Tokyo University of Agriculture and Technology

RISC-Vアーキテクチャにおける同時マルチスレッディング(SMT)の実現

東京農工大学 中條研究室 修士1年

長岡慶太

自己紹介

Tokyo University of Agriculture and Technology

長岡 慶太

- 2021年3月 東京農工大学 工学部 情報工学科 卒業
- 卒論テーマ：RISC-Vにおけるマルチスレッドアーキテクチャの実現
- 現在，東京農工大学 中條研究室 修士1年
- FPGAやSMTプロセッサに関する研究に関心がある

背景：命令レベル並列性の限界

Tokyo University of Agriculture and Technology

スカラ
プロセッサ

```
        ⋮
add     x3, x4, x5
sub     x4, x3, x1
bne     x3, x4, LABEL
li      x6, 10
        ⋮
```

1命令ずつ実行

➡ **遅い**

命令レベル並列性
を引き出すと

スーパースカラ
プロセッサ

```
        ⋮
add     x3, x4, x5
sub     x4, x3, x1
bne     x3, x4, LABEL
li      x6, 10
        ⋮
```

3命令ずつ実行

➡ **速い？**

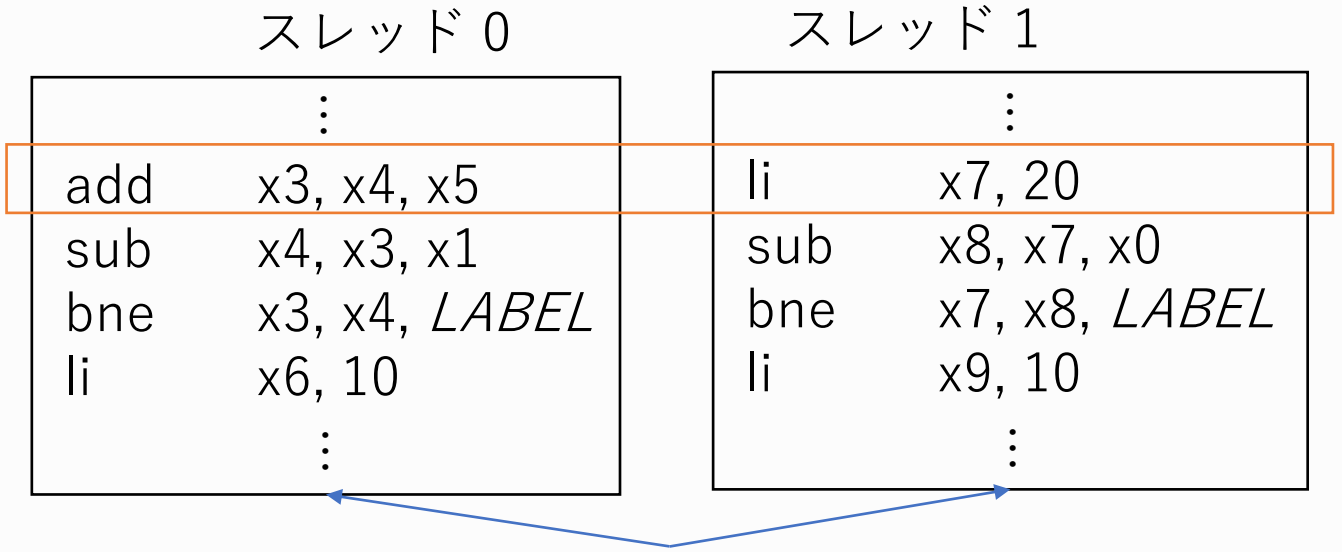
しかし

オペランドの依存関係により**同時処理できる命令数は限られる**

命令レベル並列性には限界がある！

背景：スレッドレベル並列性

異なる複数の命令ブロック（スレッド）を用意



スレッドをまたいで実行することにより
命令依存問題を回避

オペランド依存関係がない
＝ **同時実行可能**

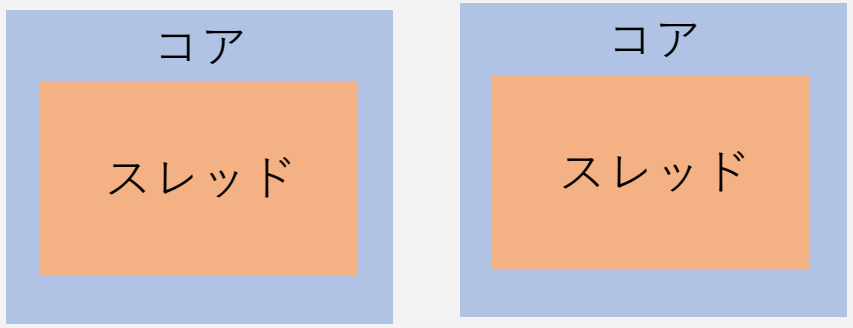
スレッドレベル並列性を利用することで
性能向上

背景：SMTの概要

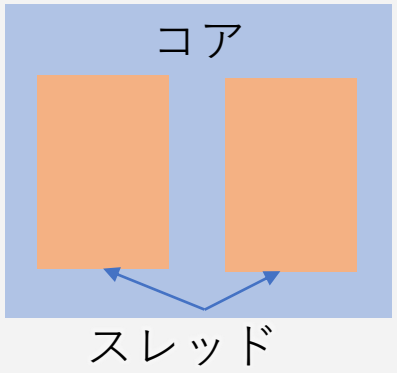
Tokyo University of Agriculture and Technology

SMTとは・・・
同時マルチスレッディング (Simultaneous Multi-Threading)

通常のマルチコア
(2コア/2スレッド)



SMT
(1コア/2スレッド)



