|  |
| --- |
| **ASCA16機能仕様書** |
|  |
| **Rev 0.00** |
| **2020/10/05** |
| 杉浦 創 |

**版歴**

|  |  |  |  |
| --- | --- | --- | --- |
| 日付 | Rev | 名前 | コメント |
| 2020.10.05 | 0.00 | 杉浦 | 下書き |
|  |  |  |  |

目次

[1 概要 5](#_Toc53765355)

[1.1 RISCアーキテクチャASCA16 5](#_Toc53765356)

[1.2 ASCA16と実装 6](#_Toc53765357)

[2 ASCA16コアアーキテクチャ 7](#_Toc53765358)

[2.1 命令実行ブロック 7](#_Toc53765359)

[2.2 レジスタ 8](#_Toc53765360)

[2.3 命令セット 9](#_Toc53765361)

[3 ASCA16COREブロック 10](#_Toc53765362)

[3.1 外部端子 10](#_Toc53765363)

[3.2 基本構成と命令実行 12](#_Toc53765364)

[3.3 フォワーディングとNOP命令 15](#_Toc53765365)

[3.4 メモリスタックと関数 18](#_Toc53765366)

[3.5 例外コントローラとシステムバスI/F 20](#_Toc53765367)

[3.6 システムバスI/F 22](#_Toc53765368)

[4 命令制御ブロック 23](#_Toc53765369)

[4.1 外部端子 23](#_Toc53765370)

[4.2 命令分岐と分岐予測 25](#_Toc53765371)

図目次

[図 1.1 ASCA16アーキテクチャ 6](#_Toc55241498)

[図 2.1 ASCA16CORE (概要) 7](#_Toc55241499)

[図 2.2レジスタ 8](#_Toc55241500)

[図 2.3 命令セット 9](#_Toc55241501)

[図 2.4 メモリ空間 10](#_Toc55241502)

[図 3.1 ASCA16CORE(詳細) 11](#_Toc55241503)

[図 3.2 外部端子(ASCA16CORE) 12](#_Toc55241504)

[図 3.3 ASCA16CORE (基本構造) 14](#_Toc55241505)

[図 3.4 パイプライン処理(3段) 16](#_Toc55241506)

[図 3.5 フォワーディングとNOP命令 17](#_Toc55241507)

[図 3.6 フォワーディング制御 18](#_Toc55241508)

[図 3.7 NOP命令 19](#_Toc55241509)

[図 3.8 メモリスタック 20](#_Toc55241510)

[図 3.9 関数の実行 21](#_Toc55241511)

[図 3.10 例外コントローラとシステムバスI/F 22](#_Toc55241512)

[図 3.11 システムバスI/F 24](#_Toc55241513)

[図 4.1 命令制御ブロック 25](#_Toc55241514)

[図 4.2 外部端子(instrctl) 26](#_Toc55241515)

[図 4.3 分岐予測ブロック 27](#_Toc55241516)

表目次

[表 3‑1 外部端子リスト(ASCA16CORE) 12](#_Toc55241517)

[表 4‑1 外部端子リスト(instrctl) 26](#_Toc55241518)

# 概要

## RISCアーキテクチャASCA16

本仕様書では教育用RISCアーキテクチャである ASCA16 (Arm-like Simple CPU Architechture 16 bit)とその実装について解説する. ASCA16は1ワード 16ビット , 5ビット固定長の命令をもつRISCアーキテクチャであり, 算術論理演算, 分岐, サブルーチンコールやメモリアクセス命令を揃えている. 基本的な命令の組み合わせで複雑な処理を行うRISCアーキテクチャを採用しているため, ASCA16をマイクロプロセッサとして実装することによって汎用的な処理が可能となる.

本アーキテクチャは以下の2点で強みを持っている.

1) 回路構造がシンプルであること

シンプルな構造のメリットは, 多くの設計者に理解が得られやすく可読性の高いコードであること,

冗長なデータパスが減るため, 回路面積, 製造コスト, 電力消費が小さいことである.

