

# コンピュータ アーキテクチャ I 第8回

---

塩谷 亮太 (shioya@ci.i.u-tokyo.ac.jp)

東京大学大学院情報理工学系研究科 創造情報学専攻

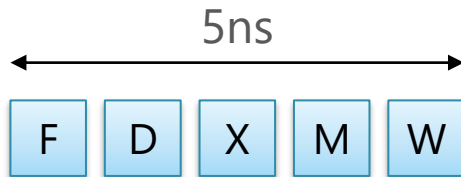
# 課題の解説

---

# 課題 7

## ■ 前提：

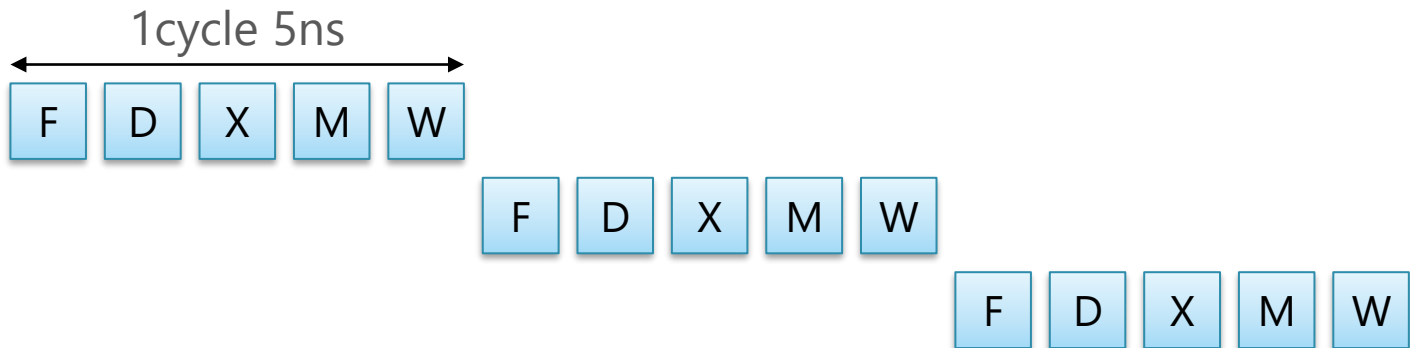
- 各命令は以下の5つの処理を経て実行されるものとする
  - F：フェッチ, D：デコード, X：演算,  
M：メモリアクセス, W：書き戻し
- これらの各処理には 1 nano second (ns) がかかるものとする
- 演算子しか行わずメモリアクセスを伴わない命令でも, 必ず M を経るものとする
- たとえば以下の命令を実行するために必要な時間は 5ns
  - 「必要な時間」とは, フェッチから書き戻しが終わるまでの時間とする



# 課題 7

- (1) その場合の最大動作周波数はいくつになるか述べよ
  - $1 \text{ 秒} / 5\text{ns} = 200\text{GHz}$
- (2) 以下の命令列を実行するのに必要な時間を計算せよ
  - $5 \text{ ns} \times 3 \text{ 命令} = 15\text{ns}$

add x1←x2+x3  
sub x2←x3+x4  
add x5←x6+x7



# 課題 7

- 前記の前提の下で, F,D,X,M,W の5ステージからなるパイプライン・プロセッサを構成することを考える  
ただし, in-order スカラ・プロセッサであるものとする

- (3) その場合の最大動作周波数はいくつになるか述べよ

- 1 秒 / 1 ns = 1GHz

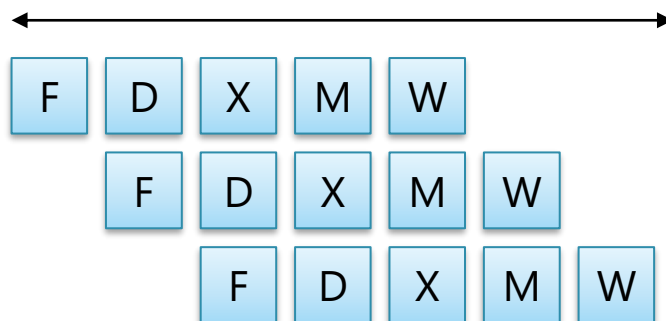
- (4) 以下の命令列を実行するのに必要な時間を計算せよ

add x1←x2+x3

sub x2←x3+x4

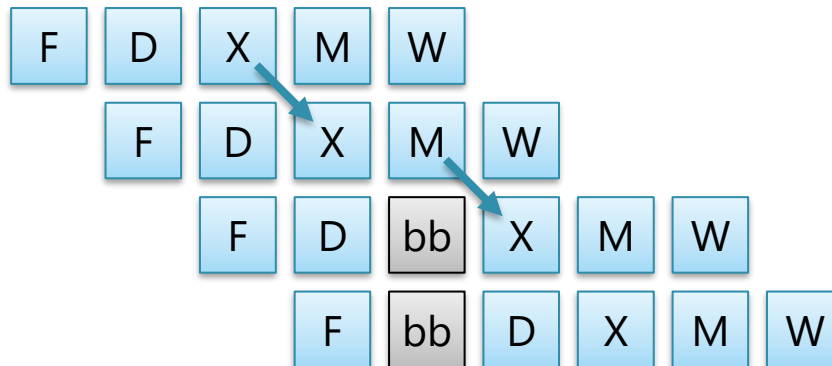
add x5←x6+x7

- 7ns



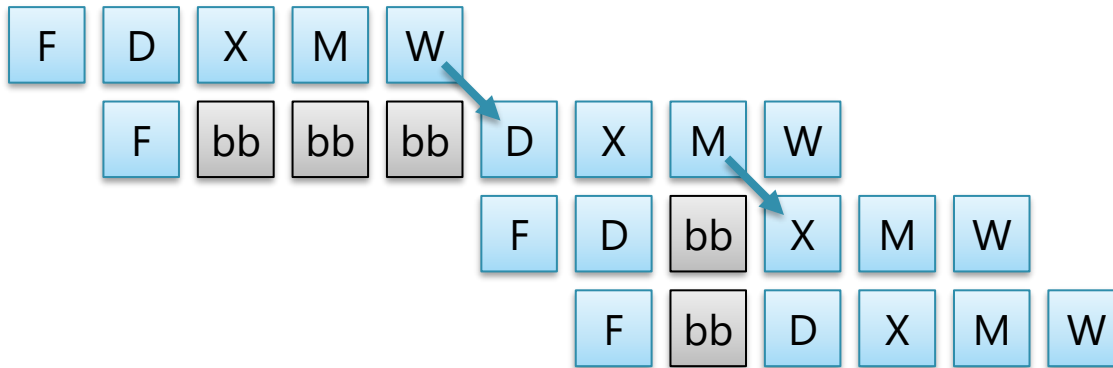
# 課題 7

- (5) 以下の命令列を実行するのに必要な時間を計算せよ  
依存関係のために必要な場合のみパイプラインを適宜ストールして実行するものとせよ  
add x1 ← x2 + x3  
ld x2 ← (x1)  
add x5 ← x2 + x7  
ld x2 ← (x3)
- 9ns (フォワーディングありの場合、すいませんこれを指示に入れるのを忘れてました)



## 課題 7

- (5) 以下の命令列を実行するのに必要な時間を計算せよ  
依存関係のために必要な場合のみパイプラインを適宜ストールして実行するものとせよ  
add x1 ← x2 + x3  
ld x2 ← (x1)  
add x5 ← x2 + x7  
ld x2 ← (x3)
- 12ns (フォワーディングなしの場合の別解)



## 課題 7

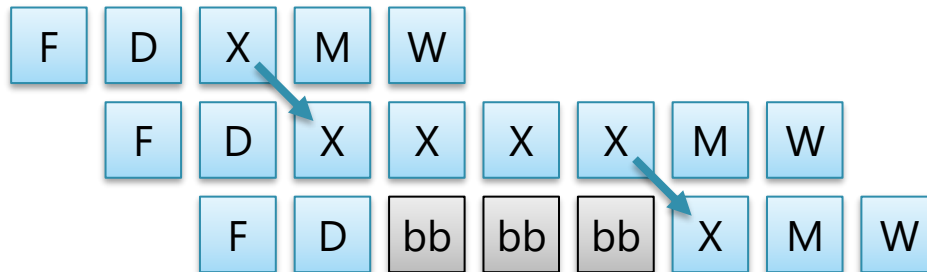
- (6) 以下の命令列を実行するのに必要な時間を計算せよ  
ここで mul は乗算命令であり X に 4 サイクルが必要である。  
依存関係のために必要な場合のみパイプラインを適宜ストールして実行するものとせよ

add x1 ← x2 + x3

mul x2 ← x1 + x4

add x5 ← x2 + x7

- 10ns (フォワーディングありの場合)





## 課題 7

- (7) 以下の命令列を実行するのに必要な時間を計算せよ  
ここで beq はオペランドが等しい時に分岐する分岐命令である  
プロセッサは分岐予測を行うものとし, beq が分岐予測ミスを起こして LABEL に  
飛んだあとにフラッシュされてやり直した場合を想定せよ

```
li x1 ← 1
```

```
li x2 ← 1
```

```
beq x1==x2, LABEL
```

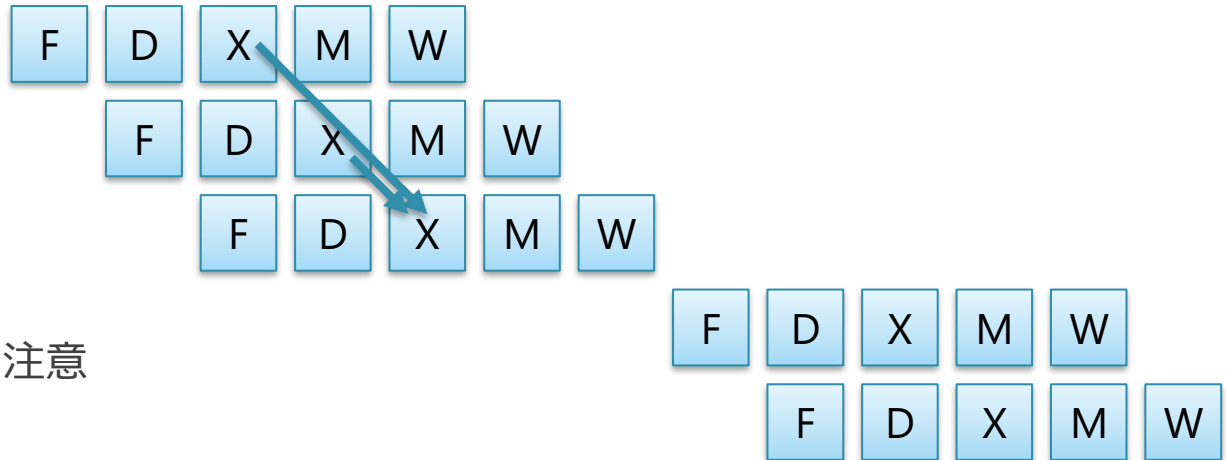
add  $x_1 \leftarrow x_2 + x_3$

**LABEL:**

```
add x2 ← x3 + x4
```

- 13ns  
フォワーディング  
ありの場合

やり直した後に  
結局 LABEL の下の  
命令も実行することに注意



# 性能のモデル

---

- CPU の性能（処理速度） = 単位時間あたりに処理できる命令数
- 性能は以下の 2 要素のかけ算で決まる：
  1. クロック周波数 =  
1 秒あたり何サイクル分の処理ができるか
  2. Instructions per cycle (IPC) =  
1 サイクルあたり何命令処理できるか

