AXEのカスタムLSI設計サービス、ROS2通信ハードウェアIP提供

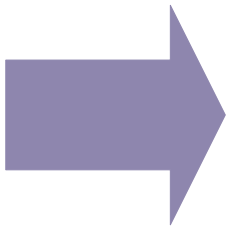
2025/DEC

AXE, Inc.

もう、CPUは(簡単には)速くならないよ

- ついに、微細化 限界
- 微細化 による
  - 高集積
  - 高クロック周波数
  - コア数 増加

は**終了**



- 専用 ロジック回路による
  - 高速化
  - 省 消費電力

を行うしか

背景:
カスタムLSI
設計&製造の
民主化

技術者 不足をStop!日本の半導体 産業 復興!

·OSSの開発ツールで、LSI 開発

- 無料ツールの使い手が増える → 技術者不足 解消!
  - 半導体 設計 技術者
  - 論理回路 設計 技術者

·半導体 工場は、「お高くない」ものもある

- 65nm とか、安くて かなりいい

·レガシーファブの活性化

産総研 直下のAIST Solutionsも、ロングテール半導体開発を推進

Open-Source は、ロングテール半導体開発の夢を見るか？

- Do Open-Source Dream of Long-Tail Semiconductors? –

11.17(金) 15:30-16:10 | 展示会場内　Room E

AIST Solutions
半導体事業 プロデューサー　岡村 淳一

AIST SOLUTIONS　2023/12/06　ONLY for 2023 Edge-Tech Conference

AISol の半導体事業の目標

国内の半導体アセット（チップ製造能力）を
本プロジェクトのプラットフォームに再整備することで
専用半導体の設計の参入障壁を下げ、
国内産業が専用半導体にて国際競争を勝ち抜く環境を提供する。

AIST SOLUTIONS

Open Source Silicon for Japan

AIST SOLUTIONS　2023/12/06　ONLY for 2023 Edge-Tech Conference

国内もLSI開発の民主化 推進

日本も、国の金で 施設を用意

ふくおかIST(公益財団法人 福岡県産業・科学技術振興財団)
福岡システムLSI総合開発センター
「システムＬＳＩ設計試作センター」

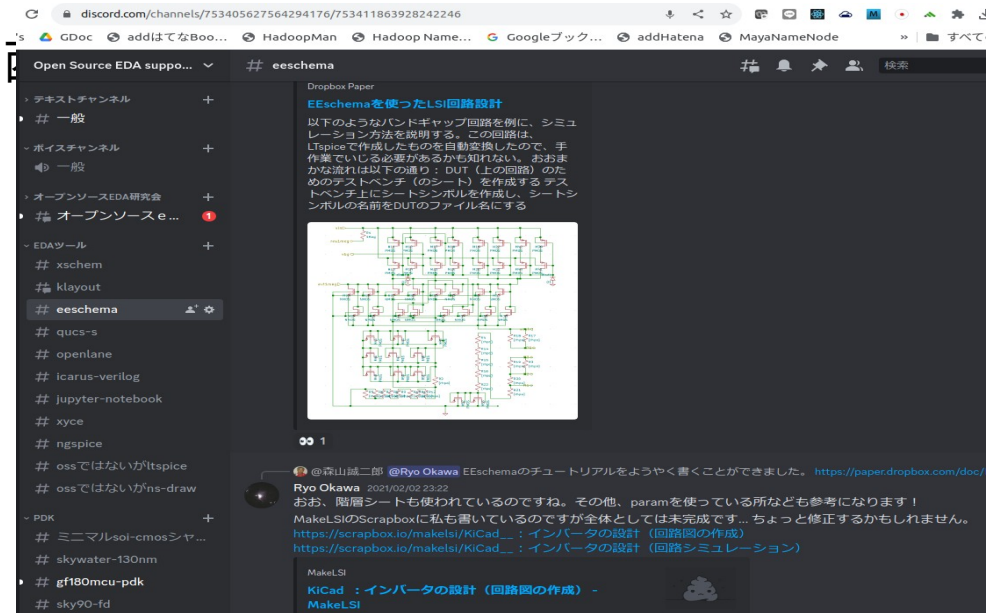
- http://www.ist.or.jp/lsi/pg04_02.html

- ベンチャー企業が半導体の設計ツールを無料(安価)で利用できる
- LSI設計、 少量 試作できる
  - 50～100万円 あれば、LSIの少量生産ができる仕組みがある

Open Source EDA Supporters (discode)

https://discord.com/channels/753405627564294176/932209975558832128

- OSS EDAをサポートする人たちの集まり
  - 親切で優しい
- ふくおかIST のOSS EDAサーバのユーザ会
  - サーバを維持するモチベーション
- オレオレ Open PDKを開発したり: 森山氏
- オレオレ Standard Cellライブラリを開発できるツールの開発者が居たり:
- オープンソースEDAフォーラム @福岡

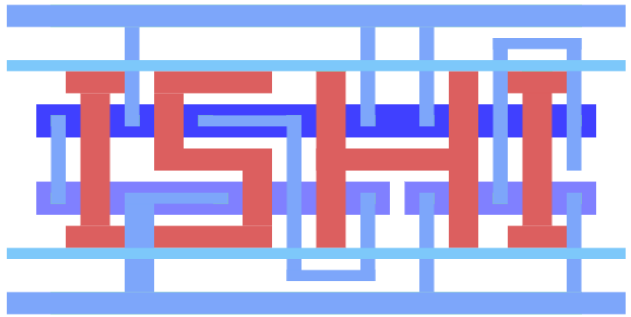


OpenSUSI,AIST Solの団体

- AIST Solutionsは、2024年4月5日付で一般社団法人OpenSUSIを設立しました。
- https://www.semiconportal.com/archive/editorial/industry/240423-aistsolutions.html より引用
- 産業技術総合研究所が日本版「半導体ICの民主化」プロジェクトを進めていることがわかった。産総研の総責任者である理事長の石村和彦氏（図1）は、産総研が開発した技術を社会実装して世の中の役に立たせようという石村改革を就任以来進めてきた。2023年設立したAIST Solutionsは社会実装の先頭部隊。2024年4月には半導体ICの開発に向けて「OpenSUSI」を設立した。これこそ半導体の民主化を狙った組織である。
- 4月に設立されたOpenSUSIが扱うのは、マーケットが豊富なレガシープロセスの半導体であり、ロングテール製品向けにASIC（アナログやデジタル回路を中心に設計されたハードウエアIC）やSoC（CPUなどのプロセッサを中心にソフトウエアも含めたIC）などを開発するためのサポートを行う。
- レガシープロセス:最先端の配線構造やトランジスタ構造をフルに3次元活用しない。

