CPU実験を語る会―コア編―

炭素12

WHAT IS THE “CPU EXPERIMENT”

CPU実験とは

CPU実験とは

- プロセッサを設計する
- コンパイラを作る
- 課題プログラムを動作させる

プロセッサ実験配布資料より引用

CPU実験の理念

- 計算機の仕組みを理解
- プロジェクト管理の体験

プロセッサ実験配布資料より引用

役割分担

- コア係
- コンパイラ係
- シミュレータ係
- FPU係
- （I/O係）

コア係に向いている人

1. プロセッサに興味のある人
2. 課題を手早く済ませる人
3. 暇で（バイトやサークルが無く）地下によくいる人

PROCESSOR DESIGN

プロセッサ設計

プロセッサ設計

1. 命令セットアーキテクチャ(ISA)設計
   - どの命令を？
2. マイクロアーキテクチャ設計
   - どういう仕組みで？
3. 論理回路設計
   - どう実装する？

プロセッサ設計

1. 命令セットアーキテクチャ(ISA)設計
   - どの命令を？
2. マイクロアーキテクチャ設計
   - どういう仕組みで？
3. 論理回路設計
   - どう実装する？

ISA設計

- どの命令を採用する？
- どういうオペランドを取る？
- どういうビットパターンを割り当てる？

ISA設計

- プログラマ（というかコンパイラ係）から見たプロセッサのインタフェースを決定する
  - コンパイラ係とよく相談してください

ISA設計―命令種の決定―

- 採用する命令を決める
- RISCをパクって決めると楽
  - パタヘネがMIPSなのでMIPSベースが多かった
  - パクリ元はMIPS, SPARC, POWERなどが人気

ISA設計―命令仕様の決定―

- 命令の詳しい仕様を決める
  - メモリのアドレッシング単位
    - ワードアドレッシングがオススメ
  - メモリのアドレッシングモード
  - ビット列の拡張方式
  - シフト方式
  - その他オペランドの詳細な意味

ISA設計―ビットパターンの決定―

- 機械語のビットパターンを決める
- 形式をある程度統一しておくと、実装が楽になる
  - レジスタ番号の位置を揃える
  - 似たような命令に同じ命令コード接頭辞を割り当てる

ISA設計―ビットパターンの決定―

- 3rdアーキテクチャの例

プロセッサ設計

1. 命令セットアーキテクチャ(ISA)設計
   - どの命令を？
2. マイクロアーキテクチャ設計
   - どういう仕組みで？
3. 論理回路設計
   - どう実装する？

マイクロアーキテクチャ設計

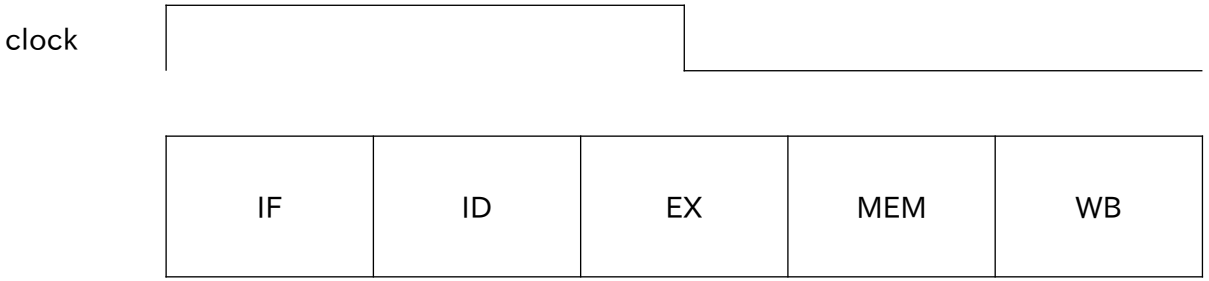
- ISAをどういう仕組みで実装するか？
  - 命令の実行方式
  - 分岐方式
  - キャッシュの有無

マイクロアーキテクチャ設計
―命令の実行方式―

- シングル（単一）クロックサイクル
  - 命令のフェッチから完了までを1クロックで行う
- マルチクロックサイクル
  - ステートマシンを用い、1命令を数クロックかけて実行する

