

ROBOCUBE のための 自走アンセンブリ言語の開発

開発者： 吉岡 良雄
弘前大学工学部電子情報システム工学科
平成 16 年 5 月 23 日版
(バージョンアップセット含む)

注意

本資料の著作権は、上記吉岡にあります。
これらを利用される方は、下記メールアドレスにご一報ください。

slyoshi@si.hirosaki-u.ac.jp

robocube@watt.co.jp

まえがき

ここ十数年のマイクロプロセッサの機能的進歩は非常に急激であり、これを利用したロボット（制御システム）が数多く製作され、一部においては商品化されている。このようなロボットにおいては、多くのセンサからの入力信号や多くの制御信号がある。多くの入出力信号を一つのマイクロプロセッサで処理する場合、回路およびプログラムが非常に煩雑なものになってしまう。そこで、一つの機能（センサからの入力処理や制御処理など）しか持たない小さなマイクロプロセッサと、メインの処理プロセッサ（ホストコンピュータという）をネットワーク接続した機能分散制御システムがある。

一方、工業高校、専門学校や大学学部等（高学年用）では、将来における制御システム（ロボット）の有用性から、マイクロプロセッサを利用して、ステップモータ等を制御する実験テーマが学生実験として導入されようとしている。また同時に、アセンブリ言語によるプログラミングも行われる。しかしながら、現在のマイクロプロセッサでは非常に機能が高く、短時間で行われる学生実験用として扱うことができない。学生実験用として扱うためには、命令数が少なく、分かり易い命令形式である必要がある。

この中で、学生実験用として有用な機能分散処理型ロボット ROBOCUBE が、1999 年 6 月に（株）システムワット社から発売された。この ROBOCUBE は、一つのブロックに一つのプロセッサを搭載し、一つの機能のみを割り当てて、複数のブロックを積み木のように自由に組み立てることができる構成である。そして、これらのブロックを接続するネットワークは共通バス接続方式を採用している。このため、ハードウェアが煩雑にならない特徴をもっている。また、ブロック間、およびブロックとパソコン（ホストコンピュータ）との情報交換にはパケットが用いられているので、パケットの特徴をも学ぶことができる。しかしながら（株）システムワット社から提供されたソフトウェア開発ツールは、リモコンおよび自走プログラムを作成できるタイル言語によるプログラム開発ツール、およびリモコン用関数ライブラリである。前者の開発ツールは、高学年用の学生実験としては不十分である。また、後者は自走用プログラムの作成ができない。そこで、アセンブリ言語風のプログラミング言語を開発し、そのアセンブラ（コンパイラ）を作成した。