# 性能のモデル

- 以下について検討していく
  - 理想的な場合の性能のモデル
  - ハザードを考慮した性能のモデル

# 理想的な場合の性能モデル

---

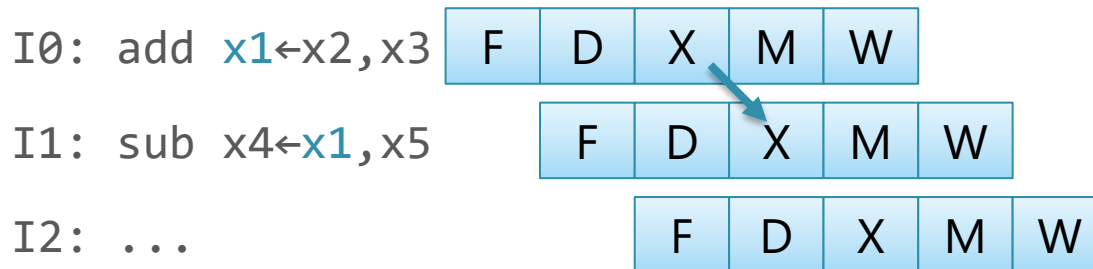
# 理想的な各モデルの性能

- 下記のそれぞれについて秒間何命令処理できるかを考える
  1. シングル・サイクル・プロセッサ
  2. スカラ・パイプライン・プロセッサ
  3. スカラ・パイプライン・プロセッサ（段数2倍）
  4. 2-way スーパースカラ・プロセッサ

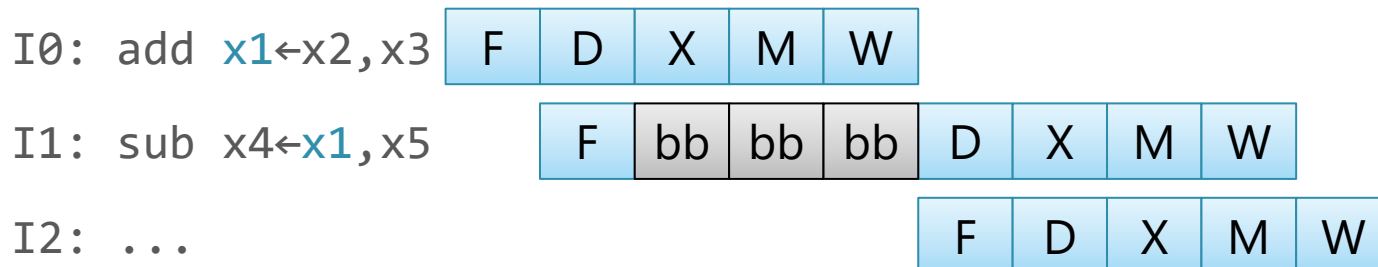
# ここからの前提： パイプラインではフォワーディングの実装を仮定

- 結果が RF に書かれる前に，後続の命令は最速のタイミングでそれを使えると想定
- つまり，依存元命令の実行ステージ（以下では X）が終わり次第，直ちに依存先の命令は実行ステージを開始できる

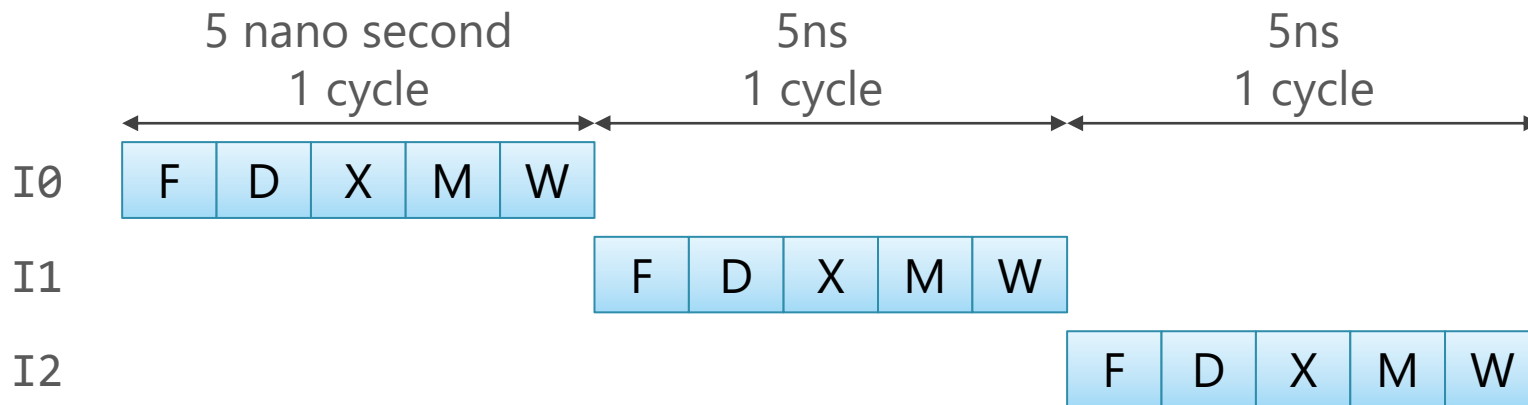
I0 の結果を I1 にフォワーディング



フォワーディングを実装しない場合，バブルが生じる



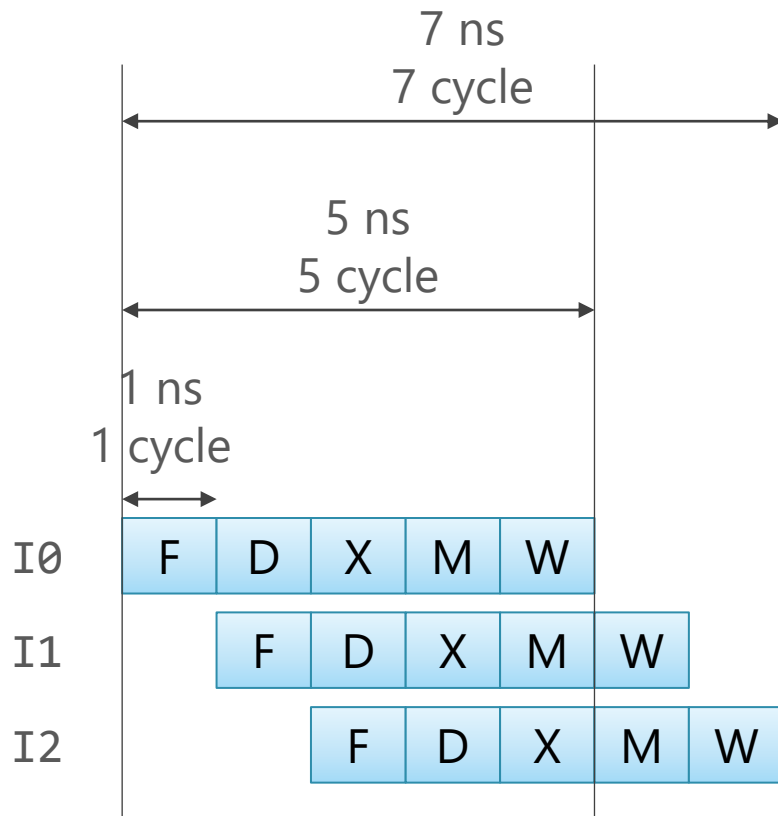
# シングル・サイクル・プロセッサ の性能



- $IPC = 1 \text{ 命令} / 1 \text{ サイクル} = 1$
- 周波数 =  $1 \text{ 秒} / 5 \text{ ns} = 200 \text{ MHz}$  (M = メガ 100万)
- 性能 (1秒あたりの命令数) =  $IPC * \text{周波数} = \text{秒間 } 200 \text{ M 命令}$