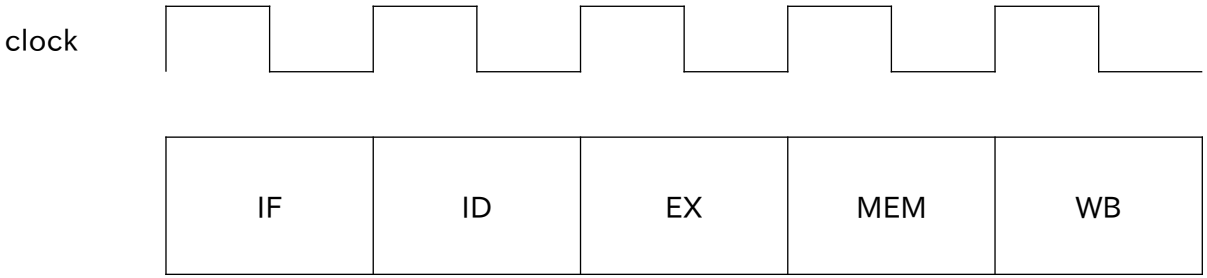
マイクロアーキテクチャ設計
―命令の実行方式―

- シングルクロックサイクル



マイクロアーキテクチャ設計
―命令の実行方式―

- マルチクロックサイクル

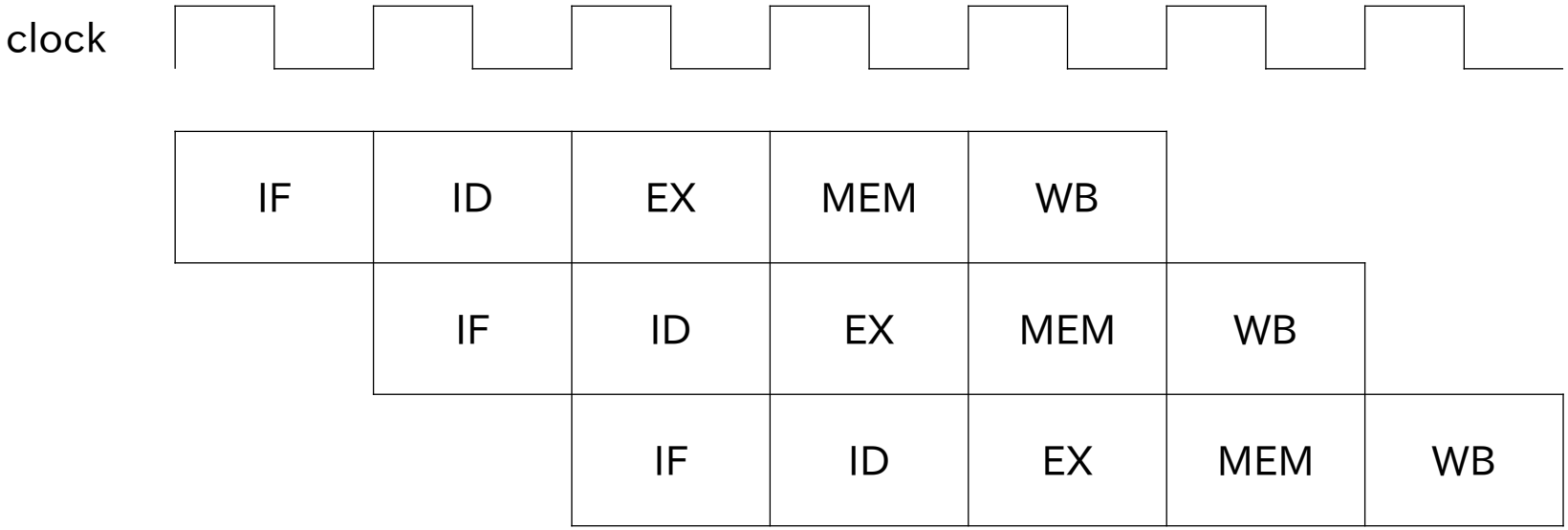


マイクロアーキテクチャ設計 ―命令の実行方式―

- パイプライン
  - 命令を、並列して実行できる部分（ステージ）に分割し、異なる命令の異なるステージをオーバーラップさせて実行する

マイクロアーキテクチャ設計
―命令の実行方式―

- パイプライン

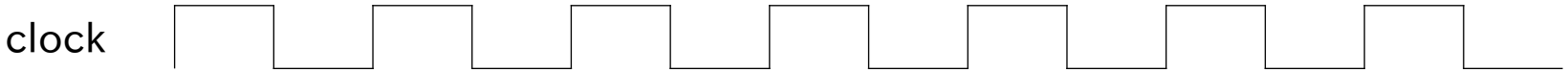


マイクロアーキテクチャ設計
―命令の実行方式―

- スーパースカラ
  - 複数命令を同時にパイプラインで実行する
  - プロセッサが並列性を検出する
- VLIW
  - 複数の処理を同時にパイプラインで実行する
  - コンパイラが並列性を検出する