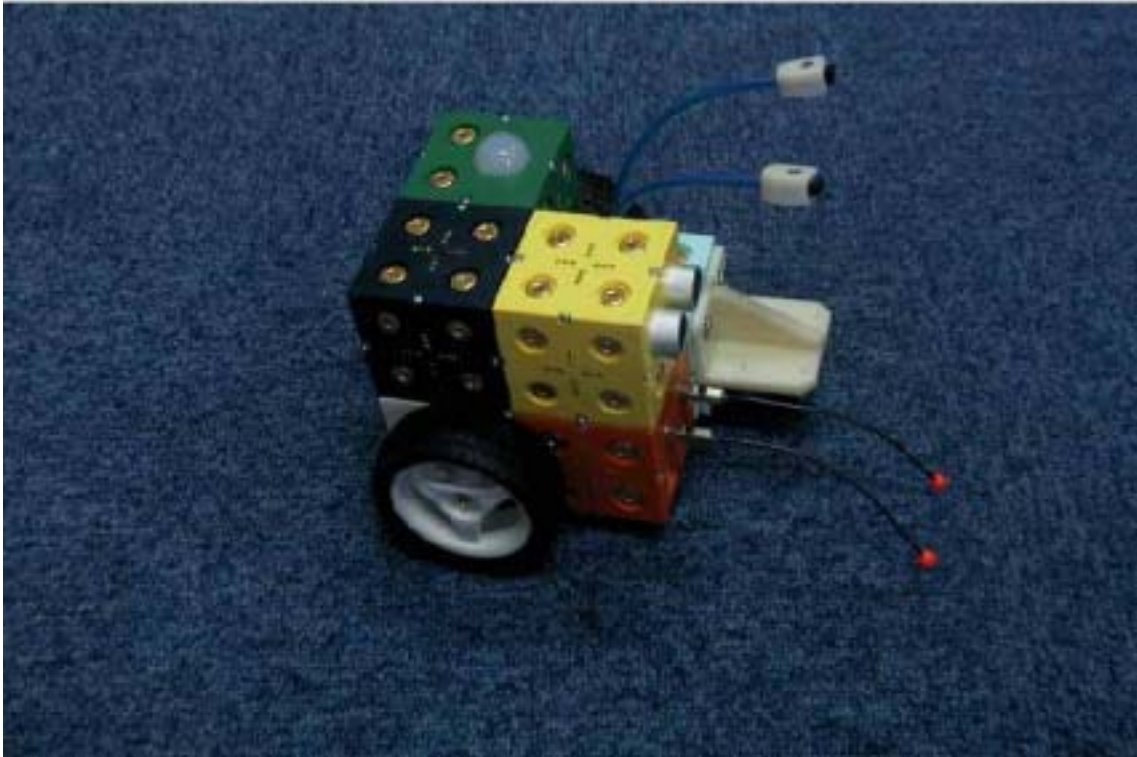
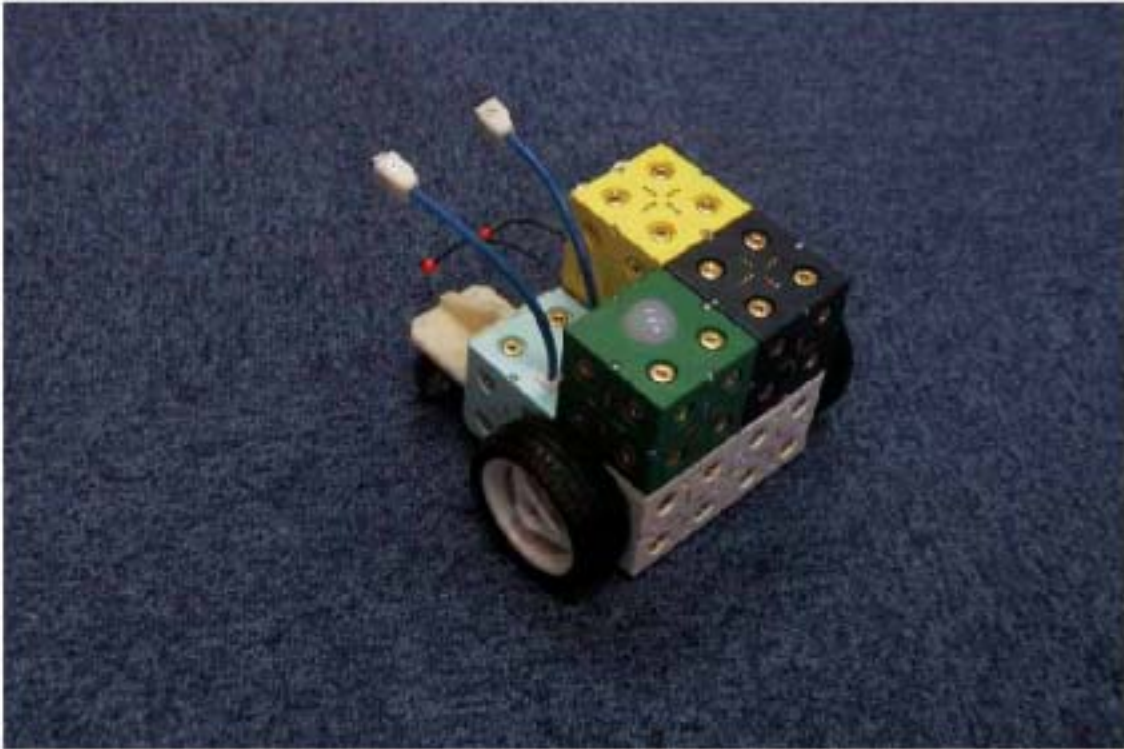
本書は、ROBOCUBE を自走で動作させるためのアセンブリ言語風プログラミング言語について述べたものである。本書で示した ROBOCUBE 用アセンブリ言語が学生実験用として利用されれば幸いである。なお、開発したアセンブラは別媒体で提供する。

2004 年 5 月 吉岡 良雄

目次

まえがき	2
1 はじめに	-1-
1.1 機能分散制御システムとは	-1-
1.2 共通バス接続方式	-2-
1.3 ROBOCUBE について	-4-
1.4 まとめ	-12-
2 ROBOCUBE 用アセンブリ言語	-13-
2.1 アセンブリ言語プログラム	-13-
2.2 アセンブラの設計	-14-
2.3 ROBOCUBE 用アセンブリ言語設計	-15-
2.4 まとめ	-32-
3 ROBOCUBE 用マイクロ命令	-33-
3.1 マクロ命令による命令設計法	-33-
3.2 マクロ命令によるプログラミング	-36-
3.3 まとめ	-46-
4 プログラム記述	-47-
4.1 サブルーティンのプログラミング	-47-
4.2 メインプログラム	-50-
4.3 プログラミングの注意点	-52-
4.4 アセンブラの実行	-54-
4.5 まとめ	-55-

Robo-Cube



Chapter 1

はじめに

マイクロプロセッサの機能が向上し、これを利用したロボット（制御システム）が数多く発表されている。このようなロボットにおいても、多くのセンサからの入力信号や多くの制御信号がある。多くの入出力信号を一つのマイクロプロセッサで処理する場合、回路およびプログラムが非常に煩雑なものになってしまう。そこで、一つの機能（センサからの入力処理や制御処理など）しか持たない小さなマイクロプロセッサと、メインの処理プロセッサ（ホストコンピュータという）をネットワーク接続した機能分散制御システムがある。

複数のプロセッサをネットワークで接続し、各プロセッサには 1 つの機能しか割り当てないようにしたシステムを 機能分散システム という。これをロボットなどの制御に利用すれば、機能分散制御システムとなる。ロボットなどでは、複数のセンサーからの入力データを処理し、複数の制御装置を制御することになる。機能分散制御システムでは、一般に 1 つのセンサーに 1 つのプロセッサを、および 1 つの制御装置に 1 つのプロセッサを割り当てる。本章では、機能分散制御システムのネットワーク接続方法を示し、本書で取り上げる ROBOCUBE の特徴と動作仕様などについて示す。