少ないリソースでマルチスレッディング可能

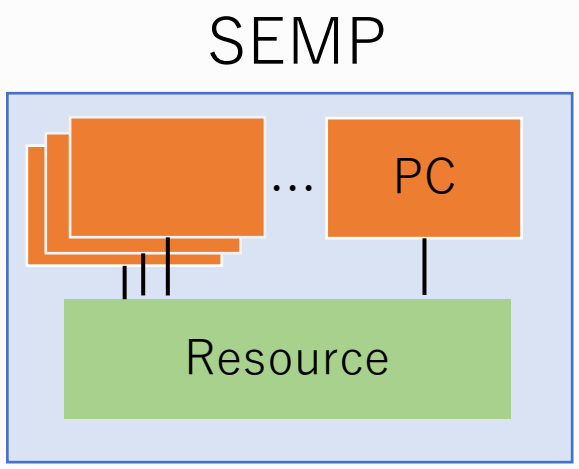
背景：過去の中條研の取り組み①

Tokyo University of Agriculture and Technology

SEMPプロセッサ

MIPSにおける同時マルチスレッディング(SMT)プロセッサを開発

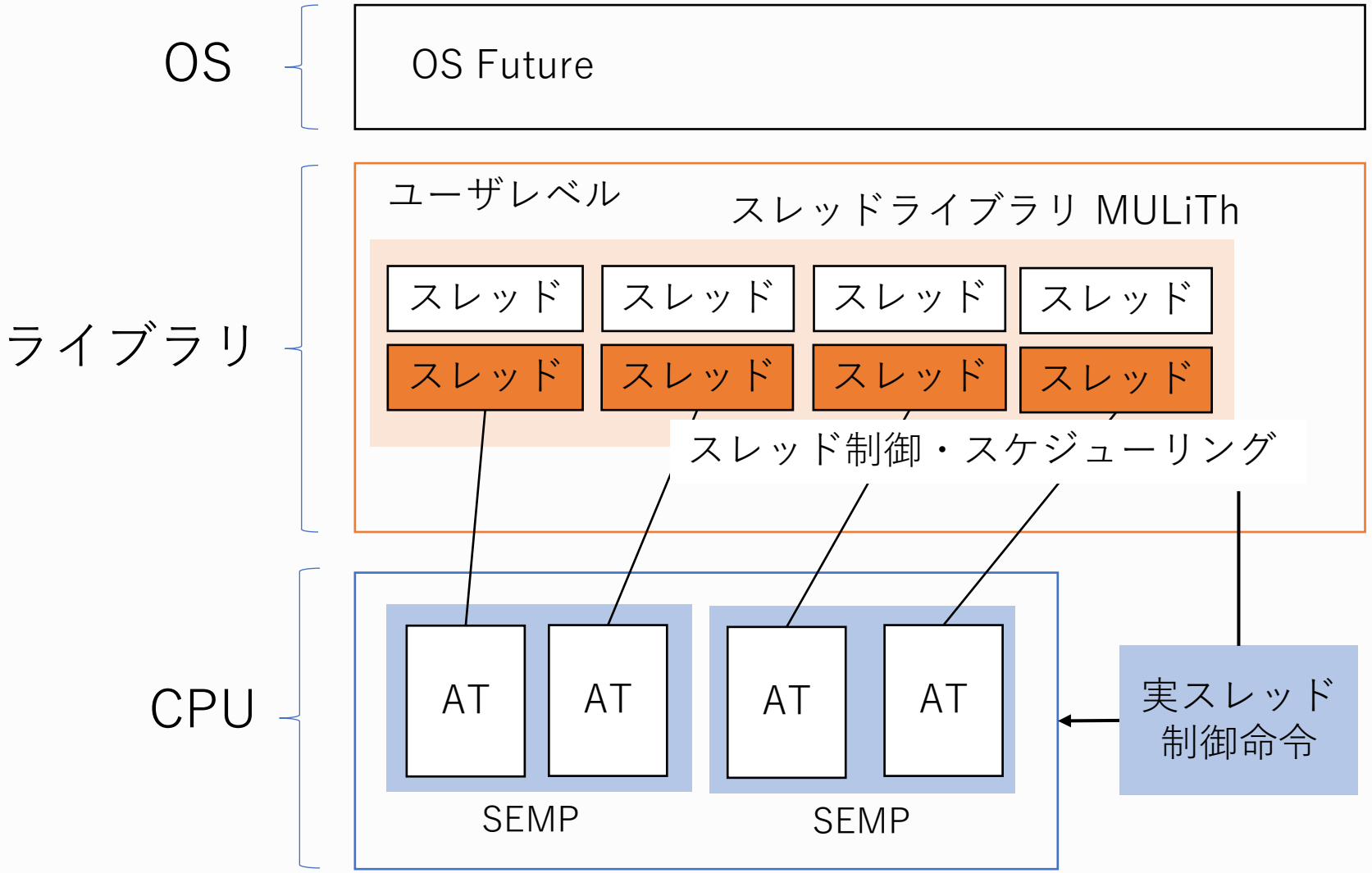
命令レベル並列性だけでなくスレッドレベル並列性を抽出することを目的



表：SEMPの仕様

パラメータ	説明
スレッド数	2~8 (実装による)
データ幅	32 bit
発行幅	2
演算器	ALU×2, Load/Store Unit×1
物理レジスタ	92 エントリ
ROB	24 エントリ
Issue Queue	8 エントリ × 2

背景：過去の中條研の取り組み②



OchiMuSシステム

- CPU～OSまでのMIPSのSMTシステム
- ユーザレベルからのスレッド制御が可能
- OSを介さないので，高速な制御が可能

SACSIS2005 最優秀独創的研究アイデア賞：加藤義人, 大和仁典, 小笠原嘉泰, 佐藤未来子, 笹田耕一, 内倉要, 中條拓伯, 並木美太郎：SMTプロセッサのFPGAへの実装と評価, SACSIS 2005, Vol.2005, No.5, pp.239-240 (2005)

背景：RISC-Vの登場

Tokyo University of Agriculture and Technology

新興命令セットアーキテクチャ(ISA)　　**RISC-V** の登場

従来のISA	RISC-V
△高額なライセンス料 △オープンではない △勝手な拡張は禁止	◎フリー ◎オープン ◎高い拡張性

自由に拡張ができ、高額なライセンス料を払わずにプロセッサを作ることが可能に！

目的

Tokyo University of Agriculture and Technology

RISC-Vプロセッサにカスタム命令を活用し SMT（同時マルチスレッディング）を実現する

- RISC-Vにおいて命令レベル並列性・スレッドレベル並列性・データレベル並列性の抽出できるプロセッサの製作を目標
- オープンソースにしていくことも検討中

実現方法

Tokyo University of Agriculture and Technology

過去に当研究室で開発したMIPSにおける
SMTプロセッサをRISC-Vに移植

MIPS
SMTプロセッサ

↓

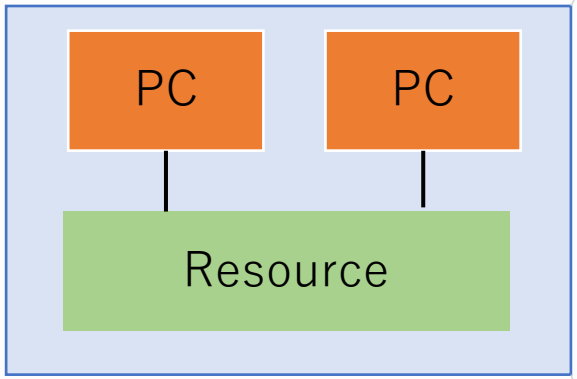
RISC-V
SMTプロセッサ

変更点
- 命令デコーダ
- データパス
- コントローラ(制御)

