オープンソースRISC-VコアIP "mmRISCシリーズ"

Open Source RISC-V Core IP "mmRISC Series"

Munetomo Maruyama (圓山宗智)

@Processing_Unit

Rev.02

自己紹介

mmRISC

- ●氏名　　：圓山 宗智（まるやま　むねとも）
- ●出身地　：京都市左京区
- ●趣味　　：1978年からマイコン・FPGA・GPUと戯れ。
- ●仕事　　：1986年から半導体メーカにてマイコンLSI・半導体デバイスの設計に従事
  （現職：マイコン・アナログIC・デジタルIC・
  パワーTr・パワーDi・車載IC・LEDデバイスの開発統括職）
- ●執筆活動：2000年からマイコンを絡めた雑誌記事と書籍を執筆

ー雑誌記事（いずれもCQ出版社）～100本

Design Wave Magazine

2006年5月号　ARMベース・システムLSI開発の事例研究

2008年6月号　ARM汎用プロセッサで使える汎用JTAGデバッガを自作する、など

インターフェース

2014年11月号～2015年6月号（連載）　並列処理プロセッサxCORE徹底研究、など

トランジスタ技術

2019年9月号（特集）Cで直叩き！超並列コンピュータGPU全速力、など

ー書籍（いずれもCQ出版社）

- ・今すぐ使えるH8マイコン基板　初版2010年、増補版2011年
- ・2枚入り超小型ARMマイコン基板　2011年
- ・ARM PSoCで作るMyスペシャル・マイコン　基板付き　2013年
- ・ARM PSoCで作るMyスペシャル・マイコン　開発編　2013年
- ・SHマイコン活用記事全集　2014年
- ・FPGA電子工作スーパキット　2016年
- ・MAX10実験キットで学ぶFPGA＆コンピュータ　2016年
- ・完全版FPGA電子工作オールインワン・キット　2016年

RISC-Vの自作へ

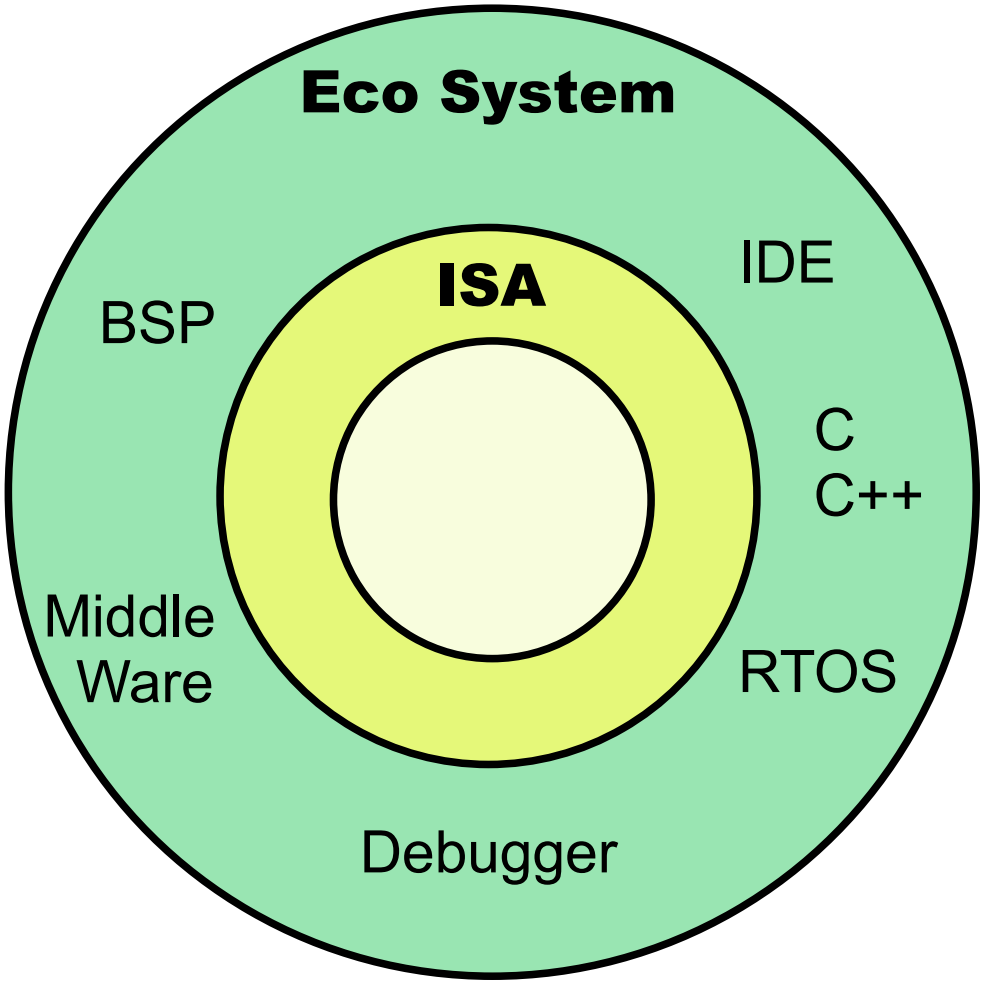
RISC-Vの本質：ISAのみ定義

利点：充実したエコ・システム

欠点：実ハードウェアがない

Big Impact !

ISA: Instruction Set Architecture

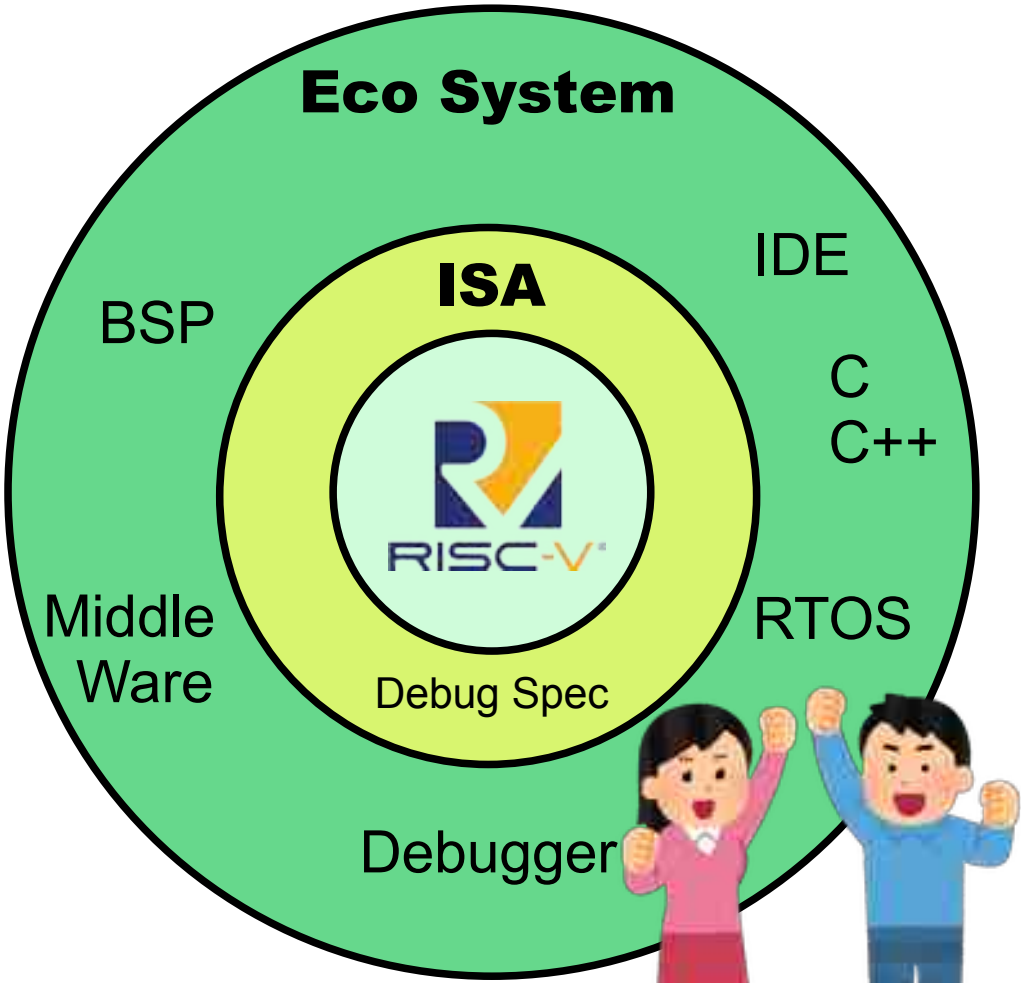


自作RISC-Vコア

利点：充実したエコ・システム

ライセンス/ロイヤルティ費なし

開発は楽しく多くを学べる



なぜ、RISC-Vを自作しようと思ったか？

【建前】

- RISC-Vはリソースがあれば自力設計するほうが得
- CPUコア論理を作れば、エコ・システムが手に入り、実用的
- 中身がよくわかったCPUコアを手にできる（White-Box）
- 多くを学べ、多くを経験できる
- 技術者教育に活用できる

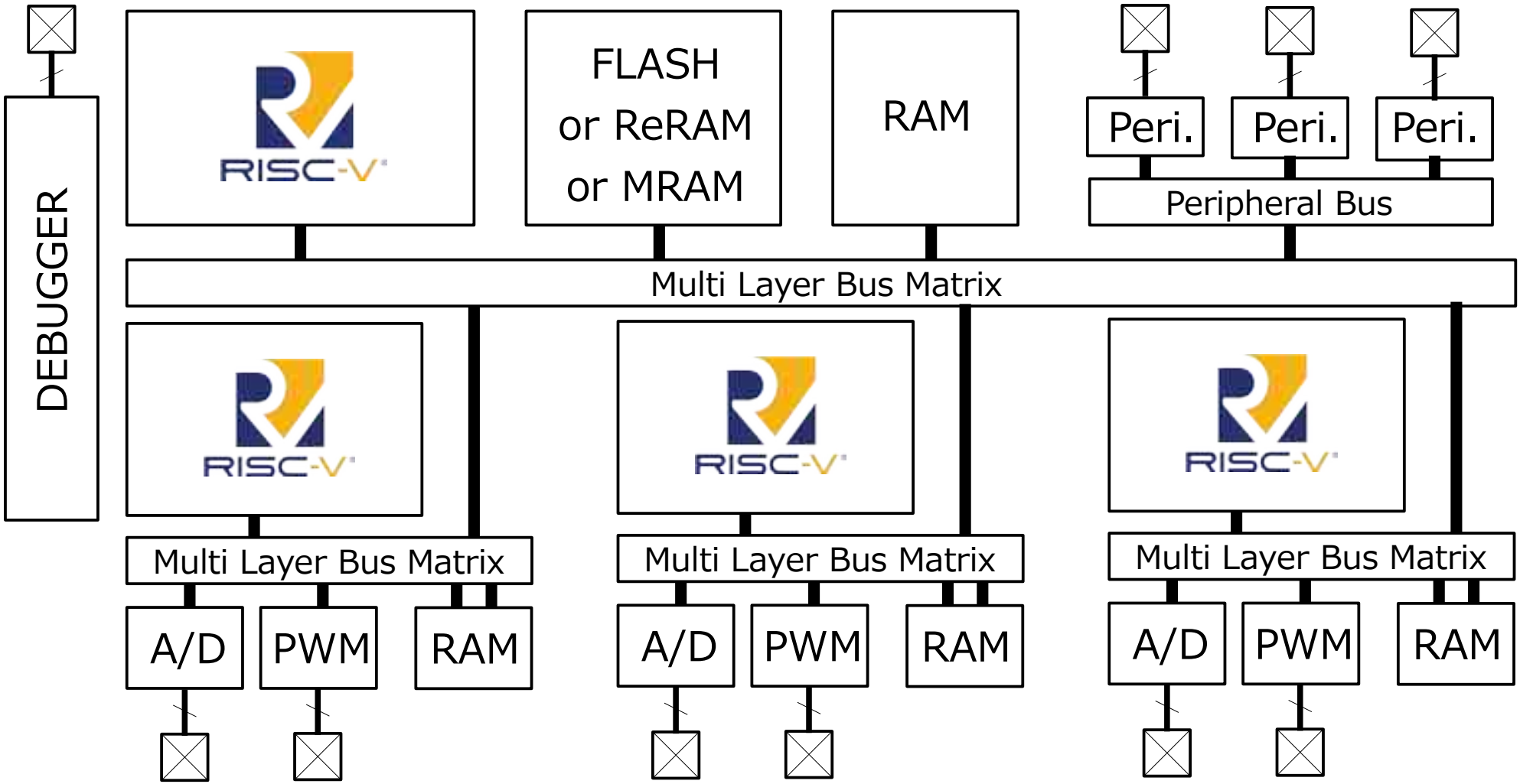
【本音】

- とにかく楽しい！

■動作クロック周波数は、低消費電力化のためできるだけ下げたい

■レイテンシが長い割り込みによるタスク切り替えは避けたい

⇨ マルチ・コア／ヘテロジニアス構造／ドメイン・スペシフィック



自作RISC-Vコア：mmRISC-1（much more RISC）

■命令セットの選択

RV32IMAFC

●メモリは、マイコン・チップに内蔵できる程度の容量（4GB以下）
→32ビット・アーキテクチャ【RV32I】

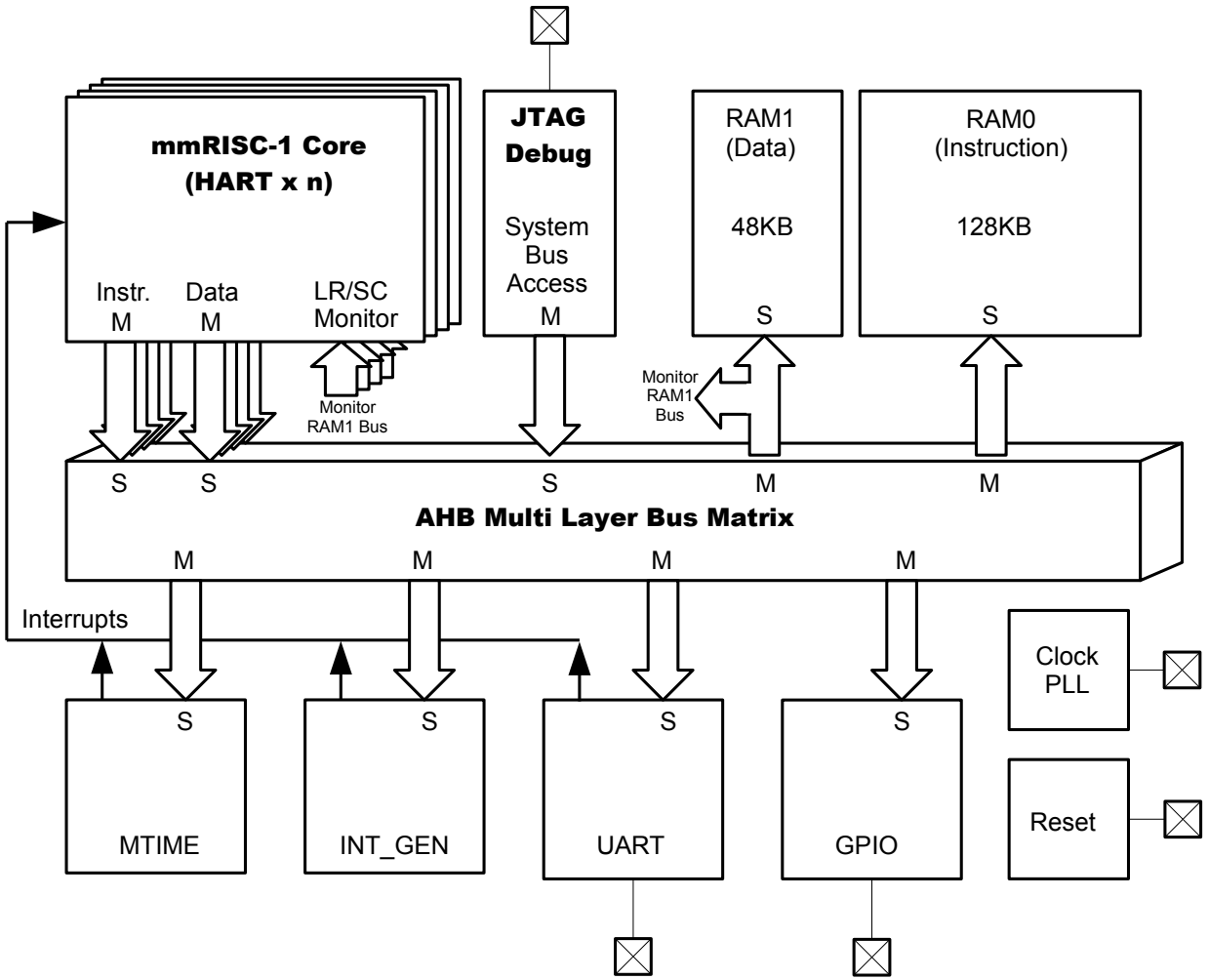
●高速デジタル・フィルタ処理
→32ビット固定小数点演算：高速1サイクル乗算【M】
→32ビット浮動小数点演算：高速1サイクル加減乗算【F】