1.1 機能分散制御システムとは

一般に、機能分散制御システムは、図 1.1 に示すように、複数のプロセッサをネットワーク状に接続し、1 つのプロセッサには 1 つの機能しか割り当てない方法である。各プロセッサの通信ポートが 1 つの場合のネットワーク接続には、共通バス接続方式および単方向性ループ状接続方式がある。この 2 種類は単純な接続方法であるため、マイクロプロセッサによるネットワーク接続方式に利用される。また、このようなネットワーク接続を行って、各プロセッサ間で情報交換を行う場合、あて先プロセッサ番号等を含むパケットで行う。以降の節において、ネットワーク接続方式の特徴と問題点、ROBOCUBE のネットワーク接

続方式や動作仕様などについて示していこう。

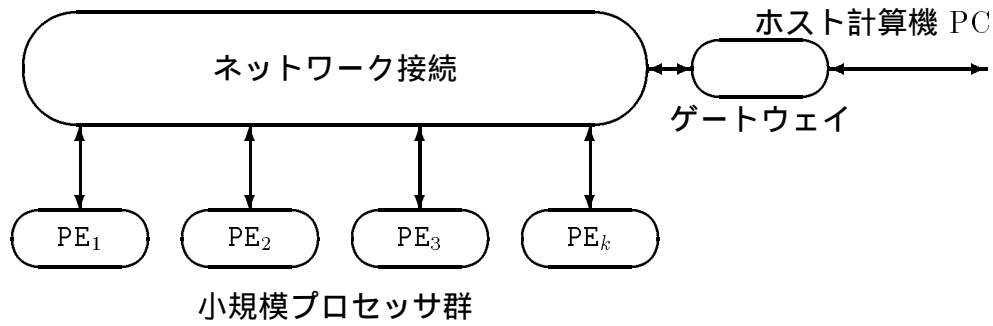


図 1.1 機能分散制御システムの一般的構成例

1.2 共通バス接続方式

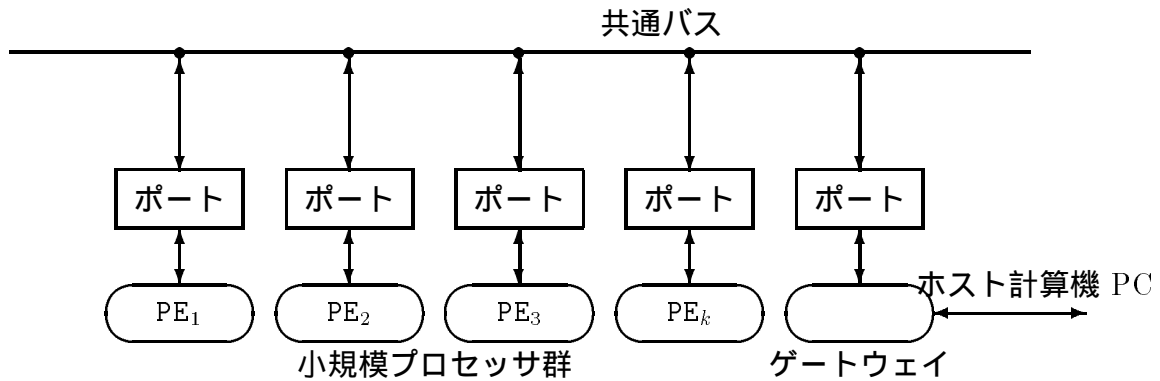


図 1.2 共通バス（イーサネット）接続方式

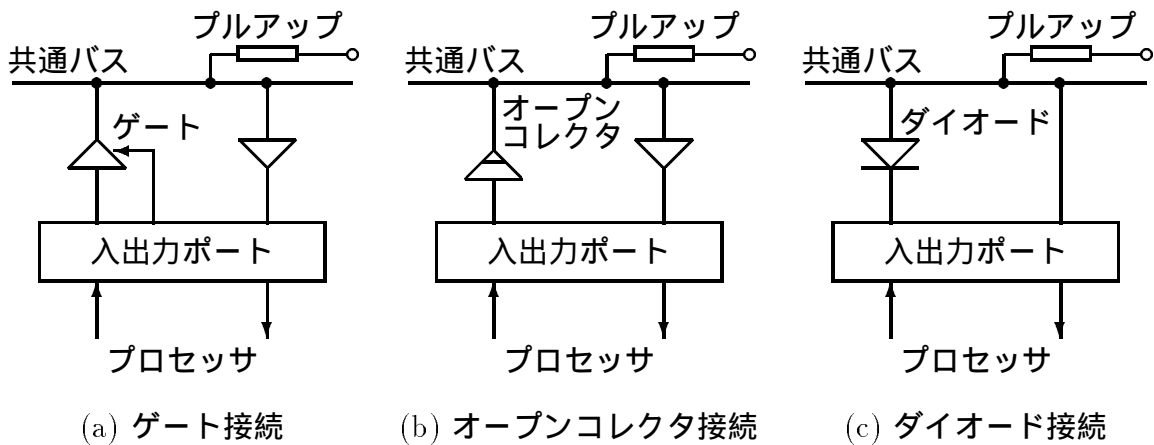


図 1.3 共通バスへの接続方法

共通バス接続方式は、図 1.2 に示すように、1本の導線（共通バス）に通信ポートを接続する方法である。ただし、各プロセッサの通信ポートから共通バスへの接続は、図 1.3 に

