Architektura procesora a wydajność

$$\text{Performance} = \frac{1}{\text{instruction count}} \times \frac{\text{instructions}}{\text{cycle}} \times \frac{1}{\text{cycle time}} = \frac{\text{IPC} \times \text{frequency}}{\text{instruction count}} \quad (1.2)$$

W jaki sposób architektura procesora wpływa na wydajność?

- zbiór instrukcji, który dobrze zaimplementowany może zmniejszyć **LICZBE INSTRUKCJI.**
- **IPC** – im więcej procesor jest w stanie przetworzyć instrukcji w cyklu tym większą wydajność
- **DLUGOSC CYKLU** – Im krótszy cykl tym większa przepustowość

Przypomnienie o architekturze skalarnej

Architektura skalarna składa się z jednego potoku z k poziomami, który każda instrukcja niezależnie od jej rodzaju musi przejść.

Ograniczenia:

- W każdej chwili maksymalnie jedna instrukcja może przebywać na każdym poziomie pipa, więc przepustowość nie ma prawa przekroczyć jednej instrukcji na cykl.
- Jeden potok dla każdego rodzaju instrukcji – nieefektywne.
- Sztywne ustawienie instrukcji doprowadza niepotrzebnych przerw w potoku spowodowanych hazardami.

Przypomnienie o architekturze skalarnej

Architektura skalarna składa się z jednego potoku z k poziomami, który każda instrukcja niezależnie od jej rodzaju musi przejść.

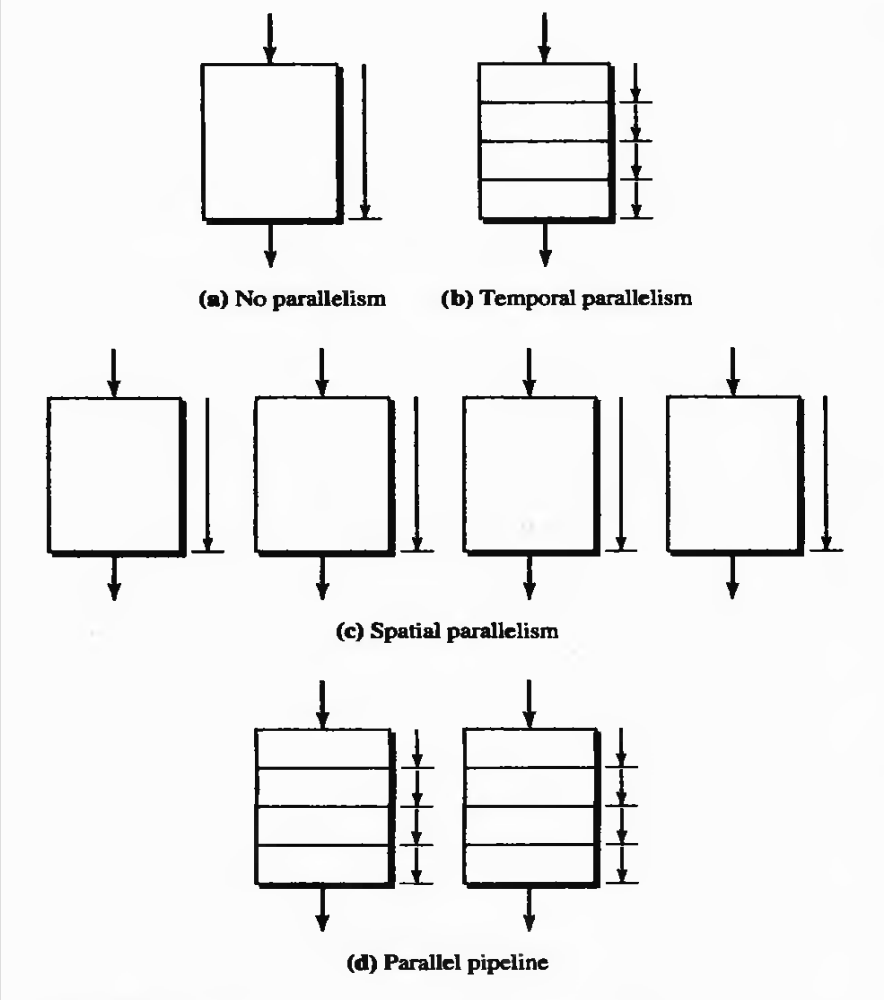
Ograniczenia:

- W każdej chwili maksymalnie jedna instrukcja może przebywać na każdym poziomie pipa, więc przepustowość nie ma prawa przekroczyć jednej instrukcji na cykl.
- Jeden potok dla każdego rodzaju instrukcji – nieefektywne.
- Sztywne ustawienie instrukcji doprowadza niepotrzebnych przerw w potoku spowodowanych hazardami.

