情報電子工学総合実験の目標

1. 現代の技術を用いて TeC を再設計し実装して動作させる。
    - 構造が分かりやすい(美しい)
    - 高速に(少ないクロック数で) 動作する
2. TeC を再設計する中で、順序立てた設計の重要性を認識する。

従来の TeC の設計目標

1990 年頃に安価に利用できた技術・素子を用いて、IE 実験室の設備で可能な範囲で、教育用の綺麗な命令体系を持った独自 CPU を搭載したコンソール付きのコンピュータを製作し、安定して動作させる。

制約

(1) 入手可能な部品の品種は限られ、部品の仕様は変更できない。
例：ALU は 74381 を使用するしかない。レジスタファイルは 74???を…

(2) IE 実験室で製作可能なプリント基板で実装する必要がある。
最大サイズ A4、配線はピン間 1 本、部品数、配線数はとにかく少なくする。

(3) 詳細な解析はできない。
配線の遅延などは未知のままになる。安全サイドの設計をするしかない。

設計方針

少ない部品をなるべく使い回すことで機能を実現する。例えば、一つの ALU でデータ計算、アドレス計算(インデクスモード)、レジスタ類のインクリメンドデクリメント、マルチプレクサの役割を果たす。

少ない部品を使い回すために複雑な制御手順を実行できる制御部が必要になる。そこで、制御部はマイクロプログラム方式を採用する。

総合実験で作る TeC の設計目標

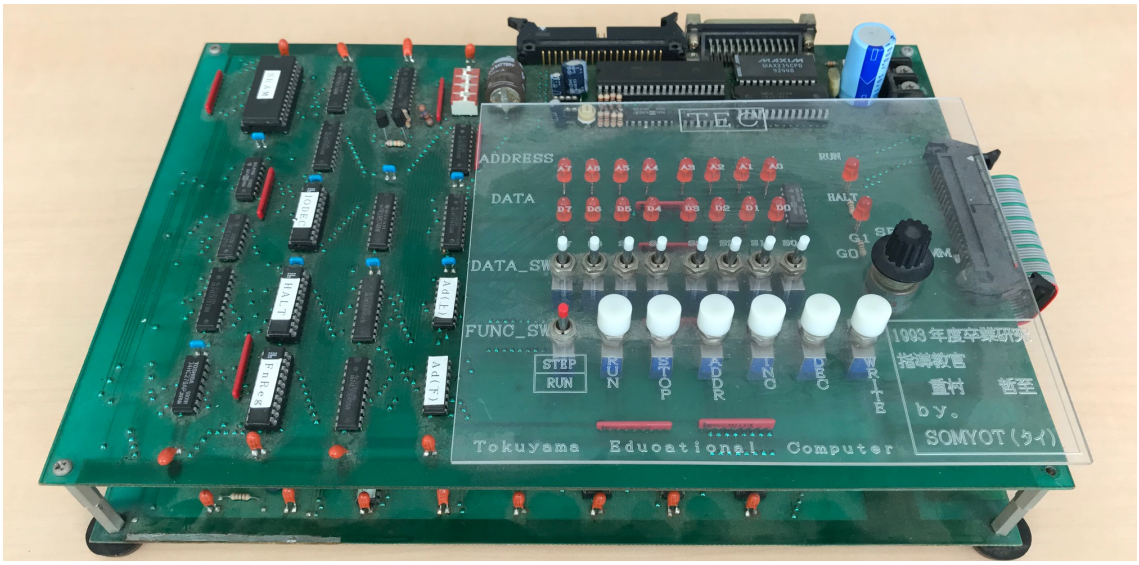
現在では、FPGA を用いることで上記の制約は全て解決できる。現代の FPGA を用いて、美しく高速な TeC CPU を再設計する。

(1) FPGA を用いれば、ALU、レジスタファイル等の仕様を自由に決定可能

(2) FPGA には十分な素子と十分な配線を集積可能

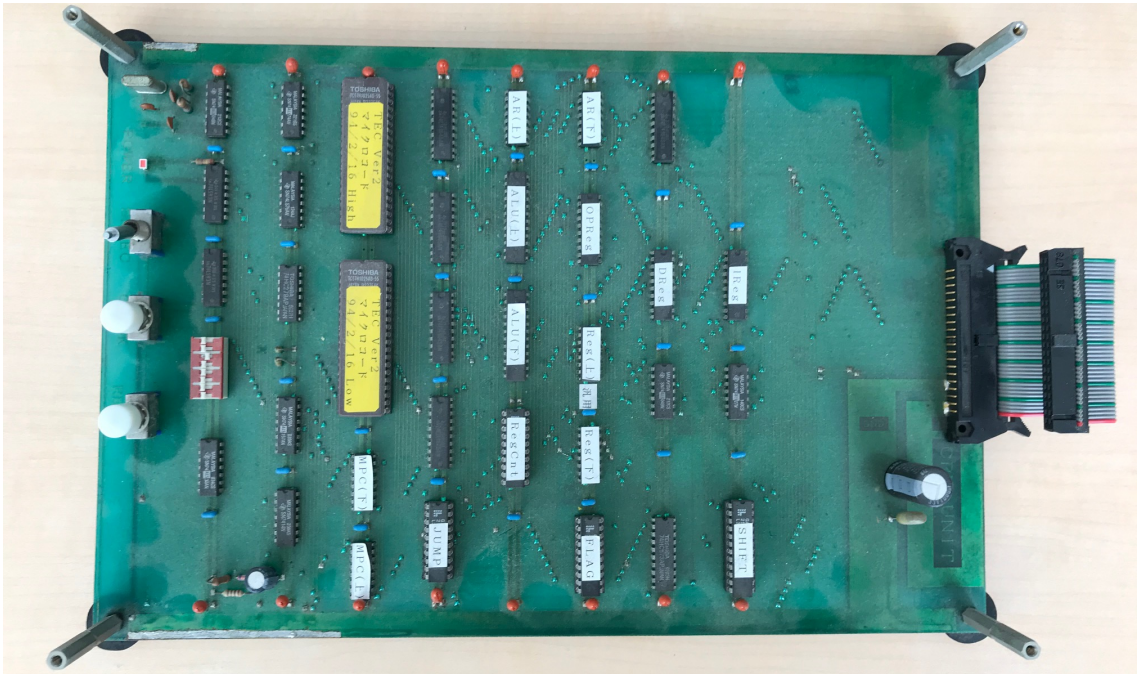
(3) FPGA ではクロック専用の配線によりクロックスキューを無視できる。
FPGA の開発システムはセットアップタイム、ホールドタイムが不足する箇所を自動的に見つけてくれる。