示すように，(a) 通信しないとき，出力ポートをハイインピーダンスにするか，または (b) オープンコレクタや (c) ダイオードで接続し，共通バスをプルアップする。なお，RS232C 用シリアルポートは，負論理の入出力であるため，(a) (b) (c) を利用して接続する方法をとる。

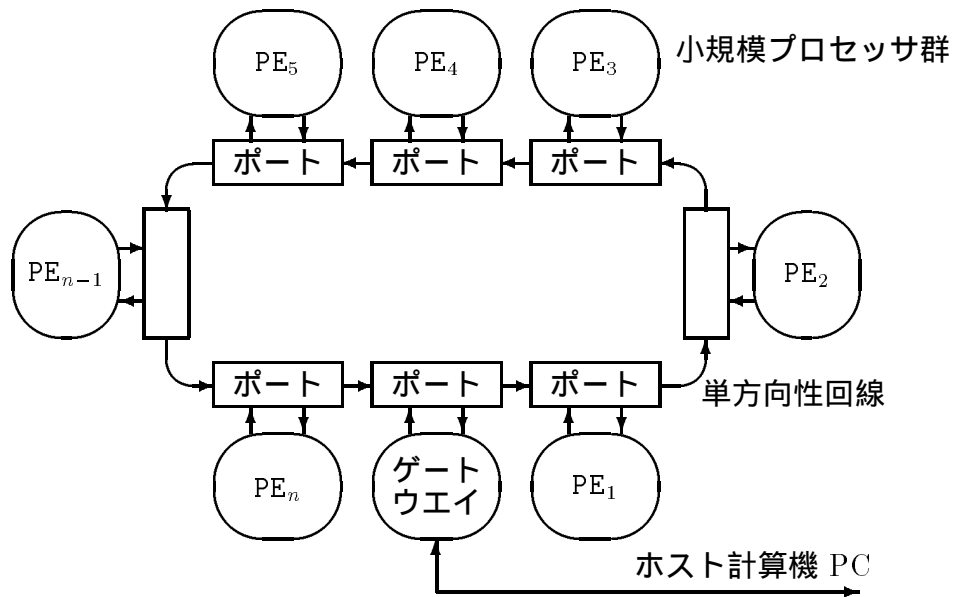


図 1.4 単方向性ループ状接続方式

この接続方式はイーサネットとして実用化されているが，各プロセッサからのパケット送信を自由に行うとパケット衝突が起こり，これを回避するプログラムは非常に複雑になる。また，衝突したパケットは再送することになり，高いスループットが得られない。すなわち，共通バス上において，再送パケットを含め，パケットの送信要求の発生がポアソン分布に従う（ランダム到着）ならば，回線上のトラヒック G とスループット S の関係は次式となる。

$$S = G \cdot e^{-2G}$$

このスループット S の最大値は， $G = 0.5$ のときであり， $S = \frac{1}{2e} \approx 0.183$ となる。すなわち，パケットの衝突が起こらないようにするためには，主プロセッサを置き，他のプロセッサに対してパケット送信権を割り当てるようにする。さらに，共通バスに複数の入出力ポートを接続するので，ファンナウトの問題があり，せいぜい 10 プロセッサしか接続することができない。

これに対して単方向性ループ状接続方式は，図 1.4 に示すように，あるプロセッサの出力ポートを次のプロセッサの入力ポートに接続し，ループ状に接続したものである。この方法

は、共通バス接続方式に見られるような送信パケットの衝突やファンナウトの問題がない。また、数十プロセッサ以上の接続が可能である。主プロセッサを置く必要がなく、複数のプロセッサによる分散処理・分散制御が可能である。従って、各プロセッサの通信プログラムが非常に簡単である。単方向性ループ状接続方式は、このように多くの利点を有するが、パケット転送時間が長くなるという欠点を有する。

1.3 ROBOCUBE について

ROBOCUBE は、(株)システムワット社が開発した機能分散処理型ロボットであり、1つのブロックに1つのプロセッサを搭載し、一つの機能のみを割り当てて、複数のブロックを積み木のように自由に組み立てるロボットである。そして、これらのブロックを接続するネットワークは、図 1.2 に示す共通バス接続方式を採用している。基本ブロックとしては、各ブロックを制御する制御ブロック(主ブロック)、車輪 1 を駆動する車輪ブロック 1、車輪 2 を駆動する車輪ブロック 2、ライトブロック、音響ブロック、タッチセンサーブロック、光センサーブロック、超音波センサーブロックの計 8 ブロックである。これらを制御するプログラムは、図 1.5 に示すパケットを主ブロックに書き込んで、主ブロックがこのパケット内容に従って、各ブロックを制御する方式である。

この ROBOCUBE のプログラムは、図 1.5 (自走プログラム)に示すように、まず最初にブロックを制御するサブルーティンが記述される。すなわち、ROBOCUBE のパケット形式は、図 1.6 に示すように、*1500 はヘッダーであり、mp 領域が (01)₁₆ である。ここで、no 領域はサブルーティンのアドレスを表し、最大 255 のプログラム領域しかもっていない。op は機能(操作命令)を(表 1.1 ~ 表 1.7 参照)、dt はパラメータを、data はタイマー値やブロック番号などのデータ値を、it はノード番号や第 2 パラメータ値を、jp は次に実行するアドレス no を、ck はチェックサムを表す。また、図 1.7 は、パソコンからリモコンのように制御するパケット形式である。ただし、図 1.7(d) のブロックからのパケットは、制御ブロックのデータ領域に格納されており、自走プログラムでの LoadAx 命令でレジスタに取り込みチェックすることができる。この受信パケットの code および data(R) の値、ビットパターンなどの意味は表 1.8 にまとめた。