Możliwe ulepszenia

- Głębszy stos
  + Mniej skomplikowane fazy – krótszy cykl.
  -Więcej faz – bardziej dotkliwe kary (i.e. Branch penalty)
- **Dynamiczny paralelizm w ramach jednego rdzenia**
  +Możliwość skonstruowania różnych jednostek wykonawczych(Brak kosztownej unifikacji)
  +Możliwość wykonywania więcej niż jednej instrukcji na każdym poziomie potoku
  -Bardziej skomplikowany hardware
- Pełna wielowątkowość
  -Konieczna duplikacja wszystkich podzespołów
  -Problemy z równoległym dostępem do pamięci

Porównanie potoków w różnych architekturach



- Jak widzicie -nie widzicie tutaj jeszcze architektury superskalarnej (suspens)

Co to właściwie jest superskalarność?

Procesory superskalarne stoją w połowie drogi między zwykłymi skalarnymi architekturami, a pełną równoległością (Rdzenie współczesnych procesorów wielowątkowych są w gruncie rzeczy superskalarne)

- Architektura procesora superskalarnego składa się z wielu potoków.
- W każdym potoku może być wykonywane instrukcje równolegle z innymi potokami. Procesor o szerokości K-pipe'ów może wykonywać do K instrukcji na jednym poziomie.
- Jednostki EXE w potokach mogą różnić się między sobą przeznaczeniem.

Tak to wygląda

- a) Procesor skalarny z pojedynczym potokiem

- b) Superskalarność -2 potoki (Intel Pentium)1993 Uważany za pierwszy procesor superskalarny.
  - U-pipe stałoprzecinkowe
  - V-pipe zmienno-przecinkowe

Co to właściwie jest superskalarność?(cd)

Procesory superskalarne zostały zaprojektowane z myślą o tym żeby poradzić sobie z ograniczeniami skalarnych architektur (slajd 2)

- Różne jednostki wykonawcze wyspecjalizowane do wykonywania określonych instrukcji (efektywne)
- Potrafią wykonywać więcej niż jedną instrukcję w tej samej fazie potoku w jednym cyklu
- Out-of-order

Początki (różne jednostki wykonujące)