ISHI会

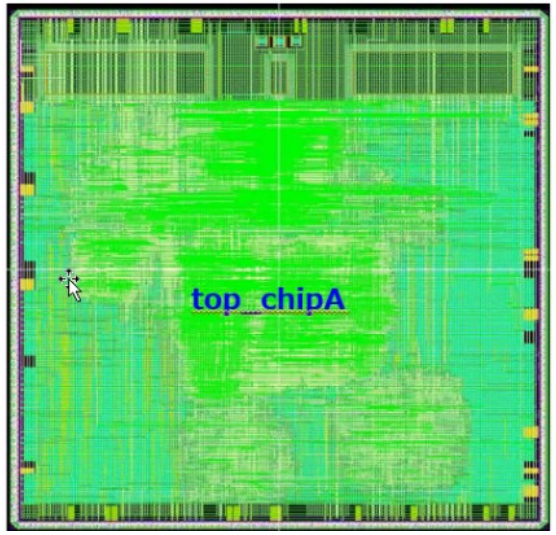
- https://ishi-kai.org/
- 「石 (チップ)」の会
- OSS EDAを実際に使用して、IPを開発する人たち
- 学識経験者/大学の先生も多数
- ワールドワイドなプロジェクトを実施中
  - 日本の担当は、オンチップ 安定化電源
- 初心者 指導も熱心に行っている
  - JASA RISC-V WGにも出席



AXEの完全オリジナルLSI ”Laxer AX1001”概要仕様

TSMC 90nmで製造,920万トランジスタ
LQFP 100pin
27/SEP/2024 京都で1stシリコン動作

俺SoC



機能	数	備考
CPU”松竹V”	1	RISC-V RV32Iに、4SIMD 8bit浮動小数点ベクトル機構、ハードウェアによるマルチタスク制御機構、多重分岐命令(特許取得済、Prolog,Lisp,JavaVMの実行を加速)と、それらを操作する命令を付加
ROS2通信機能	1	"ROS2rapper"という名称の、新開発のハードウェアによるROS2通信機能。CPUから初期化することで、自動的に通信を行う
RAM	1	プログラム・コードRAM 64KBytes,データRAM 64KBytes
SIO(UART)	3	SIO0(UART0)は、bootloadingとデバッグ・コンソールに使用される。UART1,2はユーザが自由に使用できる
GPIO	38	18本はPullUP指定可能。20本はPullDown指定可能。2本は、SIO1と兼用。2本は、SIO2と兼用。18本は外部イベント源として使用可能。(※外部イベントとは、一般CPUにおける割込のようなもの。外部イベントが発生すると、イベントについて待機しているスレッドが起床する)
AWG	1	16bit出力,1ch。出力ピンは、PWMと兼用。PWMと排他的に使用
PWM	8	出力ピンは、AWGと兼用。AWGと排他的に使用
Ether I/F	1	NICはオンチップ。PHY I/Fを持つ (PHYは外付け)

松竹V :RECONF講演賞 企業部門@2025年1月研究会 受賞

https://www.ieice.org/iss/reconf/top/?page_id=612

Laxer2 チップ 絶賛 開発中

- およそ Laxer1.5 と同等の機能&性能
- ハードウェアによるマルチ・タスク,排他/同期機構
- ROS2rapper ハードウェア搭載 (Ether MAC)
- 内蔵RAM 512KBytes
- 外付けS-RAM (Max:4MBytes)
- CPU Clock: 320MHz (予定)

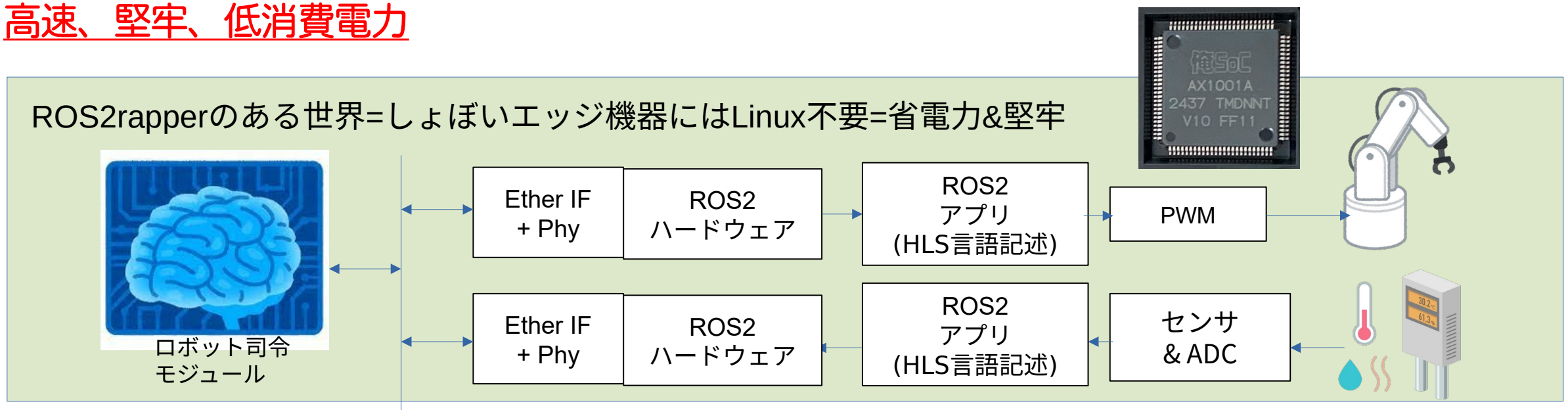
- 組込みマイクロ・コントローラとして、十分であろう

ROS2プロトコルを完全ハードウェア化した…

AXE製、”ROS2rapper”

“ROS2rapper” IPは、
オープンソース・ハードウェアで配布中.
https://github.com/AXE-jp/ros2rapper

- CPU無しで、ロボットの部品モジュールができる
  - センサとROS2プロトコルHWだけで、センサ・モジュール
  - PWMとROS2プロトコルHWだけで、アクチュエータ・モジュール
  - アプリケーションはC言語で書いておけば、すぐハードウェア論理に合成
- ロボット部品が、ゴミのようなLSIでできる ← CPU不要
- 高速、堅牢、低消費電力



オレ達のCPU「松竹V(しょうちくぶい)」の排他制御

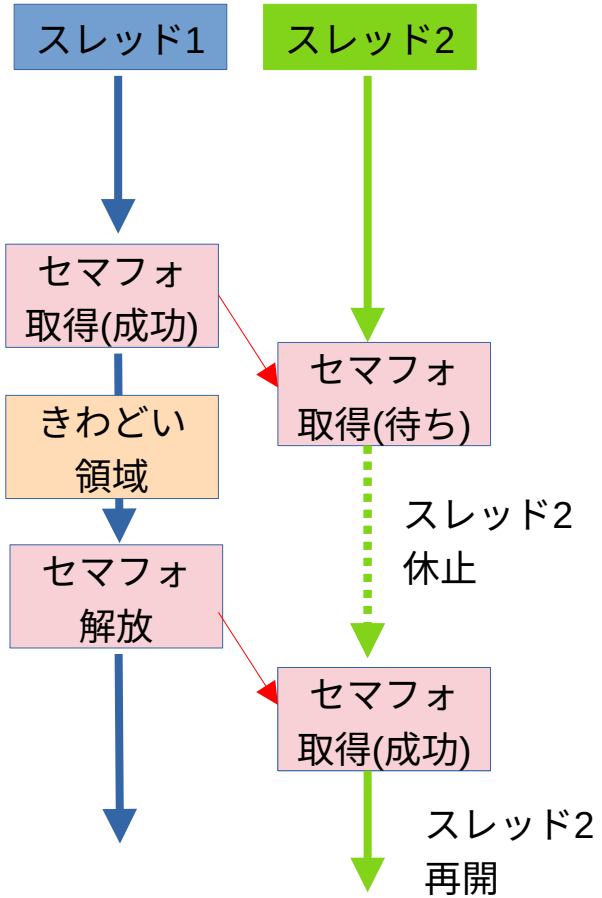
“俺SoC”,とことんOS無し

- **省電力、省メモリ、堅牢かつ高速**
- **ロボットの部品モジュールがローコストで簡単に作れる**

マルチタスクだが、
OSプログラム・コード
OSワーキング・エリア
不要!

