T-Monitor仕様書

TEF-020-S002-01.00.01
2006年 4月

T-Monitor仕様書（Ver.1.00.01）

本仕様書に関しては、下記にお問い合わせください。

T-Engineフォーラム事務局
〒141-0031　東京都品川区西五反田2-20-1　第28興和ビル
YRPユビキタス・ネットワーキング研究所内
TEL：03-5437-0572　　FAX：03-5437-2399
E-mail：office@www.t-engine.org

目次

【修正履歴】

Version 1.00.01

・Copyright 表記の変更
・書誌事項を追加
・インデントなどを調整
・表記の統一: 割り込み → 割込み、読込み → 読み込み、書込み → 書き込み
・誤植訂正：0c0b → 0x0b、SearchChar(SC) → Search(SC)
・削除：「・転送メモリーに書き込むことは～ことはありません。」(直後の内容と重複)
・コメントの表現を変更：// ～ → /* ～ */

Version 1.00.00

・誤植訂正：プレークポイント → ブレークポイント(4箇所)
・Copyright 表記の変更。

Version 1.B0.01

・モニタコマンドに、プロセスの強制終了(Kill)を追加した。
・モニタサービス関数に、サポート拡張サービス機能(tm_extsvc)を追加した。

1. 概要

T-Monitorは、T-Engineの基本モニタとしてROMに標準的に搭載され、以下の機能を持っています。

(1) システム機能

ハードウェア初期化
システムの起動
例外/割込み/トラップ処理機能

(2) デバッグ機能

メモリ操作
レジスタ操作
I/O操作
逆アセンブル
プログラム/データのロード
プログラム実行
ブレークポイント操作
トレース実行
ディスクの読込み/書込み/ブート

(3) プログラムサポート機能

モニタサービス関数

T-Monitorは、CPUやT-Engineボードのハードウェアに依存する部分が大きいため、本仕様書では共通仕様のみを規定しています。実際のT-Engineボードに対応した個別のT-Monitorの詳細は実装ごとに規定されます。

2. システム機能

2.1 ハードウェア初期化

システムがリセットされるとモニタが最初に起動されて以下の処理を行います。

(1) ハードウェアの初期化

システム起動に必要なハードウェアの基本部分の初期化を行います。

(2) ハードウェアの自己診断

メモリチェック等の必要な自己診断を行います。
システムに異常が検出された場合には、異常を通知してシステムの起動を中止します。また、デバッグ用シリアルポートが使用できる状態の場合には、デバッグ用シリアルポートにエラーメッセージを出力し、モニタのコマンド入力待ちとなります。

2.2 システムの起動

ハードウェア初期化が終了すると、以下のいずれかの起動モードにしたがってシステムを起動します。

(1) 自動起動モード

システムに装着されているディスクデバイスを順に検索し、最初に見つけたブート可能なディスクからブートしてシステムを起動します。(BootDiskコマンドと同等)
ブート可能なディスクがなかったときは、ROMに搭載されたシステムを起動します。ROMにシステムが搭載されていなかったときは、モニタのコマンド入力待ちとなります。

(2) モニタ起動モード

システムを起動せずに、モニタのコマンド入力待ちとなります。

実装によっては、上記以外の起動モードがある場合があります。起動モードの設定はボード上のDIPスイッチや不揮発メモリ上のデータなどにより行われますが、具体的な設定方法は実装に依存します。

2.3 例外/割込み/トラップ処理機能

すべての例外/割込み/トラップに対して、一元化されたベクトルテーブルを割り当てて、そこに登録されたハンドラを実行する処理を行います。具体的なベクトルテーブルの内容は実装ごとに規定されます。

未定義のベクトルに対しては、モニタの例外ハンドラが起動されるように初期化します。これにより、未定義の例外/割込み/トラップが発生した場合は、その情報がデバッグ用シリアルポートに出力され、モニタに制御が移行します。

なお、基本的にモニタ内では割込みは使用しません。モニタの動作中はすべての割込みは禁止状態となります。

3. デバッグ機能

3.1 コンソールの接続

T-Engineのデバッグ用シリアルポート(RS232C)にデバッグ用コンソールを接続してデバッグ用に使用します。接続時の通信仕様は以下の通りです。

通信速度	38,400 bps (または115,200 bps)
データ長	8 ビット
ストップビット	1 ビット
パリティ	なし
フロー制御	XON/XOFF
文字コード	ASCIIコード
受信行末	CR (0x0d)
送信行末	CRLF (0x0d, 0x0a)

※ 通信速度はボード上のDIPスイッチや不揮発メモリ上のデータなどにより行われますが、具体的な設定方法は実装に依存します。

3.2 コマンド形式

(1) コマンド

モニタはデバッグ用コンソールに、"TM> " のプロンプトを表示して、コマンド入力を待ちます。コマンドは以下の形式で入力します。1行は最大256文字までです。

〈コマンド名〉 〈パラメータ 1〉, 〈パラメータ 2〉, ... 〈改行〉

- *〈コマンド名〉*の大文字/小文字は区別されません。
- *〈コマンド名〉*と*〈パラメータ〉*は空白またはタブで区切ります。
- *〈パラメータ〉*は、','で区切ります。パラメータの一部を省略するときは、','のみを入力することによって〈パラメータ〉の対応を示します。

コマンドは ';' で区切って1行に複数のコマンドを書くことができます。
'*' で始まる行はコメント行として無視されます。改行だけの空行も無視されます。

(2) 制御コード

デバッグ用コンソールからの以下の制御コードの入力に対応します。

Ctrl-X (0x18), Ctrl-U (0x15)	入力行の取り消し
Ctrl-H (0x08), DEL (0x7f)	入力文字の1文字の取り消し
Ctrl-S (0x13, XOFF)	表示の一時中断
Ctrl-Q (0x11, XON)	表示の再開
Ctrl-C (0x03)	コマンドの強制終了
Ctrl-F (0x06), ESC [C	カーソルを右に移動(→)
Ctrl-B (0x02), ESC [D	カーソルを左に移動(←)
Ctrl-P (0x10), ESC [A	前の入力行の呼び出し(↑)
Ctrl-N (0x0e), ESC [B	次の入力行の呼び出し(↓)
Ctrl-K (0x0b)	カーソル以降を削除