2) コンピュータが持つべき基本的な機能を揃えている

分岐制御, 関数, 例外処理, パイプライン処理といった基本的な技術要素が含まれていることから,

学習者が設計・検証・実装を行うことでコンピュータアーキテクチャを実践的に習得することが可能となっている. またシンプルな構造である ため拡張性に余裕がある.

以下に本アーキテクチャの特徴を示す.

1) ハーバード・アーキテクチャ採用: 命令ROMおよびデータRAMを分離している.

2) 3段パイプライン: ASCA16 命令はFetch/Decode/Execute 3段パイプラインで実行される.

3) 命令制御ブロック(Instctl): パイプラインブロックとは別に, 分岐命令, リンク付き分岐命令によるコンテキストの切り替え, メモリスタックの制御を行う. またレジスタ・アクセスの競合を防ぐため, フォワーディング制御を行う.

4) レジスタ: 14個の汎用レジスタおよびサブルーチンや割込みハンドラで用いるリンクレジスタ(LR), スタックポインタ(SP), 演算ステータスを保持するプログラムステータスレジスタ(PSR) を用いる

5) 5ビット命令固定長: 命令を固定長にすることでシンプルな回路構造を可能としている.

## ASCA16と実装

ASCA16アーキテクチャは命令を実行するコアを提供し, またコアに加え例外処理ブロック,

システムバスI/Fを拡張することで汎用プロセッサとして実装される. 実装を以下に示す.

Exception Controllerは命令ROMから読み込まれる命令を監視し, ASCA16外部からの割込み要求に応じてInstrctlにIrq制御信号を出力すると同時にInstrctlに対する命令を出力する. Instrctlは命令の分岐を制御するブロックであるが, Irqがアサ―トされると今まで実行していた命令を中断し, Exception Controllerからの命令に従って, コンテキストを切り替える.

System Bus/IFはSoCサブシステムとのインタフェースであり, ASCA16コアと内部のシステムバスの境界にあり, サブシステムへの入出力制御を行う. サブシステムのI/Oに対する出力データついてはRAM(メモリマップトI/O)とI/O両方に書き込みを行うライトスルー方式を採用し, I/Oからの入力データについてはExecuteブロックのレジスタに読み込む.

時計 が含まれている画像

自動的に生成された説明

図 1.1 ASCA16アーキテクチャ

# ASCA16コアアーキテクチャ

本章ではASCA16コアの構成および命令について述べる. 具体的には命令実行ブロック, 命令制御ブロックおよびレジスタ, 命令セットについて解説する.

## 命令実行ブロック

ASCA16コアの概観を以下に示す. 命令実行ブロックは Fetch/Decode/Execute の3つのブロックから構成されており, パイプライン処理が可能となっている. 命令ROMから読み込んだ命令は

1. Instrctl に入力
2. Fetch に入力の2経路に分岐するが,

1)では命令分岐制御を実行し, PCの設定, フォワーディング, 命令フラッシュ, リンクレジスタの制御を行う. 2)では読み込んだ命令を実行する. Fetch/Decode/Executeブロックはそれぞれ命令処理ステージごとに処理がなされ,それぞれの出力はシステムクロックに同期し非同期でリセットされる.

スクリーンショット が含まれている画像

自動的に生成された説明

図 2.1 ASCA16CORE (概要)

## レジスタ

ASCA16コアは16ビット幅の汎用レジスタが14個と特殊レジスタ4個から構成される.

R0～R13は汎用レジスタであり, 即値命令で使用可能なものはR0～R7である.

R14はリンクレジスタ(LR)であり, サブルーチンや例外処理の際に復帰先の命令アドレスを格納する.

R15はプログラムステータスレジスタ(PSR)であり, 演算結果のフラグとして用いる.

CMP命令の結果が負ならPSR[15] (N) を1, オーバフローならPSR[14] (V) を1,

零ならPSR[13] (Z) を1, 非零ならPSR[12] (NZ) を1 とする.