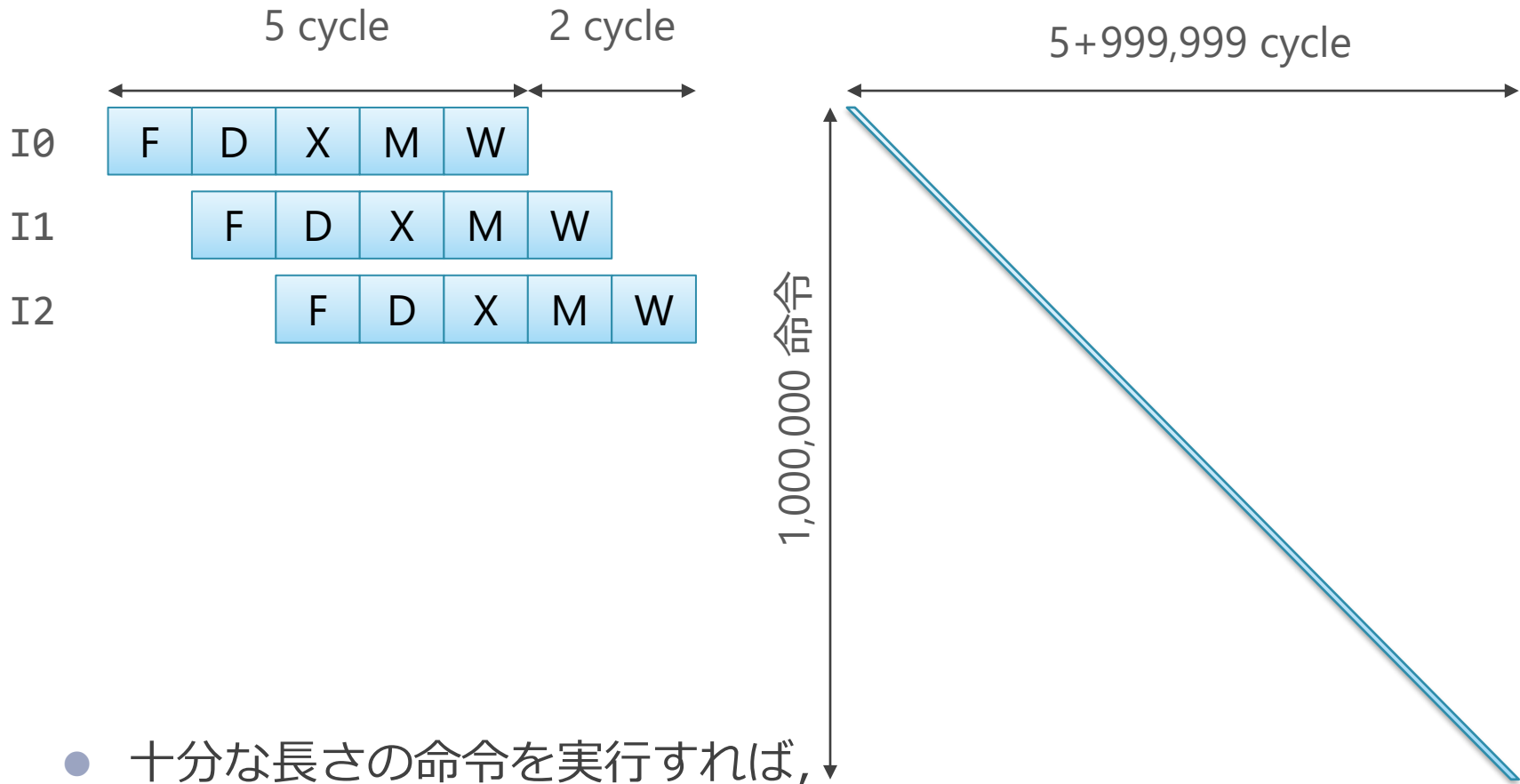


# パイプライン化されたプロセッサ の性能



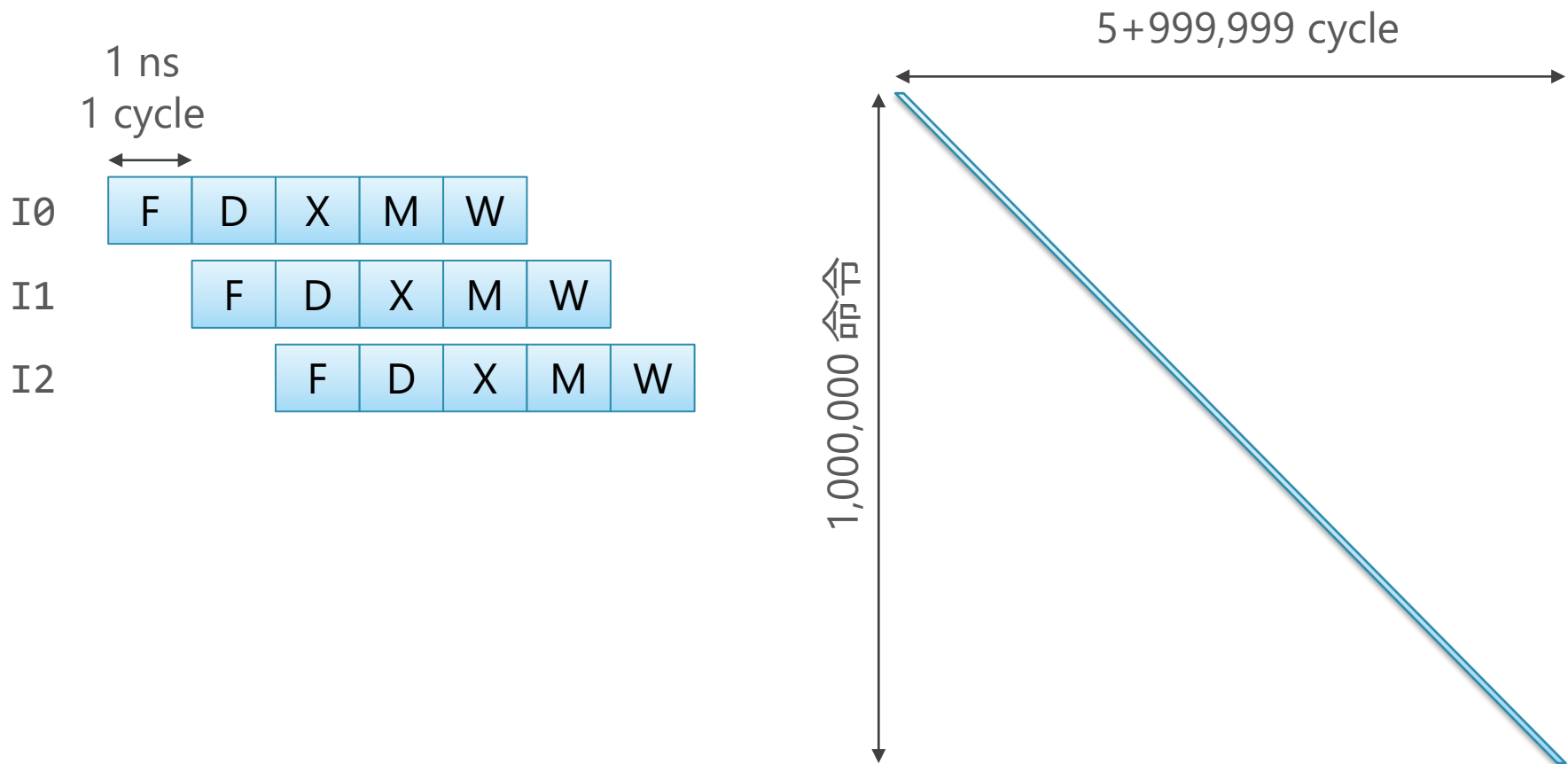
- シングル・サイクル・プロセッサの時とはサイクルの長さが変わっている事に注意
- 周波数 =  $1 \text{ 秒} / 1 \text{ ns} = 1000 \text{ MHz} = 1 \text{ GHz}$
- IPC はどう考える？ IPC = 3 命令 / 7 サイクル？

# 「十分に多く」の命令を実行した場合



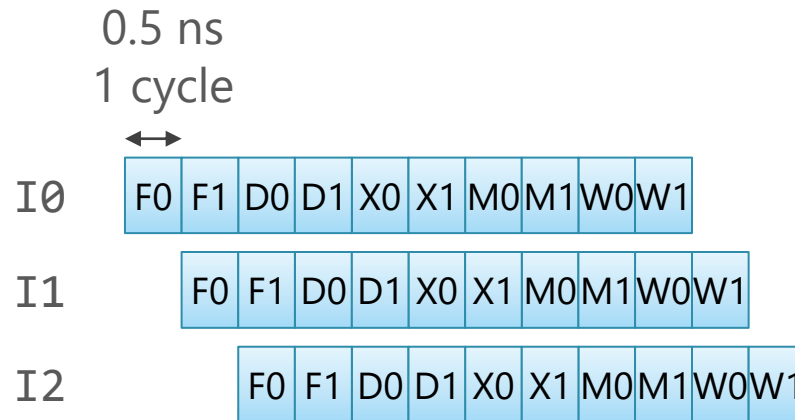
- 十分な長さの命令を実行すれば、パイプラインの長さ分は無視して近似できる
  - 右上の場合、 $\text{IPC} \approx 1,000,000 / (5 + 999,999)$  は、 $\text{IPC} \approx 1$
- 左端の F ないしは右端の W ステージの傾きのみを考えればよい

# 「十分に多く」の命令を実行した場合の性能



- $IPC \approx 1 \text{ 命令} / 1 \text{ サイクル} = 1$
- 周波数 =  $1 \text{ 秒} / 1 \text{ ns} = 1000 \text{ MHz} = 1 \text{ GHz}$
- 性能（1秒あたりの命令数） =  $IPC * \text{周波数} = \text{秒間 } 1 \text{ G 命令}$

# パイプライン段数を倍にした場合の性能

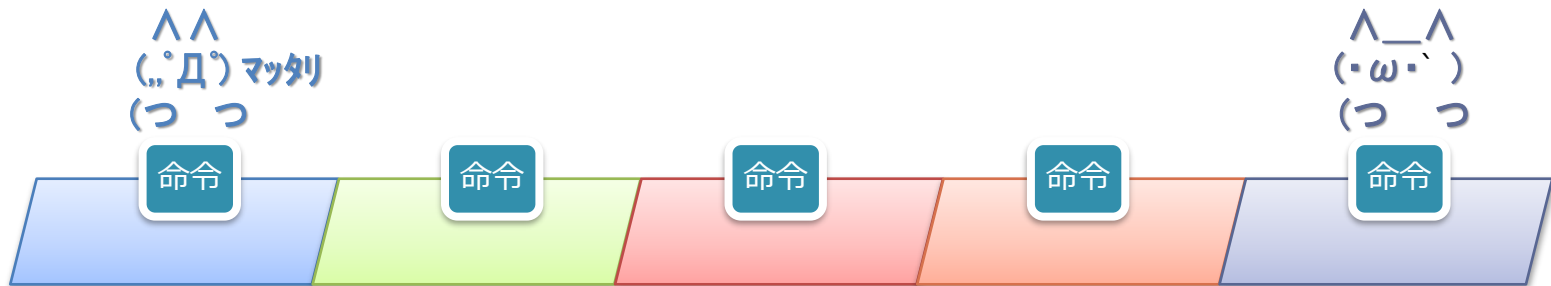


- 1 ページ前までのパイプラインのステージを半分に分割して (= 各ステージで実施する仕事を半分にして) 倍速で動かしている
- $IPC \approx 1 \text{ 命令} / 1 \text{ サイクル} = 1$
- 周波数 =  $1 \text{ 秒} / 0.5 \text{ ns} = 2000 \text{ MHz} = 2 \text{ GHz}$
- 性能 (1秒あたりの命令数) =  $IPC * \text{周波数} = \text{秒間 } 2 \text{ G 命令}$

# 各ステージで実施する仕事を半分にして、 かわりに倍速で動かしている

$IPC \approx 1 \text{ 命令} / 1 \text{ サイクル} = 1$

周波数 =  $1 \text{ 秒} / 1 \text{ ns} = 1000 \text{ MHz} = 1 \text{ GHz}$



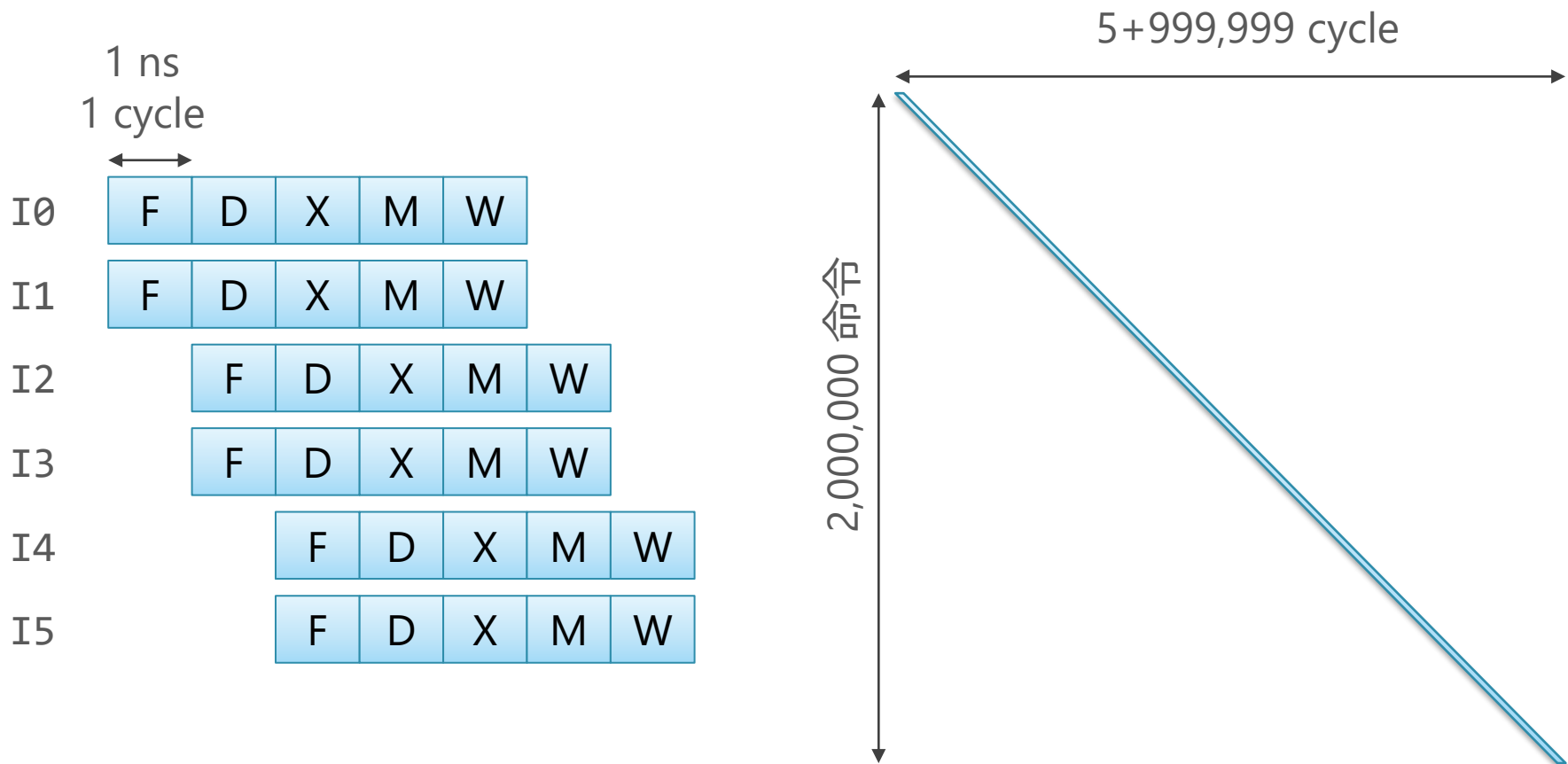
$IPC \approx 1 \text{ 命令} / 1 \text{ サイクル} = 1$