(3) 数値

数値は以下の形式で記述します。

16 進数	H'<16進数字列>	h'<16進数字列>
	0x<16進数字列>	<0～9><16進数字列>
10 進数	D'<数字列>	d'<数字列>
8 進数	Q'<8進数字列>	q'<8進数字列>
2 進数	B'<2進数字列>	b'<2進数字列>

<数字> : '0'～'9'
<2 進数字> : '0','1'
<8 進数字> : '0'～'7'
<16 進数字> : '0'～'9', 'A'～'F', 'a'～'f'

先頭に何も付けない数字列は16進数とみなされます。

(4) 文字列

文字列は、'"' で囲まれた任意の文字列で、一部のコマンドのパラメータとして使用されます。

(5) レジスタ名

レジスタ名は、CPUに依存した特殊なシンボルで、一部のコマンドのパラメータとして使用されます。

(6) 式

式は、数値またはレジスタ名を '+', '-', '*', '/' の演算子でつなげたもので、コマンドのパラメータとして記述できます。'*', '/' を含む演算も常に左から行われます。レジスタ名は、指定したレジスタの値を意味します。

(例) 8000 + d'250 -- H'80FA
1000 + 100 * 2 -- (H'1000 + H'100) * 2
R0 + 100 -- レジスタ R0 の値 + h'100

'&' は間接参照を示す演算子で、それまでの式の値のメモリアドレスの内容(ワード)を式の値とします。'&' を連続して書くことにより多段の間接参照もできます。

(例) AC000000 + 4 & -- H'AC0000004 のメモリ内容
R0 & + 8 & -- レジスタ R0 の値のメモリ内容 + 8 を
アドレスとするメモリ内容

すべてのコマンドの数値パラメータ(アドレスやサイズなど)に式が使用できます。

3.3 コマンド一覧

コマンドの説明では以下の表記を使用しています。

(～)	コマンドの省略形を示す
[～]	省略可能を示す
[～]..	省略可能の繰り返しを示す
{～|～}	選択を示す
バイト	8 ビット
ハーフワード	16 ビット
ワード	32 ビット

コマンド一覧を以下に示します。

〈コマンド名〉	〈コマンド説明〉	
Dump(D)	Dump Memory	メモリ内容の表示
DumpByte(DB)	Dump Memory	メモリ内容の表示
DumpHalf(DH)	Dump Memory	メモリ内容の表示
DumpWord(DW)	Dump Memory	メモリ内容の表示
Modify(M)	Modify Memory	メモリ内容の変更
ModifyByte(MB)	Modify Memory	メモリ内容の変更
ModifyHalf(MH)	Modify Memory	メモリ内容の変更
ModifyWord(MW)	Modify Memory	メモリ内容の変更
Fill(F)	Fill Memory	メモリ内容の埋め込み
FillByte(FB)	Fill Memory	メモリ内容の埋め込み
FillHalf(FH)	Fill Memory	メモリ内容の埋め込み
FillWord(FW)	Fill Memory	メモリ内容の埋め込み
SearchChar(SC)	Search Memory	メモリ内容のサーチ
SearchByte(SCB)	Search Memory	メモリ内容のサーチ
SearchHalf(SCH)	Search Memory	メモリ内容のサーチ
SearchWord(SCW)	Search Memory	メモリ内容のサーチ
Compare(CMP)	Compare Memory	メモリ内容の比較
Move(MOV)	Move Memory	メモリ内容の転送
InputByte(IB)	Input Port	I/Oポートからの入力
InputHalf(IH)	Input Port	I/Oポートからの入力
InputWord(IW)	Input Port	I/Oポートからの入力
OutputByte(OB)	Output Port	I/Oポートへの出力
OutputHalf(OH)	Output Port	I/Oポートへの出力
OutputWord(OW)	Output Port	I/Oポートへの出力
Disassemble(DA)	Disassemble	逆アセンブル
Register(R)	Register Dump/Modify	レジスタの表示/変更
Go(G)	Go Program	プログラムの実行
BreakPoint(B)	Set Break Point	ブレークポイント設定
BreakClear(BC)	Clear Break Point	ブレークポイントクリア
Step(S)	Step Trace	ステップトレース実行
Next(N)	Next Trace	ネキストトレース実行
BackTrace(BTR)	Back Trace	バックトレース表示
Load(LO)	Load Program/Data	プログラム/データのロード
ReadDisk(RD)	Read Disk	ディスクからの読み込み
WriteDisk(WD)	Write Disk	ディスクへの書み込み
InfoDisk(ID)	Display Disk Information	ディスク情報の表示
BootDisk(BD)	Boot from Disk	ディスクからのブート
Kill(KILL)	Kill Process	プロセスの強制終了
help(H)	Help Message	ヘルプメッセージの表示
Exit(EX)	Exit Monitor	モニタの終了

Dump Memory

メモリ内容の表示

【形式】

```
Dump(D)       [<開始アドレス>][, {<終了アドレス>|#<データ数>}]
DumpByte(DB) [<開始アドレス>][, {<終了アドレス>|#<データ数>}]
DumpHalf(DH) [<開始アドレス>][, {<終了アドレス>|#<データ数>}]
DumpWord(DW) [<開始アドレス>][, {<終了アドレス>|#<データ数>}]
```