mp	no	op	dt	data	it	jp	ck		* comments
*1500	01	00	00	00	0000	00	00	C7	subroutine
									* for wheel 0
*1500	01	01	07	06	0000	00	02	D7	wheel CW set
*1500	01	02	07	00	0000	FF	03	FF	mode set for master PE
*1500	01	03	07	00	0001	00	04	D6	wheel start
*1500	01	04	07	01	0032	00	05	DD	speed 50

```

*1500 01 05 08 00 0BB8 FF 06 32    time set 3000 for master PE
*1500 01 06 07 03 0000 00 07 DE    wheel stop
*1500 01 07 07 00 0000 FF 08 09    mode set for master PE
*1500 01 08 16 00 0000 00 00 D6    return from sub.
                                     * for wheel 1
*1500 01 09 07 00 0000 01 0A E9    mode set (stop)
*1500 01 0A 07 04 0000 01 0B F6    reset
*1500 01 0B 07 08 0000 01 0C FC    return original position
*1500 01 0C 07 01 001E 01 0D 0D    speed 30
*1500 01 0D 08 00 05DC FF 0E 50    time set 1500 for master PE
*1500 01 0E 07 09 0708 01 0F 12    position control
*1500 01 0F 16 00 0000 00 00 E4    return from sub.
                                     * for buzzer
*1500 01 10 07 00 0008 02 11 DB    mode set (music 3 mode)
*1500 01 11 07 01 000A 02 12 E7    volume 10
*1500 01 12 07 05 001E 02 13 F2    tempo 30
*1500 01 13 07 02 0001 02 14 DC    buzzer on
*1500 01 14 08 00 1388 FF 15 1A    time set 5000 for master PE
*1500 01 15 07 02 0000 02 16 DF    buzzer off
*1500 01 16 16 00 0000 00 00 D5    return from sub.
                                     * for light
*1500 01 17 07 00 0000 03 18 E2    light off
*1500 01 18 07 05 0000 03 19 E9    R-level set 0
*1500 01 19 07 06 0064 03 1A FD    G-level set 100
*1500 01 1A 07 07 00FF 03 1B 29    B-level set 255
*1500 01 1B 07 02 0003 03 1C FD    action DIM-on
*1500 01 1C 07 02 0001 03 1D FD    action light on
*1500 01 1D 08 00 1388 FF 1E 3A    time set 5000 for master PE
*1500 01 1E 07 02 0000 03 1F 00    action light off
*1500 01 1F 16 00 0000 00 00 E5    return from sub.
                                     * main program
*1500 00 00 00 00 0000 00 00 C6    mainroutine
*1500 00 01 06 00 0000 00 02 CF    start
*1500 00 02 00 01 0000 00 03 CC    CallSub wheel 0 (no=01)
*1500 00 03 00 09 0000 00 04 D6    CallSub wheel 1 (no=09)
*1500 00 04 00 10 0000 00 05 D0    CallSub buzzer (no=10)
*1500 00 05 00 17 0000 00 00 D3    CallSub light (no=17) : Stop
                                     * execution
*1500 02 00 19 00 0000 00 00 D2    Execution

```

図 1.5 ROBOCUBE への自走プログラムのパケット形式

次に, mp 領域が (00)₁₆ の部分はメインルーティンであり, この部分は, jp 領域および no 領域に従って順次実行する方法を取っている。メインルーティンでは, サブルーティンコール命令 CallSub, サブルーティンからの値をチェックして分岐する命令 CallSubAndTest, 無条件ジャンプ命令 Jump, 停止命令 Stop などがある。なお, mp 領域および no 領域がそ

それぞれ $(02)_{16}$ および $(00)_{16}$ の場合，パソコンから ROBOCUBE に対して，リモコンのように制御するパケットである。ここで，これらの命令や機能については，表 1.1 ~ 表 1.8 にまとめた。

*1500	mp=01	no	op	dt	data	it	jp	ck	$(1A)_{16}$
-------	-------	----	----	----	------	----	----	----	-------------

(a) 送信パケット形式 (subroutine mapping)

*1500	mp=00	no	op	dt	data	it	jp	ck	$(1A)_{16}$
-------	-------	----	----	----	------	----	----	----	-------------

(b) 送信パケット形式 (mainroutine mapping)

*1500	mp=02	04	op	node	block ID	00 00	ck	$(1A)_{16}$
-------	-------	----	----	------	----------	-------	----	-------------

(c) 送信パケット形式 (binding)

*1500	mp=02	00	op	dt	data	it	jp	ck	$(1A)_{16}$
-------	-------	----	----	----	------	----	----	----	-------------

(d) 送信パケット形式 (command)

*0D00	D1	02	mp	no	ck	$(1A)_{16}$
-------	----	----	----	----	----	-------------

(e) 受信パケット形式 (ACK)

図 1.6 ROBOCUBE の自走式パケット形式

*1500	mp=02	02	op	dt	param	node	00	ck	$(1A)_{16}$
-------	-------	----	----	----	-------	------	----	----	-------------

(a) 送信パケット形式 (PC からブロックへ)

*1500	mp=02	00	20	00	mask	00 00	ck	$(1A)_{16}$
-------	-------	----	----	----	------	-------	----	-------------