ハードウェア・マルチスレッド機構

- OSソフトウェア一切なしで、スレッド切り替え
- ハードウェア・セマフォで排他/同期(重要)
- LR/SC(RISC-Vの排他制御プリミティブ)もある
- 外部ピンからの入力で、スレッド起床(ハードウェアのみで)
  - 割り込みなし(割り込み相当の処理は、専用スレッドで)
- ラウンドロビン・スケジューリング

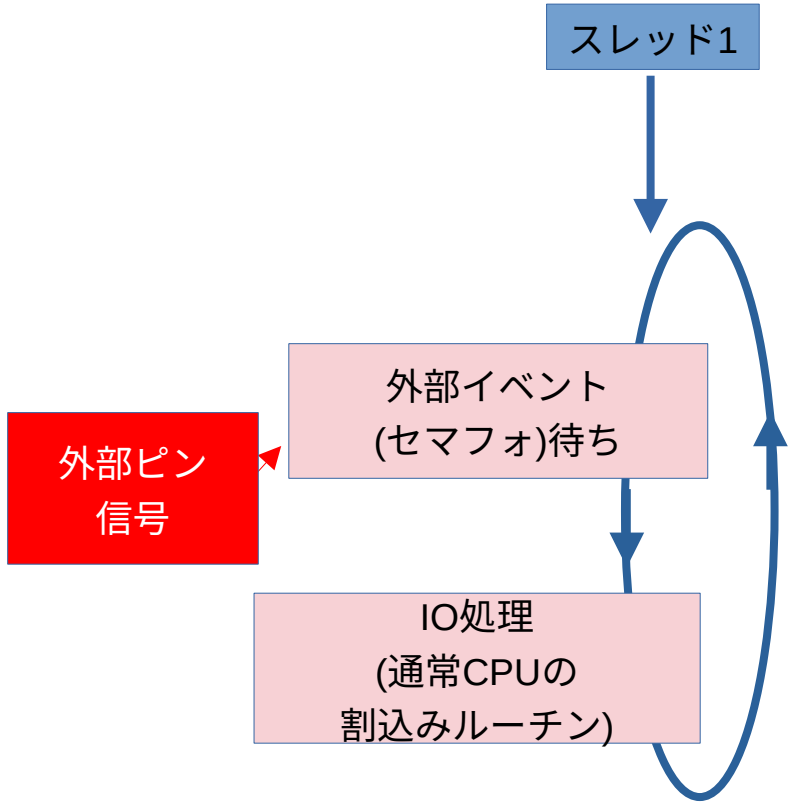


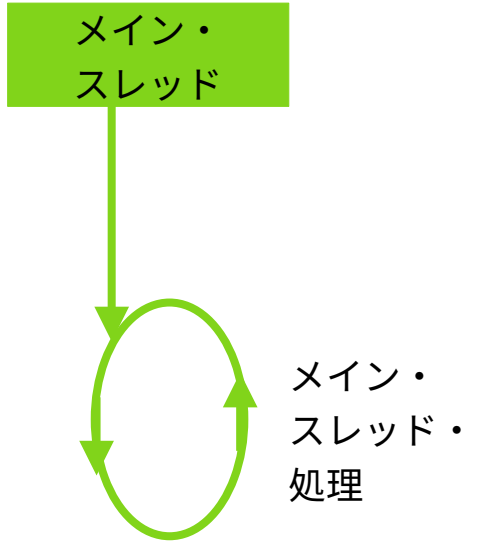
オレ達のCPU「松竹V(しょうちくぶい)」の外部イベント(割込み相当)

マルチタスクだが、
OSプログラム・コード
OSワーキング・エリア
不要!

ハードウェア・マルチスレッド機構

- OSソフトウェア一切なしで、スレッド切り替え
- 外部ピンからの入力で、スレッド起床(ハードウェアのみで)
  - 割り込みなし
- 割り込み相当の処理は、専用スレッドで。
- 割込み起動より、高速
  - レジスタ退避などしないから



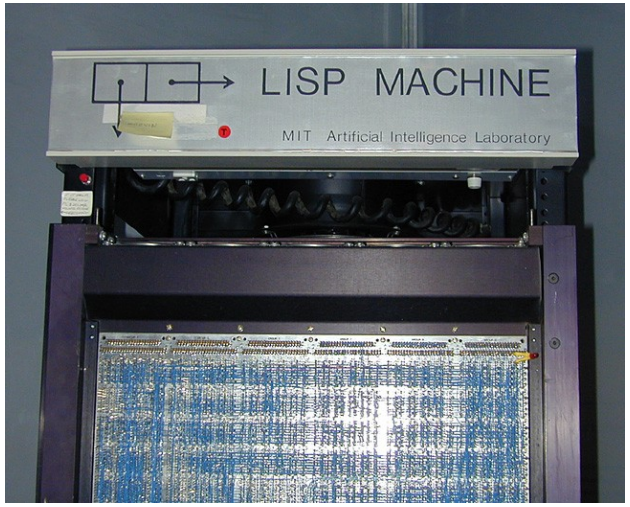


スレッド操作、セマフォ操作命令

6.5.7. カスタム命令

ニーモニック	命令フォーマット	**opcode**	**funct3**	**funct7**	**imm**
task_start	R-type	01010 (custom-1)	000	0000000	-
task_terminate	R-type	01010 (custom-1)	000	0000001	-
task_getid	R-type	01010 (custom-1)	000	0000010	-
task_read_xreg	R-type	01010 (custom-1)	000	0000011	-
task_write_xreg	R-type	01010 (custom-1)	000	0000100	-
task_sleep	R-type	01010 (custom-1)	000	0000101	-
task_yield	R-type	01010 (custom-1)	000	0000110	-
sem_init	R-type	01010 (custom-1)	000	0000111	-
sem_signal	R-type	01010 (custom-1)	000	0001000	-
sem_wait	R-type	01010 (custom-1)	000	0001001	-
task_setts	R-type	01010 (custom-1)	000	0001010	-

新命令による
Python,Ruby,
論理推論AI(Prolog,Lisp)
ネイティブ・コード高速化



ヒトは行列計算のみで生くるものにあらず

by 俺

MIT Lisp MachineやPSI(ICOTのPrologマシン,1986年)を超えたい

【特許第7421850号】 登録日【2024年1月17日】
(東京エレクトロンと竹岡の共同特許)
単純でごくフツー
1970年代末期～80年代(高級言語マシン、Lispマシン,Prologマシン全盛)に
取られていても不思議ではないような機構

論理推論は古くないんだぞ!

