Maszyna tur…

poniedziałek, 24 czerwca 2024 20:58

Uwaga! Zaczynamy od pliku dotyczącego pamięci podręcznej! Około 54:30 minuty wykładu.

PRZYPOMNIENIE IMPELENTACJI MODELU TURINGA (PO RAZ 1000000000000000):

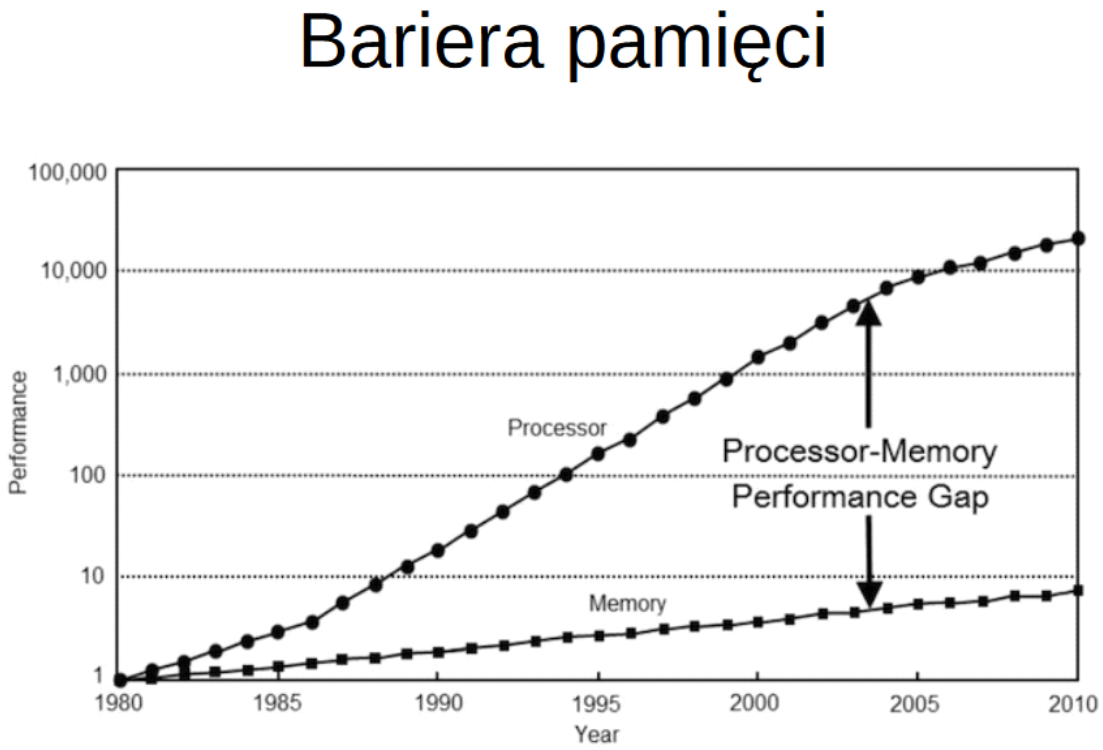
Mamy procesor, mamy pamięć operacyjną i magistralę łączącą te dwa elementy, z powodu, że ta magistrala nie przyspiesza, jej parametry nie zwiększają się tak szybko jak moc obliczeniowa pojawia się pojęcie Bariery Pamięci lub Bariery Von Neumanna.

Problem jest przede wszystkim fizyczny, o ile w samym procesorze możemy zmniejszać tranzystory…
Trochę w dużej mierze nieprawda: Im mniejsze tranzystory tym elektrony mają bliżej i wszystko szybciej może działać. To mniej więcej jest nie prawda, ale tłumaczy ideę, że im mniejsze elementy tym mogą szybciej działać.

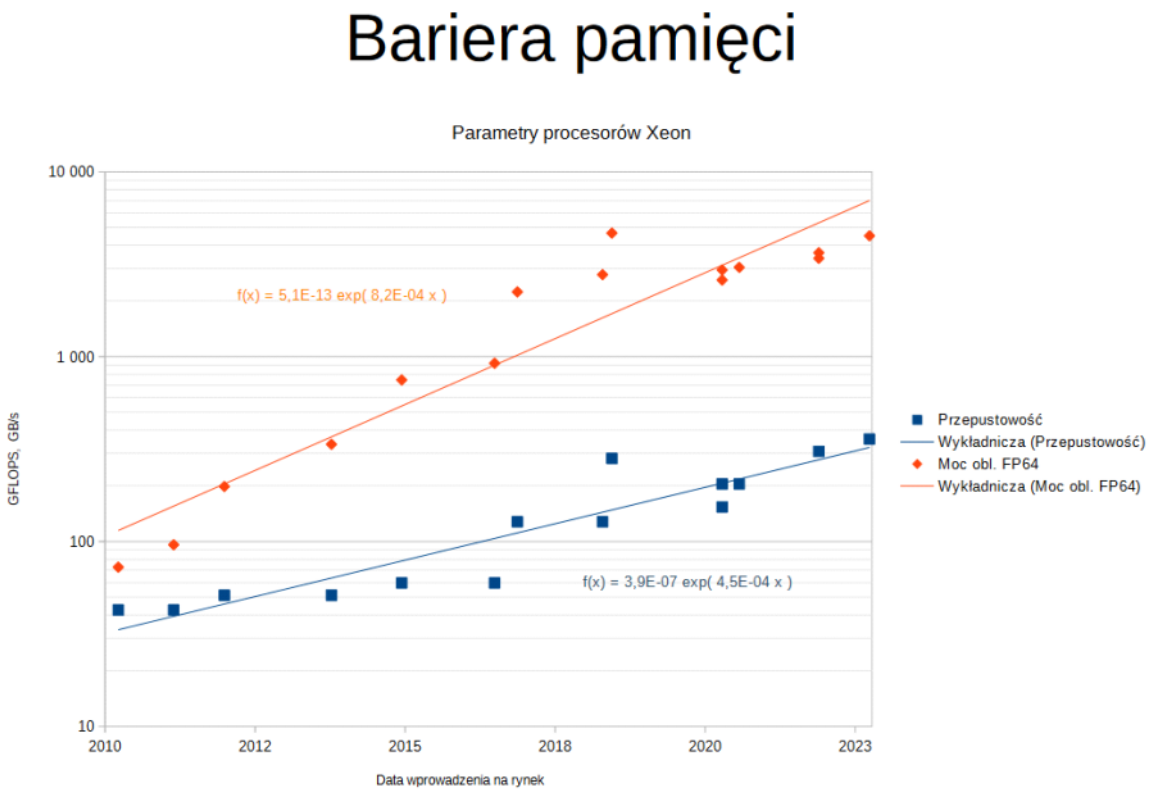
O tyle odległość między procesorem a pamięcią jest jaka jest.

