcją, jeśli procesor pracuje w trybie 32-bitowym powoduje, że operacja zostanie wykonana na obiektach 16-bitowych (np. na rejestrze CX). Ten dodatkowy bajt nosi nazwę *przedrostka rozmiaru operandu*.

C. Mechanizmy adresowania

81. *Omówić podstawowe koncepcje modyfikacji adresowych*

Do obliczenia adresu efektywnego wykorzystywać można modyfikacje adresowe. Wtedy adres lokacji pamięci, na której wykonywane jest działanie, określony jest nie tylko poprzez pole adresowe rozkazu, ale zależy również od rejestrów. Można do tego celu wykorzystywać dowolne z 8 rejestrów ogólnego przeznaczenia (np. tablica[edi]), ponadto można jeszcze użyć drugiego rejestru dodatkowo pomnożonego przez współczynnik skali 1,2,4 lub 8 (np. adresy[edi][ecx*4]). Wszystkie modyfikacje zsumowane razem z polem przesunięcia dają adres efektywny.

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

LEA i **OFFSET** obliczają przesunięcie elementu względem początku segmentu, przy czym rozkaz **LEA** wyznacza adres efektywny w trakcie wykonywania programu, podczas gdy wartość operatora **OFFSET** obliczana jest w trakcie translacji (asemblacji) programu.

83. *W jakim celu stosowany jest współczynnik skali w modyfikacjach adresowych*

Stosowany jest, gdy chcemy odwołać się do konkretnej wartości w tablicy znając indeks tej wartości w tablicy oraz rozmiar w bajtach tej wartości. Np. w EDI umieszczamy indeks w tablicy, ustawiając współczynnik skali na ilość bajtów przeznaczonych na jeden element tablicy, załóżmy 4, modyfikacja adresowa typu tablica[edi*4] pozwoli nam na wygodny dostęp do kolejnych elementów tablicy.

84. *Jaka wartość zostanie wprowadzona do rejestru EDX po wykonaniu podanego niżej fragmentu programu*

```
linie  dd   421, 422, 443, 442, 444, 427, 432
—   —   —   —   —   —   —   —   —   —
    mov    esi, (OFFSET linie) + 4
    mov    ebx, 4
    mov    edx, [ebx] [esi]
```

443, bo modyfikacje adresowe się sumują. Do ESI najpierw wrzucamy adres elementu 422, potem dodajemy przesunięcie z EBX'a, czyli przesuwamy o kolejne 4 bajty i trafiamy na 443.

D. Programowanie w asemblerze

85. *Omówić trzy podstawowe formaty wierszy źródłowych w asemblerze*

Trzema podstawowymi formatami wierszy są:

- instrukcje (zawierać mogą etykietę, mnemonik akcji, operandy, komentarz)
- deklaracje zmiennych (zawierać mogą nazwę zmiennej, dyrektywę określającą rozmiar zmiennej, początkową wartość, komentarz)

- dyrektywy

86. *Jaką rolę pełnią dyrektywy w programie asemblerowym*

Dyrektywy są poleceniami, które nie są bezpośrednio tłumaczone na kod maszynowy, ale pozwalają asemblerowi na prawidłową interpretację kodu źródłowego. Wśród dyrektyw są m.in. dyrektywy makroprzetwarzania (MACRO, ENDM, REPT, EQU, IRP, IRPC...), dyrektywy dla linkera oraz organizujące kod (SEGMENT, EXTRN, ASSUME, PROC, ENDP, ENDS) itd.

87. *Wyjaśnić znaczenie terminów: makrowołanie i makrodefinicja*

Makrodefinicja to reguła przetwarzania tekstu źródłowego na wyjściowy, a makrowołanie to odwołanie do tej reguły.

88. *Jakie zastosowanie w programowaniu w asemblerze mają dyrektywy REPT, IRP, IRC*

Są to dyrektywy powtarzania, pozwalają na wielokrotne skopiowanie do programu zaznaczonego tekstu. REPT to powielanie tekstu zadaną ilość razy, IRP i IRPC pozwalają na wprowadzenie parametrów. Parametry podaje się w nawiasach trójkątnych < >, w przypadku IRP są to wartości liczby, w IRPC kolejne znaki stringa.

E. Operacje stosu i podprogramy

89. *W jaki sposób instrukcje PUSH i POP wpływają na stan wskaźnika stosu ESP*

ESP zawiera zawsze adres wierzchołka stosu, więc PUSH zmniejsza ESP o liczbę bajtów wrzucaną na stos (2 lub 4), POP odpowiednio zwiększa. Jest tak, bo stos "rośnie w dół", czyli ku niższym adresom.

90. *Omówić zasady działania rozkazów CALL, INT i RET (IRET)*

CALL wykonuje skok do podanej procedury, zapisując na stosie ślad, czyli zawartość EIP przed skokiem (adres instrukcji do wykonania po powrocie z procedury). INT wykonuje skok do procedury, której adres znajduje się w wektorze przerwań/deskryptorów przerwań pod numerem podanym w argumencie rozkazu, z reguły są to procedury systemowe. Na stosie zostawia ślad składający się z EIP, rejestru segmentowego CS i rejestru flag procesora EFLAGS. RET i IRET służą do powrotu z takich procedur, ściągają i przywracają zapamiętane wartości ze stosu: RET dla procedur wywołanych CALL'em, IRET - INT'em

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

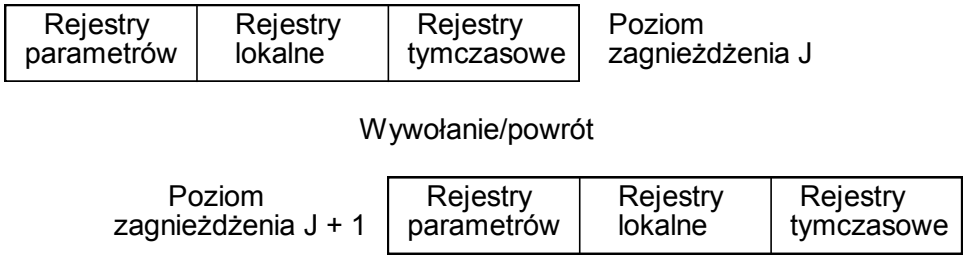
Są 3 metody przekazywania parametrów przez stos:
- standard Pascal (ładowane od lewej do prawej, podprogam sam ściąga ze stosu)
- standard C (ładowane od prawej do lewej, program wywołujący ściąga ze stosu)
- standard StdCall (ładowane od prawej do lewej, podprogram sam ściąga ze stosu, używany np. w WinAPI)

Ponadto można przekazać przez rejestry procesora (FastCall), albo przez ślad (parametry podane bezpośrednio w kodzie, tuż za wywołaniem call). Różnica między RISC'ami i CISC'ami w tym kontekście jest taka, że CISC przekazuje prawie zawsze przez stos, a RISC ma dużo rejestrów i zazwyczaj wykorzystuje je do przekazywania parametrów, co jest znacznie szybsze.