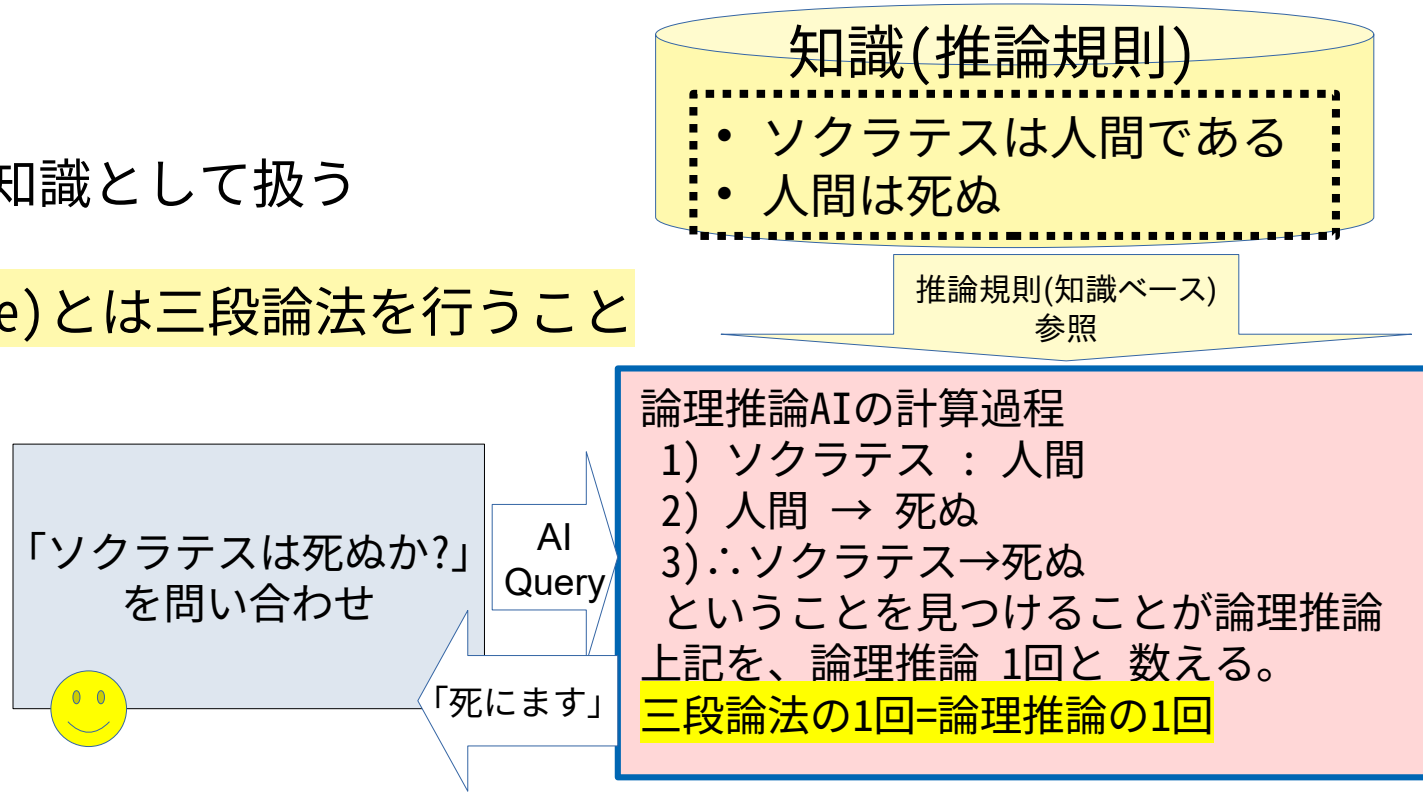
- 故障診断など、精密な出力が必要な分野では、現在も活用中

記号論理推論AIとは

- 知識ベースをもとに、記号論理推論を行う
  - 推論規則も、知識の一種
  - 推論規則も、一般的な知識も、すべて、知識として扱う

- 記号論理推論(Symbolic Logical Inference)とは三段論法を行うこと

- 推論規則は、人間が書く
  - LLMの助けを借りる
- 論理推論の特徴
  - 精密
  - 問題の原因を明らかにできる
  - 問題の解消は、推論規則の追加/修正
  - 説明のできるAI



多重分岐(レジスタ間接レジスタ指定) 新命令

【特許第7421850号】
登録日【2024年1月17日】

- Python,ruby(native code),Prolog,Lispが速くなる
- branch_reg_indirect IREG

  pc ← pc + xreg[IREG] ; xreg[n]は、n番の汎用レジスタ

  ※Branch_and_Link機能付き, callにもなる

- jump_reg_indirect IREG1,REG2

  pc ← xreg[IREG1] + REG2

  ※Branch_and_Link機能付き, JALにもなる

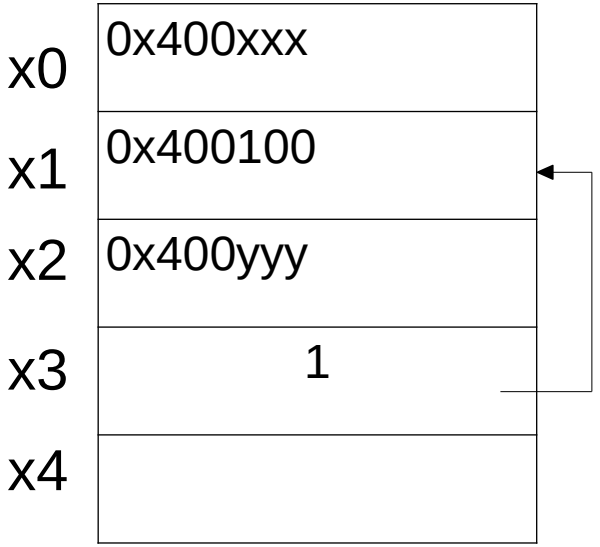
- mov_reg_regindirect dst,ireg

  dst ← xreg[ireg]