テーブル

自動的に生成された説明

図 2.2レジスタ

## 2.3 命令セット

ASCA16は以下の命令をサポートしている. 命令はNOP命令, 即値命令, 算術論理命令, メモリアクセス命令, BLX(BL)命令, 条件分岐命令の6種類に分類される. 命令コードはop[15:11] の5ビット固定, オペランドop[10:0] は即値あるいはレジスタ番号を指定する. (予約コードは省略)

テーブル が含まれている画像

自動的に生成された説明

図 2.3 命令セット

## 2.4 メモリ空間

ASCA16を実装するにあたってメモリの実装について述べる.

・ROM

命令を格納する.

・RAM

スタック領域 ヒープ領域は動的に確保されるメモリ I/O空間

ダイアグラム が含まれている画像

自動的に生成された説明

図 2.4 メモリ空間

# ASCA16COREブロック

本章ではASCA16COREについて解説する.

トップモジュールと同階層には命令ROM, データRAM, 例外処理コントローラ, システムバスI/Fが存在しASCA16を構成しているが, 　ASCA16COREは各種演算・制御を行う.

ダイアグラム, 概略図

自動的に生成された説明

図 3.1 ASCA16CORE(詳細)

## 外部端子

ASCA16COREの外部端子を以下に示す.

缶切り, ツール, ケーブル, 三脚 が含まれている画像

自動的に生成された説明

図 3.2 外部端子(ASCA16CORE)

表 3‑1 外部端子リスト(ASCA16CORE)

|  |  |  |  |
| --- | --- | --- | --- |
| 端子名 | I/O | 駆動元・先 | 説明 |
| Clk | I | ― | 立ち上がりエッジで動作するシステムクロック |
| rst\_n | I | ― | 立下りエッジで動作する非同期リセット |
| op[15:0] | I | ROM | 命令ROMから出力される命令 |
| ir\_in[15:0] | I | 例外コントローラ | 例外コントローラから出力される命令 |
| ir\_addr[15:0] | I | 例外コントローラ | 割込みハンドラの命令アドレス |
| irq | I | 例外コントローラ | 例外コントローラから出力される例外制御信号 |
| ram\_in[15:0] | I | RAM | CPUコアに読み込まれるRAMデータ |
| sysbusif\_in[15:0] | I | システムバスI/F | システムバスから読み込まれるデータ |
| pc\_out[15:0] | O | ROM | ROMへ出力するPC |
| ir\_out[15:0] | O | 例外コントローラ | ROMから読み込んだ命令を例外コントローラへ出力 |
| ram\_data[15:0] | O | RAM | RAMへ出力するデータ |
| ram\_wen | O | RAM | RAMへ出力するイネーブル信号“High”で書き込み可 |
| ram\_addr[15:0] | O | RAM | RAMへ出力するアドレス |
| sysbusif\_out[15:0] | O | システムバスI/F | システムバスへ転送するデータ |
| sysbusif\_addr[15:0] | O | システムバスI/F | システムバスへ転送するアドレス |
| sysbusif\_wen | O | システムバスI/F | システムバスのイネーブル信号“High”で書き込み可 |

・すべてのレジスタは単一クロックclk に同期し, rst\_nによって非同期にリセットされる.

・ROM入出力はPC(次プログラムアドレス)をROMへ出力し, 命令op[15:0] を読み込む

・RAM入出力は演算結果のデータ転送, 演算で利用するデータのロード, スタック領域への読み書き込み, メモリマップトI/Oへの出力で利用される.

・Exception Controller からの入力は割込みハンドラへのコンテキスト切り替え命令ir\_in[15:0], 割込み制御信号irqを入力とし, 命令op[15:0] を出力している

・System Bus I/F

## 基本構成と命令実行

ASCA16COREブロック図を以下に示す. 命令ROMから読み込んだ命令は3ステージで処理される.

ダイアグラム, 概略図

自動的に生成された説明

図 3.3 ASCA16CORE (基本構造)

1. Fetchブロック

ROMから読み込まれた16ビット命令op[15:0]を命令コードopcode[4:0],

レジスタ選択信号nREGA[3:0], nREGB[3:0], と即値データ信号opdata[7:0] に分離する.

