東邦大学

生命（いのち）の科学で未来をつなぐ

# CISCとRISC

# CISCとは？ RISCとは？

東邦大学

# いろいろなCPUの中でどんな命令があるのか？

# いろいろなCPUの中でどんな命令？

- >前に見たのは COMET II
  - モデル（実際は存在しない）CPUで
  - 命令数が少なく、それぞれ簡単な動作

# いろいろなCPUの中でどんな命令？

>前に見たのは COMET II

>周りのパソコンで使われる
Core 3i/5i/7i など (通称 x86系)
実用規模。非常に複雑な命令体系で
命令数も多く、複雑な動作の命令がある

# いろいろなCPUの中でどんな命令？

>前に見たのは COMET II

>周りのパソコンで使われる
Core 3i/5i/7i など (通称 x86系)

>サーバーで使われるPower CPU

>スマホでよく使われるARM CPU

東邦大学

# いろいろなCPUの中でどんな命令？

## 命令体系の比較

# いろいろなCPUの中でどんな命令？

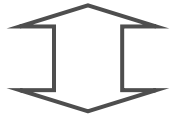
命令体系の比較

⇒　どんな組合せになっているか

# いろいろなCPUの中でどんな命令？

## 命令体系の比較

RISC ⇒ 簡単な命令、簡単な体系
　　複雑なことを命令の組合せで

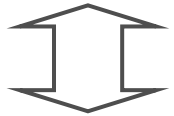


CISC ⇒ 複雑な命令、複雑な体系
　　複雑なことを１つの命令で

東邦大学

# いろいろなCPUの中でどんな命令？

## 命令体系の比較

RISC ⇒ 簡単な命令、簡単な体系
　　複雑なことを命令の組合せで

　　Reduced Instruction Set Computer

CISC ⇒ 複雑な命令、複雑な体系
　　複雑なことを１つの命令で
　　Complex Instruction Set Computer

東邦大学

# 歴史的背景（脱線）

メインフレーム時代

⇒ ハードが機能追加され複雑化（命令も複雑化）

# 歴史的背景（脱線）

メインフレーム時代

⇒ ハードが機能追加され複雑化（命令も複雑化）

⇩

LSI（＝マイクロプロセッサ）時代の幕開け（1972）

⇒ 集積度低く、1チップにたくさんは載らない

⇒ 複雑なCPUを多チップに分割するより
簡単なCPUを１チップに載せた方が早い

⇩

# 歴史的背景（脱線）

メインフレーム時代
　⇒ ハードが機能追加され複雑化（命令も複雑化）

⇩

LSI (＝マイクロプロセッサ）時代の幕開け（1972）
　⇒ 集積度低く、1チップにたくさんは載らない
　⇒ 複雑なCPUを多チップに分割するより
　　簡単なCPUを１チップに載せた方が早い

⇩

簡単なCPU (RISC) にして、クロックを早くする

⇩

# 歴史的背景（脱線）

メインフレーム時代
⇒ ハードが機能追加され複雑化（命令も複雑化）

⇩

LSI（＝マイクロプロセッサ）時代の幕開け（1972）
⇒ 集積度低く、1チップにたくさんは載らない
⇒ 複雑なCPUを多チップに分割するより
簡単なCPUを１チップに載せた方が早い

⇩

簡単なCPU (RISC) にして、クロックを早くする

⇩

今は
細線化により複雑なCPUでも楽に載るようになった

# 技術的背景（脱線）

RISCは簡単な命令を複数個組合せて
　１つのCISC命令に相当することを行う

命令の組み合せ方、レジスタの使い方などで
　コンパイラを工夫する余地が大きかった
　（工夫を競う風土が研究者に合った）

東邦大学

# RISCの特徴

# RISCの特徴

１つ１つの命令の動作は簡単
　１つの命令でいろいろなことをするのではない
　例：　演算はレジスタ・レジスタ間だけ
　　（演算命令でメモリをアクセスしない）

# RISCの特徴

１つ１つの命令の動作は簡単
　１つの命令でいろいろなことをするのではない
　例：　演算はレジスタ・レジスタ間だけ
　　（演算命令でメモリをアクセスしない）

(その代り) レジスタ数を多くする
　なるべくレジスタにデータを置いておく

# RISCの特徴

１つ１つの命令の動作は簡単
　１つの命令でいろいろなことをするのではない
　例：　演算はレジスタ・レジスタ間だけ
　　（演算命令でメモリをアクセスしない）

(その代り) レジスタ数を多くする
　なるべくレジスタにデータを置いておく

命令あたりのクロック数を少なくする

# RISCの特徴

１つ１つの命令の動作は簡単
　１つの命令でいろいろなことをするのではない
　例：　演算はレジスタ・レジスタ間だけ
　　（演算命令でメモリをアクセスしない）

(その代り) レジスタ数を多くする
　なるべくレジスタにデータを置いておく

命令あたりのクロック数を少なくする

命令の長さを固定長にする　　回路簡単化・高速化

# CISC/RISCの例

CISC　代表例はIntelのx86系プロセッサ
　8086に始まりPentiumを経て
　　現在はCore2 7i/5i/3i、サーバー用xeon

# CISC/RISCの例

CISC　代表例はIntelのx86系プロセッサ
8086に始まりPentiumを経て
現在はCore2 7i/5i/3i、サーバー用xeon

昔の代表：IBMのメインフレーム(360, 370)

# CISC/RISCの例

CISC　代表例はIntelのx86系プロセッサ
　8086に始まりPentiumを経て
　　現在はCore2 7i/5i/3i、サーバー用xeon

　昔の代表：IBMのメインフレーム(360, 370)

