ICF3-V 命令セットアーキテクチャ仕様(2017 年 12 月 28 日暫定版)

平山 直紀

http://openicf3.idletime.tokyo/

1. 概要

ICF3-V は 1999 年に製品出荷された暗号 LSI ICF3 をベースに作られたマイコン用 CPU のオープンソースハードウェア。この仕様は暫定版で従来の CPU とは違う点を説明するためにあります。このため、まだ追加、変更があります。目的は独特なアーキテクチャーを知ってもらうこと。ICF3-V の特徴は命令コードが直接ハードの制御信号になっていて組み合わせにより多彩な処理ができること。このため少ないゲート数で実装できることがメリット。目標は同じ性能で 1 ランク下のプロセスで製造できるようになること。プロセスレベルの低い生産工場でも製造できることである。

2. メリットとデメリット

少ない制御信号の組み合わせで四則演算などの多彩な処理ができるようなアーキテクチャを私が発明した。この発明は 1999 年にメインフレームの暗号 LSI ICF3 として世界中の銀行などに製品出荷されている。メリットは少ないゲート数で実装できること。デメリットは命令コードの互換性を維持しながらの高性能化が難しいこと。このためモバイルやデスクトップパソコンの用途では厳しい。性能が必要な用途では ARM や Intel の CPU があるので、その辺りは割り切ります。

3. 用途

主な用途は AES 暗号などの共通鍵暗号の秘密鍵を漏洩せずに演算をするサブプロセッサ。公開鍵暗号を演算するには、あまりにも非力なので拡張乗算器を装備することで演算できるようにする。また汎用のマイコン用 CPU としても利用できるように考えたい。

4. 基本仕様

・ハーバードアーキテクチャ風？
・命令コード 1 ワード 32bit 固定長
・すべての命令コードは 1 サイクルで実行される。命令が単体では意味がないものも多い。

・1 サイクルの遅延分岐
・ハードウェアスタック 1 つとソフトウェアスタックの予定
・汎用レジスタ 16 本
　1 サイクルで 2 つのリードと 1 つのライトが同時にできる

・プログラムメモリ　64M ワード(256MB、最大 4G ワード)
　1 サイクルでリードかライトのいずれかができる。
　フラッシュメモリで SRAM にキャッシュさせる。

・データメモリ　最大 32bit×ワード
　プログラムメモリと共有。
　ただし当面、SRAM 16KB - 256KB くらいの実装しかしないかも。

5. レジスタ、フラグ

#	名前	bit 幅	用途
1	A	32	演算レジスタ
2	B	32	演算レジスタ
3	C	32	演算レジスタ
4	D	32	演算レジスタ
5	W	32	演算結果レジスタ
6	CF	1	加算器のキャリーのフラグ
7	ZF	1	ゼロフラグ、D レジスタが 0 のとき 1。
8	I	8	ループ制御カウンタ
9	R0～R15	32	汎用レジスタ 16 本
10	PC	12	プログラムカウンタ
11	RADR	12	サブルーチンコールのためのリターンアドレス
12	CADR	12	圧縮命令用のリターンアドレス

6. 演算レジスタと汎用レジスタ間の転送

汎用レジスタから演算レジスタに転送するのに 1 サイクル
W レジスタから汎用レジスタに転送するのに 1 サイクル

7. キャリーフラグ(CF)とゼロフラグ(ZF)

CF は A レジスタに加算器からの出力をセットするとき(A=ADD)に加算器のｷｬﾘｰ信号値を取り込む。

ZF(ｾﾞﾛﾌﾗｸﾞ)は TESTD 命令が実行されたとき Dﾚｼﾞｽﾀが 0 のとき 1 の値をｾｯﾄし、それ以外のとき 0 の値がｾｯﾄされる。

8. ループ制御レジスタ

レジスタ I はﾙｰﾌﾟ制御専用のレジスタ。

9. 命令コードフォーマット

bit31	bit30	bit29	bit28	bit27	bit26	bit25	bit24
オペコード						ONE	NOT

bit23	bit22	bit21	bit20	bit19	bit18	bit17	bit16
加算器左	加算器右		Rn=W	BC レジスタの入力			

bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
A レジスタの入力		D レジスタの入力		(7 通りの意味)			

bit0-11 は、次の 7 通りの意味

●汎用レジスタのリード/ライト

bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
				Y(リード レジスタ番号)			

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
X(リード レジスタ番号)				Z(ライトレジスタ番号)			

