ORGANIZACJA I ARCHITEKTURA KOMPUTERÓW

Badanie czasów dostępu do pamięci DDR4

Prowadzący: Dr inż. Tadeusz Tomczak

Czw TN 11:15

Marcin Waloch 241274

Mateusz Woźniak 241142

Spis treści

1. Wstęp teoretyczny

RAM jest pamięcią szybkiego dostępu wykorzystywaną przez procesor do przechowywania programów, instrukcji oraz danych. W ramach projektu badamy zjawiska struktury pamięci dla DDR 4 SDRAM(Double Data Rate Synchronous Dynamic Random Access Memory (version 4)). Dynamiczność oznacza wymaganie odświeżania segmentów w regularnych odstępach czasu ze względu na rozładowywanie się kondensatorów odpowiadających za przechowywanie danych.

Aby móc prawidłowo przeprowadzić pomiary czasów dostępu do RAM należy zapełnić pamięć podręczną procesora, która zawsze wczytuje całą linie(64 B) z pamięci głównej zawierającą ostatnio odczytywany adres pamięci głównej. Zdecydowaliśmy się na zapełnianie całej pamięci podręcznej procesora, ponieważ nie wiedzieliśmy jak uzyskać dostęp do L2 przynależnego do rdzenia procesora oraz jak rozgraniczyć kiedy poszczególne segmenty są używane, cały zapełniany cache ma rozmiar 7,25 MB.

2. Cele i założenia projektu

Głównym celem projektu była analiza wybranego modelu pamięci DDR na podstawie jej dokumentacji, zaobserwowanie zjawisk działających na różnych strukturach owej pamięci oraz faktyczny pomiar i analiza tychże zjawisk.

Zostało to osiągnięte poprzez napisanie programu dokonującego czasu dostępu do poszczególnych struktur pamięci, analizę wyników pomiarów oraz porównanie ich do czasów dostępu deklarowanych przez producenta. Ostatnim etapem było wyciągnięcie wniosków na podstawie różnic pomiędzy osiągniętymi wynikami i czasami podanymi w dokumentacji.

Najważniejsze założenia projektu prezentują się następująco:

- Analizowana była pamięć *DDR4 2133MHz*, model *HMA451S6AFR8N-TFN0* firmy Hynix, który zbudowany jest z 8 układów *H5AN4G8NAFR-TFC*.
- Wszystkie dane losowe w projekcie wygenerowane zostały przy użyciu funkcji *rand().*
- Podstawową metodą uzyskiwania pomiarów czasów dostępu była procedura *rdtscp*.
- Wszystkie pomiary wykonane zostały wykonane na maksymalnej częstotliwości pracy procesora (wykorzystywany jest procesor Intel i7 6700HQ). Prędkość ta dla jednego rdzenia wynosi 3.5 GHz.Wykresy i statystyki zostały utworzone na podstawie 1000 pomiarów.

3. Etapy realizacji projektu

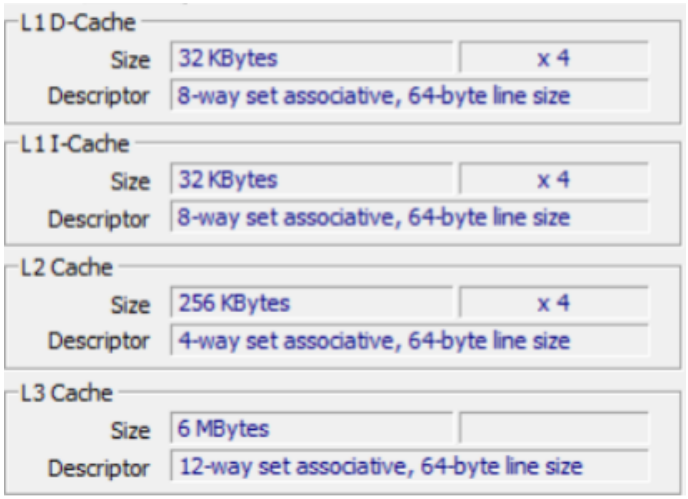
1. Pozyskanie dokumentacji pamięci na której przeprowadzane są badania

Ustalono model badanej pamięci DDR(HMA451S6AFR8N-TFN0, sk-hynix) oraz pozyskano jej dokumentacje. Następnie ustalono model układów elektronicznych z których zbudowana jest pamięć(H5AN4G8NAFR-TFC).