RISC　代表例はMIPS, PowerPCなど
　現在はあまり残っていない

# CISC/RISCの例

CISC　代表例はIntelのx86系プロセッサ
　8086に始まりPentiumを経て
　　現在はCore2 7i/5i/3i、サーバー用xeon

　昔の代表：IBMのメインフレーム(360, 370)

RISC　代表例はMIPS, PowerPCなど
　現在はあまり残っていない

　ビジネス的に数の論理が強く、Intelとその互換機
　　が圧倒的に多い（必ずしも技術的優位ではない)

# RISCとCISCの比較

| | RISC | CISC |
|---|---|---|
| 回路の複雑さ | | |
| 個々の命令の動作 | | |
| 命令当りクロック | | |
| 命令の長さ | | |
| 命令の種類 | | |
| レジスタ数 | | |
| プログラムの命令数 | | |

# RISCとCISCの比較

| | RISC | CISC |
|---|---|---|
| 回路の複雑さ | 単純 | 複雑 |
| 個々の命令の動作 | | |
| 命令当りクロック | | |
| 命令の長さ | | |
| 命令の種類 | | |
| レジスタ数 | | |
| プログラムの命令数 | | |

# RISCとCISCの比較

| | RISC | CISC |
|---|---|---|
| 回路の複雑さ | 単純 | 複雑 |
| 個々の命令の動作 | 簡単 | いろいろ |
| 命令当りクロック | | |
| 命令の長さ | | |
| 命令の種類 | | |
| レジスタ数 | | |
| プログラムの命令数 | | |

# RISCとCISCの比較

| | RISC | CISC |
|---|---|---|
| 回路の複雑さ | 単純 | 複雑 |
| 個々の命令の動作 | 簡単 | いろいろ |
| 命令当りクロック | 少ない | いろいろ |
| 命令の長さ | | |
| 命令の種類 | | |
| レジスタ数 | | |
| プログラムの命令数 | | |

# RISCとCISCの比較

| | RISC | CISC |
|---|---|---|
| 回路の複雑さ | 単純 | 複雑 |
| 個々の命令の動作 | 簡単 | いろいろ |
| 命令当りクロック | 少ない | いろいろ |
| 命令の長さ | 短い | いろいろ |
| 命令の種類 | | |
| レジスタ数 | | |
| プログラムの命令数 | | |

# RISCとCISCの比較

| | RISC | CISC |
|---|---|---|
| 回路の複雑さ | 単純 | 複雑 |
| 個々の命令の動作 | 簡単 | いろいろ |
| 命令当りクロック | 少ない | いろいろ |
| 命令の長さ | 短い | いろいろ |
| 命令の種類 | 少ない | 多い |
| レジスタ数 | | |
| プログラムの命令数 | | |

# RISCとCISCの比較

| | RISC | CISC |
|---|---|---|
| 回路の複雑さ | 単純 | 複雑 |
| 個々の命令の動作 | 簡単 | いろいろ |
| 命令当りクロック | 少ない | いろいろ |
| 命令の長さ | 短い | いろいろ |
| 命令の種類 | 少ない | 多い |
| レジスタ数 | 多い | 少ない |
| プログラムの命令数 | | |

# RISCとCISCの比較

| | RISC | CISC |
|---|---|---|
| 回路の複雑さ | 単純 | 複雑 |
| 個々の命令の動作 | 簡単 | いろいろ |
| 命令当りクロック | 少ない | いろいろ |
| 命令の長さ | 短い | いろいろ |
| 命令の種類 | 少ない | 多い |
| レジスタ数 | 多い | 少ない |
| プログラムの命令数 | 多くなる | 少ない |

# CISC vs RISC 論争の結果

一方の絶対優位はない
　短い命令を多数使うか、長い命令を少数か
　CISCの方が命令読出し回数は減るが、
　命令種類が多く解読に時間がかかる

# CISC vs RISC 論争の結果

一方の絶対優位はない
　短い命令を多数使うか、長い命令を少数か
　CISCの方が命令読出し回数は減るが、
　命令種類が多く解読に時間がかかる

チップ複雑度の上限制約はどんどん緩和し
　CISCに対する制限ではなくなった

# CISC vs RISC 論争の結果

一方の絶対優位はない
　短い命令を多数使うか、長い命令を少数か
　CISCの方が命令読出し回数は減るが、
　命令種類が多く解読に時間がかかる

チップ複雑度の上限制約はどんどん緩和し
　CISCに対する制限ではなくなった

現在は、CISC命令セットの一部をRISCで
　実現するなどの手法が取られる

東邦大学

# CISC・RISCのまとめ

RISCとは

　　が　　なCPU (’80頃に新提案)

⇒ 簡単化　何が？

⇒　　化　何が？

⇒ 代償

CISCとは

　命令体系が　　な(従来の)CPU

比較して、

# CISC・RISCのまとめ

RISCとは

命令体系が［　　］なCPU (’80頃に新提案)

⇒ 簡単化　何が？［　　］

⇒［　　］化　何が？［　　］

⇒ 代償　［　　］

CISCとは

命令体系が［　　］な(従来の)CPU

比較して、［　　］

# CISC・RISCのまとめ

RISCとは

命令体系が簡単なCPU ('80頃に新提案)

⇒ 簡単化　何が？

⇒ 化　何が？

⇒ 代償

CISCとは

命令体系が な(従来の)CPU

比較して、

# CISC・RISCのまとめ

RISCとは

命令体系が簡単なCPU ('80頃に新提案)

⇒ 簡単化　何が？ 命令の内容、命令の数

⇒ [　　]化　何が？ [　　]

⇒ 代償 [　　]