Procesory RISC mają ponadto standardowy mechanizm do wywoływania podprogramów – dostępne rejestry dzieli się na 3 części – rejestry z parametrami

przekazanymi do procedury, rejestry na zmienne procedury i rejestry tymczasowe, które zostaną przekazane następnej procedurze.

Te 3 zestawy rejestrów tworzą tzw. okno. Przy wywołaniu kolejnej procedury procesor tworzy kolejne okno; okna nakładają się na siebie – rejestry tymczasowe stają się rejestrami z przekazanymi parametrami w procedurze na kolejnym stopniu zagłębienia. W przypadku dużej ilości stopni zagłębienia, najstarsze okna przenosi się do pamięci.



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

Gdyby kompilatory stosowały zmienne w segmencie danych, byłyby to zmienne globalne, dostępne dla wszystkich innych funkcji, co w wielu przypadkach byłoby nieprawidłowe. Takie deklarowanie zmiennych ponadto zwiększa objętość kodu i czas jego wykonania. Korzystanie ze stosu umożliwia szybkie alokowanie i zwalnianie pamięci.

F. Lista rozkazów (instrukcji) procesora

93. *Omówić zasady działania rozkazów wykonujgcych działania na blokach danych*

W celu ułatwienia przepisywania lub inicjalizacji dużych obszarów danych, zdefiniowano rozkazy przesyłające bezpośrednio dane między dwiema lokacjami pamięci. Zdefiniowano także przedrostek **REP**, służący do wykonywania rozkazów tego typu w pętli, której ilość przebiegów kontroluje rejestr ECX. Do rozkazów działających na blokach należą:

- **MOVSB, MOVSW, MOVSD** – przesyłają dane z lokacji pod adresem **ds:[esi] (source index)** do lokacji **es:[edi] (destination index)** grupami po bajcie, słowie lub podwójnym słowie
- **LODSB, LODSW, LODSD** – ładowanie danych z pamięci wskazywanej przez **ds:[esi]** do rejestru **AL, AX** lub **EAX**
- **STOSB, STOSW, STOSD** – zapis danych z rejestru do pamięci wskazywanej przez **es:[edi]**
- **CMPSB, CMPSW, CMPSD** – porównywanie danych w pamięci, używane z przedrostkiem **REPE** lub **REPNE** służy do wyszukiwania par danych w blokach pamięci
- **SCASB, SCASW, SCASD** – porównanie danych w pamięci z zawartością rejestru **AL, AX, EAX**, używane z przedrostkiem **REPE** lub **REPNE** służy do wyszukiwania konkretnych danych w bloku pamięci

94. *Co oznacza termin arytmetyka nasycenia w odniesieniu do operacji MMX*

W operacjach MMX obok zwykłej arytmetyki „przewinięcia” stosowana jest także arytmetyka nasycenia, która jest wykorzystywana m.in. przy przetwarzaniu obrazów, np. rozjaśnianie obrazów jest ograniczone przez maksymalną jasność. W arytmetyce nasycenia, gdy wynik przekracza wartość minimalną/maksymalną, za wynik przyjmuje się tę wartość, niezależnie, jak bardzo różni się od rzeczywistego wyniku.

95. *W jaki sposób rozkaz IRET wpływa na zawartość rejestru znaczników*

Rozkaz **IRET** służy do powrotu z procedury wywołanej rozkazem **INT.** Odtwarza on stan sprzed wywołania tej procedury, przywracając ze stosu rejestr **CS, EIP** oraz **rejestr flag procesora**.

G. Sterowanie pracą urządzeń zewnętrznych

96. *Omówić podstawowe koncepcje komunikacji komputera z urzgdzeniami zewnętrznymi*

Komunikacją komputera z urządzeniami zewnętrznymi zajmują się specjalne układy pośredniczące, które dopasowują standardy sygnałowe procesora i płyty głównej do specyficznych wymagań poszczególnych urządzeń. Od strony procesora komunikacja odbywa się poprzez zapis/odczyt rejestrów zainstalowanych na układach wejścia/wyjścia (zwanych czasem sterownikami lub kontrolerami). Układy te znajdują się na kartach rozszerzeń lub na płycie głównej. Stosowane są dwie metody dostępu do tych rejestrów:

- ***współadresowalne układy we/wy*** – rejestry udostępniane są jako zwykłe komórki pamięci w przestrzeni adresowej pamięci, np. komunikacja z pamięcią ekranu
- ***izolowane we/wy*** – rejestry urządzenia dostępne są w odrębnej przestrzeni adresowej zwanej przestrzenią adresową we/wy lub przestrzenią portów; do przesyłania danych z i do takich układów służą rozkazy IN i OUT

Zlecenie, by urządzenie wykonało pewną operację wymaga zazwyczaj podjęcia wielu działań, kolejno sprawdzenia stanu urządzenia, wysłanie poleceń do urządzenia, zgromadzenie danych w buforze, przesłanie danych, sprawdzenie wykonania danych metodą przeglądania (odpytywania) lub oczekiwanie na przerwanie.