2. Porównanie danych wyczytanych z dokumentacji z danymi odczytywanymi z programu CPUID CPU-Z.

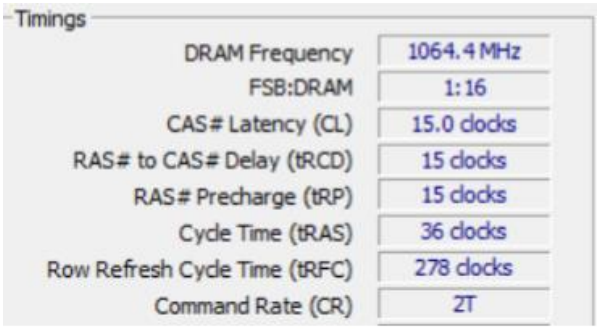
Szczególną uwagę zwróciliśmy na:

- wielkości pamięci cache procesor(pomiary programu zgodne z dokumentacją procesora):



Rysunek 1 Rozmiary pamięci podręcznej uzyskane za pomocą programu CPUID

- Rzeczywista częstotliwość pamięci DDR oraz ilość cyklów pamięci potrzebnych na zwrócenie zawartości pamięci pod zadanym adresem:



Rysunek 2 ilość cykli pamięci RAM wymaganych na dokonanie podanych operacji(np. Czas odpowiedzi), uzyskano za pomocą programu CPUID

Producent deklaruje czas opóźnienia adresu bramkowania(*CAS latency*) zgodny z wynikami działania *CPUID*.

Przeliczając na cykle procesora otrzymujemy w przybliżeniu **3.5/1.064 * 15 ~= 49 cykli procesora**

Chipset	Intel	Skylake-H	Rev.	07
Southbridge	Intel	Skylake-H PCH	Rev.	31

Rysunek 3 Model płyty głównej w jednostce pomiarowej, dane uzyskane przy pomocy programu CPUID

3. Badanie opóźnień wynikających z użycia MCU

Nie byliśmy w stanie pozyskać informacji na temat opóźnienia wynikającego z przetwarzania zapytań procesora do pamięci głównej poprzez *MCU(memory control unit)*. Na podstawie informacji z dokumentacji o wykorzystywanym standardzie - *Unbuffered Small Outline DIMM* oraz wiarygodnego artykułu[2], którego autor twierdzi że MCU dla tego typu układu nie wnosi żadnych znaczących narzutów, możemy przyjąć że owe obciążenia nie wnoszą narzutu większego niż rząd wielkości i wraz ze zwiększeniem się czasów odpowiedzi modułu pamięci zmniejszają swój stosunkowy narzut.

4. Zdefiniowanie sposobu przeprowadzenia testów

Jako najważniejsze narzędzie do testów wybraliśmy aplikację typu *pointer-chaser,* która to polega na badaniu czasu wykonania przejścia poprzez łańcuch połączonych ze sobą wskaźników. Takie rozwiązanie pozwala nam na w miarę możliwości nieskomplikowane operowanie na dystansie, jaki chcemy przebyć w pamięci, oraz na rozmiarze obszaru pamięci, na którym chcemy operować.

5. Wybranie sposobu pomiaru wykonywanych operacji(czasu dostępu)

Po wstępnej analizie pomiarów z wykorzystaniem biblioteki *std::chrono* czy też *QueryPerformanceCounter* odrzuciliśmy wspomniane narzędzia, ze względu na duży narzut wnoszony na pomiary. Za najlepsze rozwiązanie uznaliśmy zastosowanie procedury *rdtscp* stosowanej wprost z poziomu assemblera.1

6. Odczytanie z dokumentacji wielkości struktur pamięci

Na podstawie dokumentacji odczytaliśmy dokładną hierarchię struktur pamięci i wyznaczyliśmy ich rozmiary potrzebne w dalszych pomiarach.

