あまり詳しくない人のための
Meltdown/Spectre

許　先明

2018/07/15

Who am I

- 略歴
  - 1995年、株式会社インターリンク入社
  - 1997年、株式会社インターネット総合研究所入社
  - 2001年、株式会社インターネットシーアンドオー出向
  - 2005年、株式会社IRIコミュニケーションズ転籍（現ブロードバンドセキュリティ）
  - 2010年、株式会社LAC入社
  - 2014年、株式会社ブロードバンドタワー入社
  - 2017年、株式会社IoTスクエア転籍
- 活動
  - v6推進協議会 IPv4枯渇TF 教育・テストベッドWG メンバー（2009～現任）
  - v6推進協議会 セキュリティWG メンバー（2010～現任）

今年のお題

- 今年は本当にネタを考えるのに苦労しました
  - Security関連の話題は多すぎるくらいある
  - 内容が「いつもと同じ」か「難しすぎる」かのどちらか...
  - どうしよう？

- 結局、Meltdown and Spectreをお題に選びました
  - Spectreはスペルミスではありません(笑)
  - 1/3に脆弱性が公開されて、一部で大騒ぎになったアレです
  - 技術的な資料は作りました。プロ向け解説ばかりやっていました
  - 今回は、あまり詳しくない方に伝える資料にしてみようと...
    - うまく行っているかは正直わかりません

Meltdown and Spectre

- 2018年1月2日
- The Register(https://www.theregister.co.uk)紙がMeltdown/Spectreの脆弱性情報をすっぱ抜いて公開
  - Intel CPUに欠陥があるとしてその内容をリリース
- 日本時間1月3日、大騒ぎになりました

時系列

- 2017/06/01　GoogleがSpectreに関する情報を「関係者」に通知
- ～2017/07/28　GoogleがMeltdownに関する情報を「関係者」に通知
- 2017/12/06　AppleがMeltdownに対応したOSをRelease(macOS High Sierra 10.13.2)
- 2017/12/20　LWN.netでLinux カーネルのKPTI機能の記事が公開
- 2018/01/01　Linuxのパッチを元にCPUのバグの存在を指摘する記事が公開
- 2018/01/02　The RegisterがIntelのCPUに欠陥があるとして今回の静寂性を取り上げ
- 2018/01/03　Google Project Zeroが詳細を発表
  - Intelが正式に脆弱性の存在を認める
  - MSやAmazonが対応方針を発表
- 2018/01/04　グラーツ工科大学から脆弱性の詳細情報が公開される
  - MSが緊急のOS Updateを配布
- 以後、様々な対応や動きがあった...書ききれません

結局、これはなに？

- 正式な情報
  - Variant 1 Bounds check bypass (CVE-2017-5753)
  - Variant 1.1 Bounds check bypass on stores (CVE-2018-3693)
  - Variant 1.2 Read-only protection bypass (CVE unknown)
  - Variant 2 Branch target injection (CVE-2017-5715)
  - Variant 3 Rogue data cache load (CVE-2017-5754)
  - Variant 3a Rogue System Register Read (CVE-2018-3640)
  - Variant 4 Speculative Store Bypass (CVE-2018-3639)
- 共通のポイント
  - 現代のCPUにおける、投機実行およびOut of Order実行時に利用するCacheデータを、外部からの観測で推測することができることに起因する、データの不正な取り出し