97. *Omówić podstawowe elementy systemu przerwań stosowanego w komputerach PC*

W nowszych komputerach PC system przerwań obsługiwany jest przez układ **APIC**, który pełni rolę „sekretarki" procesora, a także wspomaga prace wieloprocesorową. Wcześniej przerwania obsługiwane były przez dwa układy typu **8259 (PIC)**. Sygnały przerwań z poszczególnych urządzeń przesyłane są przez linie oznaczone symbolem **IRQ (interrupt request)**, układ **APIC** umożliwia dołączenie 24 takich linii. Z każdą linią **IRQ** skojarzony jest deskryptor przerwania w tablicy deskryptorów przerwań, tablicę tą inicjalizuje się programując układ **APIC** podczas inicjalizacji systemu operacyjnego. Po wystąpieniu przerwania sprzętowego, bezpośrednio przed uruchomieniem programu obsługi przerwania na stosie zapisywany jest ślad, który umożliwia powrót do przerwanego programu. Program obsługi przerwania kończy instrukcja **IRET**, która powoduje wznowienie wykonywania przerwanego programu poprzez odtworzenie rejestrów (**E)IP, CS**, i **(E)FLAGS**, na podstawie śladu zapamiętanego na stosie.
Układ **APIC** składa się z **jednostki lokalnej** w procesorze i **jednostki wejścia/wyjścia** na chipsecie płyty głównej. Jednostka wejścia/wyjścia rejestruje nadchodzące sygnały przerwań i wysyła je do jednostki lokalnej korzystając z wewnętrznej **magistrali ICC**; jednostka lokalna odbiera przerwanie i zgłasza je do procesora. Dostęp do rejestrów układu **APIC** realizowany jest techniką **współadresowania** (rejestry dostępne są w przestrzeni adresowej procesora).

98. *Jakg rolę podczas wywoływania podprogramów systemowych pełni tablica wektorów (deskryptorów) przerwań*

W wielu systemach zastosowano wywoływanie podprogramów systemowych w sposób pośredni, za pośrednictwem tablicy adresowej, zawierającej adresy podprogramów systemowych; w takim przypadku programista nie podaje adresu podprogramu, ale indeks w tablicy adresowej. W kolejnych wersjach systemu adresy zawarte w tablicy ulegają zmianom,

ale odwołania do konkretnych podprogramów realizowane są poprzez te same indeksy w tablicy adresowej.

99. *Wyjaśnić co oznacza termin "przestrzeń adresowa portów"*

Odrębna przestrzeń adresowa, w której dostępne są rejestry urządzenia (układów we/wy). Do zapisu i odczytu danych w przestrzeni adresowej portów stosuje się instrukcje IN i OUT oraz ich rozszerzenia.

100. *W jaki sposób organizowana jest obsługa wyjątków procesora*

W trakcie wykonywania programu przez procesor występują sytuacje uniemożliwiające dalsze wykonywanie programu, np. niezindentyfikowany kod rozkazu, próba zmiany zawartości lokacji poza dozwolonym adresem itd.; wystapienie takich sytuacji powoduje wygenerowanie wyjątku przez procesor. Wyjątek powoduje zapamiętanie śladu na stosie, wyzerowanie znacznika IF i rozpoczęcie wykonywania podprogramu właściwego dla określonego wyjątku.

101. *Wyjaśnić co oznacza termin priorytet przerwania*

Przerwaniom sprzętowym przypisano priorytety, są to liczby określające „ważność" przerwania w sensie kolejności wykonywania. Mniejsza liczba oznacza wyższy priorytet. Najwyższy priorytet posiada przerwanie generowane przez zegar systemowy. Możliwe jest przerwanie programu obsługi przerwania, jeśli nadejdzie przerwanie o wyższym priorytecie i podjęcie obsługi przerwania „ważniejszego". Warunkiem przyjęcia przerwania jest stan znacznika **IF=1**, w praktyce programy obsługi przerwań na czas swojego działania zerują znacznik **IF**.

102. *Jaką rolę w komunikacji z urządzeniami zewnętrznymi pełni obszar współadresowalny pamięci*

Jest to jedna z metod dostępu do zawartości rejestrów układów wejścia/wyjścia (sterowników, kontrolerów). Rejestry urządzenia są udostępniane jako zwykłe komórki pamięci w przestrzeni adresowej pamięci. Stosując tą metodę można odczytywać i zapisywać rejestry urządzenia tak samo jak zwykłe komórki pamięci np. za pomocą rozkazu MOV. W taki sposób działa np. pamięć ekranu w trybie tekstowym.

103. *W jakich okolicznościach używa się przerwań niemaskowalnych*

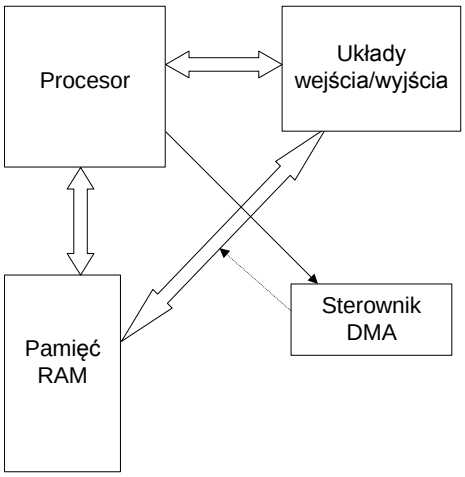
Przerwania niemaskowalne są to przerwania sprzętowe, które nie mogą być blokowane przez wyzerowanie znacznika IF. Stosowane są do sygnalizacji zdarzeń wymagających natychmiastowej obsługi niezależnie od stanu systemu, na przykład w przypadku wykrycia błędu pamięci RAM.

104. *Podać zasady wyświetlania znaków w trybie tekstowym poprzez bezpośredni zapis do pamięci ekranu*

Znaki wyświetlane na ekranie w trybie tekstowym są odwzorowaniem obszaru pamięci od odkreślonego adresu fizycznego. Każdy znak wyświetlany na ekranie opisany jest przez 2 bajty: bajt parzysty zawiera kod ASCII znaku, a nieparzysty atrybut, czyli sposób wyświetlania (kolor tła, kolor litery, miganie). Pamięć ekranu składa się z 4000 bajtów (80 kolumn x 25 linii, każdy znak zajmuje 2 bajty).

105. *Omówić podstawowe zasady działania układów DMA*

Układy DMA służą do przesyłania danych między urządzeniem zewnętrznym a pamięcią RAM z pominięciem procesora. Ma to na celu zwiększenia wydajności procesora, który nie musi zajmować się czasochłonnymi operacjami na pamięci. Układ DMA otrzymuje od procesora informacje o rodzaju informacji (odczyt/zapis), adresie urządzenia we/wy, adresie obszaru pamięci RAM do zapisania/odczytania i liczbę bajtów do przetworzenia. Po zakończeniu zadania układ DMA informuje o tym fakcie procesor generując przerwanie.