Configuration		**512Mb x 8**
Bank Address	# of Bank Groups	4
	BG Address	BG0~BG1
	Bank Address in a BG	BA0~BA1
Row Address		A0~A14
Column Address		A0~ A9
Page size		1 KB

Rysunek 4 Konfiguracja i hierarchia struktur pamięci w badanym modelu pamięci

Na podstawie czego mogliśmy wyznaczyć następujące wielkości:

- Kolumna wskazuje na pojedyncze słowo 8-bitowe (1 bajt), czyli na najmniejszą adresowalną jednostkę informacji pamięci komputerowej,
- 1024 kolumn (*2^10*) składa się na jeden wiersz, a więc jest on rozmiarów *1KB*
- 32768 wierszy (*2^15*) składa się na jeden bank, który zatem ma rozmiar *32MB* (*2^25B*),
- 4 banki (*2^2*) składają się na grupę banków, jej rozmiar to *128MB* (*2^27B*)
- 4 grupy banków (*2^2*) składają się na moduł pamięci o rozmiarze *512MB* (*2^29B*)
- I na szczycie hierarchii strukturalnej jest cała kość pamięci złożona z 8 takich modułów, jej całkowity rozmiar to *4GB* (*2^32B*).

7. Pierwsze próby wytworzenie kodu

Nasz kod w początkowych fazach projektu pozwalał na utworzenie łańcucha wskaźników o zadanych parametrach takich jak odległość pomiędzy wskaźnikami oraz obszar zajmowanej przez łańcuch pamięci. Jednakże już to pozwoliło nam odkryć pierwsze zależności chociażby pomiędzy granicami obszaru pamięci podręcznej a czasem przejścia przez dany łańcuch. Zaobserwowaliśmy także zależność pomiędzy odległością pomiędzy poszczególnymi wskaźnikami (ang. *Stride),* a czasem przejścia.

8. Dodanie funkcji zapełniającej cache i ponowna analiza

Bardzo szybko przekonaliśmy się o potrzebie wykonania osobnej funkcji zapełniającej pamięć podręczna komputera, ze względu na wpływ pamięci podręcznej na nasze pomiary. Zrealizowane zostało to za pomocą wygenerowania bufora pamięci o rozmiarze pamięci podręcznej z pewnym nadmiarem, oraz przejściem poprzez ów bufor za pomocą łańcucha wskaźników.

9. Analiza wygenerowanego kodu assemblerowego

Ten właśnie etap pozwolił nam się zorientować jak duże mogą być narzuty spowodowane poprzez pisanie w języku programowania wysokopoziomowego czy też poprzez wykorzystywanie funkcji którego działania do końca nie rozumiemy. Okazało się, że część kodu która mierzyliśmy, była znacznie większa niż byśmy się tego spodziewali.

10. Zminimalizowanie ilości komend asemblerowych

Kolejnym i właściwie kluczowym etapem było sprowadzenie obecnego kodu pomiarowego do formy w której mieliśmy już pewność jakie dokładnie działania są mierzone. Co ciekawe wyniki uzyskane po tej zmianie nie odbiegały znacząco od wyników uzyskiwanych już wcześniej, co sugerowało wykonywanie większości operacji spowodowanych przez narzut w pamięci podręcznej. Zadbaliśmy także, aby kod w formie wstawki assemblerowej był zgodny z ABI systemu Windows[3].

```
/*ABI WINDOWSOWE WYMAGA ZACHOWANIA:
        RDI/EDI         - Zachowany
        RSI/ESI         - Zachowany
        RBX/EBX
        RBP/EBP

W poniższym ciągu instrukcji modyfikowane są wyłącznie rejestry:
        EAX,EDX i ECX   przez instrukcje rdtscp
        EDI             jako miejsce przechowywania adresu pamieci
*/
__asm {
    push esi
    push edi

    mov edi, dword ptr[ptr2]
    rdtscp
    mov dword ptr[t1], eax
    mov edi, dword ptr[edi]
    rdtscp
    mov dword ptr[t2], eax

    pop edi
    pop esi
}
```