【説明】

指定したアドレス範囲のメモリ内容を以下の<単位>で表示します。

Dump, DumpByte	バイト単位	<データ数>はバイト数
DumpHalf	ハーフワード単位	<データ数>はハーフワード数
DumpWord	ワード単位	<データ数>はワード数

対象となるアドレス範囲は以下のいずれかです。

<開始アドレス> ～ <終了アドレス> + <単位> - 1
<開始アドレス> ～ <開始アドレス> + <データ数> * <単位> - 1

<開始アドレス>、<終了アドレス>が<単位>バイト境界でないときは、<単位>バイト境界に調整されます。

<開始アドレス>を省略すると、前回の Dump Memory コマンドの次のアドレスから表示します。

<終了アドレス>を省略すると、<単位>にかかわらず 64 バイト分表示します。

対象アドレス範囲内のメモリのみを指定された単位でアクセスします。メモリへの書き込みは一切行いません。

【使用例】

```
TM> Dump AC100000
AC100000: 00 09 80 04 45 03 E0 05 E0 09 00 0A 00 0B 56 0C ....E.........V.
AC100010: 04 0D 00 0E 03 01 E0 03 E1 05 E8 FF 8E 00 00 00 ................
AC100020: 1B D6 1B D6 1B D6 1B D6 9E 00 00 00 8E 00 01 C0 ................
AC100030: C6 16 D0 0C 00 FF 80 46 80 10 00 00 88 12 22 4C .......F......"L

TM> DumpHalf AC100000, AC100010
AC100000: 0900 0480 0345 05E0 09E0 0A00 0B00 0C56  ....E.........V.
80100010: 040D                                     ..

TM> DumpWord AC100000, #9
AC100000: 04800900 05E00345 0A0009E0 0C560B00  ....E.........V.
AC100010: 0E000D04 03E00103 FFE805E1 0000008E  ................
AC100020: D61BD61B                             ....
```


Modify Memory

メモリ内容の変更

【形式】

```
Modify(M)       [<開始アドレス>][, <設定値>]..
ModifyByte(MB)  [<開始アドレス>][, <設定値>]..
ModifyHalf(MH)  [<開始アドレス>][, <設定値>]..
ModifyWord(MW)  [<開始アドレス>][, <設定値>]..
```

【説明】

指定した<開始アドレス>のメモリ内容を以下の<単位>で変更します。

Modify, ModifyByte	バイト単位
ModifyHalf	ハーフワード単位
ModifyWord	ワード単位

<開始アドレス>が<単位>バイト境界でないときは、<単位>バイト境界に調整されます。
<開始アドレス>を省略すると、前回の Modify Memory コマンドの次のアドレスから変更します。
<設定値>には<式>または<文字列>を指定します。<式>は<単位>バイトの値として設定されます。<文字列>は<単位>バイト境界になるように最後に 0 が詰められたバイトデータ列として設定されます。<設定値>は最大 128 バイト分まで続けて指定できます。
<設定値>を省略すると、対話形式によりメモリ内容を変更します。対話形式では以下の入力は特別な意味となります。

'.'	コマンドを終了する。
'^'	1 つ前のアドレスに戻る。
(改行のみ)	設定せずに次のアドレスに進む。

対象アドレス範囲内のメモリのみを指定された単位でアクセスします。対話形式の場合以外はメモリの読み込みは一切行いません。

【使用例】

```
TM> ModifyByte AC100000
AC100000: 00 -> 12
AC100001: 09 -> 34
AC100002: 80 -> ^
AC100001: 34 -> .

TM> ModifyHalf AC100000, "ABCD", 56, 78

TM> ModifyWord AC100000
AC100000: 44434241 ->
AC100004: 00780056 -> .
```


Fill Memory

メモリ内容の埋め込み

【形式】

```
Fill(F)       <開始アドレス>,{<終了アドレス>|#<データ数>},<設定値>[,<設定値>]..
FillByte(FB) <開始アドレス>,{<終了アドレス>|#<データ数>},<設定値>[,<設定値>]..
FillHalf(FH) <開始アドレス>,{<終了アドレス>|#<データ数>},<設定値>[,<設定値>]..
FillWord(FW) <開始アドレス>,{<終了アドレス>|#<データ数>},<設定値>[,<設定値>]..
```

【説明】

指定したアドレス範囲のメモリに<設定値>列を以下の<単位>で繰り返して埋め込みます。

Fill, FillByte	バイト単位	<データ数>はバイト数
FillHalf	ハーフワード単位	<データ数>はハーフワード数
FillWord	ワード単位	<データ数>はワード数

対象となるアドレス範囲は以下のいずれかです。

<開始アドレス> ～ <終了アドレス> + <単位> - 1
<開始アドレス> ～ <開始アドレス> + <データ数> * <単位> - 1

<開始アドレス>、<終了アドレス>が<単位>バイト境界でないときは、<単位>バイト境界に調整されます。

<設定値>には<式>または<文字列>を指定します。<式>は<単位>バイトの値として設定されます。<文字列>は<単位>バイト境界になるように最後に 0 が詰められたバイトデータ列として設定されます。<設定値>は最大 128 バイト分まで続けて指定できます。

対象アドレス範囲内のメモリのみを指定された単位でアクセスします。メモリの読み込みは一切行いません。

【使用例】

```
TM> Fill AC101000, #10, 57

TM> Dump AC101000, #12
AC101000: 57 57 57 57 57 57 57 57 57 57 57 57 57 57 57 57 WWWWWWWWWWWWWWWW
AC101010: 00 00                                           ..

TM> FillWord AC101000, AC10101f, 12, 34

TM> Dump AC101000, #22
AC101000: 12 00 00 00 34 00 00 00 12 00 00 00 34 00 00 .......4.......4
AC101010: 12 00 00 00 34 00 00 00 12 00 00 00 34 00 00 .......4.......4
AC101020: 00 00                                       ..
```