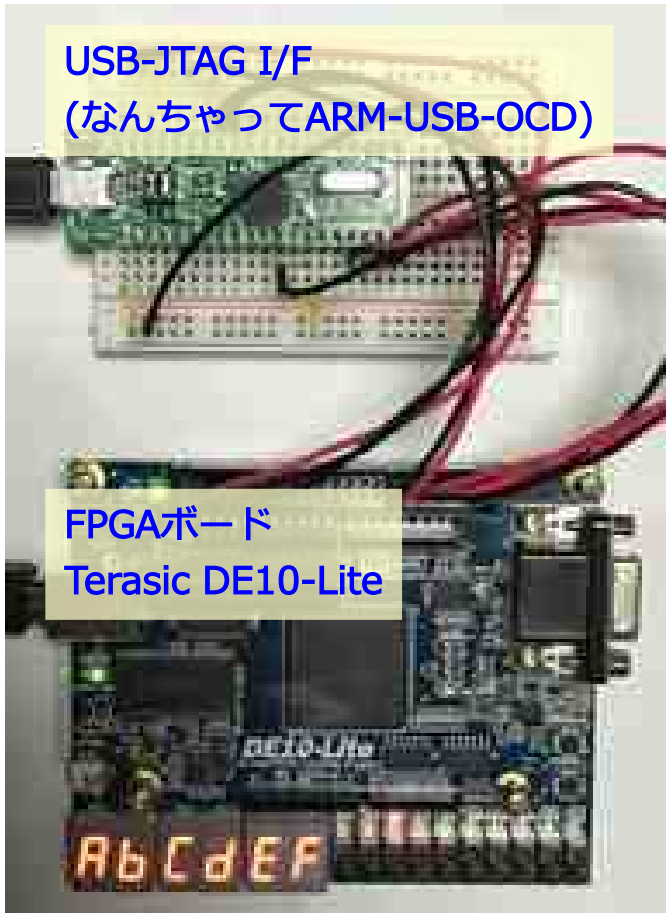
●マルチ・コア間でメモリ内変数を共有
→アトミック・メモリ・アクセス命令【A】

●コード・サイズの縮小
→16ビット長の圧縮命令【C】

- RV32IMAFCが動作し、
  JTAGデバッグできることを目的とした
- シンプルなシステム構成（GPIO+UART）

→昨年のRISC-V Days Tokyo 2021 Autumnで発表



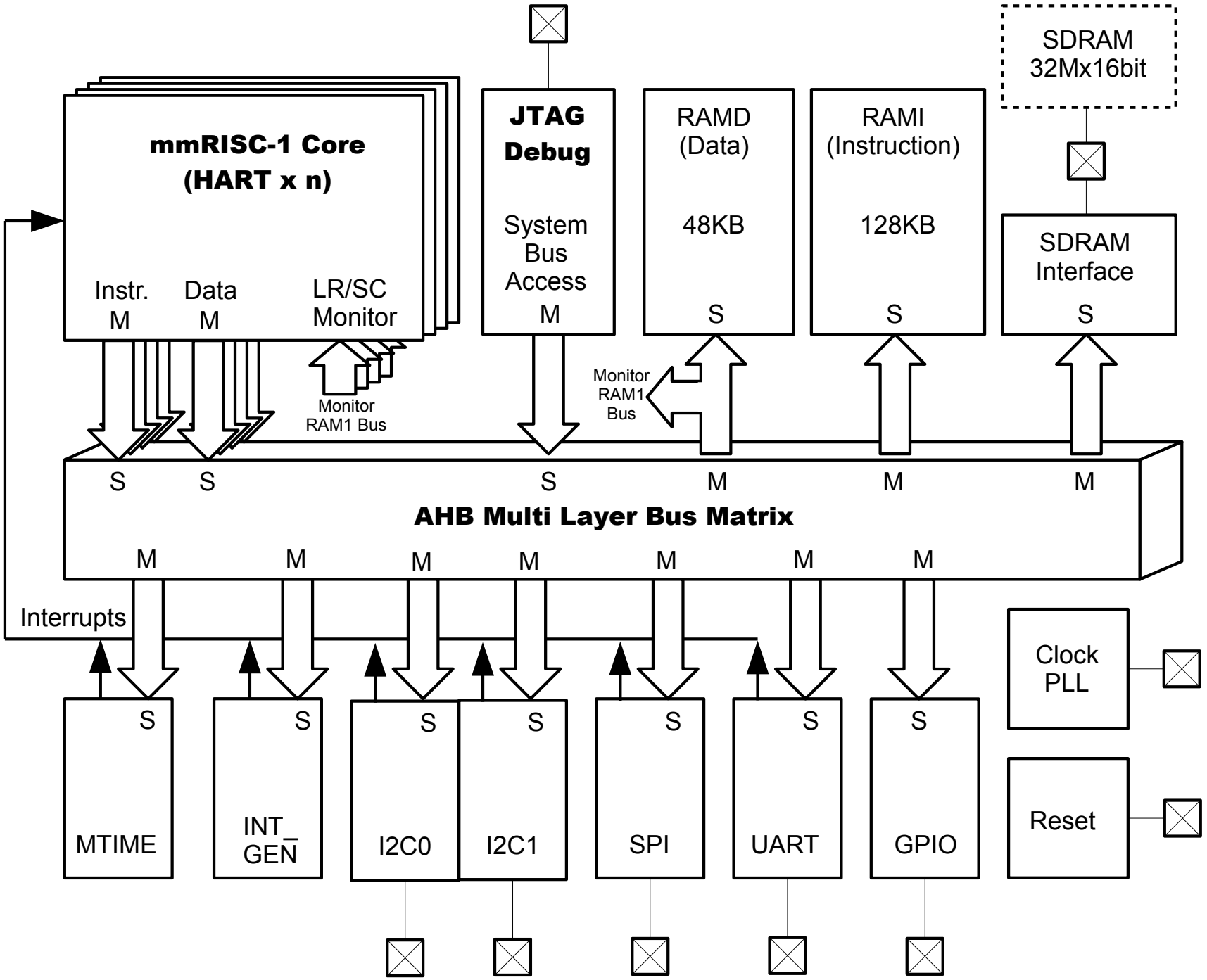


実用版
mmRISC-1

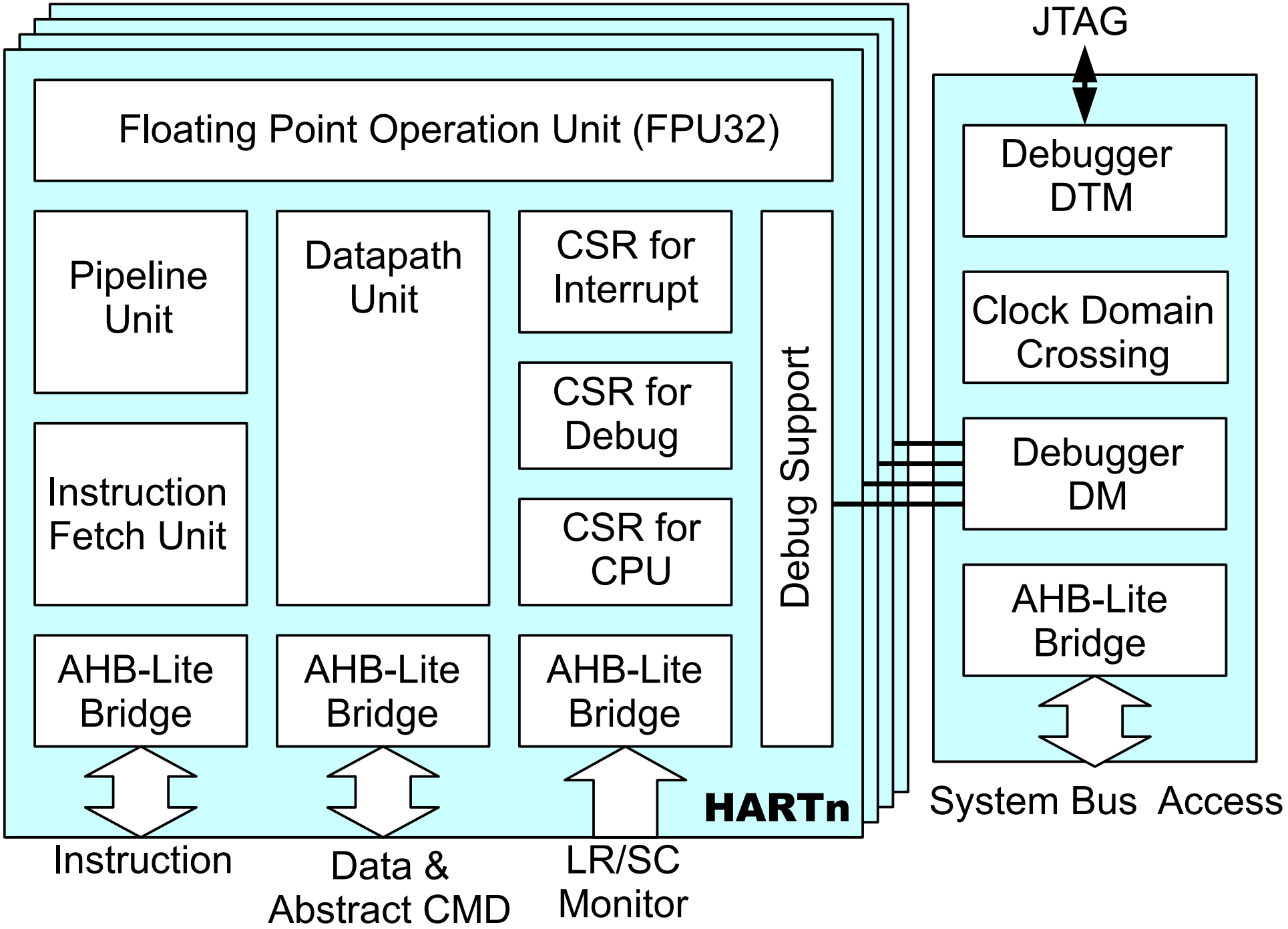
自作RISC-V CPUコア「mmRISC-1」の仕様

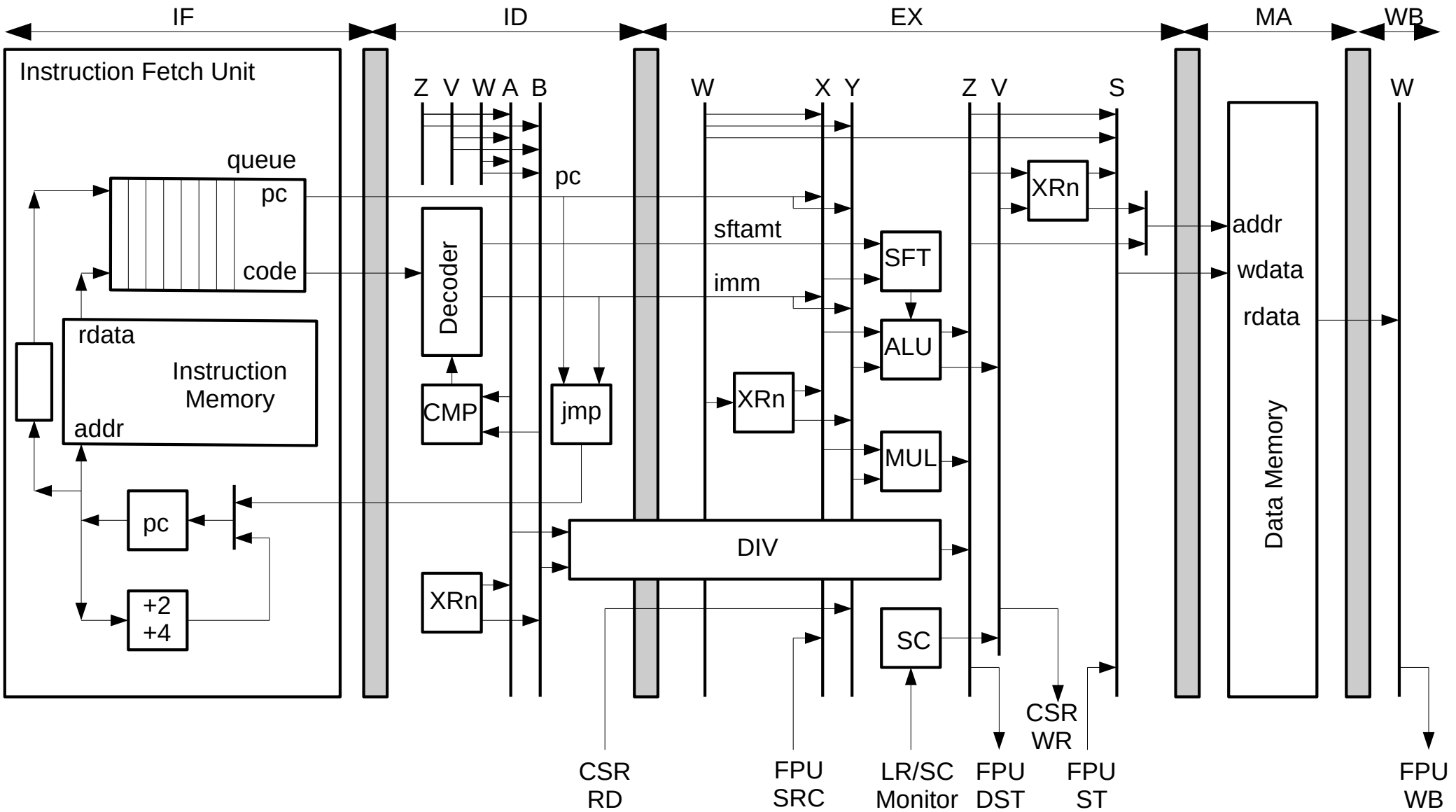
項　目	内　容
CPUコア名称	mmRISC-1 (Much More RISC)
命令セット・アーキテクチャ(ISA)	RV32IM[A][F]C　　[ ]は選択可能
Hart数（CPUコア数）	1 hart ～ 1,000,000 harts
特権モード	Machine Modeのみ
パイプライン段数	整数命令：3～5段、浮動小数点命令：5～6段
32ビット整数乗算性能	1-cycle
32ビット浮動小数点演算性能	1-cycle (FADD.S/FMUL.S/FMADD.S)
デバッグ・サポート機能 **（実用CPUコアのために必須）**	RISC-V規格の「External Debug Support Ver.0.13.2」に準拠 JTAGインタフェース ハードウェア・トリガ x 4
割り込み	External + Machine Software + Machine Timer +**User IRQ：64入力** 全ての割り込み要因に独立ベクタをアサイン IRQ割り込み要求毎に16段階の優先レベル指定可、ネスティング可
バス・インタフェース	AHB-Lite（命令／データ／LR-SCモニタ／デバッグ・アクセス）
RTL	Verilog-2001, System Verilog
検証	論理シミュレーション、システム検証（FPGA + OpenOCD + Eclipse）
Github	`https://github.com/munetomo-maruyama/mmRISC-1`