初代 TeC



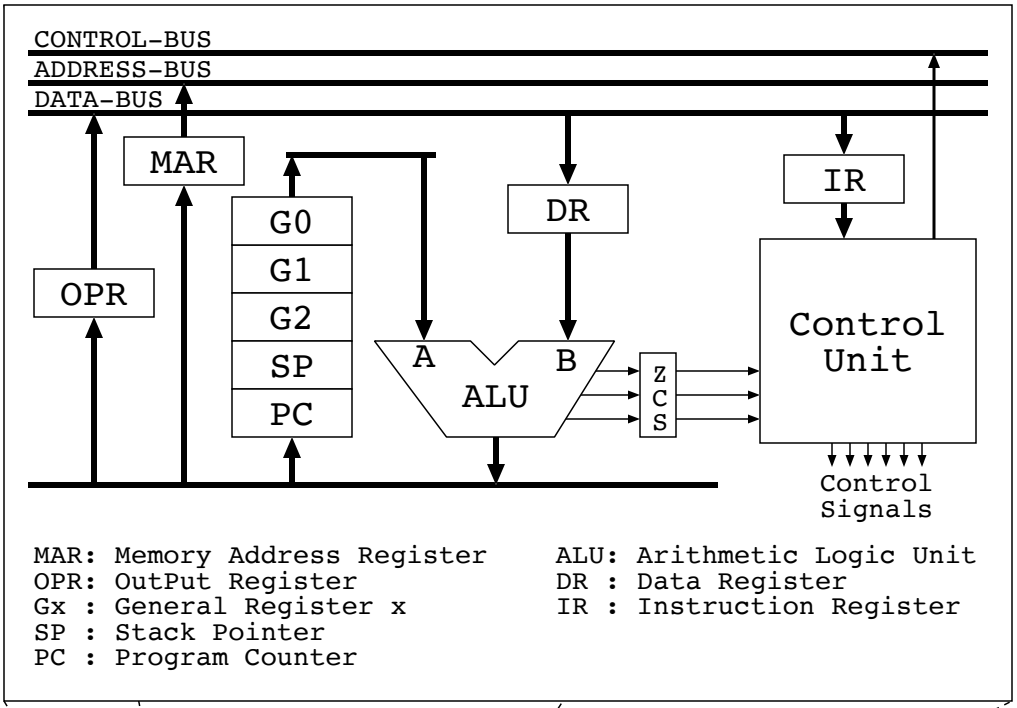
1. TeC6 と基本機能はほぼ同じ
2. 基板は A4×２枚
3. 上段の基板がメモリ（S-RAN），I/O（8251，8255），コンソールパネル

初代 TeC の CPU



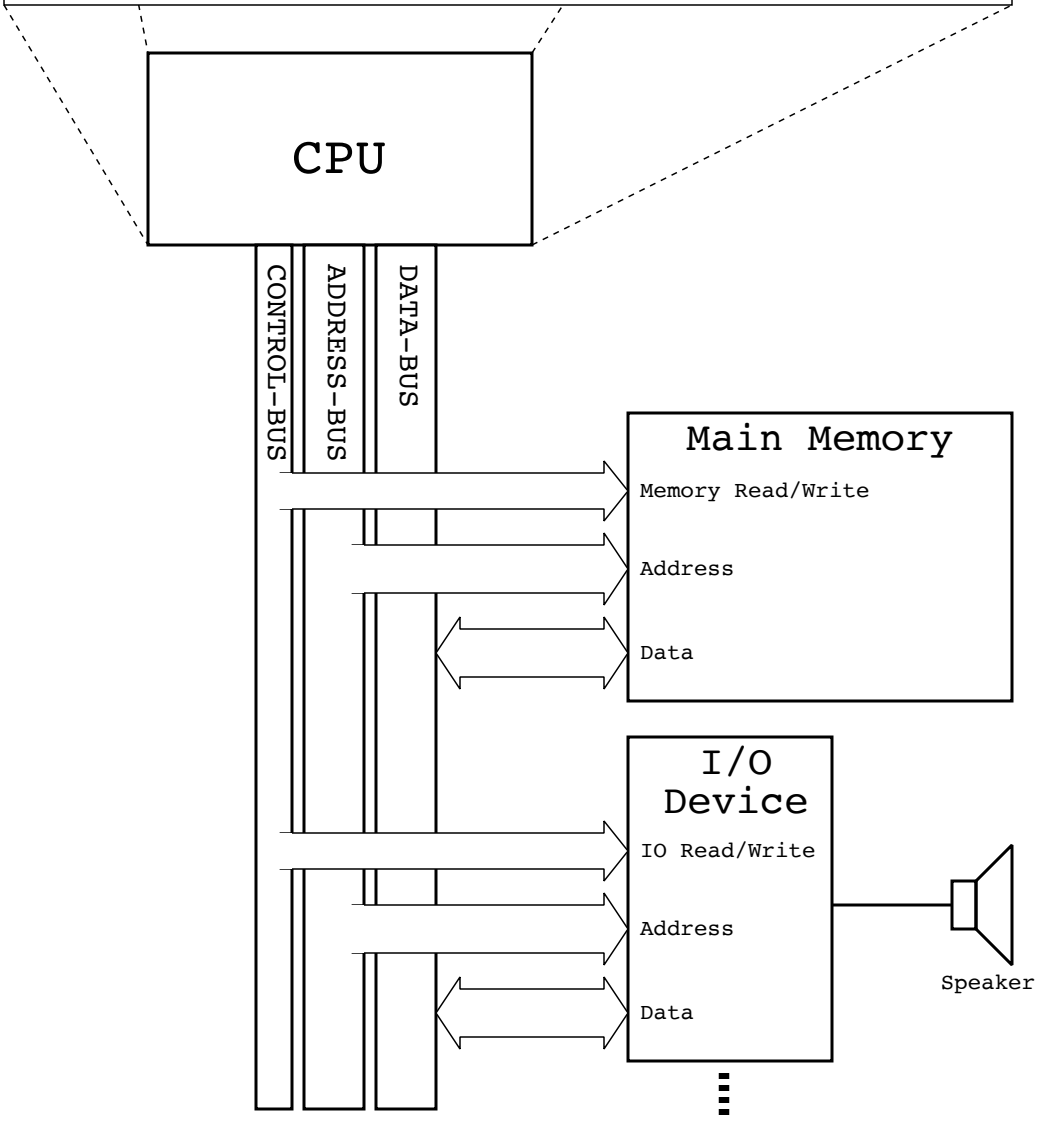
1. 汎用部品（TTL，ROM）と GAL で構成
2. 実際に製作するために，部品点数と配線を少なくする必要があった.
3. マイクロプログラムで複雑な制御をすることで解決した.