H. Architektury CISC i RISC

106. *Porównać charakterystyczne elementy architektury procesorów CISC i RISC*

Komputery o architekturze CISC charakteryzują się dużą ilością rozkazów, złożonymi i różnorodnymi metodami adresowania, różnymi długościami i czasami wykonania rozkazów i realizacją rozkazów opartą na technice mikroprogramowania. W celu zwiększenia wydajności procesorów stworzono architekturę RISC, opierającą się na założeniach wynikających z badań dotychczasowych programów, które wykazały, że złożone instrukcje stosuje się rzadko i nie wpływają na zwiększenie wydajności. Procesory RISC posiadają stosunkowo niewiele trybów adresowania, ograniczoną listę rozkazów o stałej długości, dużą ilość rejestrów ogólnego przeznaczenia, pozwalającą na zredukowanie liczby operacji na pamięci operacyjnej. Sterowanie wykonywaniem rozkazów odbywa się układowo, z intensywnym wykorzystaniem przetwarzania potokowego.

107. *Omówić zasady przekazywania parametrów do podprogramu stosowane w procesorach RISC*

Ze względu na dużą liczbę rejestrów ogólnego przeznaczenia w procesorach RISC, stosuje się zazwyczaj przekazywanie parametrów przez te rejestry, co jest dużo szybsze niż z wykorzystaniem stosu (jak w architekturze CISC). Każda procedura posiada dostęp do kilku grup rejestrów, są to rejestry na zmienne globalne, na parametry przekazane podczas wywołania, na zmienne lokalne i na zmienne tymczasowe, przekazywane do wywoływanych procedur. Trzy ostatnie grupy tworzą tzw. okno. Wywołując kolejną procedurę, procesor przełącza się do korzystania z innego okna rejestrów, przy czym kolejne okna zachodzą częściowo na siebie – rejestry na zmienne tymczasowe procedury bazowej stają się rejestrami na parametry przekazane do wywołanej procedury. W przypadku dużej głębokości zagęszczenia konieczne staje się zapamiętanie argumentów starszych procedur w pamięci operacyjnej.

108. *Opisać zasady działania pamięci statycznych i dynamicznych*

Pamięci statyczne zbudowane są z przerzutników dwustanowych (każdy to min. 6 tranzystorów). Ustawienie przerzutnika nie zalezy od czasu i trwa aż do zmiany informacji. Wyłączenie zasilania powoduje utratę informacji. Przerzutnik musi znajdowac się w stanie przewodzenia lub nie (przez co wydziela dużą ilość mocy). Pamięci statyczne są znacznie szybsze ale i znacznie droższe od pamięci dynamicznych.

Komórka pamięci dynamicznej zbudowana jest z tranzyzstora i mikrokondensatora. Tranzystor służy do ładowania kondesatora. Działanie pamieci oparte jest na magazynowaniu ładunku. Stan 1 jest nietrwały w czasie w wyniku nieuniknionych upływności, konieczne jest więc odświeżanie pamięci. Wykonuje się to zazwyczaj co kilka ms. Zawartość pamięci musi być też odświeżona każdorazowo po wykonaniu odczytu, gdyż operacja odczytu powoduje rozładowanie kondensatora. Pamięci te charakteryzują się większym stopniem integracji, znacznie niższą ceną i poborem energii, ale za to czas dostępu jest kilkunastokrotnie dłuższy niż w przypadku pamięci statycznej.

109. *Omówić algorytmy dostępu i aktualizacji zawartości pamięci podręcznej*

W trakcie wykonywania rozkazów procesor szuka najpierw rozkazów i danych w pamięci podręcznej. Jeśli podana informacja zostaje znaleziona (trafienie), przesyłana jest do procesora, jeśli nie ma jej w pamięci podręcznej, jest pobierana z pamięci głównej i umieszczana w pamięci podręcznej w postaci całego bloku (zasada lokalności). W trakcie zapisu procesor stara się zapisać do pamięci podręcznej, co wiąże się z koniecznością zapewnienia spójności zawartości pamięci podręcznej i głównej. Wyszukiwanie danych w pamięci podręcznej odbywa się najczęściej zgodnie z jedną z następujących koncepcji:

- ***pamięć podręczna z adresowaniem asocjacyjnym*** – pamięć podręczna podzielona jest na wiersze, a każdy wiersz na etykietę i blok. Etykietę stanowią starsze bity adresu, w bloku znajdują się dane skopiowane z pamięci głównej spod adresów rozpoczynających się od danej etykiety. Wyszukiwane odbywa się przez przeszukiwanie całej pamięci podręcznej i porównywanie etykiety z odpowiednimi bitami wyszukiwanego adresu, co jest realizowane przez dość skomplikowane i kosztowne układy
- ***pamięć podręczna z odwzorowaniem bezpośrednim*** – jest bardziej skomplikowana, ale tańsza w realizacji. Występuje np. w procesorach Pentium. Nie występuje konieczność przeszukiwania pamięci, bo część adresu (pole obszaru) wskazuje jednoznacznie miejsce, gdzie w pamięci podręcznej mogą znajdować się poszukiwane dane. W wierszu, oprócz danych bloku, zapamiętana musi być pozostała część adresu (etykieta), gdyż wiele bloków pamięci głównej może być zapisane w tym samym miejscu pamięci podręcznej i istnieje konieczność sprawdzenia, czy istniejące tam dane są tymi poszukiwanymi. Aby zwiększyć prawdopodobieństwo trafienia, stosuje się pamięci wielokanałowe (w których w jednym wierszu jednocześnie znajdować się może większa ilość bloków o tych samych polach obszaru, a różnych etykietach).