1. Decode ブロック

Executeブロックから読み込んだ汎用レジスタR0～R13を読み込み, nREGA[3:0] とnREGB[3:0]によってオペランド対象のレジスタ値を選択して出力する. opdata[7:0] は出力先で16ビットに拡張されるが, 符号拡張されるか, 0拡張されるかを命令コードopcode[4:0] によって選択する.

1. Execute ブロック

RAMからのデータram\_in[15:0], レジスタ値regB,A[15:0], 即値opdata[7:0] の各データを用いて演算を実行する. 演算(データに対して何もしない演算も含む)および, 演算結果はopcode[4:0] により選択されて各レジスタおよびデータRAMに格納される.

制御信号とデータ信号について説明する.

1. 制御信号

ROMから読み込まれた命令から制御信号がデコードされる. さらにInstrctlブロックではROMから読み込んだ命令がBEQなどの分岐命令の場合, 命令分岐処理を行うために, nop\_en, fwd\_en, lr\_recoven, lr\_seten信号を各ステージのデータ選択制御として用いる.

・op[15:11]: opcodeをデコードしてALU演算・オペランド選択・ビット拡張に用いる.

・op[10:3] : op[10:7], op[6:3] (即値命令はop[10:8]) は演算に用いるExecuteブロックのレジスタを選択する信号である. また即値命令は

1. データ信号

ROMから読み込まれた命令が即値を含む場合はこれをデータ信号として演算に利用する. さらにExcuteブロックのレジスタ, メモリやシステムバスI/Fのデータ入力をデータ信号として演算に用いる.

演算結果は命令に従ってレジスタ, メモリ, システムバスI/Fそれぞれに格納される.

ASCA16COREはFetch/Decode/Executeの3段パイプラインにより命令処理を行う.

例えばR0, R1 に即値16’d1, 16’d2をロードした後でこれらをADDした場合は以下のようになる.

(赤色: Fetch出力側　緑色: Decode出力側　青色: Execute出力側)

カレンダー が含まれている画像

自動的に生成された説明

図 3.4 パイプライン処理(3段)

## フォワーディングとNOP命令

同一レジスタに対するRead-After-Write命令を検知するとフォワーディングを有効にするfew\_enがHighとなり, 1サイクル前の命令によって書き込まれたレジスタを読み込むようにした.

また, 条件分岐制御では分岐予測を行っているため, 予測が外れた場合や例外処理で命令実行を停止したいときにNOP命令を有効にする. nop\_enがHighとなりFetchブロックのopcode[4:0] を5’b00000にジャックすることで命令パイプラインをフラッシュする.

ダイアグラム, 概略図

自動的に生成された説明

図 3.5 フォワーディングとNOP命令

フォワーディング制御の波形を以下に示す.

LDRL命令によりレジスタR0, R1, R2 に対して16’d1, 16’d2, 16’d3 を書き込んだ後に

ADD命令でレジスタR1, R2 を加算している. 一連の命令はR2に対してWriteした直後のサイクルでReadするため, ADDはWriteした後のR2の値を選択する必要がある.

fwd\_enがHighになることでADD命令直前のサイクルで書き込まれたR2 を選択することでRead-After-Writeに起因するレジスタ競合を回避する.

(赤色: Fetch出力側　緑色: Decode出力側　青色: Execute出力側)

グラフ が含まれている画像

自動的に生成された説明

図 3.6 フォワーディング制御

NOP命令実行時の波形を以下に示す.

CMP実行の結果 PSR[13](Zeroフラグ) が1 となり, 次サイクルのBLT(Branch Less Than) 命令の条件がFalseになるため, 分岐予測(4.2節)によってあらかじめ読み込んでいた分岐先の命令ADDI, ADDを命令パイプラインからフラッシュしている.

Fetch-Decode間でNOP命令によるジャックがあるため, Fetchブロックの出力のあとで命令がNOPにオーバーライドされ, Decodeステージのopcode[4:0]からNOP命令が出力される.