CISCとは

命令体系が[　　]な(従来の)CPU

比較して、[　　]

# CISC・RISCのまとめ

RISCとは

命令体系が簡単なCPU ('80頃に新提案)

⇒ 簡単化　何が？ 命令の内容、命令の数

⇒ 高速化　何が？

⇒ 代償

CISCとは

命令体系が　な(従来の)CPU

比較して、

# CISC・RISCのまとめ

RISCとは

命令体系が簡単なCPU ('80頃に新提案)

⇒ 簡単化　何が？ 命令の内容、命令の数

⇒ 高速化　何が？ 命令の実行、クロック

⇒ 代償

CISCとは

命令体系が　な(従来の)CPU

比較して、

# CISC・RISCのまとめ

RISCとは

命令体系が簡単なCPU ('80頃に新提案)

⇒ 簡単化　何が？ 命令の内容、命令の数

⇒ 高速化　何が？ 命令の実行、クロック

⇒ 代償　　プログラムの命令数が増える

CISCとは

命令体系が　　な(従来の)CPU

比較して、

# CISC・RISCのまとめ

RISCとは

命令体系が簡単なCPU (’80頃に新提案)

⇒ 簡単化　何が？ 命令の内容、命令の数

⇒ 高速化　何が？ 命令の実行、クロック

⇒ 代償　　プログラムの命令数が増える

CISCとは

命令体系が複雑な(従来の)CPU

比較して、

# CISC・RISCのまとめ

RISCとは

命令体系が簡単なCPU ('80頃に新提案)

⇒ 簡単化　何が？ 命令の内容、命令の数

⇒ 高速化　何が？ 命令の実行、クロック

⇒ 代償　　プログラムの命令数が増える

CISCとは

命令体系が複雑な(従来の)CPU

比較して、一方が優位ということは無い

# CISCとRISCの考え方について分かりましたか？

次へ

(ハーバードアーキテクチャ)