mmRISC-1 FPGAシステムのブロック図

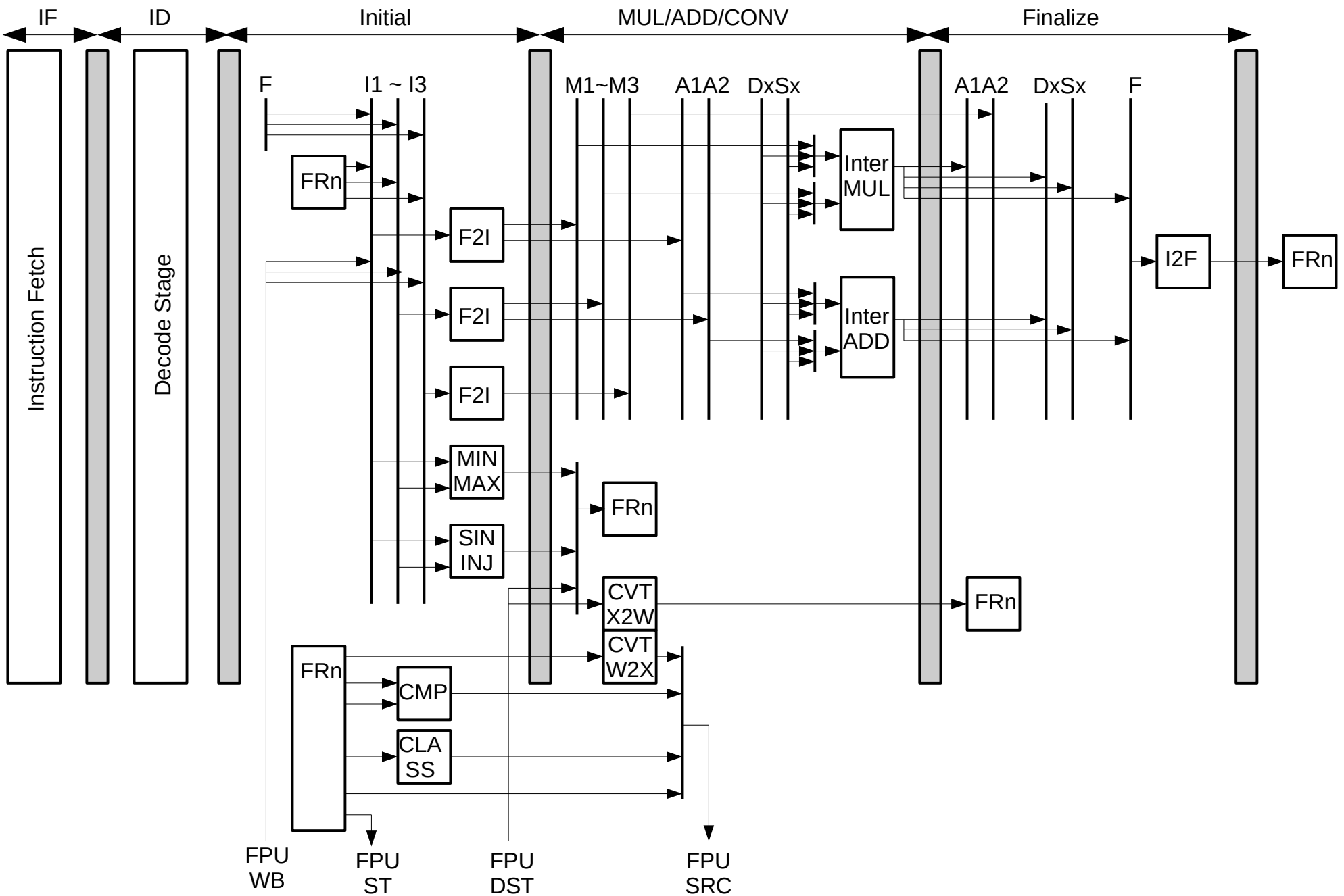


「mmRISC-1」CPUコアのブロック図





mmRISC-1 FPU32のパイプライン制御



https://github.com/munetomo-maruyama/mmRISC-1

●周辺機能追加
・I2C
・SPI
・SDRAM I/F
　DE10-Liteボード上のSDRAM ISSI IS42S16320D（64MB：32Mx16bits）
　専用の簡易コントローラ

●サンプル・アプリ追加
・Touch LCD制御

●Raspberry PiをRISC-V統合開発環境化
・GNU Toolchain on RasPi
・Open OCD on RasPi
・Eclipse on RasPi

RISC-V開発環境：Ubuntu版

MacBook Pro (intel)

Parallels Desktop（仮想PC）

Ubuntu Desktop (x64)

Editor (geany etc.)

Eclipse
ソフト開発IDE

Open OCD
JTAG I/F

RISC-V GNU Tool Chain

Questa
論理SIM

Quartus
FPGA合成

- この環境で、RISC-Vの論理設計、論理シミュレーション、FPGA合成ソフト開発、JTAGデバッグまで一貫して作業できる。

OpenOCD JTAG Debugger
FTDI FT2232D

USBケーブル

(USB-JTAG)

JTAG &
UART

USBケーブル

(USB Blaster)

Terasic DE10-Lite FPGAボード

RISC-V開発環境：Raspberry Pi版

Raspberry Pi 4 model B (4GB、64bit OS)：ソフトウェア開発

Eclipse
ソフト開発IDE

Open OCD
JTAG I/F

RISC-V GNU Tool Chain

JTAG &
UART

Ubuntu / Windows 11 PC （64bit OS)：ハードウェア開発

Questa
論理SIM

Quartus
FPGA合成

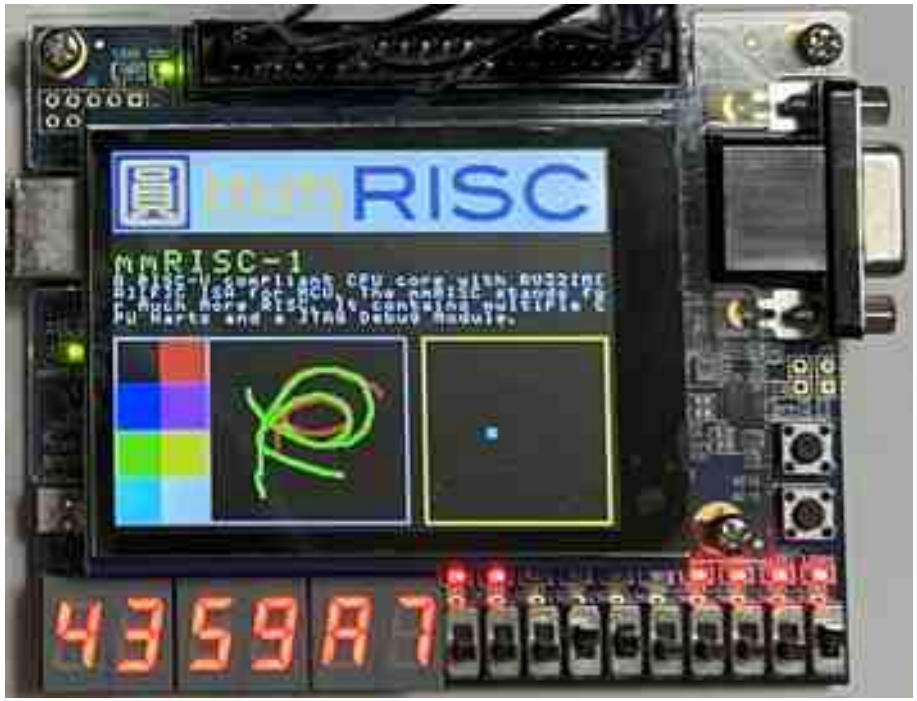
USBケーブル

(USB Blaster)

Terasic DE10-Lite FPGAボード

「DE10-Liteボード＋LCDパネル」がカッコいい

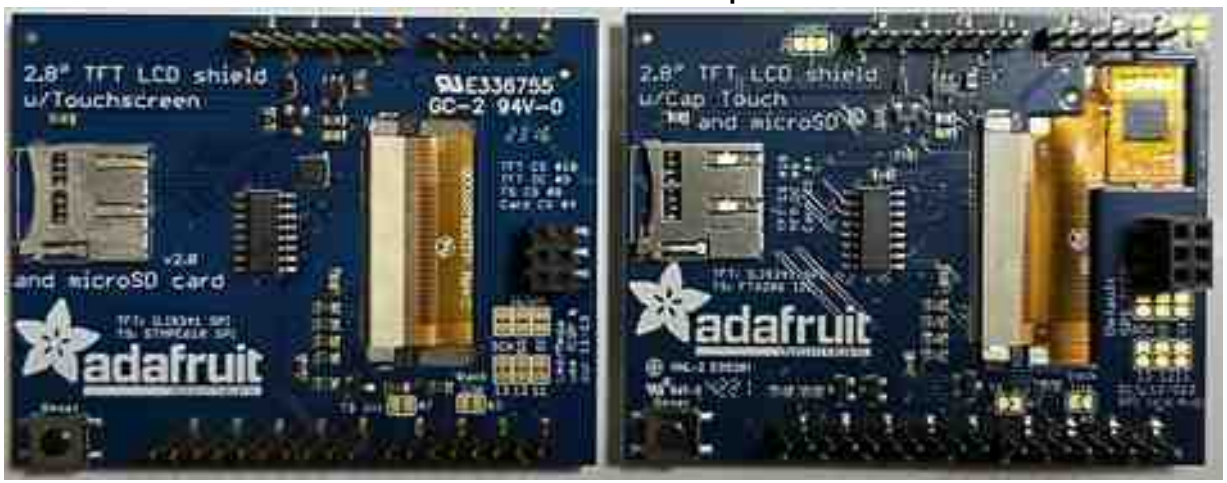
Terasic DE10-Lite FPGAボード



【左】ミニ・ペイント
【右】加速度センサにより
基板の傾きでボール移動

※Resistive Touchか
Capacitive Touchを
自動判定

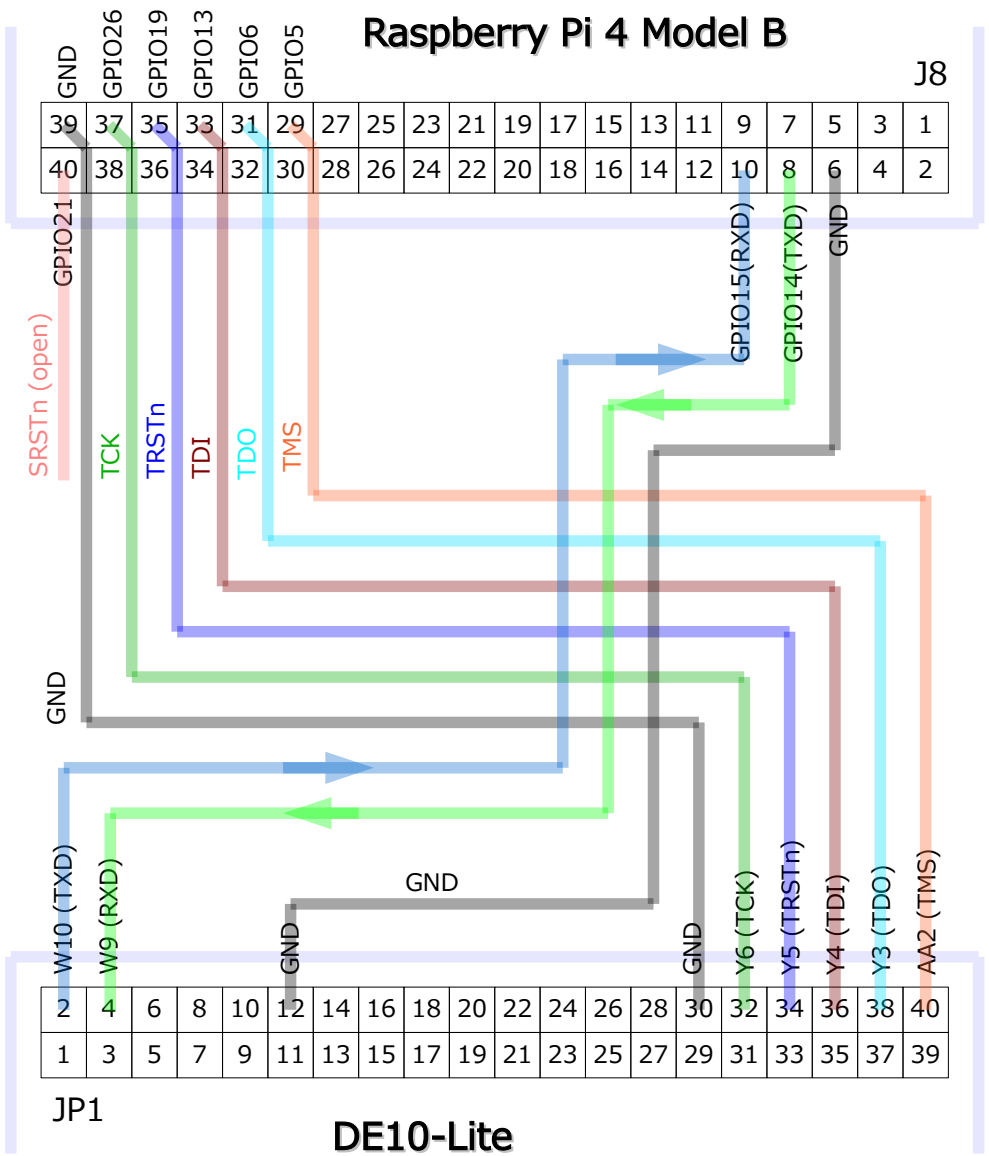
Resistive Touch Panel　　Capacitive Touch Panel