TeC CPUの内部構造



ALUの機能

A		
B		
A	+	B
A	-	B
A	+	1
A	-	1
A	and	B
A	or	B
A	xor	B
...		



TEC6命令表

Ver. 2.3 (2006.3.25)

※ステート数：イミディエイト／ダイレクト／インデクスド
(ジャンプ命令では、条件不成立／ダイレクト／インデクスド)

ニーモニック	命令	第１バイト OP	第１バイト GRXR	第２バイト	フラグ変化	ステート数※	動作
NO	No Opration	0000	00 00	————	×	3	何もしない
LD	Load	0001	GR XR	aaaa aaaa	×	5/7/7	GR <- [EA]
ST	Store	0010	GR XR	aaaa aaaa	×	-/7/7	[EA] <- GR
ADD	Add	0011	GR XR	aaaa aaaa	○	5/7/7	GR <- GR + [EA]
SUB	Subtract	0100	GR XR	aaaa aaaa	○	5/7/7	GR <- GR - [EA]
CMP	Compare	0101	GR XR	aaaa aaaa	○	5/7/7	GR - [EA]
AND	Logical And	0110	GR XR	aaaa aaaa	○	5/7/7	GR <- GR & [EA]
OR	Logical Or	0111	GR XR	aaaa aaaa	○	5/7/7	GR <- GR | [EA]
XOR	Logical Xor	1000	GR XR	aaaa aaaa	○	5/7/7	GR <- GR ^ [EA]
SHLA	Shift Left Arithmetic	1001	GR 00	————	○	4	GR <- GR << 1
SHLL	Shift Left Logical	1001	GR 01	————	○	4	GR <- GR << 1
SHRA	Shift Right Arithmetic	1001	GR 10	————	○	4	GR <- GR >> 1
SHRL	Shift Right Logical	1001	GR 11	————	○	4	GR <- GR >>> 1
JMP	Jump	1010	00 XR	aaaa aaaa	×	-/5/6	PC <- EA
JZ	Jump on Zero	1010	01 XR	aaaa aaaa	×	4/5/6	if Zero PC <- EA
JC	Jump on Carry	1010	10 XR	aaaa aaaa	×	4/5/6	if Carry PC <- EA
JM	Jump on Minus	1010	11 XR	aaaa aaaa	×	4/5/6	if Sign PC <- EA
CALL	Call subroutine	1011	11 XR	aaaa aaaa	×	-/6/7	[--SP]<-PC, PC<-EA
~~IN~~	~~Input~~	~~1100~~	~~GR 00~~	~~0000 pppp~~	×	8	~~GR <- IO[P]~~
~~OUT~~	~~Output~~	~~1100~~	~~GR 11~~	~~0000 pppp~~	×	8	~~IO[P] <- GR~~
PUSH	Push Register	1101	GR 00	————	×	6	[--SP] <- GR
~~PUSHF~~	~~Push Flag~~	~~1101~~	~~11 01~~		×	6	~~[--SP] <- FLAG~~
POP	Pop Register	1101	GR 10	————	×	6	GR <- [SP++]
~~POPF~~	~~Pop Flag~~	~~1101~~	~~11 11~~		○	6	~~FLAG <- [SP++]~~
~~EI~~	~~Enable Interrupt~~	~~1110~~	~~00 00~~		×	4	~~割り込み許可~~
~~DI~~	~~Disable Interrupt~~	~~1110~~	~~00 11~~		×	4	~~割り込み禁止~~
RET	Return from subroutine	1110	11 00	————	×	6	PC <- [SP++]
~~RETI~~	~~Return from Interrupt~~	~~1110~~	~~11 11~~		×	6	~~PC <- [SP++], EI~~
HALT	Halt	1111	11 11	————	×	4	停止