# ハーバードアーキテクチャ

# ハーバードアーキテクチャとは？

CPUの内部構造の作り方
の１つ
「ノイマン型」を少しはみ出す

# 復習：　「ノイマン式」

## 何だったか思い出してみよう

# 復習：　「ノイマン式」

何だったか思い出してみよう

1. プログラム(可変)内蔵方式
2. 逐次処理方式
3. 単一メモリ方式

# 復習：　「ノイマン式」

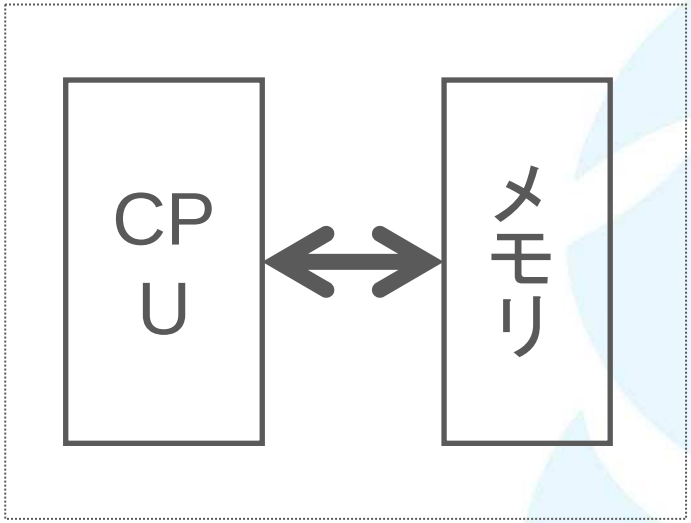
何だったか思い出してみよう

1. プログラム(可変)内蔵方式
2. 逐次処理方式
3. 単一メモリ方式 ⇦ これを はみ出す

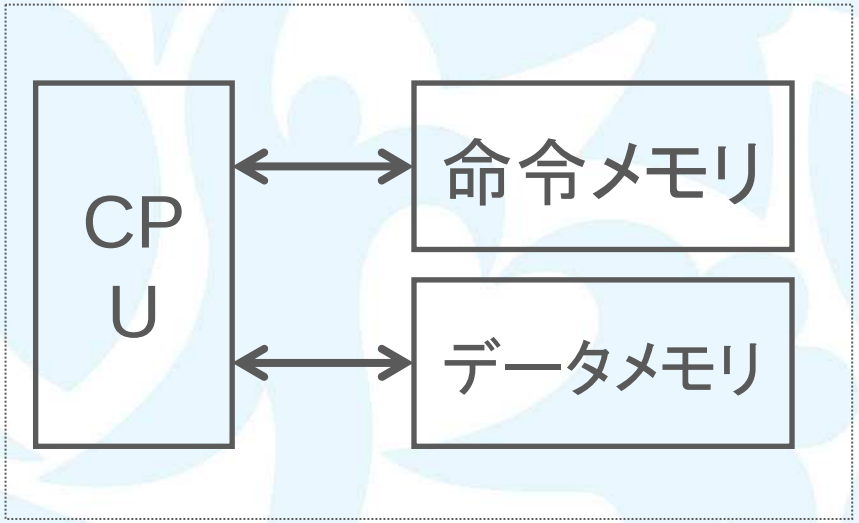
東邦大学

# ハーバードアーキテクチャとは

ノイマン型
アーキテクチャ



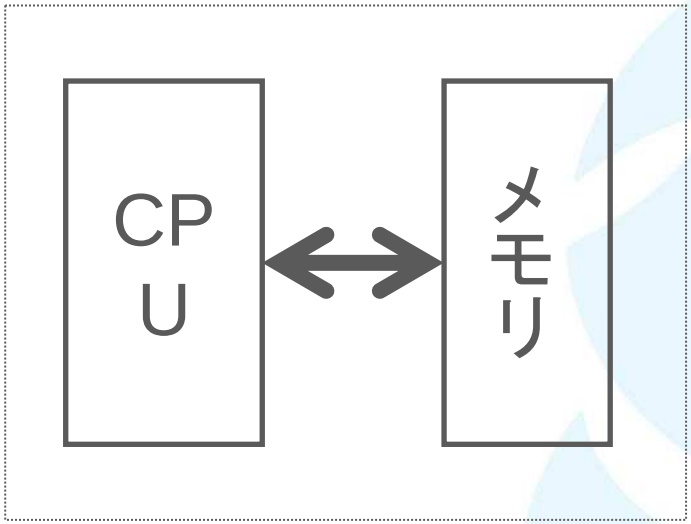

ハーバード
アーキテクチャ




東邦大学

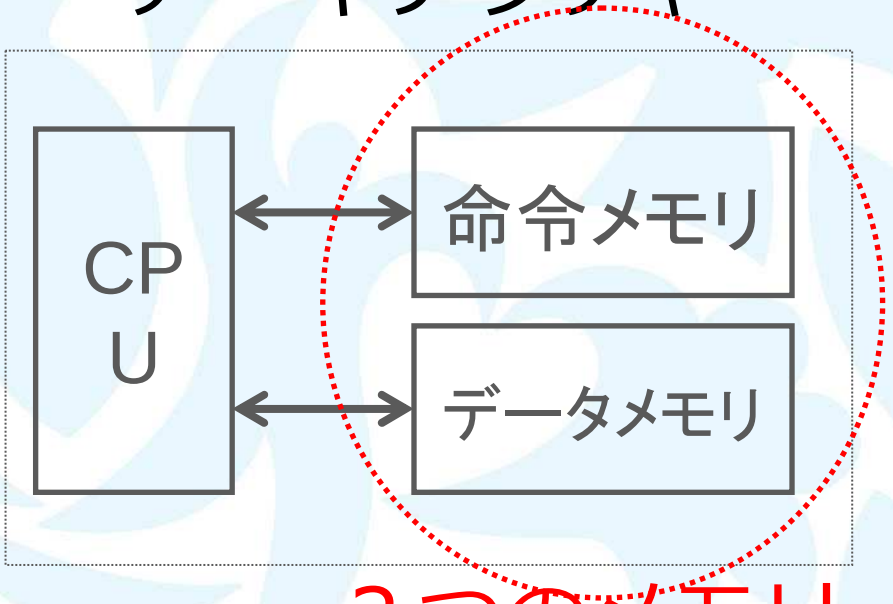
# ハーバードアーキテクチャとは

ノイマン型
アーキテクチャ




単一メモリ

ハーバード
アーキテクチャ




２つのメモリ

東邦大学

# ハーバードアーキテクチャの考え方

ノイマン型では１本だけなので
　アクセスが混雑する　～　「隘路」

# ハーバードアーキテクチャの考え方

ノイマン型では１本だけなので