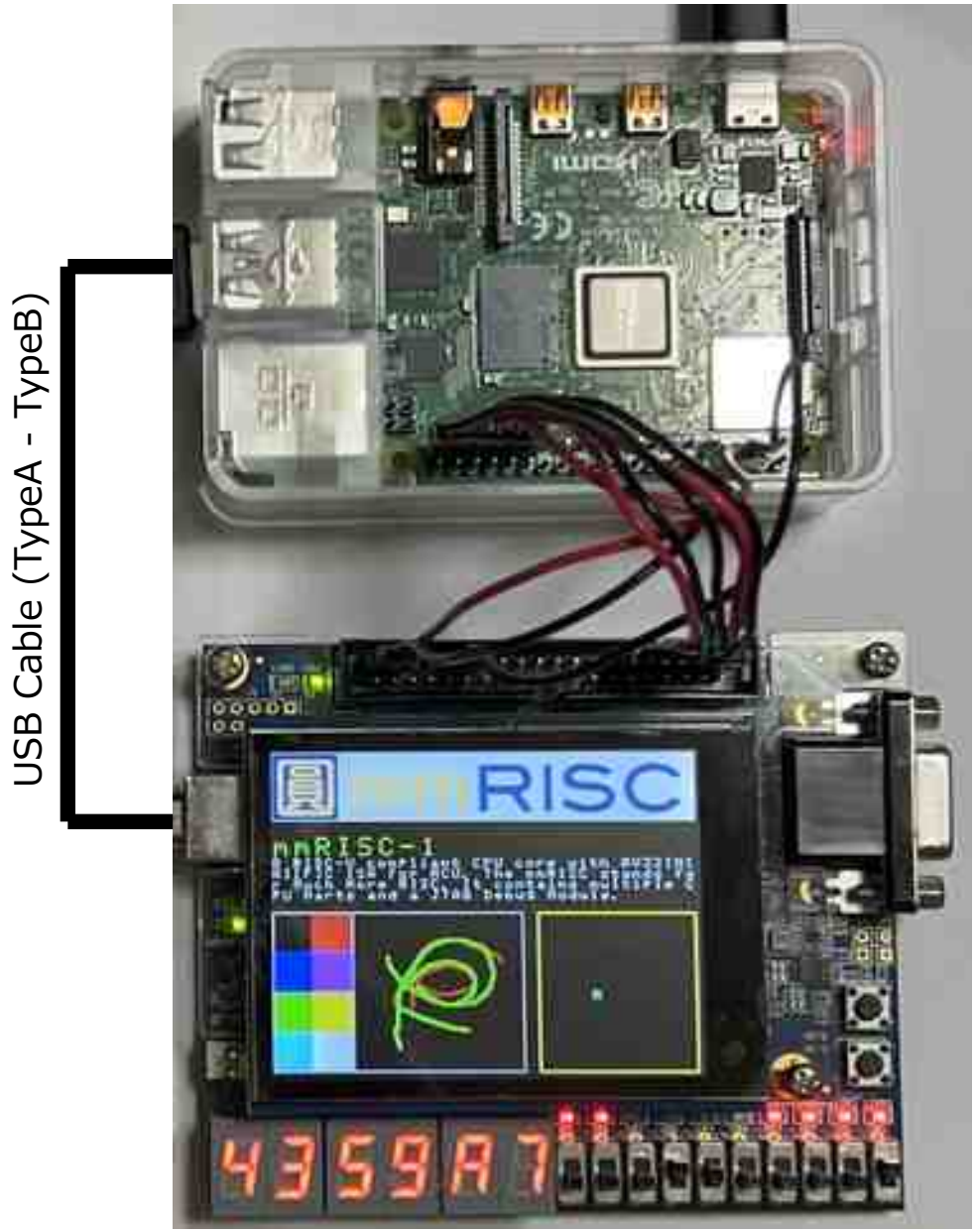


Adafruit 2.8” TFT Touch Shield v2

接続方法：Raspberry Pi版（mmRISC-1）

●接続方法





OpenOCD Configuration for Raspberry Pi

●Raspberry Pi用：...mmRISC-1/openocd/openocd_rpi.cfg

```
adapter driver bcm2835gpio
adapter speed 1000
transport select jtag

#Raspberry Pi 3B/3B+
#bcm2835gpio_peripheral_base 0x3F000000

#Raspberry Pi 4B
bcm2835gpio_peripheral_base 0xFE000000

# Transition delay calculation: SPEED_COEFF/khz - SPEED_OFFSET
# These depend on system clock, calibrated for stock 700MHz
# bcm2835gpio speed SPEED_COEFF SPEED_OFFSET
bcm2835gpio_speed_coeffs 236181 60

# Each of the JTAG lines need a gpio number set: tck tms tdi tdo
# Header pin numbers: 37 29 33 31
bcm2835gpio_jtag_nums 26  5 13  6

# If you define trst or srst, use appropriate reset_config
# Header pin numbers: TRST - 35, SRST - 40
bcm2835gpio_trst_num 19
# reset_config trst_only
bcm2835gpio_srst_num 21
# reset_config srst_only srst_push_pull
# or if you have both connected,
reset_config trst_and_srst srst_push_pull

proc init_targets {} {

    set _CHIPNAME riscv
    jtag newtap $_CHIPNAME cpu -irlen 5

    set _TARGETNAME $_CHIPNAME.cpu
    target create $_TARGETNAME riscv -endian little -chain-position $_TARGETNAME -coreid 0

#    init
}
```

GPIOピン・ヘッダへの信号アサイン

BCM2835 GPIO	Header Pin No.	Signal	Note
GPIO14	Pin 8	TXD (output)	*/dev/ttyS0*
GPIO15	Pin 10	RXD (input)	*/dev/ttyS0*
GPIO05	Pin 29	TMS	
GPIO06	Pin 31	TDO	
GPIO13	Pin 33	TDI	
GPIO19	Pin 35	TRSTn	
GPIO26	Pin 37	TCK	
GPIO21	Pin 40	SRSTn	not used
GND	Pin 6, Pin 9 Pin 14, Pin 20 Pin 25, Pin 30 Pin 34, Pin 39	GND	

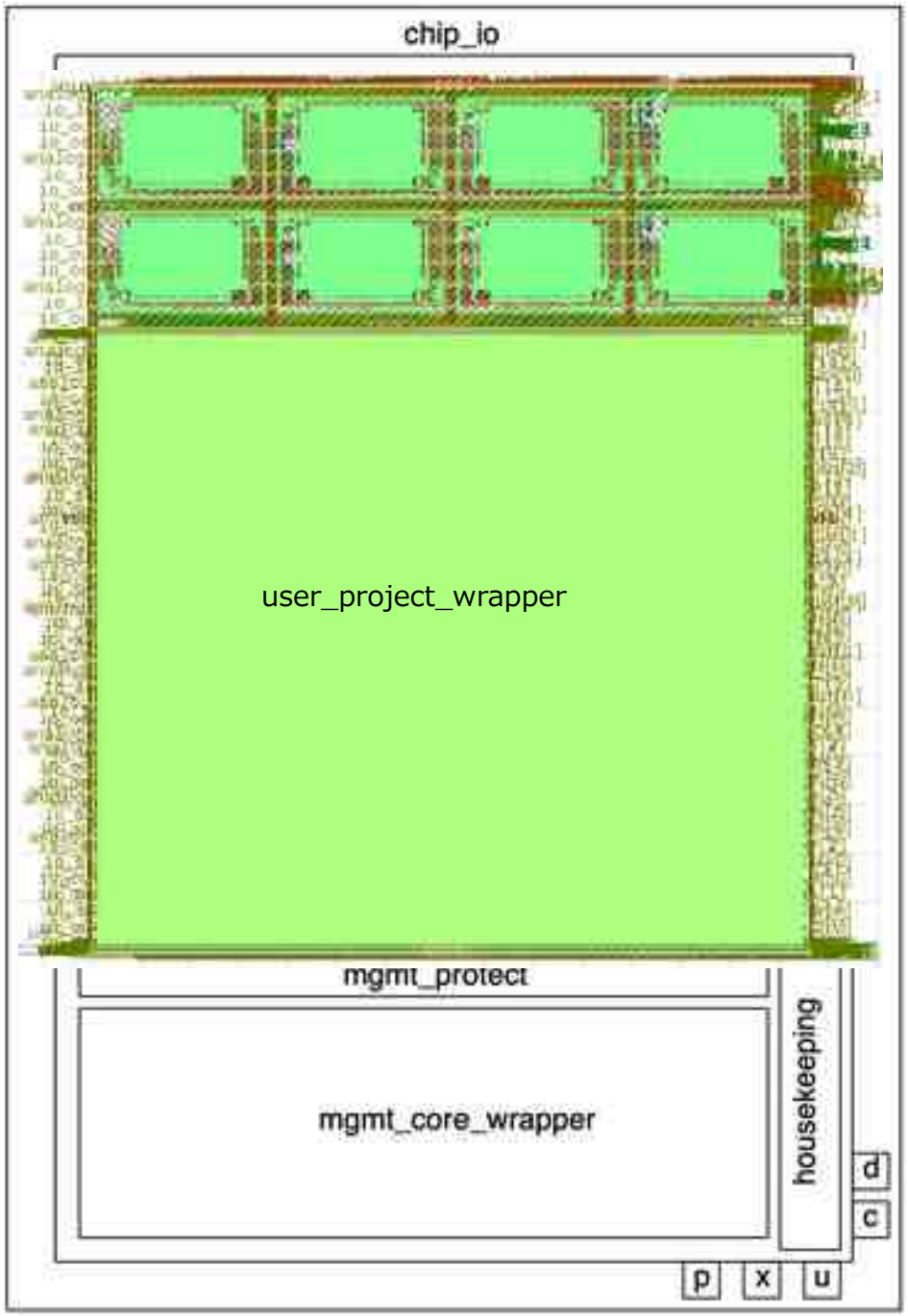
mmRISC-1 in Efabless Caravel Chip

工事中
立入禁止

- ロジックは、未検証の超仮論理だが
- sky130の2KB 1rw1r SRAMマクロとロジックを混ぜて実装はできた
- Pre Checkまでパス
- ツールとフローはOKそう

※RAMを90°回転させると電源が繋がらずLVSエラー
※南側にmanagement coreがあるのでそちらにマクロを置かない方が配線混雑防止できる
※Detail Routingの時、PCメモリをかなり食う
Parallels Desktop上のUbuntuで動かしたが27GB割り当ててあげないとコケる
（PCが32GB機なので結構きつい）

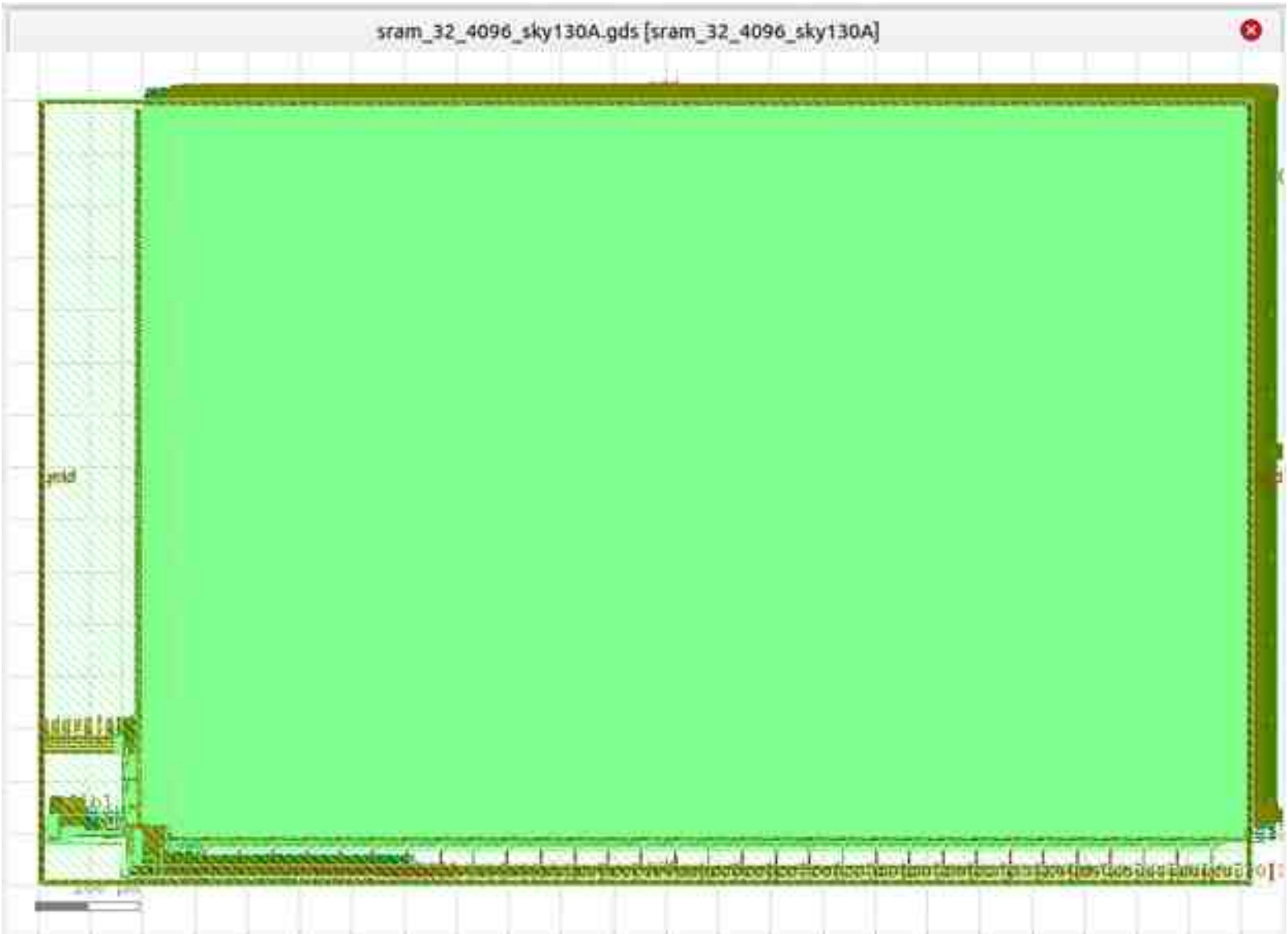
- アンコをしっかり機能設計＆機能検証してNTO目指して仕上げる！



RAM Compiler OpenRAMの試行

- SkyWater 130nm
- 2KB (32bitx 512word) : 0.4mm^2、120MHz
- 16KB (32bitx4096word) : 3.5mm^2、67MHz