110. *W jaki sposób zasada lokalności wiąże się z pamięcią podręczną*

Obserwacje statystyczne wielu programów wskazują, że w jednostce czasu (np. w ciągu 1 ms) program odwołuje się tylko do stosunkowo niewielkiej liczby komórek pamięci, zawierających rozkazy i dane; wobec tego najczęściej używane dane i rozkazy można umieścić w niezbyt dużej, ale bardzo szybkiej pamięci statycznej, z której procesor może odczytywać (i zapisywać) informacje bez nadmiernie długiego oczekiwania. Wynika z tego, że pobierając dane do pamięci podręcznej, warto pobierać od razu całe bloki pamięci, gdyż istnieje znaczne prawdopodobieństwo wykorzystania także sąsiednich komórek. Pozwala to na zredukowanie konieczności wymiany informacji między procesorem a pamięcią główną, co z kolei wiąże się ze zwiększeniem wydajności systemu.

111. *Na czym polega problem zapewnienia spójności zawartości pamięci operacyjnej i pamięci podręcznej*

Istotnym problemem jest zapewnienie spójności zawartości pamięci operacyjnej (głównej) i pamięci podręcznej. Problem ten nie występuje jeśli pamięć operacyjna używana jest tylko do przechowywania rozkazów. Stosowane są dwie podstawowe metody:

- metoda *zapis przez* (ang. write-through) wykonuje zapis do pamięci głównej po każdej operacji zapisu w pamięci podręcznej. Wadą tej metody jest często występujące oczekiwanie na wykonanie zapisu w pamięci głównej, co podważa celowość stosowania pamięci podręcznej;
- metoda *zapis z opóźnieniem* (ang. write-back), polega na tym, że zamiast natychmiastowego zapisu bloku do pamięci głównej, zmienia się tylko bit stanu, oznaczający, że wiersz bufora został zmodyfikowany. Zmodyfikowany blok jest kopiowany do pamięci głównej, dopiero, gdy trzeba go zastąpić innym; jednak w systemach wieloprocesorowych czy też przypadku używania transmisji DMA zapewnienie spójności tą metodą może być problemem.

112. *Na czym polega różnica w sterowaniu mikroprogramowym, a układowym procesora*

Wykonywanie rozkazu przez procesor rozpoczyna się pobraniem rozkazu z pamięci, po czym identyfikowany jest jego kod – na tej podstawie jednostka sterująca w procesorze wydaje odpowiednie dyspozycje dla jednostki arytmetyczno-logicznej, rejestrów i innych podzespołów procesora, kierując odpowiednio przepływem i przetwarzaniem danych. Istnieją dwa podstawowe sposoby komunikacji jednostki sterującej procesora: jednostka sterująca mikroprogramowalna i układowa.

- ***sterowanie mikroprogramowe*** – sterowanie za pomocą wewnętrznego mikroprocesora z własnym wskaźnikiem instrukcji, wykonującego mikroprogram zapisany w pamięci ROM lub w tablicy logicznej **PLA (programmed logic array)**. Mikroprogram składa się z szeregu mikrorozkazów, odpowiedzialnych za pojedyncze operacje sterujące przemieszczaniem informacji między podzespołami i rejestrami procesora. Pozwala to na łatwe tworzenie nowych wersji procesorów o bardziej rozbudowanej liście rozkazów przy niezmienionych rozmiarach układów oraz na względnie łatwe usuwanie błędów
- ***sterowanie układowe*** – złożony układ cyfrowy zawierający bramki, przerzutniki i inne podzespoły. Istotnym elementem takiego sterowania jest licznik sekwencji. Po załadowaniu kolejnego rozkazu do rejestru rozkazów bity kodu operacji wraz z licznikiem sekwencji podawane są na wejście układu logicznego, który generuje odpowiednie sygnały sterujące. Pozwala to na nieco szybsze wykonywanie rozkazów, ale jest mniej elastyczne na poziomie projektowania i nieodpowiednie dla złożonych formatów rozkazów.

113. *W jaki sposób we współczesnych procesorach przewiduje się zachowanie instrukcji skokowych*

Aby zminimalizować prawdopodobieństwo zajścia konieczności unieważnienia potoku i napełniania go nowymi rozkazami przy skokach warunkowych, stosuje się predykcję skoku. Występuje kilka rodzajów takiego przewidywania:

- przewidywanie statyczne – nie zależy od historii wykonanych skoków, przyjmuje się np., że skok zawsze nastąpi, że nigdy nie nastąpi, lub, że zawsze nastąpi dla pewnych rodzajów rozkazów, dla innych nie nastąpi – to ostatnie podejście zapewnia ponad 75% prawdopodobieństwo sukcesu.
- przewidywanie dynamiczne – oparte na historii wykonanych skoków. Rejestruje się wystąpienie lub niewystąpienie skoku za pomocą bitu skoku. Jeśli skok poprzedno wystąpił, zakłada się, że wystąpi ponownie. W takim podejściu w przypadku pętli błąd zdarza się dwukrotnie. Lepsze rezultaty uzyskuje się stosując więcej bitów historii lub

zaawansowane tablice historii skoków, na podstawie których procesor wybiera bardziej prawdopodobną ścieżkę wykonania programu.

I. Proste przykłady kodowania w asemblerze

114. *Ile bajtów zarezerwuje asembler na zmienne opisane przez poniższe wiersze*

```
v1      db      ?
v2      dw      4 dup (?), 20
v3      db      10 dup (?)
```

21 bajtów. v1 to zmienna bajtowa, v2 to tablica z pięcioma elementami dwubajtowymi, v3 to tablica z dziesięcioma elementami bajtowymi.

115. *Na czym polega błąd w poniższym fragmencie programu*

```
const2 db       ?
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
        mov     const2, 256
```

256 to liczba, która nie mieści się w zmiennej bajtowej, bo zajmuje 9 bitów. W jednym bajcie mieszczą się liczby do 255 włącznie.

116. *Wskazać błędy zawarte w podanych niżej fragmentach programów (niektóre fragmenty nie zawierają błędów)*

```
a)
k       EQU     1024
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
        mov     k, ax
b)
temp    db      ?
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
        mov     temp, ax
c)
temp    db      ?
t3      db      10
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
        mov     temp, t3
```

a) k jest stałą zdefiniowaną dyrektywą EQU, czyli przy asemblacji wystąpienia „k“ będą zastąpione wartością 1024. Umieszczenie wartości rejestru AX w wartości stałej nie jest możliwe
b) AX to rejestr dwubajtowy, zmienna temp jest jedynie bajtowa
c) nie jest możliwe bezpośrednie przesłanie danych pomiędzy lokacjami pamięci. Dane skopiować można jedynie z wykorzystaniem stosu lub rejestru.