Search Memory

メモリ内容のサーチ

【形式】

```
Search(SC)       <開始アドレス>,{<終了アドレス>|#<データ数>},<検索値>[,<検索値>]..
SearchByte(SCB) <開始アドレス>,{<終了アドレス>|#<データ数>},<検索値>[,<検索値>]..
SearchHalf(SCH) <開始アドレス>,{<終了アドレス>|#<データ数>},<検索値>[,<検索値>]..
SearchWord(SCW) <開始アドレス>,{<終了アドレス>|#<データ数>},<検索値>[,<検索値>]..
```

【説明】

指定したアドレス範囲のメモリ内容に<検索値>列があるかどうかを以下の<単位>で検索し、<検索値>列が見つかったときは、その先頭のアドレスを表示します。最大 64個の表示を行った時点で検索を打ち切ります。

Search, SearchChar, SearchByte	バイト単位	<データ数>はバイト数
SearchHalf	ハーフワード単位	<データ数>はハーフワード数
SearchWord	ワード単位	<データ数>はワード数

対象となるアドレス範囲は以下のいずれかです。

<開始アドレス> ～ <終了アドレス> + <単位> - 1
<開始アドレス> ～ <開始アドレス> + <データ数> * <単位> - 1

<開始アドレス>、<終了アドレス>が<単位>バイト境界でないときは、<単位>バイト境界に調整されます。

<検索値>には<式>または<文字列>を指定します。<式>は<単位>バイトの値として検索されます。<文字列>は<単位>バイト境界になるように最後に 0 が詰められたバイトデータ列として検索されます。<検索値>は最大 128 バイト分まで続けて指定できます。

対象アドレス範囲内のメモリのみを指定された単位でアクセスします。メモリへの書き込みは一切行いません。

【使用例】

```
TM> SearchChar AC101000, AC10101f, 12
AC101003:
AC10100B:
AC101013:
AC10101B:

TM> SearchWord AC101000, #20, 12, 34
AC101000:
AC101008:
AC101010:
AC101018:
```


Compare Memory

メモリ内容の比較

【形式】

Compare(CMP) 〈開始アドレス〉, {〈終了アドレス〉|#〈バイト数〉}, 〈比較先アドレス〉

【説明】

指定したアドレス範囲のメモリ内容を〈比較先アドレス〉からのメモリ内容と比較し、内容が異なっているアドレスとメモリ内容をバイト単位で表示します。最大 64 個の表示を行った時点で比較を打ち切ります。

対象となるアドレス範囲は以下のいずれかです。

〈開始アドレス〉 ～ 〈終了アドレス〉
〈開始アドレス〉 ～ 〈開始アドレス〉 + 〈バイト数〉 - 1

対象アドレス範囲内のメモリのみをバイト単位でアクセスします。メモリへの書き込みは一切行いません。

【使用例】

```
TM> Compare AC100000, AC100fff, AC110000

TM> Compare AC100000, AC100fff, AC120000
AC100020: 34 -> AC120000: 00
AC100021: 56 -> AC120000: 00
        :       :
```


Move Memory

メモリ内容の転送

【形式】

Move(MOV) 〈開始アドレス〉, {〈終了アドレス〉|#〈バイト数〉}, 〈転送先アドレス〉

【説明】

指定したアドレス範囲のメモリ内容を〈転送先アドレス〉に転送します。転送元と転送先のアドレス範囲が重複しても構いません。

対象となるアドレス範囲は以下のいずれかです。

〈開始アドレス〉 ～ 〈終了アドレス〉
〈開始アドレス〉 ～ 〈開始アドレス〉 + 〈バイト数〉 - 1

対象アドレス範囲内のメモリのみをバイト単位でアクセスします。転送元メモリへの書き込みや転送先メモリの読み込みは一切行いません。

【使用例】

TM> Move AC100000, #1000, AC110000

Input Port

I/O ポートからの入力

【形式】

InputByte(IB) <I/Oアドレス>
InputHalf(IH) <I/Oアドレス>
InputWord(IW) <I/Oアドレス>

【説明】

指定した<I/Oアドレス>から以下の<単位>でデータを読み込んで表示します。

InputByte	バイト単位
InputHalf	ハーフワード単位
InputWord	ワード単位

<I/Oアドレス>が<単位>バイト境界でないときは、エラーとなります。

指定した I/O アドレスのみを指定された単位でアクセスします。I/O ポートへの書き込みは一切行いません。

メモリマップド I/O のシステムでは、I/O アドレスとして指定したメモリアドレスを読み込みます。

【使用例】

```
TM> InputByte 310
310: 5F
```


Output Port

I/O ポートへの出力

【形式】

OutputByte(OB)　<I/Oアドレス>,<バイトデータ>
OutputHalf(OH)　<I/Oアドレス>,<ハーフワードデータ>
OutputWord(OW)　<I/Oアドレス>,<ワードデータ>

【説明】

指定した<I/Oアドレス>に以下の<単位>でデータを書き込みます。

OutputByte	バイト単位
OutputHalf	ハーフワード単位
OutputWord	ワード単位

<I/Oアドレス>が<単位>バイト境界でないときは、エラーとなります。
指定した I/O アドレスのみを指定された単位でアクセスします。I/O ポートへの読み込みは一切行いません。
メモリマップド I/O のシステムでは、I/O アドレスで指定したメモリアドレスへ書き込みます。

【使用例】

```
TM> OutputHalf 310, 513F
```


Disassemble

逆アセンブル

【形式】

Disassemble(DA)　[<開始アドレス>][, <命令ステップ数>]

【説明】

　指定した<開始アドレス>から、指定した<命令ステップ数>分を逆アセンブルした結果を表示します。

　<開始アドレス>を省略すると、前回の Disassemble コマンドの次のアドレスから逆アセンブルします。ただし、プログラムの実行後にブレークや例外などでモニタに戻ってきた時は、その時点の PC レジスタの値が Disassemble コマンドの<開始アドレス>になります。