16KBでもtACC遅め
SKY130は2-port未対応？

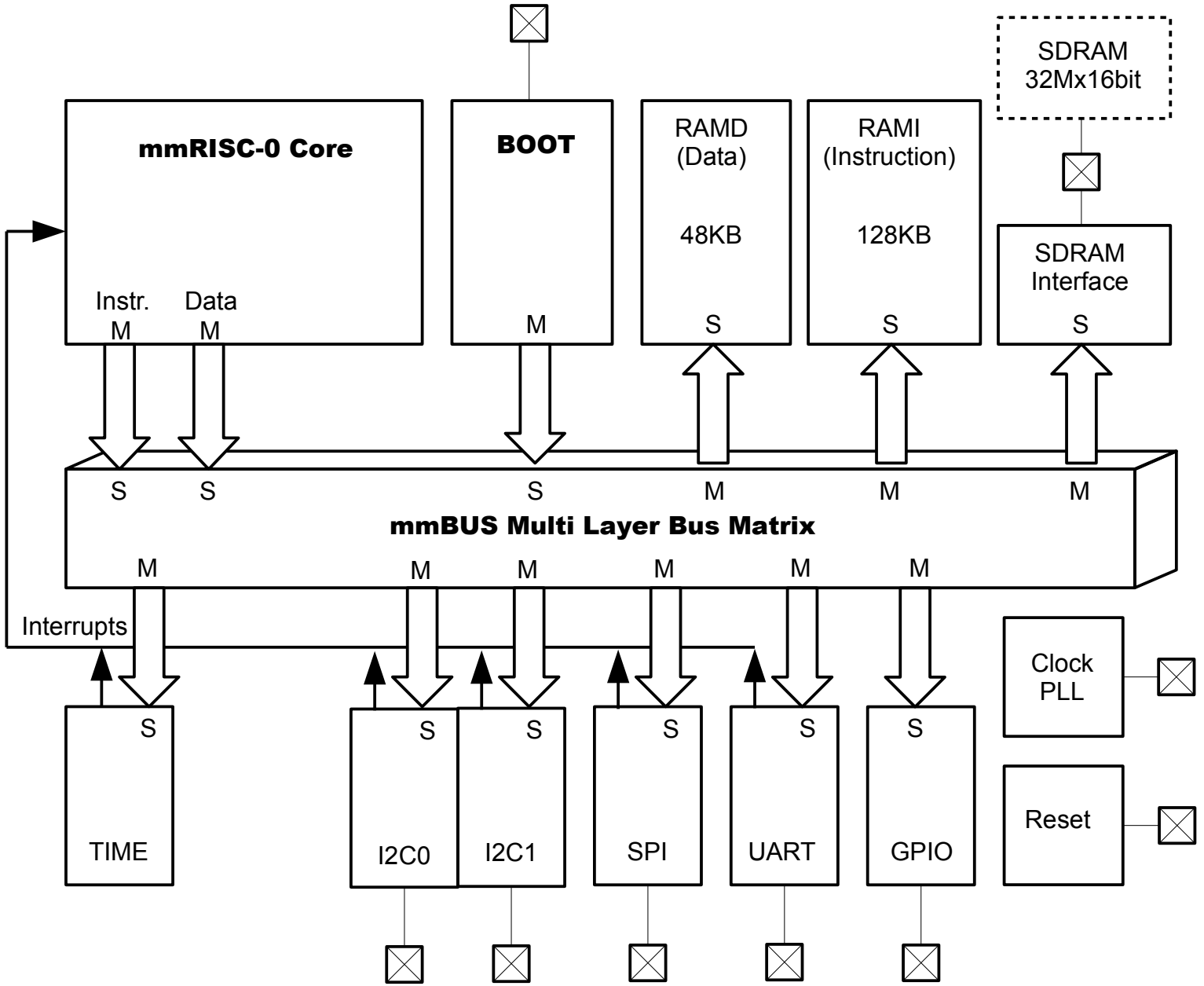


簡易版（教育用）
mmRISC-0

簡易版mmRISC-0と、実用版mmRISC-1

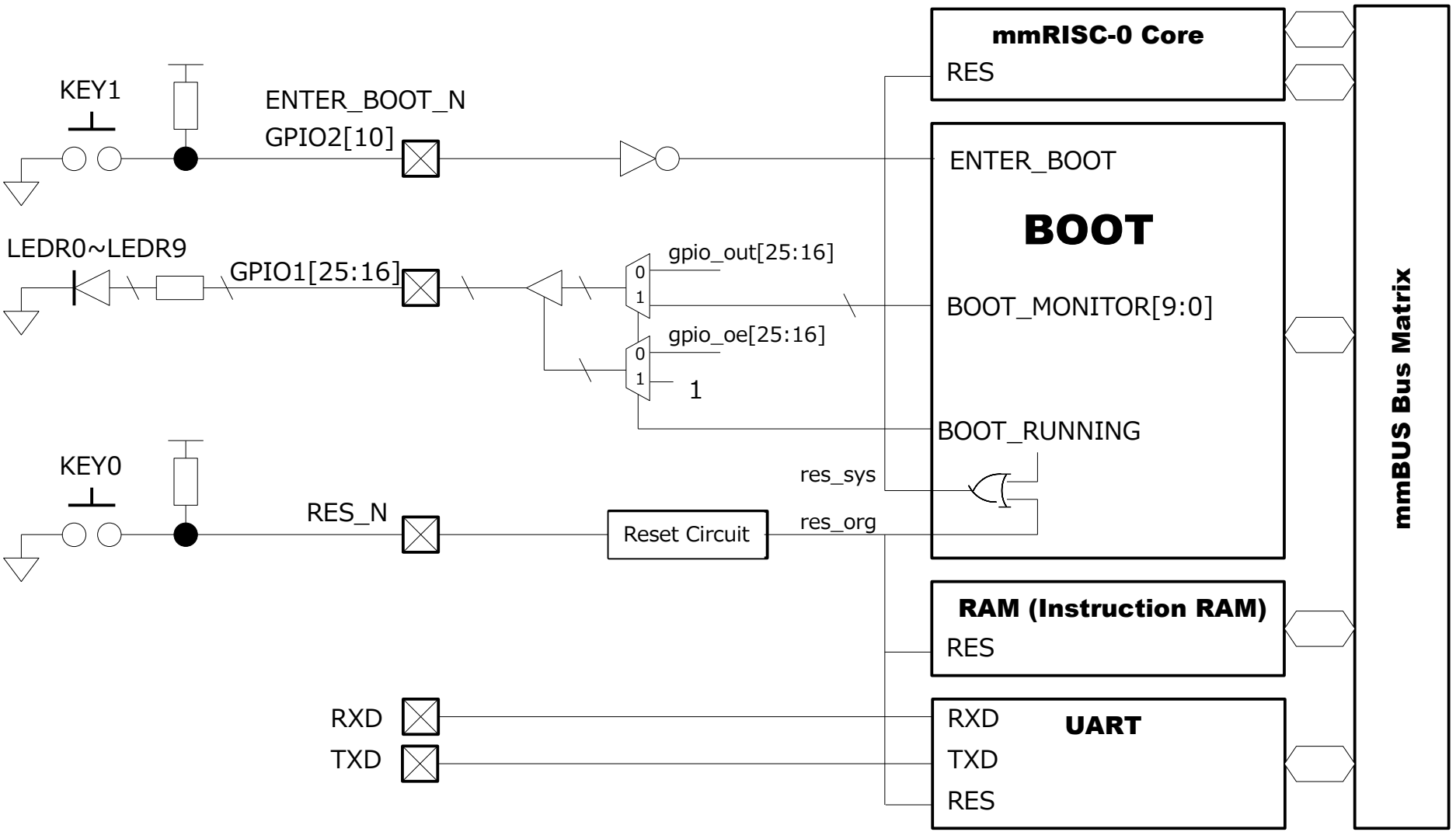
項目	mmRISC-0 (簡易版)	mmRISC-1 (実用版)
RISC-V ISA アーキテクチャ	RV32I	RV32IMC RV32IMAC RV32IMAFC
パイプライン段数	3～5段	整数：3～5段 浮動：5～6段
バス	独自	AHB-Lite
プログラム・ロード	シリアル・ローダ	デバッガ機能が対応
JTAGデバッガ機能	なし	RISC-V標準仕様搭載
割込み入力	EXT+TIME	EXT+TIME+SOFT +IRQx64(優先度x16レベル)
周辺機能	UART/I2C/SPI/GPIO TIME/SDRAM	UART/I2C/SPI/GPIO MTIME/INTGEN/SDRAM

mmRISC-0 FPGAシステム・ブロック図



BOOTモジュール (1)

- mmRISC-0にはデバッグ機構不搭載
- 専用ブート・ローダ「BOOT」を実装
- リセット時にブート・モードに入れ、BOOTモジュールが一時的にバス・マスタになり
  UARTが受信したロード・データを命令RAMへライト。終了したらCPU動作開始。

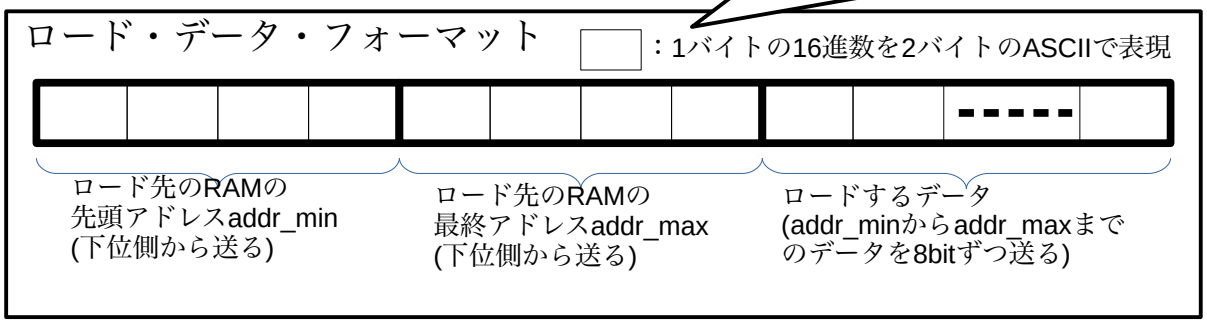


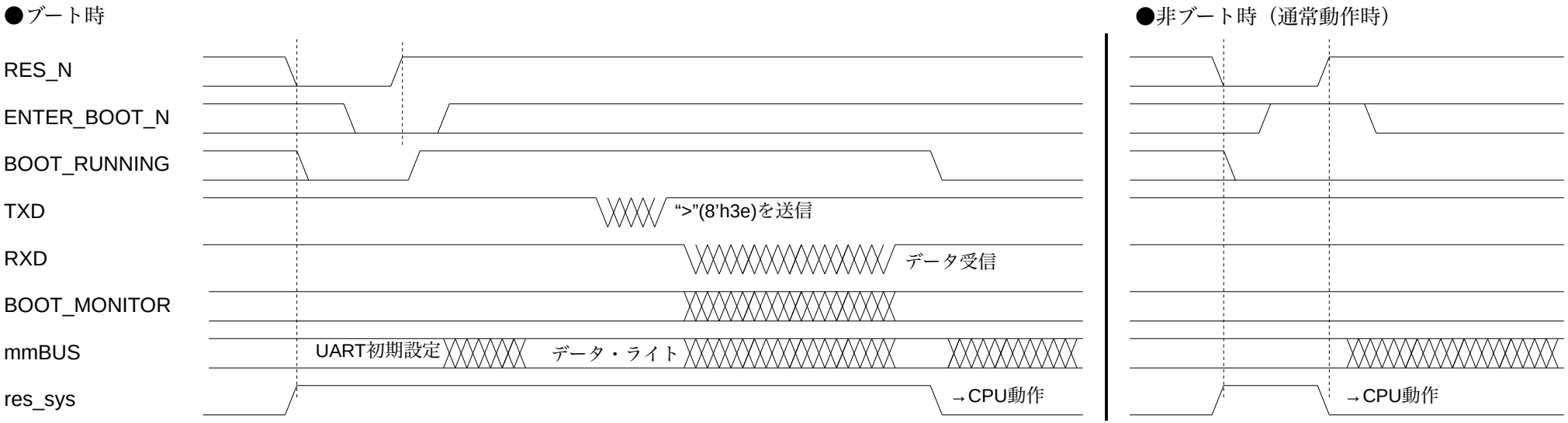
BOOTモジュール (2)

●ブート時の動作

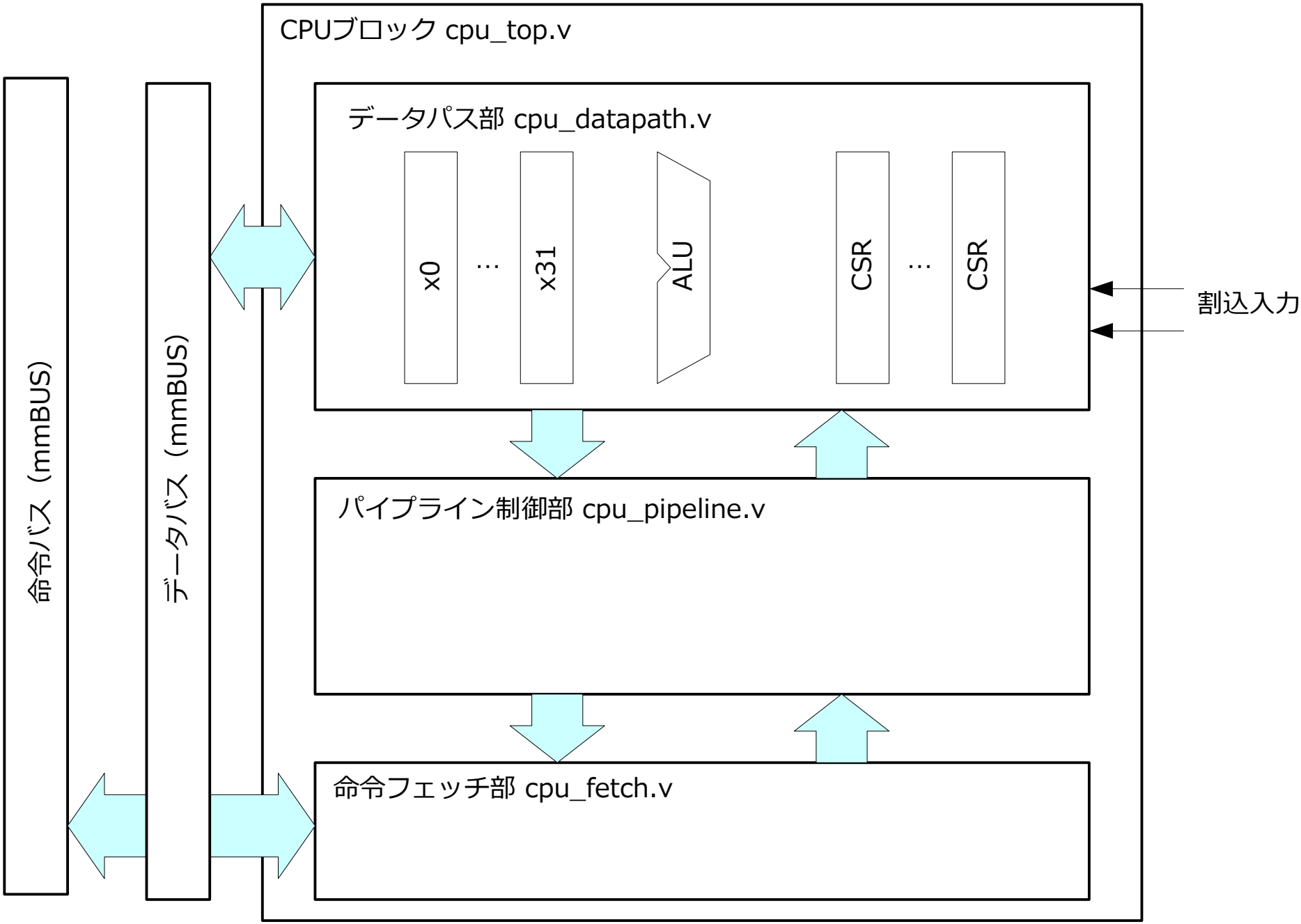
(1) RES_N（res_org）ネゲート時にENTER_BOOT_NがアサートされていたらBOOT動作開始
（FPGAボード上では、KEY0とKEY1を押して、KEY0を離してからKEY1を離す）
(2) BOOT_RUNNINGがアサートされ、命令格納用RAMとUARTだけがリセット解除される
（CPUはrリセットされたままなので停止している）
(3) BOOTモジュールがUARTをアクセスして初期設定（ボーレート115200bps）
(4) BOOTモジュールがUARTをアクセスして「>」（プロンプト）を送信
(5) BOOTモジュールがUARTの受信データを待つ
(6) データを受信したら、指定されたアドレス範囲に
ロード・データを順次ライトしていく。
(7) この間、アドレスのうちaddr[17:8]の10bitを
BOOT_MONITORに出力し、FPGAボード上の
LEDを点滅させる。
(8) 全データ（ロード・データのバイト数＝
addr_max − addr_min + 1）を受信したらBOOT_RUNNINGをネゲートし、システム全体のリセットが解除され、CPUが動作開始する。

●mmRISC-0/tools/hex2ascii.c
ロード・データ・フォーマットを
Intel Hexから生成するプログラム
※ビルド方法
> gcc -o hex2ascii hex2ascii.c
※使用方法
> ./hex2ascii intel.hex > rom.txt



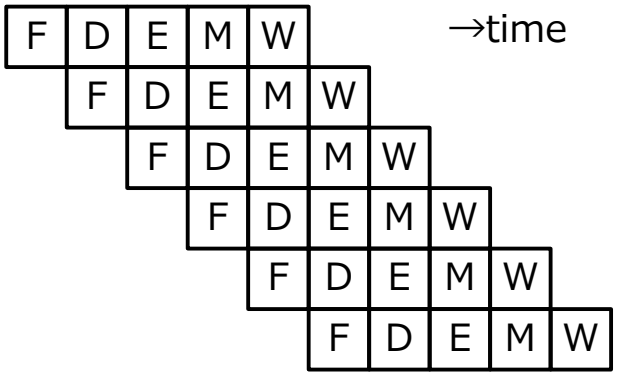


CPUコアmmRISC-0のブロック図



命令のパイプライン制御

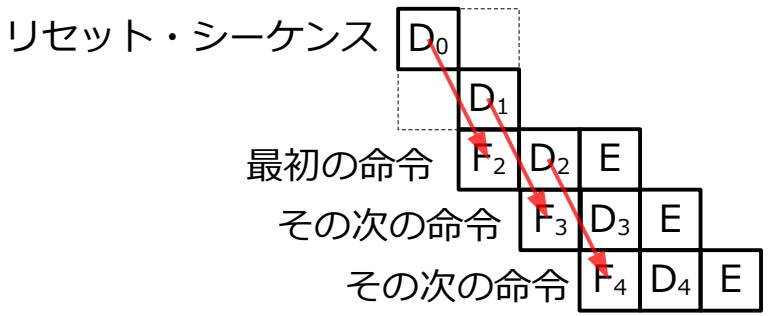
●パイプライン制御の基本は5段



ステージ	意味	内容	備考
F (IF)	Instruction Fetch	命令メモリから命令コードをフェッチ	
D (ID)	Instruction Decode	フェッチした命令コードをデコード	
E (EX)	Execution	汎用レジスタ同志や汎用レジスタと即値との間で演算	●コントローラ系CPUでは、単純化のためこのステージで演算結果を汎用レジスタに戻すことが多い。 mmRISC-0/1もこの方式。 ●プロセサ系CPUでは、MA段でメモリ・エラーが発生した場合、CPU状態を当該命令実行前の状態に戻すためEX段の演算結果をWB段で汎用レジスタに書き戻すことが多い。
M (MA)	Memory Access	EX段の演算結果をアドレスとしてデータ・メモリをアクセス	
W (WB)	Write Back	MA段でデータ・メモリからリードしたデータを汎用レジスタに書き戻す。	●プロセサ系CPUでは、EX段の演算結果をWB段で汎用レジスタに書き戻す。

