コンピュータ アーキテクチャ I　第９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

すいません

- 難易度のバランス調整を間違った気がします
  - かなり正答率が低かった

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

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： $\boldsymbol{Ct}$
  - 分岐予測ミスの発生回数： $\boldsymbol{Nm} = Ni \times Pb \times Pm$
    - 実行命令数： Ni
    - 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： $\boldsymbol{Cp}$
- 実行サイクル数 Cr は，理想サイクル数 Ct に対して，
  - $\boldsymbol{Cr = Ct + Nm \times Cp}$
  - 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：$\boldsymbol{Ct = 1M}$（メガ）
  - 分岐予測ミスの発生回数：　$\boldsymbol{Nm = Ni \times Pb \times Pm = 0.025M}$
    - 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：　$\boldsymbol{Cp = 4}$
- 実行サイクル数 Cr は
  - $\boldsymbol{Cr = Ct + Nm \times Cp = 1M + 0.1M = 1.1M}$
  - 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 Cr を命令数 Ni で正規化すると，
  - $\frac{Cr}{Ni} = \frac{Ct}{Ni} + \frac{(Nm \times Cp)}{Ni} = \frac{Ct}{Ni} + \frac{(Ni \times Pb \times Pm \times Cp)}{Ni} = \frac{Ct}{Ni} + Pb \times Pm \times Cp$
- 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
  - $IPCr = \frac{1}{\frac{Ct}{Ni} + Pb \times Pm \times Cp} = \frac{1}{\frac{1}{IPCt} + Pb \times Pm \times Cp}$
  - ここで $IPCr$ は実際の IPC，$IPCt$ は理想 IPC

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：Ct
  - 何らかのハザードの発生回数：　$Nh = Ni \times Pi \times Ph$
    - 実行命令数：　Ni
    - ハザードを起こす命令の出現率：　Pi
    - その命令毎のハザード発生率：　Ph
  - ハザード時のサイクル数の増加：Cp
- 実行サイクル数 Cr は，理想サイクル数 Ct に対して，
  - $Cr = Ct + Nh \times Cp$

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
  - それぞれおきた回数が $Nh..$，ペナルティが Cp … とする
- 実行サイクル数 Cr は，理想サイクル数 Ct に対して，
  - $Cr = Ct + Nha \times Cpa + Nhb \times Cpb + Nhc \times Cpc + \cdots$
- 実行命令数を Ni とすると，$IPCr = Ni/Cr$

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（１）

- 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

- 上記のベースライン CPU の 実行サイクル数は，実行命令数を Ni とすると
  - $Cr = Ct + Nma \times Cpa + Nmb \times Cpb$
  - $= \frac{Ni}{IPCt} + (Ni \times 0.2) \times 1 + (Ni \times 0.2 \times 0.3) \times 9$
  - $= \frac{Ni}{2} + Ni \times 0.2 + Ni \times 0.54 = 1.24 \times Ni$

- $IPCr$ は Ni を Cr で正規化したものなので，
  - $IPCr = \frac{Ni}{Cr} = \frac{Ni}{1.24 \times Ni} \approx 0.81$

課題８（１）

- ベースライン CPU の動作周波数を 1 とすると，その性能は
  周波数 1 × IPC 0.81 = 0.81

- 「3-way スーパスカラにする」
  - $IPCr = \frac{1}{\frac{1}{3} + 0.3 \times 1 + 0.2 \times 0.3 \times 9} \approx \frac{1}{1.17} \approx 0.85$
    - 先ほどは丁寧にサイクル数を 1 回計算したが，最初から「１命令あたりのサイクル数」として考えて IPC をいきなり求めても結果は同じ
  - 周波数は変わらず 1 なら性能も 0.85

課題８（１）

- 「2-way のまま15段パイプラインにする」
  - $IPCr = \frac{1}{\frac{1}{2} + 0.2 \times 1 + 0.2 \times 0.3 \times \mathbf{14}} \approx \frac{1}{1.54} \approx 0.65$
    - 分岐予測ミスペナルティが 10 から 15 に
  - 10 → 15 段になったことにより周波数が 1.5 倍に
  - 性能は 1.5 × 0.65=0.97
- 「2-way のまま分岐予測器を改良する」
  - $IPCr = \frac{1}{\frac{1}{2} + 0.2 \times 1 + 0.2 \times \mathbf{0.2} \times \mathbf{9}} = \frac{1}{1.1} \approx 0.94$
  - 周波数は変わらないので，性能も 0.94

課題８（１）

- まとめると，
  - ベースライン：0.81
  1. 3-way スーパスカラにする：0.85
  2. 2-way のまま15段パイプラインにする：0.97 最も速い
  3. 2-way のまま分岐予測器を改良する：0.94

課題 8（2）

- (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

課題 8（2）

- ベースラインの消費エネルギーを 1 とすると：0.81/1=0.81

1. 3-way スーパスカラにする：$0.85/1.5 \approx 0.57$
2. 2-way のまま15段パイプラインにする：$\frac{0.97}{1.5^2} \approx 0.43$
3. 2-way のまま分岐予測器を改良する：$\frac{0.94}{1.1} \approx 0.85$

メモリと，キャッシュの基本

今日の内容

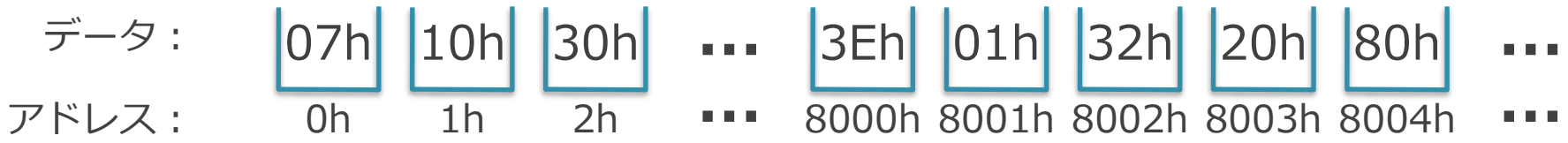
1. メモリ
2. キャッシュの基本

メモリ

メモリ

- メモリ：RAM（Random Access Memory）
  - 複数のデータを記憶する回路
  - 配列のように，位置を指定して読み書きする

メモリ

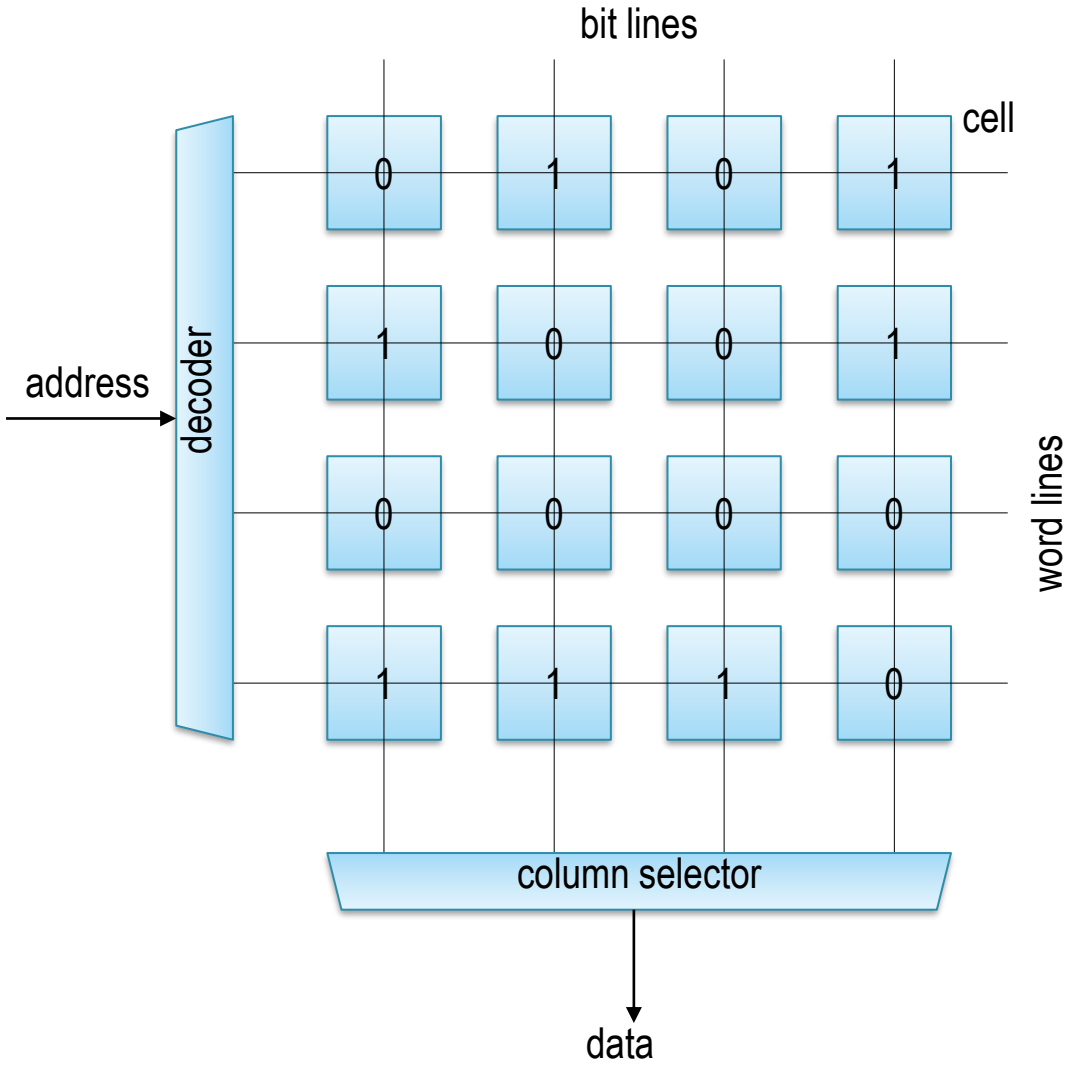


- メモリは命令列と，計算するデータを保持する
  - 単一の巨大な配列があると思えばよい
  - C 言語の配列は，これを切り出してユーザーに見せている
- 数字が入る箱がたくさん並んでいるイメージ
  - アドレス：箱の通し番号（住所）
  - データ　：箱の中身の数字

メモリ

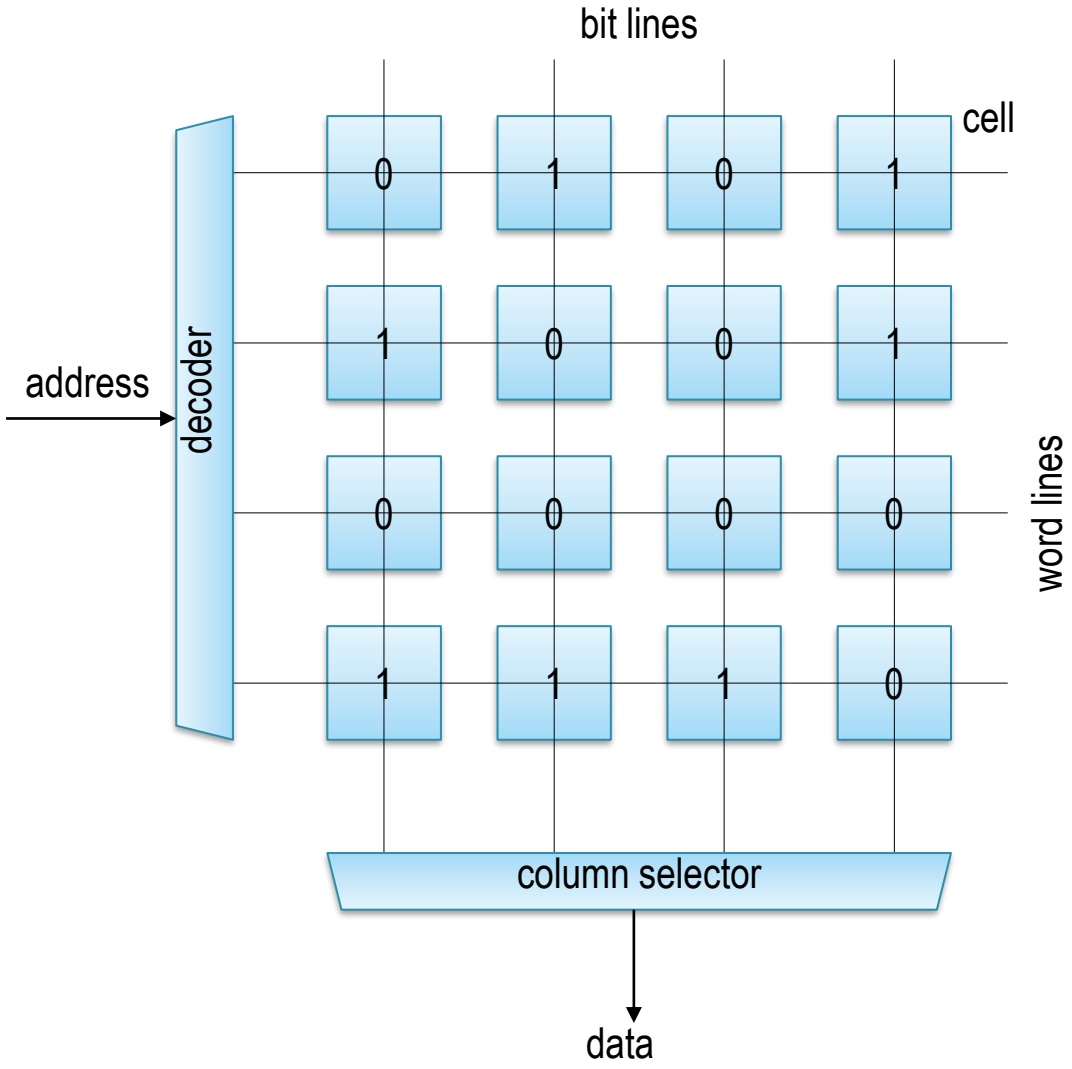
1. **メモリの基本**
    1. 構造
    2. 動作
    3. 容量と速度
2. メモリの詳細
    1. SRAM と DRAM
    2. なぜメモリが存在するのか？

メモリの基本構造：セルを行列状に配置



- セル：
  - 1ビットの情報を記憶
- ビットライン：
  - 列方向の配線
- ワードライン：
  - 行方向の配線
- デコーダ：
  - アドレスをデコードしてワードラインをアサート
- カラム・セレクタ：
  - ビットライン出力から１つを選んで出力