117. *Podać kilka instrukcji zerujących rejestr AX*

Na przykład:
- mov AX, 0
- xor AX, AX
- sub AX, AX
- and AX, 0

118. *Czym różni się działanie poniższych instrukcji*

```
mov    ebx, OFFSET table+4
lea    ebx, table+4
```

Wynik nie różni się, obie ładują adres efektywny drugiego elementu tablicy table. Różnica tkwi jedynie w specyfice instrukcji, patrz C2

119. *Wyjaśnić działanie poniższego fragmentu programu*

```
start: mov    ecx, 3
       sub    ax, 10
       loop   start
```

Program wykonuje nieskończoną pętlę, gdyż licznik obiegów pętli CX w każdym obiegu przywracany jest do tego samego stanu, powodującego skok do początku pętli. Wartość AX zmniejsza się o 10 w każdym wykonaniu, po zmniejszeniu do 0 następuje „przewinięcie".

EGZAMIN 2008

120. *Czemu wprowadzono tryb V86*

Można powiedzieć, że jest to podtryb trybu chronionego, procesor z punktu widzenia wykonywanego programu zachowuje się prawie dokładnie tak samo, jak w trybie rzeczywistym, nie tracąc przywilejów systemu Windowsa (jest to zarazem, przyczyną powstania, aby nie tracić przywilejów systemowych w DOSie). Niestety, nie wszystkie programy DOSowe, tolerują ten tryb i wymagają przełączenia na tryb rzeczywisty. Należy również dodać, że większość eksperymentów, które można dokonać na sprzęcie w trybie rzeczywistym, jest wykonywana w trybie V86 przez usługi systemowe.

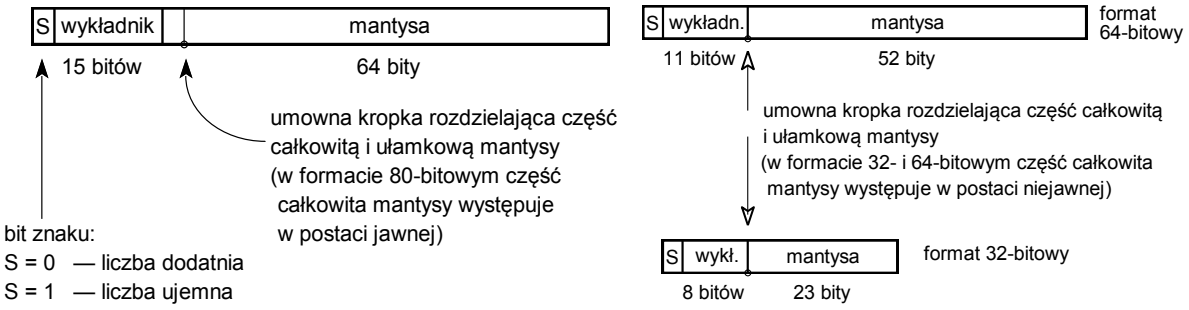
121. *Podaj treść jednego rozkazu, który zaneguje bity o nr 32-16 (starszą część) rejestru EDI, a resztę pozostawi bez zmian*

```
xor    edi, 0FFFF0000h   ; 1111 1111 1111 1111 0000 0000 0000 0000
```

122. *Czemu procesor przy rozkazie ADD zapisuje obie flagi: CF i OF*

Instrukcja ADD, używa ten sam algorytm dla liczb bez znaku, jak i liczb ze znakiem (kodowane w U2) ze względu na to, że np.: $(15)_{10} - (2)_{10}$, jest równoważne z zapisem $(15)_{10} + (-2)_{10}$. Tym samym, dla obu przypadków można wykorzystać, zwykły szkolny „słupkowy" algorytm dodawania. Niestety, różnice powstają podczas występowania nadmiaru i trzeba rozdzielić nadmiar dla liczb bez znaku (CF) i ze znakiem (OF), aby móc dokonać poprawne obliczenia.

123. *Uzupełnić brakujgce bity mantysy w podanej liczbie zmiennoprzecinkowej w formacie float, liczbę -0,3*



16383 = 3FFFH dla formatu 80-bitowego, czyli

$$liczba = mantysa \times 2^{wykladnik - 16383}$$

1023 = 3FFH dla formatu 64-bitowego (double) czyli

$$liczba = (mantysa + 1) \times 2^{wykladnik - 1023}$$

127 = 7FH dla formatu 32-bitowego (float), czyli

$$liczba = (mantysa + 1) \times 2^{wykladnik - 127}$$

Liczbę przedstawiamy w postaci iloczynu:
[mantysa] * 2[wykładnik]

Dodatkowo musi być spełniony warunek normalizacji (z powodu niejawnej jedynki, mantysa +1):
1 ≤ [mantysa] < 2

$1 \le 0{,}3 / 2^{[\text{wykładnik}]} < 2 \rightarrow 1 \le 0{,}3 / 2^{-2} < 2 \rightarrow 1 \le 0{,}3 / 0.25 < 2 = 1 \le 1{,}2 < 2$

Tutaj mamy liczbę typu float (32-bity), dlatego użyje: [wykładnik] = [jakaś_liczba] - 127

$1{,}2 * 2^{-2} = 1{,}2 * 2^{125 - 127} = 1{,}2 * 2^{(01111101) - 127}$

125	1
62	0
31	1
15	1
7	1
3	1
1	1
	0 (aby było 8-bitów, float ma wykładnik 8-bitowy)

Teraz należy: 1,2 – 1 (*niejawna jedynka*) = 0,2 → zamienić na kod zero-jedynkowy:

	2 (0,2)
0	4
0	8
1	6
1	2
0	4
0	8
1	6
1	2

0 4
0 8
… … (aż do uzyskania zera, albo potrzebnej długości mantysy)

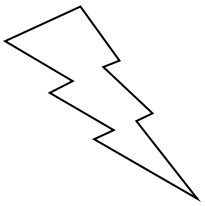
Wynik dla -0,3 (32-bitowa, float):

znak	wykładnik	mantysa
1	01111101	00 11 00 11 00 11 00 11 010

124. *Podane wartości rejestrów mm0 i mm1 i trzeba było wykonać na nich dodawanie z nasyceniem (rozkaz MMX)*