マイクロアーキテクチャ設計
―命令の実行方式―

- スーパースカラ・VLIW



IF	ID	EX	MEM	WB		
IF	ID	EX	MEM	WB		
	IF	ID	EX	MEM	WB	
	IF	ID	EX	MEM	WB	
		IF	ID	EX	MEM	WB
		IF	ID	EX	MEM	WB

マイクロアーキテクチャ設計
―命令の実行方式―

- インオーダー実行
  - プログラムの順序通りに命令を実行する
- アウトオブオーダー実行
  - 命令の実行順序を動的に変更する

マイクロアーキテクチャ設計 ―分岐方式―

- 遅延分岐
  - 遅延スロット（分岐先が確定するまでの間）に、コンパイラが関係ない命令を移動・挿入しておく
- 分岐予測と投機実行
  - 直近の分岐の情報から次の分岐の成立・不成立を予測
  - 分岐先が確定する前に予測先の命令を実行

マイクロアーキテクチャ設計 —1stアーキテクチャ—

- マルチクロックサイクルがオススメ
  - シングルクロックサイクルにしようとしても、結局メモリアクセスに時間がかかる

マイクロアーキテクチャ設計 —2ndアーキテクチャ—

- お好きにどうぞ
  - 命令長56bit×VLIWの夢は潰えたけどな

プロセッサ設計

1. 命令セットアーキテクチャ(ISA)設計
   - どの命令を？
2. マイクロアーキテクチャ設計
   - どういう仕組みで？
3. 論理回路設計
   - どう実装する？

論理回路設計

- どういうモジュールに分割する？
- 制御はどうする？
- データバスはどうする？
- 外部との通信はどうする？

論理回路設計―モジュール設計―

- 適度な大きさのモジュールに分割しよう
  - RS232C受信・送信、メモリコントローラ、バスアービタ……
  - VHDLでサポートしているモジュール単位はcomponent

論理回路設計―モジュール設計―

- 一般的なソフトウェアの設計原則がある程度当てはまる（かも？）
  - 少ないインタフェース
    - 通信するモジュールの数は少なく
  - 小さいインタフェース
    - モジュール間でやりとりするデータは小さく
  - 単一責任
    - 一つの仕事は一つのモジュールだけが担当する

論理回路設計―制御―

- 一番難しい部分
- タイミングチャートを書こう
  - これを見ながらステートマシンを設計する

論理回路設計―メモリ―

- ハーバードアーキテクチャがオススメ
  - 命令はBlock RAMに載る
  - プログラムのデータはSRAMに載せて
  - ノイマンアーキテクチャの犠牲者が約1名

論理回路設計—I/O—

- I/Oはバグる
  - でもデータにノイズはそんなに無い
    - タイミングが合ってないのが原因？
  - 信頼性を高めておくと精神的に安心する
  - オープンソースの実装が参考になる
    - UARTでググってみよう

WHAT A F**CKING LANGUAGE!

VHDL

VHDL

- 米国防総省が開発したハードウェア記述言語
  - 元々はデジタル回路を文書にするための言語
  - シミュレーションや論理合成もできる
  - cf.米国防総省が開発したプログラミング言語
    - COBOL
    - Ada

VHDL

- VHDL文法のいろいろ
- VHDLを楽に書こう

VHDL

- VHDL文法のいろいろ
- VHDLを楽に書こう

VHDL文法のいろいろ

- 構造体を使おう
  - コードがすっきりして見通しが良くなる
- numeric_stdパッケージを使おう
  - http://www.synthworks.com/papers/vhdl_math_tricks_mapld_2003.pdf

VHDL

- VHDL文法のいろいろ
- VHDLを楽に書こう

VHDLを楽に書こう

- EmacsのVHDLプラグインはほげ
- EclipseのVHDLプラグインはそこそこ便利
- マクロテンプレートを使う班があったり
- VHDLを他言語から生成するという手も