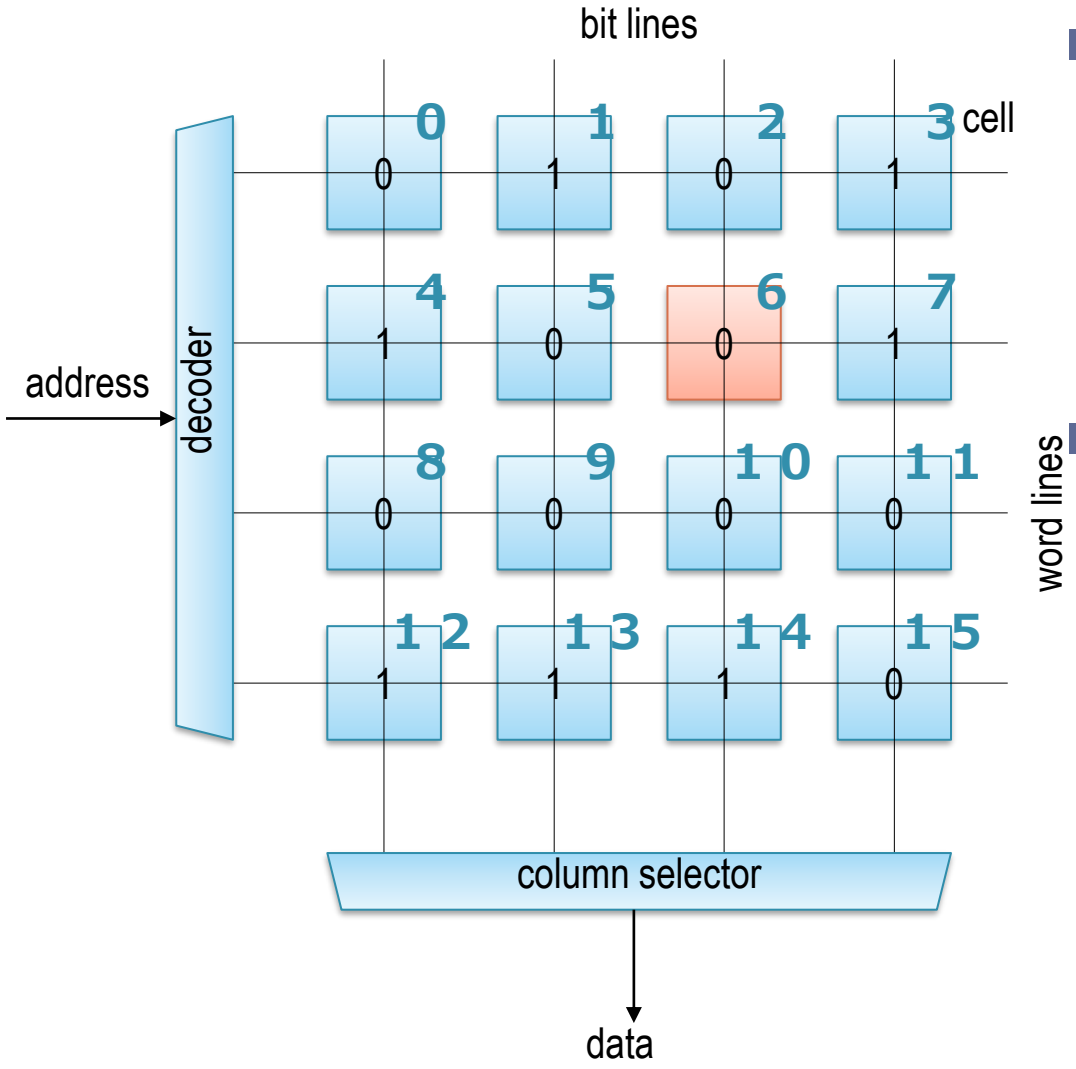
メモリの読み出し操作



1. アドレスのデコード
2. ワードラインのアサート（行の指定）
3. ビットラインへの１行分のデータの読み出し
4. カラムセレクタによる目的のビットの選択

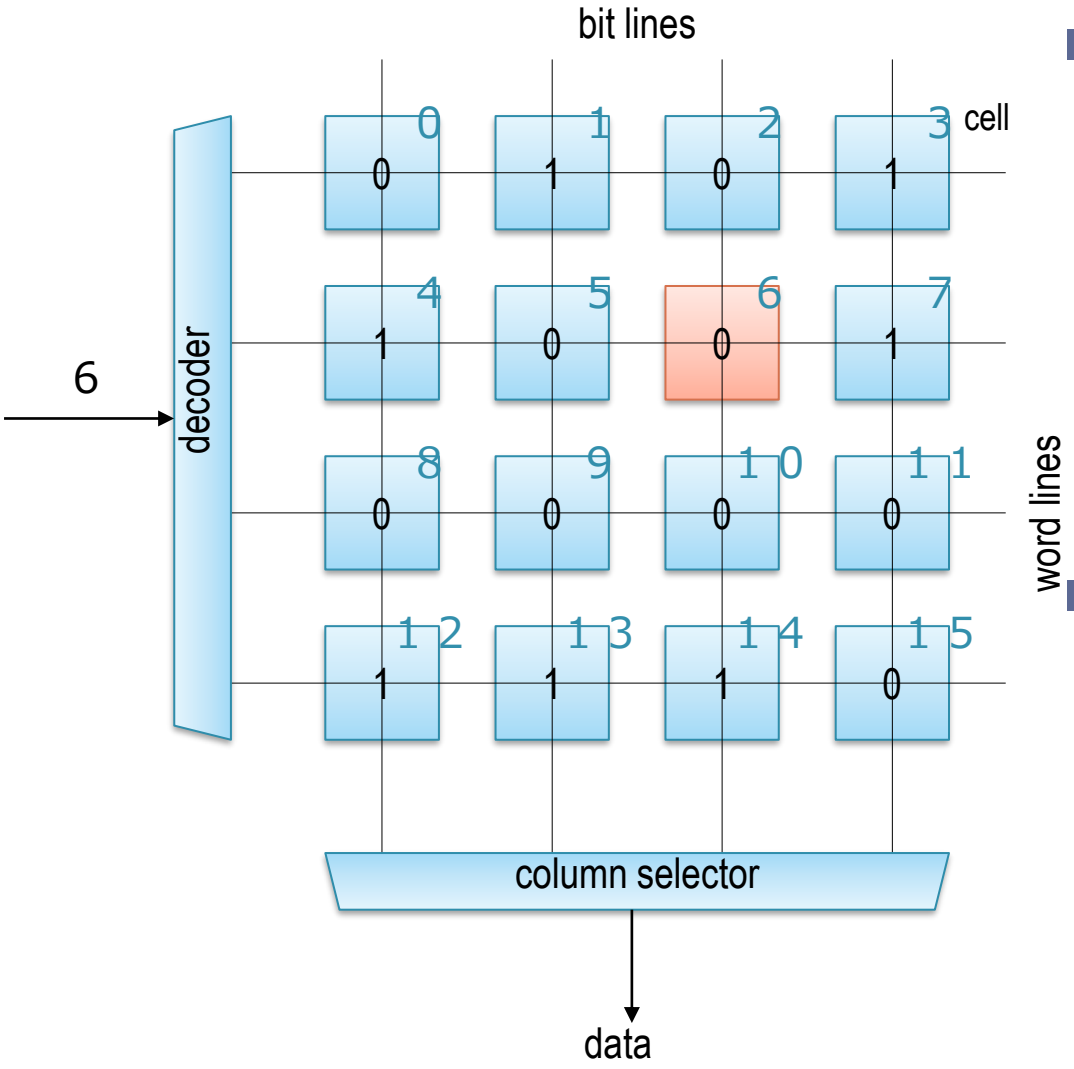
- （書き込みはこれの逆を行う

メモリの読み出し動作の例



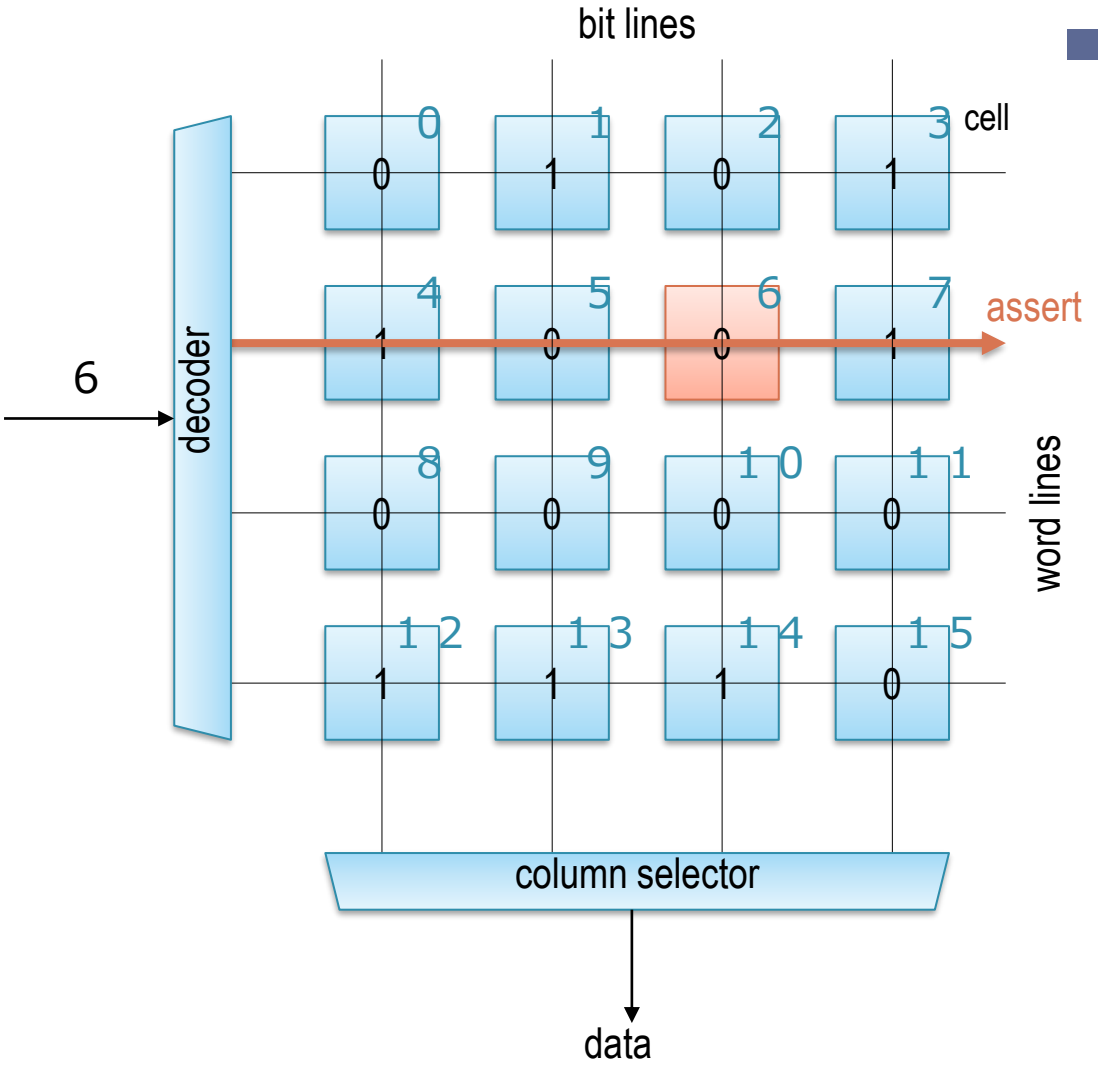
- 要素数 $16 = 2^4$
  - アドレスは4ビット
  - 各セルの右上の数字がアドレス
- このメモリの読み出し操作を例として説明
  - アドレス６のセルを読み出す

メモリの読み出し動作（１）
アドレスのデコード



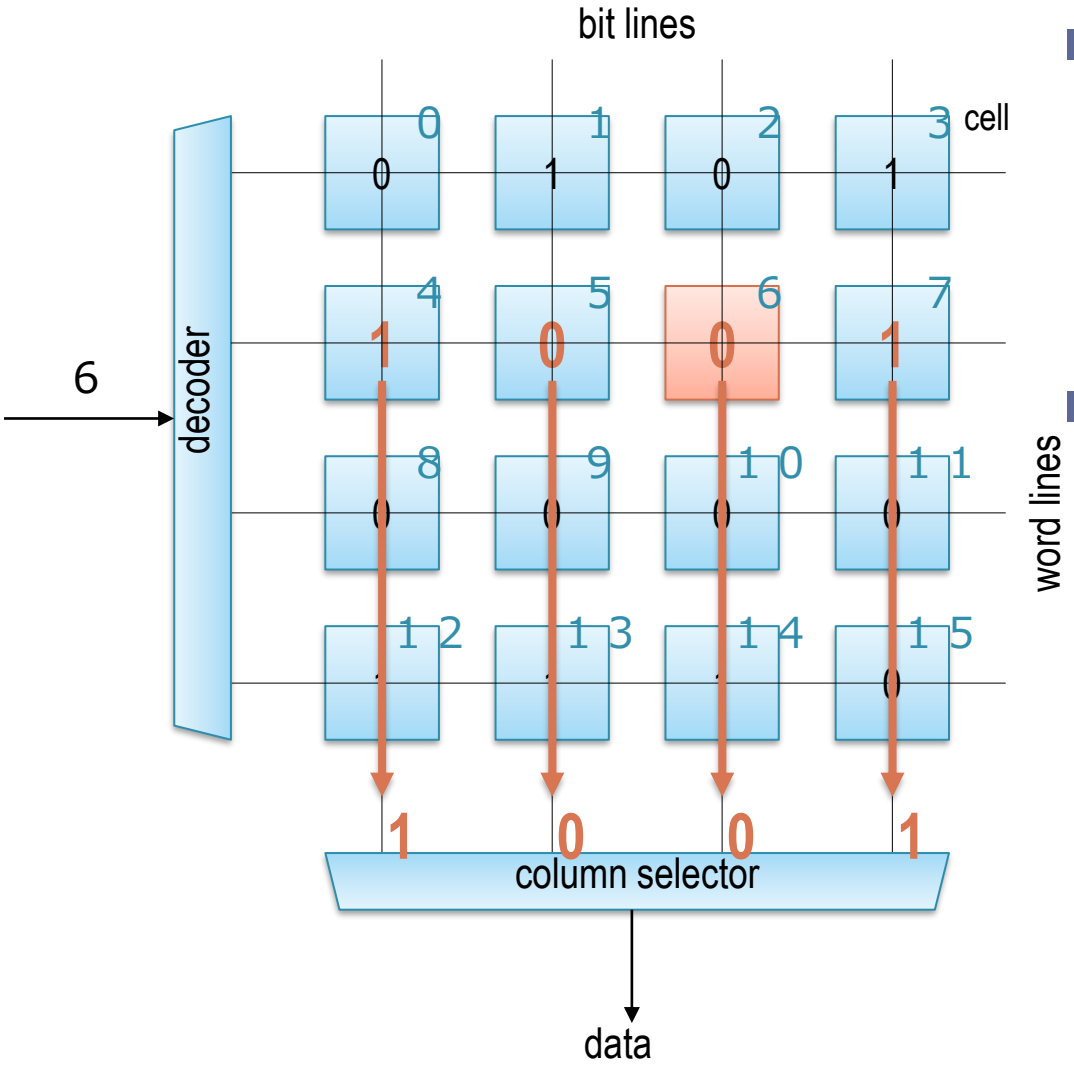
- どの行を読むか決める
  - 各行は４つセルがある
  - アドレスを４で割って切り捨てた行目が読むべき場所
  - 6 =0110（２進数）
- デコーダ
  - 数字を対応するワンホット信号に変換する回路
  - ワンホット信号：
    - n 本のうち，１つだけが１で他が０の信号
  - アドレス上位の２ビットをデコード