周波数 =  $1 \text{ 秒} / 0.5 \text{ ns} = 2000 \text{ MHz} = 2 \text{ GHz}$

1 人が 1 サイクルに行う仕事を半分にして、かわりに倍速で次に送る



# 2-way スーパースカラ・プロセッサの性能



- $IPC \approx 2 \text{ 命令} / 1 \text{ サイクル} = 2$
- 周波数 =  $1 \text{ 秒} / 1 \text{ ns} = 1000 \text{ MHz} = 1 \text{ GHz}$
- 性能（1秒あたりの命令数） =  $IPC * \text{周波数} = \text{秒間 } 2 \text{ G 命令}$

# ここまでの性能は理想的な性能

## ■ 理想的な各モデルの性能

- シングル・サイクル・プロセッサ : 秒間 200M 命令
- スカラ・パイプライン・プロセッサ : 秒間 1G 命令
- スカラ・パイプライン・プロセッサ (段数 2 倍) : 秒間 2 G 命令
- 2-way スーパスカラ・プロセッサ : 秒間 2 G 命令

## ■ 現実には・・・

- 回路的な要因により (第 5 回講義) ,
  - スーパスカラにするとステージごとの回路が増えて遅延が増加 → 周波数は落ちる
  - パイプライン段数を 2 倍にしても周波数は倍にならない
- 各種のハザードにより, ストールが起きて IPC が下がる (第 6 回講義)

# ハザードを考慮した性能のモデル

---



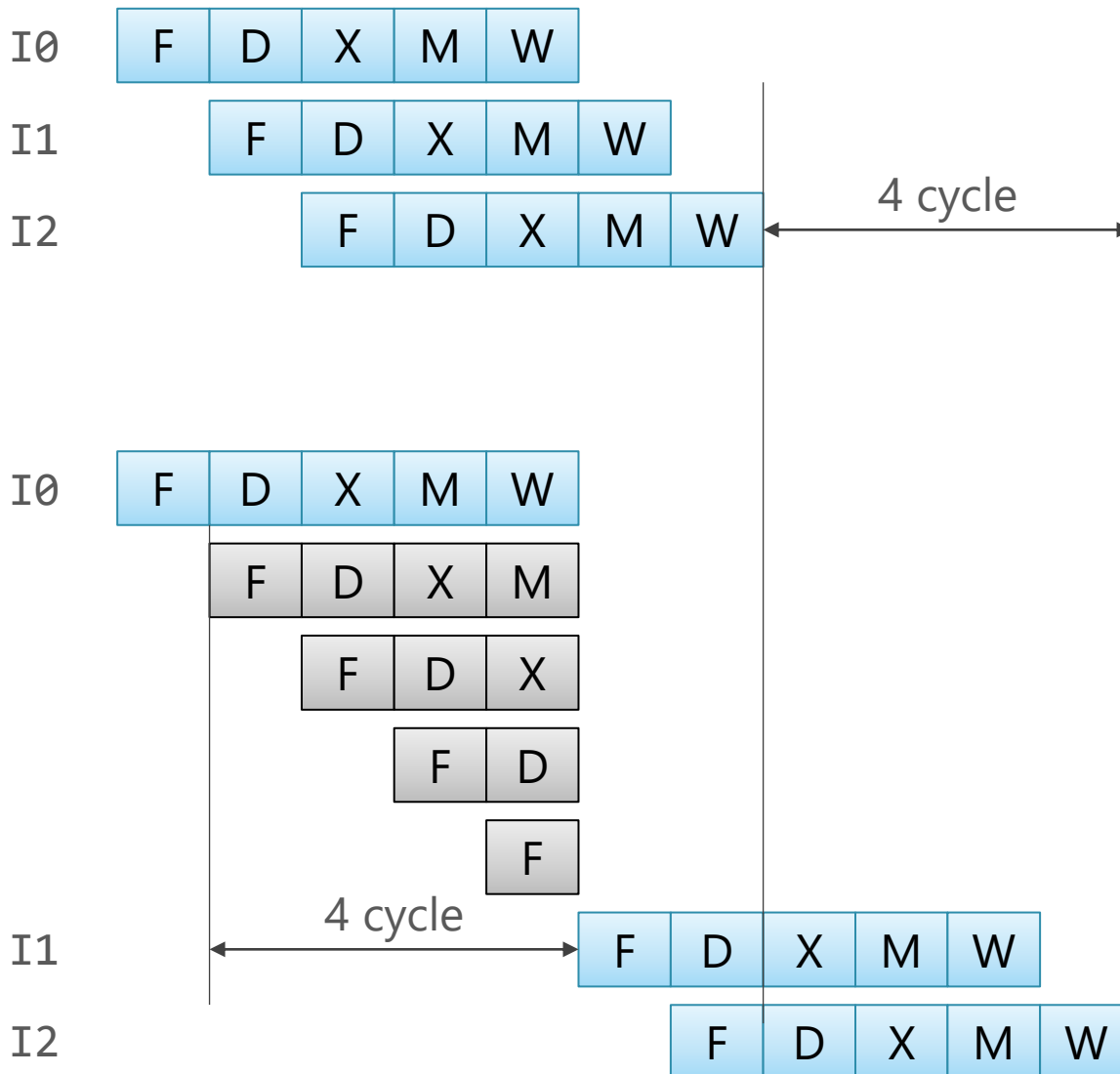
# 性能のモデル

- 以下について検討
  - 理想的な性能のモデル
  - ハザードを考慮した性能のモデル

# ハザードによる IPC の低下

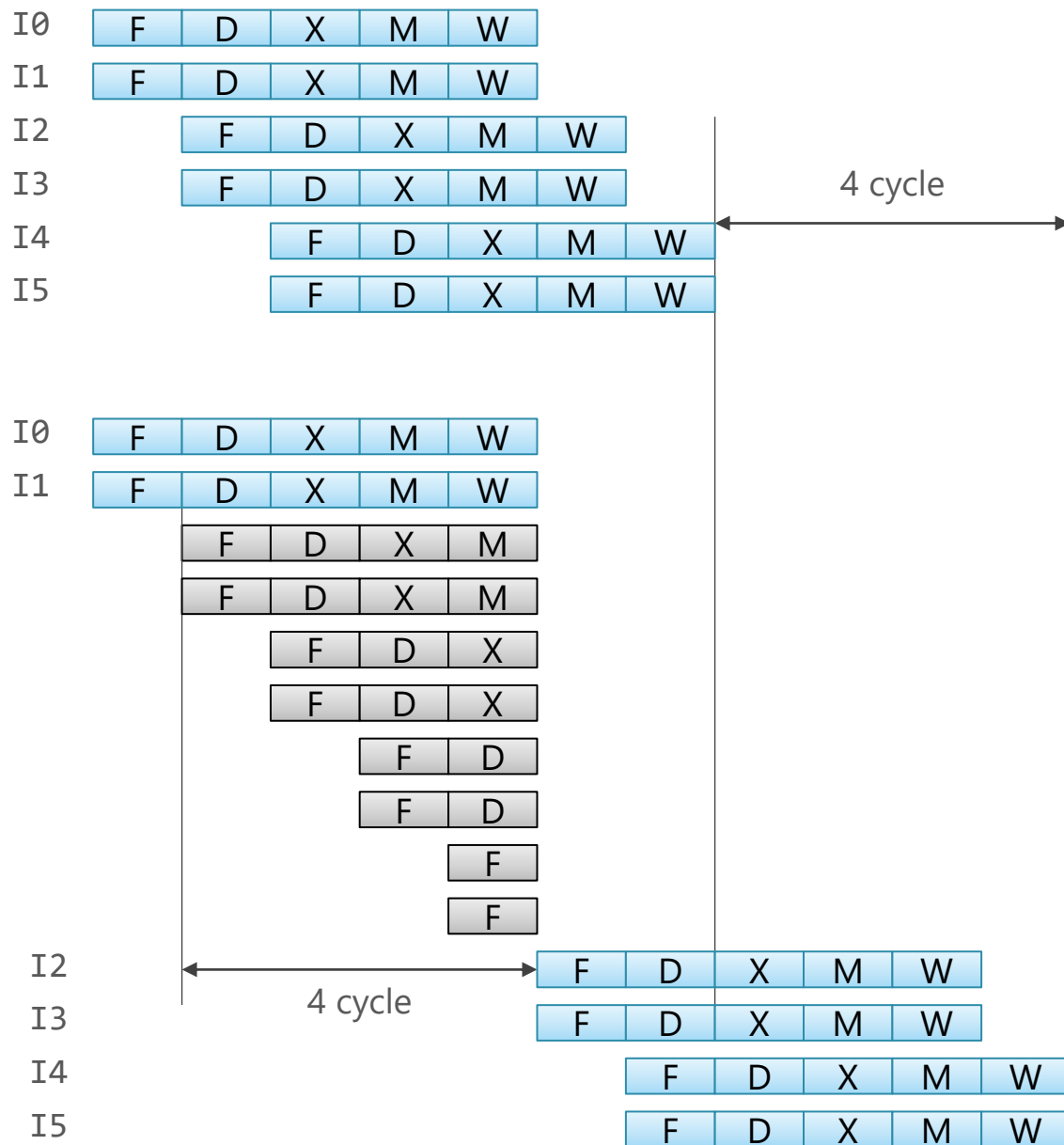
- 以下を例として、具体的な IPC の数字の変化を説明
  - 分岐予測ミス
  - ロードに依存した命令によるデータハザード
- キャッシュによる影響も大きい
  - （この講義ではまだ説明していないが、後日説明