Dodawanie, odejmowanie wykonuje się według metody, którą przedstawiłem w **zad 6**, należy uwzględnić, że każdy rejestr *mm[cyfra]* może mieścić 8 ÷ 1 liczb, które są niezależne między sobą (nie ma dostępu do pojedynczych składowych). To znaczy, że trzeba przyjąć pewne założenia ile jest liczb (wyraźnie napisać na kartce), a później dokonać obliczeń (bez żadnych przeniesień, między różnymi liczbami). Jeśli będzie podany rozkaz, to można wszystko wyczytać z niego (patrz **zad 29**):

S – arytmetyka nasycenia
U – liczby bez znaku
L – młodsza część
H – starsza część
B – bajt (8-bit)
W – słowo (16-bit)
D – podwójne słowo (32-bit)
Q – poczwórne słowo (64-bit)



Np.: **PADDUSB** – dodawanie bajtowo liczb bez znaku, z arytmetyką nasycenia

125. *Podaj różnice pomiędzy systemami MIMD z pamięcią wspólną, MIMD ze rozproszoną pamięcią wspólną*

Patrz pyt. 24

126. *Co to jest API*

API (ang. **A**pplication **P**rogram **I**nterface) – jest to interfejs programowania aplikacji, oferujący użytkownikowi zazwyczaj zbiór różnych funkcji, stałych, oraz zmiennych potrzebnych i umożliwiających działanie programów, bądź wykonywanie zestawu elementarnych operacji. Możemy wyróżnić:

BIOS API – oferuje elementarny zestaw operacji, niezależny od konstrukcji konkretnego komputera (może „wyrównywać" różnice w konstrukcji sprzętu). Przy jego pomocy są wykonywane podstawowe czynności, tuż przed włączeniem systemu operacyjnego, tak jak przetestowanie pamięci, podłączonego sprzętu (np.: jeśli karta graficzna jest niedociśnięta, to zasygnalizuje „piskając" głośniczkiem), itd. W samym systemie operacyjnym (np.: Windows), to jest raczej rzadko stosowany ze względu, że pracuje w trybie 16-bitowym i nie jest przystosowany do zaawansowanych metod obsługi, lecz znalazł szerokie zastosowanie w systemie DOS.

Win32 API – używane w systemach Windows, jest to bardzo obszerny zbiór funkcji do tworzenia okien programów, elementów interfejsu użytkownika, obsługi zdarzeń, dostępu do sprzętu, itd. Zazwyczaj funkcje zawarte są w bibliotekach .dll (w 16-bitowych wersjach z rozszerzeniem .exe) dostarczonych wraz z systemem operacyjnym. Ze względu na to, że liczba bibliotek rośnie wraz z każdą wersją nowego Windowsa, to niekiedy powstają problemy uruchomienia aplikacji napisanej w starej wersji systemu, pod kontrolą nowej.

127. *Jaki związek mają rejestry RAX, RBX, … z rejestrami EAX, EBX, …*

Rejestry należą do 64-bitowego rozszerzenia architektury IA-32, nazwanego **AMD64** (bądź **IA-32e**), gdzie jedną z głównych zmian (jest ich dużo) było rozszerzenie rejestrów do 64-bitów (dodając przedrostek R, zamiast E), oraz 8 zupełnie nowych rejestrów ogólnego przeznaczenia R8÷R15. Nadal można stosować operandy 32-bitowe, nawet jest dostęp do młodszych części, więc program napisany pod architekturę IA-32, będzie działał w architekturze AMD64, ale na odwrót już nie.

128. *Opisać procedury przydzielania pamięci dynamicznej na stosie i odwoływania się do niej*

Patrz zad. 21

129. *W jaki sposób procesor jest informowany o rozmiarze operandu (16 czy 32 bity), którego ma użyć w rozkazie*

Patrz zad. 9 i 80

130. *Dlaczego możemy wymiennie używać rozkazów CMP i SUB ?. Dlaczego to drugie rozwiązanie jest niewygodne*

Patrz zad. 4

131. *Przedstawić budowę i opisać działanie mikroprogramowalnej jednostki sterującej*

Patrz zad 112

132. *Bajt* **SIB** *- kiedy go używamy*

Jeżeli w 32-bitowym trybie (wyłącznie tam), kodowany rozkaz zawiera wartość **r/m=100**, to oznacza że instrukcja zawiera dodatkowy bajt, zwany **SIB.** Opisuje on sposób obliczania adresu efektywnego, czyli określa współczynnik skali (**ss**), pierwszy rejestr modyfikacyjny (**base**), oraz drugi rejestr modyfikacyjny (**index**, nie można użyć rejestru **ESP**). Wzór na adres efektywny: **[pole_adresowe]** (można pominąć) + **[base] + [index]*[ss]**.

133. *Jak wrzucić jakąś wartość, do wskaźnika adresowanego (EIP)*

Patrz zad. 59

134. *Maskowanie wyjątków koprocesora w językach wysokiego poziomu*

Różnego rodzaju obliczenia, mogą powodować wyjątki koprocesora, które prowadzą do powstawania wartości specjalnych. Te wartości z powodzeniem, mogą być argumentami

do dalszych obliczeń, często nawet prowadzą do uzyskania poprawnego wyniku końcowego, bądź dostatecznie dobrego przybliżeniem (ułamki nieskończone). Z tego powodu, „wyjątki łagodne”, *takie jak: dzielenie przez zero, niedomiar, niedokładny wynik (np.: 1/3, wtedy dokonuje zaokrąglenia), operand nienormalizowalny.* Domyślnie się je maskuje, ale oczywiście niektórych sytuacjach, jak w przeliczeniach bankowych jest to niezalecane. Rodzaje wyjątków:

- *niedokładny wynik* (**PE – P**recision **E**xception) – zapisanie ułamka nieskończonego (np.: 1/3), nie można dokładnie zapisać binarnie w formacie zmiennoprzecinkowym, ponieważ mantysa zajmuje określoną liczbę bitów
- *nadmiar* (**OM**) – gdy wartość bezwzględna wyniku jest zbyt duża, aby ją zapisać. Powstaje wtedy wartość specjalna, nieskończoność
- *niedomiar* (**UE** – **N**umeric **U**verflow) – wynik jest różny od zera, ale jego wartość bezwzględna jest tak mała, że nie można przedstawić w formacie zmiennoprzecinkowym. Powstaje wtedy wartość specjalna, kodowana według pewnych reguł (*niedomiar stopniowy*)
- *operand nienormalizowalny* (**DE**) – powstaje, gdy jeden z operandów wykonywalnego rozkazu jest nienormalizowalny (ma bardzo małą dokładność, wynik może być niezadowalający). Uważa się, że jest to niegroźny wyjątek i zazwyczaj się maskuje
- *dzielenie przez zero* (**ZE**) – powstaje, kiedy dzielimy liczbę różną od zera, przez zero. Wynikiem operacji jest wartość specjalna, nieskończoność.
- *niedozwolona operacja* (**IE**) – powstaje, gdy niemożliwe jest żadne inne działanie, np.: próba obliczenia pierwiastka z liczby ujemnej, próba użycia pustego rejestru stosu, itd. Powstaje wtedy *nieliczna – NaN (Not a Number)* (wykładnik same jedynki, mantysa różna od zera). Jeśli jeden z operandów jest NaN`dem, to wynik też będzie NaN`dem.

Rodzaje zaokrągleń (decyduje przyjmowana wartość bitów RC w rejestrze koprocesora):

- zaokrąglenie w kierunku liczby najbliższej [**RC = 00**]
- zaokrąglenie w kierunku zera (obcięcie dodatkowych bitów) [**RC = 11**]
- zaokrąglenie w górę (do +niesk) [**RC = 10**]
- zaokrąglenie w dół (do -niesk) [**RC = 01**]

135. *Omówić pojęcie: Interfejs ABI*

ABI (ang. **A**pplication **B**inary **I**nterface) – zespół reguł i ustaleń, które decydują o współpracy między oprogramowaniem, a systemem operacyjnym. ABI różni się od API tym, że dotyczy programów w wersji skompilowanej (binarnej, wykonywalnej), a nie w formie kodu źródłowego. Definiuje sposób wywoływania funkcji, przekazywania argumentów, oraz odbierania zwracanej wartości.

136. *Metody przekazywania parametrów w procesorach RISC*

Patrz pyt. 91, 106, 107

137. *Po co w znakowaniu tekstu występuje bajt BOM i BMP*

BOM (**B**yte **O**rder **M**ark) – znacznik kolejności bajtów. Są dwa sposoby, przechowywania liczb w pamięci komputera:

- *„mniejsze niżej” (little endian)* – mniej znaczące bity, znajdują się w niższych adresach w pamięci, np.: liczba 123456H, będzie przechowywana jako: 56H 34H 12H

- *„mniejsze wyżej” (big endian)* - naturalna kolejność przechowywania, np.: liczba 123456H, będzie przechowywana jako: 12H 34H 56H

W celu zasygnalizowania sposobu kodowania, dodaje się przed tekstem znacznik **BOM** (składający się z kilku bajtów), np.: jeśli mamy zamiar wysłać tekst, kodowany w UTF-16 za pomocą „mniejsze niżej”, to wysyła najpierw dwa bajty **FFh FEh.** Przy kodowaniu w **UTF-8**, znacznik **BOM** można pominąć, ponieważ domyślnie jest (koniecznie trzeba użyć) sposobu przechowywania bajtów „mniejsze niżej”.

BMP (**B**asic **M**ultilingual **P**lane) – 16-bitowy podzbiór (32-bitowego) standardu **ISO 10646**, obejmujący wszystkie używane znaki.

138. *Wywoływanie funkcji w standardzie WinApi (czyli std call)*

Patrz pyt. 91

139. *Jakie wybrać adresowanie gdy, nie znamy adresu podczas kompilacji*

Naturalnie, należy wybrać adresowanie za pomocą modyfikacji adresowych, np.: mov ecx, tablica[ebx][eax*4]. Pierwszy rejestr nazywa się rejestrem modyfikacji (**base**), drugi rejestrem modyfikacji (**index**, nie można użyć rejestru **ESP**), a *4 jest współczynnikiem skali (**ss**, może być *1, *2, *4, *8). Wzór na adres efektywny: **[pole_adresowe]** (można pominąć) + **[base] + [index]*[ss]**.

Użycie modyfikacji adresowej, pozwala nam w prosty sposób odwoływać się do danych, nie znając ich końcowego adresu. Jeśli mamy do czynienia z tablicami, wystarczy podać jej OFFSET, index który nas interesuje i współczynnik skali (jeśli tablica jest typu bajt, to użyjemy ss=1, jeśli słowo, to ss=2, podwójne słowo ss=4, itd.).

140. *Kiedy zamiast mnożenia i dzielenia, używamy przesunięć SAL i SAR*

Najpierw trzeba wyjaśnić, że występuje kilka typów przesunięć bitowych, które można podzielić na:

- *logiczne* – zwyczajne przesunięcie bitowe (**SHL, SHR**)
- *arytmetyczne* – to samo, co przesunięcie logiczne, ale traktują zawartość rejestru, lub komórki pamięci, jako liczbę ze znakiem (**SAL, SAR**)
- *cykliczne* – wychodzące bity są zwracane, z drugiego końca rejestru (**ROL, ROR, RCL, RCR**)

Przesunięcia bitowego *SAL*, można użyć jeśli chcemy pomnożyć liczbę przez wielokrotność liczby 2. Każde przesunięcie o jedną pozycje w lewo, jest traktowane jako pomnożenie przez 2^k, gdzie k- oznacza liczbę pozycji. Bity wychodzące poza rejestr, lub komórkę pamięci przechodzą przez flagę **IF**. Zapozatym, jeśli przesuwamy tylko o jedną pozycje, to ustawia się flaga **OF** (OF=1, jeśli dwa najstarsze bity przed przesunięciem są NIEjednakowe, przeciwnym wypadku OF=0).

Przesunięcie bitowe *SAR*, służy do dzielenia przez 2^k, nie ustawia flagi **OF**, oraz podczas przesuwania (tak jak wyżej, bity przechodzą przez flagę **IF**), jest powielany bit znaku.

Użycie tych rozkazów jest znacznie szybsze, niż użycie instrukcji MUL i DIV, więc jeśli tylko jest możliwość, to warto zastąpić (mnożenie/dzielenie przez wielokrotność 2).