●分岐命令のアドレス(B/CALL/BI/BCF/BZF/BC 命令)

bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
				アドレス上位 4bit			

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
アドレス下位 8bit							

●ループ制御レジスタに即値代入と汎用レジスタへのライト

bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
				即値(上位 4bit)			

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
即値(下位 4bit)				Z(ライト　レジスタ番号)			

●SHL/SHR/ROT 命令

bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
				-	SHL/SHR/ROT		シフト量

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
シフト量(下位 4bit)				Z(ライト　レジスタ番号)			

●B=IMM 命令

bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
				即値(上位 4bit)			

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
即値(下位 4bit)				Z(ライト　レジスタ番号)			

●B=IMM 命令+IMM12 命令

bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
				即値(上位 4bit)			

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
即値(下位 8bit)							

●B=IMM 命令+IMM16 命令

bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
即値(上位 8bit)							

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
即値(下位 8bit)							

10. 特殊な命令コード

EXIT 命令、モード 0

bit31	bit30	bit29	bit28	bit27	bit26	bit25	bit24
EXIT のオペコード　'111111'						モード '00'	

bit23	bit22	bit21	bit20	bit19	bit18	bit17	bit16
終了コード						-	

bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
読み出せる汎用レジスタのマスク値(オプション)							

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
読み出せる汎用レジスタのマスク値(オプション)							

EXIT 命令、モード 1　異常終了(オプション)

bit31	bit30	bit29	bit28	bit27	bit26	bit25	bit24
EXIT のオペコード　'111111'						モード '01'	

bit23	bit22	bit21	bit20	bit19	bit18	bit17	bit16
終了コード						D 情報 3(オプション)	

bit15	bit14	bit13	bit12	bit11	bit10	bit9	bit8
デバッグ情報 1 (オプション)							

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
デバッグ情報 2 (オプション)							