(b) 送信パケット形式 (送信マスク設定)

*1500	mp=02	03	00	00	data	00 00	ck	$(1A)_{16}$
-------	-------	----	----	----	------	-------	----	-------------

(c) 送信パケット形式 (ストローブ)

*0D00	node	code	data(R)	ck	$(1A)_{16}$
-------	------	------	---------	----	-------------

(d) 受信パケット形式 (ブロックから PC へ)

図 1.7 ROBOCUBE の PC 制御用パケット形式

表 1.1 ROBOCUBE の命令・機能 (1/4)

op	dt	data	int	jmp	機能
00	00	0000	00	00	SEQ_TILE (Program Start)
00	sub adr	nvbit	jp2	jp1	SEQ_TILE (Call Sub)
01	sub adr	0000	jp2	jp1	TEST_TILE (Call Sub and Test)
02	00	msec	00	jp1	SEQ_TIMERM (Wait)
03	jp3	0000	jp2	jp1	SEQ_RANDOM (Random Jump)
04	0~9	value	00	jp1	SET_COUNT (Counter Set)
05	0~9	0000	00	jp1	SEQ_COUNT (Counter Judge, Dec Counter)
06	00	00	00	jp1	NOP_TILE (Jump)
06	00	00	00	00	Execution Stop (Stop)
07	cmd	data	node	jp1	Set Mode, etc.
08	00	msec	00	jp1	SEQ_TIMERS (Wait Timer)
09	code	0000	node	jp1	WAIT_STATUS (Wait State)
0A	00	value	00	jp1	SEQ_LOAD (LoadAx value)
0A	01	node	00	jp1	SEQ_LOAD (LoadAx Analog data fram node)
0A	02	node	00	jp1	SEQ_LOAD (LoadAx Digital data fram node)
0A	03	node	00	jp1	SEQ_LOAD (LoadAx Status fram node)
0B	01, 02	value	node	jp1	SEQ_SAVE (SaveAx), 01:analog, 02:digital
0C	00	0000	00	jp1	SEQ_POP (PopAx, SP++, SP → AX)
0D	00	0000	00	jp1	SEQ_PUSH (PushAx, AX → SP, SP--)
0E	00	0000	00	jp1	SEQ_CPL (ComplimentAx)
0F	0~15	0000	jp2	jp1	SEQ_TEST (TestAx, bit test)
10	00	value	jp2	jp1	SEQ_AND (AndAx value)
10	01, 02	node	jp2	jp1	SEQ_AND (AndAx node), 01:analog, 02:digital
11	00	value	jp2	jp1	SEQ_OR (OrAx value)
11	01, 02	node	jp2	jp1	SEQ_OR (OrAx node), 01:analog, 02:digital
12	00	value	jp2	jp1	SEQ_CMP (CompareAx value)
12	01, 02	node	jp2	jp1	SEQ_CMP (CompareAx node), 01:analog, 02:digital
13	00	value	jp2	jp1	SEQ_LARGE (LargeAx value)
13	01, 02	node	jp2	jp1	SEQ_LARGE (LargeAx node), 01:analog, 02:digital)
14	00	value	jp2	jp1	SEQ_SMALL (SmallAx value)
14	01, 02	node	jp2	jp1	SEQ_SMALL (SmallAx node), 01:analog, 02:digital
15	00	0000	00	jp1	SEQ_UPDATE_TEST
16	00	0000	00	00	Return
25	00	0000	00	jp1	Pop Two Data (SP++, SP → BX, SP++ → AX)
26	00	0000	jp2	jp1	SEQ_NOT (NotAx)
27	00	0000	jp2	jp1	SEQ_NE (Not Equal)
28	00	0000	00	jp1	SEQ_START
29	00	0000	00	jp1	SEQ_END
2A	00	0000	00	jp1	GET_DIGL_SW
2B	00	0000	00	jp1	SEQ_MARKER

表 1.2 ROBOCUBE の命令・機能 (2/4)

op	dt	機能
17	00	RESET_SEQ
18	00	STOP_SEQ
19	00	RUN_SEQ (Execution Start)
1A	00	GO_SEQ
1B	00	STEP_SEQ
1C	00	SET_START_SEQ
1D	00	SET_BREAK_SEQ
1E	00	RESET_BREAK_SEQ
1F	00	CLR_BREAK_SEQ
20	00	SET_REDIRECT
21	00	SET_WATCH_SEQ
22	00	RESET_WATCH_SEQ
23	00	SET_RAM_SEQ
24	00	RESET_RAM_SEQ

表 1.3 ROBOCUBE の命令・機能 (3/4)