1GHz to 30 cm!
8GHz to 3,5 cm!
A od procesora do pamięci jest z 6,7,15.

Zmiany w wydajności pamięci i procesora na przestrzeni lat:



Dane dla najszybszych Xeonów z ostatnich 15 lat:



Z każdej rodziny Tomczak wybrał najszybsze Xeony i obliczył ich moc obliczeniową, w tej chwili jesteśmy w zakresie pojedynczych TFLOPS-ów. Wykładnik 8 coś
Jeśli chodzi o pamięć w tej chwili jesteśmy na poziomie kilkuset GB/s. To są wykresy w skali logarytmicznej. Wykładnik 4 coś.
Moce obliczeniowe rosną coraz bardziej w stosunku do szybkości pamięci.
Co więcej, wcześniej moc obliczeniowa rosła jeszcze szybciej, od pewnego czasu odpuścili, bo to nie miało sensu. Nie ma molżliwości dostarczenia tylu danych, żeby to wykorzystać.

Jak piszemy programy problem nie jest w tym, żeby wybierać instrukcje, które każdy cykl procesora będą wykorzystywały maksymalnie tylko żeby jak najmniej tracić na przesyłaniu danych.

Zazwyczaj pisze się kody ograniczone przepustowością danych, a jeszcze częściej opóźnieniami danych. W Riscach nic nie da się zrobić normalnymi technikami.

Jak sobie z tym radzić?
W wielu kodach te same dane, albo dane bliskie sobie są przetwarzane wielokrotnie albo są przetwarzane w przedziale czasowym. Dane znajdujące się blisko siebie są przetwarzane w bliskim sobie czasie.

Pętla - z natury swojej jest wielokrotnie przetwarzana i trzeba wielokrotnie instrukcje z pamięci odczytywać jak i dane, które przetwarzamy też mogą być bliskie sobie.

Jak macie zmienną tymczasową w pamięci.
To prawdopodobnie używacie jej więcej niż jeden raz.
I to nas prowadzi do dwóch bardzo ważnych terminów:

Lokalność przestrzenna - jeśli używaliśmy jakichś danych istnieje szansa, że będziemy używać danych z bliskiego im otoczenia

Lokalność tymczasowa - jeśli używaliśmy jakichś danych to prawdopodobnie użyjemy ich znowu

Dlatego ma sens umieszczenie w procesorze pamięci, które będą przechowywały dane, które podlegają tym zjawiskom. Bo jeśli będziemy ich używać wielokrotnie to nie ma sensu ich wiele razy przesyłać z pamięci.

Tylko dobrze by było, żeby te bufory w procesorze były z punktu widzenia programisty niewidoczne. Przynajmniej bezpośrednio. I w praktyce one rzeczywiście są niewidoczne z takiego funkcjonalnego punktu widzenia. Natomiast bardzo mocno wpływają na czas wykonania kodu. Czyli one są widoczne pośrednio. (A pod koniec wykładu pojawią się też wersje jawnie zarządzane przez programistę, takie rozwiązania też mają swoje zalety)

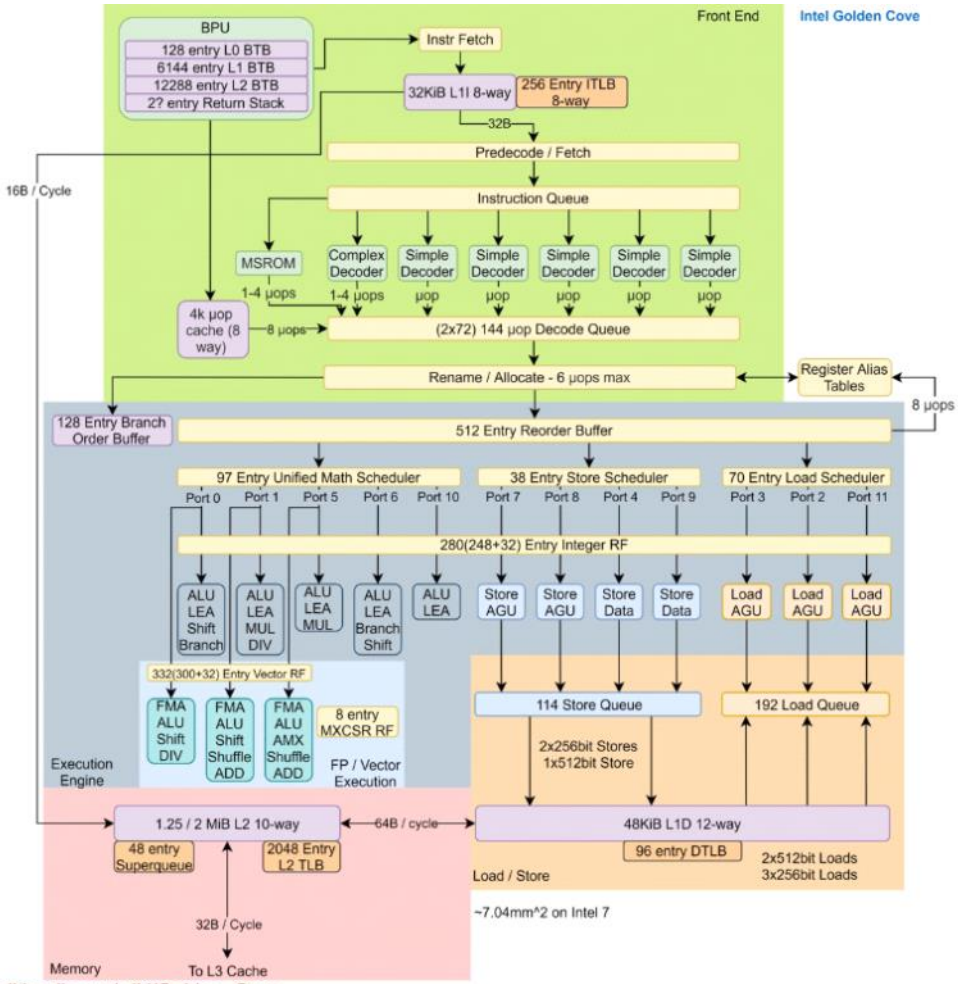
Ale na początku załóżmy, że bufory tych danych wielokrotnie używanych są zarządzane automatycznie w sposób niewidoczny dla programisty.