メモリの読み出し動作（２）
ワードラインのアサート



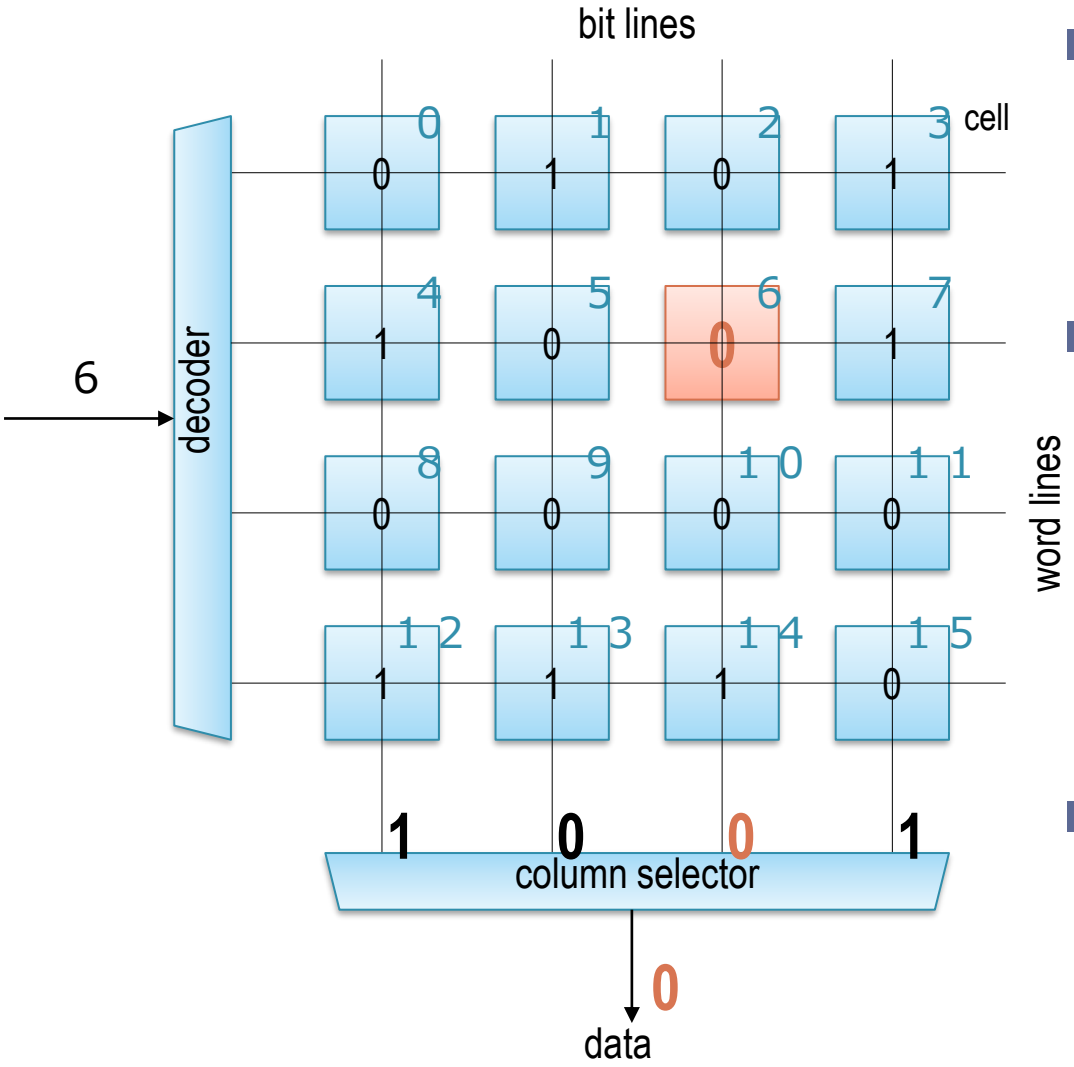
- 読み出す行に対応したワードラインをアサート
  - 先ほどのデコード結果を使う

メモリの読み出し動作（３）
ワード（１行分のデータ）の読み出し



- ワードラインがアサートされると，
  そこに接続されたセルがビットラインに自身の中身を流す
- これにより，１行分のデータが読み出される

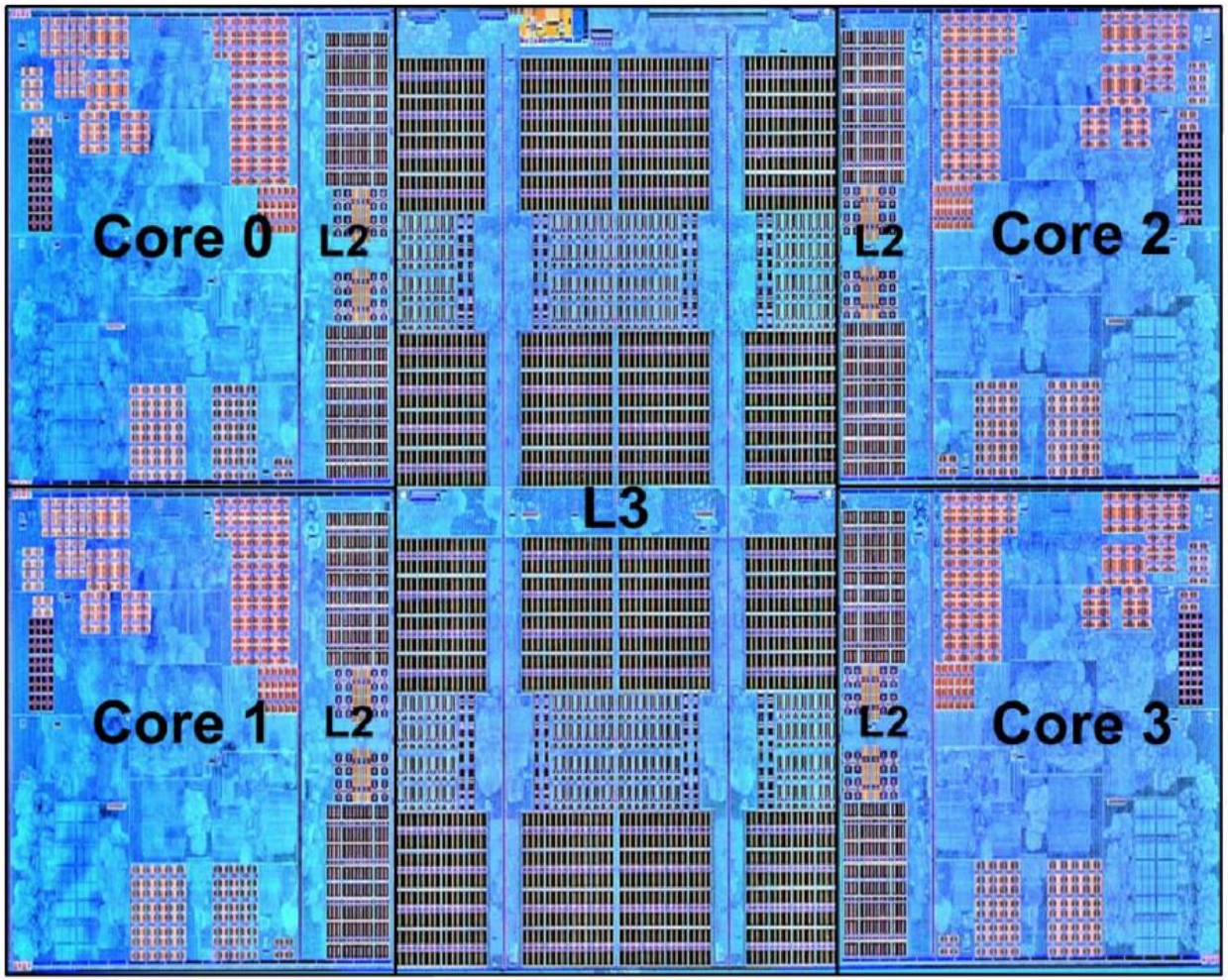
メモリの読み出し動作（４）
列の選択



- 読み出された１行分のデータから，最終的に必要な１ビットを選択
- どの列を読むか決める
  - 各列は４つセルがある
  - アドレスを４で割った余りの列が読むべき場所
  - ６=0110（２進数）
- カラム・セレクタによりビットを選択する
  - 読み出し幅によってはこの部分がないこともある

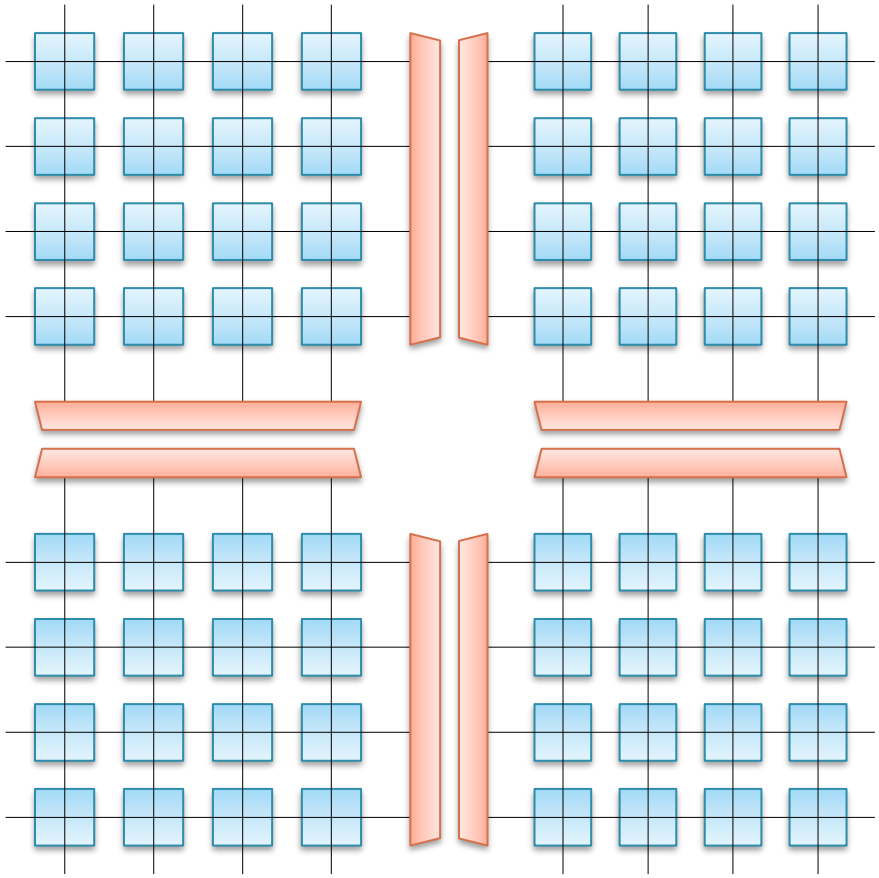
AMD Zen という CPU のチップ写真

Teja Singh et al., Zen: An Energy-Efficient High-Performance ×86 Core より



- 赤や紫の「田」の字の構造の部分が全部メモリ（SRAM）
  - レジスタやキャッシュ，各種テーブルなど

「田」の字の構造



- 「田」の字の構造になっているのは，
  - 縦線がデコーダで左右に対してワードラインをアサート
  - 横線でビットラインのデータを拾う

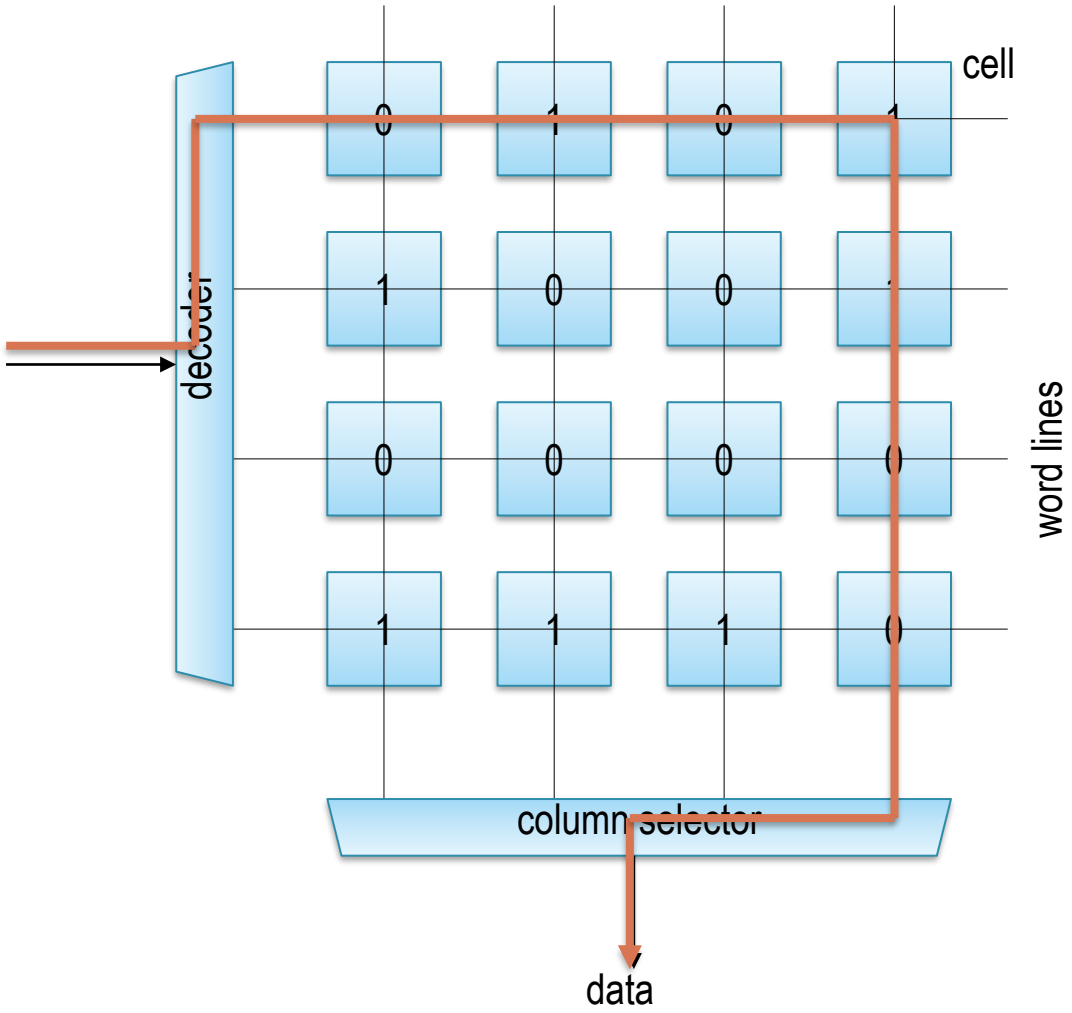
メモリ

1. メモリの基本
    1. 構造
    2. 動作
    3. **容量と速度**
2. メモリの詳細
    1. SRAM と DRAM
    2. なぜメモリが存在するのか？
    3. マルチポート・メモリ
    4. メモリを対象にしたアタック