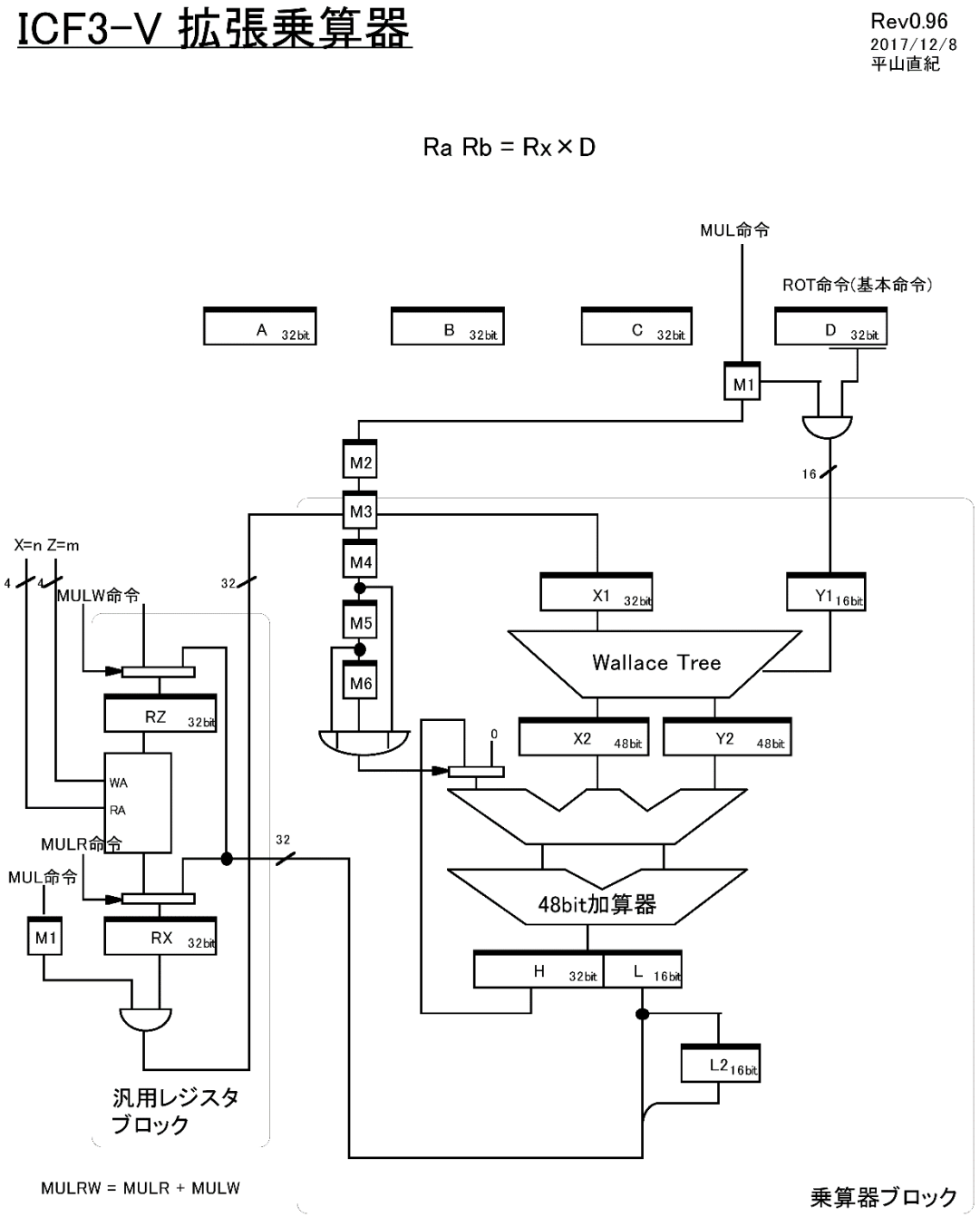
11. ニモニック

ニモニック	bit	値	意味
NOP	26-31	0	何もない
I= n		1	I= n (即値)
B(ラベル)		2	無条件分岐
BCF0(ラベル)		3	BCF0 は CF が 0 なら分岐。
BCF1(ラベル)		4	BCF1 は CF が 1 なら分岐。
BZF0(ラベル)		5	BZF0 は ZF が 0 なら分岐。
BZF1(ラベル)		6	BZF1 は ZF が 1 なら分岐。
BC(ラベル)		7	C レジスタの最下位ビットが 0 なら分岐
BI(ラベル)		8	I=I-1 , I≠0 なら分岐
BIMUL		9	乗算ループ用
BIDIV		10	除算ループ用
BIMUL16		11	BIMULDIV は 16bit 以下の乗算用
CALL(ラベル)		12	サブルーチンコール
RET		13	サブルーチンからのリターン
TESTD		14	D レジスタをテストして ZF をセット
LD		15	B レジスタで示されるアドレスのデータメモリを読む。次サイクルで A=MEM
ST		16	A レジスタをデータメモリにストア
SHL=[n,C] SHR=[n,C] ROT=[n,C]		17	W = D << n W = D >> n W = ROT(D,n) シフト量が 0 のとき C レジスタの値
AND		18	W = A & (ADDR)
OR		19	W = A | (ADDR)
XOR		20	W = A ^ (ADDR)
IMM12		21	即値を 12bit に拡大
IMM16		22	即値を 16bit に拡大(A,D レジスタ不可)
CCALL(アドレス)		23	圧縮命令開始(通常利用することはない)
CMODE		24	圧縮命令モードに移行
CRET		25	圧縮命令終了、次も圧縮命令

NRET		26	圧縮命令終了、次は通常命令
SETK		27	次サイクルで圧縮命令のオペランドをセット。
SETKZ		28	次サイクルで圧縮命令のオペランドをセット。Z を X にシフト
KH		29	オペランドの最上位ビット 2 ビットをセット
拡張領域 MUL		59	拡張乗算器の制御 (乗算は拡張乗算器がなくても可能。乗算を高速化したい場合に拡張するオプション)
拡張領域 MULR		60	
拡張領域 MULW		61	
拡張領域 MULRW		62	
EXIT		63	終了コード、リードできる汎用レジスタの設定
ONE	25		加算器のキャリーIN に 1 を入力する
NOT	24		加算器の右の入力を NOT 演算する
A=[ADD,RX,MEM]	22-23		A,B,C,D レジスタへの入力を選択。この組み合わせとオペコードで四則演算と余が計算できる。添付の「ICF3-V 全体ブロック図」を参照。B=MEM や C=W などは B=LSH と C=RSH の組み合わせが存在しないことを利用する
B=[ADD,LSH,RSH,RX] C=[ADD,LSH,RSH,RX,W]	18-21		
D=[ADD,RY,W0]	16-17		
Rn=W	15		W レジスタの内容を汎用レジスタ n 番に
ADDL=A	14		
ADDR=[B,C,D]	12-13		
X=n	8-11		汎用レジスタ n 番をリード
Y=n	4-7		汎用レジスタ n 番をリード

12 拡張乗算器(案)

32×16 の乗算器。汎用レジスタ×D レジスタの乗算が可能。32×32 ではなくて 32×16 なのは汎用演算器コアのクロックと同期させるため。さらに暗号向けの連続乗算ができるようにデータパスを設計した。今回、ブースの採用を見送ったが、実装してみて 2 ビットブースに変更することもある。