~~VHDL~~を楽に書こう

- Verilogを使った班も

PROCESSOR DEBUG

プロセッサのデバッグ

プロセッサのデバッグ

- PC上でのデバッグ
- FPGA基板上でのデバッグ

プロセッサのデバッグ

- PC上でのデバッグ
- FPGA基板上でのデバッグ

PC上でのデバッグ
―HDLシミュレータ―

- シミュレータで発見したバグは比較的簡単に原因が探せる
- HDLシミュレータが大活躍
  - ModelSim
    - メンター・グラフィックス社の製品
    - 高機能・高速
  - GHDL
    - オープンソースのソフトウェア
    - 地下に来られない時にも使える

PC上でのデバッグ
―シミュレーションテスト―

- SRAMやRS232C通信のシミュレーションモデルを作る
  - 全部ひっくるめてテストできる
- テストはなるべく自動化する
  - 手作業が入るとミスる上にテストが面倒くさくなる

プロセッサのデバッグ

- PC上でのデバッグ
- FPGA基板上でのデバッグ

FPGA基板上でのデバッグ

- PCからの操作でメモリやレジスタが覗けるようにしておくと便利かも？
  - 10erの方がCPU実験を語る会で言ってたような
- LEDを付けた拡張基板を活用するとよいかも？

FPGA基板上でのデバッグ

- でもやっぱり基本的に難しい

PROJECT MANAGEMENT

プロジェクト管理

プロジェクト管理

- 進捗管理
- タスク管理
- バージョン管理
- 班員間のコミュニケーション

プロジェクト管理

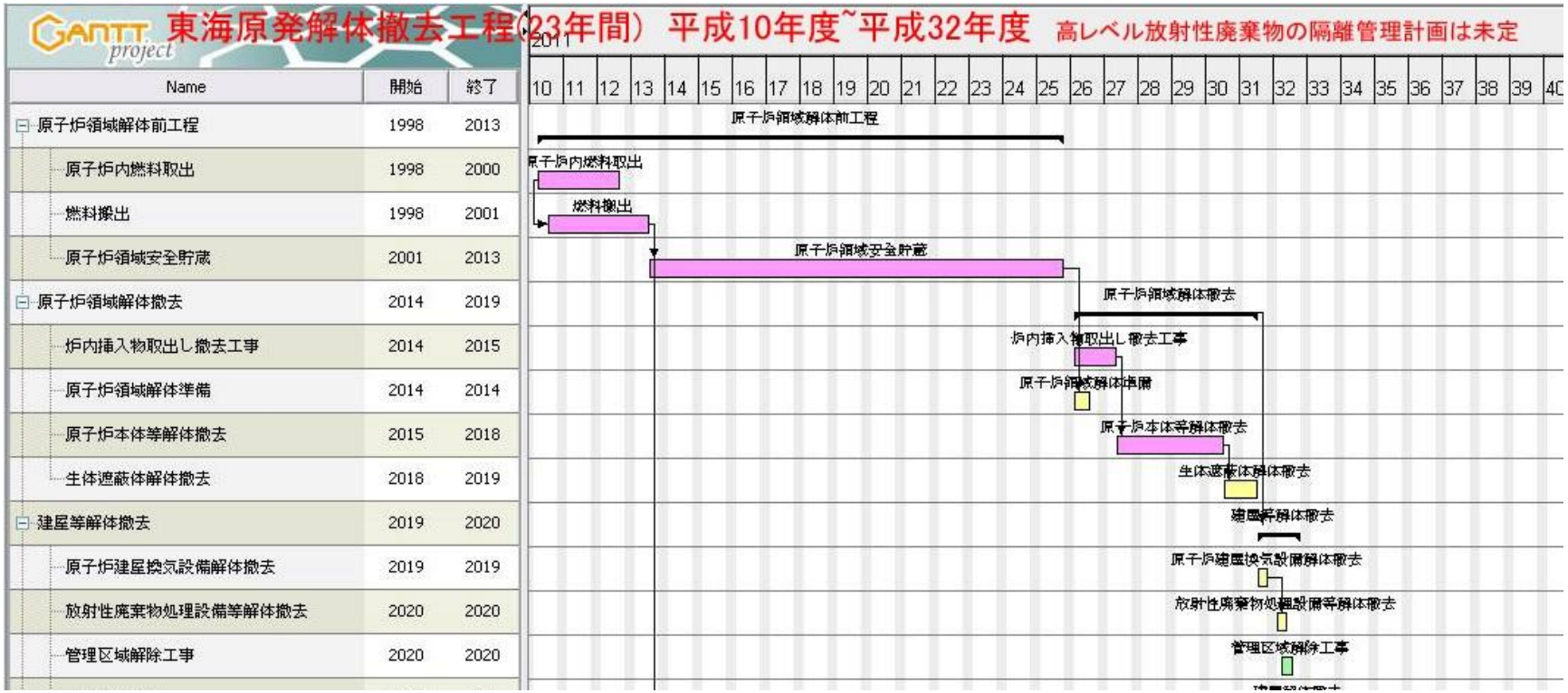
- 進捗管理
- タスク管理
- バージョン管理
- 班員間のコミュニケーション