MILCZĄCY FILAR

wtorek, 25 czerwca 2024 00:10

Są dwa rodzaje buforów danych lokalnych:
A) Różnego rodzaju kolejki:
-rozkazu
-odczytu
-zapisu
Tomczak zaczyna wskazywać jakieś kolejki i nie wiem o co mu chodzi. Ktoś tu disuje mnie, niech… Czyli są kolejki, które służą do tego, żeby przechowywać dane w takiej kolejności w jakiej nadchodzą. (one nas mniej interesują)
B) Pamięci, które mogą przechowywać dane dostępne w sposób niezależny od ich ułożenia w pamięci. Dodatkowo te bufory muszą być tak skonstruowane, żeby jakoś efektywnie wiązać jakich danych używamy i gdzie one są w tych buforach przechowywane.

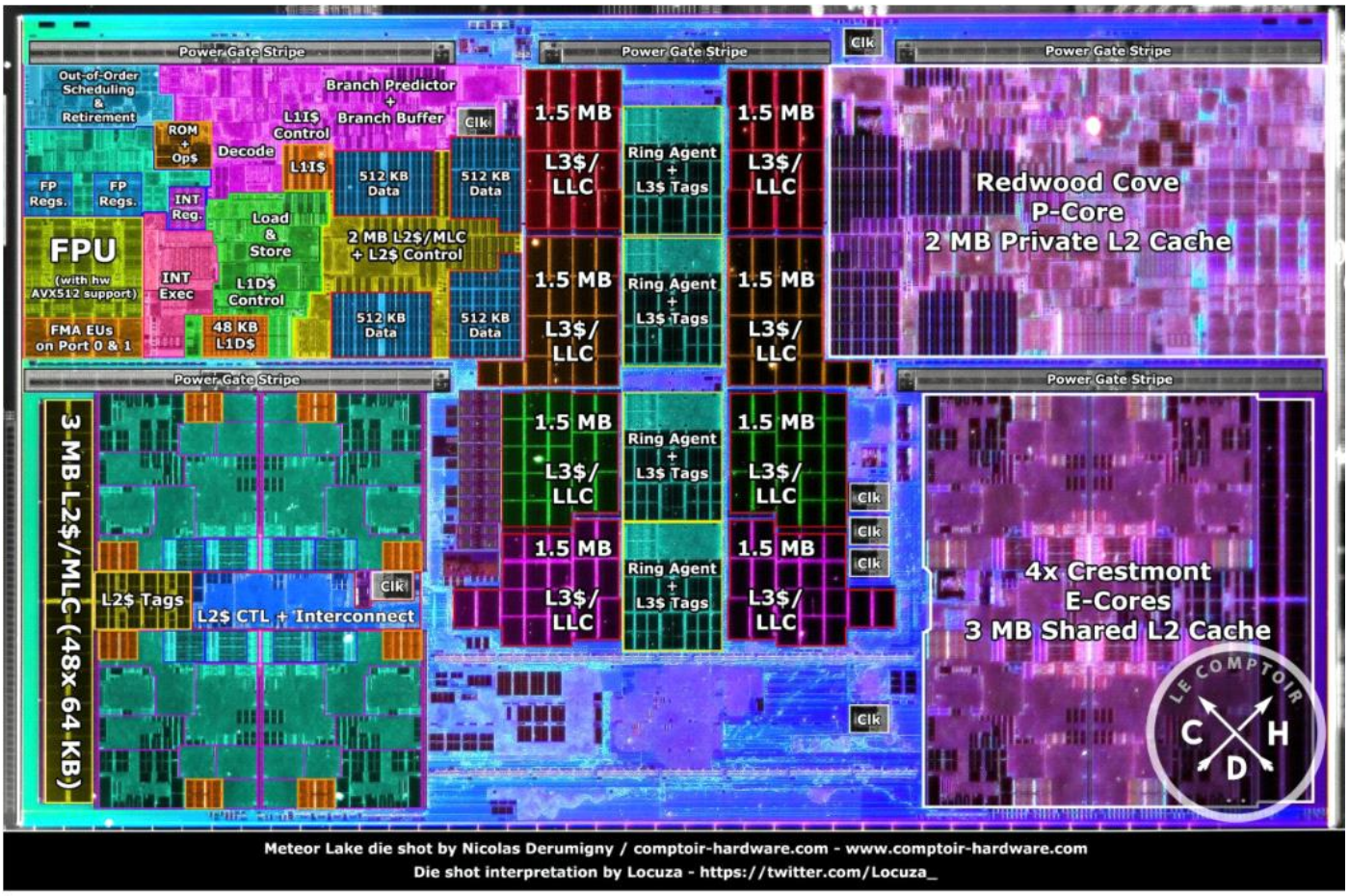
Dlaczego tak długo Tomczak o tym mówi i dlaczego ten wykład zajmie jeszcze tak dużo czasu?
Bo jak popatrzymy na ten schemat:



11

Źródło: https://en.wikipedia.org/wiki/Golden_Cove

Tu mamy jeszcze jakieś bufory, tu mamy bufory, tu mamy bufory, tu mamy bufory, tu mamy podłączenie do bufora, tu mamy bufory i tu mamy bufory



Źródło: https://wccftech.com/intel-14th-gen-meteor-lake-cpu-die-shot-redwood-cove-crestmont-cores-compute-tile/

Tomczak opisuje obrazek po kolei każde coś…FPU, rejestry zmiennoprzecinkowe, jednostkę stałoprzcinkową, układ dekodowania i przewidywania skoków,
a cała reszta to są pamięci podręczna.

CAŁA RESZTA!!!!

MOŻE NIE JEST TO NAJWAŻNIEJSZY ELEMENT PROCESORA, ALE NA PEWNO NAJWIĘKSZY!!!