GR	意味
00	G0
01	G1
10	G2
11	SP

XR	意味
00	ダイレクトモード
01	G1インデクスドモード
10	G2インデクスドモード
11	イミディエイトモード

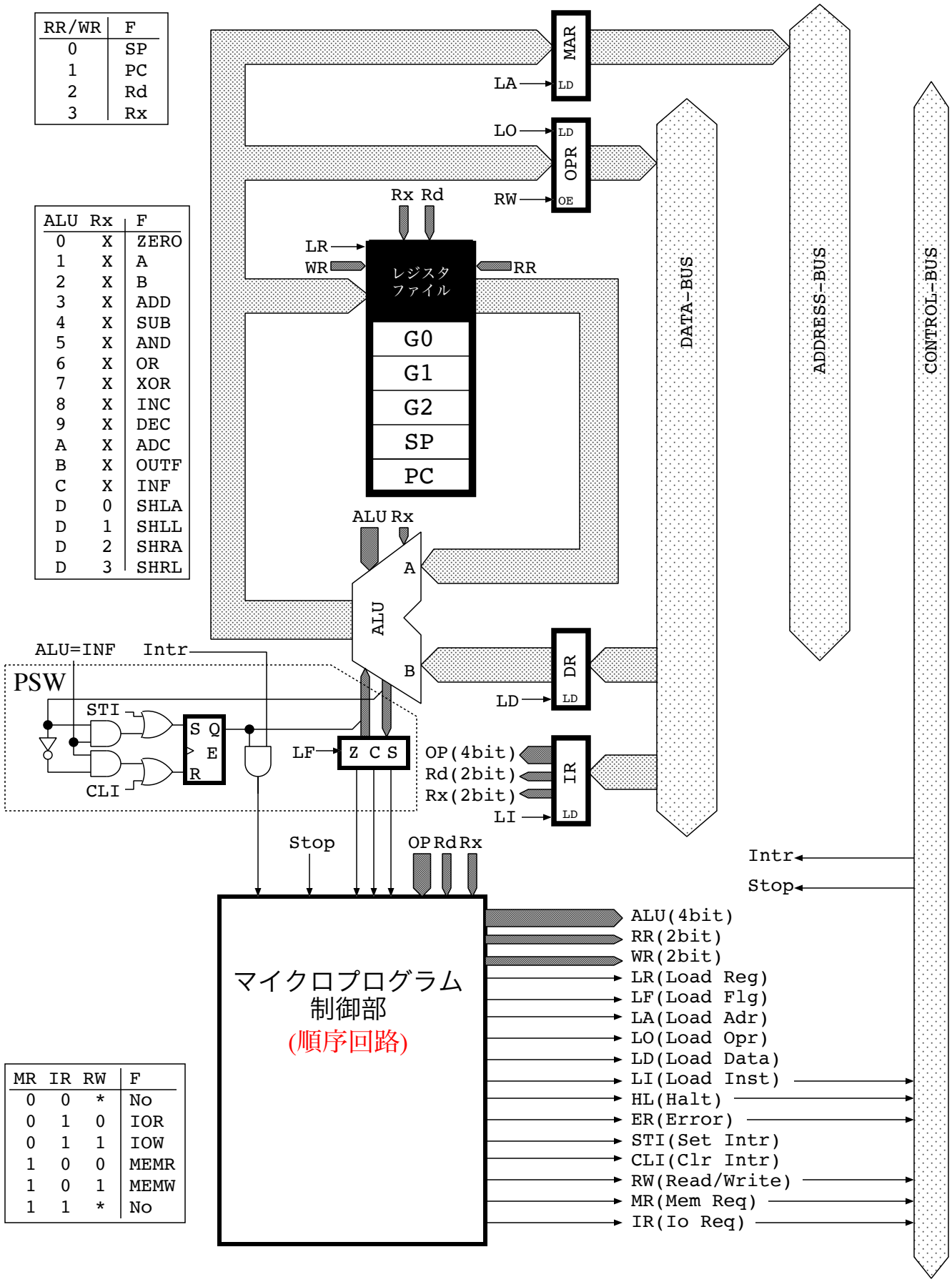
メモリマップ

Addr	内容
00 | ~~DB~~ FF	RAM
~~DC~~	~~INT0 割り込みベクタ~~
~~DD~~	~~INT1 割り込みベクタ~~
~~DE~~	~~INT2 割り込みベクタ~~
~~DF~~	~~INT3 割り込みベクタ~~
~~E0~~ | ~~FF~~	~~ROM(IPL)~~

INT1 : SIO 受信割り込み
INT2 : SIO 送信割り込み

I/Oマップ

Addr	Read/Write
0	Data-Sw/b0:Beep
1	Data-Sw/b0:Speaker
2	SIO-Data/SIO-Data
3	b7:Tx Ready ,b7:Tx STI b6:Rx Ready/b6:Rx STI
4	空き/空き
...	....
F	空き/空き



TeC6 CPUのブロック図

TeC の CPU が命令を実行する様子

1. TeC が実行するプログラムの例

```
番地   機械語   ラベル   命令     オペランド
00     14               LD       G1,03H
01     03
02     FF               HALT
03     AB               DC       0ABH
```

2. 上のプログラムがメモリに格納されている状態での CPU の動作

① PC を００H にする（ユーザの操作）
② RUN ボタンを押す（ユーザの操作）
③ 実行が開始されると CPU はクロック毎に動作する

マイクロ操作		制御信号								
ステート	操作	ALU	RR	WR	LR	LA	LI	LD	MR	HL
1 共通	PC→MAR, Stop チェック	1	1			1				
2 共通	メモリ→IR、PC++	8	1	1	1		1		1	
3 共通	命令デコード、PC→MAR	1	1			1				
4LD	メモリ→DR、PC++	8	1	1	1			1	1	
5LD	DR→MAR	2				1				
6LD	メモリ→DR							1	1	
7LD	DR→G1、１共通に戻る	2		2	1					
１共通	PC→MAR, Stop チェック	1	1			1				
2 共通	メモリ→IR、PC++	8	1	1	1		1		1	
3 共通	命令デコード、PC→MAR	1	1			1				
4HALT	Halt 出力、１共通に戻る									1

制御信号で空白は「０」の意味

イクロプログラム ROM の VHDL 記述を生成する．TBL2VHD は「シンボル表ファイル」と，リスト 3.2 に示した「デコード ROM 表ファイル」から，デコード ROM の VHDL 記述を生成する．これらのプログラムは C 言語で記述され UNIX 上で動作する．

本マイコンの開発では，コンソールパネルの制御方式や命令セットアーキテクチャを決定するために，実際に動く状態の試作機を製作し，試行錯誤を繰り返した．このように頻繁な設計変更が可能だったのは，マイクロプログラム制御方式を採用し，かつ，マイクロプログラム開発環境を準備できたためである．

3.5.2　μＡＳＭ

μＡＳＭは，マイクロアセンブリ言語用の 2 パス方式のアセンブラであり，C 言語を用いて約 900 行で記述されている．以下ではμＡＳＭの入力となるマイクロアセンブリ言語の文法について説明する．