●IDステージのシーケンサが命令を制御する

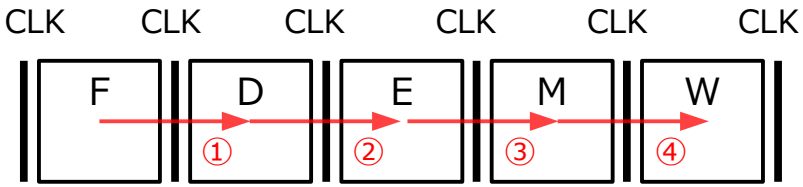
●IDステージが命令フェッチIFを起動する



・リセット・シーケンスは命令フェッチ無しで起動。
・リセット・シーケンスは2ステップ構成。
・リセット・シーケンスのIDステージ・シーケンスの
D_0が最初の命令（PC）の新規(分岐)命令フェッチF_2を起動。
D_1がその次の命令（PC+4）の連続命令フェッチF_3を起動。
・最初の命令のD_2が2命令先の連続命令フェッチF_4を要求。

※最初の命令のアドレスは0x90000000（論理合成時に指定）

●IDステージによる命令全体の制御

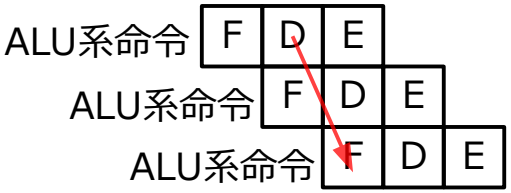


IDステージで生成した制御信号を命令の各ステージに向け、パイプライン的に順番に転送していく。
①IFからIDが命令コードを受け取る。
②IDからEXが制御信号を受け取る。全ての制御信号を受け取る。
③EXからMAが制御信号を受け取る。MA関係とWB関係の制御信号を受け取る。
④MAからWBが制御信号を受け取る。WB関係の制御信号を受け取る。

各命令のパイプライン制御(1)

●ALU系（Reg-Reg/Reg-Imm）命令

```
LUI/AUIPC
ADDI/SLTI/SLTIU/XORI/ORI/ANDI
SLLI/SRLI/SRAI
ADD/SUB/SLL/SLT/SLTU/XOR/SRL/SRA/OR/AND
CSRRW/CSRRS/CSRRC
CSRRWI/CSRRSI/CSRRCI
```

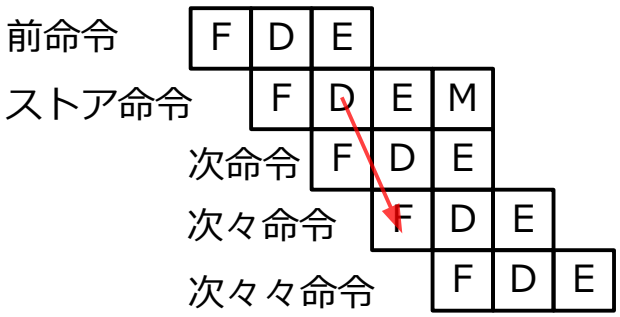


・ALU系命令は、Eステージで、ソース・レジスタを読み出し、
演算を行い、その結果をディスティネーション・レジスタに書き込む。
前後の命令でディスティネーション・レジスタとソース・レジスタが
同一でも、干渉し合わず実行できる。
・Dステージで2命令先の命令フェッチを起動する。

ALU：Arithmetic Logic Unit

●メモリ・ストア命令

```
SB/SH/SW
```

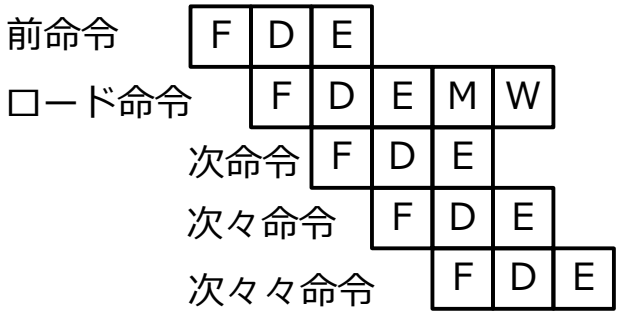


・ストア命令は、Eステージでメモリのアクセス先アドレスを計算して、
ライト・データを準備し、次のMステージでメモリにライトする。

各命令のパイプライン制御(2)

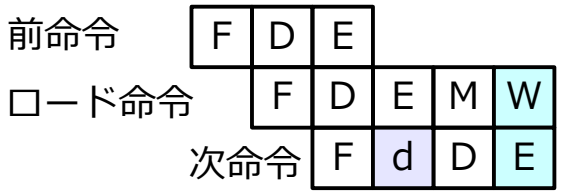
●メモリ・ロード命令　　LB/LH/LW/LHB/LWU

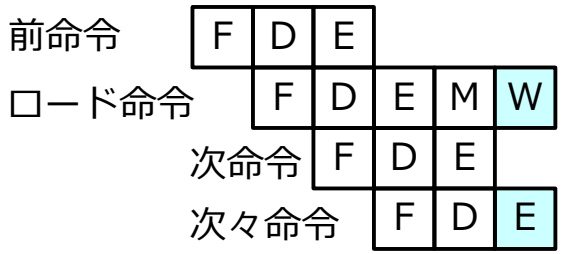
(1) ロード命令のディスティネーション・レジスタを後続の命令が参照しない場合

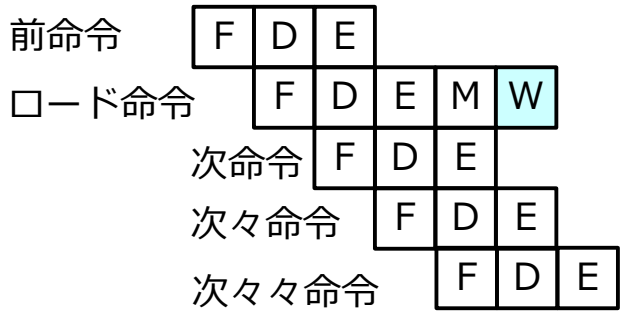


・ロード命令は、Eステージでメモリのアクセス先アドレスを計算して、
Mステージでメモリをリードして、リード・データをWステージで
デスティネーション・レジスタに格納する。

(2) ストール制御：ロード命令のディスティネーション・レジスタを後続の命令が参照する場合







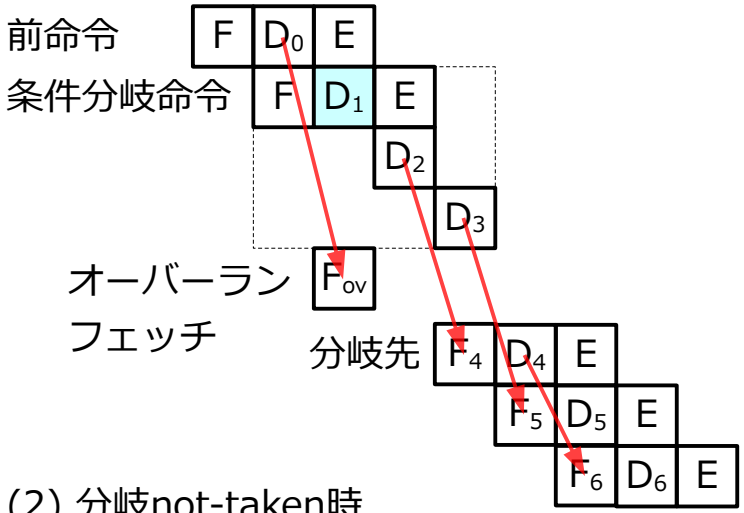
・ロード命令のデスティネーション・レジスタを後続命令が参照する場合は、
ロード命令のWステージと、当該後続命令が参照するステージが重なるまで
その後続命令のDステージをストールさせる。
・Wステージではまだデスティネーション・レジスタへ書き戻す直前であり
後続命令でそのデータを参照するステージはレジスタ本体を参照せず、
Wステージで書き戻す直前の値を参照する（レジスタ・フォワーディング）。

各命令のパイプライン制御(3)

mmRISC

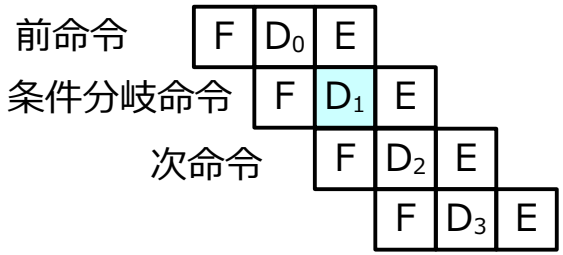
●条件分岐命令　BEQ/BNE/BLT/BGE/BLTU/BGEU

(1) 分岐taken時



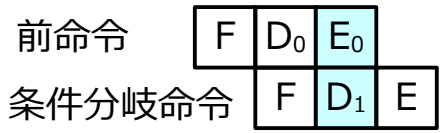
・条件分岐命令は、前命令までの実行結果が反映された汎用レジスタの大小判定で分岐するかどうかをD_1で判断する。
・条件分岐命令のD_1で分岐する判断をしたら、その後の動作は無条件分岐命令と同様である。分岐taken時の条件分岐命令は3ステップで実行される。

(2) 分岐not-taken時

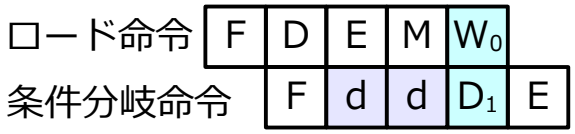


・条件分岐命令のD_1で分岐条件不成立の場合は、通常命令と同様にD_1で2命令先の命令フェッチを起動し、1ステップで終了する。

(3) ストール制御：前の命令のデスティネーション・レジスタを条件分岐命令が参照する場合



・直前命令のE_0のデスティネーション・レジスタを条件分岐命令のD_1が参照する場合は、D_1時点ではデスティネーションレジスタに書き込まれていないので、書き込まれる前のALU出力を参照する（レジスタ・フォワーディング）。

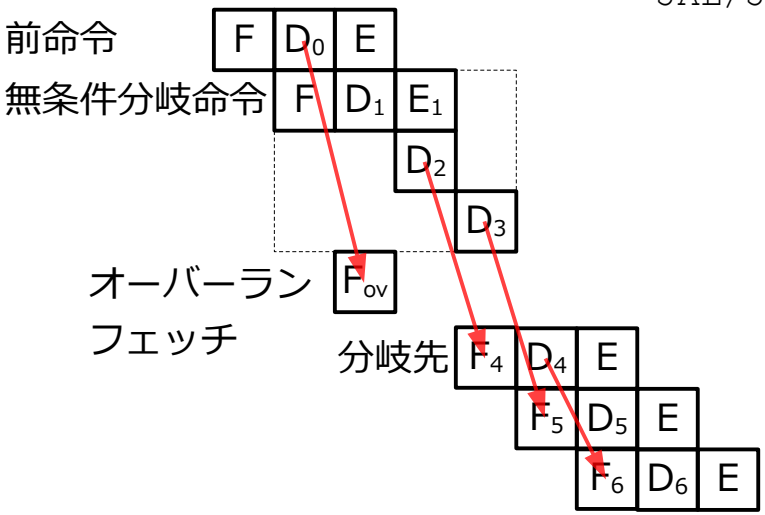


・ロード命令のデスティネーション・レジスタを条件分岐命令のD_1が参照する場合は、D_1をストールしてW_0ステージにおけるレジスタに書き戻す直前の値を参照する。

各命令のパイプライン制御(4)

●無条件分岐命令

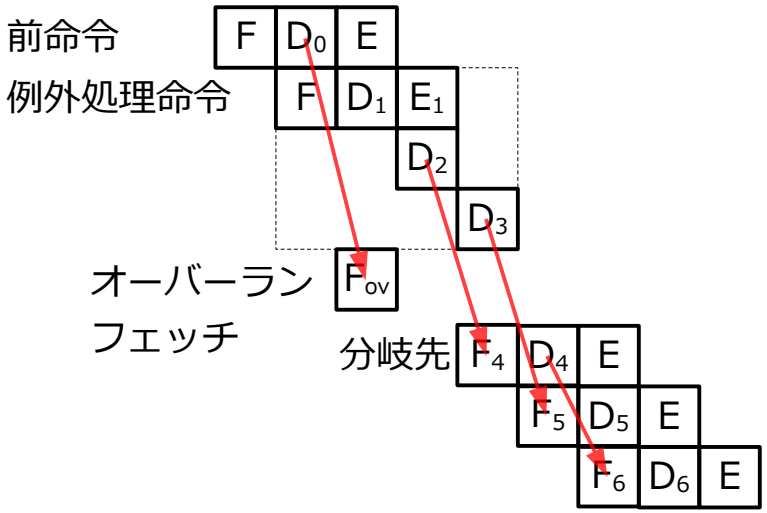
JAL/JALR



- 無条件分岐命令直前の命令のD_0がその2命令先のフェッチF_{OV}を起動しているが、それはオーバーラン・フェッチとなり無視。
- 無条件分岐命令のIDステージは3ステップ構成。
- 無条件分岐命令の最初のD_1は命令フェッチを起動しない。
  ただしD_1後のE_1で分岐先アドレスを計算。
  E_1で計算した分岐先アドレスからD_2が分岐先命令のフェッチF_4を起動。
  D_3がその次の命令のフェッチF_5を起動する。
- 分岐先の命令からは通常通り動作。

●例外処理関連命令

EBREAK/ECALL/MRET



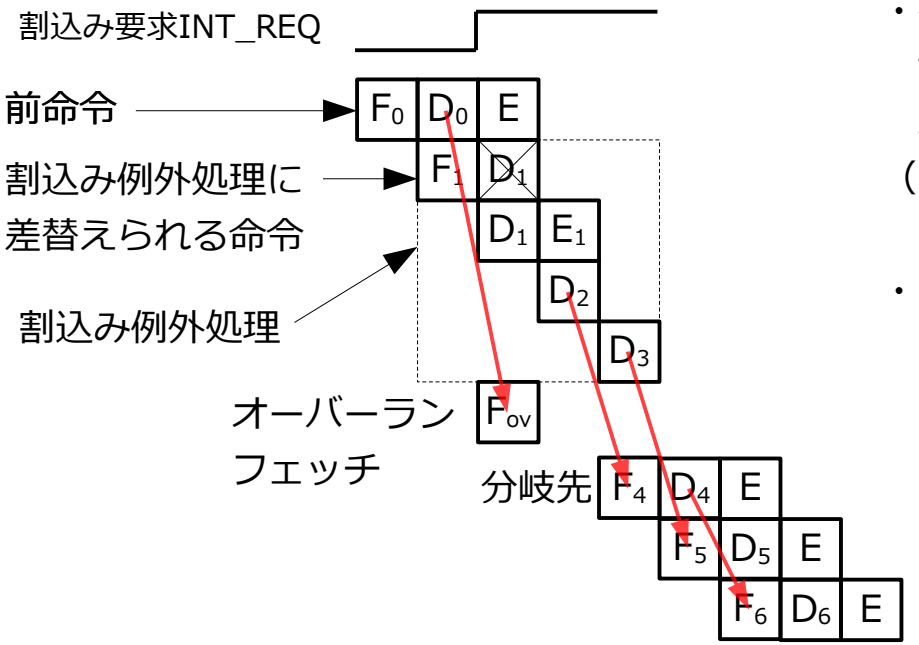
- 基本的に無条件分岐命令と同様。

- EBREAK・ECALL命令
  E_1：CSRのMSTATUS/MEPC/MCAUSE/MTVALを更新
  分岐先：CSRのMTVECとMCAUSEにより決定

- MRET命令
  E_1：CSRのMSTATUSを更新
  分岐先：CSRのMEPCが示すアドレス

各命令のパイプライン制御(5)

●割込み例外処理



・各命令の最初のDステージにおいて、割込み要求信号INT_REQをサンプリングして、アサートされていたら、その命令のデコード以降を割込み例外処理のシーケンスに置き換える。
（フェッチしていた命令コードは捨てる。）