# 分岐予測ミスによる実行サイクルの増加



- I2 の実行が 4 サイクル遅れている
- 4サイクル分の処理を取り消してやり直しているため

# スーパスカラの場合



- I5 の実行が 4 サイクル遅れている
- 4サイクル分の処理を取り消してやり直しているため

# 分岐予測ミスによる実行サイクルの増加のモデル

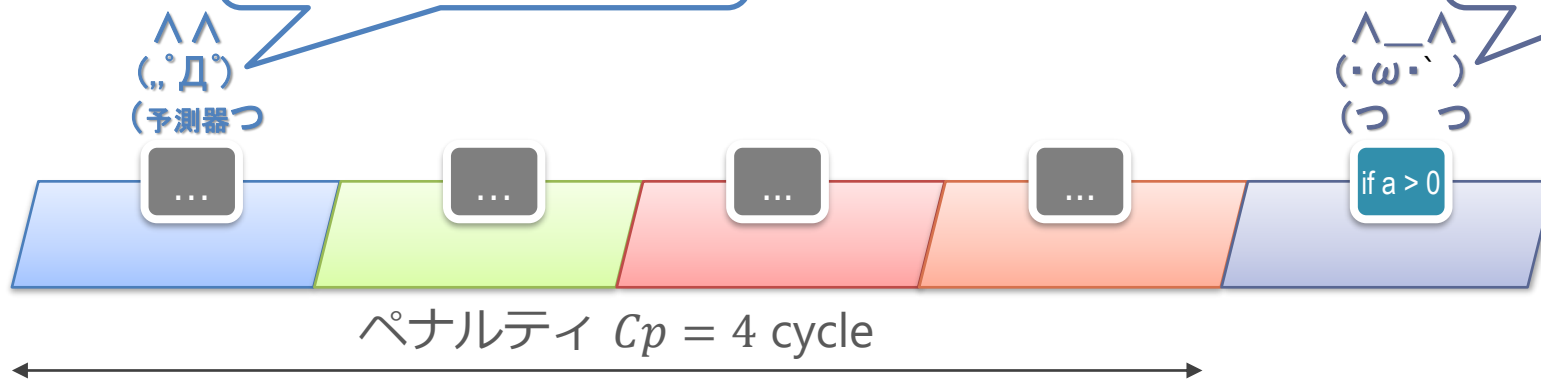
## ■ まとめると…

- 予測ミス毎に、実行サイクル数がペナルティの分だけ伸びる
  - ペナルティは（パイプライン段数 - 1）サイクル
  - スカラでもスーパスカラでも同じ

# ペナルティは（パイプライン段数 - 1）サイクル 取り消される命令数は増えるが，時間は同じ

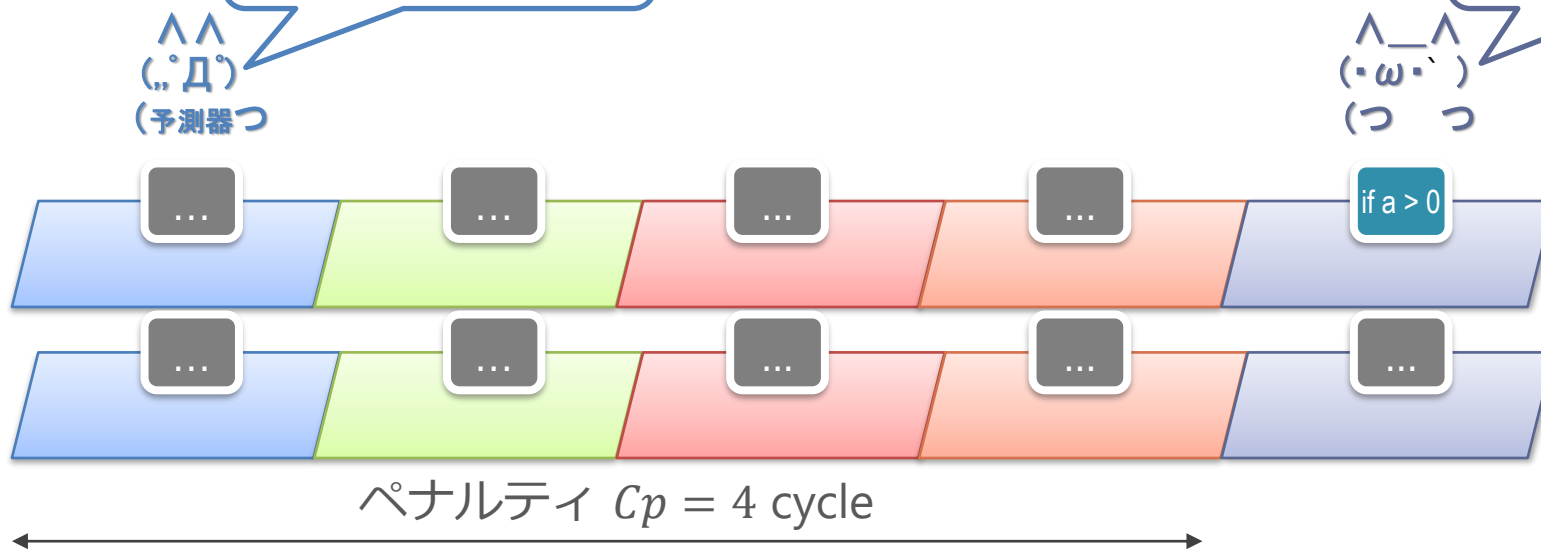
else じゃなかったかー

間違っとするがな



大損害まったなし

ライン2本分  
廃棄やん...



# 分岐予測ミスによる実行サイクルの増加のモデル

## ■ 以下のようにおいた場合,

- ミスがない理想的な実行の際のサイクル数 :  $C_t$
- 分岐予測ミスの発生回数 :  $N_m = N_i \times P_b \times P_m$ 
  - 実行命令数 :  $N_i$
  - プログラム中の分岐命令の出現率 :  $P_b$
  - 分岐命令毎の予測ミス発生率 :  $P_m$
- 分岐予測ミス・ペナルティ :  $C_p$

## ■ 実行サイクル数 $C_r$ は, 理想サイクル数 $C_t$ に対して,

- $C_r = C_t + N_m \times C_p$
- 意味 : 予測ミスの発生回数×ペナルティ だけ実行時間が伸びる

# 具体的な値を入れてみる

## ■ 以下のように置いた場合,

- 理想的な実行の際のサイクル数 :  $Ct = 1M$  (メガ)
- 分岐予測ミスの発生回数 :  $Nm = Ni \times Pb \times Pm = 0.025M$ 
  - 実行命令数 :  $Ni = 1M$
  - 分岐命令の出現率 :  $Pb = 0.25$   
(プログラムは大体 4 命令に 1 つぐらい分岐が出てくる)
  - 分岐命令毎の予測ミス発生率 :  $Pm = 0.1$
- 分岐予測ミス・ペナルティ :  $Cp = 4$

## ■ 実行サイクル数 $Cr$ は

- $Cr = Ct + Nm \times Cp = 1M + 0.1M = 1.1M$
- 分岐予測ミスにより 10% 実行サイクル数が伸びている



# IPC で考えると

- 最終的な性能を考える上で IPC の方が都合がよい

- 実行サイクル数  $C_r$  を命令数  $N_i$  で正規化すると,

- $$\frac{C_r}{N_i} = \frac{C_t}{N_i} + \frac{(N_m \times C_p)}{N_i} = \frac{C_t}{N_i} + \frac{(N_i \times P_b \times P_m \times C_p)}{N_i} = \frac{C_t}{N_i} + P_b \times P_m \times C_p$$

- IPC は命令数を実行サイクル数で割ったもの = つまり上記の逆数

- $$IPC_r = \frac{1}{\frac{C_t}{N_i} + P_b \times P_m \times C_p} = \frac{1}{\frac{1}{IPC_t} + P_b \times P_m \times C_p}$$

- ここで  $IPC_r$  は実際の IPC,  $IPC_t$  は理想 IPC

# IPC の式のまとめ

■ 
$$IPC_r = \frac{1}{\frac{1}{IPC_t} + Pb \times Pm \times Cp}$$

- 実際の IPC :  $IPC_r$
- 予測ミスがなかった場合の理想 IPC :  $IPC_t$
- プログラム中の分岐命令の出現率 :  $Pb$
- 分岐命令毎の予測ミス発生率 :  $Pm$
- 分岐予測ミス・ペナルティ（サイクル） :  $Cp$
- （実行命令数が項から消えている

# スカラの5段パイプライン・プロセッサのIPC

- $$IPC_r = \frac{1}{\frac{1}{IPC_t} + Pb \times Pm \times Cp}$$

- ここで,  
理想 IPC       $IPC_t = 1,$   
分岐発生率     $Pb = 0.25,$   
予測ミス率     $Pm = 0.2,$   
ペナルティ     $Cp = 4$  (5段なので  $5 - 1 = 4$  サイクル) とすると,

- $IPC_r = \frac{1}{1 + 0.25 \times 0.2 \times 4} \simeq 0.83$
- つまり分岐予測ミスが無かった場合の理想 IPC と比べて  
17% ぐらい IPC が落ちている

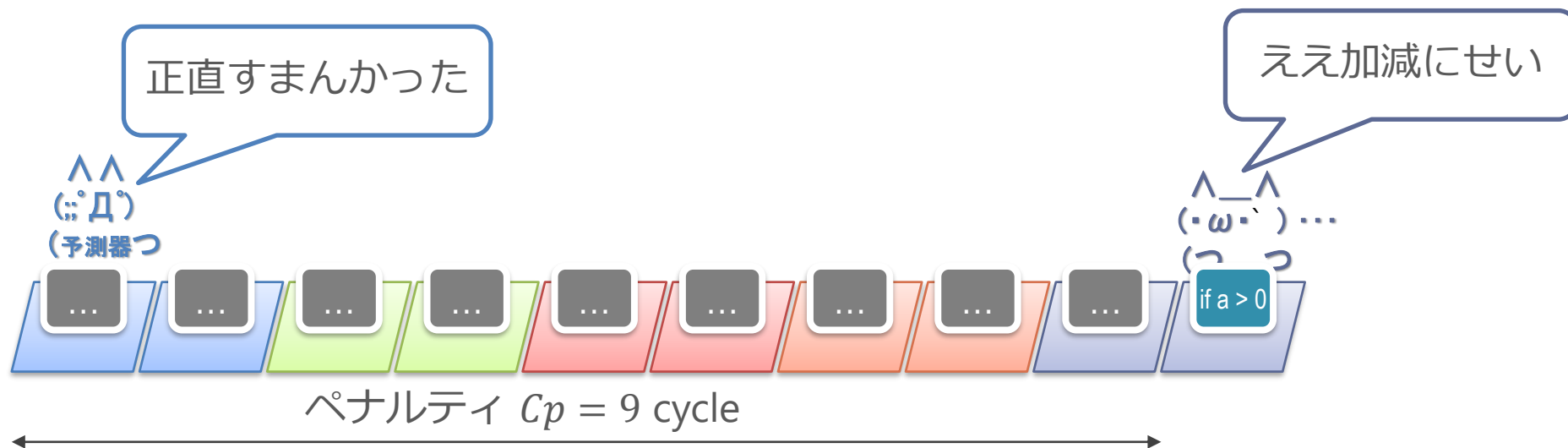
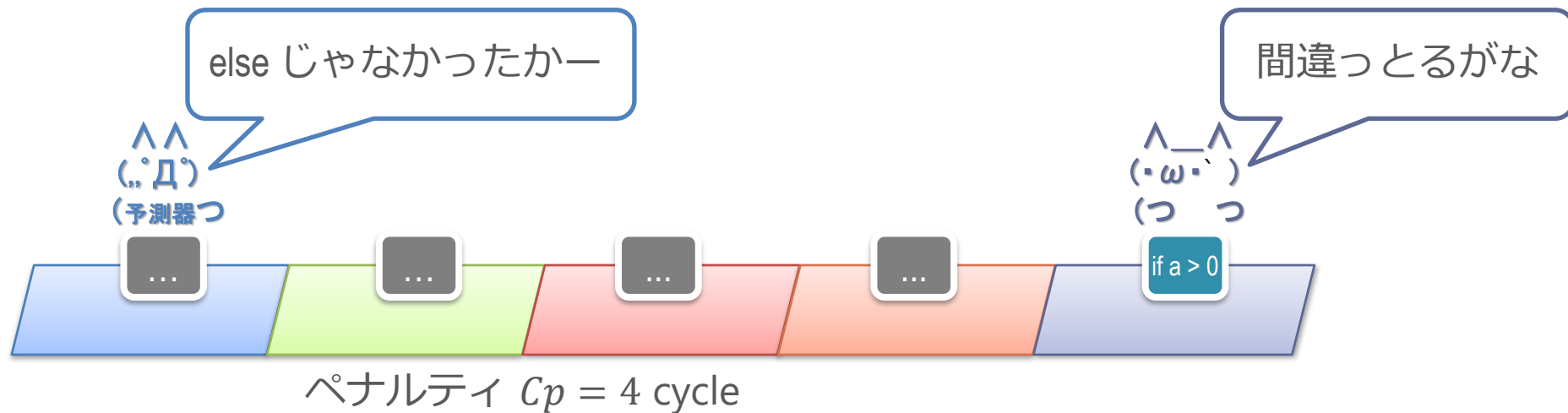
# パイプライン段数を2倍にした時のIPC

■ 
$$IPC_r = \frac{1}{\frac{1}{IPC_t} + Pb \times Pm \times Cp}$$

■ パイプライン段数を2倍にすると...