PAMIĘĆ PODRĘCZNA

wtorek, 25 czerwca 2024 00:32

Pamięć podręczna - znajduje się między procesorem, a pamięcią operacyjną i jest podłączona do magistrali łączącej procesor z pamięcią operacyjną. Jeżeli procesor chce coś odczytać z pamięci operacyjnej sterownik pamięci podręcznej sprawdza czy to co procesor chce odczytać nie jest przypadkiem zapisane w pamięci podręcznej. Jeżeli te dane są przechowywane w pamięci podręcznej to sterownik pamięci podręcznej może je przesłać do procesora a zablokować procesorowi dostęp do pamięci.

Pamięć podręczna jest szybka. Wielokrotnie od pamięci operacyjnej natomiast jest mniejsza, bo jak jest szybka to musi być mała. Również dlatego, że jest fizycznie bliżej procesora.

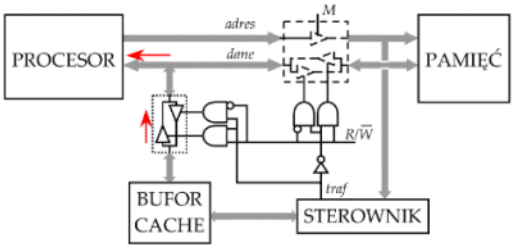
Co do tego, że jest mniejsza:
Zupełnie inny dekoder adresu będzie dla magistrali ośmio-bitowej a zupełnie inny dla 30-bitowej, głębokość rośnie logarytmicznie ale rośnie.

Co się dzieje gdy w pamięci podręcznej nie ma tych danych, które chcemy odczytać?
Wtedy dane muszą być przesłane z pamięci operacyjnej, ale przy okazji bufor pamięci podręcznej może je sobie zapisać gdzieś w środku, I wtedy przy następnym dostępie procesor będzie mógł otrzymać te dane z kopii, która się znajduje blisko.

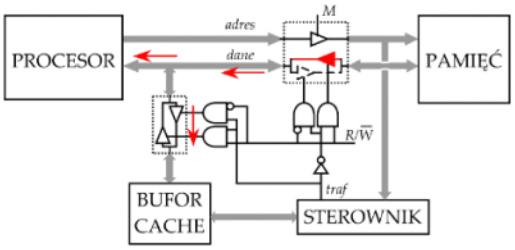
Z pamięcią podręczną wiążą się dwa bardzo ważne terminy:
TRAFIENIE- Sytuacja gdy podczas próby odczytania dane są w pamięci podręcznej (mniej operacji i szybciej)

CHYBIENIE- Sytuacja gdy podczas próby odczytania/zapisania/zapisania danych nie ma w pamięci podręcznej (więcej operacji i wolniej)

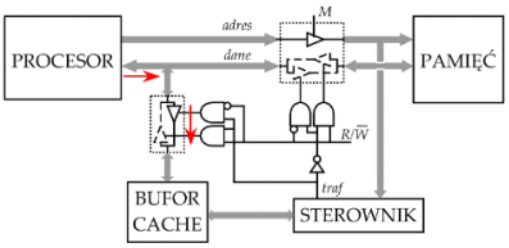
trafienie:



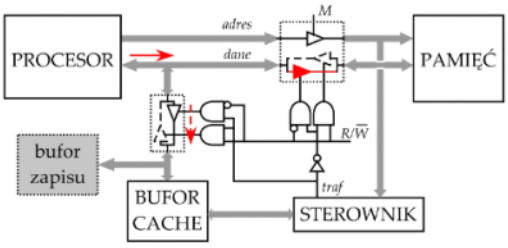
chybienie:



chybienie: etap ① – odczyt z pamięci (blokowy) (→ usunięcie linii → wypełnieni
etap ② – kopiowanie odczytanego bloku



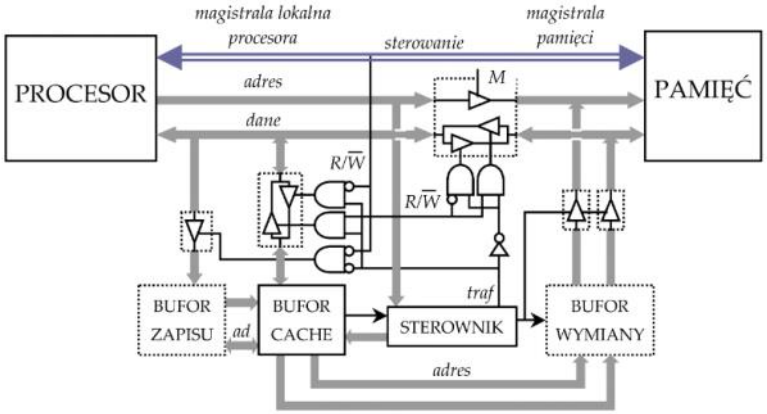
trafienie:



chybienie:

chybienie: etap ① – (*no allocate on write*) zapis do pamięci/ bufora zapisu,
etap ② – ... /aktualizacja odczytu zawartością bufora zapisu

Cała sztuka używania pamięci podręcznej polega na tym, żeby mieć jak najwięcej trafień i jak najmniej chybień.



Współpraca pamięci podręcznej i pamięci głównej (*traf* – sygnał trafienia w buforze *cache*, *R/W* – odczyt/zapis (1/0), *M* – transfer/blokada adresu (1/0))

BUFOR ZAPISU - służy do przechowywania zapisówk żeby nie blokować operacji jak się dane zapisują do pamięci.
BUFOR WYMIANY (ang. Victim cache) - te rzeczy, które są z bufora pamięci podręcznej są usuwane są przenoszone do niego, bo… może jednak niepotrzebnie usunęliśmy

W tym buforze są linie.

I. Linia - pewna liczba bajtów w pamięci operacyjnej, w tej chwili 64 bajty w typowych rozwiązaniach. (chociaż w praktyce można mieć od 16 do 256 bajtów na jedną linię) I to są 64 bajty spod adresu podzielnego przez 64 bajty. Pierwszy bajt spośród tych pasujących do linii znajduje się pod adresem podzielnym przez rozmiar linii. W ten sposób bardzo łatwo można powiązać adres danej z numerem linii. Nie trzeba się zastanawiać, która linia wiąże się, z który adresem generowanym przez procesor.

BARDZO WAŻNE!!!