メモリの性質

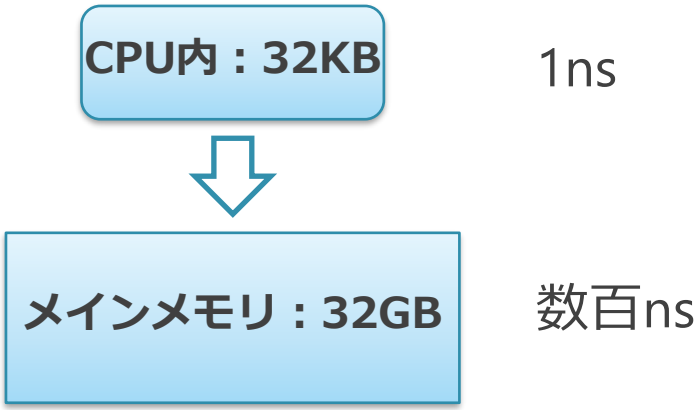
- 容量（大きい or 小さい）
  - どのぐらい数のデータを覚えることができるか
- 速度（速い or 遅い）
  - どのぐらいの速さで読み書きできるか
- 容量と速度にはトレードオフがある
  - 「大きくて速くいメモリ」は作ることができない

アクセス時間は容量の平方根ぐらいに比例



- 格子状に並んだセルの外周部の信号線の長さがアクセス時間を決める
- 容量（セルの数）が一定の場合，正方形に近くするのが最も経路が短くなる
- メモリ容量
  = (外周部の長さ)^2

アクセス時間は容量に直接は比例しない
（容量の平方根ぐらいに比例



- 格子状に並んだセルの外周部の長さがアクセス時間を決めるから
  - メモリ容量 = (外周部の長さ)^2
  - アクセス時間は，容量に対して線形には伸びない

速度

- 信号線の長さがなぜ問題に？
  - 電気信号が伝わる速度はとても速いのでは？
- 計算してみる：
  - 光の速度：　　秒速 30万Km ＝ 3×10^8 m
  - CPU の動作周波数：　　3GHz　　＝ 3×10^9 Hz
  - 1回の処理で光が進める長さ = (3×10^8) / (3×10^9) = 0.1m = 10cm
- 光の速度でも大して進めないぐらい，今のコンピュータは速い
  - 回路中の電気信号の伝達は光よりもだいぶ遅い

メモリの速度と容量のまとめ

- 速さと大きさにはトレードオフがある
  - 「大きくて遅いメモリ」か「小さくて速いメモリ」しか作れない
  - 容量を大きくすると，その分メモリ内の信号線が伸びる
- 実際には，セルをどのような方式で作るかでも大きく変わる
  - しかし，上記のトレードオフは常になりたつ

メモリ

1. メモリの基本
    1. 構造
    2. 動作
    3. アクセス時間

2. **メモリの詳細**
    1. SRAM と DRAM
    2. なぜメモリが存在するのか？

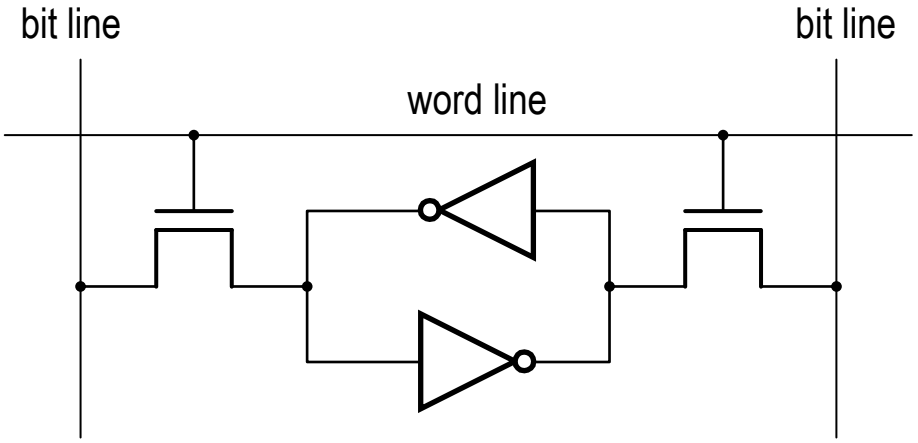
メモリのバリエーション

- メモリは基本的にはみな同じ構造を取る
  - 情報を記憶するセルの中身が方式ごとに異なる
- SRAM と DRAM を簡単に紹介
  - 現代のコンピュータはおおよそこの２種類で作られている
  - この講義的には性質の大ざっぱな違いがわかっていればよい
  - （細かい動作まではあまりわかってなくても良い

Static Random Access Memory (SRAM)

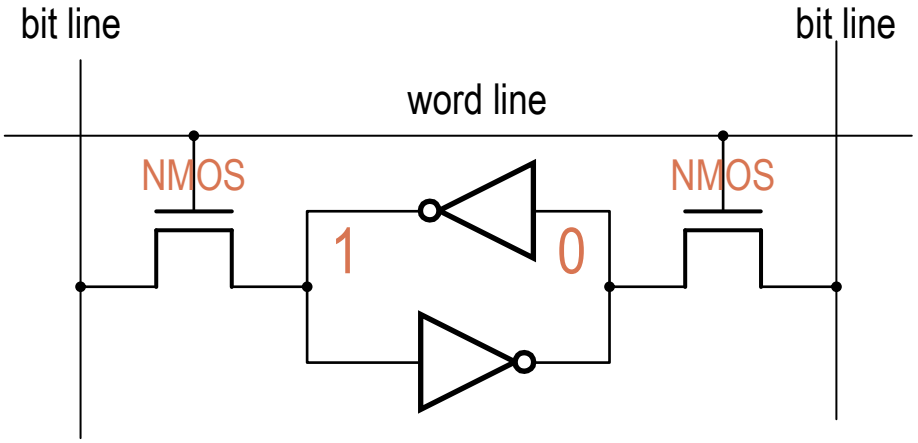
- セルにインバータ（NOT ゲート）のループを使った方式
- Static とはどう言う意味か？
  - 動作が静的（何もしなければ電流をほとんど消費しない）
  - 電源を入れている限りは内容が消えない
- 一般的にはメモリの中で最も高速
  - CPU 内の論理回路と同じトランジスタを使って作るから
  - レジスタなどは SRAM で作られている

SRAM のセル（1bit）



- インバータのループにより 1 bit を記録
  - ループの左右が [1:0] あるいは [0:1] の 2 つの定常状態を持つ
- D-FF とかと同じ原理
  - ただし，クロックに同期させるための仕組みがない分単純

SRAM の読み出し

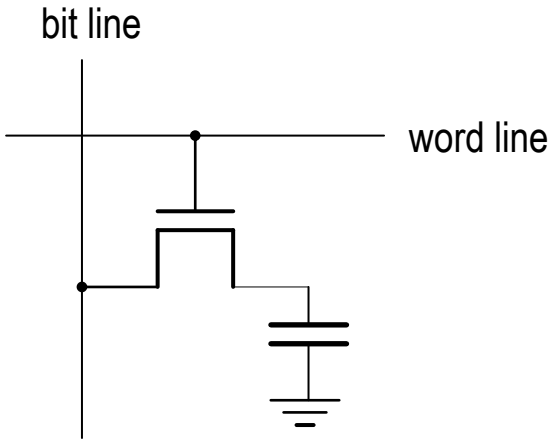


1. 読み出す行のワードラインが高電位（1）に
2. その行の NMOS が ON になりビットラインとインバータが接続
3. ビットラインを通じてセルの内容が伝えられる

Dynamic Random Access Memory (DRAM)

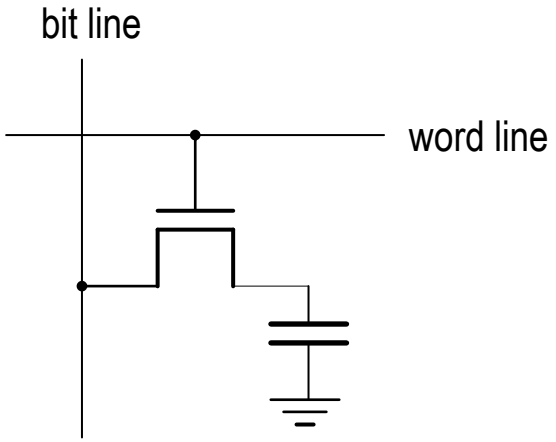
- セルにコンデンサを使った方式
- Dynamic とはどう言う意味か？
  - 動作が動的：電荷がもれて記憶が消えていく
- 速度は遅いが，セルを小さく作れるので容量が稼げる
  - メイン・メモリは DRAM で出来ているのが普通

DRAM のセル



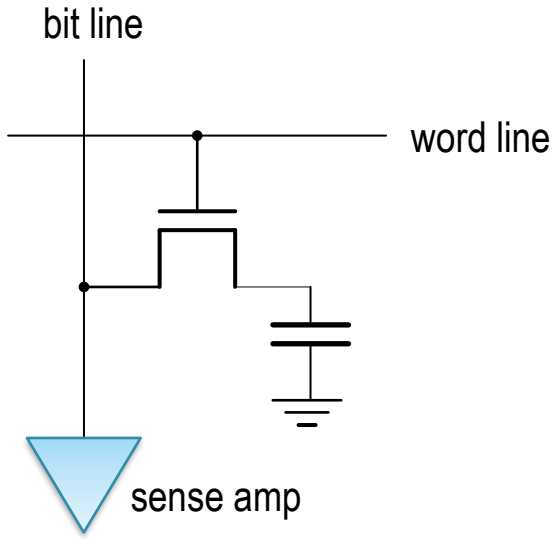
- セルをコンデンサによって構成
  - 電荷がたまっていれば１，そうでなければ０
  - 時間が経つと自然に電荷が抜けてデータが消える
- 電荷を維持するため，定期的にリフレッシュと呼ばれる操作を行う
  - 具体的には，一回読んで同じデータを書き戻す

DRAM の読み出し



1. ビットラインに電荷をプリチャージ
   - ビットライン全体を一種のコンデンサだと思う
   - 高電位にして電荷を信号線にためる
2. 読み出す行のワードラインを高電位に
   - NMOS が ON になりビットラインと接続
3. セルの状態に応じてディスチャージが起きる
   - セルに電荷がある（１）→ ビットラインの状態は変わらない
   - セルに電荷がない（０）→ ビットラインの電位が下がる

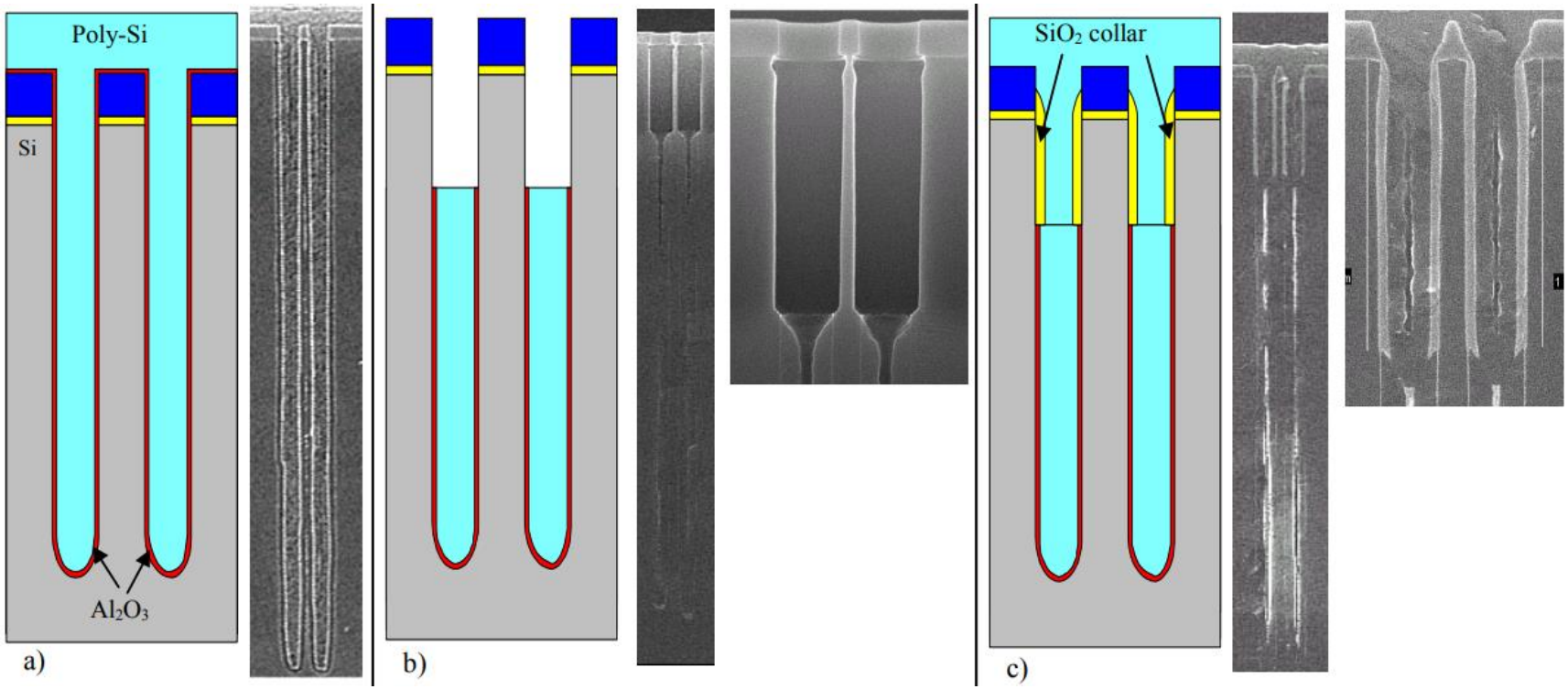
SRAM の場合との違い



- 読み出し時に内容が破壊される
  - ビットラインとコンデンサが接続されると，電荷が流れ出す
- コンデンサがビットラインをディスチャージする速度は遅い
  - センスアンプと呼ばれる微小な電圧の変化を増幅する回路を使う

DRAM セルのコンデンサの作り方

H. Seidl et al, A Fully Integrated Al2O3 Trench Capacitor DRAM for Sub-100nm Technology より



- チップの表から裏に向けて深い穴を掘って表面積を稼ぐ
  - コンデンサの容量は表面積に比例
  - トレンチ（塹壕の意味）と呼ばれる
  - 特殊な工程が必要なので通常のトランジスタと混ぜて作るのが難しい
    - DRAM は CPU とは違う専用のチップで製造される