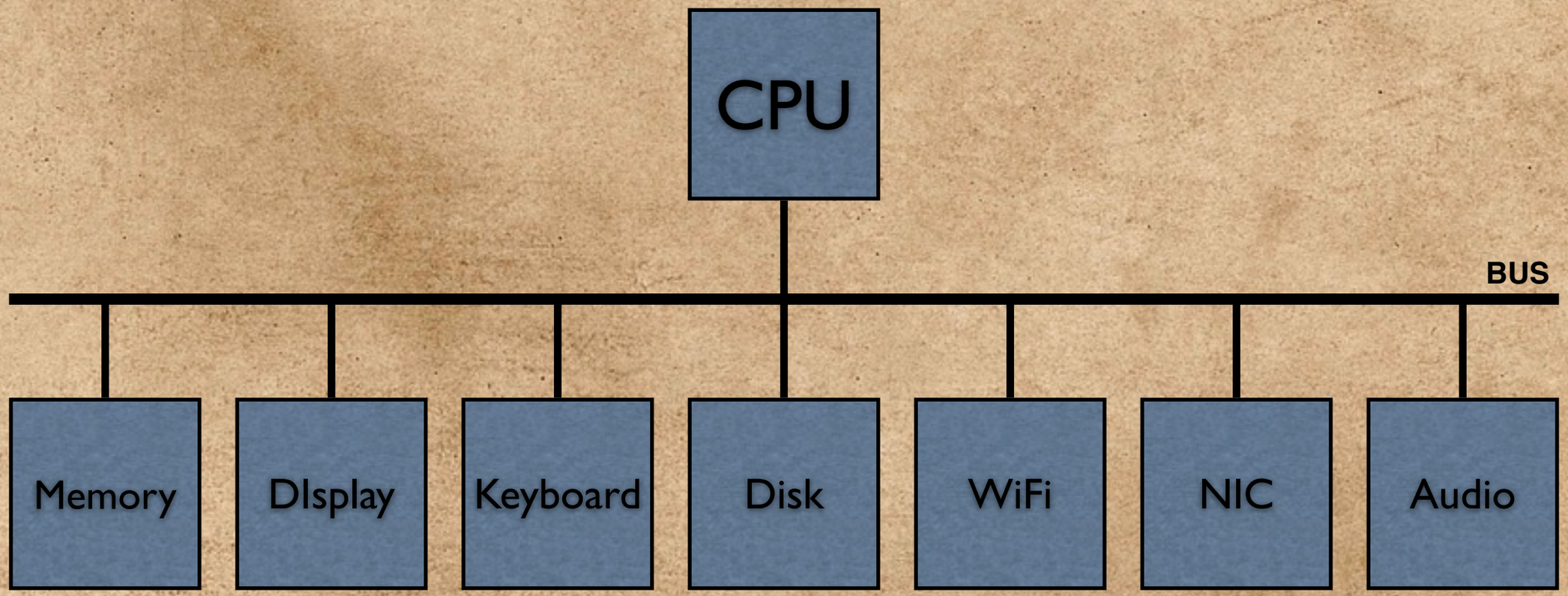
はい！
なんのことだかわかりません

これでわかる人は、
このプレゼンを聞く必要がありません(笑)