Co jest zapisane w pojedynczej linii? Z naszego punktu widzenia 3 rzeczy:

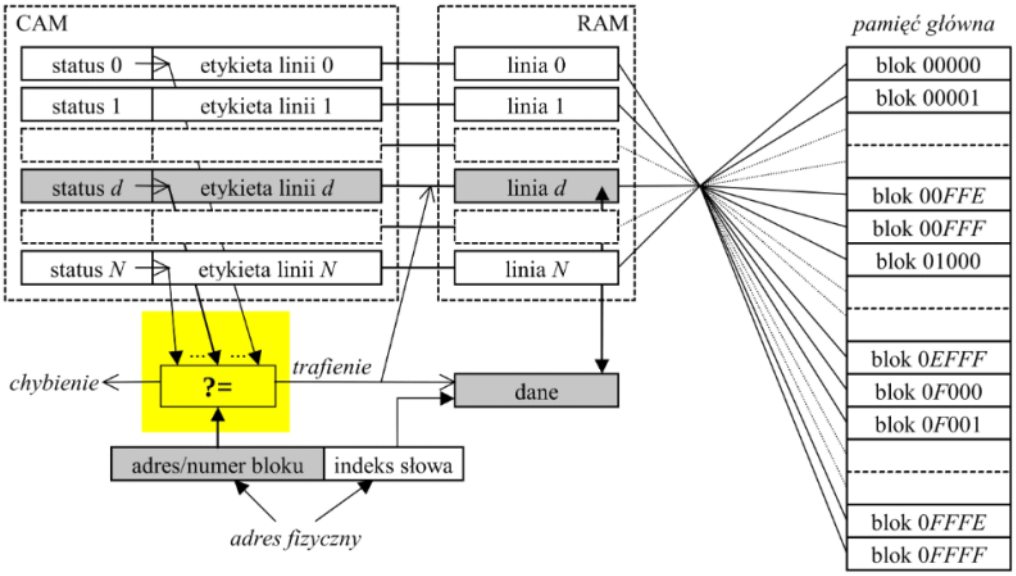
1. Kopia danych z pamięci operacyjnej (oczywiście)
2. Etykieta (ang. tag) - informacja wiążąca to co jest w linii z rzeczywistym adresem tych danych w pamięci operacyjnej. Zazwyczaj jest to część adresu danych z pamięci operacyjnej.
3. Dodatkowe bity mówiące czy w tej linii są jakieś dane czy nie.

Ten rozmiar linii dobrze jakby był potęgą dwójki, bo wtedy wystarczy, że z adresu odejmiemy ileś bitów i już mamy prosty sposób wybierania jednego bajtu z linii.

KOLEJNE BARDZO WAŻNE!!!

Wewnętrzna organizacja pamięci podręcznej (Bardzo mocno wpływa na sposób użycia pamięci podręcznej co bardzo mocno wpływa na szybkość):

Odwzorowanie całkowicie skojarzeniowe



Bufor całkowicie asocjacyjny (ang. *fully associative*)

W dowolnej linii mogą być dowolne fragmenty pamięci głównej.

Co musi się znajdować w takiej linii:

- Dane z pamięci głównej (kopia)
- Ta część adresu tych danych, która mówi, która to jest linia z pamięci głównej. (Jeżeli w linii są 64 bajty to 6 najmniej znaczących bitów adresu może zostać odcięte na wybranie bajtu z linii. Pozostała część adresu może być traktowana jako numer w pamięci głównej. W tej linii pamięci podręcznej przechowywana jest ta reszta pamięci adresu. (jak procesor chce odczytać dane z pamięci adresu z tej pierwszej linii to wystarczy w buforze pamięci podręcznej sprawdzić, w której z tych etykiet linii znajduje się adres zawierający same zera na tych bitach)

I TYLE.

Problemy w tej organizacji:

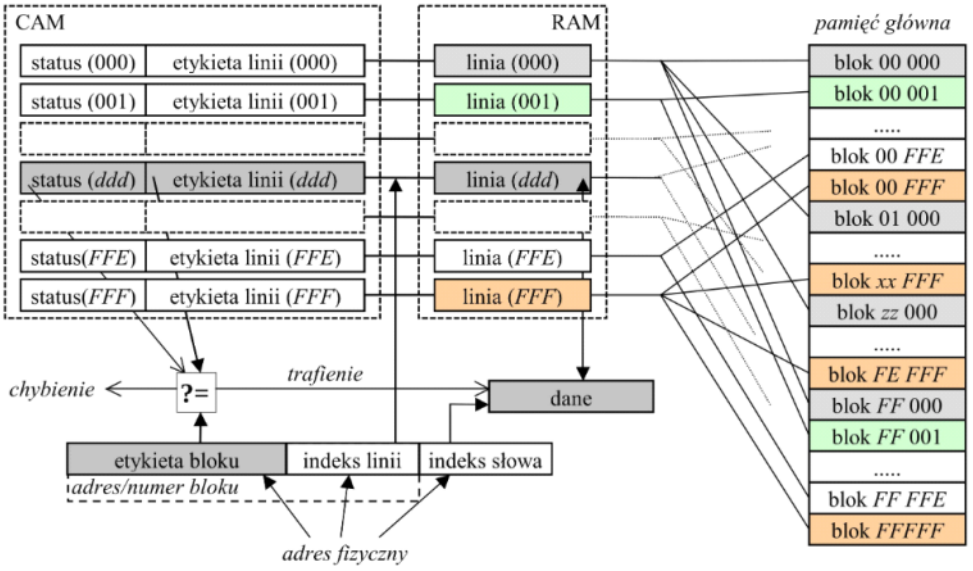
- Musimy przeszukać wszystkie linie, jeśli będzie ich dużo układ przeszukujący będzie bardzo duży, a po drugie z definicji będzie wolny, ta szybkość będzie zależała od logarytmu, ale złożoność będzie zależała od liczby linii

I z tego powodu tego rodzaju pamięci stosuje się bardzo rzadko i są raczej małe. W ten sposób, 3, 30

czy 300MB pamięci zrobić się nie da.