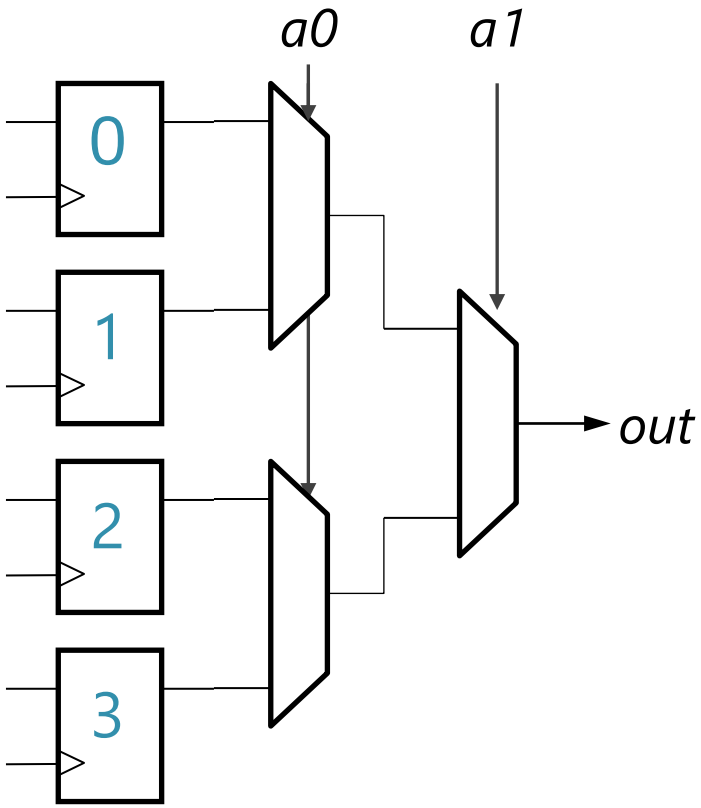
メモリ

1. メモリの基本
    1. 構造
    2. 動作
    3. アクセス時間
2. メモリの詳細
    1. SRAM と DRAM
    2. **なぜメモリが存在するのか？**

メモリの存在する理由

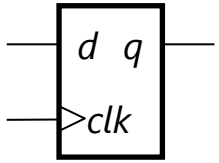
- D-FF とマルチプレクサがあれば，等価な機能は実現できる
  - しかし実際には，メモリ専用の回路が用意される
- メモリ専用の回路が用意される理由：
  - D-FF とマルチプレクサよりも，高密度に実装したいから
  - 同じチップ面積で，より多くの情報を記憶したい

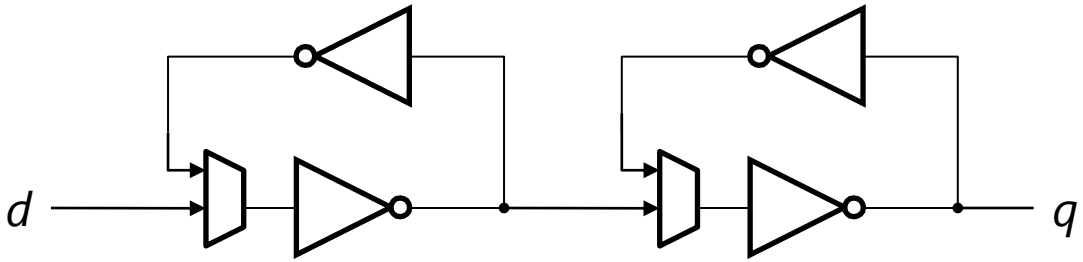
D-FF とマルチプレクサによる 4bit の RAM

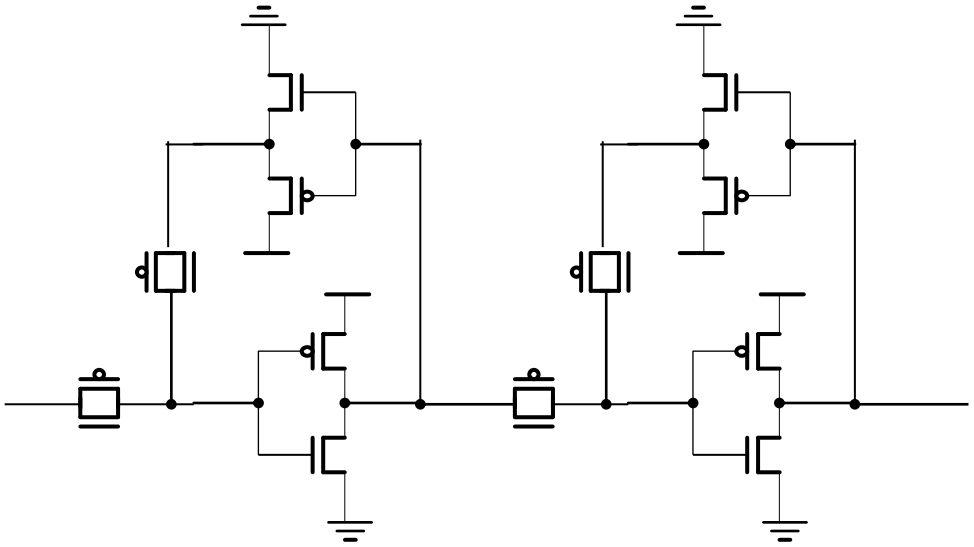


- ４エントリの LUT を D-FF で構成してみる
  - 中身を憶える 4つの D-FF
  - 場所を指定して選択する2段のマルチプレクサ

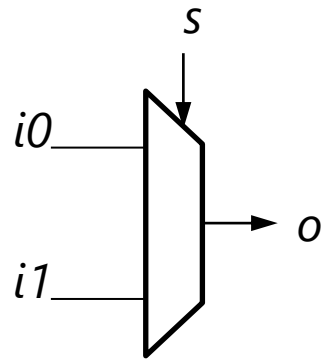
D-FF：トランジスタ 16個

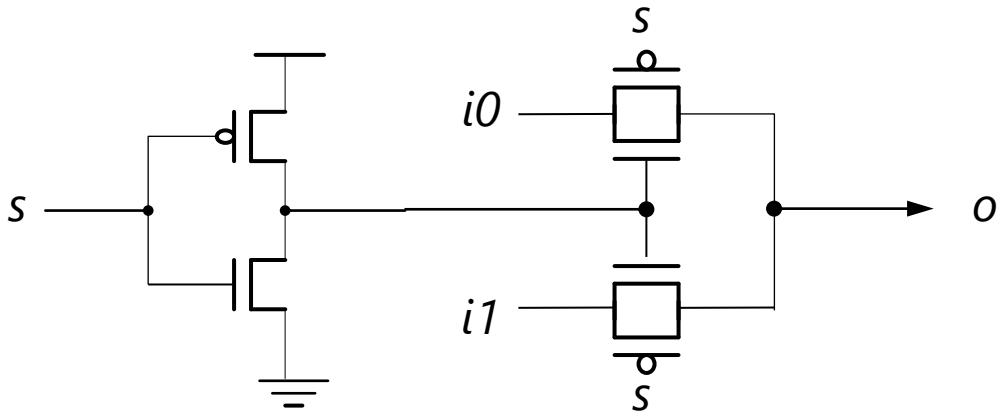






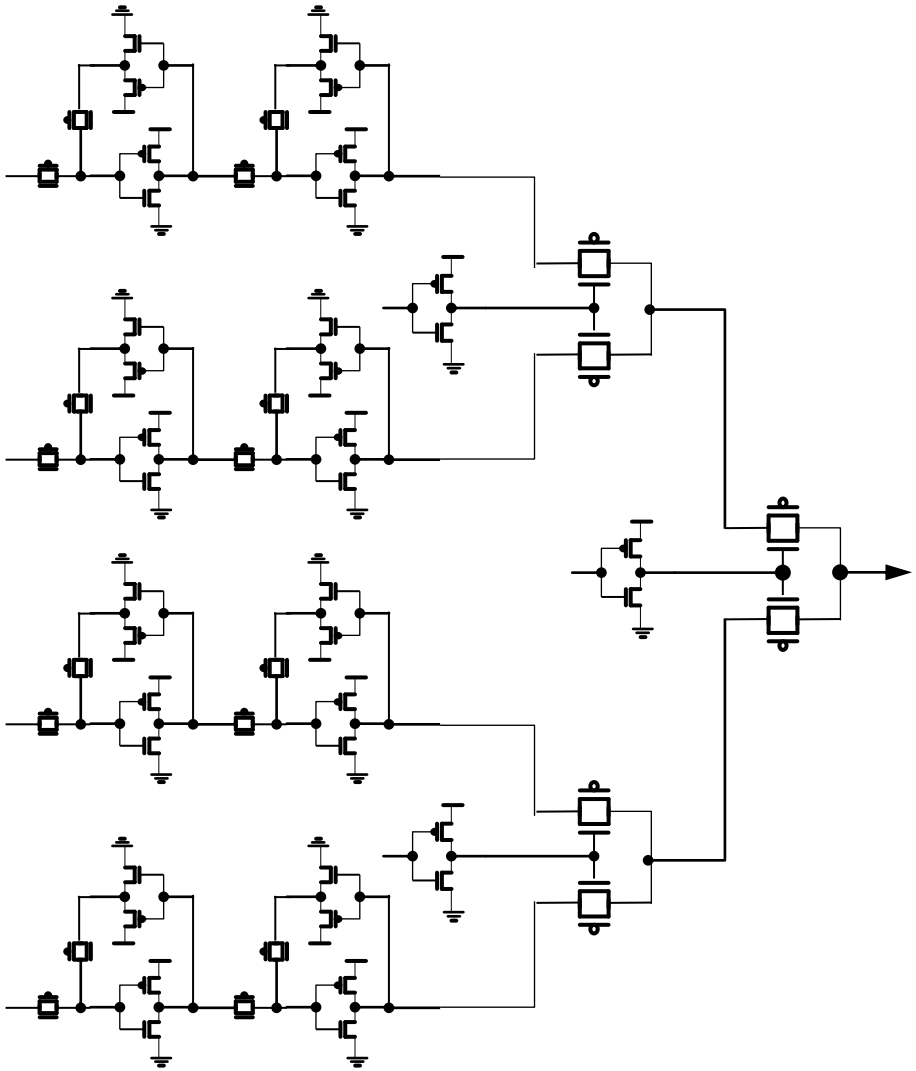
マルチプレクサ：トランジスタ 6個





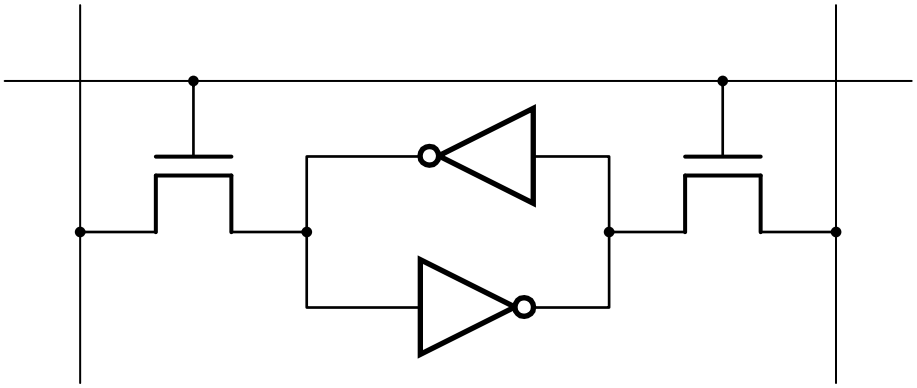
4bit メモリに必要なトランジスタ数：82

16*4+6*3=82

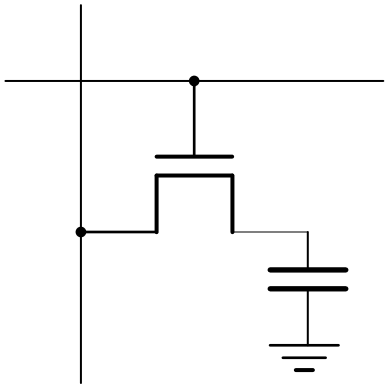


SRAM と DRAM のセル（1bit）

SRAM：2 + 2*2 = 6



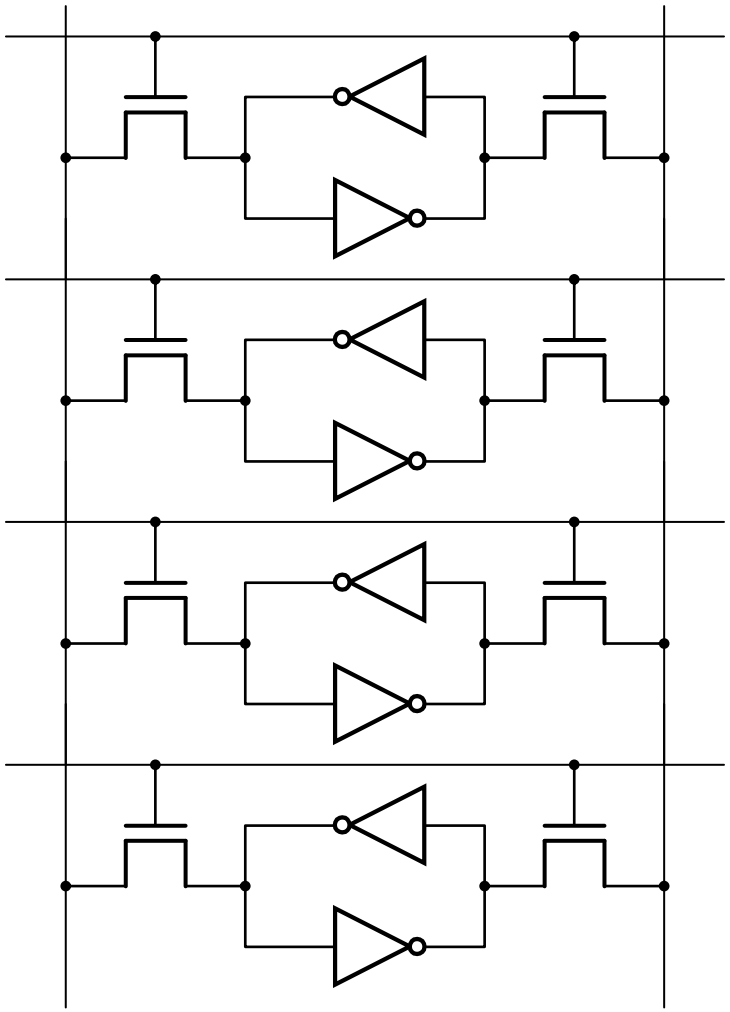
DRAM：1 +コンデンサ 1



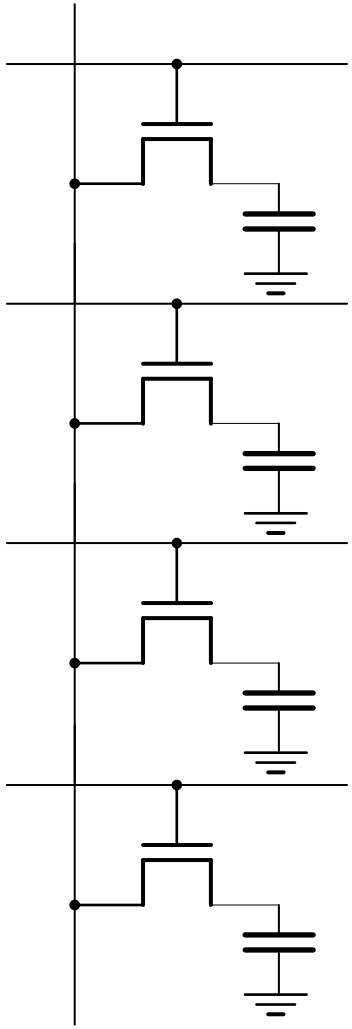
- D-FF よりも圧倒的に単純

SRAM と DRAM（4bit）

SRAM：6*4 = 24



DRAM：4 +コンデンサ 4



メモリのまとめ
（おおよそこのぐらいの事がフワッと分かってれば良い

- メモリ（RAM: Random Access Memory）
  - 複数のデータを記憶し，配列のように位置を指定して読み書きする
  - なるべく少ない回路で なるべく多くの記憶を行うために存在
- 構造とアクセス時間：
  - 1bit の記憶を行うセルを格子状にならべた構造を持つ
  - 読み書きにかかる時間：
    - 記憶容量の平方根程度に比例
    - 信号線の長さに比例するため，速度と容量にはトレードオフがある