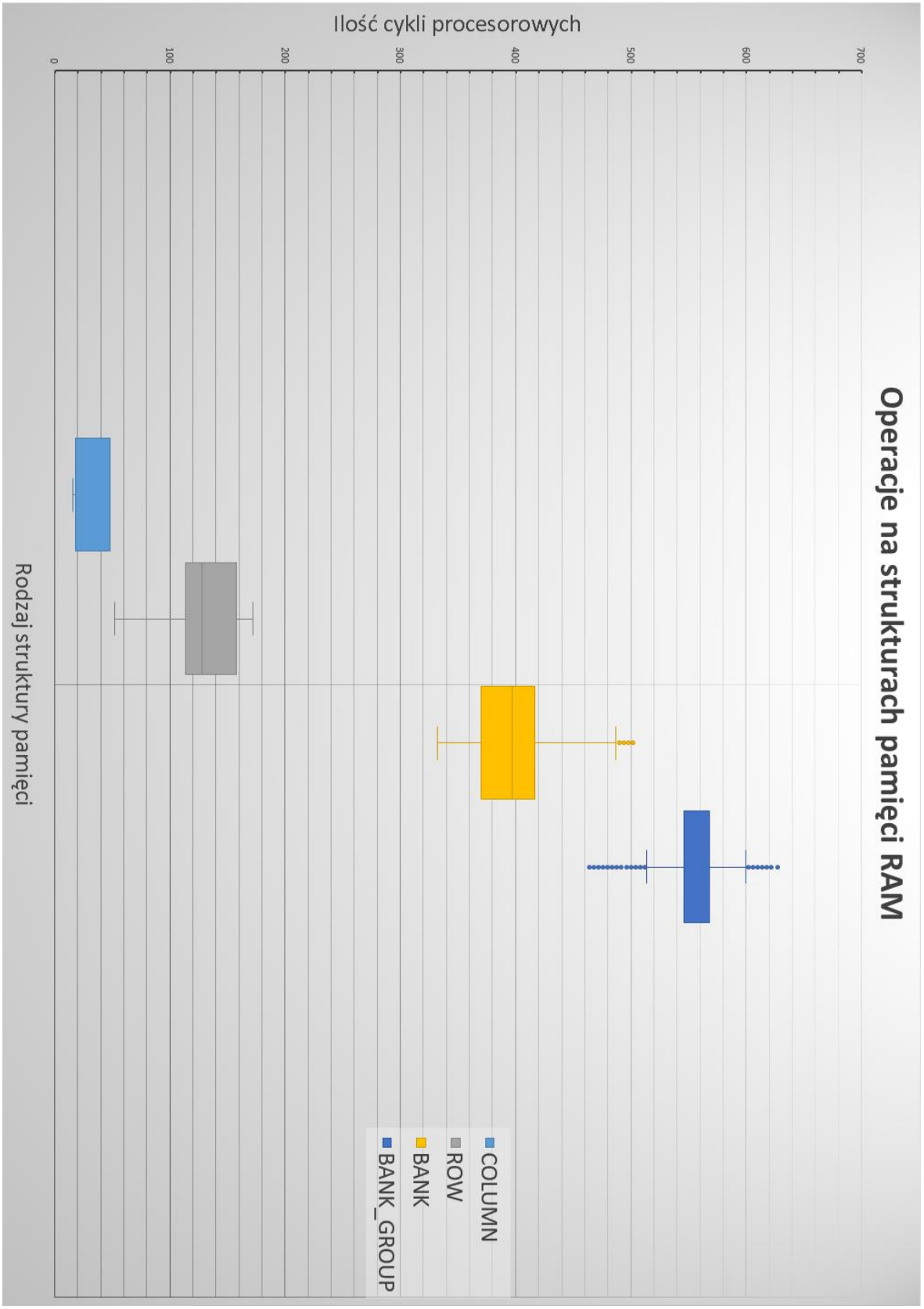
Rysunek 5 Końcowa wersja części kodu odpowiedzialnego za faktyczny pomiar

11. Analiza i obróbka uzyskanych danych

Poprzez wykonanie pomiarów na określonych strukturach parę setek razy uzyskaliśmy potrzebne nam dane. Na ich podstawie sporządziliśmy wykres typu boxplot przedstawiający różnice w czasach dostępu do różnych struktur pamięci.