DLATEGO NAJWIĘCEJ STOSUJE SIĘ TEGO:
Z przeplotem - adres sjest podzielony tak, że kolejne linie są poprzeplatane

Odwzorowanie bezpośrednie – z przeplotem



Odwzorowanie bezpośrednie (ang. *direct mapped*) z przeplotem (ang. *interlace*) bloków

Adres linii/bajtu jest dzielony nie na dwa elementy , ale na trzy :

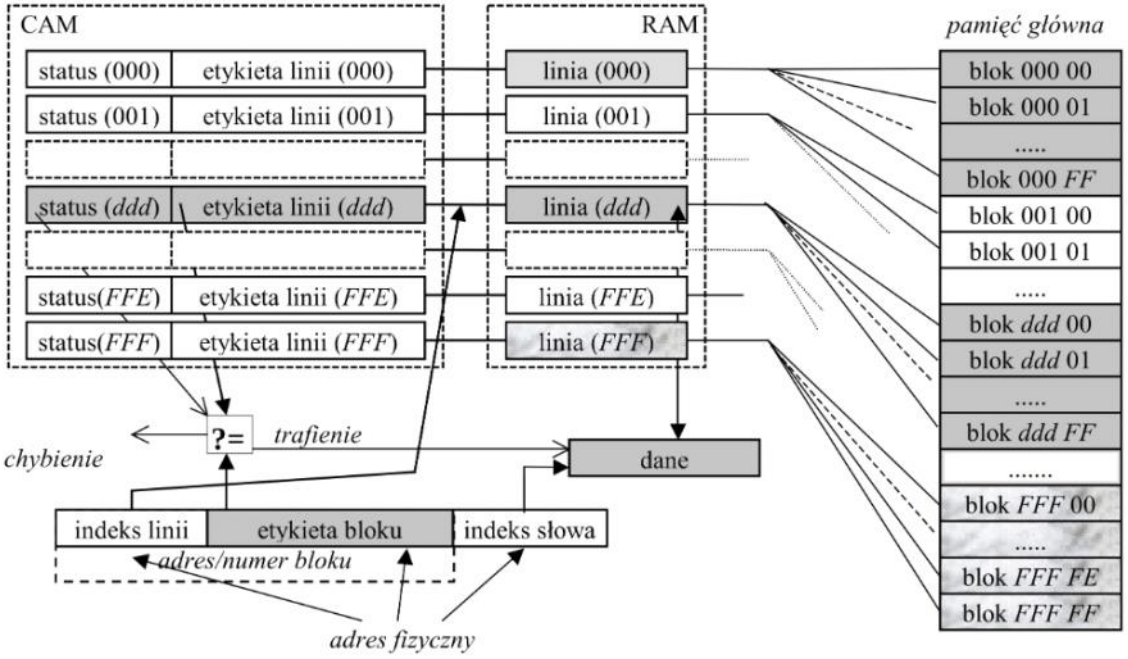
- Indeks bajtu w linii (indeks słowa)
- Indeks linii - to zależy od liczby linii w pamięci podręcznej
- Etykieta linii

Jeżeli procesor wytwarza jakiś adres i żąda dostępu do tej komórki o tym adresie to:

1. Etykieta jest porównywana z polem etykiety linii tylko dla tej jednej linii, która jest wybrana środkową częścią adresu (w ten sposób znacznie upraszamy układ porównujący) Czyli wybiearmy tylko te linie, które pasują do indeksy linii sprawdzając czy zgadza się etykieta. Wtedy wystarczy, że w każdym takim bloku sprawdzona zostanie jedna linia.

Ale co to powoduje:
Wszystkie odwołania do komórek pamięci, które mają tę część taką samą ,a tę inną trafiają ylko do tej jendej linii. Tak jak wyżej jest to schematycznie zaznaczone na obrazku. Najbardziej j widać beżowe, linie odległe od siebie o tyle czyli te, które mają etykiety równe zero, jeden, dwa, trzy, cztery, te linie pasują tylko do tej jednej linii. I jak napiszemy kod, w którym elementy listy będą tak duże, że wskaźnik będzie zawsze pod takim adresem to wszystkie odwołania do każdego elementu listy będą powodowały, że będziemy w tej linii mieli nieważne dane i będziemy musieli te dane wypisać z powrotem do pamięci odczytyać z pamięci nowe dane i tak za każdym razem.

Wersja bez przeplotu:

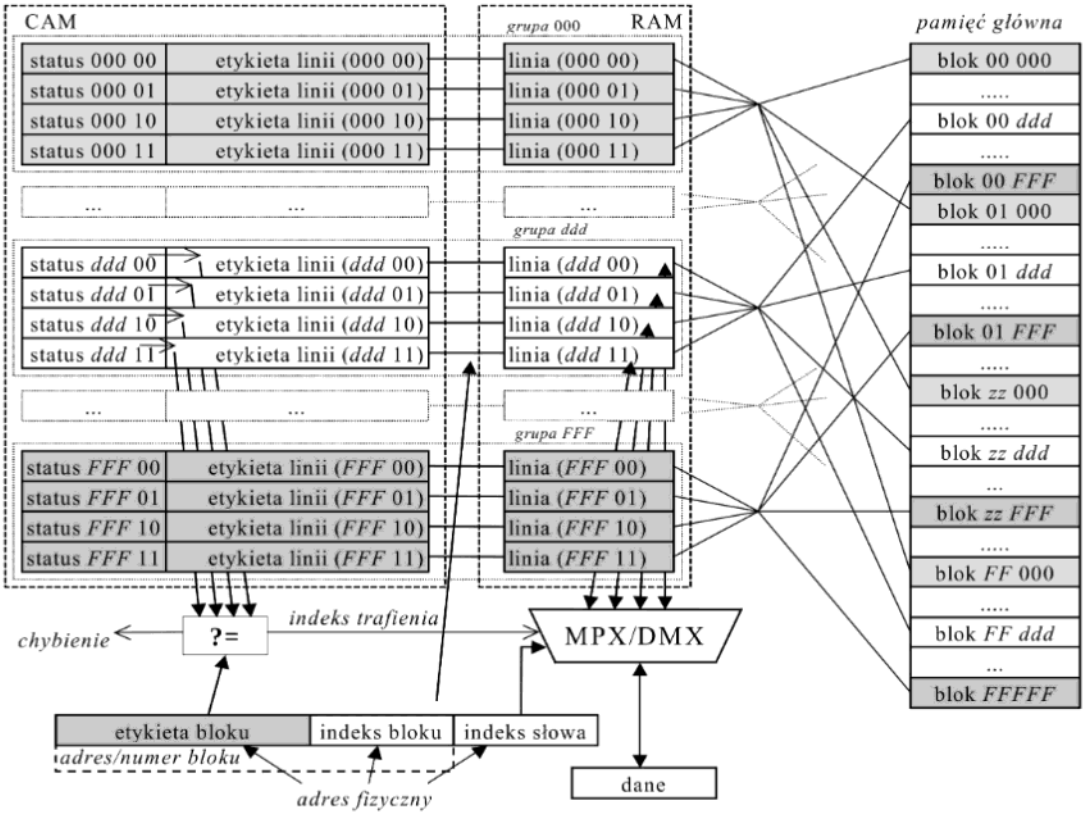


Odwzorowanie bezpośrednie bez przeplotu – bardzo częsty konflikt odwzorowania

Do jednej linii, np. 000 pasuje ciągły zestaw linii w pamięci operacyjnej.
Która wersja jest lepsza?
Z odwzorowaniem z przeplotem, zazwyczaj operujemy na danych bliskich sobie i w tej wersji dane bliskie sobie pasują do różnych linii!!!
Natomiast w tej wersji danych bloskich sobie zmieści się tylko jedna linia.
Więc takich z odzworowaniem bezpośrednim raczej się nie stosuje chyba, że do jakichś specjalnych zastosowań