メモリのまとめ
（おおよそこのぐらいの事がフワッと分かってれば良い

- SRAM：
  - セルにインバータのループを使ったもの
  - 高速だが低密度
  - 電源を入れている限りは記憶が消えない
- DRAM
  - セルにコンデンサを使ったもの
  - 低速だが高密度
  - 電源を入れていても記憶が消えていく
    - リフレッシュが必要

キャッシュ

キャッシュとは？

- 「キャッシュ」って，ブラウザのあれ？
- ちょっと違うけど，原理は同じもの

原理は同じ

- ブラウザのキャッシュ
  - WEB サーバーからページを取ってくるのは遅い
  - 1回見たページを PC やスマホの「キャッシュ」に置いておく
  - 2回目からは表示が速い
- メモリのキャッシュ
  - メイン・メモリからデータを取ってくるのは遅い
  - 1回読んだデータを CPU の「キャッシュ」に置いておく
  - 2回目からは読み込みが速い

性能へ大きく影響するし，影響範囲も広い

- キャッシュが問題になることは非常に多い
  - ほとんどのプログラムで性能に大きな影響を与えている

例：行列積の実装と性能

- 三重ループとして実装できる
  - ループの順番は任意に入れ替え可能

```
for (int k = 0; k < SIZE; k++)
    for (int j = 0; j < SIZE; j++)
        for (int i = 0; i < SIZE; i++)
            a[k][j] += b[k][i] * c[i][j];
```

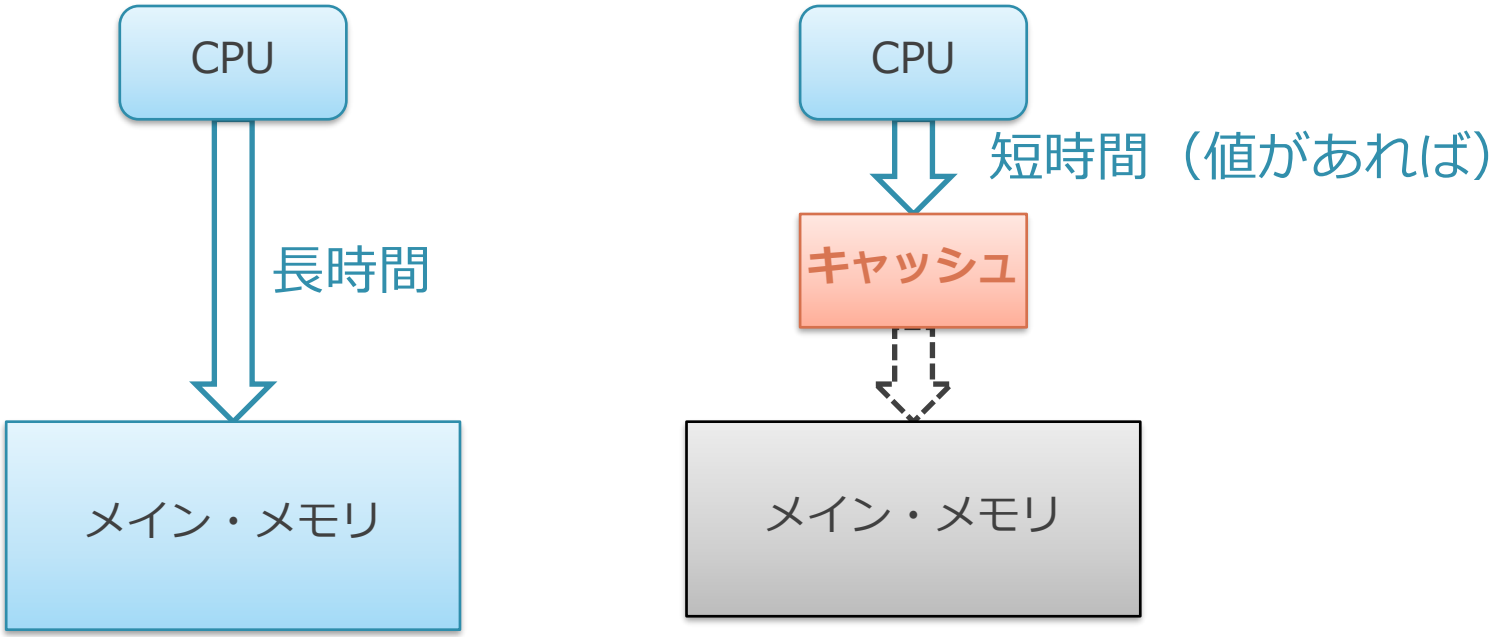
- 単にループの順番を入れ替えるだけで処理時間が大きく変化する
  - 外側から k j i の順 → 178秒
  - 外側から k i j の順 → 20秒
  - 外側から j i k の順 → 1100秒
- この変化は，キャッシュを有効に働かせているかどうかに由来
  - 計算量や全体としてのメモリ使用量は全く同じなのに

内容

1. **キャッシュの基本的な考え方**
    1. 基本的な原理と構造
    2. 容量の性能への影響

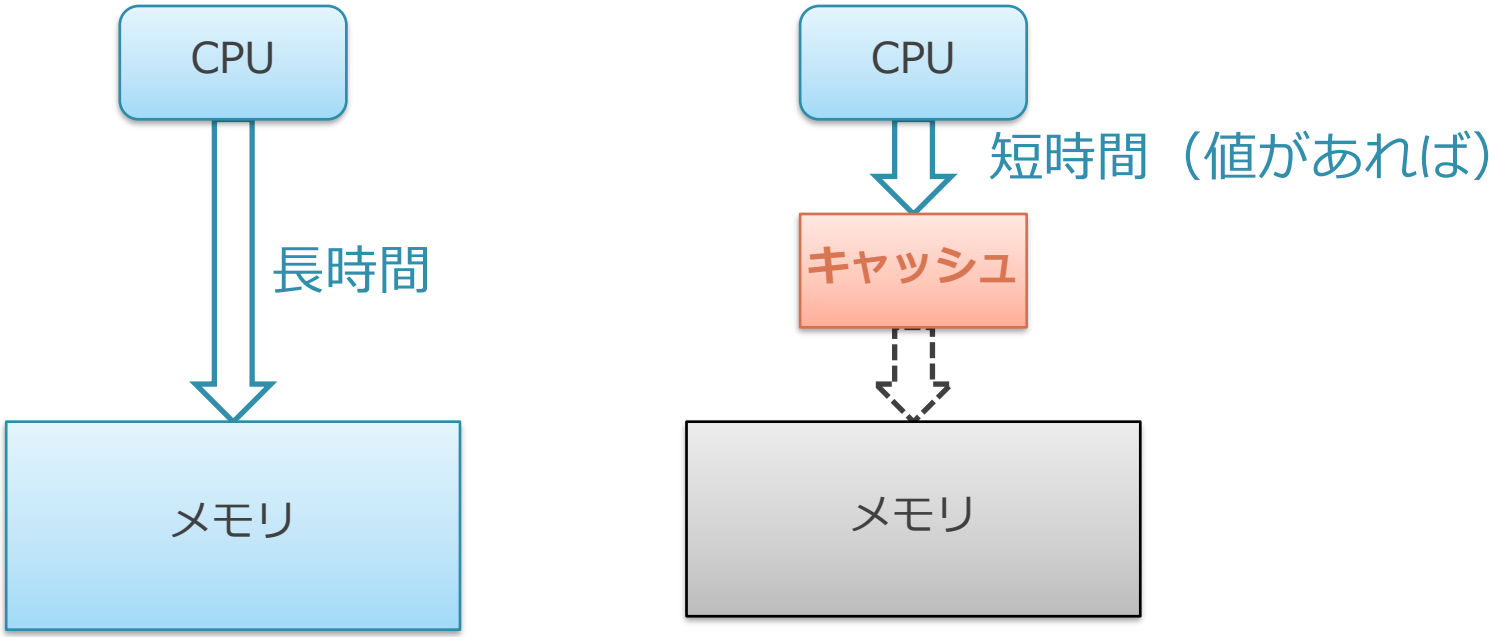
記憶階層

- 以下を階層的に組み合わせる
  - 小さいけど速いメモリ：キャッシュ
  - 大きいけど遅いメモリ：メイン・メモリ
- 背景：大きくて速いメモリは作れない



キャッシュの動作

- ソフトの性質：
  - 一度使用した値は，すぐにまた使う可能性が高い
  - ブラウザのキャッシュと同じ
- **一度利用した値を入れておくことで，2度目からは高速に**

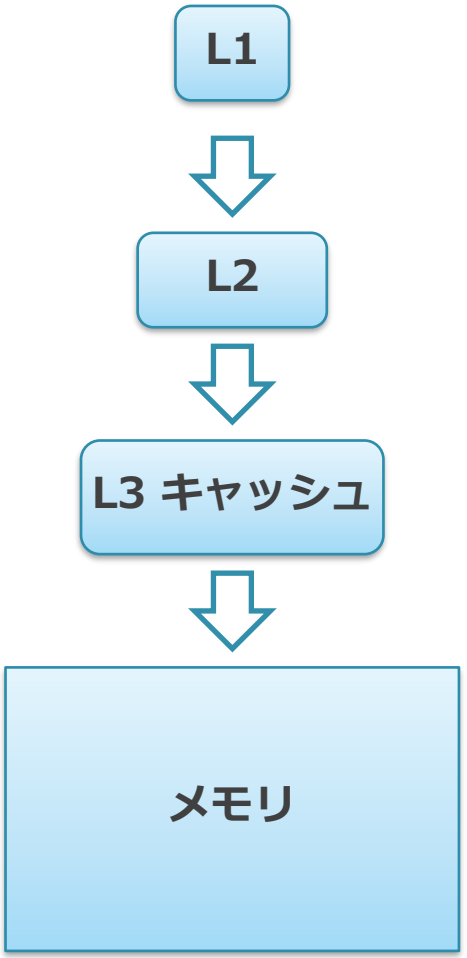


時間的局所性

- 「一度使用した値は，すぐにまた使う可能性が高い」という性質
- なぜか？
  - 人間は，「普通は」関連のある処理同士を近くに書く
  - プログラムの各行ごとに全く関係無い処理を混ぜたりは「普通は」しない
- 関連する処理は，当然同じデータを使う可能性が高い
  - 同じ入力データの使いまわし
  - 前の行の結果を次の行の入力で使う
    - for ループなどはこれらの典型

実際には多層の構造になっている

- 中間の速さと大きさのキャッシュが
  複数組み合わされている
  - PC に使われている CPU だと L1 ～ L3 まで
    あるのが一般的
  - L は Level の略で,
    大きいほど（高次）CPU から遠い



キャッシュの基本的な考え方のまとめ

- 記憶階層
  - 小さいけど速いメモリ：キャッシュ
  - 大きいけど遅いメモリ：メイン・メモリ
- 時間的局所性
  - 一度使用した値は，すぐにまた使う可能性が高い性質のこと
- 実際には多層の構造になっている
  - 中間の速さと大きさのキャッシュが複数組み合わされている