本SMTプロセッサの仕様

Tokyo University of Agriculture and Technology

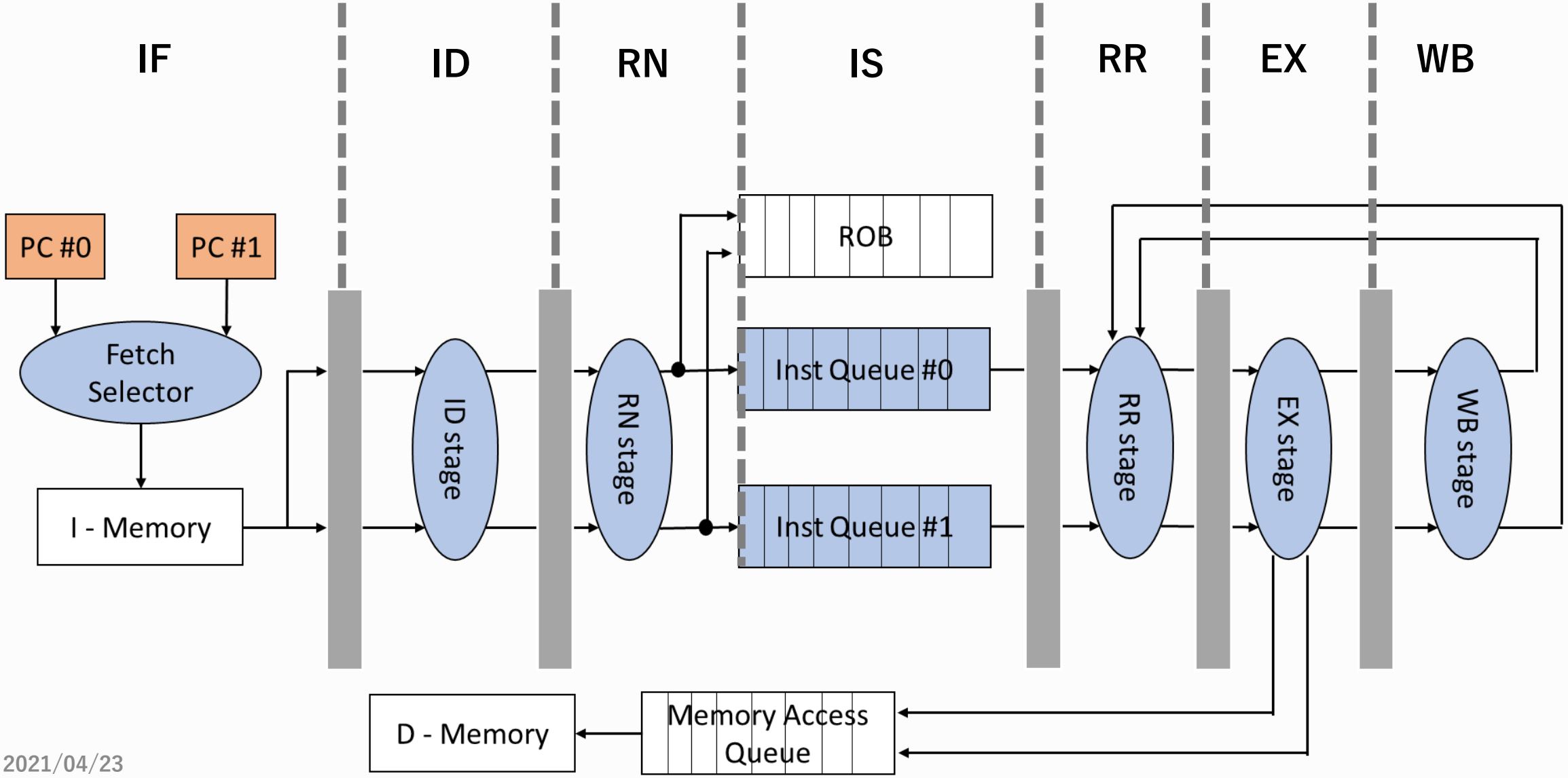
RISC-V SMT



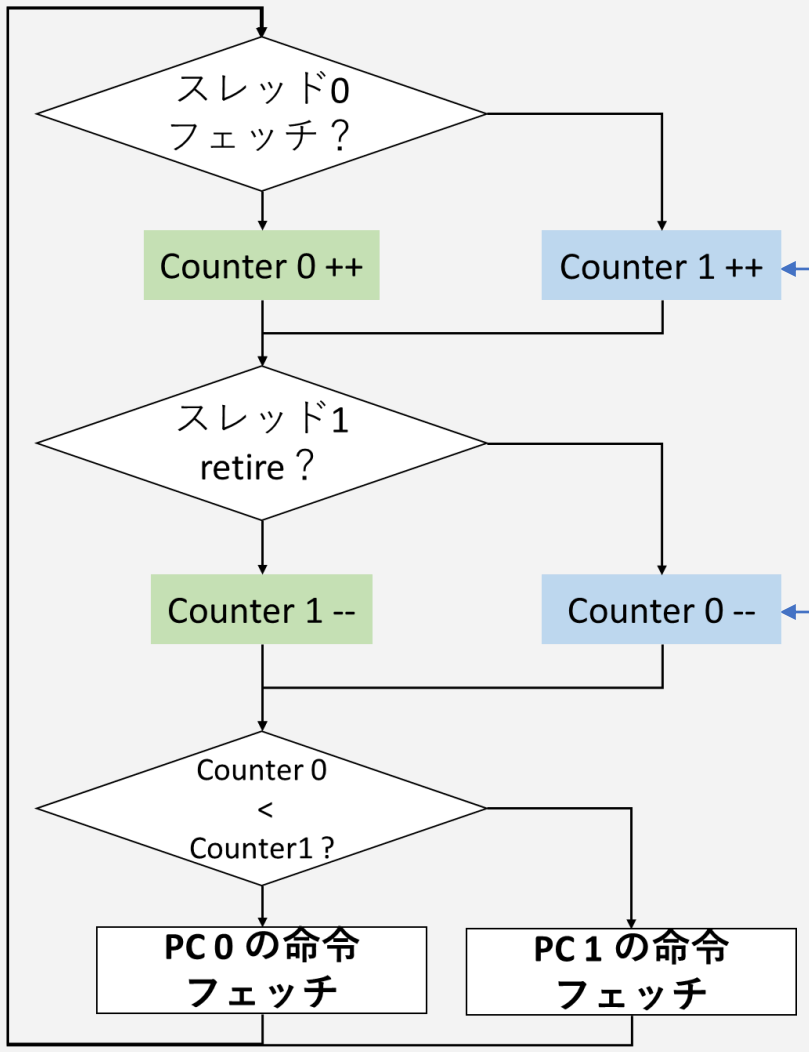
Out-of-Order
スーパースカラ

パラメータ	説明
命令セットアーキテクチャ	RV32I+カスタム命令
記述言語	Verilog HDL
キャッシュ	なし
実スレッド数	2
データ幅	32 bit
発行幅	2
演算器	ALU×2, Load/Store Unit×1
物理レジスタ	92 エントリ
ROB	24 エントリ
Issue Queue	8 エントリ × 2