　アクセスが混雑する　～　「隘路」

⇩

２本別にすればよい？

# ハーバードアーキテクチャの考え方

ノイマン型では１本だけなので
　アクセスが混雑する　～　「隘路」

⇩

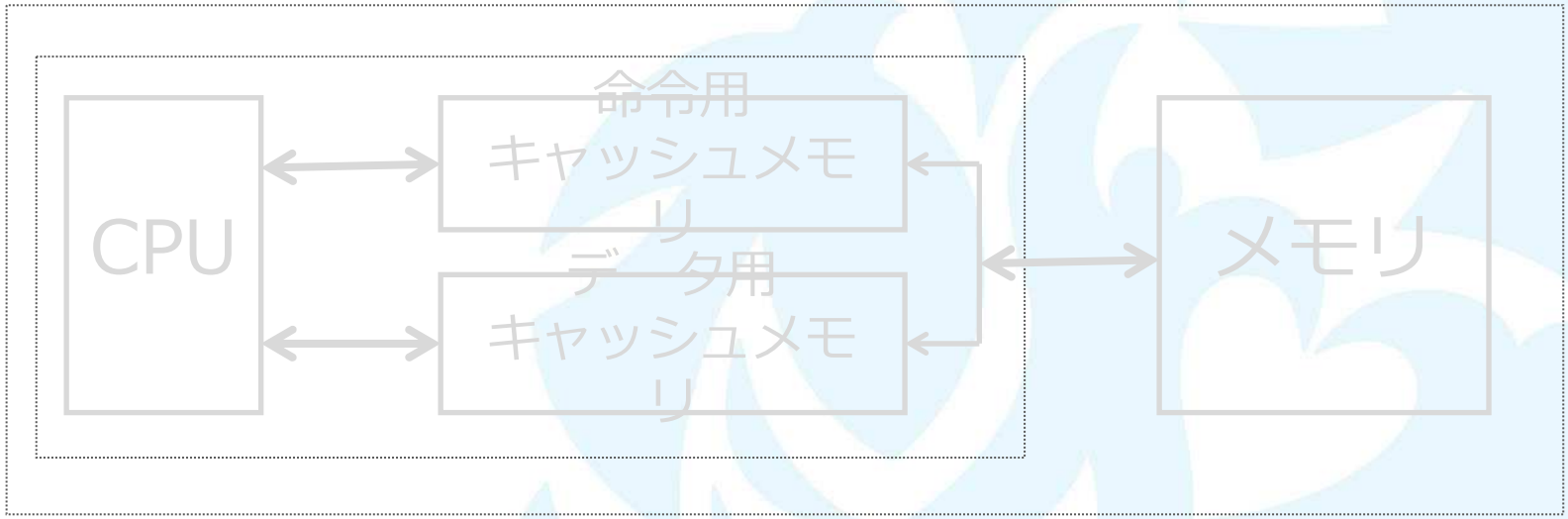
２本別にすればよい？

⇩

メモリを予め区分
　⇒　容量の柔軟性が失われる

# ハーバードアーキテクチャの考え方

## 現実には「キャッシュ」(後述)までの経路を２本にするのが普通

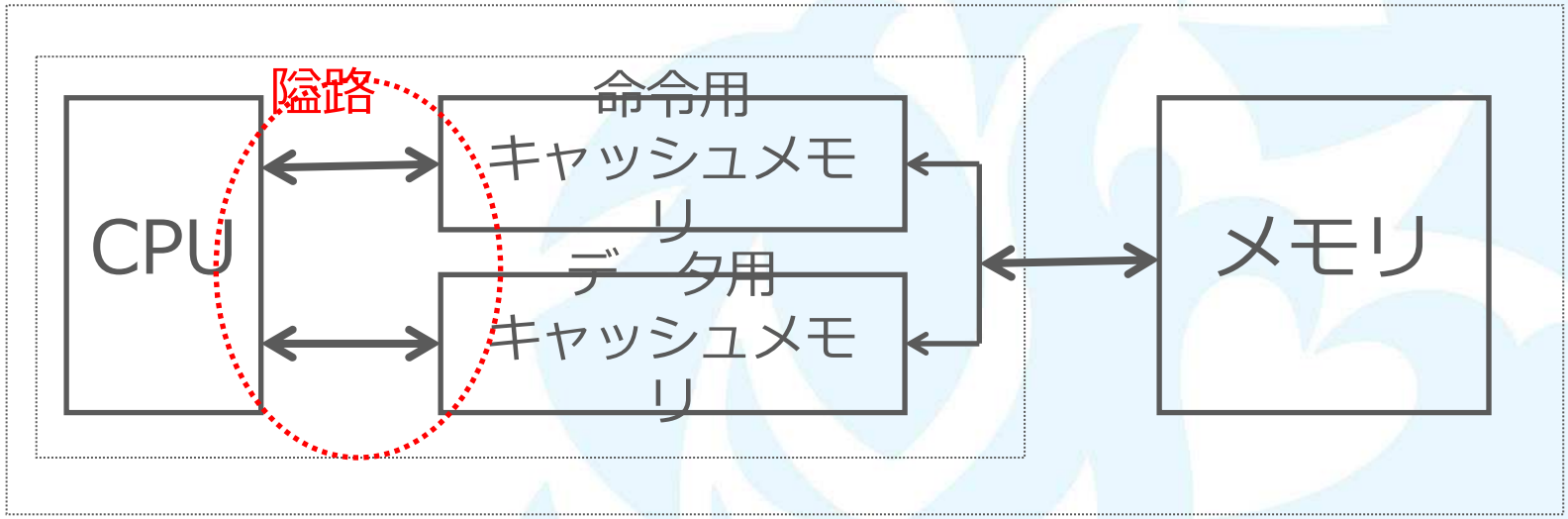





東邦大学

# ハーバードアーキテクチャの考え方

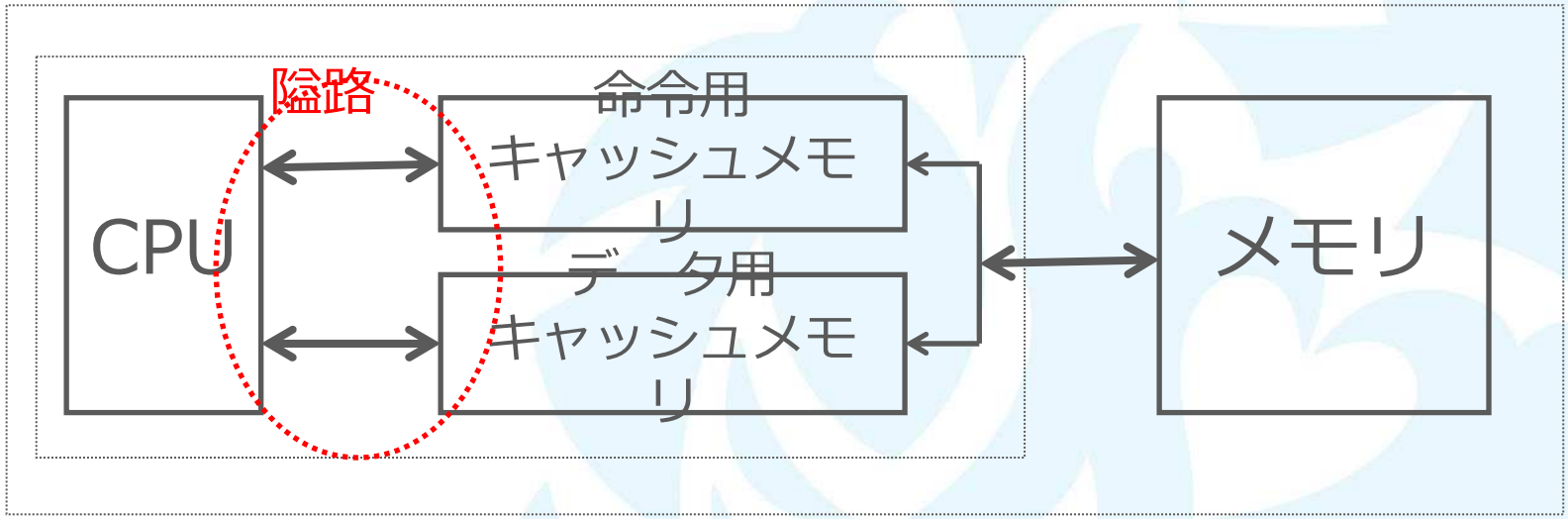
現実には「キャッシュ」(後述)までの
経路を２本にするのが普通




東邦大学

# ハーバードアーキテクチャの考え方

現実には「キャッシュ」(後述)までの
経路を２本にするのが普通




更に、命令とデータでアクセスの動きが違い、
キャッシュの振舞いが変わるので、別々がよい



東邦大学

# ハーバードアーキテクチャのまとめ

ハーバードアーキテクチャとは

- [redacted]を[redacted]分離
  - ⇒ [redacted]避けられる
  - ⇒ 代償 [redacted]
  - ⇒ 現状 [redacted]を分離