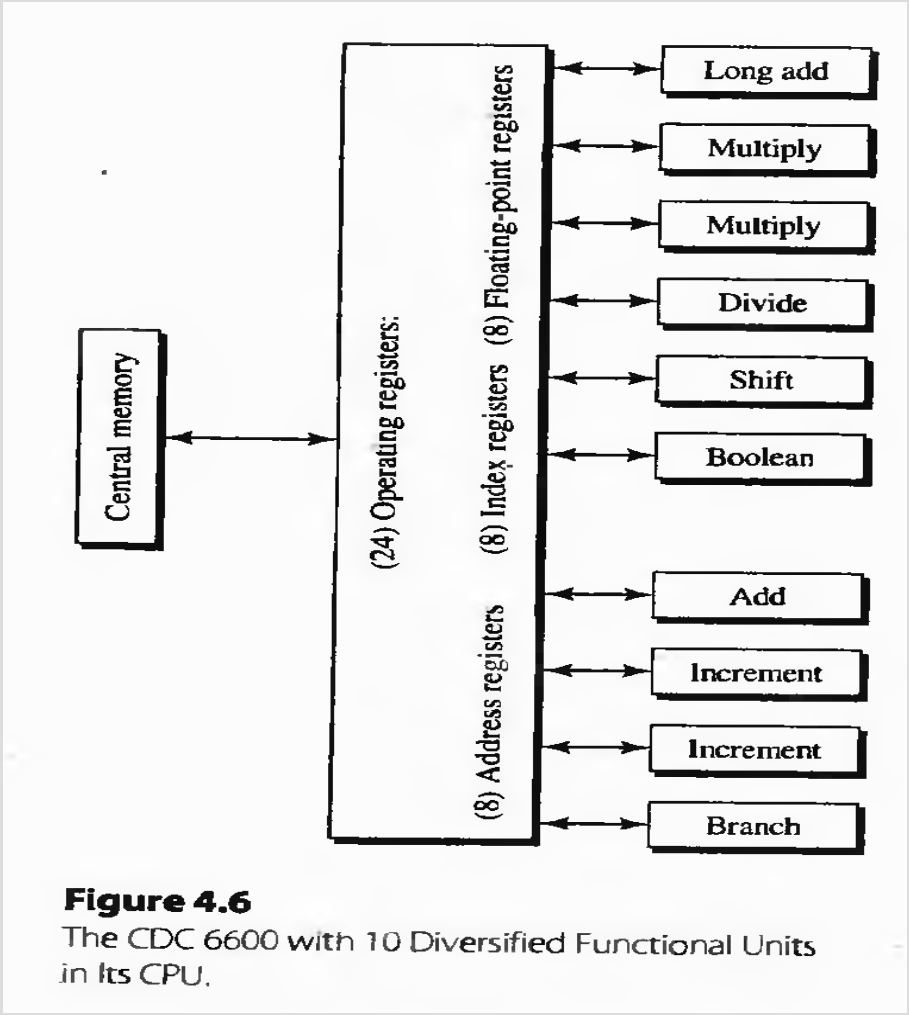


Figure 4.6
The CDC 6600 with 10 Diversified Functional Units in Its CPU.

CDC 6600 – jeszcze nie (jeszcze długo nie) procesor superskalarny. 1963

- Posiada jeden potok
- Różne, niezależne jednostki wykonywające
- Pierwszy superkomputer

(Audience)Dlaczego nie superskalarny?

Superskalarność – krok naprzód

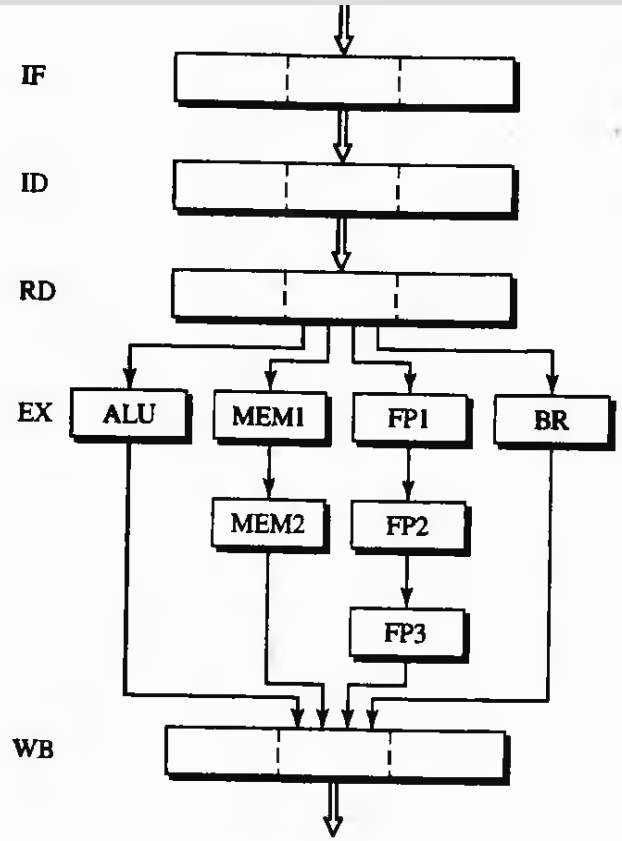


Figure 4.5
A Diversified Parallel Pipeline with Four Execution Pipes.

Pierwszą generację współczesnych procesorów skalarnych już widzieliśmy (6 slajd)

Schemat bardziej rozbudowanej architektury superskalarnej – 4 różne rodzaje potoków. Zwróćcie uwagę na różną głębokość EX w różnych potokach (1 poziom – 1 cykl)

ALU – stałopozycyjne,
MEM – dostęp do pamięci,
FP -zmiennopoycyjne,
BR - skoki

Superskalarność - słabostki

Dla procesora o s potokach

- Logika potencjalnie wzrasta s-krotnie.
- Potencjalnie potrzebne s^2 połączeń pomiędzy buforami potoków.
- Czas działania jednostek wykonywalnych dostosowany do najdłużej pracującej jednostki EXE (naprawialne)