op	dt	data	機能
07	00	0~7	SET_OFF (リレー, インタフェース)
07	00	RC_Param	SET_MODE (モータ, ライト, ブザー, 超音波センサー等)
07	01	0~7	SET_ON (リレー, インタフェース)
07	01	value	SET_LEVEL (モータ 0~100, ブザー 0~255)
07	01	RC_Param	SET_LEVEL (超音波センサー)
07	02	RC_Param	ACTION (On or Off)
07	03	0000	MOTOR STOP
07	03	0~7	INVERT (リレー, インタフェース)
07	04	0000	MOTOR RESET
07	05	value	AD_DL1 (インタフェース 0~4095)
07	05	value	SET_DATA (ブザー 8~250)
07	05	value	SET_RED (ライト赤 0~255)
07	05	value	SET_ALARM (左右赤外線センサー 0~255)
07	06	0000	MOVE_CW
07	06	value	SET_ALARM1 (超音波センサー 0~3000, 左赤外線センサー 0~255)
07	06	value	AD_DC1 (インタフェース 0~4095)
07	06	value	SET_GREEN (ライト緑 0~255)
07	07	0000	MOVE_CCW
07	07	value	SET_ALARM2 (超音波センサー 0~3000, 右赤外線センサー 0~255)
07	07	value	AD_UC1 (インタフェース 0~4095)
07	07	value	SET_BLUE (ライト青 0~255)
07	08	0000	MOVE_HOME
07	08	value	AD_UL1 (インタフェース 0~4095)
07	09	0000	AD_OFF1 (インタフェース)
07	09	value	MOVE_POS (モータ -30000~30000)
07	0A	0000	AD_ON1 (インタフェース)
07	0A	value	SET_LIMP (モータ -30000~30000)

表 1.4 ROBOCUBE の命令・機能 (4/4)

op	dt	data	機能
07	0B	0000	AD_REQ1 (インタフェース)
07	0B	value	SET_LIMM (モータ -30000 ~ 30000)
07	0C	0000	SET_OFF1 (インタフェース)
07	0C	value	SET_PASSP (モータ -30000 ~ 30000)
07	0D	0000	SET_ON1 (インタフェース)
07	0D	value	SET_PASSM (モータ -30000 ~ 30000)
07	0E	0000	REQ_POS (モータ, 超音波センサー)
07	0E	0000	REQ_STATE1 (インタフェース)
07	0F	0000	REQ_POSR (モータ)
07	0F	value	AD_DL2 (インタフェース 0 ~ 4095)
07	10	0000	REQ_STATE (インタフェース以外)
07	10	value	AD_DC2 (インタフェース 0 ~ 4095)
07	11	value	AD_UC2 (インタフェース 0 ~ 4095)
07	12	value	AD_UL2 (インタフェース 0 ~ 4095)
07	13	0000	AD_OFF2 (インタフェース)
07	14	0000	AD_ON2 (インタフェース)
07	15	0000	AD_REQ2 (インタフェース)
07	16	0000	ACTION1 (インタフェース)
07	17	0000	ACTION2 (インタフェース)
07	18	0000	ACTION3 (インタフェース)
07	19	value	AD_DL3 (インタフェース 0 ~ 4095)
07	1A	value	AD_DC3 (インタフェース 0 ~ 4095)
07	1B	value	AD_UC3 (インタフェース 0 ~ 4095)
07	1C	value	AD_UL3 (インタフェース 0 ~ 4095)
07	1D	0000	AD_OFF3 (インタフェース)
07	1E	0000	AD_ON3 (インタフェース)
07	1F	0000	AD_REQ3 (インタフェース)
07	20	0000	ACTION4 (インタフェース)
07	23	value	AD_DL4 (インタフェース 0 ~ 4095)
07	24	value	AD_DC4 (インタフェース 0 ~ 4095)
07	35	value	AD_UC4 (インタフェース 0 ~ 4095)
07	36	value	AD_UL4 (インタフェース 0 ~ 4095)
07	37	0000	AD_OFF4 (インタフェース)
07	38	0000	AD_ON4 (インタフェース)
07	39	0000	AD_REQ4 (インタフェース)

表 1.5 RC_Param (ACTION)

引数	値	意味
OFF	00	ブザーおよびライト停止
ON	01	ブザーおよびライト開始
DIM_OFF	02	ライトの DIM OFF
DIM_ON	03	ライトの DIM ON
ONE.SHOT	02	ブザーのワンショット

表 1.6 RC_Param (SET_MODE)

引数	値	意味
POS_MODE	00	車軸の位置制御モード
VEL_MODE	01	車軸の速度制御モード
INACTIVE	00	超音波センサー停止モード
DIS_MODE	01	超音波センサーの距離感知モード
ALM_MODE	02	超音波センサーのアラームモード
DIR_MODE	00	赤外線センサーのダイレクトモード
REF_MODE	01	赤外線センサーの反射モード
CONTINUOUS	00	ライトおよびブザーの連続モード
MODE0	00	ライトおよびブザーのあらゆる点滅モード
FLASH1	01	ライトおよびブザーの速い点滅モード
FLASH2	02	ライトおよびブザーの間欠点滅モード
FLASH3	03	ブザーのサイレンモード
MODE5	05	ブザーのピーポーモード
MODE6	06	ブザーの音楽1モード
MODE7	07	ブザーの音楽2モード
MODE8	08	ブザーの音楽3モード

表 1.7 RC_Param (SET_LEVEL)