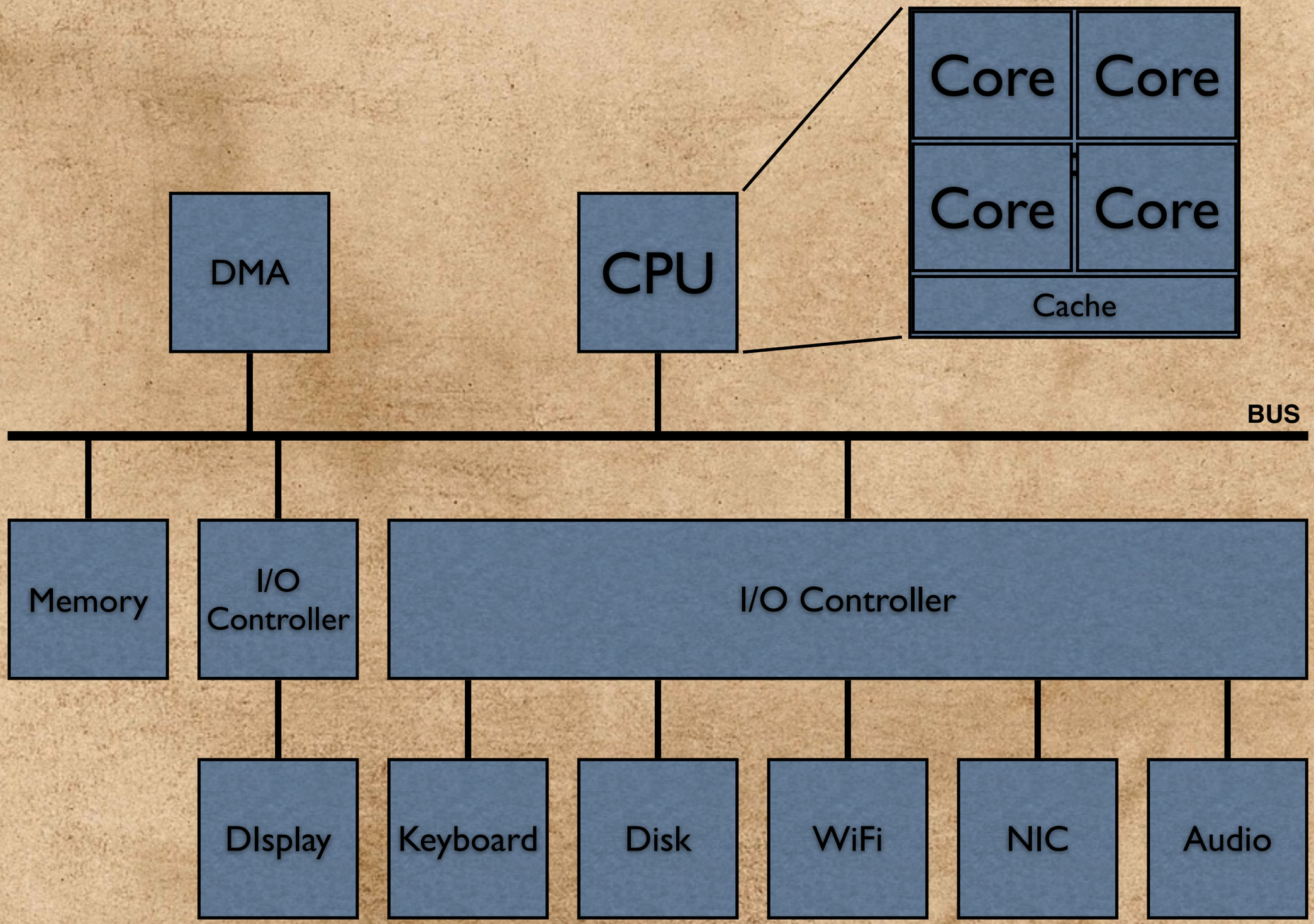
Computerとはなんでしょうか？

- 大元を辿れば「計算機」
  - そろばん、計算尺など、計算するための道具
  - 半導体の発展により、電子卓上計算機＝電卓が発明された
  - 今ではコンピュータに計算機のイメージはないが、日本語では電子計算機とも...
- 現代の（一般的な）コンピュータの構造
  - 計算・制御装置：CPU
  - 記憶装置：Memory・Disk
  - 入出力装置：キーボード・ディスプレイ・Disk・WiFi...

Computerの基本構造



現代のComputerの構造



現代のComputerの問題の一部

- CPUが激烈に高速なのに、BUSが遅い
  - CPU内部処理<CPU内Cacheアクセス<<Memoryアクセス<<<<I/Oアクセス
- CPU内部で行う処理が複雑化
  - CPU内にCoreと言う名のCPUが複数あるような構造
  - Coreの内部で余っている回路を使うことで処理できる計算量を増やすような機構
- CPUへの命令の複雑化
  - １命令を処理するためにかかる時間が一定しない
  - １命令のために必要な演算器（加減乗除、整数・浮動小数処理等）の多様化
    - Multimedia命令と言う名の行列演算器などもある...

現代のCPUの命令処理(単純な加算処理)

- 例えば、1+1（Add 1,1）をCPUは以下のように命令を処理する

1.実行する命令（加算処理命令）を読み込む

2.左項の値をメモリーから読み出す

3.記憶領域(Register)に左項の値を投入

4.右項の値をメモリーから読み出す

5.記憶領域(Register)に右項の値を投入

6.命令を判断し、Dispatcherが加算器にRegisterを引き渡し実行させる

7.加算器からデータを取得し、記憶領域(Register)に結果を記録する

- これらのそれぞれの処理ごとに、それなりの時間がかかる

現代のCPUの祖先 Pentium©

- 現在皆さんが使っているCPUはいわゆるCISC(Complex Instruction Set Computer)と呼ばれる命令セット設計のもの
- RISC(Reduced Instruction Set Computer)と呼ばれる命令セットアーキテクチャもある
- CISCは、比較的抽象化された命令セットの体系である
- RISCは、細かく分解された単純な命令セットの体系である
  - 1命令処理にかかる時間が同じなので、命令のスケジューリングがしやすい
  - 必要となるトランジスタ数が少ない
- 現代のCPUは例えて言えばRISC CPU上で動作するCISC CPU Emulatorのようなものである
- 現代のCPUの基本設計は「Pentium©」によって確立された
  - 究極を言えば、CPU内のCoreはPentiumのようなものと言える