　<ステップ数>を省略すると、16 ステップ分逆アセンブルします。

　※ 逆アセンブル機能の有無は実装依存となります。

【使用例】

TM> Disassemble AC1000d8
　　<逆アセンブル結果>

Register Dump/Modify

レジスタの表示/変更

【形式】

Register(R)　[<レジスタ名>[, <設定値>]]

【説明】

指定した<レジスタ名>の内容を変更します。<設定値>を省略すると、指定した<レジスタ名>の内容を表示します。

指定可能な<レジスタ名>は、CPU に依存します。大文字/小文字は区別されません。

<レジスタ名>として、<レジスタ>群を示す 1 文字の以下の名前が使用できます。

G	汎用レジスタ
C	制御/システムレジスタ
D	DSPレジスタ
F	浮動小数点レジスタ
A	全レジスタ

<レジスタ名>を省略した場合は、汎用レジスタ全部を表示します。

※ 具体的なレジスタ名は実装依存となります。

【使用例】

```
TM> Register
        <汎用レジスタ全部の表示>

TM> Register C
        <制御/システムレジスタ全部の表示>

TM> Register R0, 1234567

TM> Register R0
R0 : 01234567
```


Go Program

プログラムの実行

【形式】

Go(G)　[<実行開始アドレス>][, <実行終了アドレス>]

【説明】

指定した<実行開始アドレス>からプログラムを実行します。<実行終了アドレス>は一時的なソフトウェアブレークポイントとして設定され、<実行終了アドレス>に到達した時点でモニタに戻ります。
<実行開始アドレス>を省略すると、現在の PC レジスタの値のアドレスから実行します。
プログラムからは以下のいずれかの場合にモニタに戻ります。

・設定したブレークポイントに達したとき。
・プログラムで対応していない例外が発生したとき。
・プログラムからモニタサービス関数によりモニタに入ったとき。

【使用例】

```
TM> Go AC1000d8, AC10434
Break (S) at AC10434            ※ at XXXX は次に実行される命令の PC
```


Set Break Point

ブレークポイントの設定

【形式】

BreakPoint(B)　[<ブレークアドレス>[,<ブレーク属性>][, <実行コマンド>]]

【説明】

指定した<ブレークアドレス>に指定した<ブレーク属性>のブレークポイントを設定します。パラメータを省略すると、設定されている全てのブレークポイントを表示します。

<ブレーク属性>は以下のいずれかを指定します。省略すると S になります。

S ：<ブレークアドレス>の命令に到達したら、その命令を実行する直前に停止します。
E ：<ブレークアドレス>の命令に到達したら停止します。
R ：<ブレークアドレス>のメモリをリードしたら停止します。
W ：<ブレークアドレス>のメモリをライトしたら停止します。
RW：<ブレークアドレス>のメモリをリードまたはライトしたら停止します。

S はソフトウエアブレークポイントで、ブレークアドレスにトラップ命令等を埋め込んでソフトウエア的に停止させる方法です。そのため、メモリに書き込む必要があり、ROM などの書き込めない領域にある命令にはブレークポイントを設定できません。ソフトウエアブレークポイントは、最大 8 個まで設定できます。

E, R, W, RW はハードウエアブレークポイントで、ハードウエアの機能によって停止します。ROM などの書き込めないメモリ上の命令に対してもブレークポイントが設定できます。なお、ブレーク条件を満たした直前に停止するか、直後に停止するかは、ハードウエアの機能に依存します。ハードウエアが、直前/直後の両方に対応している場合は、直前を標準とします。この場合、直後を選択するためのブレーク属性 + が指定できる場合があります。

R, W, RW には、オペランドサイズを指定することができる場合があります。オペランドサイズを指定できる場合、R, W, RW に続けて :B（バイト）:H（ハーフワード）:W（ワード）を指定します。オペランドサイズが指定されなかった場合は :B（バイト）として扱います。

ブレーク属性の例

E+	命令実行直後に停止
R:W	ワードデータをリードする直前に停止
RW+	バイトデータをリードまたはライトした直後に停止

ハードウエアブレークポイントはハードウエア機能に依存するため、詳細はハードウエアごとに異なります。ハードウエアブレークポイントの機能がまったくない場合や、逆に、上記以外の機能を持つ場合もあります。

<実行コマンド>はブレークしたときに実行するモニタのコマンド文字列を最大 80 文字で指定します。<実行コマンド>に Go コマンドを入れるとブレーク後に自動的に継続して実行されます。

※ 具体的に対応しているブレーク属性は実装依存となります。

【使用例】

```
TM> BreakPoint AC100100, "Register R0; Go"
```


Clear Break Point

ブレークポイントのクリア

【形式】

BreakClear(BC)　[<ブレークアドレス>][, <ブレークアドレス>]..

【説明】

　指定した<ブレークアドレス>の設定をクリアします。<ブレークアドレス>を省略すると、設定されている全てのブレークポイントをクリアします。

【使用例】

TM> BreakClear AC100100

Step Trace

ステップトレース実行

【形式】

Step(S)　[<実行開始アドレス>][, <命令ステップ数>]

【説明】

　指定した<実行開始アドレス>から指定した<命令ステップ数>だけ、実行した命令の逆アセンブル表示を行いながらプログラムをトレース実行します。
　逆アセンブル表示される命令は、次に実行される命令です。つまり、1 ステップ実行した後に次に実行する命令を表示します。
　逆アセンブル機能が実装されていない場合はアドレスとメモリ内容のみが表示されます。
　<命令ステップ数>を省略すると 1 ステップとなります。<実行開始アドレス>を省略すると、現在の PCレジスタからトレース実行します。トレース実行中はすべてのブレークポイントは無効になります。