13 BIMUL/BIDIV 命令は遅延分岐なし

通常、分岐は遅延分岐 1 スロット入りますが、BIMUL/BIDIV では分岐アドレスは利用されず、I レジスタが 0 でなければデコードフェーズでプログラムカウンタのインクリメントを抑止します。これにより乗算、除算で 2 命令使っていたのを 1 命令にします。

14 BIMUL16 命令

16bit × 16bit + 32bit

BC = A×C+B　18 サイクル

```
    BC(MUL16)
    I=15;C=RSH
    ADDL=A;ADDR=B;B=ADD
MUL16:
    BIMUL16;ADDR=B;B=ADD;C=RSH
```

I レジスタに代入する数字を必要最低限の値にすることで高速化ができる。

15 BIMUL/BIDIV 命令ディレイ対策

BIMUL/BIDIV 命令がディレイのクリティカルパスになりそうなので対策方法を考える。デコードフェーズで判定をして BIMUL/BIDIV レジスタにセット、実行フェーズでは BIMUL/BIDIV レジスタを使う。

16 EXIT 命令

セキュリティの必要がなく汎用のマイコン CPU でいい場合には、終了コードさえ実装すれば大丈夫です。セキュリティが必要な用途では、少なくともモード 0 の汎用レジスタの読み出し可能マスク値を実装します。モード 1 はデバッグ機能なので、余力があれば実装します。

17 B,C レジスタのセレクタ

B,C のレジスタは連携して乗算や除算で利用するための特殊な入力をセレクトします。ほぼない組み合わせがあるので、そこに汎用レジスタなどの入力を割り当てます。

#	B				C				割当
	0	ADD	LSH	RSH	0	ADD	LSH	RSH	
0	○				○				
1	○					○			
2	○						○		
3	○							○	
4		○			○				
5		○				○			
6		○					○		
7		○						○	
8			○		○				
9			×			×			**C=RX**
10			○				○		
11			×					×	**B=RX**
12				○	○				
13				×		×			**B=IMM**
14				×			×		**C=W**
15				○				○	

18 圧縮命令モード(案)

ICF3-V は OS を搭載して大容量のプログラムを動作させることは想定しておらず、いくつかの機能を実装するマイコンのような使い方を想定している。このため使われない機能のハードウェアを省き、ソフトウェアで実装することでトータルでのゲート効率を高めることを考えていた。しかしながら 16 ビット命令が基本の SH マイコンや 16 ビットの圧縮命令をサポートする ARM、RISC-V に対してプログラムメモリの効率が場合によっては 4 倍以上悪くなることがあった。これではプロセッサのゲート数が非常に少ないメリットが活かされない。そこで 16 ビットの圧縮命令をサポートすることを検討した。
ARM や RISC-V のように通常の命令と圧縮命令を混在させることが可能だが、圧縮命令モード、通常命令モードを行き来することになる。モード遷移が必要になるが、命令コー

ドに圧縮命令を示すフィールドは存在しない。RISC-V では圧縮命令を示すフィールドのために実質、16 ビットではなくて 15.5 ビットしか使えない。
ICF3-V の 16 ビットの圧縮命令は、すべてユーザー定義の命令。AND、OR、条件分岐など必要な命令を自分で作る。基本的には 1 命令 8 ワードまで。16 ビットの先頭 6 ビットがオペコードで全部で 64 命令を作ることが可能。8 ワードで収まらない場合、プログラムメモリの他の領域に分岐しても構わない。したがってかなり長いワードでも 1 命令に圧縮できる。また 2 命令分の 16 ワードを 1 つの圧縮命令にすることもできる。圧縮命令 16 ビットの下位 10 ビットは IR レジスタの下位 12bit として圧縮命令の定義の中で利用できる。
圧縮命令開始前の 1 サイクル前に X=a,Y=b,Z=c が必ずはいっています。
圧縮命令開始のサイクルでオペランドが自動的にセットされていることに注意してください。

圧縮命令の実装例 OR Rx,Ry,Rz (Rz = Rx | Ry)

```
A=RX;D=RY
OR;ADDR=D;D=W0
SETK;ADDR=D
CRET;Rn=W
NOP
```

圧縮命令の実装例 AND Rx,Ry,Rz (Rz = Rx & Ry)

```
A=RX;D=RY
AND;ADDR=D;D=W0
SETK;ADDR=D
CRET;Rn=W
NOP
```