本SMTプロセッサの構造図



本SMTプロセッサの機構①



命令フェッチセレクタ … フェッチするスレッドの選択

それぞれのスレッドが**カウンタ**を持つ

- スレッド0カウンタ
- スレッド1カウンタ

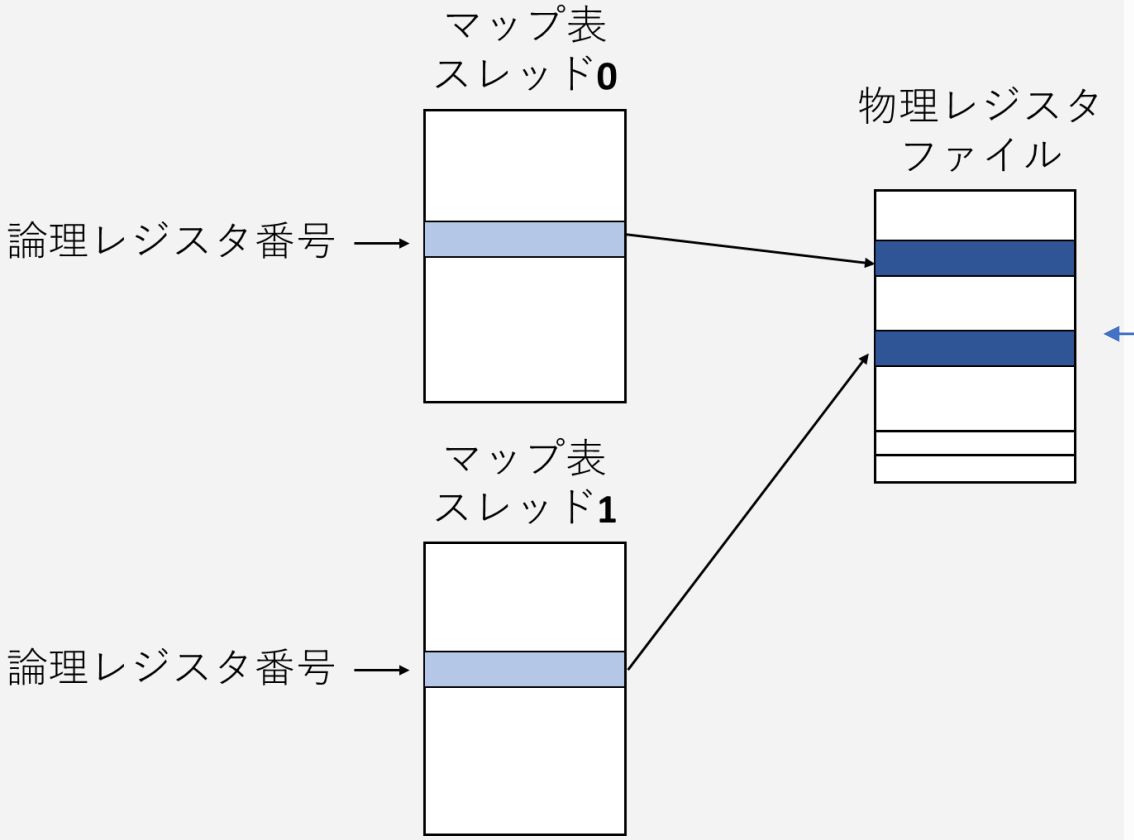
各スレッドで

- 命令フェッチのとき カウンタ++
- 命令リタイアのとき カウンタ --

常にカウンタ値が少ないスレッドの命令をフェッチ

➡**プロセッサ内のスレッドの命令数を一定にできる**

本SMTプロセッサの機構②



レジスタリネーミング

各スレッドで

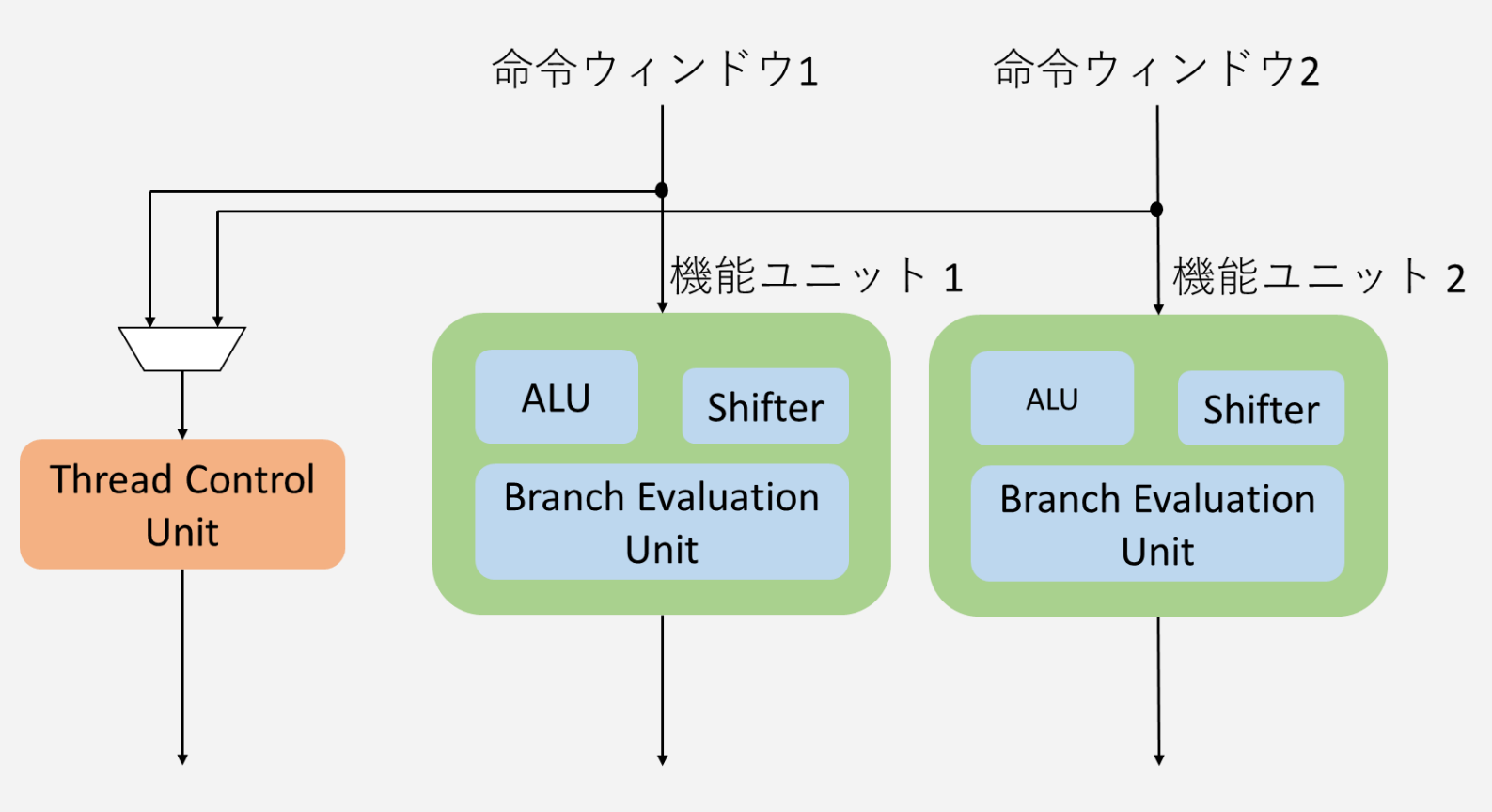
- 物理レジスタファイルを持たない
- マップ表をそれぞれ持つ

スレッド間で
物理レジスタファイルを共有する

シングルスレッド動作時でも物理レジスタを最大限使える

➡ **リソースの効率的な利用が可能**

本SMTプロセッサの機構③



実行ステージでは

機能ユニット

- ALU
- Shifter
- 分岐判定ユニット

とスレッド制御を行う**スレッドコントロールユニット(TCU)**をもつ

本SMTプロセッサの特徴

Tokyo University of Agriculture and Technology

スレッド制御命令は**ユーザ命令**として実装

- ➡ システムコールを呼び出す必要がなく**高速動作**

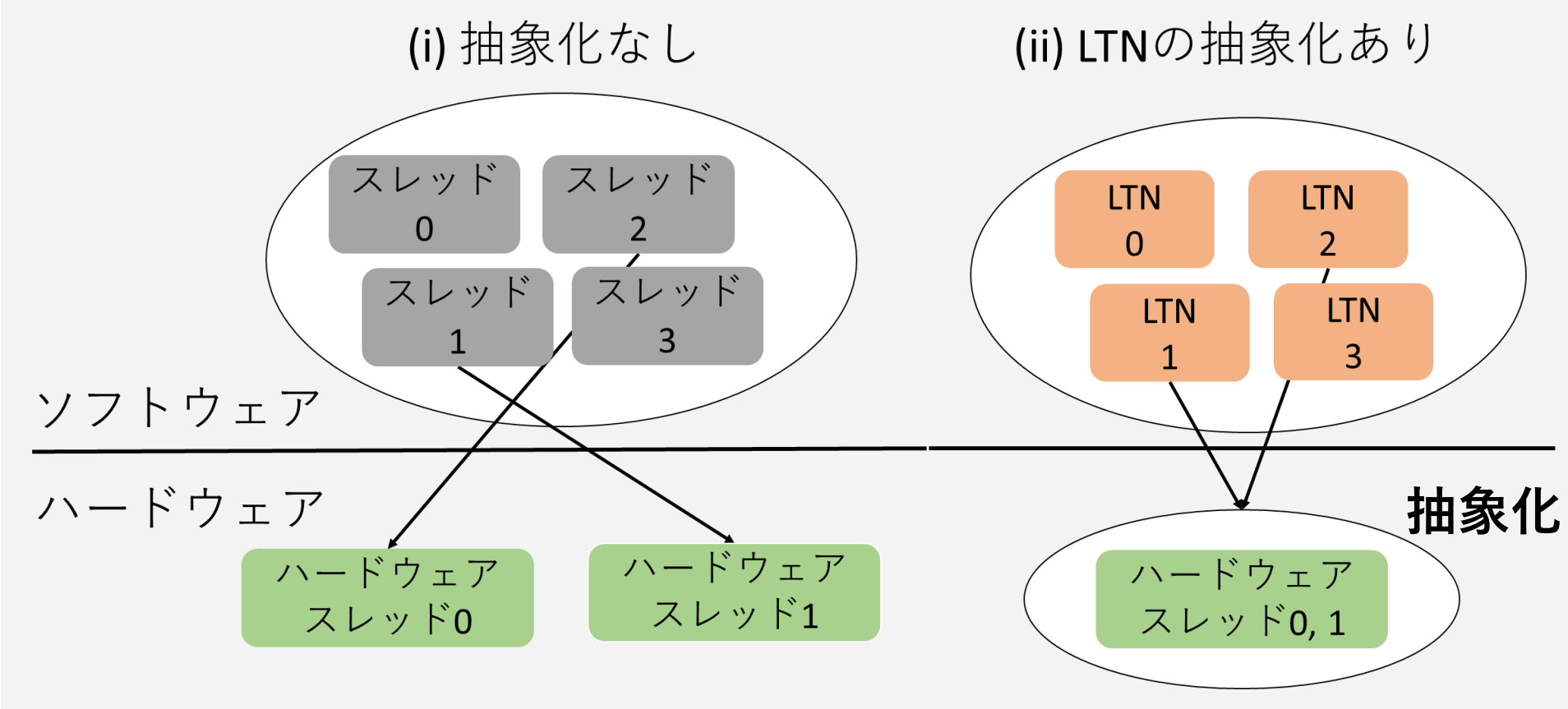
スレッド制御命令はRISC-Vカスタム命令により定義

- ➡ 既存のコンパイラに**少ない改造で利用可能**

LTN (論理スレッド番号)による実スレッドの抽象化

- ➡ **LTN**で制御対象のスレッドを指定すればよいので