【使用例】

```
TM> Step , 4
   <逆アセンブル表示-1>
   <逆アセンブル表示-2>
   <逆アセンブル表示-3>
   <逆アセンブル表示-4>
```


Next Trace

ネキストトレース実行

【形式】

Next(N)　[<実行開始アドレス>][, <命令ステップ数>]

【説明】

指定した<実行開始アドレス>から指定した<命令ステップ数>だけ、実行した命令の逆アセンブル表示を行いながらプログラムをトレース実行します。サブルーチン呼び出し命令のときは、サブルーチン全体を 1 つの命令とみなしてトレースします。

逆アセンブル表示される命令は、次に実行される命令です。つまり、1 ステップ実行した後に次に実行する命令を表示します。

逆アセンブル機能が実装されていない場合はアドレスとメモリ内容のみが表示されます。

<命令ステップ数>を省略すると 1 ステップとなります。<実行開始アドレス>を省略すると、現在の PCレジスタからトレース実行します。トレース実行中はすべてのブレークポイントは無効になります。

※ ネキストトレース実行機能の有無は実装依存となります。

【使用例】

```
TM> Next, 4
   <逆アセンブル表示-1>
   <逆アセンブル表示-2>
   <逆アセンブル表示-3>
   <逆アセンブル表示-4>
```


Back Trace

バックトレース表示

【形式】

BackTrace(BTR) [<フレームポインタ>][, <表示履歴数>]

【説明】

BackTrace コマンドは、現在のフレームポインタまたはパラメータで指定したフレームポインタの値からスタック内に保存されている関数呼び出しの履歴を表示します。

現在のフレームポインタからの履歴表示の場合は、最初に現在の PC レジスタの値を表示します。パラメータでフレームポインタを指定した場合には、PC レジスタは表示されません。続いて関数呼び出しのリターンアドレスの履歴をさかのぼって表示します。

<表示履歴数>が指定されなかった場合は 16、指定された場合はその数を最大数として、可能な数だけ履歴をさかのぼって表示します。

※ バックトレース表示機能の有無は実装依存となります。また、スタックに関数呼び出しの履歴が残されていないときは正しく動作しません。

【使用例】

```
TM> BackTrace, 2
PC = 80101758
 <-- 80100420
 <-- 80100016
```


Load Program/Data

プログラム/データのロード

【形式】

Load(LO) 〈プロトコルとデータ形式〉[, 〈ロード開始アドレス〉]

【説明】

デバッグ用コンソールからシリアルポートを通してプログラムやデータをメモリ上にロードします。

〈プロトコルとデータ形式〉は、下記のいずれかを指定します。

〈ロード開始アドレス〉は、ロードするメモリアドレスを指定します。データ形式によって指定する場合と指定しない場合があります。

	〈プロトコル〉	〈データ形式〉	〈ロード開始アドレス〉
S	無手順	S-フォーマット(S3)	不要
XS	XMODEM	S-フォーマット(S3)	不要
XM	XMODEM	メモリイメージデータ(無変換)	必要

【使用例】

```
TM> Load XS
Loaded: AC100000 -> AC1023f8

TM> Load XM, AC120000
Loaded: AC120000 -> AC12FFFF
```


Read Disk

ディスクからの読込み

【形式】

ReadDisk(RD) 〈デバイス名〉, 〈開始ブロック番号〉, 〈ブロック数〉, 〈メモリアドレス〉

【説明】

〈デバイス名〉で指定したディスクの〈開始ブロック番号〉から〈ブロック数〉分、ディスクから〈メモリアドレス〉へデータを読み出します。

ブロックサイズは、デバイスやメディアごとに規定されます。

（例） pca　　PCカード(ATA/CF) #1
pcb　　PCカード(ATA/CF) #2
※ 具体的なデバイス名は実装依存となります。

【使用例】

TM> ReadDisk pca, 1, 20, AC140000

Write Disk

ディスクへの書き込み

【形式】

WriteDisk(WD) 〈デバイス名〉, 〈開始ブロック番号〉, 〈ブロック数〉, 〈メモリアドレス〉

【説明】

〈デバイス名〉で指定したディスクの〈開始ブロック番号〉から〈ブロック数〉分、〈メモリアドレス〉からディスクへデータを書き込みます。

ブロックサイズは、デバイスやメディアにより異なります。

（例） pca　　PCカード(ATA/CF) #1
pcb　　PCカード(ATA/CF) #2
※ 具体的なデバイス名は実装依存となります。

【使用例】

```
TM> WriteDisk hda, 100, 20, AC140000
```


Display Disk Information

ディスク情報の表示

【形式】

InfoDisk(ID) 〈デバイス名〉

【説明】

〈デバイス名〉で指定したディスクの情報を表示します。表示される情報は以下のものです。

1ブロックのバイト数
全ブロック数

※ 具体的なデバイス名は実装依存となります。

【使用例】

```
TK> InfoDisk pca
Format: Bytes/block: 512  Total blocks: 8192
```


Boot from Disk

ディスクからのブート

【形式】

BootDisk(BD)　[<デバイス名>]

【説明】

<デバイス名>で指定したディスクからブートします。指定されたディスクからブートできない場合は、モニタのコマンド入力待ちに戻ります。

<デバイス名>が省略された場合は、ディスクを検索して最初に見つかったブート可能なディスクからブートします。ディスクの検索順序は、取り外しが可能なディスク、取り外しが不可能なディスクの順となりますが、詳細は実装依存となります。ブート可能なディスクがないときは、モニタのコマンド入力待ちに戻ります。

（例）pca0　　PCカード(ATA/CF) #1 の先頭区画
　　pcb1　　PCカード(ATA/CF) #2 の2番目の区画
　　※ 具体的なデバイス名は実装依存となります。