- $IPC_t = 1$ ,  $Pb = 0.25$ ,  $Pm = 0.2$  は変わらない
- ペナルティは  $Cp = 9$  (10段なので  $10 - 1 = 9$ ) に
- $IPC_r = \frac{1}{1 + 0.25 \times 0.2 \times 9} \simeq 0.69$
- 分岐予測ミスが無かった場合の理想IPC と比べて31% ぐらい性能が落ちている
  - パイプライン段数が5段の時よりIPC低下が大きい

# パイプライン段数が深い時は予測器の精度が重要



# パイプライン段数を2倍にした時の性能

- 全体の性能は周波数×IPCで決まる
  - 段数を倍にしたことで、周波数は2倍に
  - IPCは0.69
  - 性能は  $2 \times 0.69 = 1.38$

# パイプライン段数を2倍にした時のIPC

## ■ まとめると,

- 5段パイプの理想的な性能 : 1
- 分岐予測ミスありの性能 : 0.69
- 10段パイプの理想的な性能 : 2
- 10段で予測ミスあり : 1.38 (0.69 の 1.66倍)

## ■ わかること :

1. パイプライン段数を深くするとIPCへの影響が増加する
  - 同じ分岐予測ミス発せ率でも, より大きくIPCが下がる
2. 予測ミス率0.2で10段パイプだと, 全体の3割ぐらいの時間は無駄な実行をしている
  - $1.38 / 2 = 0.69$

# スーパスカラにした時の IPC

■ 
$$IPC_r = \frac{1}{\frac{1}{IPC_t} + Pb \times Pm \times Cp}$$

■ スーパスカラにすると...

- $Pb = 0.25$ ,  $Pm = 0.2$ ,  $Cp = 4$  は変わらない
- パイプラインが2本に増えたので  $IPC_t = 2$  に

- $$IPC_r = \frac{1}{\frac{1}{2} + 0.25 \times 0.2 \times 4} \simeq 1.42$$

- 分岐予測ミスが無かった場合の理想 IPC に比べると大幅に低い



# スーパスカラにすると、その分取り消される命令が増える

else じゃなかったかー

間違っとするがな

^^  
(.°Д°)  
(予測器つ

^^  
(.ω°)  
(つ つ



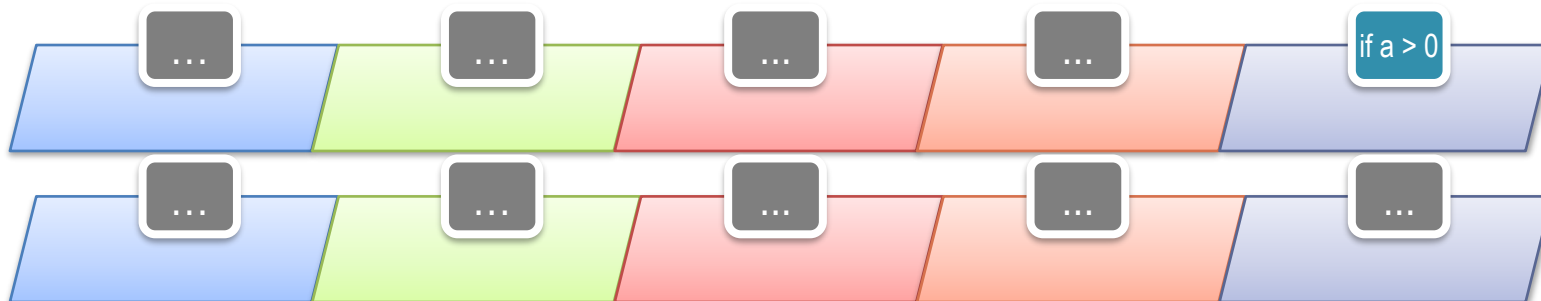
取り消される命令=4命令

大量廃棄やな・・・

また違うやん

^^  
(.°Д°)  
(予測器つ

^^  
(.ω°)  
(つ つ

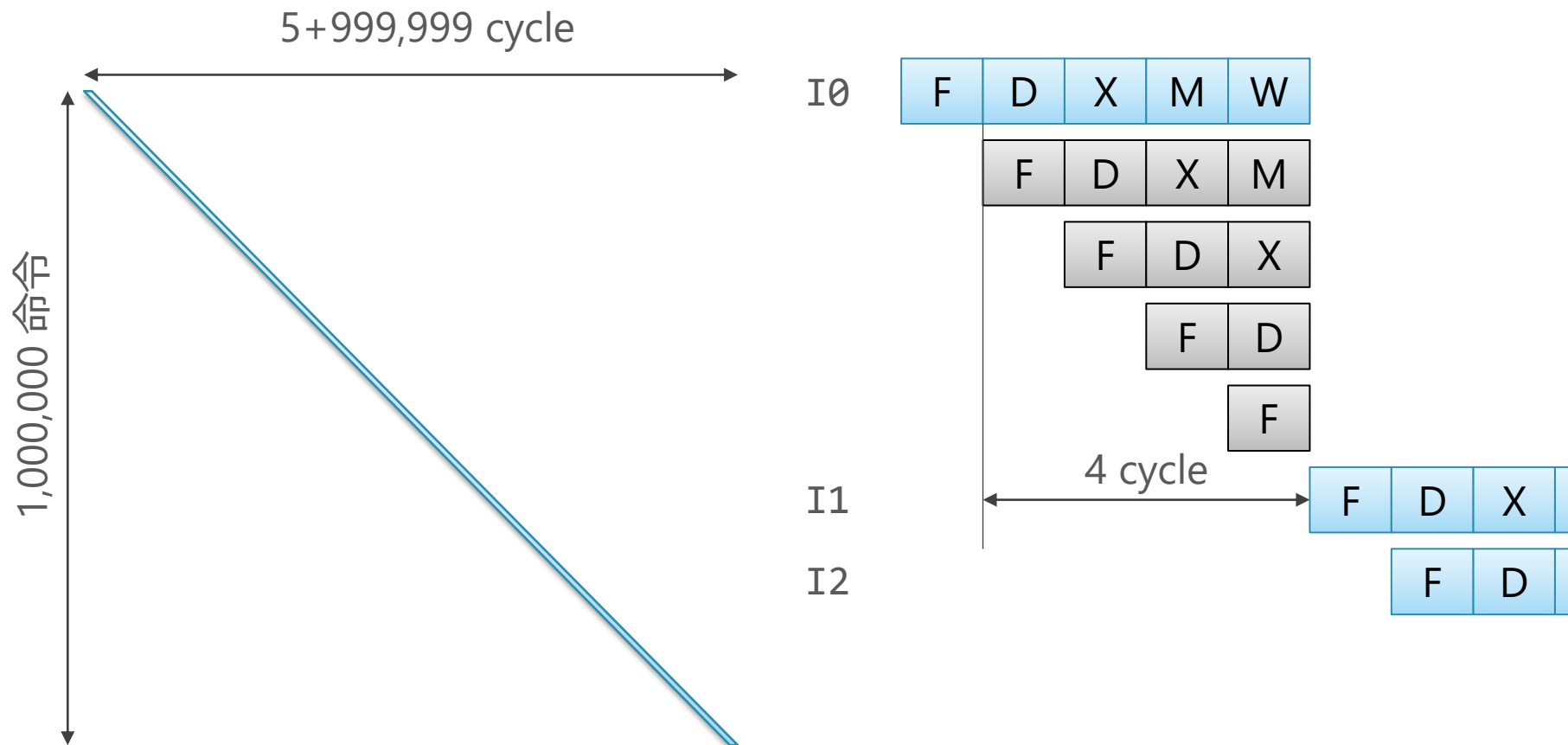


取り消される命令=9命令

# 分岐予測ミスによる IPC の低下のまとめ

- 以下のような場合に、より予測ミスの影響が大きくなる
  - パイプラインを深くする（周波数を向上させる）場合
  - スーパスカラにしてパイプライン本数を増やす場合
- 裏を返すと…
  - パイプラインを深くしたり本数を増やす回路を追加する代わりに、予測器を強化した方が性能が上がることも
  - バランスが大事
    - へちよい分岐予測器のまま他を強化しても無駄と言う事も

# パイプラインの長さと性能

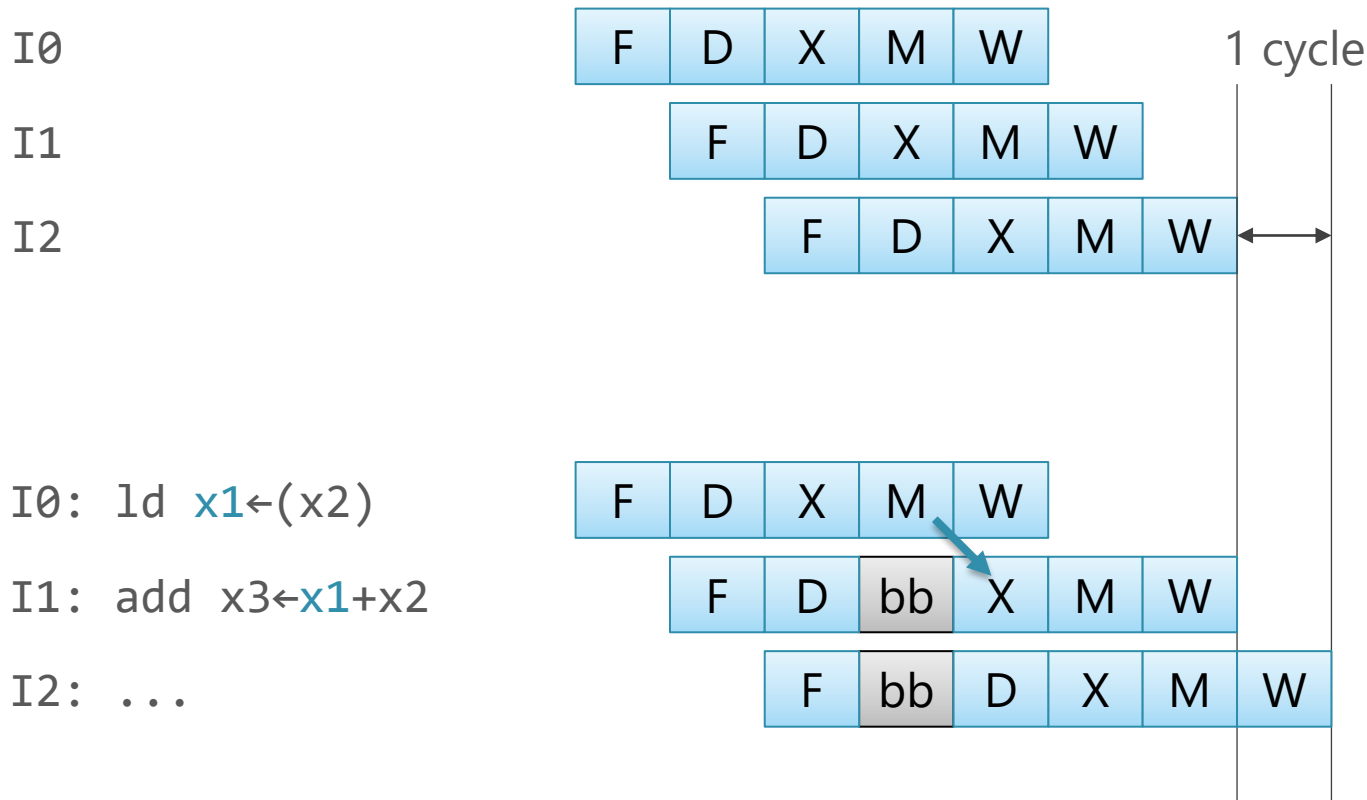


- パイプライン段数は、理想的なパイプラインではほとんど性能に影響しない
  - 最初の 1 命令分しか実行時間に影響しないから
- 分岐予測ミスは、起きる度にパイプラインの長さが表面化する