Różne opóźnienia jednostek wykonywających - problem

W naiwnym podejściu do superskalarności instrukcje wykonywane równolegle powinny równocześnie kończyć swoje działanie (i.e. ALU wykonywane współbieżnie z FP powinno czekać 2 cykle) – nieefektywne

Rozwiązanie:

- Out-of-order, dynamic pipelines
- Multi-entry buffers with reordering(potrzebne do zaimplementowania out-of-order)

Single-entry buffers – jakie są każdy widzi. Multiple-entry buffer

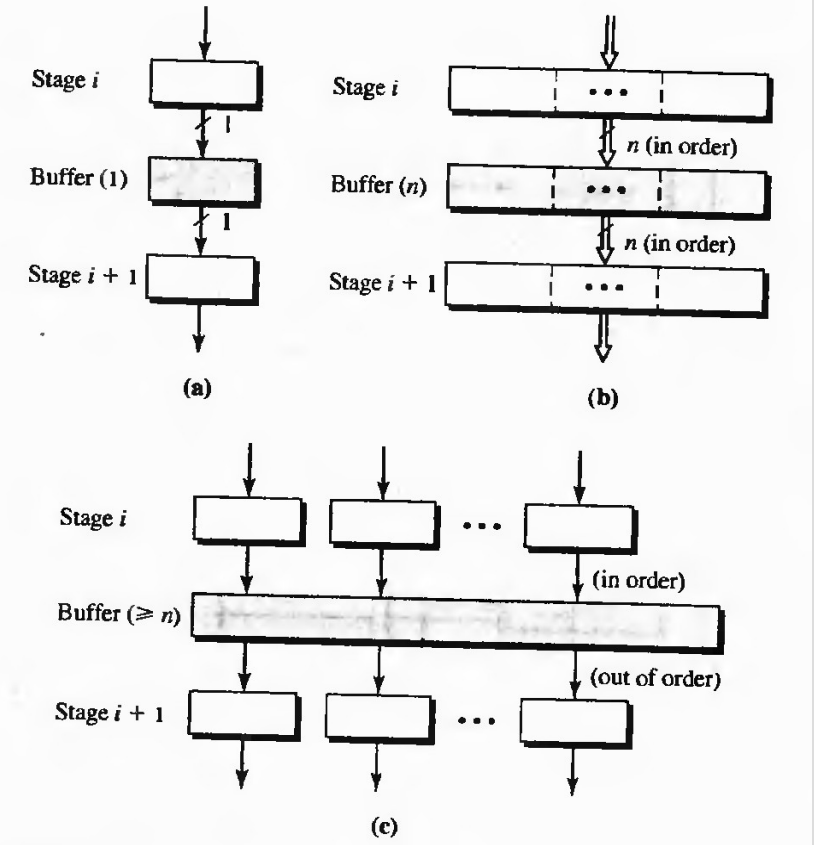


Figure 4.8
Interpipeline-Stage Buffers: (a) Single-Entry Buffer; (b) Multientry Buffer; (c) Multientry Buffer with Reordering.

W zwykłych procesorach potokowych bufory znajdują się pomiędzy wszystkimi fazami potoku. Działają na zasadzie zatrzasków – wypuszczając wyjście jednej fazy na wejście drugiej w momencie przejście instrukcji. Multiple-entry bufer działa podobnie z tym że mają więcej pól

Multi-entry buffer with reordering

Chcąc umożliwić procesorowi przetwarzanie instrukcji w bardziej wyrafinowany i efektywniejszy sposób niż blokowanie krótkich instrukcji, które wykonują się równocześnie z dłuższymi na poziomie EXE, potrzebujemy takiej struktury bufora która pozwoli nam na częściową wymianę jego wpisów.

Multi-entry buffer with reordering cd.