12. Sformułowanie wniosków oraz spostrzeżeń na podstawie uzyskanych danych

Ostatnim etapem było porównanie uzyskanych pomiarów z czasami podanymi przez producenta pamięci oraz wyciągniecie wniosków.



Rysunek 6 Wykres zależności zbadanego czasu dostępu od wymuszenia zmiany konkretnej struktury pamięci

4. Uzyskane wyniki

Na wykresie możemy zaobserwować ewidentną zależność pomiędzy rodzajem struktury (czyli także jej rozmiarem), a czasem dostępu do tejże właśnie struktury pamięci. Bardzo ważnym było uzmysłowienie sobie dokładnej hierarchii i budowy układu pamięci, tak aby dobrać właściwe wielkości testowe, które powodowały wymuszenie konkretnej struktury pamięci.

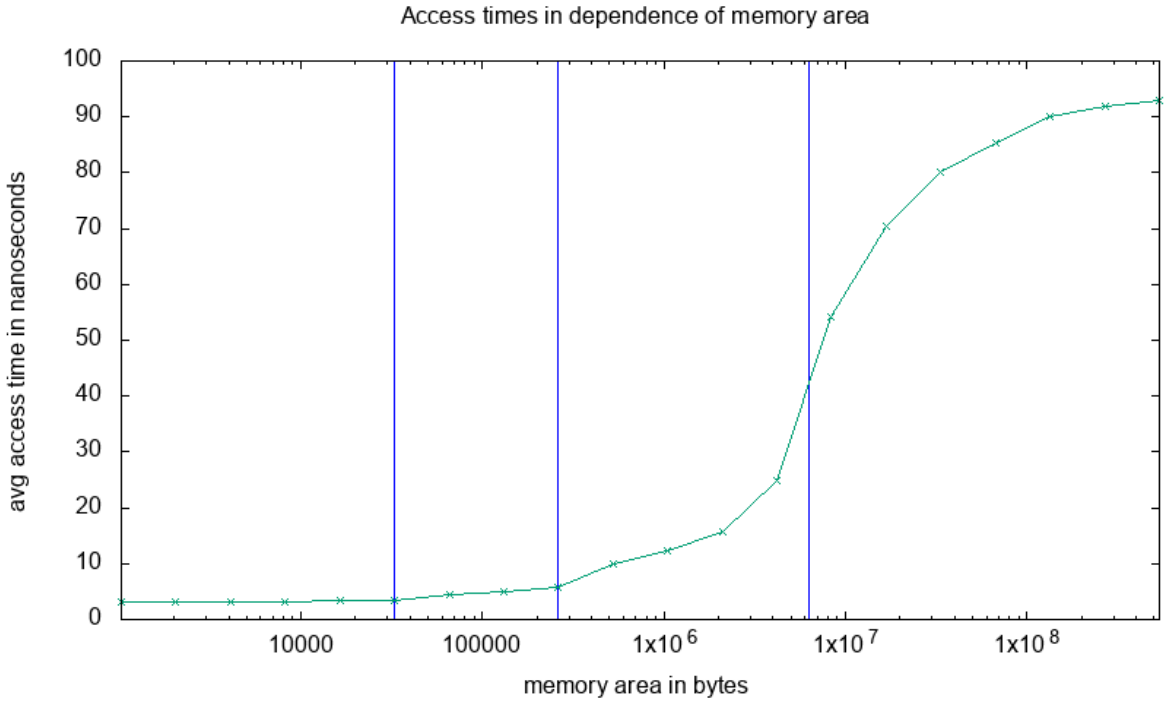
Pierwszą wielkością którą zmierzyliśmy był czas dostępu do kolumny. Wielkość podana w dokumentacji tzw. *CAS latency* (ang. *Column address strobe latency*) wynosi **około 49 cykli zegarowych**. Natomiast pomiary, które my uzyskaliśmy, zawierają się w przedziale **[18;50] cykli zegarowych,** a średnia pomiarów wynosi **42 cykli zegarowych**. Łatwo zauważyć, iż nasze pomiary są nie do końca pokrywają się z danymi podanymi przez producenta. Spodziewalibyśmy się raczej zawyżonych pomiarów.