ソースプログラムの形式

ソースファイルは，行の連なりからなるテキストファイルである．μＡＳＭのソースプログラムでは英字大文字と小文字は区別されない．

- 行フォーマット
  行は，ラベル，μＡＳＭ命令，コメントからなる．一行に複数のμＡＳＭ命令を記述することができる．空白[8] はトークンの前後にいくつでも挿入することができる．但し，行頭に空白を置いた場合は「ラベルが存在しない」ことを表す．

  行 ＝ [ラベル] [μＡＳＭ命令　{, μＡＳＭ命令}] [; コメント]

- ラベル
  行の最初の文字が英字の場合は，その行にラベルがあるものとみなされる．ラベルは英字で始まり，その後に任意の長さの英数字が続く文字列である．ラベルがない行の場合は，行を空白で始める．ラベルの長さは 8 文字以内でなければならない．

μＡＳＭ命令

μＡＳＭは，μＡＳＭ命令からマイコンのマイクロプログラムを生成する．CPU の構成要素毎に一つのμＡＳＭ命令を記述する．例えば,「(1) ALU とフラグの動作は ALU 命令」,「(2) ALU の計算結果をどのレジスタに書き込むかは LD 命令」によって記述される．一行に複数のμＡＳＭ命令を書いた場合，それらはマイクロプログラム ROM の一語に格納され，同時に並行して実行される．以下では，各μＡＳＭ命令について解説する．

[8] 空白は，C 言語のライブラリ `isspace` 関数によって判別される．

- **ALU 命令**
  ALU とフラグの動作を記述する命令である．ALU，RR，LF フィールドに格納するマイクロ命令を決定する．ニーモニックの書式は次の通りである．

  ALU 命令 = alu(ALU 機能 [, レジスタ名 [,FL]])

  ALU 機能がレジスタファイル（A 入力）を使用する機能の場合は，ALU 命令にレジスタ名を記述する必要がある．レジスタ名によって RR フィールドのマイクロ命令が決定される．また，FL オプションを付けた場合は LF マイクロ命令が生成される．

ALU 命令に指定できる機能	
ALU 機能	意味
ZERO	0 を出力
REG	レジスタをそのまま出力
ADD	レジスタファイルとデータレジスタの和を出力
SUB	レジスタファイルとデータレジスタの差を出力
AND	レジスタファイルとデータレジスタの論理積を出力
OR	レジスタファイルとデータレジスタの論理和を出力
XOR	レジスタファイルとデータレジスタの排他的論理和を出力
INC	レジスタファイルに 1 加えた値を出力
DEC	レジスタファイルから 1 引いた値を出力
OUTF	PSW の値を出力
INF	データレジスタの値を PSW に書き込む
SFT	レジスタファイルを 1 ビットシフトして出力

ALU 命令のレジスタ名	
レジスタ名	意味
SP	スタックポインタ（レジスタファイルの一部）
PC	プログラムカウンタ（レジスタファイルの一部）
RD	IR の Rd フィールドで指定されるレジスタファイルのレジスタ
RX	IR の Rx フィールドで指定されるレジスタファイルのレジスタ
DR	データレジスタ（ALU 機能が REG の場合だけ指定できる）

- **LD 命令**
  ALU の計算結果をどのレジスタに書き込むか記述する μＡＳＭ命令である．WR，LR，LA，LO フィールドがこの μＡＳＭ命令によって決定される．ニーモニックの書式は次の通りであり，複数のレジスタを指定することができる．ただし，レジスタファイルの複数のレジスタを同時に記述することはできない．

  LD 命令 = ld(レジスタ名 ｛ , レジスタ名 ｝)

LD 命令のレジスタ名	
レジスタ名	意味
SP	スタックポインタ（レジスタファイルの一部）
PC	プログラムカウンタ（レジスタファイルの一部）
RD	IR の Rd フィールドで指定されるレジスタファイルのレジスタ
RX	IR の Rx フィールドで指定されるレジスタファイルのレジスタ
AR	メモリアドレスレジスタ（MAR）
OPR	アウトプットレジスタ（OPR）

- **BUS 命令**
  外部バスの動作を決定するμＡＳＭ命令である．LD，LI，RW，MR，IR フィールドが，この命令によって決定される．ニーモニックの書式は次の通りであり複数の動作を記述できるが，その組み合わせが有効な意味を持つかμＡＳＭはチェックしない．

BUS 命令 = bus(動作 {, 動作})

BUS 命令で指定できる動作	
動作	意味
IOREQ	IR マイクロ命令を生成（バスは I/O アクセス）
MREQ	MR マイクロ命令を生成（バスはメモリアクセス）
OE	RW マイクロ命令を生成（バスは書き込みモード）
LDDR	LD マイクロ命令を生成（データレジスタに読込む）
LDIR	LI マイクロ命令を生成（命令レジスタに読込む）

- ジャンプ命令
  ジャンプマイクロ命令を生成するμＡＳＭ命令である．この命令によって，JP，JA フィールドが決定される．ニーモニックの書式は次の通りである．

ジャンプ命令 = jop(アドレス) | j(アドレス) | jcc(条件, アドレス)

jop 命令は JOP マイクロ命令を，j 命令は JMP マイクロ命令を，jcc 命令は Jcc マイクロ命令を生成する．「アドレス」には，整数またはソースプログラム中で定義されたラベルを記述する．「アドレス」が JA フィールドに出力される．

Jcc 命令の条件	
条件	意味
INT	CPU に停止信号か，割込み許可状態で割込み信号が入力されている
STP	CPU に停止信号が入力されている
NJP	機械語命令のジャンプ条件が成立していない
IM	IR の Rx フィールドの値がイミディエイトモードを意味している
DI	IR の Rx フィールドの値がダイレクトモードを意味している

- その他の命令

  HL，ER，STI，CLI フィールドを決定する四つの命令がある．halt 命令は HL マイクロ命令，err 命令は ER マイクロ命令，sti 命令は STI マイクロ命令，cli 命令は CLI マイクロ命令を生成する．

  その他の命令 = halt | err | sti | cli

3.5.3　BIN2VHD

μＡＳＭが生成したマイクロプログラムファイルと，リスト 3.4 に示す VHDL 記述の雛型ファイルから，マイクロプログラム ROM の VHDL 記述を出力するプログラムである．出力は，Xilinx FPGA のブロック RAM を記述する VHDL である．BIN2VHD は C 言語を用い約 130 行で記述されている．