圧縮命令の実装例　MUL Rx , Ry , Rz (Rz = Rx × Ry)

```
A=RX;D=RY;B=ADD
I=32;ADDR=D;C=ADD
BIMUL;ADDL=A;ADDR=B;B=RSH;C=RSH
SETK;ADDL=A
CRET;Rn=W
NOP
```

圧縮命令の実装例 ADD Rx , Ry , Rz (Rz = Rx + Ry) / set CF

```
    A=RX;D=RY
    A=ADD;ADDL=A;ADDR=D;SETK
    CRET;Rn=W
    NOP
```

圧縮命令の実装例 ADC Rx,Ry,Rz (Rz = Rx + Ry +CF) / set CF

```
    C=LSH;A=RX;D=RY
    BC(CMODE_ADC);ADDL=A;A=ADD              # CF をクリア
    ADDL=A;ADDR=D;A=ADD;C=LSH
    A=ADD;ADDL=A;ONE;C=LSH
CMODE_ADC:
    BC(CMODE_ADC);C=ADD
    ADDL=A;D=ADD
    ADDR=C;NOT;ONE;A=ADD
CMODE_ADC2:
    ADDR=D;SETK
    CRET;Rn=W
    NOP
```

圧縮命令の実装例　B n　無条件ジャンプ

```
    B(n)
    CMODE
```

圧縮命令の実装例　BCF n キャリーフラグが 1 なら分岐

```
    BCF1(n)
    CMODE
```

圧縮命令の実装例　NMODE / TESTD / EXIT0 / EXIT1

オペランドに命令種を設定。圧縮 op コードの最後に割り当てるといい。

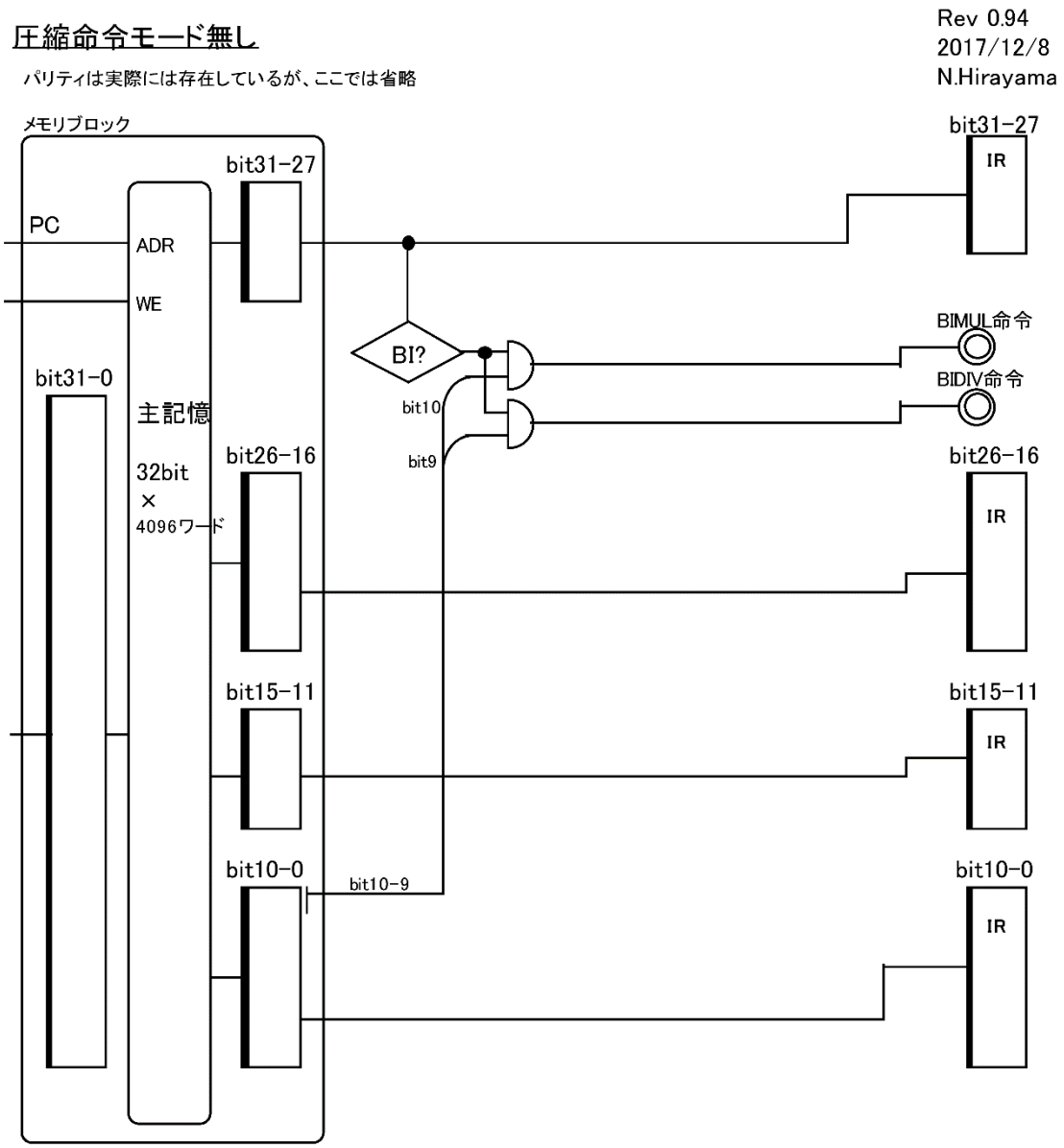
```
    I=n
    BI(CMODE_NMODE)
    NOP
    BI(CMODE_TESTD)
    NOP
    BI(CMODE_EXIT0)
    NOP
    EXIT(1)
CMODE_NMODE:
    NRET
    NOP
CMODE_TESTD:
    TESTD
    CRET
    NOP
CMODE_EXIT0:
    EXIT(0)
```