  ソフトウェア開発の負担を軽減

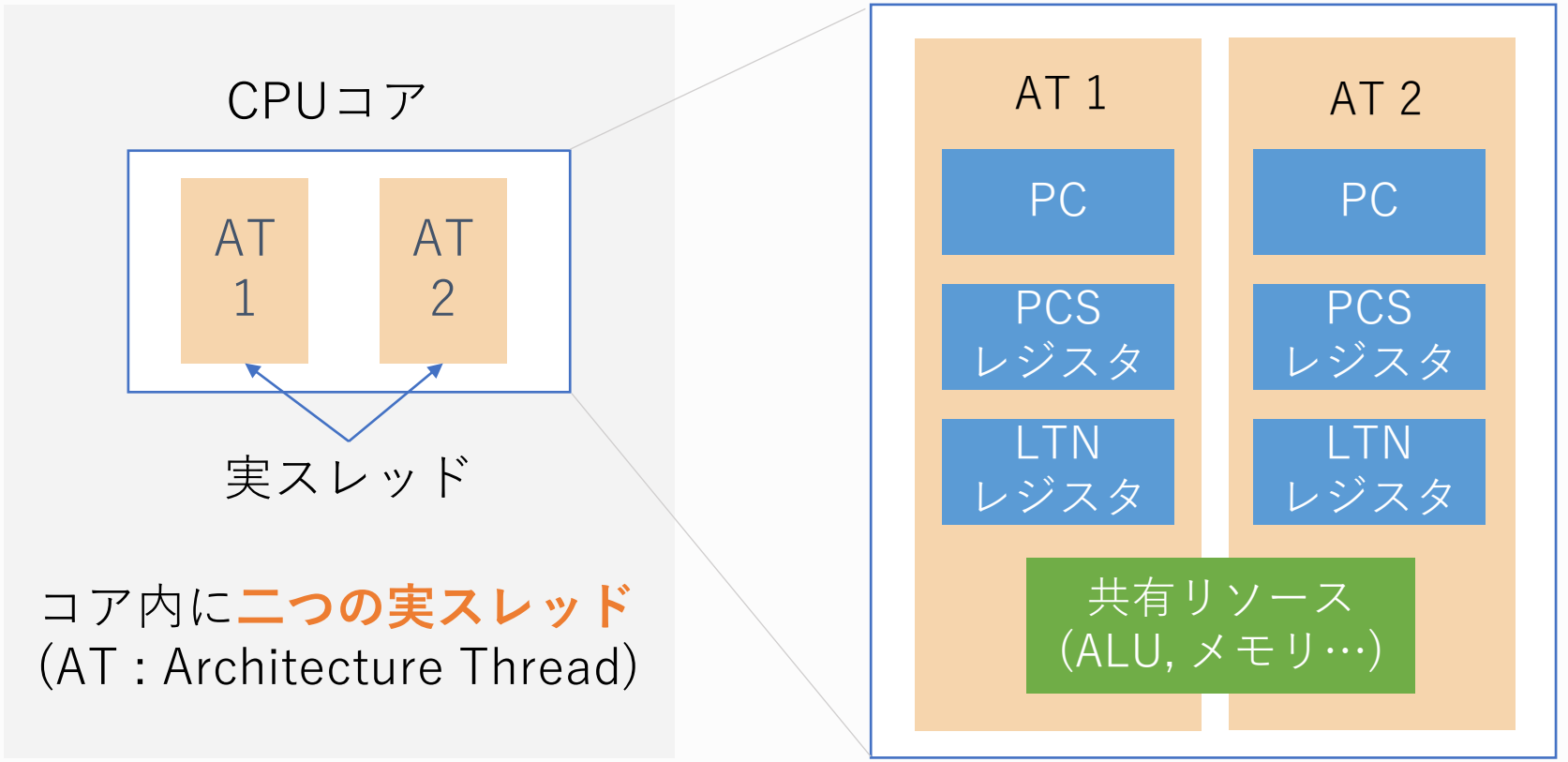
実スレッドの抽象化



スレッド制御では,
ハード側でLTN から ハードウェアスレッド番号の変換を行う

➡ **ソフト側はLTNの番号を指定するだけでよい**

各スレッドのリソース



各実スレッドが持つハードウェアリソース

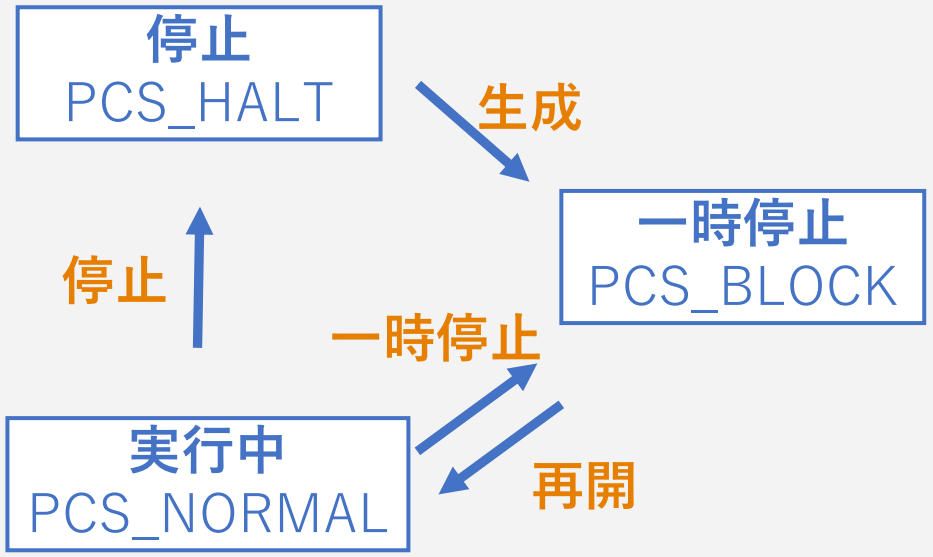
- プログラムカウンタ，**PCSレジスタ**，**LTNレジスタ**
- ALUやメモリなどの共有リソース

スレッド状態遷移と制御命令

Tokyo University of Agriculture and Technology

スレッドを実現するには,
スレッドの状態管理と制御命令が必要となる

スレッドの状態遷移



定義したスレッド制御命令

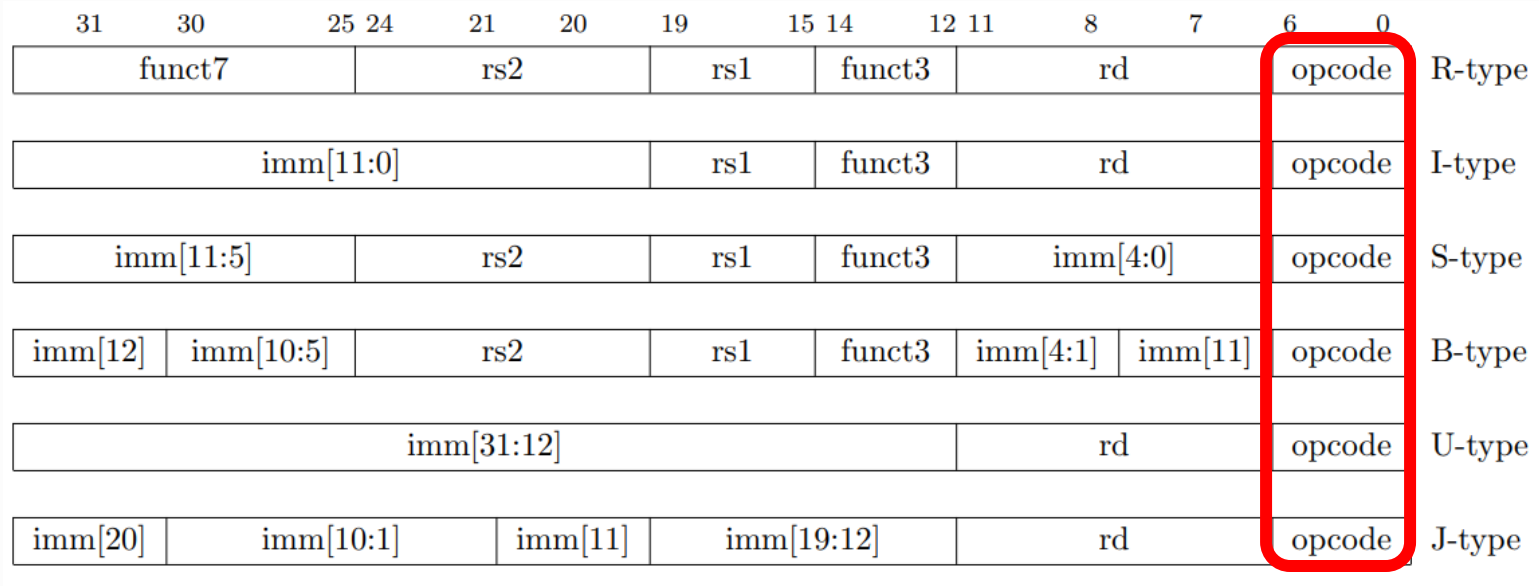
スレッド**生成**	PALLC
レジスタ**転送**	FWD
スレッド**一時停止**	PBLK
スレッド**再開**	PUBLK
スレッド**削除**	PDALL

スレッドを実現するため
5つの制御命令を定義した

スレッド制御命令の定義

Tokyo University of Agriculture and Technology

RISC-Vではopcodeにより，命令の形式を指定



RV32Iの命令フォーマット

カスタム命令を作るために新しいopcodeの定義が必要

スレッド制御命令の定義

Tokyo University of Agriculture and Technology

Opcode[6:2] のマップ