# ハーバードアーキテクチャのまとめ

ハーバードアーキテクチャとは

メモリへの経路を　　分離

⇒　　避けられる

⇒ 代償

⇒ 現状　　を分離

# ハーバードアーキテクチャのまとめ

ハーバードアーキテクチャとは

メモリへの経路をデータと命令に分離

⇒ 避けられる

⇒ 代償

⇒ 現状 を分離

# ハーバードアーキテクチャのまとめ

ハーバードアーキテクチャとは

メモリへの経路をデータと命令に分離

⇒ メモリアクセスの隘路が避けられる

⇒ 代償

⇒ 現状 を分離

# ハーバードアーキテクチャのまとめ

ハーバードアーキテクチャとは

メモリへの経路をデータと命令に分離

⇒ メモリアクセスの隘路が避けられる

⇒ 代償　メモリを区分～柔軟性に欠ける

⇒ 現状　　を分離

東邦大学

# ハーバードアーキテクチャのまとめ

ハーバードアーキテクチャとは

メモリへの経路をデータと命令に分離

⇒ メモリアクセスの隘路が避けられる

⇒ 代償　メモリを区分～柔軟性に欠ける

⇒ 現状　CPU～キャッシュまでを分離

# ハーバードアーキテクチャの考え方について分かりましたか？

次へ
(マイクロプログラムへ)