Pentiumの何が凄かったか？

- Pentiumは
  - CISCの皮を被ったRISC系の実装を行った商用で最も成功したCPU
  - 処理の高速化のために、Pipelineと呼ばれるアーキテクチャ（後述）を採用した
    - 正確には Instruction Pipeline(IP)
  - このPipelineこそが、今回の脆弱性の遠因

- Pipelineとは？
  - 命令処理能力を向上させる設計技法のひとつであり、現代のCPUの高速化の基本
  - 1命令を、細分化された「独立して実行できる工程に分割」する
  - 分割された各工程を並列化（正確には時分割）して、全体の処理時間を大幅に減少させる

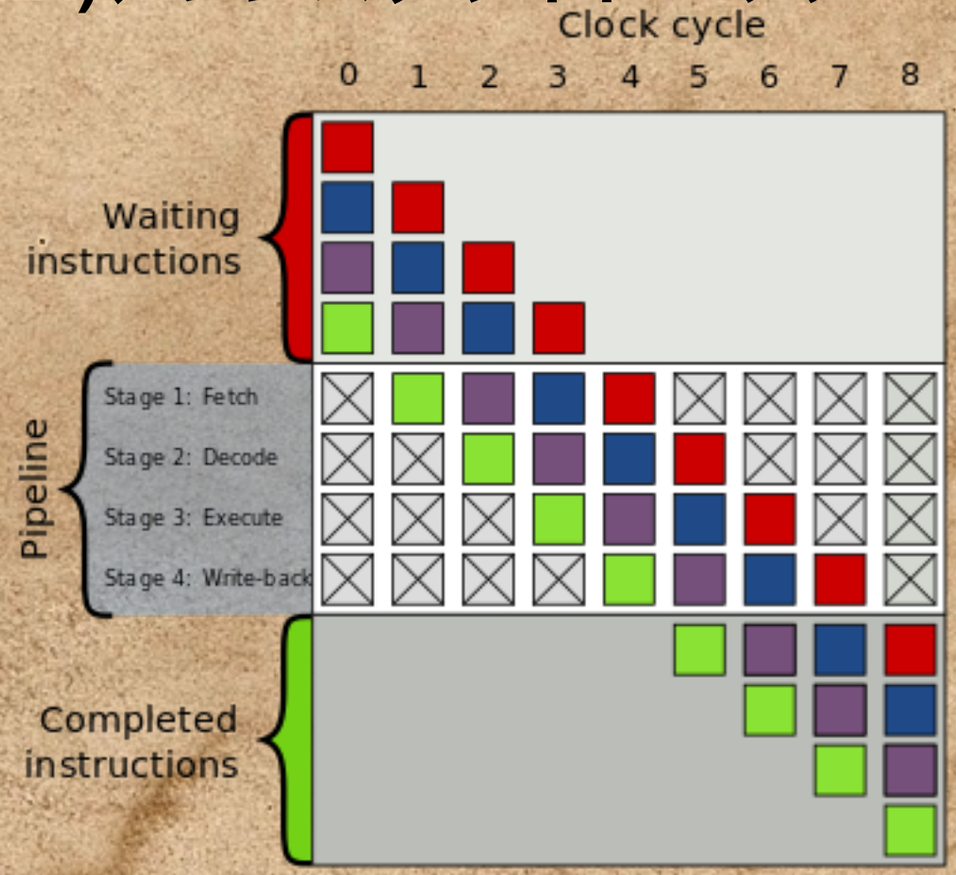
Instruction Pipeline

- Instruction Pipeline(以下IP)は、命令スループットを向上させる設計技法のひとつ
  - IPを持つプロセッサは、命令の処理にあたって、細分化された「独立して実行できる工程」に分割する。
  - 分割された各工程を並列化して全体の処理時間を大幅に軽減する
  - 命令fetch(IF)、命令Decode(ID)、Execute(EX)、レジスタライトバック(WB)などのように内部処理を細分化する