区画(パーティション)を持つディスクを<デバイス名>として指定した場合、指定された区画が起動区画に設定されていなくても、その区画からブートします。

【使用例】

TM> BootDisk

Kill Process プロセスの強制終了

【形式】

Kill

【説明】

アプリケーションプロセスに例外が発生してモニタに移行したときに、例外が発生したプロセスを強制終了して、システムの全体の動作を継続します。Kill コマンドを実行するとモニタには戻りません。

※ アプリケーションプロセスの概念は上位のOS/ミドルウェアなどで実現されるため、プロセス強制終了機能の有無は実装依存となります。

【使用例】

TM> Kill

Help Message

ヘルプメッセージの表示

【形式】

```
Help(H)  [<コマンド名>]
?        [<コマンド名>]
```

【説明】

指定した<コマンド名>のコマンドの使用方法を表示します。
<コマンド名>を省略したとき、あるいは<コマンド名>が間違っているときは、コマンドの一覧を表示します。

【使用例】

```
TM> ? DumpByte
DumpByte(DB) [<start_addr>][, {<end_addr>|#<data_cnt>}]
```


Exit Monitor

モニタの終了

【形式】

Exit(EX)　[<パラメータ>]

【説明】

モニタを終了して、システムを終了します。
<パラメータ>が 0 または省略したときは、システムを停止して電源をオフします。
<パラメータ>が -1 のときは、システムをリセットして再起動します。

【使用例】

TM> Exit

4. プログラムサポート機能

モニタでは、プログラムで利用するために以下のモニタサービス関数を提供しています。

Enter Monitor	モニタへ入る
Get Character	コンソールから1文字入力
Put Character	コンソールへ1文字出力
Get Line	コンソールから1行入力
Put String	コンソールへ文字列出力
Execute Command	モニタコマンドの実行
Read Disk	ディスクからの読み込み
Write Disk	ディスクへの書き込み
Info Disk	ディスク情報の取り出し
System Exit	システム終了
Extension SVC	拡張サービス機能

エラーを返す関数では、エラーコードは T-Kernel と同じものを使用します。

モニタサービス関数のアセンブラでの呼び出し形式は CPU に依存しますが、C 言語から呼び出すための C ライブラリ関数が提供されます。

	Enter Monitor
モニタへ入る	

【C ライブラリ関数呼び出し形式】

```
void tm_monitor( void )
```

【リターン値】

なし

【説明】

プログラムからモニタへ入り、モニタのコマンド入力待ちになります。
モニタの Go コマンドによりプログラムの実行を再開することができます。正常に再開できれば、tm_monitor() から戻ります。

Get Character

コンソールから 1 文字入力

【C ライブラリ関数呼び出し形式】

```
INT tm_getchar( INT wait )
```

【リターン値】

```
>=  0  :  入力した文字コード
    -1  :  入力なし(wait == 0 のとき)
```

【説明】

デバッグ用コンソールから 1 文字(バイト)を入力します。入力した文字はエコーバックしません。
入力がないときは、wait == 0 のときは -1 を戻し、wait != 0 のときは入力されるまで待ちます。

Put Character

コンソールへ 1 文字出力

【C ライブラリ関数呼び出し形式】

INT tm_putchar(INT c)

c ： 出力する文字コード

【リターン値】

-1 ： Ctrl-C が入力された
0 ： Ctrl-C は入力されていない

【説明】

デバッグ用コンソールへ 1 文字(バイト)を出力します。
出力中に、Ctrl-C (0x03) が入力された場合、出力を中断して、-1 を戻します。
出力する文字が LF コード(0x0A)のときは、CR コード(0x0d) と LF コード(0x0A)の 2 文字を出力します。

Get Line

コンソールから 1 行入力

【C ライブラリ関数呼び出し形式】

INT tm_getline(UB *buff)

buff ： 入力文字列を格納するメモリの先頭アドレス

【リターン値】

>= 0 ： 入力した文字数
-1 ： Ctrl-C が入力された

【説明】

デバッグ用コンソールから改行(0x0d)、または Ctrl-C(0x03) が入力されるまでの 1行を入力して、指定されたメモリアドレスに格納します。
文字列の最後には NULLコード(0) を格納します。改行、および Ctrl-C は格納されません。
buff には十分な領域を用意しておく必要があります。あふれには関知しません。
入力した文字はエコーバックするとともに、以下の特殊キーの処理を行います。

Ctrl-X (0x18), Ctrl-U (0x15)	入力行の取り消し
Ctrl-H (0x08), DEL (0x7f)	入力文字の 1 文字の取り消し
Ctrl-F (0x06), ESC [C	カーソルを右に移動(→)
Ctrl-B (0x02), ESC [D	カーソルを左に移動(←)
Ctrl-P (0x10), ESC [A	前の入力行の呼び出し(↑)
Ctrl-N (0x0e), ESC [B	次の入力行の呼び出し(↓)
Ctrl-K (0x0b)	カーソル以降を削除

Put String

コンソールへ文字列出力

【C ライブラリ関数呼び出し形式】

INT tm_putstring(UB *buff)

buff ： 出力文字列を格納しているメモリの先頭アドレス

【リターン値】

-1 ： Ctrl-C が入力された
0 ： Ctrl-C は入力されていない

【説明】

指定されたメモリアドレスから 1 文字(バイト)ずつ NULLコード(0) までの文字をデバッグ用コンソールに出力します。

出力中に、 Ctrl-C (0x03) が入力された場合、出力を中断して、-1 を戻します。

文字列内の LF コード(0x0A)に対しては、CR コード(0x0d) と LF コード(0x0A)の 2 文字を出力します。

Execute Command

モニタコマンドの実行

【C ライブラリ関数呼び出し形式】

INT tm_command(UB *buff)

buff ： モニタコマンド文字列を格納しているメモリの先頭アドレス

【リターン値】

0 ： モニタコマンドを実行した
リターンしない ： モニタへ移行した

【説明】

指定されたメモリアドレスに格納されている文字列(NULLコード(0)で終了)をモニタのコマンド(列)として実行後、プログラムへ戻ります。
文字列が空の場合は、モニタへ移行し、プログラムへは戻りません。