Następna badana strukturą był wiersz. Z dokumentacji wiemy, iż *RAS* (ang. *Row Address Strobe*) wynosi w przybliżeniu **116 cykli**. Tutaj już uzyskaliśmy pomiary większe niż czas zadeklarowany w dokumentacji. Przeważająca część pomiarów umiejscowiona jest w zakresie **[114;172] cykli zegarowych.** Ich średnia wynosi **130 cykli**, a więc występuje tutaj już pewne przebicie.

Kolejną strukturą pamięci, której zmianę wymuszaliśmy, były banki. Czasy wynikające z dokumentacji sugerują około **165** cykli zegarowych procesora. Uzyskane przez nas pomiary są co najmniej 2 razy większe. Tutaj zakres naszych pomiarów wynosi **[343;504] cykli**, a średnia owych pomiarów wynosi około **397 cykli.**

Ostatnia i największa strukturą pamięci na której operowaliśmy, była grupa banków. Niestety nie byliśmy w stanie znaleźć odpowiednich informacji na temat czasu dostępu do pamięci w innej grupie banków. Pomiary mieszczą się tutaj w zakresie **[464;625] cykli.** Ich średnia wynosi **546 cykli**.

W początkowych fazach projektu udało nam się także zaobserwować takie zjawiska jak wpływ pamięci podręcznej na czas dostępu do danych czy też prefetching danych w tak zwanym *strided memory access.*



Rysunek 7 Wykres zależności od badanego obszaru i czasu dostępu z zaznaczonymi obszarami pamięci podręcznej L1, L2 i L3

Na wykresie możemy zauważyć nagły skok czasów dostępu w momencie wykroczenia poza obszar pamięci podręcznej. Pozwoliło to nam na określenie momentu w którym pamięć podręczna przestaje mieć tak wielki wpływ na pomiary w efekcie czego mogliśmy już przejść do faktycznego scenariusza testowego związanego z projektem.

5. Wnioski i spostrzeżenia

- Pomiary wskazują na prawidłowe zależności pomiędzy czasem dostępu a wymuszaniem konkretnej struktury pamięci. Jednakże są one parokrotnie większe niż czasy dostępu podane przez producenta. Wiąże się to z narzutem wprowadzonym przez języki programowania wysokiego poziomu, pewną niedokładnością narzędzi pomiarowych oraz najprawdopodobniej niedopracowanym scenariuszem pomiarowym.
- Czas i wysiłek jaki poświęciliśmy na minimalizowaniu kodu napisanego w języku C++, tak aby kod zdeassemblowany odpowiadał naszym oczekiwaniom i wyobrażeniom, uzmysłowiły nam jak wiele rzeczy się dzieje, niewidocznych na pierwszy rzut oka, podczas korzystania z języków programowania wysokiego poziomu. (A także jak wiele rzeczy może się tam jeszcze kryć). Pozwoli to nam na ostrożniejsze posługiwanie się owymi językami w przyszlości.
- Struktura pamięci w naszym wyobrażeniu nie była szczególnie skomplikowana, jednakże mnogość różnych procesów, jakie mogą się na owej pamięci odbywać (jak chociażby zjawisko prefetching), pokazały nam jak skomplikowane potrafią być układy pamięci zwłaszcza w obecnych czasach.
- Na przyszłość prawdopodobnie warto jasno określić cel programu oraz scenariusz testowy, tak aby wszelka praca włożona w napisanie kodu nie zakończyła się napisaniem kodu spełniającego kompletnie inną funkcję.

Bibliografia

[1] https://en.wikichip.org/wiki/intel/core_i7/i7-6700hq, potwierdzone przy pomocy programu cpu-z

[2] https://www.atpinc.com/blog/computer-memory-types-dram-ram-module

[3] https://docs.microsoft.com/pl-pl/cpp/build/x64-software-conventions?view=vs-2019