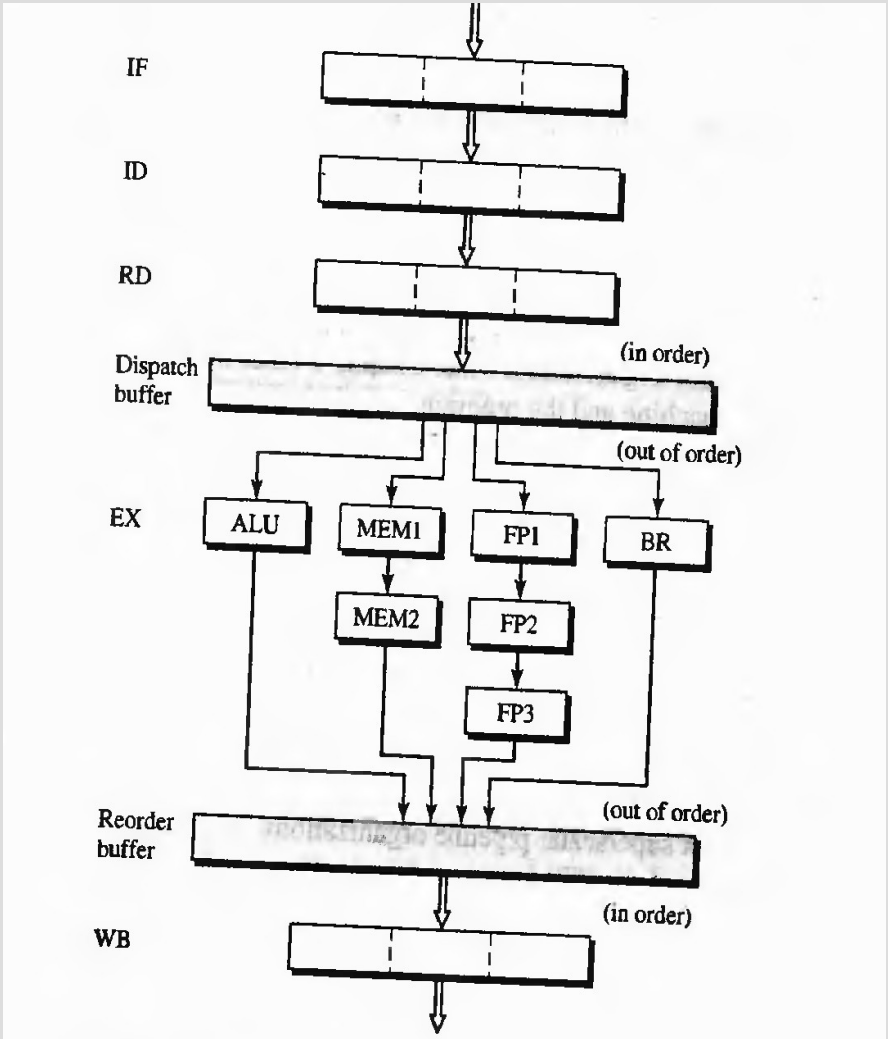
- Jednym z potrzebnych ulepszeń jest umożliwienie połączenia pomiędzy różnymi wejściami buforów (**chyba w celu forwardingu, ale nie wiem)**. Można to zrealizować przez połączenie pól w łańcuch i nadanie buforom funkcjonalności kolejki

- Kolejnym niezbędnym ulepszeniem jest bezpośrednie adresowanie pól w buforze. Dzięki temu procesor otrzyma możliwość niezależnego dostępu do pól buforów przechowujących informacje z różnych pipe'ów.
  Taki buffor może być zaimplementowany jak nieduża wieloportowa pamięć ram, albo pamięć asocjacyjna z dodanym tagiem.

Dynamiczny potok

Mając do dyspozycji bufory które umożliwiają niezależny i bezpośredni dostęp do pól różnych potoków możemy przystąpić do konstruowania potoku który będzie przetwarzał instrukcje niekoniecznie w takiej kolejności w jakiej do niego dotarły, optymalizując przepustowość procesora. Nazywa się to dynamicznym potokowaniem.

Dynamiczny potok ciąg dalszy



Ważnym elementem dynamicznego potoku są bufory przed i po jednostkach wykonujących. Dbają o to żeby zachować kolejność instrukcji do zapis, jednocześnie optymalizując przepustowość.

.

Dynamiczny potok – bufory.

- Dispatch buffor – zostaje zapełniony zdekodowanymi instrukcjami i przekazuje je dalej na odpowiednią jednostkę wykonującą. Przekazuje instrukcje niekoniecznie w takiej kolejności w jakiej je otrzymał. Optymalizuje kolejność.