inst[4:2] / inst[6:5]	000	001	010	011	100	101	110	111 (> 32b)
00	LOAD	LOAD-FP	*custom-0*	MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	48b
01	STORE	STORE-FP	*custom-1*	AMO	OP	LUI	OP-32	64b
10	MADD	MSUB	NMSUB	NMADD	OP-FP	*reserved*	*custom-2/rv128*	48b
11	BRANCH	JALR	*reserved*	JAL	SYSTEM	*reserved*	*custom-3/rv128*	≥ 80b

Inst[6:0]

	6						0
custom-0	0	0	0	1	0	1	1
custom-1	0	1	0	1	0	1	1
custom-2	1	0	1	1	0	1	1
custom-3	1	1	1	1	0	1	1

この4つのOpcodeは
カスタム命令として使用可能

スレッド制御命令の定義

Tokyo University of Agriculture and Technology

custom-0	0	0	0	1	0	1	1

独自のスレッド制御命令をRISC-Vのopcode cutom-0を使用し，
以下のように定義した

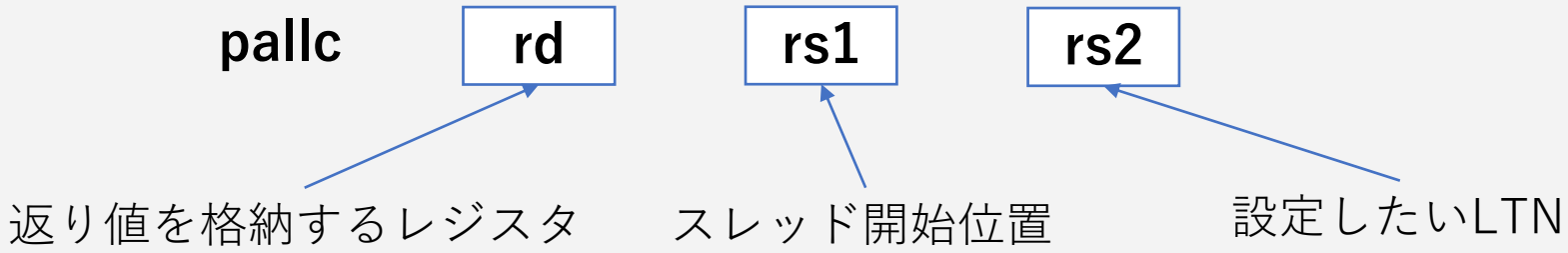
31 25	24 20	19 15	14 12	11 7	6 0	
-	rs2	rs1	000	rd	0001011	PALLC
-	-	rs1	001	rd	0001011	PDALL
-	-	rs1	010	rd	0001011	PBLK
-	-	rs1	011	rd	0001011	PUBLK
-	rs2	rs1	100	rd	0001011	FWD