右図が４段パイプラインの例。

各命令が細分化されて流れていくのがわかる。

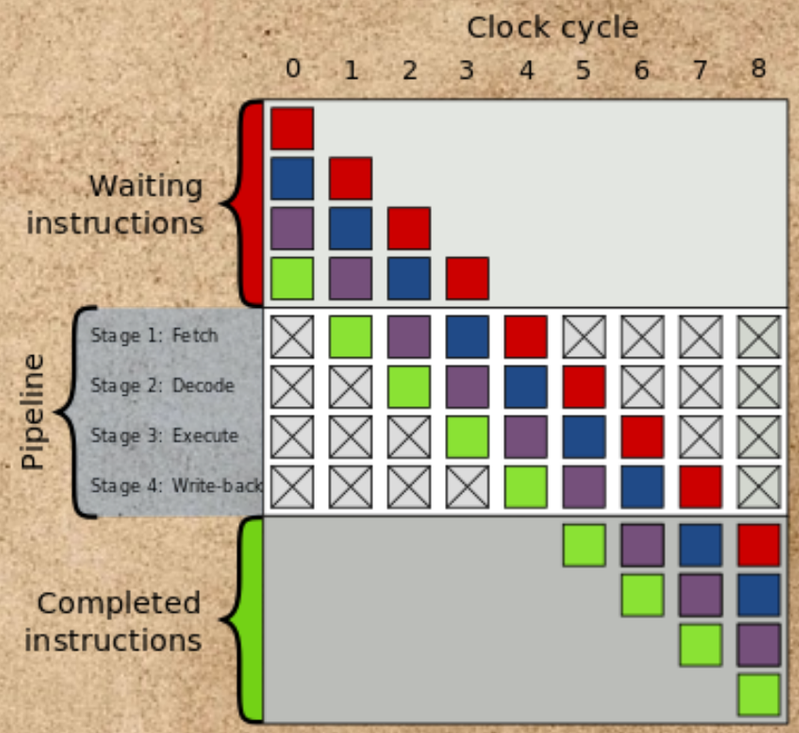
なお、Pentiumの場合、Pipelineは9段

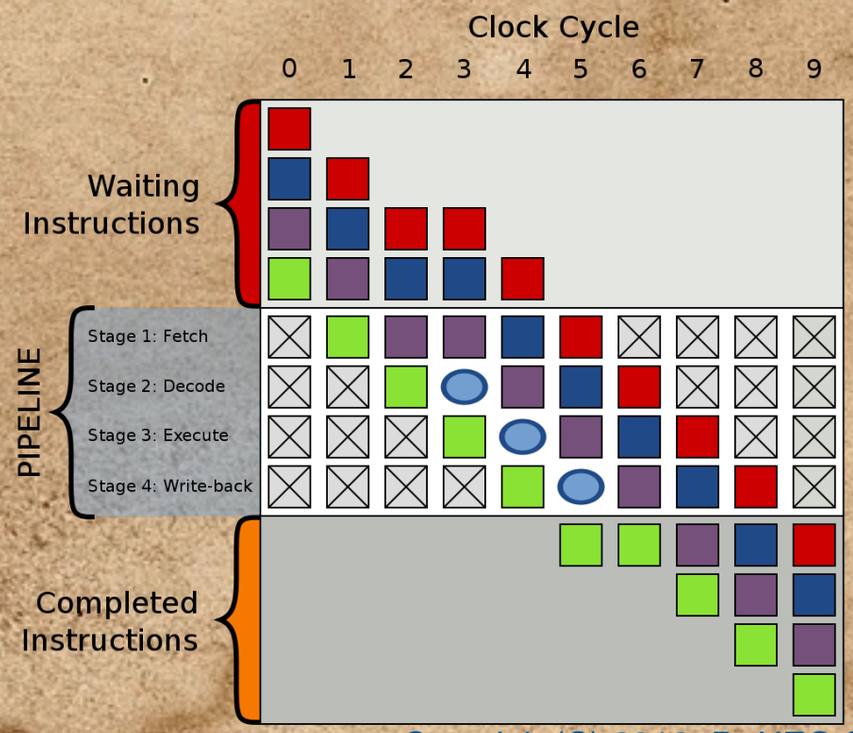


WikiPediaより

Instruction Pipelineにおける問題

- Pipelineによって、CPUの内部処理能力は大幅に向上した
  - メモリーや周辺装置からデータを読み出す速度は向上しなかった
  - その結果、CPUが「メモリーやI/Oからデータを取得する」タイミングで待ち時間が発生する
    - このPipelineに処理待ちで穴が空くことをBubbleが発生すると呼ぶ
  - Bubbleが発生すると、当然、その先の処理ができない
  - 結果、効率が上がらないことになる→これが性能向上に致命的だった