(赤色: Fetch出力側　緑色: Decode出力側　青色: Execute出力側)

カレンダー

自動的に生成された説明

図 3.7 NOP命令

## メモリスタックと関数

関数を実装するためにはLRにPCの値を退避させ, 他にもPSRと汎用レジスタを退避させる必要があり, データはRAMのスタック領域に格納される. これを制御するモジュール図を以下に示す.

ダイアグラム, 概略図

自動的に生成された説明

図 3.8 メモリスタック

関数呼び出しから復帰までの波形を示す. レジスタR0を引数として参照渡しして, 関数の戻り値をR0に格納する処理を行っている. (赤色: Fetch出力側　緑色: Decode出力側　青色: Execute出力側)

1. BL/BLX命令: 関数の命令が格納されている命令アドレス(0x1131)に分岐し, BLX命令で関数呼び出し元で実行していた命令アドレス(0x1031)に復帰する. 関数を呼び出す際に呼び出し元のPCをLRへの格納を有効にするlr\_setenがHighとなり, set\_lr[15:0] (PC値+1)をLRに書き出す.

逆に関数の処理を呼び出し元に戻す際は, lr\_recovenをHighにしてLRの値をPCに書き出す.

LRはPC+16’d1に上書きされるため, BL/BLX命令ではメモリスタックにLRを退避させる.

また復帰動作は必ずBLXでレジスタにLR(0xF) を指定しなければならない.

1. PUSH/POP命令: 関数が呼び出されると関数で上書きされるレジスタの退避を行う. レジスタをメモリスタックに退避させる場合はPUSHを実行し、関数の呼び出し元に戻る際はPUSHでスタックした逆順にPOPし, レジスタ復帰を行う. なおPUSHはpush\_en, POPはpop\_enがHighの時に有効となる. またBL/BLX命令で関数を呼び出す際は分岐前の命令アドレスを退避させるためにpush\_enをHighにアサ―トし, 呼び出し元に戻る際はpop\_enをHighにアサ―トする.

テーブル

自動的に生成された説明

図 3.9 関数の実行

## 3.5 例外コントローラ

例外コントローラおよびシステムバスI/F との間の入出力データはASCA16COREでは以下のブロック図のように利用される.

ダイアグラム, 概略図

自動的に生成された説明

図 3.10 例外コントローラとシステムバスI/F

制御信号とデータ信号について説明する.

1. 制御信号

ROMから読み込まれた命令から制御信号がデコードされる. さらにInstrctlブロックではROMから読み込んだ命令がBEQなどの分岐命令の場合, 命令分岐処理を行うために, nop\_en, fwd\_en, lr\_recoven, lr\_seten, set\_lr信号を各ステージのデータ選択制御として用いる. また例外処理の際, 例外処理コントローラからir\_in[15:0] とirqがInstrctlブロックに入力され, ROMの例外処理命令に分岐する.

・op[15:11]: opcodeをデコードして利用する.

・op[10:3] : op[10:7], op[6:3] (即値命令はop[10:8]) はExecuteブロックのレジスタを選択する信号である.

1. データ信号

ROMから読み込まれた命令が即値を含む場合はこれをデータ信号として演算に利用する. さらにExcuteブロックのレジスタ, メモリやシステムバスI/Fのデータ入力をデータ信号として演算に用いる.

演算結果は命令に従ってレジスタ, メモリ, システムバスI/Fそれぞれに格納される.

## 3.6 システムバスI/F

システムバスI/Fとのインタフェースの機能について述べる.

ダイアグラム, 概略図

自動的に生成された説明

図 3.11 システムバスI/F

# 命令制御ブロック

本章ではASCA16COREの命令実行制御を行うブロックinstrctlについて解説する.

命令制御ブロックinstrctlを以下に示す. instrctlは以下3個のモジュールから構成される.

1. フォワーディング制御ブロック: fwdctl
2. リンクレジスタ制御ブロック : lnctl
3. 命令分岐制御ブロック : bctl

ダイアグラム, 概略図

自動的に生成された説明

図 4.1 命令制御ブロック

## 4.1 外部端子

instrctlの外部端子を以下に示す.