funct3フィールドで各命令の判別

スレッド生成命令

Tokyo University of Agriculture and Technology

スレッド生成命令 PALLC



A) 成功した場合：

PC	＝ rs1
PCS レジスタ	＝ PCS_BLOCK
LTN レジスタ	＝ rs2
rd レジスタ	＝ 1（成功）

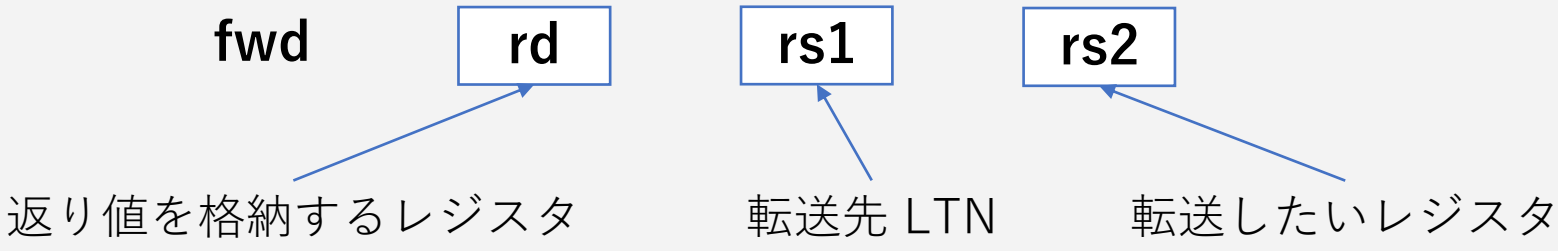
B) 失敗した場合：

PC	＝ --
PCS レジスタ	＝ --
LTN レジスタ	＝ --
rd レジスタ	＝ 0（失敗）

レジスタ転送命令

Tokyo University of Agriculture and Technology

レジスタ転送命令 FWD



A) 成功した場合：

PC	= --
PCS レジスタ	= --
LTN レジスタ	= --
rd レジスタ	= 1（成功）

B) 失敗した場合：

PC	= --
PCS レジスタ	= --
LTN レジスタ	= --
rd レジスタ	= 0（失敗）

スレッドブロック命令

Tokyo University of Agriculture and Technology

スレッドブロック命令 PBLK

A) 成功した場合：

PC	= --
PCS レジスタ	= PCS_BLOCK
LTN レジスタ	= --
rd レジスタ	= 1（成功）

B) 失敗した場合：

PC	= --
PCS レジスタ	= --
LTN レジスタ	= --
rd レジスタ	= 0（失敗）

スレッド再開命令

Tokyo University of Agriculture and Technology

スレッド再開命令 PUBLK

A) 成功した場合：

PC	= --
PCS レジスタ	= PCS_NORMAL
LTN レジスタ	= --
rd レジスタ	= 1（成功）

B) 失敗した場合：

PC	= --
PCS レジスタ	= --
LTN レジスタ	= --
rd レジスタ	= 0（失敗）

スレッド削除命令

Tokyo University of Agriculture and Technology

スレッド削除命令 PDALL

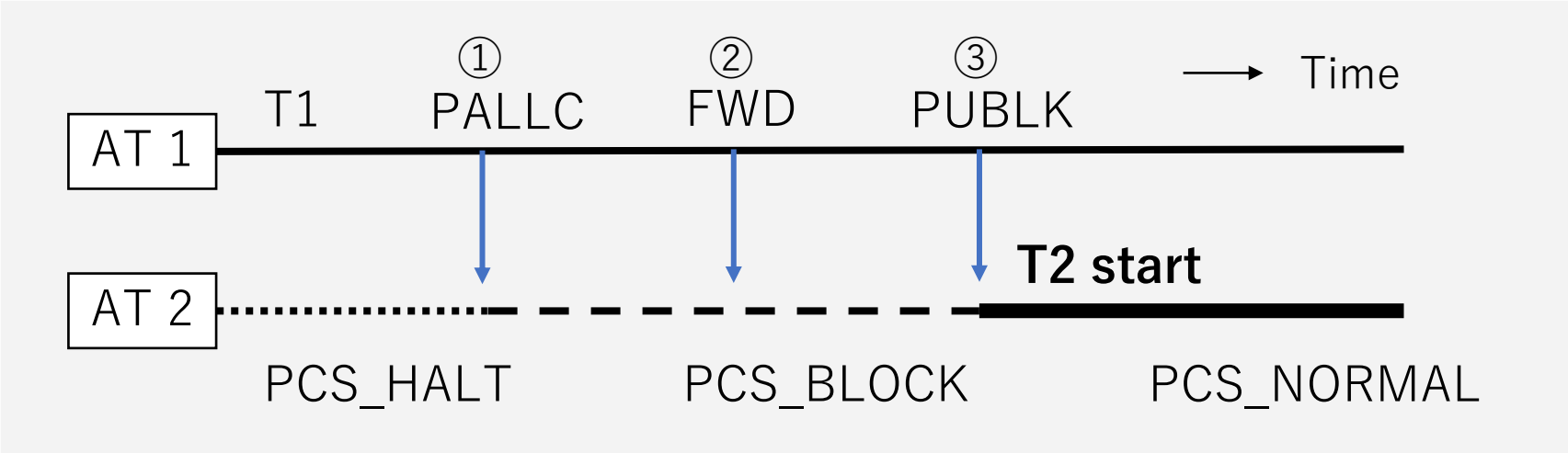
A) 成功した場合：

PC	= --
PCS レジスタ	= PCS_HALT
LTN レジスタ	= --
rd レジスタ	= 1（成功）

B) 失敗した場合：

PC	= --
PCS レジスタ	= --
LTN レジスタ	= --
rd レジスタ	= 0（失敗）

スレッド生成手順



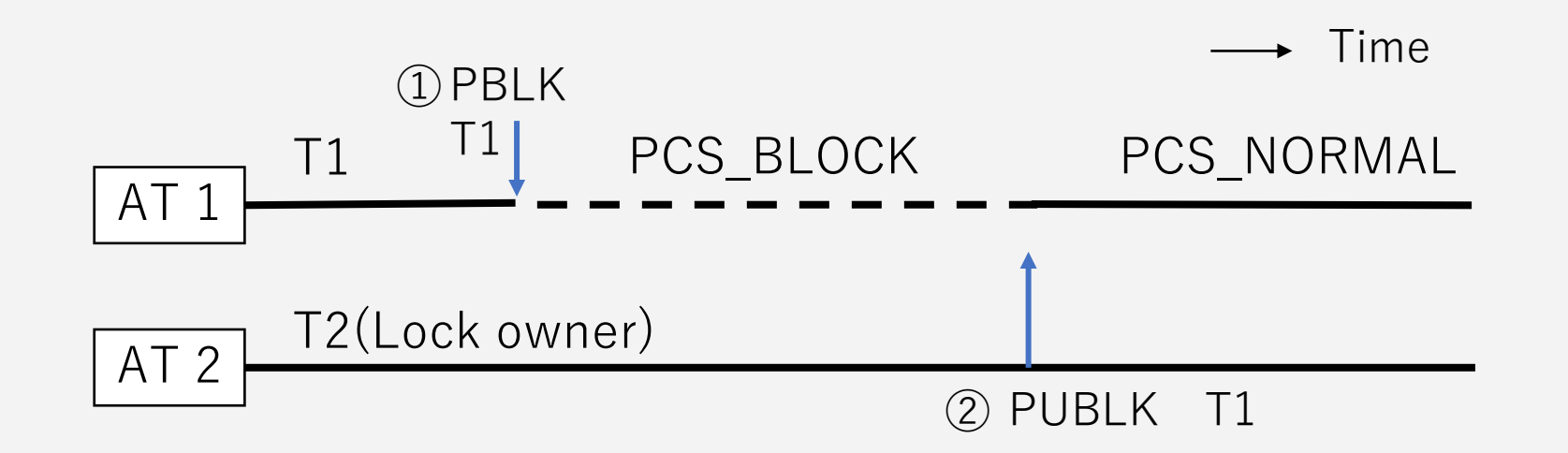
① T1がPALLC命令を発行して LTNを設定

② 1.が成功であれば，FWD命令でレジスタを初期化

③ 2.が成功であれば，PUBLK命令でT2のスレッドを開始