Read Disk

ディスクからの読み込み

【C ライブラリ関数呼び出し形式】

```
INT tm_readdisk( UB *dev, INT sblk, INT nblks, VP addr )
```

dev　：　デバイス名を格納しているメモリの先頭アドレス
sblk　：　開始ブロック番号
nblks　：　ブロック数
addr　：　メモリアドレス

【リターン値】

0　：　正常終了
< 0　：　エラーコード

E_NOEXS	デバイスが存在しない
E_NOMDA	メディアがない
E_IO	入出力エラー
E_PAR	パラメータ不正
E_MACV	メモリーにアクセスできない

【説明】

デバイス名で指定したデバイスの開始ブロック番号からブロック数分だけ、メモリアドレスにディスクから読み込みます。

ブロックサイズは、デバイスやメディアごとに規定されます。

（例）pca　PCカード(ATA/CF) #1
pcb　PCカード(ATA/CF) #2
※ 具体的なデバイス名は実装依存となります。

Write Disk

ディスクへの書き込み

【C ライブラリ関数呼び出し形式】

```
INT tm_writedisk( UB *dev, INT sblk, INT nblks, VP addr )
```

dev　：　デバイス名を格納しているメモリの先頭アドレス
sblk　：　開始ブロック番号
nblks　：　ブロック数
addr　：　メモリアドレス

【リターン値】

0　：　正常終了
< 0　：　エラーコード

E_NOEXS	デバイスが存在しない
E_NOMDA	メディアがない
E_IO	入出力エラー
E_PAR	パラメータ不正
E_MACV	メモリーにアクセスできない
E_RONLY	書き込み禁止

【説明】

メモリアドレスの内容を、デバイス名で指定したデバイスの開始ブロック番号からブロック数分だけディスクに書き込みます。
ブロックサイズは、デバイスやメディアごとに規定されます。

（例）pca　PCカード(ATA/CF) #1
pcb　PCカード(ATA/CF) #2
※ 具体的なデバイス名は実装依存となります。

Info Disk

ディス情報の取り出し

【C ライブラリ関数呼び出し形式】

INT tm_infodisk(UB *dev, INT *blksz, INT *nblks)

dev : デバイス名を格納しているメモリの先頭アドレス
blksz : ブロックサイズ(バイト数)を格納するメモリの先頭アドレス
nblks : 全体のブロック数を格納するメモリの先頭アドレス

【リターン値】

0 : 正常終了
< 0 : エラーコード

E_NOEXS	デバイスが存在しない
E_NOMDA	メディアがない
E_IO	入出力エラー
E_MACV	メモリーにアクセスできない

【説明】

デバイス名で指定したデバイスのブロックサイズ(バイト数)、および全体のブロック数を取り出します。

※ 具体的なデバイス名は実装依存となります。

System Exit

システム終了

【C ライブラリ関数呼び出し形式】

```
void tm_exit( INT mode )
```

mode : 0 : システムを終了し電源を切ります。
-1 : システムをリセットして再起動します。

【リターン値】

リターンしない

【説明】

システムを終了し、電源オフまたはリセットします。

Extension SVC

拡張サービス機能

【C ライブラリ関数呼び出し形式】

```
INT tm_extsvc( INT fno, INT p1, INT p2, INT p3 )
```

fno : 拡張サービス機能番号
p1, p2, p3 : パラメータ1 ～ 3

【リターン値】

0 : 正常終了
< 0 : エラーコード

【説明】

拡張サービス機能番号で指定した拡張サービス機能を実行します。
拡張サービス機能番号、パラメータ、リターン値はすべて、モニタの実装依存となります。

5. ブートの詳細

5.1 ブート処理の概要

システムのブートは、通常、以下に示す手順で行います。

(1) ブートデバイスの検索
(2) 一次ブートプログラムのロード
(3) 二次ブートプログラムのロード
(4) オペレーティングシステムのロード

モニタでは、この内の (1), (2) の処理を行います。
また、一次ブートプログラムや二次ブートプログラムがディスクへアクセスするための、モニタサービス関数を提供します。

5.2 ブートデバイスの検索

ブートデバイスの検索順序は、通常、以下に示す順序で行われますが、具体的には実装に依存します。

(1) メディアが取り外せるデバイス　(フロッピーディスクや CD-ROM)
(2) ドライブが取り外せるデバイス　(PC カードや USB 接続のドライブ)
(3) 取り外せないデバイス　(内蔵ディスク)

区画(パーティション)を持つディスクでは区画情報を参照し、起動区画となっている区画のみを検索対象とします。区画情報の詳細については、PC での標準仕様にしたがいます。

5.3 一次ブートプログラムのロードと起動

ブートするディスクの先頭ブロック(区画を持つディスクでは区画の先頭ブロック)をメモリに読み込みます。これが、一次ブートプログラムとなります。
ブロックサイズが 512バイト未満の場合は、先頭ブロックから 512バイト以上となる分だけの連続したブロックを読み込みます。
メモリに読み込んだ一次ブートプログラムへ制御を移します。このとき、モニタは次の情報を一次ブートプログラムへ渡します。

```
#define L_DEVNM         8       /* デバイス名の長さ */

typedef struct BootInfo {
        UB      devnm[L_DEVNM]; /* ブートするディスクの物理デバイス名 */
        INT     part;           /* ブートする区画の番号(0～, -1:区画なし) */
        INT     start;          /* ブートする区画の位置(先頭ブロック番号) */
        INT     blksz;          /* ブートするディスクのブロックサイズ(バイト) */
} BootInfo;
```

一次ブートプログラムをロードするアドレスや、一次ブートプログラムへのパラメータの渡し方などは実装ごとに規定されます。