圧縮命令の CALL の実装がまだ未検討。必須な機能だと思うので後で実装する。

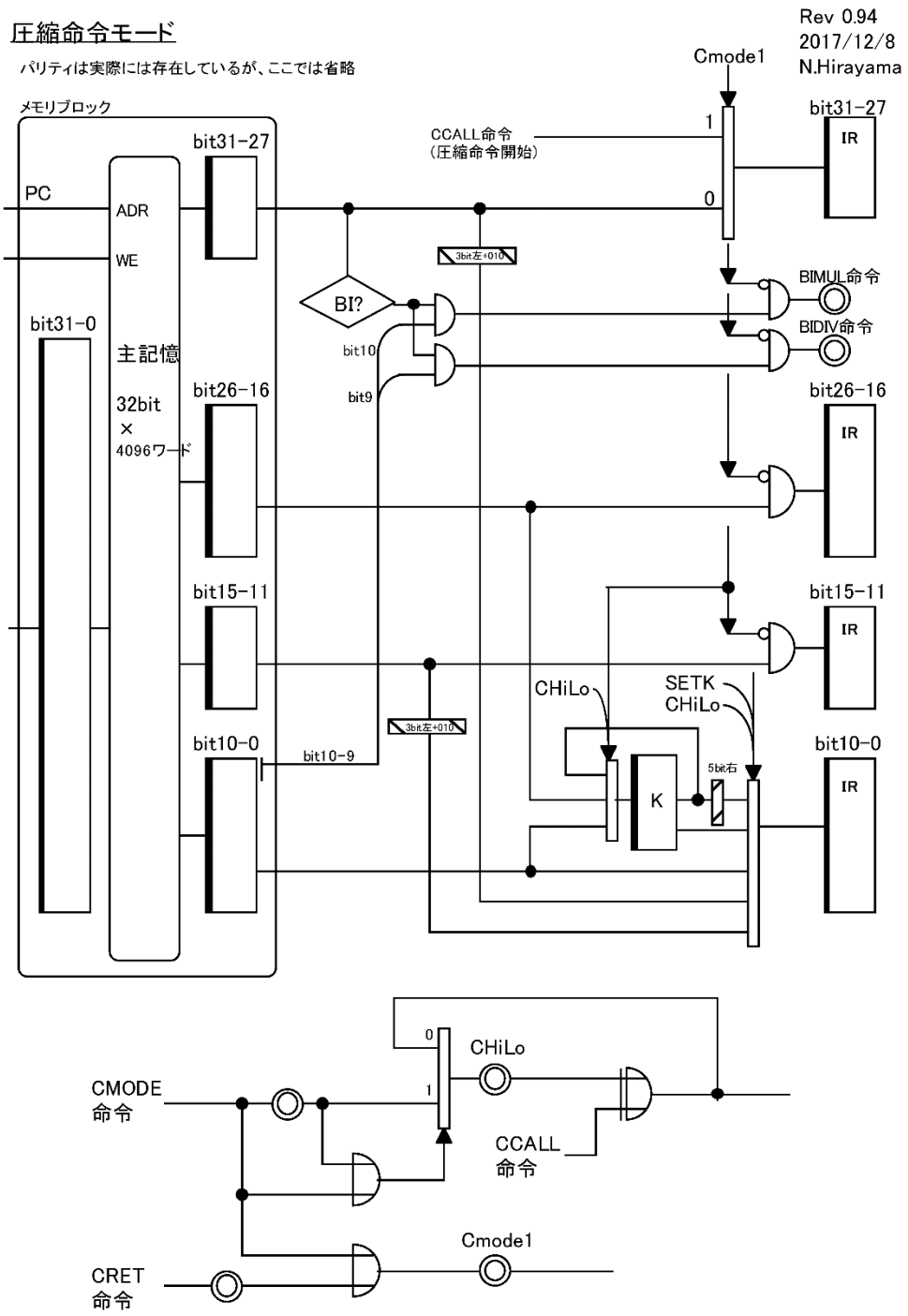
圧縮命令モードの論理実装について

通常命令の命令実行フェーズの前段に若干の論理を追加するだけで、ほぼ実装できる(予定)

●論理実装前



●論理実装後


圧縮命令モード
パリティは実際には存在しているが、ここでは省略
Rev 0.94
2017/12/8
N.Hirayama
メモリブロック
PC
ADR
WE
bit31-0
主記憶
32bit
×
4096ワード
bit31-27
bit26-16
bit15-11
bit10-0
bit10-9
BI?
bit10
bit9
3bit左+010
3bit左+010
Cmode1
CCALL命令
（圧縮命令開始）
1
0
IR
BIMUL命令
BIDIV命令
CHiLo
SETK
CHiLo
K
5bit右
CMODE
命令
CRET
命令
CCALL
命令
Cmode1

19 一人パイプライン(疑似パイプライン) (案)

ICF3 の命令コードのフォーマットにはフェッチ、実行、ストアなどを 1 命令で並列に動作させることをサポートしていた。これを発展させ疑似パイプラインとして整備する案。
このためバイパスを強化、わかりやすいニモニックに変更した。

疑似パイプライン								
疑似ニモニック	フェッチ	ロード	実行	ストア				ニモニック
ADD R0,R1,R8	F	L	E	S				X=0;Y=1
ADD R2,R3,R9		F	L	E	S			X=2;Y=3;A=RX;D=RY
ADD R4,R5,R10			F	L	E	S		X=4;Y=5;A=RX;D=RY;ADDL=A;ADDR=D
ADD R6,R7,R11				F	L	E	S	X=6;Y=7;A=RX;D=RY;ADDL=A;ADDR=D;R8=W
								A=RX;D=RY;ADDL=A;ADDR=D;R9=W
								ADDL=A;ADDR=D;R10=W
								R11=W
疑似ニモニック	フェッチ	ロード	実行	ストア				
ADD R0,R1,**R8**	F	L	E	S				X=0;Y=1