# ハザードによる IPC の低下

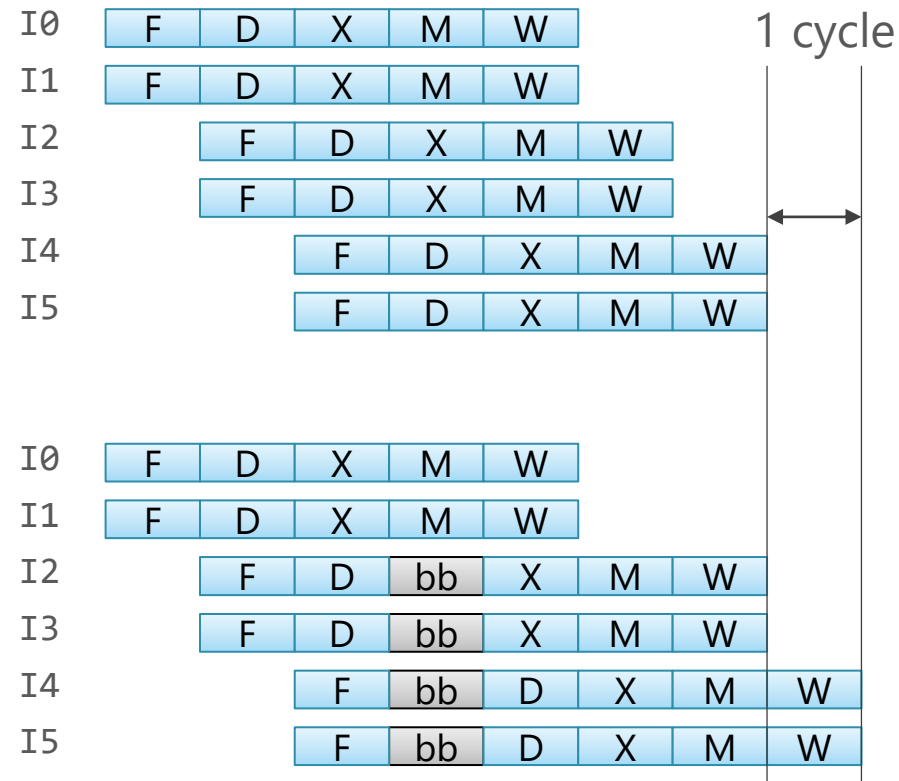
- 以下を例として、具体的な IPC の数字の変化を説明
  - 分岐予測ミス
  - ロードに依存した命令によるデータハザード

# ロードに依存した命令によるデータハザード



- パイプライン全体が、理想的な場合と比べて1サイクル遅れている

# スーパースカラの場合も同じ



- パイプライン全体が、理想的な場合と比べて1サイクル遅れている

# ロードによるデータハザードの実行サイクルの増加のモデル

## ■ 以下のようにおいた場合,

- ミスがない理想的な実行の際のサイクル数 :  $C_t$
- ロードのデータハザードの発生回数 :  $N_m = N_i \times P_l \times P_h$ 
  - 実行命令数 :  $N_i$
  - ロードの出現率 :  $P_l$
  - ロード命令毎のハザード発生率 :  $P_h$
- ハザード時のサイクル数の増加 :  $C_p$

## ■ 実行サイクル数 $C_r$ は, 理想サイクル数 $C_t$ に対して,

- $C_r = C_t + N_m \times C_p$
- 分岐予測ミスの時とほぼ同じ

# 性能のモデルの一般化

---



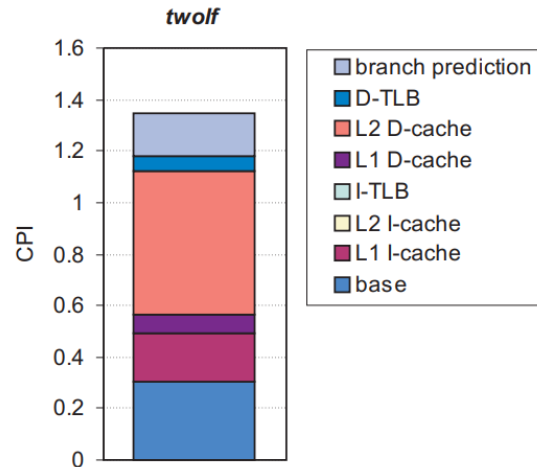
# 一般化できる

- 以下のようにおいた場合,
  - 理想的な実行の際のサイクル数 :  $C_t$
  - 何らかのハザードの発生回数 :  $N_h = N_i \times P_i \times P_h$ 
    - 実行命令数 :  $N_i$
    - ハザードを起こす命令の出現率 :  $P_i$
    - その命令毎のハザード発生率 :  $P_h$
  - ハザード時のサイクル数の増加 :  $C_p$
- 実行サイクル数  $C_r$  は, 理想サイクル数  $C_t$  に対して,
  - $C_r = C_t + N_h \times C_p$

# 一般化できる

- ハザード a, ハザード b, ハザード c ... とあった時
- 実行サイクル数  $C_r$  は, 理想サイクル数  $C_t$  に対して,
  - $C_r = C_t + N_{ma} \times C_{pa} + N_{mb} \times C_{pb} + N_{mc} \times C_{pc} + \dots$
  - これを積み上げ棒グラフで表したのが CPI Stack

# CPI Stack



**Figure 1.** Example CPI stack for the *twolf* benchmark.

- Cycles Per Instruction (CPI) は 1 命令あたりのサイクル数
  - $Cr = Ct + Nma \times Cpa + Nmb \times Cpb + Nmc \times Cpc + \dots$  を命令数で正規化して図示
  - $Ct = 0.25$  ぐらいなので, 4 way スーパスカラが基準?
- どのようなハザードによって性能が決まっているのかを直感的に知ることができる

# Out-of-order スーパースカラ・プロセッサの場合

- Out-of-order スーパースカラ・プロセッサではもっと複雑
  - 長時間かかる命令による待ちがあった場合,  
それを待つ間に別の後ろにある命令を実行できる
  - 色々なハザードが同時並行で起きうる
- 単純な線形加算のモデルでは扱えない
  - $C_r = C_t + N_{ma} \times C_{pa} + N_{mb} \times C_{pb} + N_{mc} \times C_{pc} + \dots$
  - これはハザード発生時に CPU 全体がストールするからなり立つ
  - ハザード同士がオーバラップして起きると上手く扱えない
- 色々な方法が研究されている

# ここまでのまとめ

## ■ 性能は以下の 2 要素のかけ算で決まる：

1. クロック周波数
2. Instructions per cycle (IPC)

## ■ 色んなバランスが大事

- クロック周波数だけを上げてても IPC が上がらないとだめ
- スーパスカラの同時実行数を増やしても IPC が増えない事もある
- 予測器だけ強くしても IPC があまり上がらないことも

# 実際には構成をどう決めるのか？

- だいたい最初に以下あたりの条件が決まっている
  - 要求性能
  - 使える回路面積（チップの大きさ）
  - 許容出来る消費電力

# いろんな戦略がありえる

- 回路面積が一定の場合、どちらが良いか？
  - パイプラインの本数を増やす
  - 予測器を複雑にして精度を上げる
- 消費電力の上限が一定の場合
  - 消費電力はクロック周波数の2～3乗で増える
  - クロック周波数をあえて下げるかわりに、パイプライン本数を増やしてIPCを上げる
- 色々シミュレーションやモデル化して決める

# まとめ

- 性能は以下の 2 要素のかけ算で決まる：
  1. クロック周波数
  2. Instructions per cycle (IPC)
- 以下について検討
  - 理想的な場合の性能のモデル
  - ハザードを考慮した性能のモデル
- 性能は色々な要素の相互作用で決まる



## 課題 8

### ■ 以下のような条件を考える

- 10段のパイプラインを持つ 2-way スーパースカラプロセッサであり, 理想的には  $IPC=2$  で実行できる
- 全実行命令におけるなんらかのデータハザードの発生率は 0.2
- このデータハザード発生時は 1 サイクル実行時間が伸びるものとする
- 全実行命令における分岐命令の出現率は 0.2
- 分岐予測ミス率は 0.3

- ### ■ (1) この CPU を改良する際,
- 「3-way スーパースカラにする」
  - 「2-way のまま15段パイプラインにする」
  - 「2-way のまま分岐予測器を改良する」

のどれが最も性能が上がるかを性能を計算して検討せよ

- この CPU を 3-way スーパースカラにすると理想的には  $IPC=3$  で実行できるがデータハザードの発生率は 0.3 に上昇するとする
- また, 分岐予測器を改良すると分岐予測ミス率が 0.2 にまで削減されるとする

## 課題 8

- (2) ここで効率を表す「性能エネルギー比（性能 / 消費エネルギー）」という指標を導入する。この数字が大きいほど小さなエネルギーで速く動くことを意味する。前述の3つの改良方針のうち、以下の追加の条件のもとで、最も性能エネルギー比が良いものはどれかを計算して検討せよ。
  - クロック周波数を  $N$  倍にすると、消費エネルギーは  $N^2$  倍増える
  - 3-way スーパースカラにすると消費エネルギーは 1.5 倍になる
  - 分岐予測器を改良すると消費エネルギーは 1.1 倍になる

# 提出方法

## ■ 以下を提出：

### 1. 課題 8：

- 提出は Moodle の「課題 8」のところからお願いします
- 紙に書いた場合は写真を撮ってアップロードしてください

### 2. 感想や質問：

- 「感想や質問」のところに投稿してください
- わからない場所がある場合、具体的に書いてもらえると良いです

## ■ 提出締め切り

- Moodle に設定した締め切りまで（6/25 日曜日の 23:59 頃、要確認）

## ■ 注意：

- 課題の出来は、ある程度努力したあとがあれば良しです
  - 必ずしも正解していなくても良いです

# 質問とか感想

---

# 質問とか感想

- C言語やJava等は静的プログラミング言語、Python等は動的プログラミング言語と言われますが、その所以がわかりました。（静的命令スケジューリングor動的命令スケジューリングを行うかどうか）
  - 静的や動的，までは言葉の意味が同じなんだけど，スケジューリングを行うかどうかではなかったりです
- Unityで使用するため、C#を少しだけ学んでいるのですが、C#やjavaのstaticと静的命令スケジューリングには関係がありますか？
  - コンパイル時に決めて実行時に変化させないことを静的とかstatic といいます

# 質問とか感想

- 51ページのスーパスカラ・プロセッサの例でALU 2 で演算した結果がデータ・メモリにいかずに書き込みデータ 1 に行ってますが、なんでデータ・メモリにいかないのか教えていただきたいです。
- 「上の方の ALU でのみメモリアクセス命令ができる」という構造にして構造を単純化しています