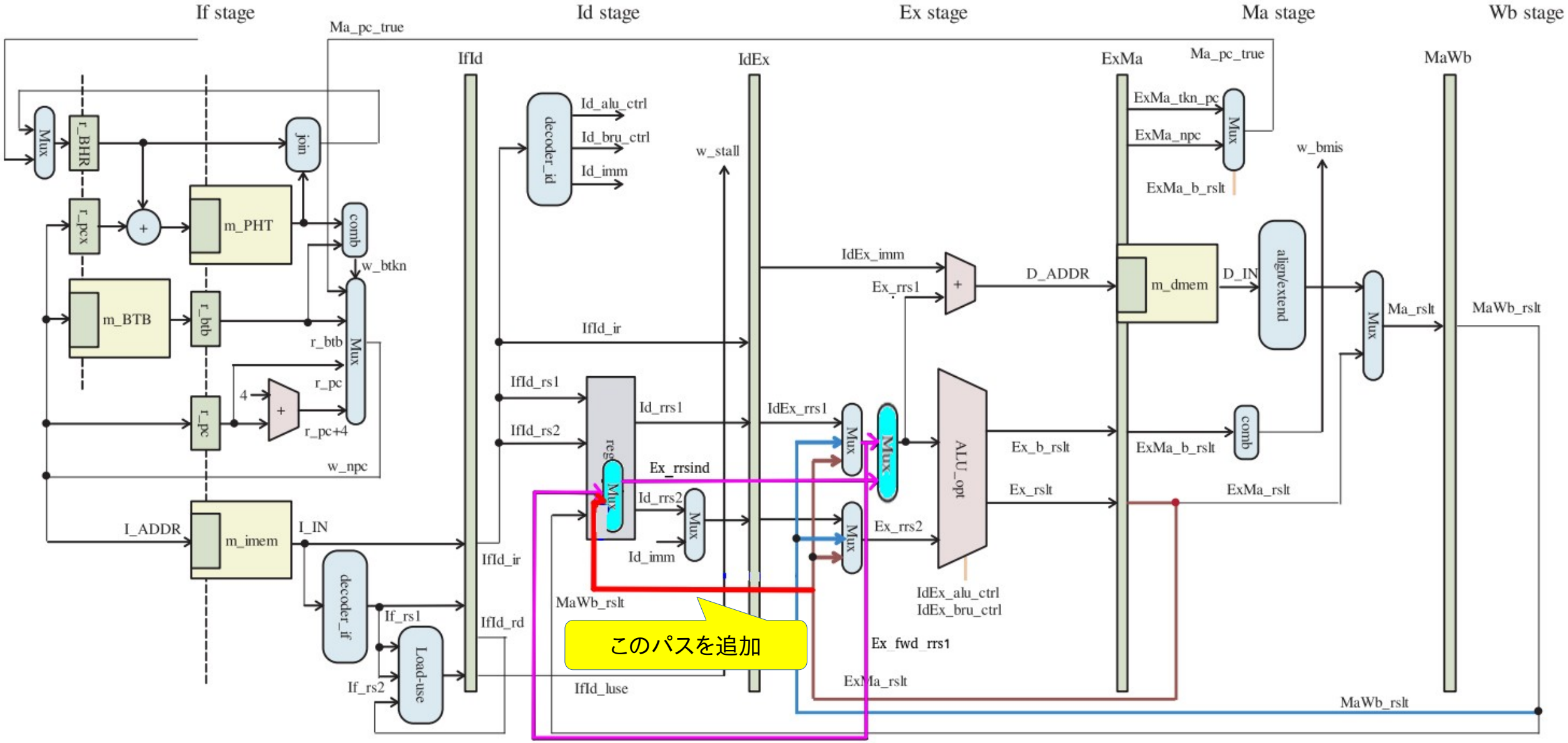
- mov_regindirect_reg ireg,src

  xreg[ireg]←src

IREGがx3(内容=1)のとき
X1が指される。
参照の場合、実効値0x400100



mov_reg_regindirect, branch_reg_indirect 追加後のパイプライン



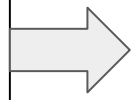
2022/JUL/15 a

俺SoC LSIでは、追加したパスは、クリティカル・パスにはならない

GNU-PrologのWAM(抽象マシン)コード

- GNU-Prolog の WAM コードでは、switch_on_term で 1つめの arity の型
  変数/アトム/整数/リスト/構造体
によって分岐する
- 分岐先で各arityの値をチェックする

```
foo(1, a, X) :- bar(1, X).
foo(a, [c,d], X) :- bar(2, X).
foo([a,b], c(d), X) :- bar(3, X).
foo(a(b), 1, X) :- bar(4, X).
```



```
predicate(foo/3,1,static,private,monofile,global,[
    switch_on_term(1,4,2,6,8),

label(1),
    try_me_else(3),

label(2),
    get_integer(1,0),
    get_atom(a,1),
    put_value(x(2),1),
    put_integer(1,0),
    execute(bar/2),

label(3),
    retry_me_else(5),

label(4),
    get_atom(a,0),
    get_list(1),
    unify_atom(c),
    unify_list,
    unify_atom(d),
    unify_nil,
    put_value(x(2),1),
    put_integer(2,0),
    execute(bar/2),

label(5),
    retry_me_else(7),

label(6),
    get_list(0),
    unify_atom(a),
    unify_list,
    unify_atom(b),
    unify_nil,
(以下略)
```

PI_Switch_On_Term()の内容

ランタイム関数PI_Switch_On_Term() では、1つめのarityであるA(0)のタグ(下位3ビット)によって、次に実行する分岐先を返すようになっている

```
CodePtr FC
PI_Switch_On_Term(CodePtr c_var,
          CodePtr c_atm, CodePtr c_int,
          CodePtr c_lst, CodePtr c_stc)
{
 WamWord word, tag_mask;
 CodePtr codep;


 DEREF(A(0), word, tag_mask);
 A(0) = word;


 if (tag_mask == TAG_INT_MASK)
   codep = c_int;
 else if (tag_mask == TAG_ATM_MASK)
   codep = c_atm;
 else if (tag_mask == TAG_LST_MASK)
   codep = c_lst;
 else if (tag_mask == TAG_STC_MASK)
   codep = c_stc;
 else                          /* REF or FDV */
   codep = c_var;


 return (codep) ? codep : ALTB(B);
}
```

RISC-V 64bit Gnu Prolog コンパイルド・バイナリとその高速化

gplcでコンパイルした native codeをobjdump -d の一部

```
00000000000096b8 <X0_ap__a3>:
    96b8: 00000517      auipc a0,0x0
    96bc: 02050513      addi a0,a0,32 # 96d8 <X0_ap__a3+0x20>
    96c0: 00000597      auipc a1,0x0
    96c4: 02458593      addi a1,a1,36 # 96e4 <X0_ap__a3+0x2c>
    96c8: 00000617      auipc a2,0x0
    96cc: 05460613      addi a2,a2,84 # 971c <X0_ap__a3+0x64>
    96d0: 014900ef      jal ra,996e4 <Pl_Switch_On_Term_Var_Atm_Lst>
    96d4: 00050067      jr a0
    96d8: 00000517      auipc a0,0x0
    96dc: 04050513      addi a0,a0,64 # 9718 <X0_ap__a3+0x60>
    976c: 00ab3823      sd a0,16(s6)
    9770: f49ff06f      j 96b8 <X0_ap__a3>
    9774: 00000013      nop
```

多重分岐の飛び先アドレスを
レジスタにセット

<u>レジスタ間接によるレジスタ指定ができる新命令を追加</u>
<u>特にジャンプ命令</u>
<u>ロード命令、ストア命令</u>
例えば、WAMコード switch_on_term をアセンブリ・コードにする場合に、型を示すタグ(3bit)によって、レジスタ間接によって指定されたレジスタの値(アドレス)にジャンプすると高速になる
すなわち

```
jal   ra,Pl_Switch_On_Term@PLT ;サブルーチン・コール
                               なので、とても遅い
jr    a0
```