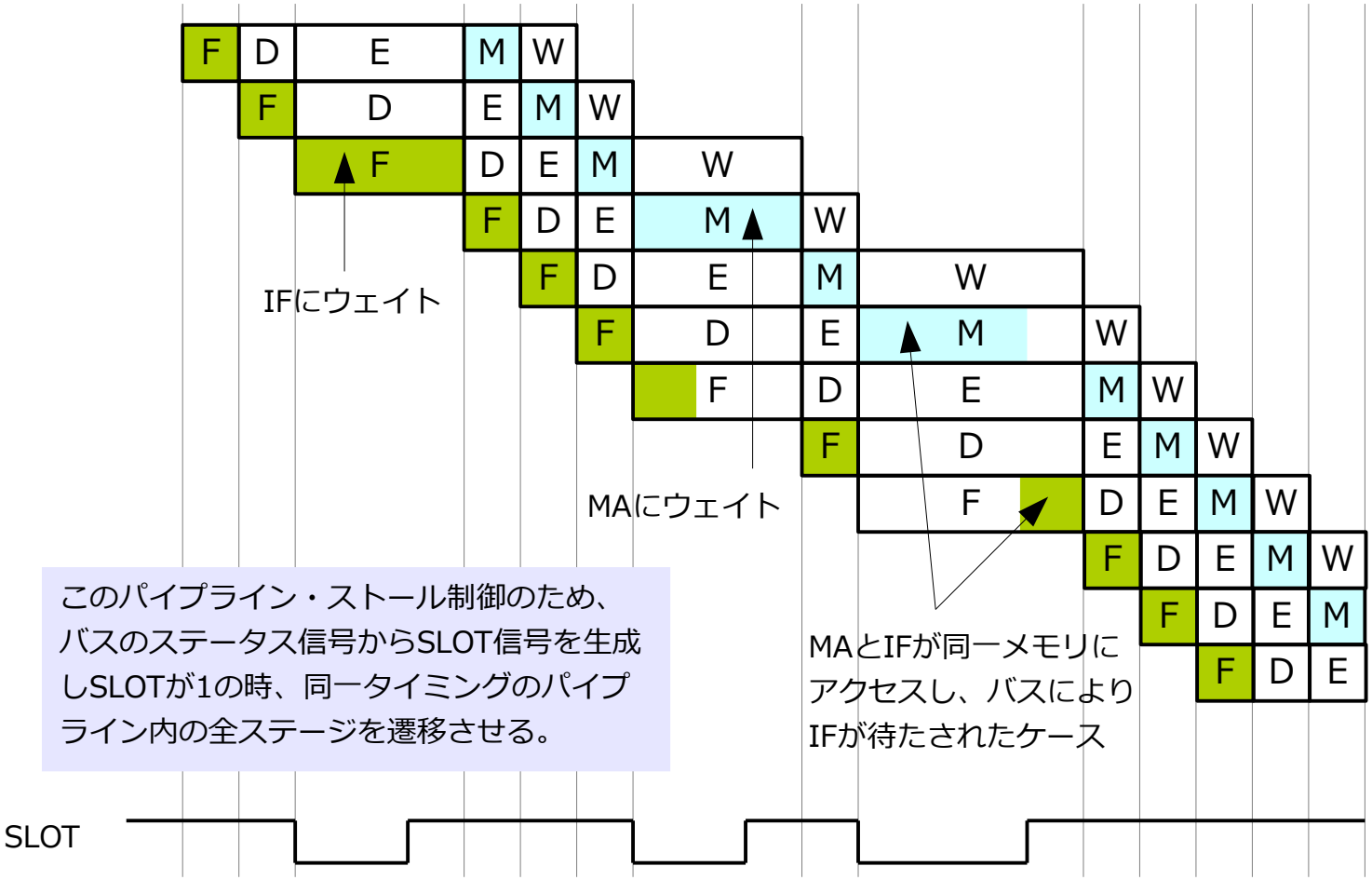
・割込み例外処理のパイプラインは基本的に無条件分岐命令と同様。
E_1：CSRのMSTATUS/MEPC/MCAUSE/MTVALを更新
分岐先：CSRのMTVECとMCAUSEにより決定

パイプライン・ストール(1)

パイプライン・ストール：何らかの条件で、命令のパイプラインの流れが一時停止すること

●ストール要因(1)

命令フェッチやデータ・アクセスのためのメモリ・アクセスにウェイトが入った時

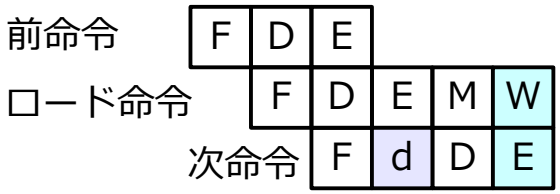


※CPUによっては、パイプラインのストール要因が発生しても、パイプラインに入った命令ステージのうち可能なものはできるだけ先に進めるタイプのマイクロ・アーキテクチャを採用するものもある。mmRISCでは、単純化のためストール要因が発生したら、同時刻の全てのステージをストールさせている。

●ストール要因(2)

先行命令が更新しようとしている汎用レジスタの内容を、後行命令が参照しようとしている時

(a) ロード命令のデスティネーションを後続命令のEXステージが参照するケース



ロード命令のWBステージが
後続命令でそのロード結果を参照するステージと
同一タイミングになるまでストール

```
lw   x9, 0(x8)
add x11, x10, x9
```

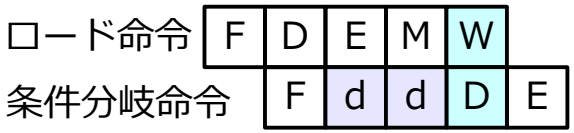
```
lw   x9, 0(x8)
lw   x10, 0(x9)
```

```
lw   x9, 0(x8)
sw   x9, 0(x10)
```

```
lw   x9, 0(x8)
add x9, x10, x11
```

ロード命令のデスティネーションが
後続命令のデスティネーションと
競合してもその後続命令はストール
(そうしないと、実行結果の順序が入れ替わってしまう。)

(b) ロード命令のデスティネーションを後続命令（条件分岐命令）のIDステージが参照するケース



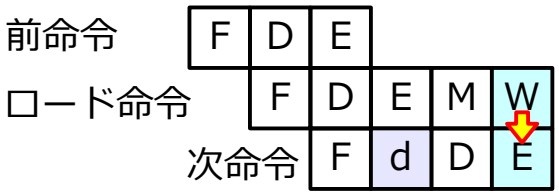
ロード命令のWBステージが
後続命令でそのロード結果を参照するステージと
同一タイミングになるまでストール

```
lw   x9, 0(x8)
beq x9, x10, loop
```

d　IDステージがストール中は、制御信号を出力しない（無効化）

レジスタ・フォワーディング：パイプライン・ストールをできるだけ減らすため、
デスティネーション・レジスタがライトされる前に後続命令がその値を参照できるようにする。

●WBステージからEXステージへのレジスタ・フォワーディング



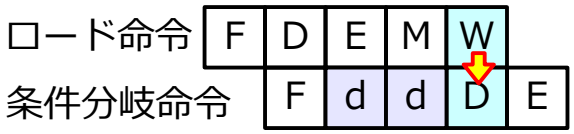
ロード命令のWBステージが終わらないと
デスティネーション本体が更新されないので
WBステージで更新しようとしている値を
後続命令のEXステージが参照できるように
専用パス経由でフォワーディングする。

```
lw   x9, 0(x8)
add  x11, x10, x9
```

```
lw   x9, 0(x8)
sw   x9, 0(x10)
```

```
lw   x9, 0(x8)
lw   x10, 0(x9)
```

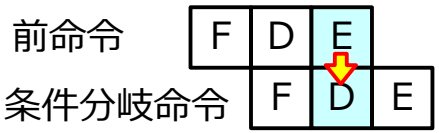
●WBステージからIDステージへのレジスタ・フォワーディング



ロード命令のWBステージが終わらないと
デスティネーション本体が更新されないので
WBステージで更新しようとしている値を
後続命令のIDステージが参照できるように
専用パス経由でフォワーディングする。

```
lw   x9, 0(x8)
beq  x9, x10, loop
```

●EXステージからIDステージへのレジスタ・フォワーディング

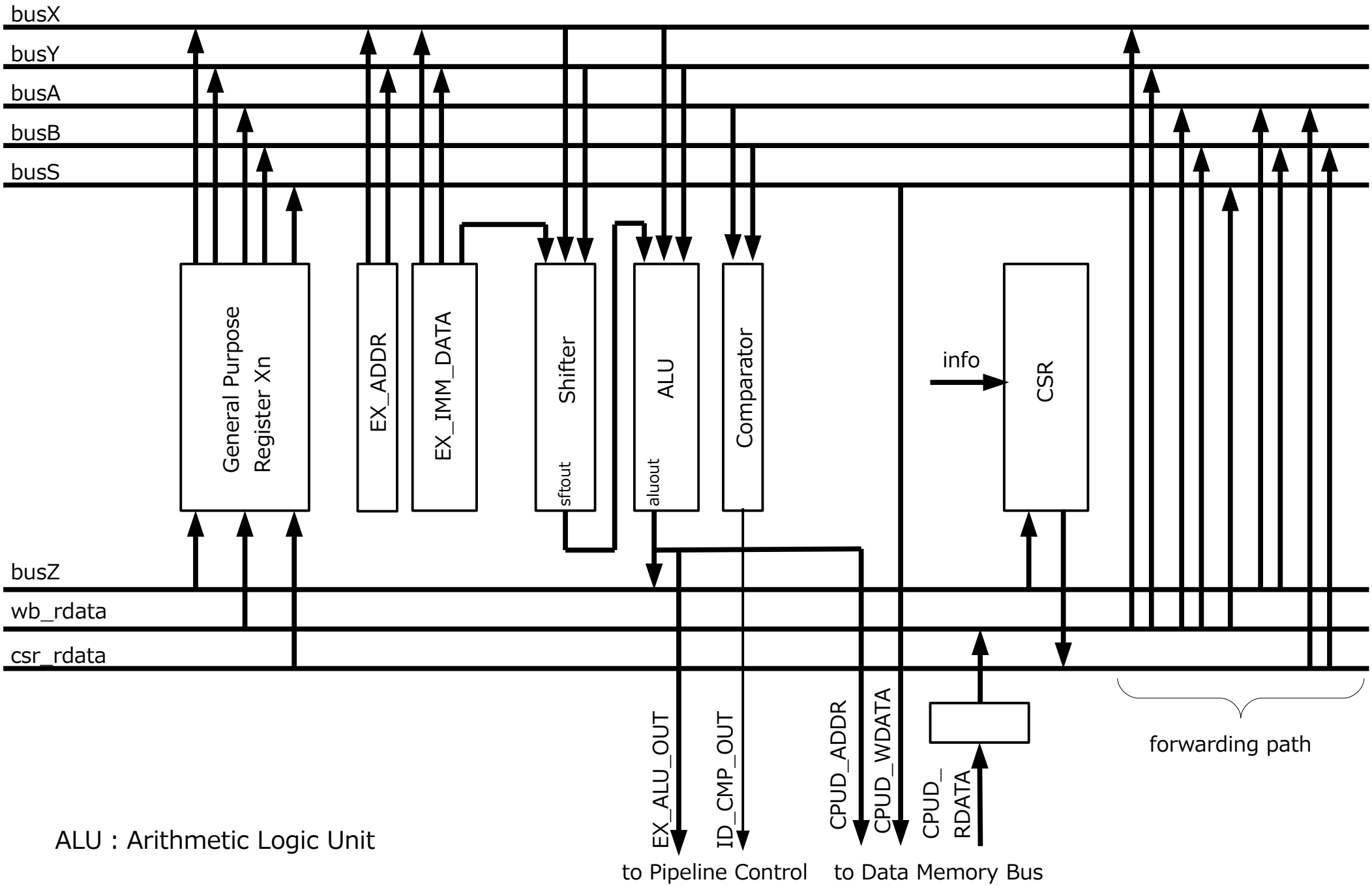


ALU命令のEXステージが終わらないと
デスティネーション本体が更新されないので
EXステージで更新しようとしている値を
後続命令のIDステージが参照できるように
専用パス経由でフォワーディングする。

```
add  x10, x9, x8
beq  x10, x11, loop
```

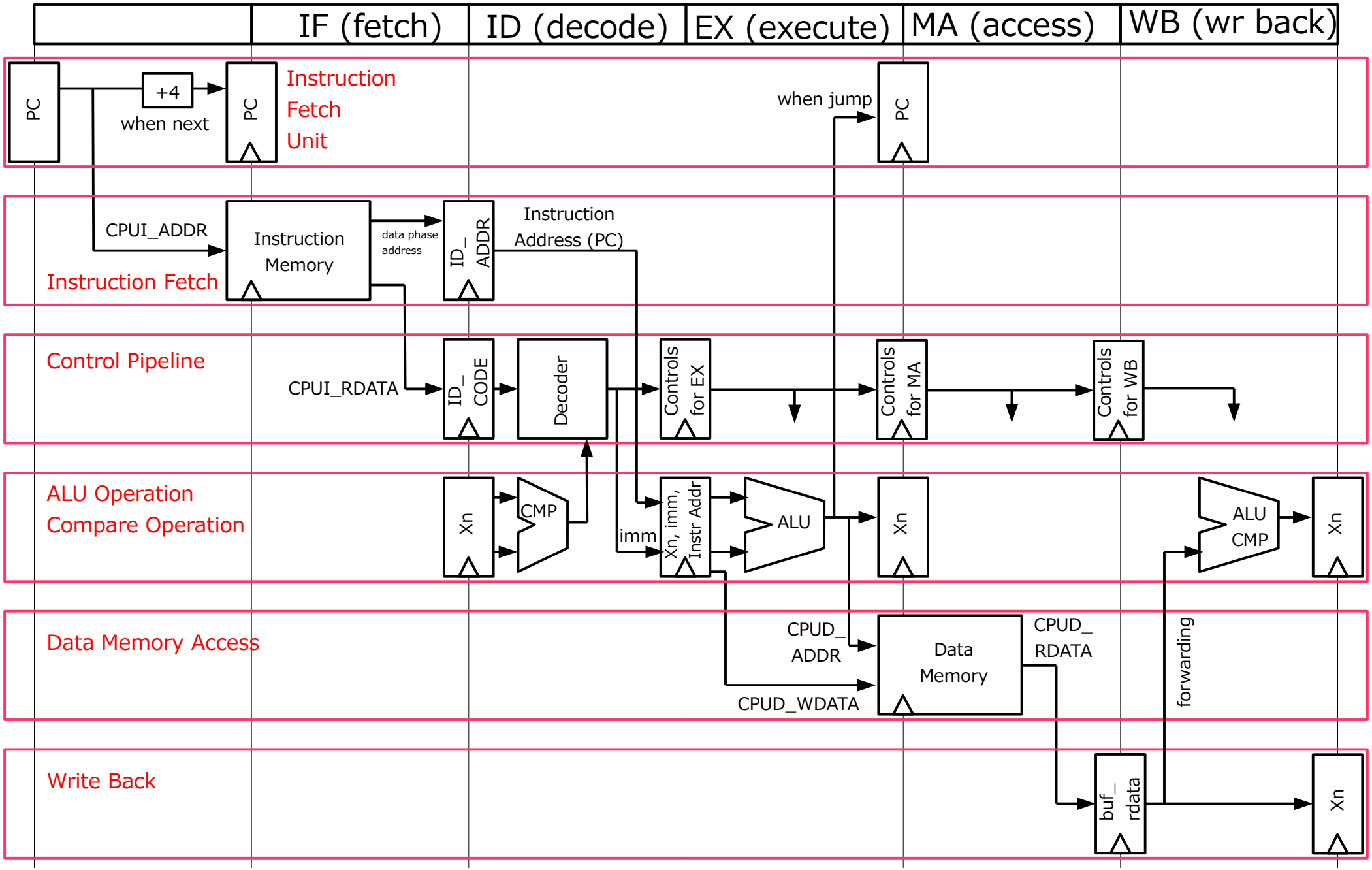
```
csrrw x9, MEPC, x8
beq   x9, x10, skip
```