# 質問とか感想

- 今回の課題はかなり難しかったです。beqやオペランド、フラッシュなどの言葉はせめてその講義資料に詳しく説明を書いて欲しかったです。そもそもの問題文の意味がわからないと解けるものも解けないなと思いました。(私の勉強不足ですが、、、すみません)
- また、このような問題は期末テストにも出す予定ですか？？
- これはある程度は書いていたつもり：
  - 「ここで beq はオペランドが等しい時に分岐する分岐命令である」  
「beq が分岐予測ミスを起こして LABEL に飛んだあとにフラッシュされてやり直した場合を想定せよ」
    - 最初の文は、これが beq 命令の意味の説明になってるはず
    - 「オペランド」は第2回の講義で説明して、以降も使っていたはず
    - 「フラッシュ」は確かに言ってなかったかも。すみません
      - ◇ 講義中では「取り消す」と言っていたと思う

# 質問とか感想

- 動的命令スケジューリングについて、性能で考えるとout of order実行を使った方が良いのではと思ったのですが、in order実行の方が優っているケースなどはあるのでしょうか。
- 現在のプロセッサではin-orderに使われることは珍しいですか？
  - 簡単な分、小さく安く省電力に作れるので、性能がいらないときにはよく使われます



# 質問とか感想

- 今回の、最初に講義をして、質問を最後にする授業だと集中できるので良かったです！
- 前回の授業の進め方とてもありがたいです。

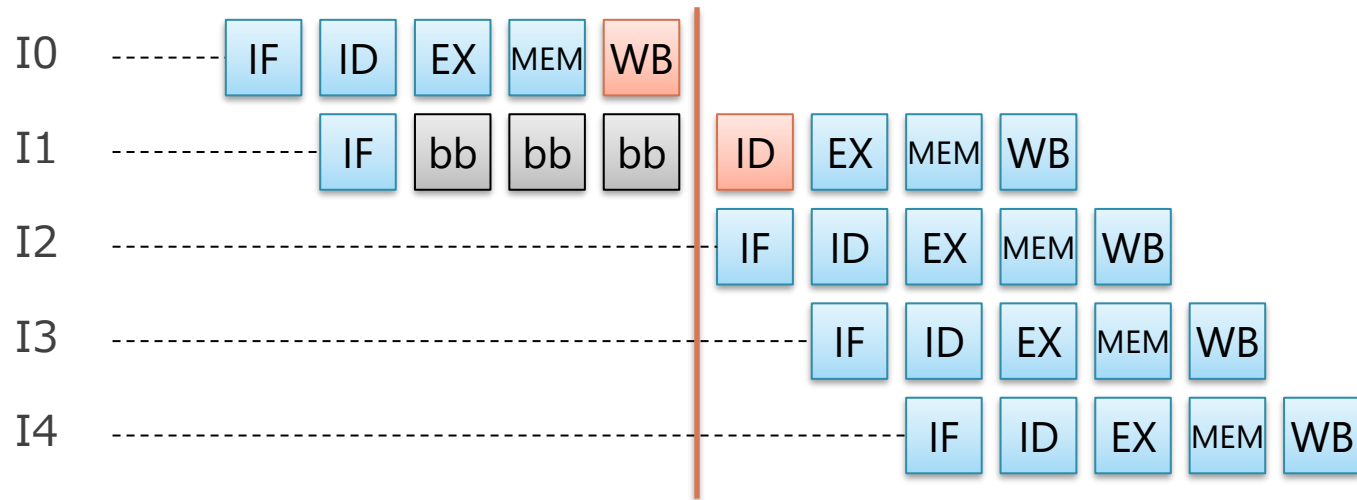
# 質問とか感想

- 今回の講義の内容では無いのですが、前回の課題6.1(4)の解説について、RV32I Base Instruction Setの表を見るとrs2が左でrs1が右となっているため、x3とx4の順番は逆（0000000 00010 00011 000 00010 0110011→0x00218133）ではないのでしょうか。
- すいません、その通りでした

# 質問とか感想

- 講義資料の図を見ていると、ストールした際のバブルがフェッチの後・デコードの前に入る場合とデコードの後・演算の前に入る場合がありますが（第6回p78のI1と第7回p93のI2など）、前者はフォワーディングをしておらず、後者はフォワーディングをしていることによる違いでしょうか。
  - その通りです

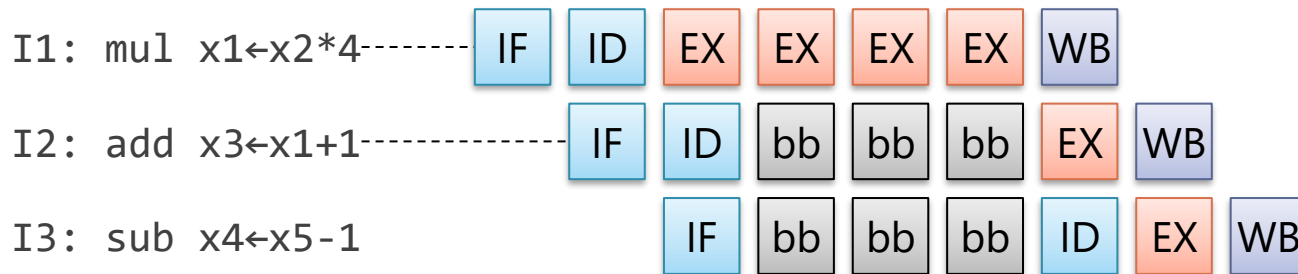
# 1. ストールさせる



- I0 の WB が終わるまで、後続の命令を遅らせる
  - I1 の ID が、I0 の WB の右にくるまでストール
  - I1 は I0 の結果を使える
- 欠点：プログラムの実行がとても遅くなる

# in-order 実行と out-of-order 実行の違い

- I1 の mul は4サイクルかかる
  - In-order 実行だと, I3 は I1 に依存していないが待たされる
  - プログラムに書かれた順に実行するため



# 質問とか感想

- 課題に取り組んでいてわからなくなってしまったのですが、MEMやWBそれぞれにおいて異なるレジスタでも同時に読み込み、書き込みはできないけど、WBとIDは重なっても読み込み、書き込みは同時にできるのはなぜですか？
- レジスタファイルが同時に読みと書きができるように作られているからですが、ちょっと質問の意味がわかってないです

# 質問とか感想

- 静的スケジューリングについて、コンパイラが頑張って作っているという話でしたが、どのような感じで作っているのでしょうか？手当たり次第に交換してうまくいくか試しているのでしょうか？
- 内部で一種のシミュレーションをしてスケジュールするなど、色々な方法があります

- (5)の四つ目の命令は3つ目の命令でx2の演算を終えたら四つ目をスタートしてよいという解釈で合っていますか。つまり、3つ目の命令のXが終わったら四つ目の命令のDを始めるということです。



- 先週の課題の解説を聞いて、LABELとEXITは同じような扱いでよいのかと気になりました。
  - ラベルは変数みたいなもので、任意の名前をつけられます
  - C 言語の goto で使うラベルと同じです

# 質問とか感想

- スライド69で、書き込むレジスタをX1からX4に置き換えてしまうと、順番は入れ替えても大丈夫なようになるかもしれないが、X1の値を、「 $X2 + 1$ 」から「 $X3 \pm 1$ 」に書き換えるという内容であったときに $X1 \leftarrow X4$ という工程が増えてしまうので結局速度的には何も変わらないのではないかと（むしろ遅くなることもあり得るのではないかと）思いました。
  - これは x1 に書き込んでいた命令の書き込み先を x4 にできれば、工程は増えない
- スライド78で分岐を乗り越えた並び替えが生じることがあると書いていたので、依存関係のある流れ全体をまとめて命令1つとした方がいいのではないかと考えました。

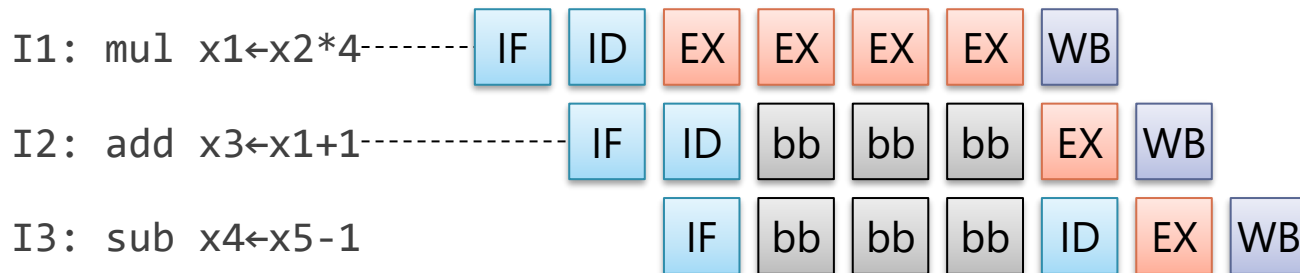
- パイプラインプロセッサで、命令と次の命令に依存関係がある時は、基本的に前の命令の演算(X)が終わったあとなら次の命令の演算ができるが、前の命令がldの時はメモリアクセス(M)まで待ってから次の命令の演算をするという解釈で合っていますか？

# 質問とか感想

- 前回のスライド p 9 3 にかかっている I 2 の I D は I 1 の W B が終了しないと処理できないと思っていたのですが、違うのでしょうか。これは E X がかぶらなければいいということですか。
  - フォワーディングを前提にしています
- 1つ前で代入した数を演算で使いたい場合、1つ前の書き込みと今やりたい演算を同時に行うことは可能なのでしょうか。
  - フォワーディングすればできます

# in-order 実行と out-of-order 実行の違い

- I1 の mul は4サイクルかかる
  - In-order 実行だと, I3 は I1 に依存していないが待たされる
  - プログラムに書かれた順に実行するため



- 79ページのaとbが同じ場所を指している可能性がある、と言うのがよくわかりませんでした。具体的にどのような状況ですか。
  - `int i = 0;`  
`func(&i, &i, 0);`  
とした場合, a と b は同じ場所をさす

# 静的スケジューリングが難しい例 2

## ■ 例 2 : ポインタ参照の順番を入れ替えるのは難しい (不可能ではない)

- 2 行目と 3 行目のメモリ・アクセスを入れ替えることは困難
- うかつにやると意味が変わる

```
1: func(int* a, int* b, int z){  
2:     *a = z + 1; // z+1 より先に c = *b を始めたい  
3:     int c = *b; // しかし a は b と同じ場所を  
                  // 指している可能性がある
```

# 質問とか感想

- 依存関係がある時のパイプラインの考え方がよく理解できていませんでした。
  - 例えば
  - `add x1 < -x2 + x3...` ①
  - `add x2 < -x1 + x4...` ②
- だと①が完了する前に②の実行が始まると①の結果が反映されないで①の実行が完了してから②の実行が始まりますか？



- フェッチやデコードの意味がイマイチ分かってなくて、課題をよく分からないまま進めてしまいました。フェッチやデコードの意味の解説と、どの命令でもFDXMWの工程を必ず踏むという認識でいいのか、教えていただきたいです。また周波数の求め方がよく分からなかったなので、解説していただけると助かります。

# 質問とか感想

- スーパースカラプロセッサは、2wayよりもたくさんの分岐を持つものも存在しますか？
- また、分岐数を増やすとデータ依存も増えるため、どこかで性能が頭打ちになるのでしょうか？

# 質問とか感想

- 私には授業のスピードが速いです。スライドを理解している間に置いて行かれてしまいます。

- 課題を解くのに例題をつけていただけるとありがたいです。自分では理解しているつもりでも、課題を解いている間にこの解き方であっているのかを確認する材料が欲しいです。