# マイクロプログラムアーキテクチャ

# マイクロプログラムアーキテクチャとは？

CPUの内部構造の作り方
の１つ

作り方の工夫

# マイクロプログラムの考え方

プログラムがマイクロ？

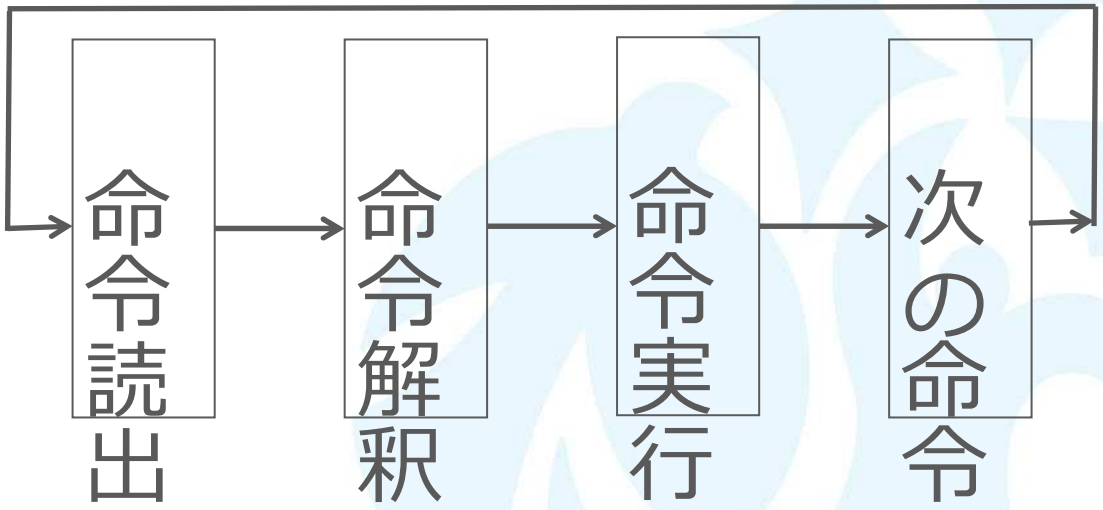
# マイクロプログラムの考え方

１つの命令の実行は４つのステップ

何だったかな？

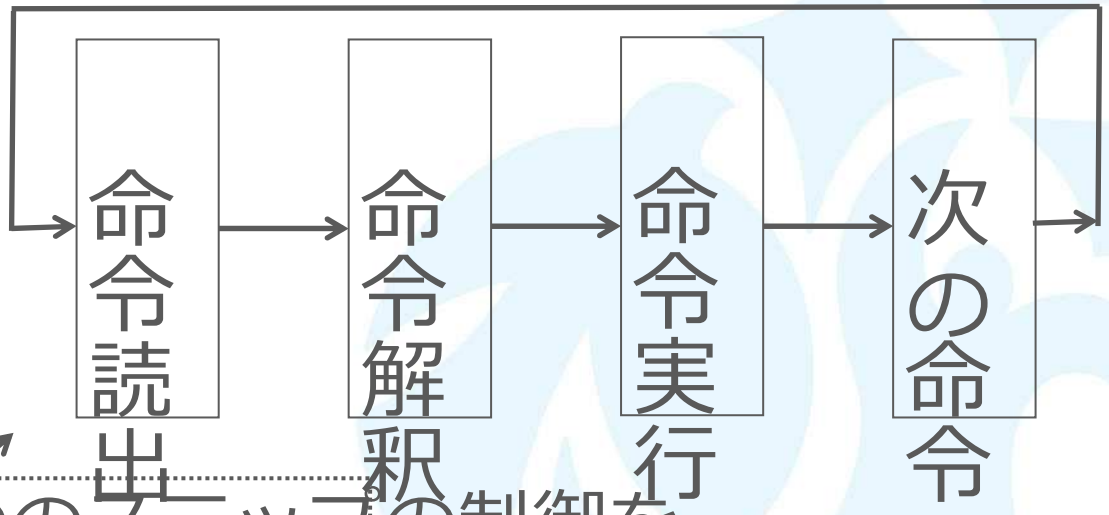
# マイクロプログラムの考え方

## １つの命令の実行は４つのステップ



# マイクロプログラムの考え方

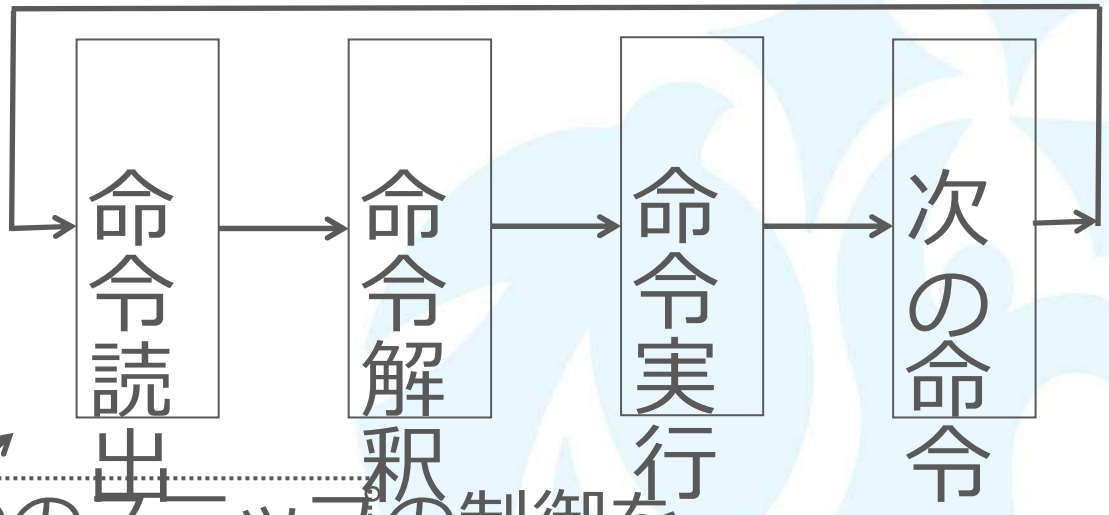
１つの命令の実行は４つのステップ




４つのステップの制御を
ハード(論理回路)で実現してもいいけれど
プログラムもどきで制御してもいいよね

# マイクロプログラムの考え方

１つの命令の実行は４つのステップ




４つのステップの制御を
ハード(論理回路)で実現してもいいけれど
プログラムもどきで制御してもいいよね

# マイクロプログラムの考え方

## １つの命令の実行の４ステップの制御を

# マイクロプログラムの考え方

１つの命令の実行の４ステップの制御をプログラムもどきで行う

# マイクロプログラムの考え方

１つの命令の実行の４ステップの制御を
プログラムもどきで行う

↓

マイクロプログラムと呼ぶ

# マイクロプログラムの考え方

１つの命令の実行の４ステップの制御を
プログラムもどきで行う

↓

マイクロプログラムと呼ぶ

⇕

普通にプログラムするものは
マクロプログラムと呼ぶこと**も**ある

# マイクロプログラムの考え方

言ってしまえば

非常に高速で単純なコンピュータがあって
それが１つ１つの命令を読出し・解釈・実行
しているモデル

# マイクロプログラムの考え方

マイクロプログラム　ハード(布線・回路)