を、下記1命令で置き換えて、高速化
branch_reg_indirect IREG ; pc ← pc + xreg[IREG]

この2行の部分を、
あるレジスタ中の値で指定された、
別なレジスタ中の値を番地として、
jumpするようなコードに変更する
(コンパイラが新命令を使用するように変更)
新命令「branch_reg_indirect %rax」
で置き換えて、高速化

Lisp, Haskellなど高級な言語は同様に高速化可能だろう
(動的に型チェックをする言語の実行系)

```
%%% Prologソース
ap([],Y,Y).
ap([A|X],Y,[A|Z]) :- ap(X,Y,Z).
%% ap(X,Y,[a,b,c]).
```

加速命令 使用の実測値

実行時間

テスト・プログラム	新命令 不使用(ms) N	加速命令 使用(ms) U	加速率(%) (N-U)/N*100	備考
queens.pl (16queen) (10回繰り返し)	61,638	58,930	4.39	みんな大好き 8queen
cal.pl (10回繰り返し)	11,026	10,827	1.80	万年カレンダ 数値計算
ham.pl (10回繰り返し)	52,591	52,324	0.51	???

- とてもシンプルな機構(線を引っ張るだけ)
- 4%以上の加速が得られているので、十分 効いている
- コスパ良し

- gplc(GNU Prologコンパイラ)にオプション --emit-regind を新設
- 測定環境
  - FPGA(Nexys Video)に松竹V CPUコア, 命令コード・メモリ=256KB, データ・メモリ=1MB
  - Clock: 50MHz
  - ベア・メタル用runtimeルーチンを用意

付録

ジャンプ命令に基づくパイプライン処理を制御するプロセッサ及びプログラム

(東京エレクトロンと竹岡の共同特許)

特許【第7506718号】

登録日　【2024年6月18日】

ジャンプ命令に基づくパイプライン処理を制御するプロセッサ及びプログラム

(東京エレクトロンと竹岡の共同特許)

【第7506718号】

登録日　【2024年6月18日】

特　許　証
(CERTIFICATE OF PATENT)

特許第７５０６７１８号
(PATENT NUMBER)

発明の名称
(TITLE OF THE INVENTION)　ジャンプ命令に基づくパイプライン処理を制御するプロセッサ及びプログラム

特許権者
(PATENTEE)　東京都千代田区神田東松下町１９番地１　ステージファースト神田１１　４０４号

たけおかラボ株式会社

（その他別紙記載）

発明者
(INVENTOR)　竹岡　尚三

（その他別紙記載）

出願番号
(APPLICATION NUMBER)　特願２０２２－１２５９１３

出願日
(FILING DATE)　令和　４年　８月　５日 (August 5,2022)

登録日
(REGISTRATION DATE)　令和　６年　６月１８日 (June 18,2024)

この発明は、特許するものと確定し、特許原簿に登録されたことを証する。

(THIS IS TO CERTIFY THAT THE PATENT IS REGISTERED ON THE REGISTER OF THE JAPAN PATENT OFFICE.)

令和　６年　６月１８日 (June 18,2024)

特許庁長官
(COMMISSIONER, JAPAN PATENT OFFICE)

濱野幸一

特　許　証　（続葉　１）
(CERTIFICATE OF PATENT)

特許第７５０６７１８号 (PATENT NUMBER)

特願２０２２－１２５９１３ (APPLICATION NUMBER)

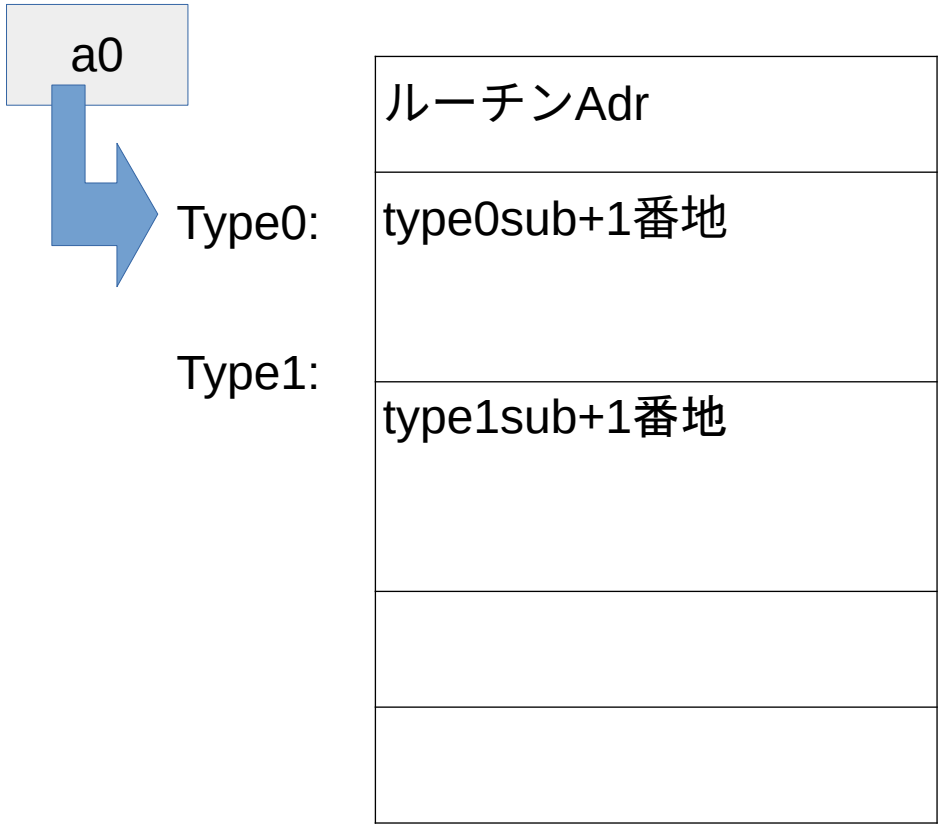
特許権者
(PATENTEE)　東京都港区赤坂五丁目３番１号
東京エレクトロン株式会社

発明者
(INVENTOR)　木下　喜夫

［以下余白］

ごく普通のディスパッチ方法(既存ソフトウェア)

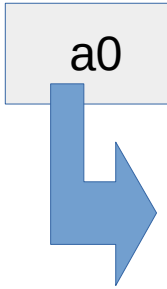
ld a1,[a0] ;a0レジスタが指している番地から、値(ジャンプ先)をa1レジスタにロード
jump a1 ;a1レジスタの内容アドレスへジャンプ (プログラム・カウンタへa1の内容を転送)



- **a0レジスタの内容で、ディスパッチ・テーブルを引き**
- **そのテーブルの内容(ルーチンの先頭の命令)を CPU コアの各 部分にセットする**

特許出願アイデア1:ディスパッチ高速化1

- fast1_multi_branch a0 命令を考案
- ディスパッチ・テーブルに、分岐先アドレスと共に、分岐先(次に実行する)命令も置いておく
  - 命令フェッチが高速になる



Type0:

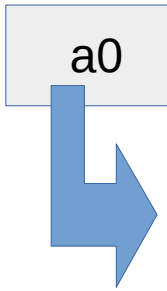
Type1:

ルーチンAdr	先頭命令
type0sub+1番地	先頭命令
type1sub+1番地	先頭命令
	:
	:

- **a0レジスタの内容で、ディスパッチ・テーブルを引き**
- **そのテーブルの内容(ルーチンの先頭の命令)を CPU コアの各 部分にセットする**
- **このテーブルは、CPUコアの内部にもつ。比較的 小容量で高速**

特許出願アイデア2:高速ディスパッチ命令2

- fast2_multi_branch a0　命令を考案
- ディスパッチ・テーブルに、分岐先アドレスと共に、
  - 分岐先(次に実行する)命令のデコード済み情報(CPUコア内情報)と
  - 分岐先の次の命令(次の次に実行する)命令も置いておく
- ※最初の命令のフェッチとデコードを省略する。2番目の命令フェッチを省略する



	ルーチンAdr	2番目命令	先頭命令のデコード済みコア内部情報
Type0:	type0sub+2番地	2番目命令	ShiftRight a0,a0,3をデコードした情報ビット列
Type1:			
	xxx+2番地	2番目命令	tag_add a0,a0,10をデコードした情報ビット列
		:	
		:	

- **a0レジスタの内容で、ディスパッチ・テーブルを引き**
- **そのテーブルの内容を CPU コアの各 部分にセットする**
- **このテーブルは、CPUコアの内部にもつ。比較的 小容量で高速**

以上

AIもやってます 「ごまめ®」
健康のためのAI

お問い合わせ
https://www.axe.bz/
mail to: eigyo@axe.bz

付録

オレ達のCPU「松竹V(しょうちくぶい)」

機械学習AI 加速用 ベクトル計算ユニット

エッジデバイスでも
AI処理を!

・8bit float,4SIMD,ベクトル・パイプライン
・動的クロック切り替え機構で500KHz～320MHzで、動作
※最高 480MHz (動的クロック切り替え非対応)
・Vector ExtensionのImplementation-defined Constant Parameters(実装固有パラメータ)
ELEN:32 ,VLEN: 1024

・ベクトルレジスタ
1024ビットのベクトルレジスタ×32個
ベクトルレジスタはすべてのタスクで共有(実体は1つ)

機械学習AIの推論に最適な8bit浮動小数点演算をベクトル処理

- エッジ機器内での、AI処理
↓
- データ通信量の削減

・分散処理による、全体の負荷の分散

32ビットのスカラ浮動小数点数レジスタ(Fレジスタ): 32個も備える

8bit浮動小数点 ベクトル演算命令

- AX1001の8bit浮動小数点数のフォーマット
  - 仮数部:3bit, 符号:1bit 指数部:4bit, 指数バイアス=7

6.7.8. サポートするCSR

Vector Extensionで規定される、以下のCSRをサポートしている。

CSRアドレス	アクセス	名前	概要
0xC20	R	vl	Vector length
0xC21	R	vtype	Vector data type register
0xC22	R	vlenb	VLEN/8 (vector register length in bytes)

6.7.6. サポートする命令

- V拡張
  - vsetvli
  - vsetivli
  - vsetvl
  - vle.v
  - vse.v
  - vlse.v
  - vsse.v
  - vlm.v
  - vsm.v
  - vadd.{vv, vx, vi}
  - vsub.{vv, vx}
  - vand.{vv, vx, vi}
  - vor.{vv, vx, vi}
  - vxor.{vv, vx, vi}
  - vsll.{vv, vx, vi}
  - vsrl.{vv, vx, vi}
  - vsra.{vv, vx, vi}
  - vfadd.{vv, vf}
  - vfsub.{vv, vf}
  - vfmul.{vv, vf}
- F拡張
  - flw

オレ達のCPU「松竹V(しょうちくぶい)」

- **省電力、省メモリ、堅牢かつ高速**
- **ロボットの部品モジュールがローコストで簡単に作れる**

論理推論 AI加速 機構をRISC-Vコアに追加

- 特許取得済み
- 第7506718号,2024年6月18日（東京エレクトロンと共同特許)

- GnuPrologのコンパイルド・バイナリを加速
  - 詳細は、付録参照のこと

エッジデバイスでも
大脳的処理を!

その他のカスタム命令

6.4.2. カスタム命令

ニーモニック	命令フォーマット	opcode	funct3	funct7	imm
mov_reg_regindirect	I-type	00010 (custom-0)	000	-	Don't care
mov_regindirect_reg	I-type	00010 (custom-0)	001	-	Don't care
branch_reg_indirect	I-type	00010 (custom-0)	010	-	Don't care
jump_reg_indirect	R-type	00010 (custom-0)	011	0000000	Don't care

6.4.2.3. branch_reg_indirect

プログラムカウンタ相対で分岐を行う。プログラムカウンタに加算する値をソースオペランドをレジスタ間接で指定する。後続命令のアドレス（pc+4）をレジスタrdに格納する。

本命令は、次の操作を行う。

```
x[rd] <- pc + 4
pc <- pc + x[x[rs1]]
```

6.4.2.4. jump_reg_indirect

分岐を行う。分岐先のベースアドレスをレジスタ間接で指定する。後続命令のアドレス（pc+4）をレジスタrdに格納する。

本命令は、次の操作を行う。

```
rd <- pc + 4
pc <- x[x[rs1]] + x[rs2]
```

Laxer AX1001 消費電力概要

	テストプログラム名	CPUクロック周波数・消費電力(mW)					
-	-	320MHz			1.79MHz		
-	-	実測(1.0v)mA	実測(3.3v)mA	1.0V+3.3v計(mW)	実測(1.0v)mA	実測(3.3v)mA	1.0V+3.3v計(mW)
	メモリテスト	177.0	5.0	193.5	10.0	0.0	10.0
	有意義なことはしていない	113.0	11.0	149.3	30	10	63.0

参考	※実際には、DC-DCコンバータなどで損失が出るので、仮の理想値	電池容量(mW換算)	320MHz(時間)	1.79MHz(時間)
	単1形アルカリ電池容量 約10,000 mAh/1本, 3本4.5v使用	135,000	697.67	13,500.00
	単3形アルカリ電池 容量約1,000～2,900mAh/1本, 3本4.5v使用	39,150	202.33	3,915.00
	コイン電池CR2450 (3V) 620mAh, 2個(6V)使用 (※最大電流30mA/1個)	7,440	(38.45 ※電流が足りない)	744.00

付録
レジスタ間接レジスタ指定

参考:高速化対象: GNU-Prologの80x86(64bit)版オブジェクト(アセンブリ・コード)