雛型ファイルの概要

雛型ファイルの内容は，ブロック RAM（RAMB4_S16_S16[40]）を内部に含む VHDL モジュール MROM の未完成な記述である．雛型ファイルの 42〜45 行の ‘%s’[9] は，ブロック RAM の初期値を記述する部分である．BIN2VHD は，‘%s’ をマイクロプログラムファイルから入力したマイクロプログラム 32 バイト（8 語）の 16 進数表現と置き換えて出力する．出力が，初期値を書き込んで完成したブロック RAM の VHDL 記述になる．

マイクロプログラム ROM の容量

ブロック RAM の容量は 512 バイトなので，3.4.2 節で述べた「32 ビット × 256 語」のマイクロプログラムを格納するにはブロック RAM が二つ必要になる．しかし，96 語のマイクロプログラムで本マイコンの機能は全て実現できている．そこで，実際はマイクロプログラム ROM の容量を「32 ビット × 128 語」とし，一つのブロック RAM で実現することにした．そのため，現在の雛型ファイルの記述は，ブロック RAM を一つだけを含むものになっている．将来，より大きなマイクロプログラムが必要になった場合は，BIN2VHD プログラムと雛型ファイルを変更する必要がある．

32 ビットのデータ幅の実現方法

データ幅が 16 ビットのデュアルポート RAM を一つ用い，以下に説明する記述によって，データ幅が 32 ビットのマイクロプログラム ROM が実現される．

A，B 二つの 16 ビットポートを使用して 32 ビットのデータ幅を実現する．リスト 3.4 中で，37 行の「addr8a <= (’0’ & P_ADDR);」の記述によってポート A が RAM の前半を，