ハードは簡単　　ハードは複雑

# マイクロプログラムの考え方

## マイクロプログラム

ハードは簡単
変更も簡単

## ハード(布線・回路)

ハードは複雑
変更は作り直しで大変

# マイクロプログラムの考え方

## マイクロプログラム

ハードは簡単
変更も簡単
短時間で作れる

## ハード(布線・回路)

ハードは複雑
変更は作り直しで大変
作るのに時間がかかる

# マイクロプログラムの考え方

## マイクロプログラム

ハードは簡単
変更も簡単
短時間で作れる
どうしても遅い
高速化するとコスト高

## ハード(布線・回路)

ハードは複雑
変更は作り直しで大変
作るのに時間がかかる
基本的に速い

# マイクロプログラムの考え方

## 現実的には、組合せる考え方がある

### シリーズ（ファミリー）に使われた

# マイクロプログラムの考え方

現実的には、組合せる考え方がある

シリーズ（ファミリー）に使われた

高速だが高価な機種

布線(回路)による実現

# マイクロプログラムの考え方

現実的には、組合せる考え方がある

シリーズ（ファミリー）に使われた

高速だが高価な機種　　布線(回路)による実現

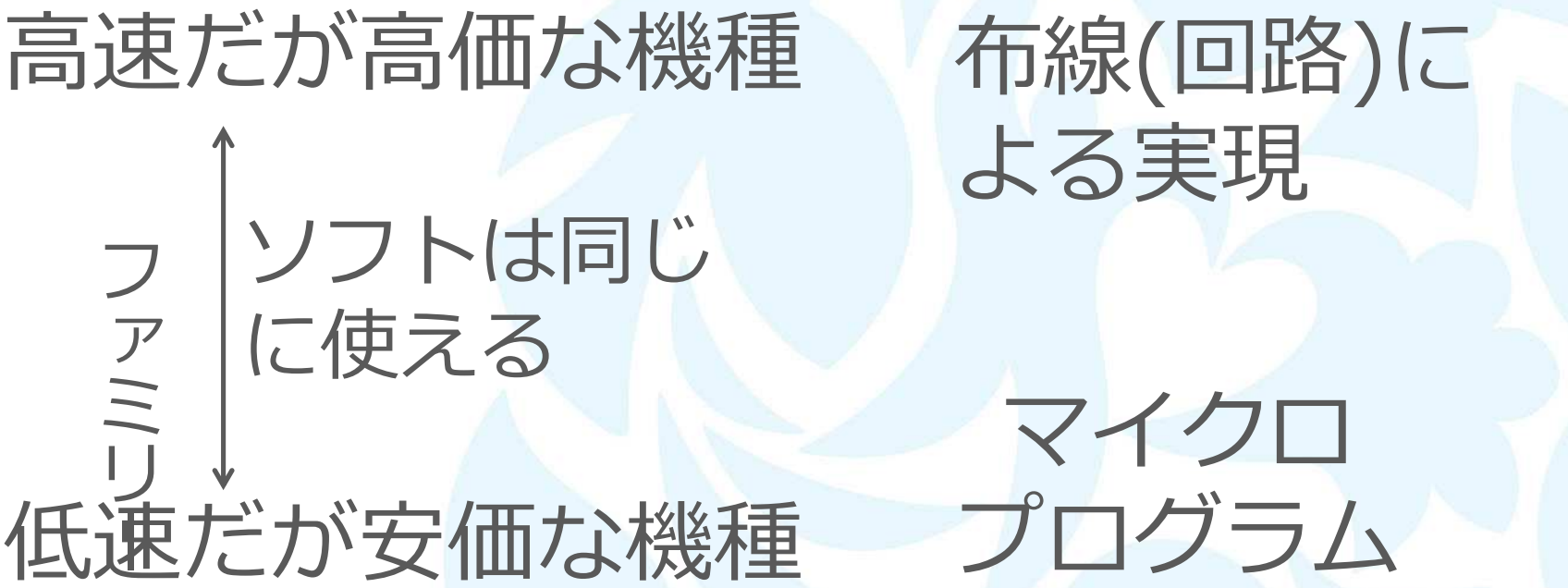
マイクロプログラム

低速だが安価な機種

# マイクロプログラムの考え方

現実的には、組合せる考え方がある

シリーズ（ファミリー）に使われた




東邦大学

# マイクロ命令の形式

マイクロ命令の作り方は
２通り考えられた

# マイクロ命令の形式

マイクロ命令の作り方は
２通り考えられた

水平型　　垂直型

命令の各ビットを
内部信号に対応させる

高速(内部処理が無い)
マイクロステップ数少ない
命令長が非常に長くなる

# マイクロ命令の形式

マイクロ命令の作り方は
２通り考えられた

## 水平型

命令の各ビットを
内部信号に対応させる

高速(内部処理が無い)
マイクロステップ数少ない
命令長が非常に長くなる

## 垂直型

内部信号をエンコード

命令長は短い
低速(デコード必要)
マイクロステップ数多い

# マイクロ命令の形式

マイクロ命令の作り方は
２通り考えられた

## 水平型

命令の各ビットを
内部信号に対応させる

高速(内部処理が無い)
マイクロステップ数少ない
命令長が非常に長くなる

## 垂直型

内部信号をエンコード

命令長は短い
低速(デコード必要)
マイクロステップ数多い

東邦大学

# マイクロプログラムのまとめ

マイクロプログラムアーキテクチャは
の実現法の
議論である

マイクロプログラムは と対比する

マイクロプログラムは
各命令を で実現
は
各命令を で実現

# マイクロプログラムのまとめ

マイクロプログラムアーキテクチャは
　１つの命令の　　の実現法の
　　議論である

マイクロプログラムは　　と対比する

マイクロプログラムは
　各命令を　　で実現

　　は
　各命令を　　で実現

# マイクロプログラムのまとめ

マイクロプログラムアーキテクチャは
１つの命令の実行サイクルの実現法の
議論である

マイクロプログラムは[　　]と対比する

マイクロプログラムは
各命令を[　　]で実現

[　　]は
各命令を[　　]で実現

# マイクロプログラムのまとめ

マイクロプログラムアーキテクチャは
　１つの命令の実行サイクルの実現法の
　　議論である

マイクロプログラムは布線論理と対比する

マイクロプログラムは
　各命令を　　で実現

　　は
　各命令を　　で実現

# マイクロプログラムのまとめ

マイクロプログラムアーキテクチャは
　１つの命令の実行サイクルの実現法の
　　議論である

マイクロプログラムは布線論理と対比する

マイクロプログラムは
　各命令を複数のマイクロ命令で実現
　　は
　各命令を　　で実現

# マイクロプログラムのまとめ

マイクロプログラムアーキテクチャは
　１つの命令の実行サイクルの実現法の
　　議論である

マイクロプログラムは布線論理と対比する

マイクロプログラムは
　各命令を複数のマイクロ命令で実現

布線論理は
　各命令を　　で実現

# マイクロプログラムのまとめ

マイクロプログラムアーキテクチャは
　１つの命令の実行サイクルの実現法の
　　議論である

マイクロプログラムは布線論理と対比する

マイクロプログラムは
　各命令を複数のマイクロ命令で実現

布線論理は
　各命令をハードの回路で実現

# マイクロプログラムアーキテクチャの考え方について分かりましたか？

次へ

(命令の実行性能)



東邦大学