ユーザ命令のみでスレッドの生成可能

スレッド同期手順



① PBLK命令を発行し自スレッド（T1)をブロック状態にする

② 1.が成功であれば，T2がT1にPUBLK命令を発行し，T1を実行状態に戻す

ブロック状態ではループ処理ではなく実スレッドの処理を中断

➡ その間リソースを有効活用できる

評価方法

Tokyo University of Agriculture and Technology

評価環境

- FPGA　　Artix-7
- 論理合成ツール　　XILINX Vivado v2020.2(64-bit)
- RTLシミュレーション　　Verilator

性能評価

SMTを使用した場合と使用しない場合の比較

- RTL シミュレーションにより実行
- ベンチマーク　riscv-tests*のmedianフィルター
- IPC（クロック数あたりの命令実行数）を計測

*https://github.com/riscv/riscv-tests

評価結果（ハードウェアリソース）

Tokyo University of Agriculture and Technology

MIPS ➡ RISC-Vへのハードウェア増加と遅延

表1：CPU コアのリソース使用量

	LUT	FF	BRAM
MIPS	12885(20%)	6461(5%)	2.0
RISC-V	13036(20%)	6980(5%)	2.0
リソース比	**1.0117**	**1.0803**	1.0

リソース比

LUT　約1 % 増加
FF　　約8 % 増加

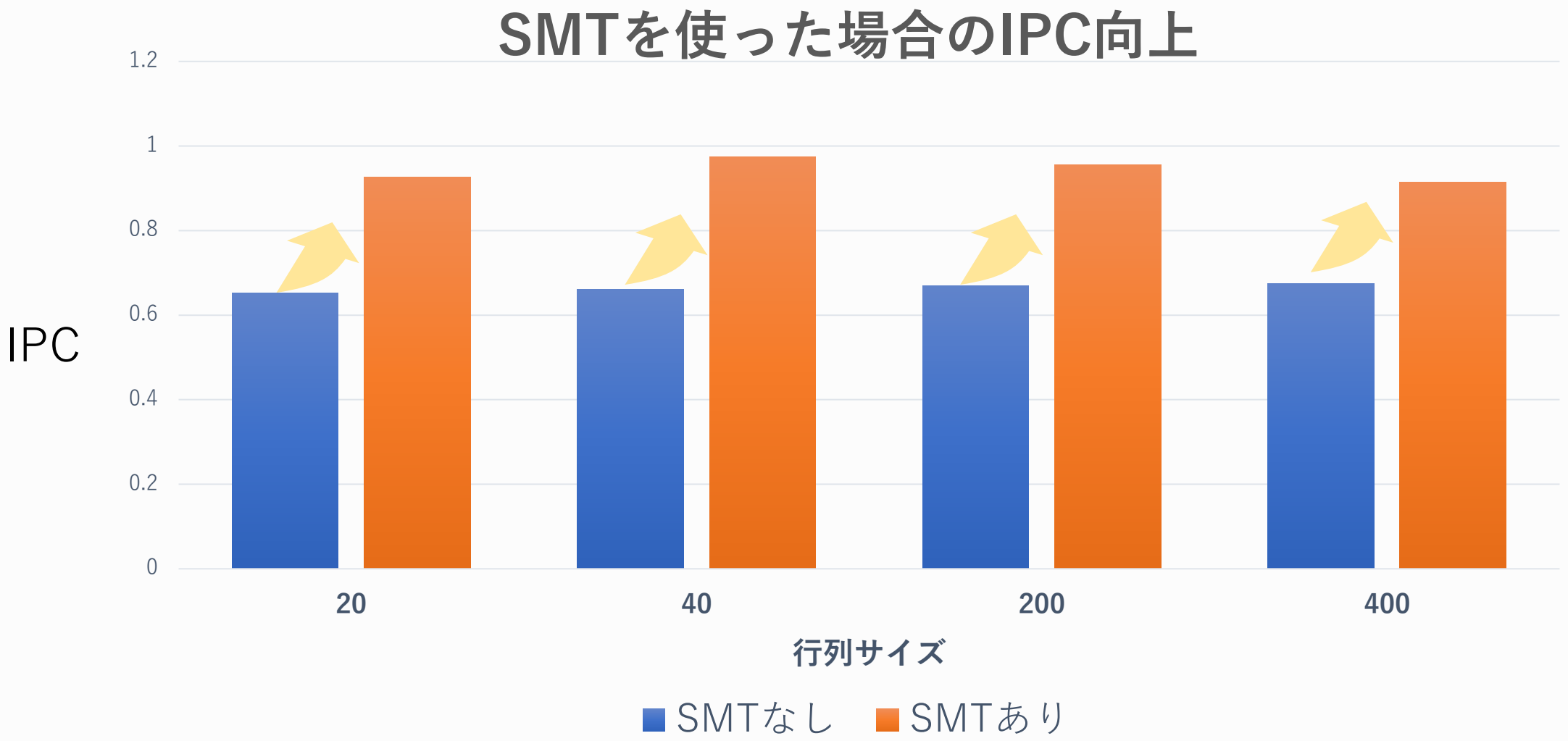
表2：データパス遅延と最大動作周波数

	データパス遅延	最大動作周波数
MIPS	9.926 ns	100.75 MHz
RISC-V	10.340 ns	**96.71 MHz**

最大動作周波数

約 4 MHz 低下

評価結果　(IPC)



最大 1.47 倍の IPC向上を確認

まとめ

Tokyo University of Agriculture and Technology

- MIPSのSMTと比較して**リソース増加と周波数の低下を抑えて移植成功**
- **SMTの活用により最大1.47倍のIPC向上達成**

RISC-VにおけるSMTプロセッサを実現
RISC-V上で**SMTの有効性も確認**

今後の展望

Tokyo University of Agriculture and Technology

- System VerilogやChiselで再設計しソース公開
  - オープンソースで誰でも拡張しやすいように
- SMTプロセッサに**ベクター拡張を組み合わせる**
  - ILP・TLPに加えてDLPを抽出
- 他プロセッサへの**SMT拡張実装**
  - 既存のUCBのBOOMプロセッサなどにSMT化をすることで検証