[9] 実際の雛型ファイルでは，一行に一つの ‘%s’ になるように改行してある．

```
A  A R W L L L L L L H E S C R M I J J
D  L R R R F A O D I L R T L W R R P A                                Page(1)
R  U                     I I
00
00                                          ####NO####
00 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 3 03 fetch   alu(reg,PC),ld(AR),jcc(INT,intr)        ; PC=>AR, if INT intr
01 8 1 1 1 0 1 0 0 1 0 0 0 0 0 1 0 0 00          alu(inc,PC),ld(PC,AR),bus(MREQ,LDIR)    ; ++PC=>AR,M=>IR
02 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 00          jop(0)                                  ; Jump on OP
03 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 4 00 intr     alu(reg,PC),ld(OPR),jcc(STP,fetch)      ; PC=>OPR, if STP fetch
04 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 3 06          alu(zero),ld(AR),jcc(INT,intr0)         ; 0=>AR,STPが変化した可能性あり
05 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 00          j(fetch)
06 9 0 0 1 0 1 0 1 1 0 0 0 0 0 0 1 0 00 intr0    alu(dec,SP),ld(AR,SP),bus(IOREQ,LDIR,LDDR); --SP=>AR,VECT=>DR
07 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 00        bus(MREQ,OE)                            ; OPR=>Mem
08 2 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 00      cli,alu(reg,DR),ld(AR)                  ; clr int,DR=>AR
09 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 00        bus(MREQ,LDDR)                          ; M=>DR
0A 2 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 00          alu(reg,DR),ld(PC),j(fetch)             ; DR=>PC, jump fetch
0B
0B 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 00 err      halt,err,j(fetch)                       ; opecode error
0C
0C                                          ####LD####
0C 8 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 1 11 ldi      alu(inc,PC),ld(PC),bus(MREQ,LDDR),j(ld2); ++PC,M=>DR
0D 8 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 7 0F ld       alu(inc,PC),ld(PC),bus(MREQ,LDDR),jcc(DI,ld0); ++PC,M=>DR,if direct ld0
0E 3 3 0 0 0 1 0 0 0 0 0 0 0 0 0 1 10            alu(add,Rx),ld(AR),j(ld1)               ; DR+Rx=>AR
0F 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 00 ld0        alu(reg,DR),ld(AR)                      ; DR=>AR,if drct ld0
10 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 00 ld1      bus(MREQ,LDDR)                          ; M=>DR
11 2 0 2 1 0 0 0 0 0 0 0 0 0 0 0 1 00 ld2        alu(reg,DR),ld(Rd),j(fetch)             ; DR=>Rd,jump fetch
12
12                                          ####ST####
12 8 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 7 14 st       alu(inc,PC),ld(PC),bus(MREQ,LDDR),jcc(DI,st0); ++PC,M=>DR,if drct st0
13 3 3 0 0 0 1 0 0 0 0 0 0 0 0 0 1 15            alu(add,Rx),ld(AR),j(st1)               ; DR+Rx=>AR
14 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 00 st0        alu(reg,DR),ld(AR)                      ; DR=>AR
15 1 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 00 st1        alu(reg,Rd),ld(OPR)                     ; Rd=>OPR
16 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 00          bus(MREQ,OE),j(fetch)                   ; DR=>M,jump fetch
17
17                                          ####ADD####
17 8 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 1 1C addi     alu(inc,PC),ld(PC),bus(MREQ,LDDR),j(add2); ++PC,M=>DR
18 8 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 7 1A add      alu(inc,PC),ld(PC),bus(MREQ,LDDR),jcc(DI,add0);++PC,M=>DR,ifdirect add0
19 3 3 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1B            alu(add,Rx),ld(AR),j(add1)              ; DR+Rx=>AR
1A 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 00 add0       alu(reg,DR),ld(AR)                      ; DR=>AR,if drct add0
1B 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 00 add1   bus(MREQ,LDDR)                         ; M=>DR
1C 3 2 2 1 1 0 0 0 0 0 0 0 0 0 0 0 1 00 add2     alu(add,Rd,FL),ld(Rd),j(fetch)          ; DR+Rd=>Rd,jump fetch
1D
1D                                          ####SUB####
1D 8 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 1 22 subi     alu(inc,PC),ld(PC),bus(MREQ,LDDR),j(sub2); ++PC,M=>DR
1E 8 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 7 20 sub      alu(inc,PC),ld(PC),bus(MREQ,LDDR),jcc(DI,sub0);++PC,M=>DR,ifdirect sub0
1F 3 3 0 0 0 1 0 0 0 0 0 0 0 0 0 1 21            alu(add,Rx),ld(AR),j(sub1)              ; DR+Rx=>AR
20 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 00 sub0       alu(reg,DR),ld(AR)                      ; DR=>AR,if drct sub0
21 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 00 sub1     bus(MREQ,LDDR)                          ; M=>DR
22 4 2 2 1 1 0 0 0 0 0 0 0 0 0 0 1 00 sub2       alu(sub,Rd,FL),ld(Rd),j(fetch)          ; DR+Rd=>Rd,jump fetch
23
23                                          ####CMP####
23 8 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 1 28 cmpi     alu(inc,PC),ld(PC),bus(MREQ,LDDR),j(cmp2); ++PC,M=>DR
24 8 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 7 26 cmp      alu(inc,PC),ld(PC),bus(MREQ,LDDR),jcc(DI,cmp0);++PC,M=>DR,ifdirect cmp0
25 3 3 0 0 0 1 0 0 0 0 0 0 0 0 0 1 27            alu(add,Rx),ld(AR),j(cmp1)              ; DR+Rx=>AR
26 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 00 cmp0       alu(reg,DR),ld(AR)                      ; DR=>AR,if drct cmp0
27 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 00 cmp1     bus(MREQ,LDDR)                          ; M=>DR
28 4 2 0 0 1 0 0 0 0 0 0 0 0 0 0 1 00 cmp2       alu(sub,Rd,FL),j(fetch)                 ; DR+Rd=>Rd,jump fetch
```

```
A  A R W L L L L L L H E S C R M I J J
D  L R R R F A O D I L R T L W R R P A                                Page(2)
R  U                    I I
29
29                                          ####AND####
29 8 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 1 2E andi    alu(inc,PC),ld(PC),bus(MREQ,LDDR),j(and2); ++PC,M=>DR
2A 8 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 7 2C and     alu(inc,PC),ld(PC),bus(MREQ,LDDR),jcc(DI,and0);++PC,M=>DR,ifdirect and0
2B 3 3 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 2D         alu(add,Rx),ld(AR),j(and1)              ; DR+Rx=>AR
2C 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 00 and0    alu(reg,DR),ld(AR)                      ; DR=>AR,if drct and0
2D 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 00 and1    bus(MREQ,LDDR)                          ; M=>DR
2E 5 2 2 1 1 0 0 0 0 0 0 0 0 0 0 0 1 00 and2    alu(and,Rd,FL),ld(Rd),j(fetch)          ; DR+Rd=>Rd,jump fetch
2F
2F                                          ####OR####
2F 8 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 1 34 ori     alu(inc,PC),ld(PC),bus(MREQ,LDDR),j(or2); ++PC,M=>DR
30 8 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 7 32 or      alu(inc,PC),ld(PC),bus(MREQ,LDDR),jcc(DI,or0); ++PC,M=>DR,if direct or0
31 3 3 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 33         alu(add,Rx),ld(AR),j(or1)               ; DR+Rx=>AR
32 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 00 or0     alu(reg,DR),ld(AR)                      ; DR=>AR,if drct or0
33 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 00 or1     bus(MREQ,LDDR)                          ; M=>DR
34 6 2 2 1 1 0 0 0 0 0 0 0 0 0 0 0 1 00 or2     alu(or,Rd,FL),ld(Rd),j(fetch)           ; DR+Rd=>Rd,jump fetch
35
35                                          ####XOR####
35 8 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 1 3A xori    alu(inc,PC),ld(PC),bus(MREQ,LDDR),j(xor2); ++PC,M=>DR
36 8 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 7 38 xor     alu(inc,PC),ld(PC),bus(MREQ,LDDR),jcc(DI,xor0);++PC,M=>DR,ifdirect xor0
37 3 3 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 39         alu(add,Rx),ld(AR),j(xor1)              ; DR+Rx=>AR
38 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 00 xor0    alu(reg,DR),ld(AR)                      ; DR=>AR,if drct xor0
39 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 00 xor1    bus(MREQ,LDDR)                          ; M=>DR