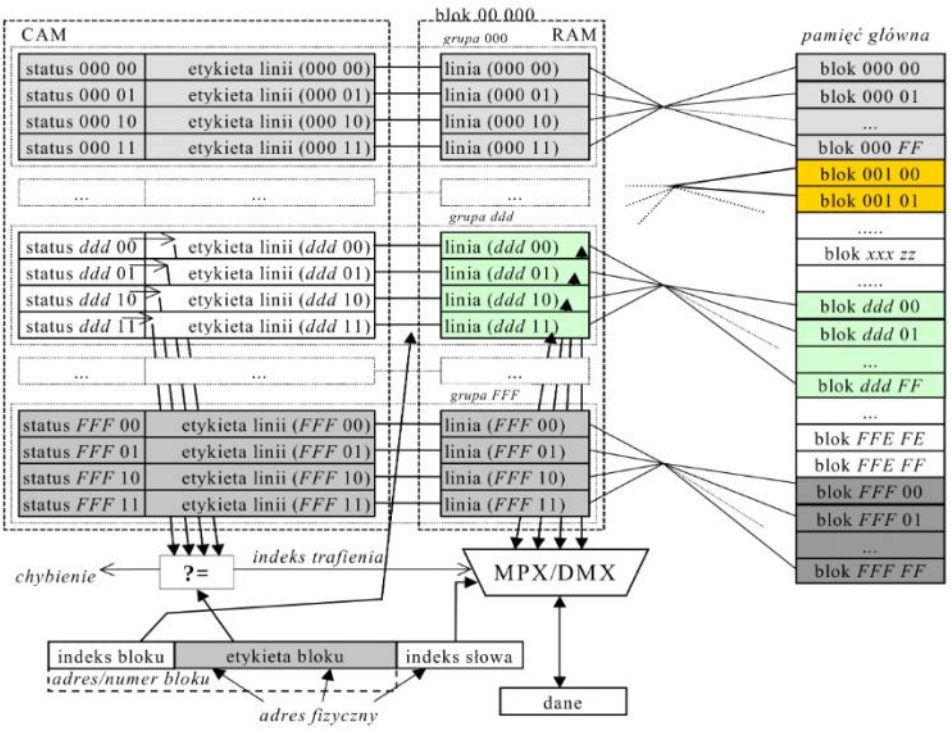
Pamięci całkowicie skojarzeniowe mają wiele zalet, ale też poważną wadę: bardzo duży koszt i są bardzo wolne .
Pamięci z odwzorowaniem bezpośrednim są szybkie i mają niski koszt, ale też całą masę wad.
W wersji bez przeplotu - będziemy mieli same chybienia jak będziemy operowali na danych bliskich sobie
W wersji z przeplotem - będziemy mieli same chybienia jak będziemy operowali na danych odległych dokładnie o rozmiar pamięci podręcznej - 16, 32 KiB, a to nie jest jakaś odległość kolosalna.

PAMIĘCI WIELODROŻNE:



Bufor grupowo-skojarzeniowy 4-drożny (4-*way set-associative*)
(uwaga: liczba linii w grupie nie musi być potęgą dwójki)

Złożenie kilka pamięci obok siebie, one wszystkie pracują naraz. Pamięć dwudrożna będzie miała dwa taki zestawy. Pamięć trzydrożna będzie miała trzy takie zestawy…
W tej chwili stosuje się pamięci wielodrożne z przeplotem i bez…



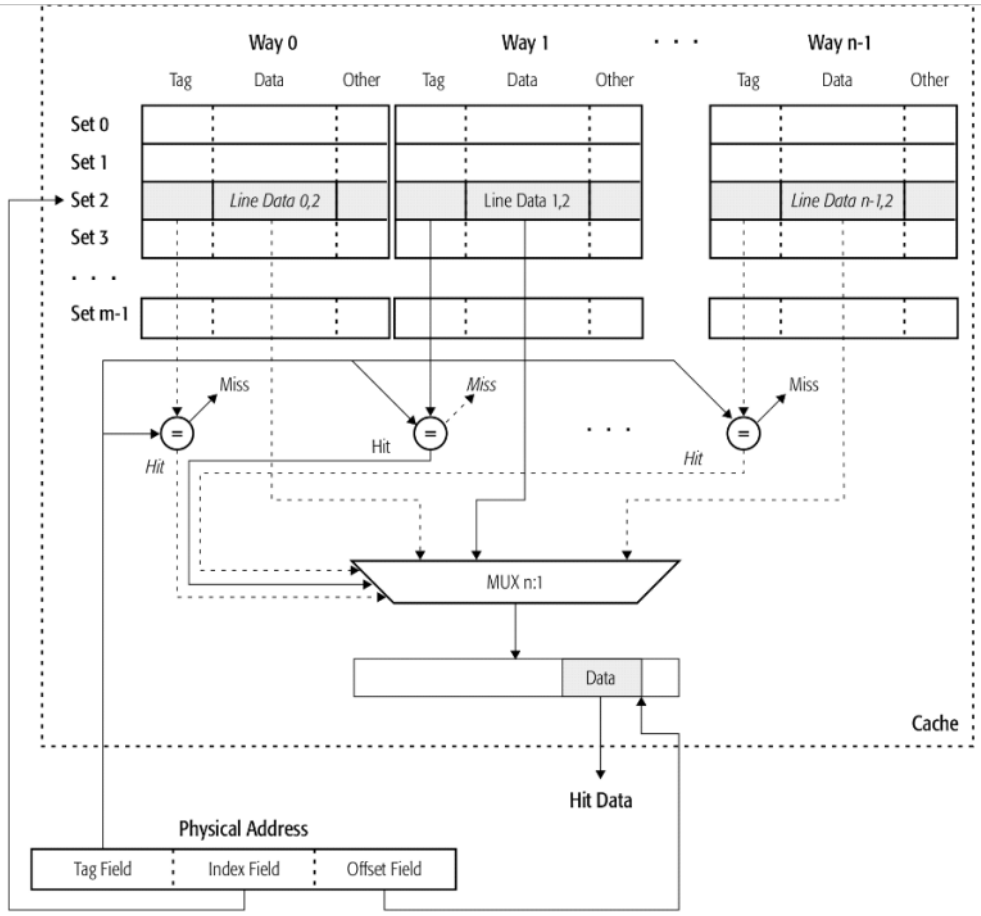
Odwzorowanie blokowo-skojarzeniowe bez przeplotu – duże ryzyko konfliktu