ADD **R8**,R3,R9		F	L	E	S			X=3;A=RX;D=RY
								A=RX;ADDL=A;ADDR=D;D=W0
								ADDL=A;ADDR=D;R8=W
								R9=W
疑似ニモニック	フェッチ	ロード	実行	ストア				ニモニック
ADD R0,R1,**R8**	F	L	E	S				X=0;Y=1
ADD R2,R3,R9		F	L	E	S			X=2;Y=3;A=RX;D=RY
ADD **R8**,R5,R10			F	L	E	S		X=5;A=RX;D=RY;ADDL=A;ADDR=D
								A=RX;ADDL=A;ADDR=D;R8=W;C=W
								ADDL=A;ADDR=C;R9=W
								R10=W

20　プログラムコードのキャッシュ(案)

プログラムのコードはフラッシュメモリに入れて、SRAM でキャッシュする仕組みを検討中。

21 割込み(案)

INTSTATUS 命令を追加、W レジスタに割り込み要因を表すデータをセット。
同時に D にもセットして、割込み要因がゼロであることが確認された場合は
ソフト割込み。次のサイクルで C レジスタに入れて、割込み要因を 1bit つづシフトしながら、すべての要因の処理を行う。

22 4G ワードのアドレス空間を分岐できる命令の追加予定

予定です。

23 ライセンスについて

OpenICF3 の公式サイトのライセンスのページを参照願います。(まだ詳細は決まっていません)
広く普及するような方向で検討しています。

24 最後に

マイコン用などの小規模な CPU の評価は、いろいろな要因があり一般の CPU よりも難しいと思います。最近は FPGA に CPU を実装できるようなので、実際に FPGA に実装していこうかと思います。もし ICF3-V に興味を持った方が、いらっしゃればメールや
Facebook などにご連絡ください。
e-mail :　nhira@icf.xii.jp　　Facebook : https://www.facebook.com/naoki.hirayama.37
Google+や twitter でも構いません。OpenICF3 の公式サイトの連絡先に URL があります。