データパスの構造



ALU : Arithmetic Logic Unit

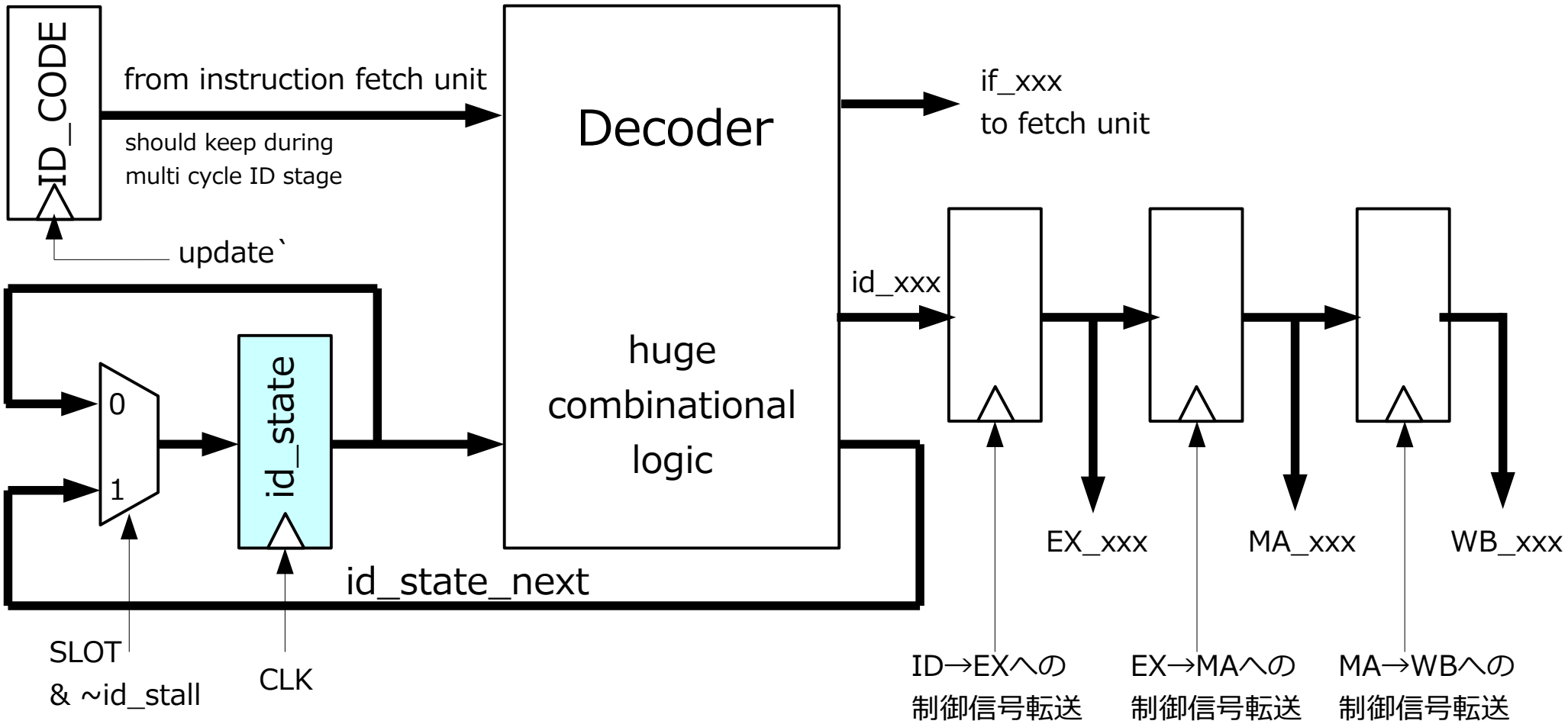
単一命令内の制御



※一つのレジスタに複数ルートから書き込まれる時のセレクタは省略　※フォワーディングは全てを表現していない。

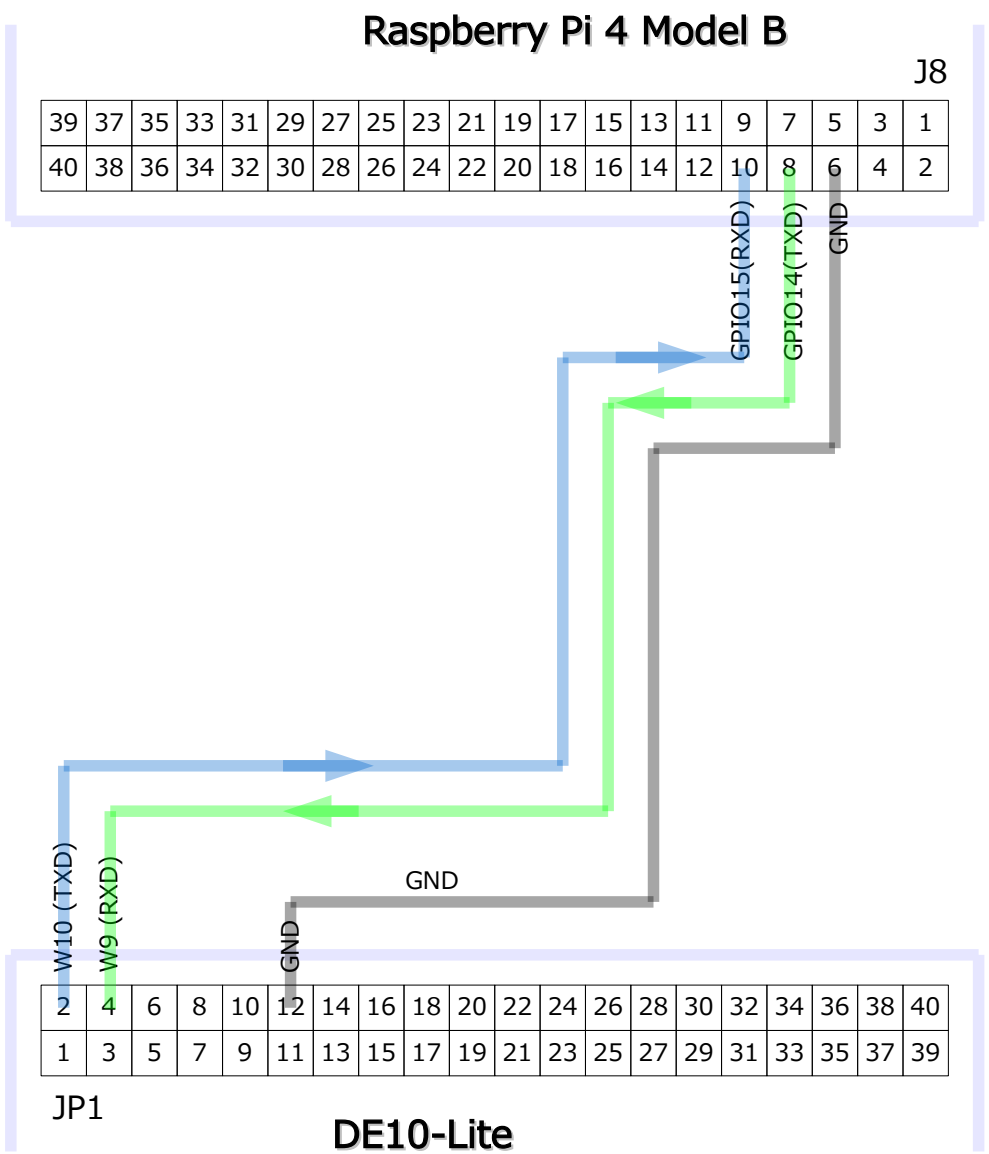
●id_state[7:0] : IDステージ・シーケンサのステート

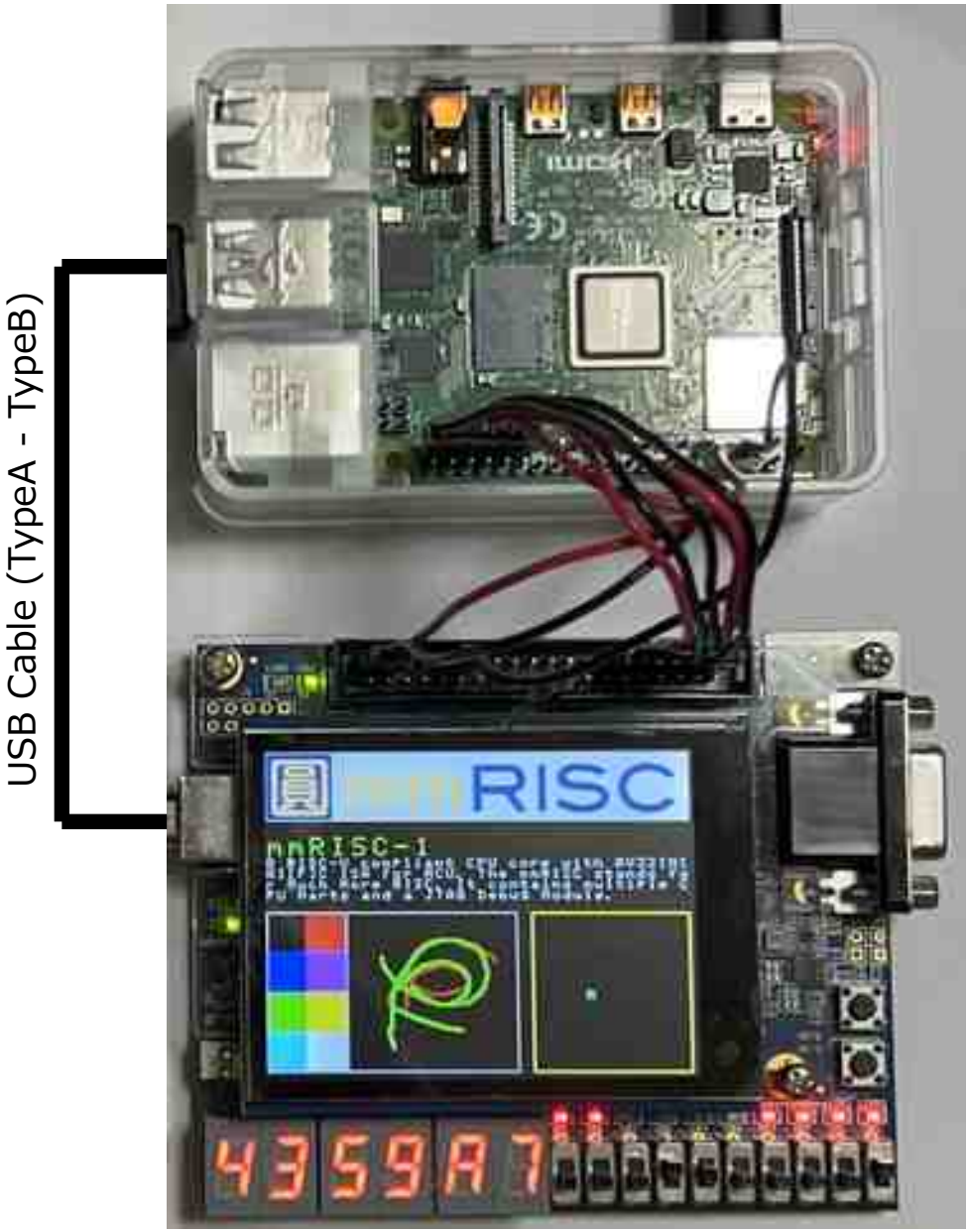
id_state[7:0]	State Description	Note
8'h00	Idle	Reset State
8'h01 ~ 8'h02	Reset Sequence	First Jump
8'h10 ~ 8'h1f	Instruction, Exception	Multi Cycle Supported



接続方法：Raspberry Pi版（mmRISC-0）

●接続方法





※FPGAコンフィグ時は、FPGA基板をPCに接続する

mmRISCセミナ

mmRISC オンライン・セミナを開催

●オープンソースCPU RISC-V入門（2022/9/10～ 2日間コースで実施）
（ZEPエンジニアリング：https://www.zep.co.jp/mmaruyama/web_seminar/risc-v/）
【概要】

- マイコンの基礎
- RISC-Vオープンソース命令セットとアーキテクチャの基礎
- mmRISC-0/mmRISC-1 CPUコアとそのシステムの論理設計
- ソフトウェア開発環境の構築と使い方
- FPGAへの実装
- デバッグ方法

【提供教材】

- FPGAボード
- カラーLCDタッチパネル
- 開発環境用Raspberry Pi
- PC用USB-JTAGインタフェース基板

●上記セミナの資料・教材・動画を提供中
（ZEPエンジニアリング：https://www.zep.co.jp/mmaruyama/movie/z-riscv-on1/）

セミナの概要

マイコンの基礎から、オープンソース命令セットやアーキテクチャの基礎、RISC-V CPUコアと
そのシステムの論理設計、ソフトウェア開発環境の構築と使い方、デバッグ方法、FPGAへの実装と活用まで

●マイコンの基礎と、オープンソース命令セットのRISC-V登場の意味を理解

●マイコン・システムのバスとメモリを理解し、論理シミュレーションで動作確認

●RISC-Vアーキテクチャの詳細と、命令セット仕様、例外処理、割込み仕様を理解

●RISC-Vのソフトウェア開発環境における具体的な開発方法と、実際のアプリケーション例とそのAPI、さらにリアルタイムOS FreeRTOSの基礎を学ぶ

●RISC-V CPUコア内部論理のパイプライン制御方式とその論理設計手法およびデバッグ・サポート機能の詳細を学び、論理シミュレーションで動作確認

●RISC-Vマイコン・システムの構成を学び、実際にFPGAの上に構築し、ソフトウェア開発環境からプログラムを動作させデバッグする

セミナで配布する教材

必要な基礎知識

- 簡単な論理回路
- 基本的なVerilog HDLの知識
- 基本的なLinuxコマンド
- C言語の基礎

セミナ目次

セミナで使用するハードウェア

#	配布	名　　称	外　　型	個数	説　　明
1		Window 10/11 (64bit版) PC (ﾃﾞｽｸﾄｯﾌﾟならﾓﾆﾀ・ｷｰﾎﾞｰﾄﾞ・ﾏｳｽも)	ー	1	FPGAの合成とｺﾝﾌｨｷﾞｭﾚｰｼｮﾝ用 USB端子x1ポートは必須
2		Raspberry Pi用周辺機器 モニタ・キーボード・マウス	ー	1式	モニタはHDMI入力 キーボード・マウスはUSB キーボードはUS/JISいずれも可
3	○	Raspberry Pi4 model B スタータ・キット（ACアダプタ、 microHDMIケーブルなど付属）		1式	RISC-V開発環境として使用 (各自のWinPCへの無償開発環境構築は トラブル多く、統一性が取れないため)
4	○	Raspberry Pi用micro SDXC		1	64GB、必要な環境インストール済
5	○	Terasic DE10-Lite FPGAボード		1	USBケーブル(A - B)付き FPGA：MAX10 10M50DAF484C7G 64MB SDRAM, 3D加速度センサ(I2C)
6	○	Adafruit 2.8" TFT Touch Shield v2 - Capacitive Touch Panel		1	Arduinoシールド、FPGA基板に搭載可 表示制御はSPI、タッチ制御はI2C
7	○	ZEPオリジナル OpenOCD JTAG Debugger		1	PC上にRISC-V開発環境を構築した際、 PCとRISC-VのJTAG端子を接続
8	○	USBケーブル(A – mini B)		1	JTAG DebuggerとPCとの接続用
9	○	メスーメス・ジャンパ 15cm		10	ラズパイ or DebuggerとFPGAを接続

mmRISCの今後

Road Map and Status

64bit Linux

mmRISC-2

RV64G

Working Linux bench
Planning CPU Core IP

Generic MCU

mmRISC-1

RV32IMAFC

CPU Core IP
FPGA System (Peripherals Added)
Working Efabless SKY130 Caravel GDSII
Planning 2-wire cJTAG / 1-wire Debugger
Planning Commercial MCU

Simple Core for Education

mmRISC-0

RV32I

CPU Core IP
FPGA System (Peripherals Added)
Public Seminar

mmRISCシリーズの今後

●mmRISC-1
- 2-wire cJTAG
- 1-wire Debug Interface
- 商用MCUチップへの無償提供と実装
- Efabless Skywater 130nm Caravel Chipへの実装

●mmRISC-2
- 64bit化
- 倍精度浮動小数点
- S-mode, U-mode
- Linux対応

I will continue to develop the mmRISC series as an individual activity.