この2行の部分を、
あるレジスタ中の値で指定された、
別なレジスタ中の値を番地として、
jumpするようなコードに変更する
(コンパイラが新命令を使用するように変更)
新命令「 branch_reg_indirect %rax 」
で置き換えて、高速化

```
X0_foo__a3:
        movq    Lpred1_1@GOTPCREL(%rip),%rdi
        movq    Lpred1_4@GOTPCREL(%rip),%rsi
        movq    Lpred1_2@GOTPCREL(%rip),%rdx
        movq    Lpred1_6@GOTPCREL(%rip),%rcx
        movq    Lpred1_8@GOTPCREL(%rip),%r8
        call    Pl_Switch_On_Term@PLT
        jmp     *%rax

Lpred1_1:
        movq    Lpred1_3@GOTPCREL(%rip),%rdi
        call    Pl_Create_Choice_Point3@PLT

Lpred1_2:
        movq    $15,%rdi
        movq    0(%r12),%rsi
        call    Pl_Get_Integer_Tagged@PLT
        test    %rax,%rax
        je      fail
        (略)
        jmp     X0_bar__a2@PLT

Lpred1_3:
        movq    Lpred1_5@GOTPCREL(%rip),%rdi
        call    Pl_Update_Choice_Point3@PLT

Lpred1_4:
        movq    ta@GOTPCREL(%rip),%rdi
        movq    0(%rdi),%rdi
        movq    0(%r12),%rsi
        call    Pl_Get_Atom_Tagged@PLT
        test    %rax,%rax
        je      fail
        (略)
        jmp     X0_bar__a2@PLT

Lpred1_5:
        movq    Lpred1_7@GOTPCREL(%rip),%rdi
        call    Pl_Update_Choice_Point3@PLT

Lpred1_6:
        movq    0(%r12),%rdi
        call    Pl_Get_List@PLT
        test    %rax,%rax
        je      fail
(以下略)
```

- GNU-Prolog のアセンブリコードでは、WAM(Prolog中間 仮想マシン)のswitch_on_termに対応するランタイム関数Pl_Switch_On_Term() を呼んでいる
- その返り値でジャンプしている

ここで、

- レジスタ間接によるレジスタ指定ができる新命令を追加
  - 特にジャンプ命令
  - ロード命令、ストア命令
- 例えば、WAMコード switch_on_term をアセンブリ・コードにする場合に、型を示すタグ(3bit)によって、レジスタ間接によって指定されたレジスタの値(アドレス)にジャンプすると高速になる
- すなわち

```
call    Pl_Switch_On_Term@PLT ;サブルーチン・コールなので、とても遅い
jmp     *%rax
```

を、下記1命令で置き換えて、高速化

branch_reg_indirect IREG ; pc ← pc + xreg[IREG]

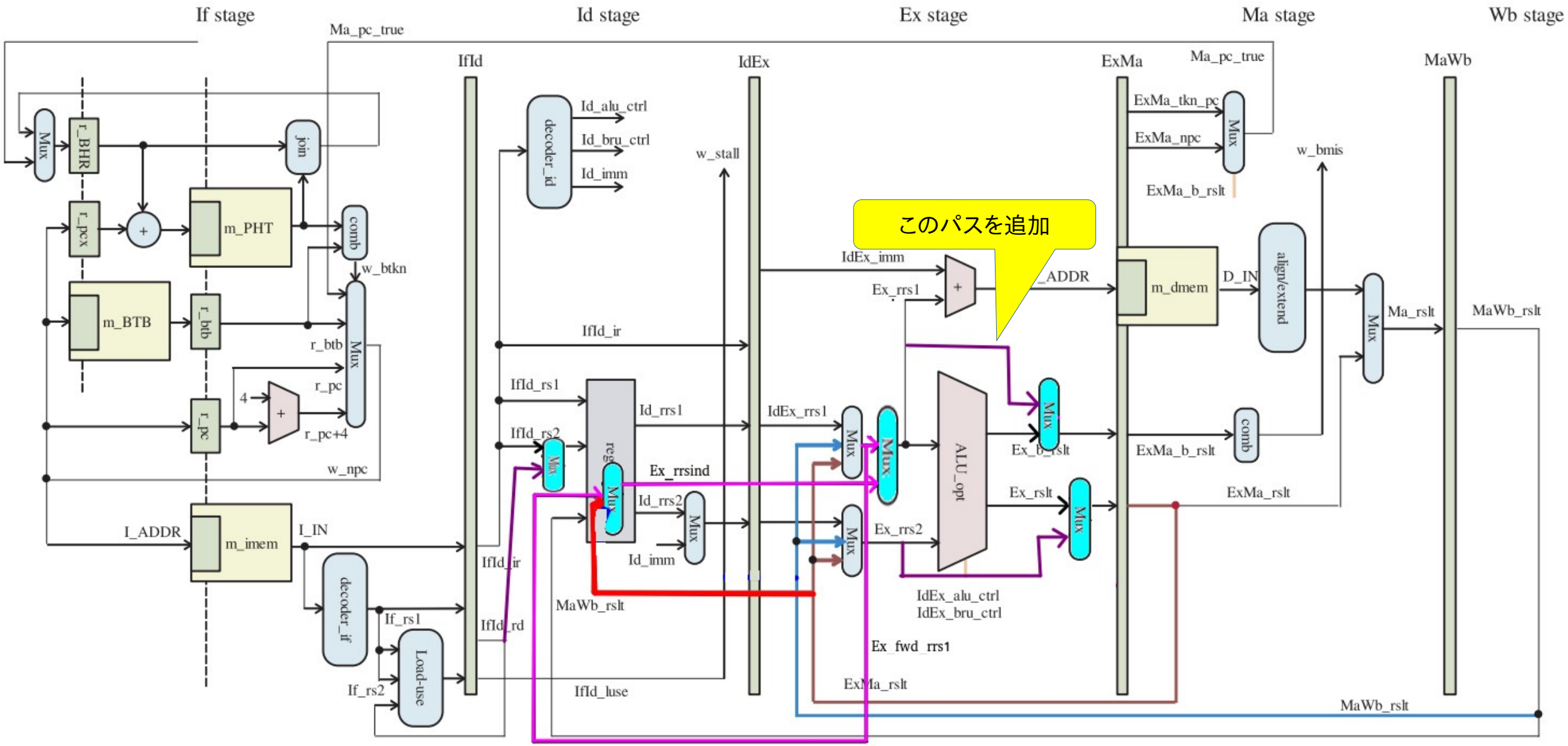
mov_reg_regindirect (レジスタ間接指定によるデータ・レジスタ指定,参照)

- `x[rd] = x[x[rs1]]`
- `I-type`
- 暫定的に、`opcode=0b0001011 (custom-0), funct3=0b000, imm=0`とした
- フォワーディング部の修正
  - 次ページ参照
- branch_reg_indirectなどjump系命令も、同様
  - 書き込みレジスタがpcになるだけ

mov_regindirect_reg(レジスタ間接指定によるデータ・レジスタ指定,ストア)

- x[x[rd]] = x[rs1]
- I-type
- 暫定的に、opcode=0b0001011 (custom-0), funct3=0b001, imm=0とした
- 変更点
  - Id stageではrdをrs2として扱う
    - rs2のフォワーディング部を利用するため
  - Ex stageではフォワーディング後のrs2をExMa_rdに書き込む
  - 次ページ参照
- mov_reg_regindirectに追加で実装した
  - Fmaxのさらなる低下はなし

mov_reg_regindirect・mov_regindirect_reg(ストア)追加後のパイプライン



2022/JUL/15 b

俺SoC LSIでは、追加したパスは、クリティカル・パスにはならない

以上