Apenddix A 乗算サンプルコード

(1) 乗算　32bit × 32bit （33 サイクル）

BC = A×C+B

```
I=32
BIMUL;ADDL=A;ADDR=B;B=RSH;C=RSH
```

(2) 乗算　16bit × 16bit (19 サイクル)

B = (A×C+B)の上位 16bit

D = (A×C+B)の下位 16bit

```
I=16
BIMUL;ADDL=A;ADDR=B;B=RSH;C=RSH
ADDR=C;D=ADD
SHR=16;D=W0
```

(3) 乗算　8bit × 8bit (11 サイクル)

B = (A×C+B)の上位 8bit

D = (A×C+B)の下位 8bit

```
I=8
BIMUL;ADDL=A;ADDR=B;B=RSH;C=RSH
ADDR=C;D=ADD
SHR=24;D=W0
```

(4) 乗算　32bit × 8bit (11 サイクル)

B = (A×C+B)の上位 32bit

D = (A×C+B)の下位 8bit

```
I=8
BIMUL;ADDL=A;ADDR=B;B=RSH;C=RSH
ADDR=C;D=ADD
SHR=24;D=W0
```

(5) 乗算　24bit × 8bit (14 サイクル)

D = A×C+B

```
I=8
BIMUL;ADDL=A;ADDR=B;B=RSH;C=RSH
ADDR=C;D=ADD
SHR=24
A=W;ADDR=B;D=ADD
SHL=8
OR;D=W0
```

Apenddix B 除算サンプルコード

(1) 除算 64bit÷32bit (65 サイクル)

BC = BC ÷ D

A = BC mod D

```
I=64;A=ADD
BIDIV;ADDR=D;ONE;NOT;A=ADD;B=LSH;C=LSH
```

(2) 除算 32bit÷32bit (33 サイクル)

C = B ÷ D

A = B mod D

```
I=32;A=ADD
BIDIV;ADDR=D;ONE;NOT;A=ADD;B=LSH;C=LSH
```

(3) 除算(A<D ケース) 64bit÷32bit (33 サイクル)

C = AB ÷ D

A = AB mod D

```
I=32
BIDIV;ADDR=D;ONE;NOT;A=ADD;B=LSH;C=LSH
```

(4) 除算(ゼロ除算対応版) 32bit÷32bit (35 サイクル)

C = B ÷ D

A = B mod D

```
TESTD
BZF1(DIVISION_BY_ZERO)
I=32;A=ADD
BIDIV;ADDR=D;ONE;NOT;A=ADD;B=LSH;C=LSH
```

Apenddix C その他サンプルコード

(1) 64bit 加算　R 1 R0 = R1R0 + R3R2 (7 サイクル)

```
   X=0;Y=2
   A=RX;D=RY
   ADDL=A;ADDR=D;A=ADD;D=ADD;X=1;Y=3
   BCF0(LBL1);A=RX;D=RY;ADDR=D
   R0=W;ADDL=A;ADDR=D;A=ADD
   ADDL=A;ONE;A=ADD
LBL1:
   R1=W
```

(2) 128bit 加算　R3R2R1R0 = R3R2R1R0 + R7R6R5R4 (16 サイクル)

```
   X=0;Y=4
   A=RX;D=RY
   ADDL=A;ADDR=D;A=ADD;X=1;Y=5
   R0=W;B=RX;D=RY
   BCF0(LBL1);A=ADD;ADDR=B                    # CF をクリアするため
   ADDL=A;ADDR=D;A=ADD;C=LSH;X=2;Y=6
   ADDL=A;ONE;A=ADD;C=LSH;X=2;Y=6
LBL1:
   R1=W;B=RX;D=RY
   BC(LBL2);A=ADD;ADDR=B                      # CF をクリアするため
   ADDL=A;ADDR=D;A=ADD;C=LSH;X=3;Y=7
   ADDL=A;ONE;A=ADD;C=LSH;X=3;Y=7
LBL2:
   R2=W;B=RX;D=RY
   BC(LBL3);A=ADD;ADDR=B                      # CF をクリアするため
   ADDL=A;ADDR=D;A=ADD
   ADDL=A;ONE;A=ADD
LBL3:
   R3=W
```