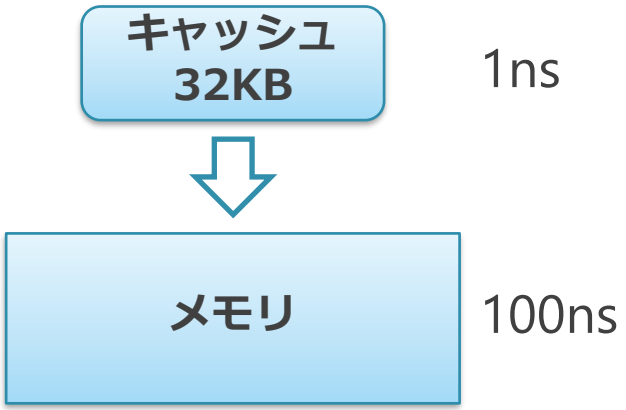
内容

1. キャッシュの基本的な考え方
    1. 基本的な原理と構造
    2. **容量の性能への影響**

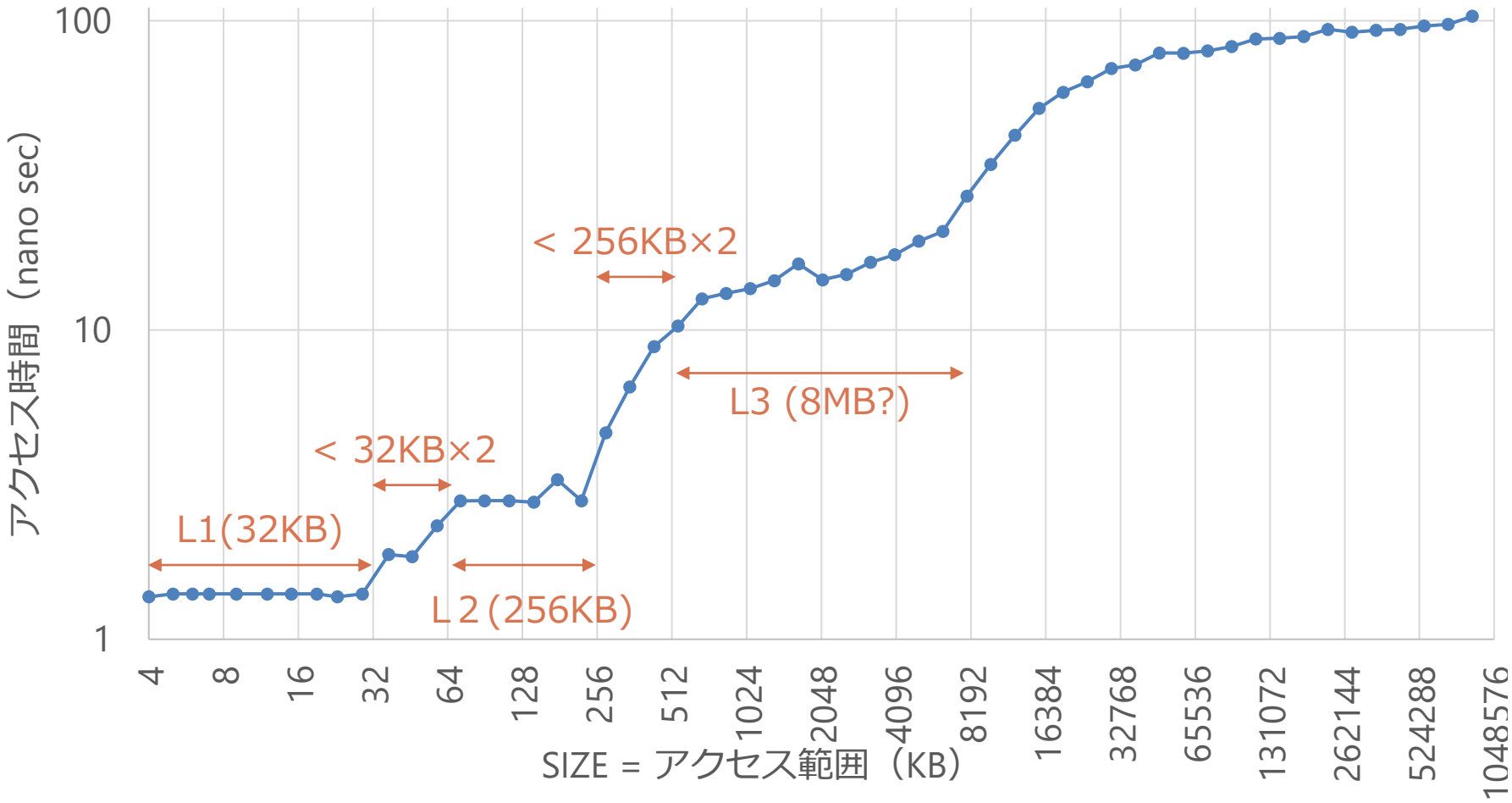
キャッシュへの性能への影響

- 下記のような二重ループを考える
  - SIZE がキャッシュの容量に収まっていれば,
    内側ループ終了後に table の全データがキャッシュに乗る
  - 次の内側の周回では全データがキャッシュに乗っているので速い！

```
for (int i = 0; i < NUM_TEST; i++) {
   uint32_t p = 0;
   for (int j = 0; j < SIZE; j++) {
      p += table[j];
   }
}
```

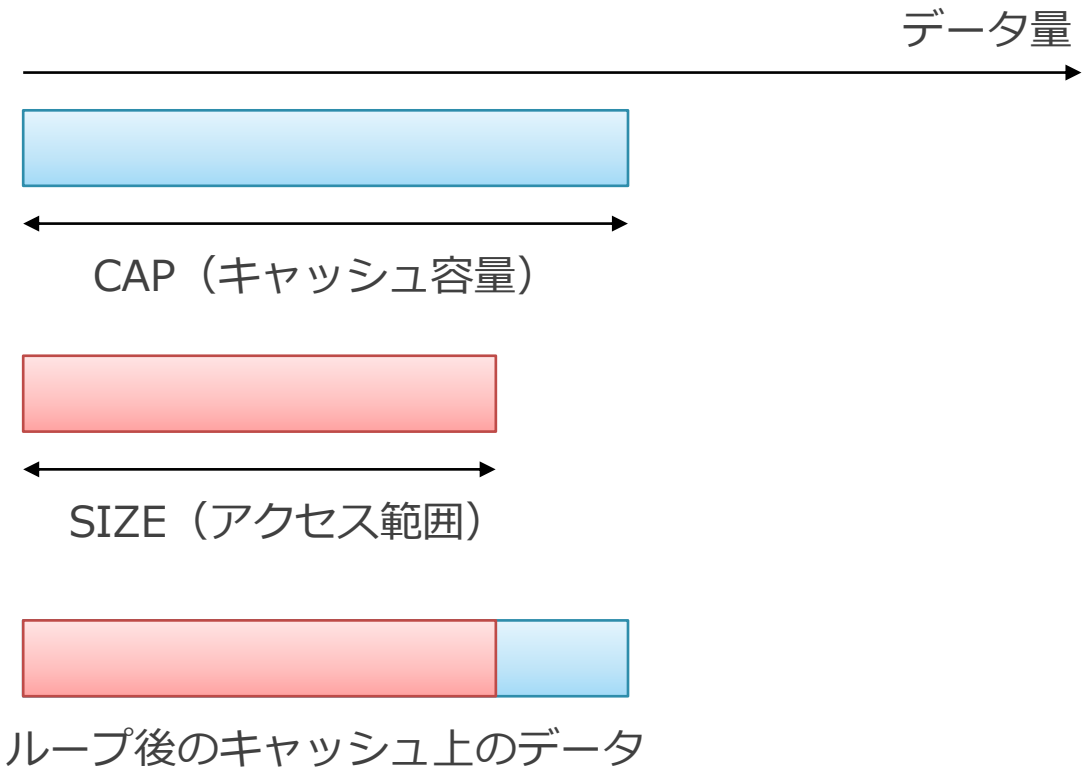


実際の測定データ



- 縦軸を基数10の対数軸，横軸を基数２の対数軸に
- 低次キャッシュの内容は必ず高次に含まれる仕様
  - （そうなるかはメーカーや世代に依存．含まれないこともある
  - 256KB+32KB ではなく 256KB で変化しはじめる

プログラム最適化の上で，重要なポイント



- 上記の状態を保つこと
  - ワーキング・セット：
    - 処理のまとまりごとに，アクセスするデータの範囲（使用量）
  - ワーキング・セットがキャッシュ容量に収まることが重要

まとめ

- メモリ
  - 構造，動作，容量と速度
  - SRAM と DRAM
  - メモリの存在理由
- キャッシュ
  - 基本的な原理と構造
  - 容量の性能への影響

課題 9

- 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 スーパスカラプロセッサであり，理想的には IPC=2 で実行できる
  - ロード命令の出現率は 0.2
  - CPU は L1 キャッシュをもつ
  - ロード命令のみが L1 キャッシュやメモリにアクセスするものとする
  - L1 キャッシュのヒット時には一切のペナルティなしで実行できる

- (1) 以下の場合の IPC を求めよ
  - ロード命令による L1 キャッシュのアクセス 1 回あたりのミス率が 0.01
  - ミスの発生時は 100 サイクル追加で時間がかかる

課題 9

- (2) 以下の場合の IPC を求めよ
  1. L1 キャッシュの容量を倍にしたもの
     - L1 キャッシュに良く当たるようになったため，ミス率が 0.006 に
  2. L2 キャッシュを追加したもの
     - L2 へのアクセス 1 回あたりのミス率は 0.1
       - L2 は L1 にミスしたときのみアクセスするものとする
     - L2 ヒット時は L1 ヒット時からの実行時間の追加が 10 サイクル
     - L2 ミス時は L1 ヒット時からの実行時間の追加が 100 サイクル

- (3) これまでに出てきた 3 つのモデルの性能を求め比較せよ
  - ただし L1 キャッシュ容量を倍にした場合，キャッシュのアクセスに時間がかかるため，周波数が 0.8 倍になるものとする

期末試験について

- いまのところ 8/7 の予定
- A4 裏表 1 枚 手書きのみの持ち込み可の予定
  - ・・・にしようと思いますが，意見がほしいです

提出方法

- 以下を提出：
  1. 課題９：
     - 提出は Moodle の「課題９」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（7/2 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

- 内容は非常に興味深く面白いのですが、一回の量が多く感じます。
  復習もあるので大変ですが頑張ります。

質問とか感想

- おそらくどんどん分岐予測の精度があってきていると思うので一般的に分岐予測ミスは何%くらいまで許容されるのか気になりました。
  - 最新のものだと 1000 命令に 4 回のミスぐらいです

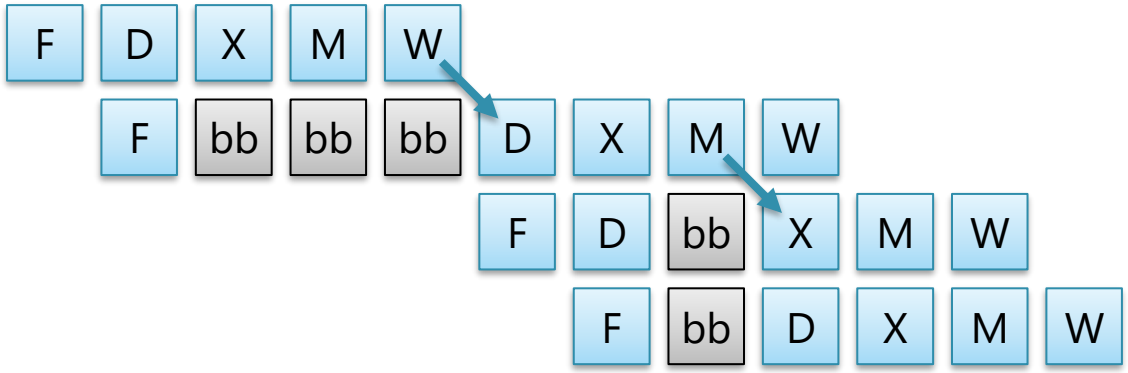
質問とか感想

- 課題7(5)のフォワーディングなしの場合の別解について、ld x2←(x1)のMからadd x5←x2+x7のXに矢印が出ていますが、add x5←x2+x7のXはld x2←(x1)のWまで終わらないとx2が使えないように感じます。フォワーディングしていなくても前の命令のMまで終わっていればその値を使えるのでしょうか。

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)

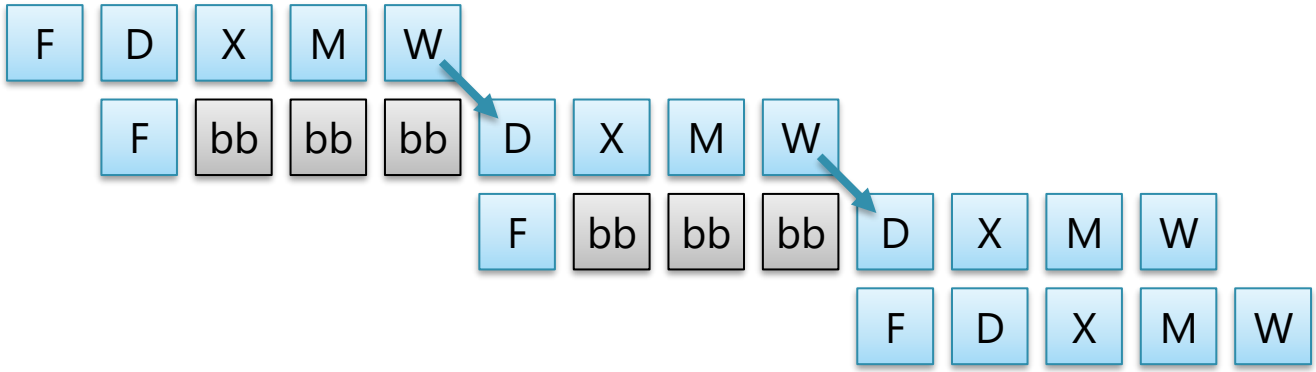
- 12ns （フォワーディングなしの場合の別解）← 間違い



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)

- 14ns （フォワーディングなしの場合の別解）



質問とか感想

- 課題7(7)の解答の図で、それぞれの段(FDXMW)がどの命令を表しているのかがよく分からなかったのですが、
- li x1←1
- li x2←2
- beq x1==x2, LABEL
- add x2←x3+x4
- add x1←x2+x3
- で合っているでしょうか。また、分岐予測により飛んだ先で実行される命令は、分岐命令のFが終わってすぐに始まるのだと思っていたのですが、もし上図の通りだとすると、分岐予測をしても、LABELに飛ぶと実行される命令add x2←x3+x4は、beq x1==x2のWが終わってから始まるのでしょうか。

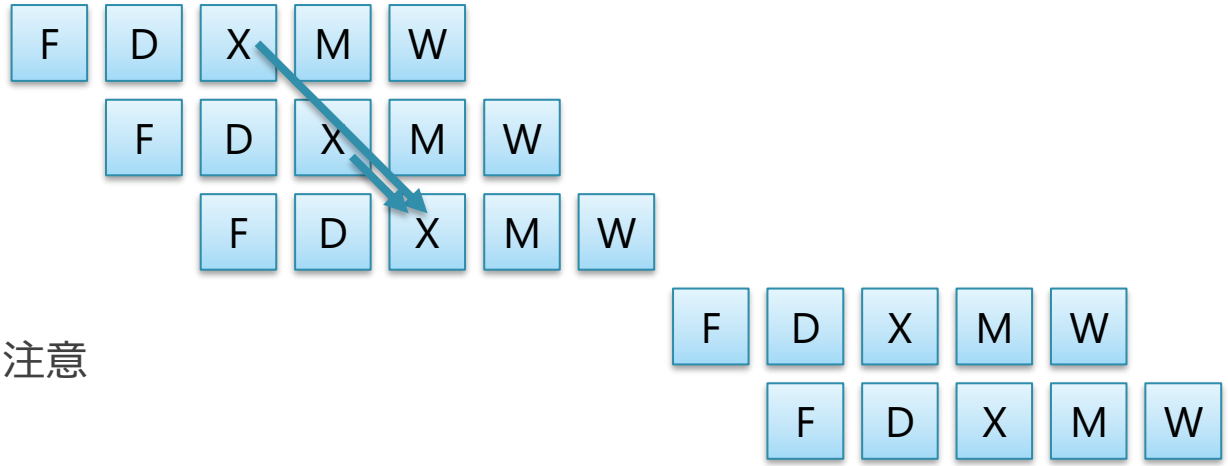
課題 7

- (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次の命令にを実行した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←2
  beq x1==x2, LABEL
  add x1←x2+x3
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



質問とか感想

- スカラ・パイプライン・プロセッサとシングル・サイクル・プロセッサのサイクルの長さが異なるということに混乱してきてしまったので、もう一度説明していただきたいです。スカラ・パイプライン・プロセッサも、命令の１サイクルは"Ｆ Ｄ Ｘ Ｍ Ｗ"の5ns なのではないのかと考えてしまいます。サイクルの定義がよく分かっていないのかもしれません。
パイプライン化した場合、最大の"Ｆ Ｄ Ｘ Ｍ Ｗ"全ての命令が並ぶため、1nsでいいということなのでしょうか。パイプライン段数を倍にした場合も同様の考え方でしょうか。