進捗管理

- とりあえずガントチャートは作成する
  - 毎週の進捗報告ではガントチャートの作成が求められる
  - 見た目は良い
  - 実際の進捗管理には使いにくい
  - なぜかみんなでガント.comを使う班が多かった

進捗管理

- ガントチャートの例



ガントチャート – Wikipedia
(http://ja.wikipedia.org/wiki/ガントチャート)より一部改変

進捗管理

- そんなにきっちり管理しなくても大丈夫
  - だいたい他の人がどんな作業やってるか把握しておけばよい
  - でも詰まってたら助けてあげてね
- 進捗偽装も大事（？）

プロジェクト管理

- 進捗管理
- タスク管理
- バージョン管理
- 班員間のコミュニケーション

タスク管理

- タスクの把握は大切
  - 進捗管理は適当でいいけど
- タスクは細かく分割する
- タスク管理ツールを使うとよい
  - プロジェクトホスティングサービス付属のツールとか

プロジェクト管理

- 進捗管理
- タスク管理
- バージョン管理
- 班員間のコミュニケーション

バージョン管理

- バージョン管理ツールは必須
- プロジェクトホスティングサービスが便利
  - GitHubやBitbucketなどいろいろ
- バージョン管理ツールに習熟しておく
  - 数GBのゴミをリポジトリに入れない

プロジェクト管理

- 進捗管理
- タスク管理
- バージョン管理
- 班員間のコミュニケーション

班員間のコミュニケーション

- 班員とのコミュニケーションをしっかりしよう
  - 日報システムはなかなかよかった
- 仕様はきちんと決めてきちんと共有しよう
  - 変更したらちゃんと連絡する
- 他の人が何をしているか把握しよう

その他

- VHDLのコーディング規約は統一しておくとよいかも？
  - コア係がFPUを見たりなんだりしやすくなる

その他

- **ちゃんと審判サーバと通信して動作することを確認すること**
  - 直前に焦った班：1
  - 直前に悟った班：1
- 入力データは**バイナリ**形式で渡される
  - 気付かなかった班：5

REFERENCES

参考資料

参考資料

- コンピュータの構成と設計 第4版
  - 通称「パタヘネ」
  - 最近第4版が出た
  - コア係は必読
  - **よく読もう**
    - この資料の構成メモを見返したら「パタヘネをよく読もう」が2枚あった

参考資料

- コンピュータアーキテクチャ 定量的アプローチ 第4版
  - 通称「ヘネパタ」
  - より進んだアーキテクチャを実装する際に参考になる

参考資料

- VHDLデジタル回路設計 標準講座
  - 評判がよいデジタル回路設計の教科書

参考資料

- Verilog HDL&VHDLテストベンチ記述の初歩
  - テストベンチを記述する際のテクニックが豊富に紹介されている

参考資料

- ザイリンクス：製品サポートと資料
  - データシートやユーザーガイドなどが役に立つ
- OpenCores
  - オープンソースハードウェアのコミュニティ

IN CLOSING

終わりに

プロセッサ実験への姿勢

- プロセッサそのものや競争することに関心・興味があるわけではないのなら、手を抜いて他のことした方が幸せ
- 低空飛行するか最高速を狙うかは他の班員とよく相談しよう

最後に

- パイプラインは難しい
- でも変態アーキテクチャに期待
- 消える人が出ないように