Pipeline Bubbleへの対処

- CPU高速化のためには、Pipeline-Bubbleを発生させないことが重要
- なぜPipeline Bubbleが発生するのか？
  - データにアクセスする際に、そのデータ取得にかかる時間がBubbleの根本原因
  - データがCPUのCache(L1 Cache)にあれば高速に処理できるが、L2/L3 cacheやメモリー、バスの向こうのI/O側にあるデータは非常にペナルティが大きい
- これ以上の高速化の方法はあるのか？
  - ある

In Order実行とOut of Order実行

In Order

1. 命令フェッチ
2. オペランド（データ）準備
3. 実行ユニットへの割当
4. 実行
5. 結果をレジスタに格納

Out of Order

1. 命令フェッチ
2. 命令にバッファエントリーを割当
3. 命令を待ち行列から実行キューに送る
4. 待ち行列内命令はオペランドを待つ
5. オペランドを受け取って実行キューへ
6. キュー内の命令に実行ユニットを割当
7. 実行
8. 結果をバッファに格納
9. バッファ内の最も古い命令の実行終了を待つ
10. 古い命令から実行結果をレジスタに格納

- 現代のCPUでは、
  CPU内キャッシュ<<メモリー内データ<<I/Oの向こうにあるデータ(DiskやNetworkなど)
  の順にデータ受け取り時間が激烈に増加する
- パイプラインが多段になればなるほど、無駄なパイプラインを作らないことが必要になる。
- CPU内のパイプラインを遊ばせないために、処理できるものから処理するというのがOut Of Order実行の肝

OoOで解決できない問題

- 近年のCPUは、性能向上のために、多段パイプライン構造を採用
- Out of Order(OoO)によって、時間のかかる処理を後回しにできるようになった
- しかし、プログラムは「条件分岐が必須」
- 条件分岐は、条件判断の結果が出るまで行き先を決定できない
  - すなわち、OoO実行することができず、待ちが増え、結果性能が落ちる

投機的実行(Speculative Execution)

- 投機的実行とは
  - CPUに、必要としないかもしれない仕事をさせること
- CPUに用いられている投機的実行の実装
  - メモリーやキャッシュへのデータのPreFetch
    - 使う可能性がある情報を先行取得する
  - 分岐予測
    - 分岐先がどうなるかを過去の実行履歴に基づいて先行予測する(Branch Prediction)
    - 確率的に実行される可能性が高い方の処理を先に実行しておく
    - 結果、予測が間違っていた場合、そこまでの結果を捨ててやり直し。
  - 積極的実行
    - 投機的実行の一種
    - 分岐予測において、「両方の分岐先のプログラムを実行」
    - 条件分岐判断の結果をみて採用する結果を決める

MeltdownとSpecreとは？

- 共通のポイント
  - 現代のCPUにおける
  - 投機実行およびOut of Order実行時に利用するCacheデータを
  - 外部からの観測で推測することができることに起因する
  - データの不正な取り出し

- つまりどういうこと？

Side Channel Attack

- **Side Channel Attack**
  - 暗号化されたデータ（等）を（直接データにアクセスせずに、他のプロセスなどが）当該データを処理している際に発生する物理的な現象を観測することによって推定する攻撃手法
    - 処理時間の違い、消費電力の違い、電磁波の違いなどが有名

- **Flush+Reload**
  - キャッシュメモリーを利用したSide Channel Attackの一種
    - 初期化、抜き取り、復元を利用する
      - Cache Region (領域) をPurge（初期化）
      - 必要な情報を読み込み、Cacheに載せる
      - On cacheとNot on cacheの際のCacheのLookup速度差を利用して復元

Meltdown and Spectre

- MeltdownとSpectreは、
  - CPU処理における投機的実行結果として
  - 破棄されるはずの「処理してしまった結果として保持」されているデータを
  - 外部からの観測を用いて推測する
  - と云う手法で攻撃する攻撃方法

- つまり
  - CPU内で処理されている以上、Cacheには捨てられるはずのデータが存在する
  - Cacheされている以上、Cacheされていない情報よりも高速にアクセスできる
  - アクセスした時の返答にかかる速度差を観測(Side Channel Attack)すれば...

ご静聴

ありがとうございました