質問とか感想

- データハザードと分岐予測ミスによる実行サイクルの増加を互換性のないものとして考えてもいいのか悩みました。

質問とか感想

- 2-way スーパスカラとパイプライン段数を2倍にしたものとの違いがわからない。
- 段数が２倍のスカラパイププロセッサと2wayスーパスカラプロセッサの違いがあまり理解できていないです。
  - 段数が２倍=ベルトコンベアを２倍分早く送れる
  - 2way =ベルトコンベアが２本ある

質問とか感想

- 分岐予測器の改良をして予測自体の計算時間が増えても、全体の命令数が多い場合の想定であれば気にしなくてもいいんですか？

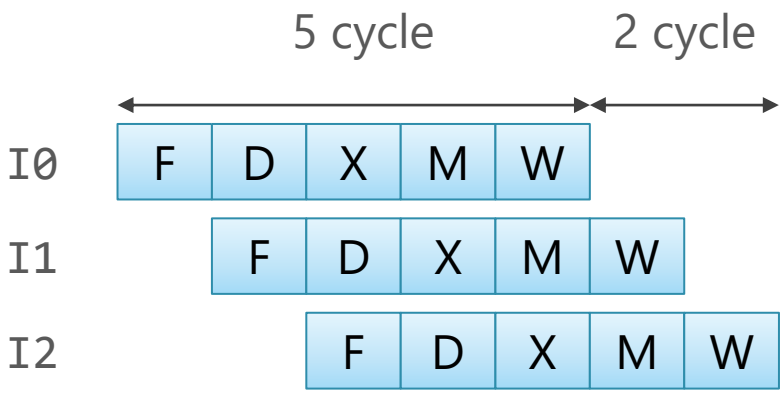
質問とか感想

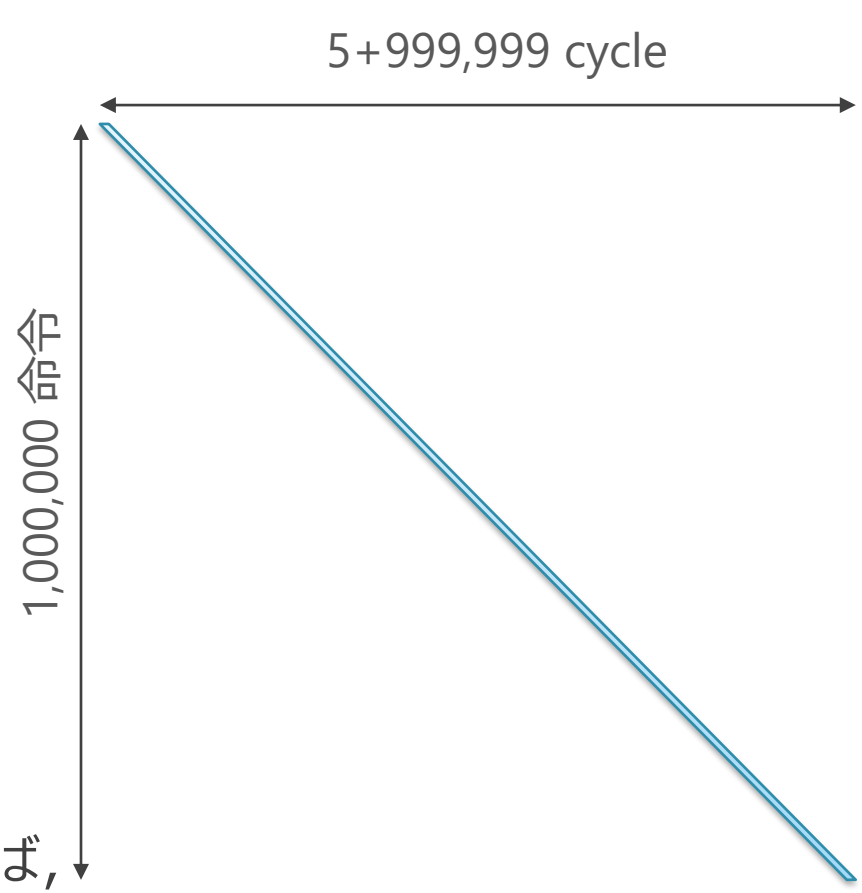
- 課題で、実行命令数Niをどのように求めるのか分かりませんでした。

質問とか感想

- 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、というところが理解できない。グラフの数値がなぜ？となっています。5+999999サイクルはどのようにしてこうなったのか。

「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，ほぼ IPC = 1,000,000/(5+999,999) は，ほぼ 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

質問とか感想

- IPCの計算が少し難しいと思いました。パイプラインを形成してもシングルサイクルプロセッサの時とIPCが変わらないというのが感覚的に少し納得しずらかったです。

- スーパースカラ・プロセッサやスカラ・パイプライン・プロセッサにしても周波数が落ちたり、IPCが下がったりしてしまうのならば結局どのプロセッサを使用してもあまり変わらないのかなと思いました。どのような時にどのプロセッサを使用するべきなのかがあれば知りたいと思いました。

質問とか感想

- フォワーディングによって普通の演算なら前の命令のXが終わったらで次の命令のXは前の命令のWまで待たずに始められますが、分岐予測の場合、ミスを起こしていたことが判明するのはXが終了した時ではなくWまで待って初めて判明するものなのでしょうか？

  - 原理的には X 終了時にできます
  - パイプラインでやりかけた事を取り消して，やり直しの準備にも時間がかかるため，便宜上 W でやり直すことにしています
    - （試験等に出すときは明記するか，どっちでも正解にします

質問とか感想

- 今回は、パソコンの性能を上げるのにも一筋縄では行かないこと、またそのやり方にも限界があることを知れたのが興味深かったです、量子コンピュータができたらこの辺りの技術的障壁は一気になくなるのではないかと期待します。

質問とか感想

- なんらかのデータハザードの具体的な例にロードによるデータハザードが含まれるという解釈で正しいですか？

質問とか感想

- やっと計算って感じがして少し楽しいです。絵がわかりやすいので増やしてほしいです

質問とか感想

- 前回の授業の感想でたくさんの人が例題などを作ってほしいという意見が出た中、例題を作らないことについての理由などは理解できました。しかし、こちらとしては、テストに向けた勉強として課題や講義資料などから得られる理解に限界があるように感じます。先生のおっしゃっていた通り、実際に自分で考えて演習しないとなかなか理解ができない部分もあるので、１回前の授業分でも構わないので練習として解ける問題などを毎授業でほしいと感じました。お忙しい中で講義資料を作られているかと思いますが、学生としてテストには万全の準備をしたい気持ちがありますのでどうかご検討をよろしくお願いします。

  - 即日は難しいので，後でもう少し追加したいと思います
  - （たくさんというよりは，２～３人だったとは思います

質問とか感想

- 内容には関係ないのですが、教室が暑すぎます。。。

質問とか感想

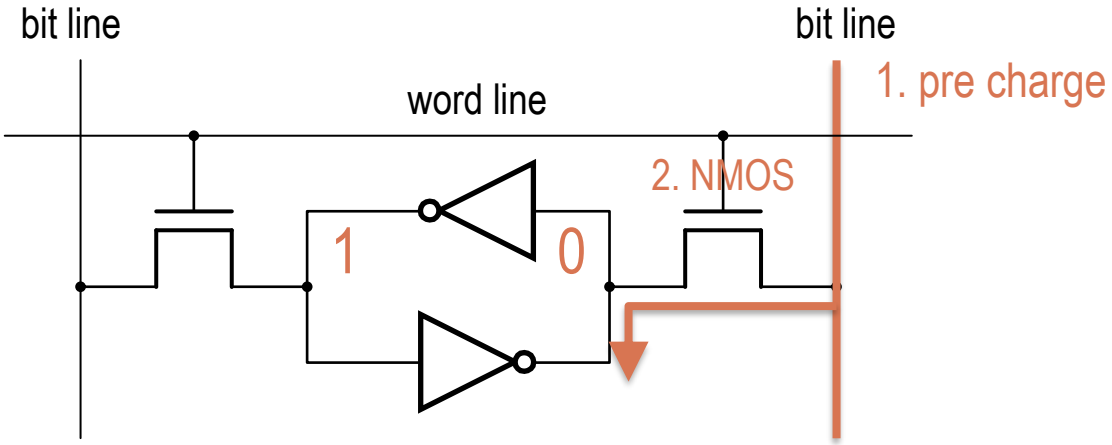
- お子さんの最新ショットを見せてください！

質問とか感想ではないですが

- 自転車で来ているのですが，帰りがけに嫁さんへのお土産にケーキやおかし等を買っています
  - なかなか家から出れないので
- この辺のオススメのお店とかがあったら，よければ教えてください

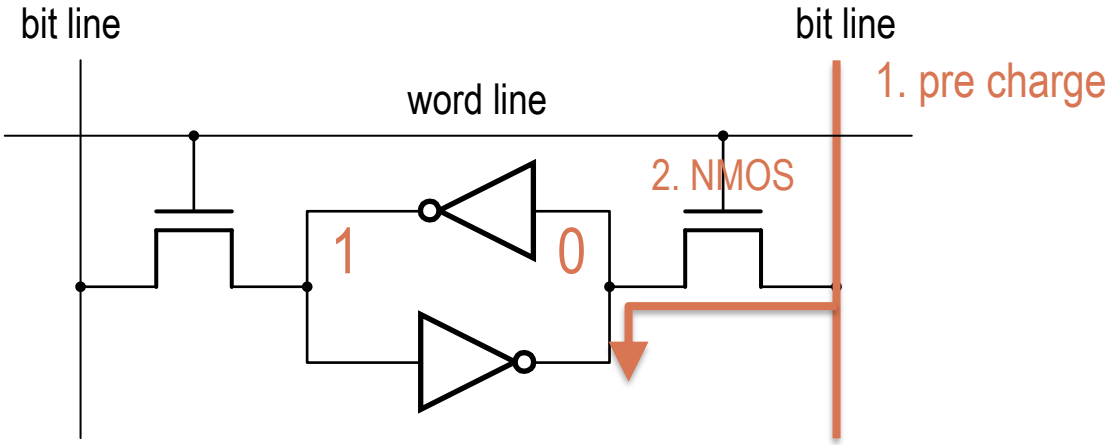
付録

SRAM の読み出し



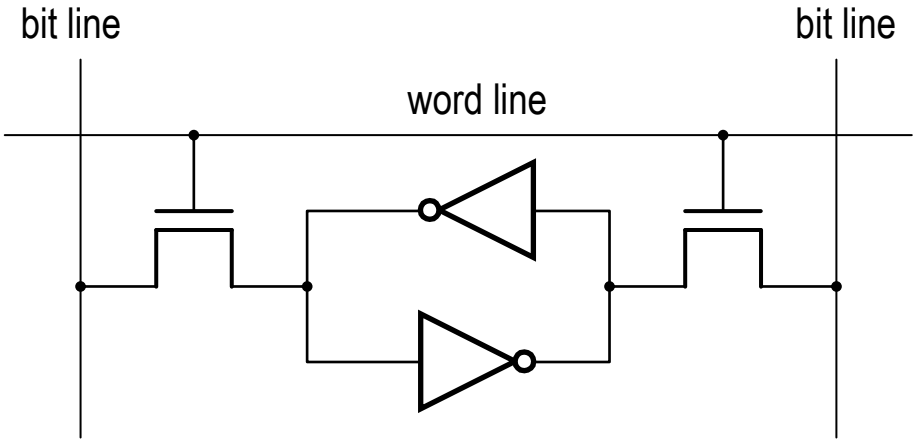
1. ビットラインのプリチャージ
   - ビットラインをあらかじめ高電位（１）にチャージする
2. ワードラインをアサート
   - ループの右にある NMOS が ON に
   - ループとビットラインが接続される
3. ビットラインのディスチャージ
   - ループの右が 0 なら，ビットラインが 0 に
   - ループの右が 1 ならそのまま

なぜプリチャージが必要なのか？



- 単にループとビットラインが接続されるだけではダメ
  - NMOS が高電位をうまく伝えられないから
    - もしループの右側が 1 で NMOS が ON になっても，ビットラインを 1 に引き上げることはできない
  - PMOS を追加すればできるが，なるべく小さく作りたいから嫌だ
- NMOS を介してビットラインの電位を下げることだけができる
  - = あらかじめ電位を上げておいて，下がったかどうかで判定

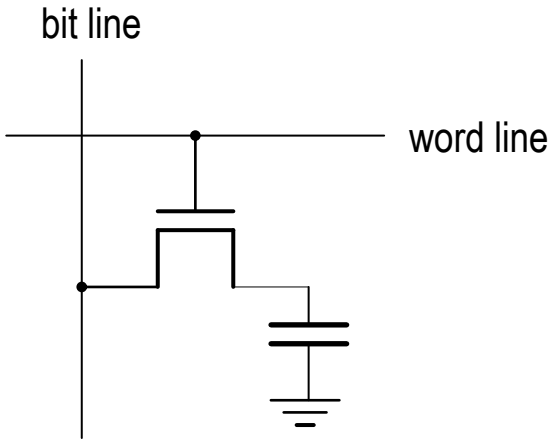
SRAM の書き込み



- 書き込み手順
  1. ループ左右のどちらか 0 にしたい方のビットラインの電位を下げる
  2. ワードラインをアサートして NMOS を ON に
  3. インバータの状態を強制的にビットライン側から低電位にする

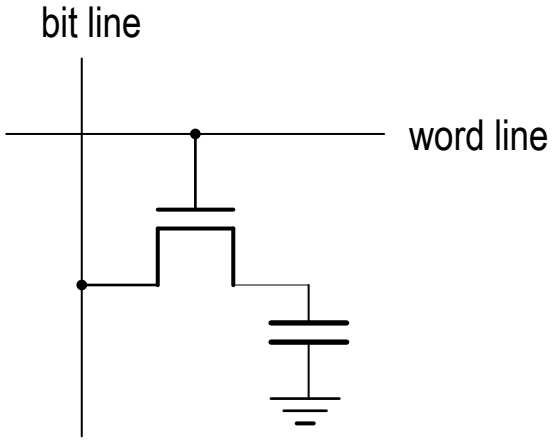
- NMOS が低電位しか通せないので，書き込みには 2 本いる

DRAM の読み出し



- 読み出しは，基本的には SRAM と同じ
  - ビットラインをプリチャージ
  - ワードラインをアサート
  - セルの状態に応じてディスチャージが起きる

DRAM の書き込み



- 書き込みは，NMOS でも 0.5 までは上げられるのでそれで行う（多分）
  - SRAM ではインバータをひっくり返す必要があるので，0.5 ではダメだった
- DRAM では 低電位（0）or 中間電位（0.5）を記録する
  - 読み出し時のプリチャージの電圧を 0.5 にして，下がったかどうかで判定