- Reorder buffer – wypełniany efektami działania jednostek wykonujących, odpowiada za przywrócenie wartości które będą zapisywane do stanu sprzed re-orderingu

Potok

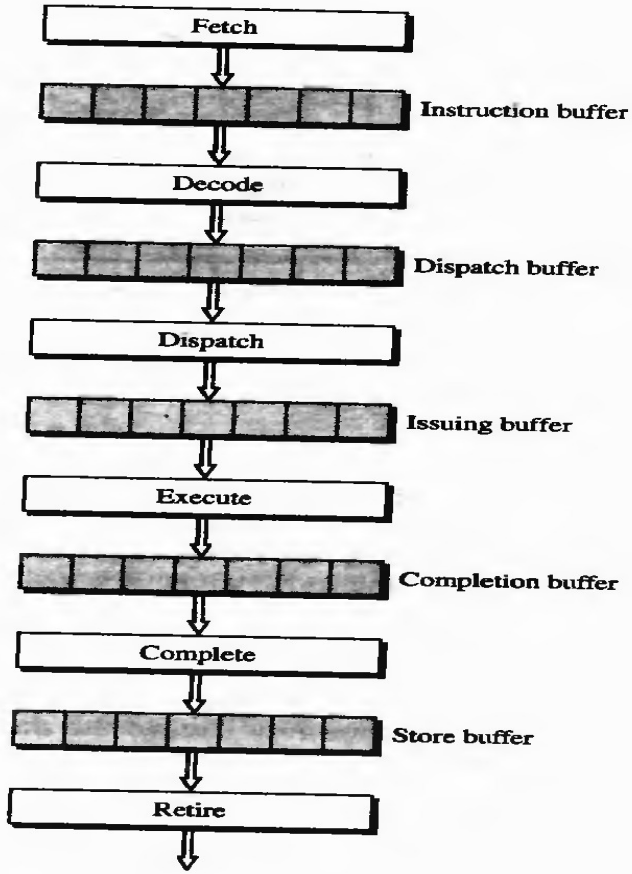


Figure 4.10
The Six-Stage Template (TEM) Superscalar Pipeline.

Przyjrzyjmy się teraz problemom związanym z budową superskalarnego potoku na poglądowym przykładzie składającym się z 5 faz i 6 potoków.

Fazy:

- Fetch
- Decode
- Dispatch
- Execute
- Complite
- Retitre

Fetching

Głównym problemem związanym z pierwszą fazą potoku jest zwiększone zapotrzebowanie na instrukcje.
Potrzebujemy do 6 instrukcji na cykl co znacznie zwiększa obciążenie magistrali. Pamięć z instrukcjami musi zostać zaimplementowana w taki sposób żeby spełniać te wymagania.
Co więcej, jeśli grupa instrukcji która powinna być wczytana naraz jest przełamana między dwa wiersze potrzebny będzie dodatkowy cykl.
A jeśli podczas w czytywania drugiego wiersza dostaniemy cache-miss? Bieda

Fetching – jak zaradzić kłopotom?

- Jednym z rozwiązań jest poinformowanie kompilatora o strukturze pamięci instrukcji żeby zapisywał instrukcje tak, żeby nie były przełamywanie pomiędzy wierszami.
- Dobrym pomysłem jest ustawianie instrukcji które są celem skoków jako pierwszych w rzędzie.

Dekodowanie Instrukcji

To jak skomplikowane zadanie będzie miała przed sobą jednostka dekodująca zależy w dużej mierze od zbioru instrukcji i ilości potoków.

Zadania:

- Rozłożenie instrukcji na części pierwsze.
- Wykrywanie zależności między instrukcjami

W przypadku złożonego ISA może być potrzebne kilka poziomów dekodowania. Wyżej wspomniany Intel Pentium z 1993 roku ma dwa poziomy dekodowania dla pewnych instrukcji (IA32). W ramach zwiększania się ilości faz zwiększają się oczywiście kary za pudła.

Dekodowanie Instrukcji cd.

- Jedną ze stosowanych technik jest tłumaczenie długich instrukcji na prostsze RISC'o podobne funkcje.
- Procesor może składać się z różnych jednostek dekodujących (jak w przypadku EXE'ków). Przykład – Pentium Pro jedna jednostka zdolna dekodować wszystkie rozkazy IA32, dwie pozostałe bardziej ograniczone.
- Kolejną ciekawą techniką jest predekodowanie. Po pudle pamięci instrukcji specjalna jednostka wstępnie dekoduje ładowany wiersz z instrukcjami. (Dodatkowe bity)