W pamięciach wielodrożnych jeśli mamy dwie drogi to jedna linia trafi do jednej z tych dróg,

a druga linia do drugiej z tych dróg i dopiero przy trzecim odwołaniu i dopiero przy trzecim odwołaniu do danej odległej o rozmiar pamięci będziemy mieli chybienie w którejś z tych dwóch. Ale w tej drugiej nadal będą przechowywane dane z tej linii.

Teraz standardem są pamięci 8 albo 16 drożne, liczbę linii wywołujących nawet konflikty, można znacznie zwiększyć.

Pamięci wielodrożne to po prostu złożenie wielu takich pamięci



27

To jest akurat AMD, jak widać mamy kolejne drogi, każda druga ma swój komparator i teraz muszą być oczywiście dodatkowe układy, które będą w stanie stwierdzić, w której z tych dróg mieliśmy trafienie, bo kopia naszych danych będzie przechowywana wtedy tylko w jednej z dróg i tylko jedna z tych dróg wystawi sygnał, że jest trafienie i wtedy trzeba wiedzieć, z której drogi należy dane przesłać do procesora. Ale co do zasady jest po prostu złożenie pamięci z odwzorowaniem bezpośrednim

Jeszcze kilka ważnych terminów:

II. Unieważnianie linii- Każda linia pamięci podręcznej może być w stanie nieważnym :
a) może być skutkiem albo włączenia procesora, wtedy z definicji wszystkie linie muszą być nieważne
b) może być skutkiem tego, że zmieniły się dane w pamięci operacyjnej np. na skutek zapisu DMA, jeśli układ DMA zapisał dane do pamięci operacyjnej, a my mamy kopię tego obszaru w pamięci podręcznej to układ DMA nie zapisuje do pamięci podręcznej, wtedy musi być przeprowadzone unieważnienie linii, to jest bardzo ważny termin
c) Kiedy na procesorze działa jakiś program i uruchamiany jest nowy program też trzeba unieważnić pamięć podręczną chociażby ze względów bezpieczeństwa, bo są ataki na pamięć podręczną, chociażby obserwując czas dostępu do komórki pamięci wiedząc czy się coś zmieniło w tej pamięci czy nie

III. Wypełnianie linii - wpisanie do linii kopii danych, procesor próbuje odczytać dane, których nie ma w pamięci podręcznej, te dane są przesyłane z pamięci operacyjnej i przy okazji wpisywane do linii

IV. Wymiana linii - jeżeli potrzebne są dane, których nie ma w pamięci podręcznej, ale w pamięci podręcznej w linii gdzie te dane miałyby być zapisane są ważne dane, wtedy trzeba te linie najpierw opróżnić i teraz:
a)dane są niezmienioną kopią z pamięci operacyjnej - wystarczy opróżnić
b)kopia była modyfikowana - trzeba zapisać z powrotem, żeby nie zginęło i dopiero potem można opróżnić linię

V. Trafienie przy odczycie - jeśli chcemy odczytać dane i ich kopia jest w pamięci podręcznej

VI. Trafienie przy zapisie - w pamięci podręcznej mamy dane z pamięci operacyjnej i procesor wykonuje zapis :

a) Scenariusz 1: Zapisujemy do pamięci operacyjnej na wylot, żeby mieć od razu tam ważne dane, przy okazji możemy zapisać też do pamięci podręcznej i wtedy mówimy o zapisie skrośnym
b) Scenariusz 2: Zapisujemy tylko do pamięci podręcznej - wtedy musimy pamiętać, że w linii pamięci podręcznej są dane zmodyfikowane i gdy będziemy tę linie opróżniali musimy wtedy zapisać ją do pamięci operacyjnej, albo gdy ktoś inny będzie chciał te dane odczytać z pamięci operacyjnej i albo ten ktoś będzie wiedział, że my w pamięci podręcznej mamy nowszą wersję danych albo nie będzie wiedział i to wtedy trzeba załatwiać programowo na etapie algorytmu.

VII. Chybienie podczas zapisu - jeśli w pamięci podręcznej nie mamy kopii danych, do których chcemy pisać możemy rozważać podejście, w którym nie będziemy pisali do pamięci podręcznej tylko zapiszemy wyłącznie do pamięci operacyjnej, a pamięci podręcznej nie dotykamy.
Przykład: Chcemy zmienić jeden bajt. Gdyby ten bajt chcieć zmienić w linii pamięci podręcznej to:
a)przesyłamy całą linię z pamięci głównej do podręcznej
b)zmieniamy ten jeden bajt
c)całą linię zapisujemy z powrotem
DUŻO RUCHU NAMAGISTRALI!
Jeśli zapiszemy bezpośrednio nie musimy robić odczytu.
I tu są różne tryby pracy.

SĄ RÓŻNE TECHNIKI WYBIERANIA SPOSÓBU ZAPISU DO PAMIĘCI!

ZASTOSOWANIE INSTRUKCJI STRUMIENIOWEJ

Przykładowy program w c pod clangiem

Po lewej jest zwykły kod, który stosuje do nanoodczytu i operacji na pamięci i tutaj trzeba wykonać dwa odczyty jeden zapis. Czas transferu wychodzi 34GB/s.
Pp zmmianie jednej instrukcji na instrukcje non-temporal czas odczytu spadł o 1,5 raza.
Rzeczywista strumieniowość zwiększyła się o 1,5 raza.