3A 7 2 2 1 1 0 0 0 0 0 0 0 0 0 0 0 1 00 xor2    alu(xor,Rd,FL),ld(Rd),j(fetch)          ; DR+Rd=>Rd,jump fetch
3B
3B                                          ####SHR/SHL####
3B D 2 2 1 1 0 0 0 0 0 0 0 0 0 0 0 1 00 sft     alu(sft,Rd,FL),ld(Rd),j(fetch)
3C
3C                                          ####JMP/JZ/JC####
3C 8 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 5 00 jmp     alu(inc,PC),ld(PC),bus(MREQ,LDDR),jcc(NJP,fetch);++PC,M=>DR
3D 2 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 7 00         alu(reg,DR),ld(PC),jcc(DI,fetch)        ; DR=>PC,if direct fetch
3E 3 3 1 1 0 0 0 0 0 0 0 0 0 0 0 1 00           alu(add,Rx),ld(PC),j(fetch)             ; DR+Rx=>PC,jump fetch
3F
3F                                          ####CALL###
3F 8 1 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 00 cal     alu(inc,PC),ld(OPR),bus(MREQ,LDDR)      ; PC+1=>OPR,M=>DR
40 9 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 00         alu(dec,SP),ld(SP,AR)                   ; --SP=>AR
41 2 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 7 00       alu(reg,DR),ld(PC),bus(MREQ,OE),jcc(DI,fetch);DR=>PC,OPR=>M,ifdrt fetch
42 3 3 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 00         alu(add,Rx),ld(PC),j(fetch)             ; DR+Rx=>PC,jump fetch
43
43                                          ####IN/OUT(C0)####
43 8 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 00 io      alu(inc,PC),ld(PC),bus(MREQ,LDDR)       ; ++PC,M=>DR
44 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 7 48         alu(reg,DR),ld(AR),jcc(DI,in)           ; DR=>AR,if drct in
45 1 2 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 00         alu(reg,Rd),ld(OPR),bus(OE)             ; Rd=>OPR
46 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 00       bus(IOREQ,OE)                           ; OPR=>IO
47 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 00       bus(OE),j(fetch)                        ; jump fetch
48 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 00 in      bus(IOREQ)                              ; IOW信号出力
49 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 00         bus(LDDR)                               ; IO=>DR
4A 2 0 2 1 0 0 0 0 0 0 0 0 0 0 0 1 00           alu(reg,DR),ld(Rd),j(fetch)             ; DR=>Rd,jump fetch
4B
4B                                          ####PUSH###
4B 1 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 00 psh       alu(reg,Rd),ld(OPR)                     ; Rd=>OPR
4C 9 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 00           alu(dec,SP),ld(SP,AR)                   ; --SP=>AR
4D 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 00       bus(MREQ,OE),j(fetch)                   ; OPR=>M,jump fetch
4E
4E                                          ####PUSHF###
4E B 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 00 pshf    alu(outf),ld(OPR)                       ; flag=>OPR
4F 9 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 00         alu(dec,SP),ld(SP,AR)                   ; --SP=>AR
50 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 00       bus(MREQ,OE),j(fetch)                   ; OPR=>M,jump fetch
```

```
A  A R W L L L L L L H E S C R M I J J
D  L R R R F A O D I L R T L W R R P A                                Page(3)
R  U                     I I
51
51                                          ####POP####
51 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 00 pop     alu(reg,SP),ld(AR)                    ; SP=>AR
52 8 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 00         alu(inc,SP),ld(SP),bus(MREQ,LDDR)     ; SP++,M=>DR
53 2 0 2 1 0 0 0 0 0 0 0 0 0 0 0 0 1 00         alu(reg,DR),ld(Rd),j(fetch)           ; DR=>Rd,jump fetch
54
54                                          ####POPF####
54 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 00 popf    alu(reg,SP),ld(AR)                    ; SP=>AR
55 8 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 00         alu(inc,SP),ld(SP),bus(MREQ,LDDR)     ; SP++,M=>DR
56 C 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 00         alu(inf),j(fetch)                     ; DR=>flag,jump fetch
57
57                                          ####EI####
57 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 00 ei      sti,j(fetch)                          ; set int, jump fetch
58
58                                          ####DI####
58 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 00 di      cli,j(fetch)                          ; clr int, jump fetch
59
59                                          ####RET####
59 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 00 ret     alu(reg,SP),ld(AR)                    ; SP=>AR
5A 8 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 00         alu(inc,SP),ld(SP),bus(MREQ,LDDR)     ; SP++,M=>DR
5B 2 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 00         alu(reg,DR),ld(PC),j(fetch)           ; DR=>PC, jump fetch
5C
5C                                          ####RETI####
5C 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 00 reti    alu(reg,SP),ld(AR),sti                ; SP=>AR, set int
5D 8 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 00         alu(inc,SP),ld(SP),bus(MREQ,LDDR)     ; SP++,M=>DR
5E 2 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 00         alu(reg,DR),ld(PC),j(fetch)           ; DR=>PC, jump fetch
5F
5F                                          ####HALT####
5F 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 00 halt    halt,j(fetch)
```

```
;
; TeC6 OPECODE DOCODE TABLE
;    Tokuyama kousen Educational Computer Ver.6
;
; Copyright (C) 2002-2004 by
;                      Dept. of Computer Science and Electronic Engineering,
;                      Tokuyama College of Technology, JAPAN
;
;   上記著作権者は，Free Software Foundation によって公開されている GNU 一般公
; 衆利用許諾契約書バージョン２に記述されている条件を満たす場合に限り，本ソース
; コード(本ソースコードを改変したものを含む．以下同様)を使用・複製・改変・再配
; 布することを無償で許諾する．
;
;   本ソースコードは＊全くの無保証＊で提供されるものである。上記著作権者および
; 関連機関・個人は本ソースコードに関して，その適用可能性も含めて，いかなる保証
; も行わない．また，本ソースコードの利用により直接的または間接的に生じたいかな
; る損害に関しても，その責任を負わない．
;
fetch   err     err     err     err     err     err     err
err     err     err     err     err     err     err     err
; 1X
ld      ld      ld      ldi     ld      ld      ld      ldi
ld      ld      ld      ldi     ld      ld      ld      ldi
; 2X
st      st      st      err     st      st      st      err
st      st      st      err     st      st      st      err
; 3X
add     add     add     addi    add     add     add     addi
add     add     add     addi    add     add     add     addi
; 4X
sub     sub     sub     subi    sub     sub     sub     subi
sub     sub     sub     subi    sub     sub     sub     subi
; 5X
cmp     cmp     cmp     cmpi    cmp     cmp     cmp     cmpi
cmp     cmp     cmp     cmpi    cmp     cmp     cmp     cmpi
; 6X
and     and     and     andi    and     and     and     andi
and     and     and     andi    and     and     and     andi
; 7X
or      or      or      ori     or      or      or      ori
or      or      or      ori     or      or      or      ori
; 8X
xor     xor     xor     xori    xor     xor     xor     xori
xor     xor     xor     xori    xor     xor     xor     xori
; 9X
sft     sft     sft     sft     sft     sft     sft     sft
sft     sft     sft     sft     sft     sft     sft     sft
; AX
jmp     jmp     jmp     err     jmp     jmp     jmp     err
jmp     jmp     jmp     err     jmp     jmp     jmp     err
; BX
err     err     err     err     err     err     err     err
err     err     err     err     cal     cal     cal     err
; CX
io      err     err     io      io      err     err     io
io      err     err     io      io      err     err     io
; DX
psh     err     pop     err     psh     err     pop     err
psh     err     pop     err     psh     pshf    pop     popf
; EX
ei      err     err     di      err     err     err     err
err     err     err     err     ret     cal     cal     reti
; FX
err     err     err     err     err     err     err     err
err     err     err     err     err     err     err     halt
; END TABLE
```