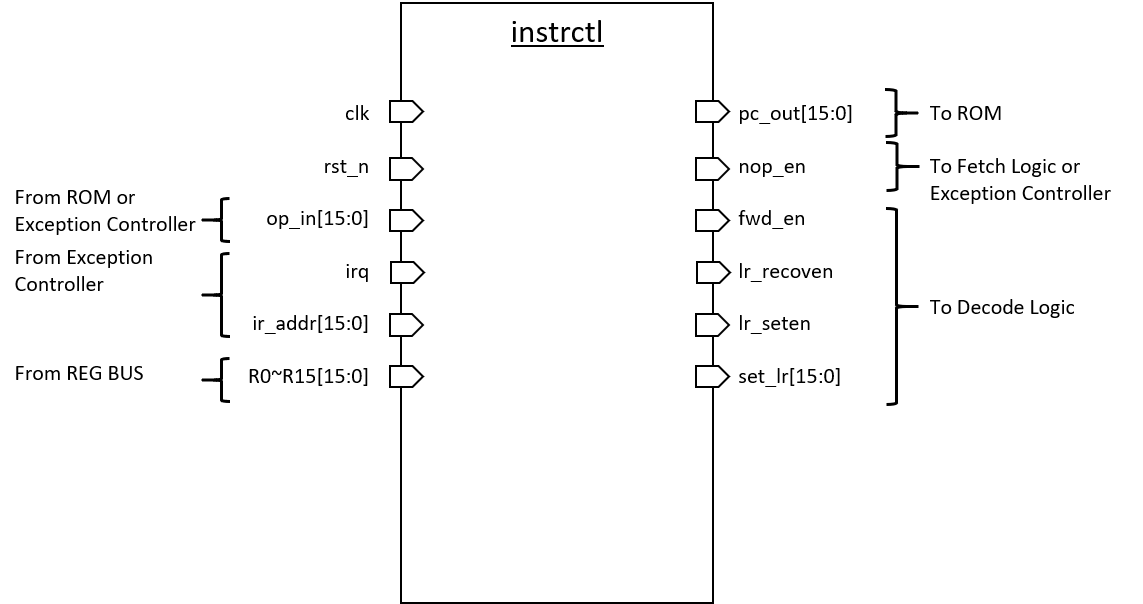


図 4.2 外部端子(instrctl)

表 4‑1 外部端子リスト(instrctl)

|  |  |  |  |
| --- | --- | --- | --- |
| 端子名 | I/O | 駆動元・先 | 説明 |
| clk | I | ― | 立ち上がりエッジで動作するシステムクロック |
| rst\_n | I | ― | 立下りエッジで動作する非同期リセット |
| op[15:0] | I | ROM/  例外コントローラ | 命令ROMあるいは例外コントローラから  出力される命令 |
| ir\_addr[15:0] | I | 例外コントローラ | 割込みハンドラの命令アドレス |
| irq | I | 例外コントローラ | 例外コントローラから出力される例外制御信号 |
| R0~R15[15:0] | I | Execute出力側 | レジスタバス |
| pc\_out[15:0] | O | ROM | ROMへ出力するPC |
| nop\_en | O | Fetch 出力側/  例外コントローラ | NOP命令制御信号 |
| fwd\_en | O | Decode 出力側 | フォワーディングを有効にする制御信号 |
| lr\_recoven | O | Decode 出力側 | 関数からの復帰を有効にする制御信号 |
| lr\_seten | O | Decode 出力側 | 関数呼び出しを有効にする制御信号 |
| set\_lr[15:0] | O | Decode 出力側 | 関数呼び出し時にLRに設定する命令アドレス |

すべてのレジスタは単一クロックclk に同期し, rst\_nによって非同期にリセットされる.

## 4.2 命令分岐と分岐予測

分岐制御ブロック(bpredictor)を以下に示す.

ダイアグラム

自動的に生成された説明

図 4.3 分岐予測ブロック