引数	値	意味
SMALL	00	超音波センサーの感度小モード
LARGE	01	超音波センサーの感度大モード
MEDIUM	02	超音波センサーの感度中モード

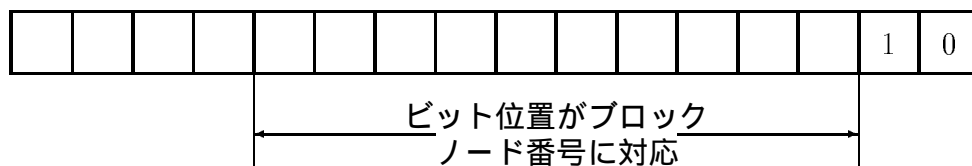


図 1.8 nvbit のビット構成

システムワット(株)社から提供されたソフトウェア開発ツールは、リモコンおよび自走プログラムを作成できるタイル言語によるプログラム開発ツール、およびリモコン用関数ライブラリ(C言語)である。前者の開発ツールは初心者用であって、大きなプログラムや複雑なプログラムを作成すると煩雑なものになってしまう。また、後者の関数ライブラリでは自走用プログラムの作成ができないこと、サブルーティンコールのアドレス設定が困難であること、などの問題がある。従って、大学学部生がこれらのソフトウェア開発ツールを用いてプログラム開発を行っても、複雑な動きをするプログラムを期待することができなかった。このため、アセンブリ言語風のプログラミング言語の開発を行った。

表 1.8 ブロックからの受信データ RC_Code

CODE	記号	data(R)	意味	対象ブロック
00	POS	値 -30000 ~ 3000	位置	車軸
01	EVENT	ビットパターン 0 ビット 1 ビット 2 ビット 3 ビット 4 ビット 5 ビット 6 ビット	アラーム状態 リミット+ リミット- PASS + PASS - 原点復帰完了 プリロード完了 インポジション	車軸
0D	TOUCH	ビットパターン 0 ビット 1 ビット 2 ビット 3 ビット 4 ビット 5 ビット 6 ビット 7 ビット	アラーム状態 触覚 1 の右 触覚 1 の上 触覚 1 の左 触覚 1 の下 触覚 2 の右 触覚 2 の上 触覚 2 の左 触覚 2 の下	タッチセンサー
0E	DIST	値 0 ~ 3000	距離	超音波センサー
0F	ALARM	ビットパターン 0 ビット 1 ビット	アラーム状態 アラーム 1 アラーム 2	超音波センサー
10	SENSE	ビットパターン 0 ビット 1 ビット	アラーム状態 アラーム 1 アラーム 2	赤外線センサー
11	X	値 -32000 ~ 32000	X の値	リモコン
12	Y	値 -32000 ~ 32000	Y の値	リモコン
13	CODE	ビットパターン 0 ビット 1 ビット ~ N ビット	ボタン状態 ボタン 1 ボタン 2 ~ ボタン N	リモコン
15	DI_STATE	ビットパターン	状態	インタフェース
16	AD_POS0	値 0 ~ 4095	CH-1 の状態	インタフェース
17	AD_POS1	値 0 ~ 4095	CH-2 の状態	インタフェース
18	AD_POS2	値 0 ~ 4095	CH-3 の状態	インタフェース
19	AD_POS3	値 0 ~ 4095	CH-4 の状態	インタフェース
1A	AD_ALARM	ビットパターン	アラーム情報	インタフェース

1.4 まとめ

本章では、複数の小規模プロセッサを利用した機能分散制御システムの一般的構成法を示し、(株)システムワット社から販売された ROBOCUBE の構成法、パケット形式とその機能仕様、および提供されたソフトウェア開発ツールの現状について述べた。いわゆる、提供された開発ツールだけでは、大学学部等の高学年教育用として不十分であることが分かった。そこで、本書では、自走する ROBOCUBE のためのプログラム開発を容易に行えるように、アセンブリ言語風のプログラミング言語を開発し、そのコンパイラ(アセンブラ)を作成したものについて、詳細に述べるものである。また、このようなシステムのアセンブラ等の開発が可能になるように、アセンブラの開発方法も述べることにする。

Chapter 2

ROBOCUBE 用アセンブリ言語

0 と 1 ,あるいは 16 進数で表される機械語命令 (ROBOCUBE ではパケット) では, プログラミングが困難である。そこで, この機械語命令と 1 対 1 対応の文字列で表現するプログラミング技法を アセンブリ言語 という。このアセンブリ言語から機械語命令に変換するプログラムを アセンブラまたは コンパイラ という。この章では, アセンブリ言語のプログラミング法, アセンブラの設計・製作法, ROBOCUBE のためのアセンブリ言語の設計およびアセンブラの設計・製作を示す。

2.1 アセンブリ言語プログラム

まず, 図 2.1 に示すように機械語命令と 1 対 1 対応の文字列で表現するプログラミング技法をアセンブリ言語という。アセンブリ言語プログラムの 1 行は, 一般に図 2.2 のように記述する。ここで, ラベル部 は複数の文字列 (最初の文字は英字) で記述し, 省略可能である (図 2.1 参照)。この部分に書かれるラベルには, そのアドレス位置が対応付けられるので, 同じラベル名があってはならない。命令部 はラベル部から 1 文字以上の空白 (または, タブ) をおき, 機械語命令に対応した文字列が記述される。なるべく機械語命令の動作を表す文字列が利用される。そして, 1 個以上の空白 (または, タブ) をおいて, オペランド部 がある。このオペランド部には数値またはラベルが書かれ, 複数のオペランドを必要とする場合はカンマで区切られる。ここに記述するラベルは, ラベル部に対応する文字列でなければならない。そして, そのラベルのもつアドレス値 (または, 数値) がこの機械語命令の オペランド値 となる。このオペランド部から 1 個以上の空白 (または, タブ) をおいて記述される文字列はコメント文となる。従って, この記述法の 1 行は, ラベル部, 機械語命令に対応した命令部, オペランド部, コメント部に分かれ, これらは 1 個以上の空白 (または, タブ) で区切られる。

番地	機械語	アセンブリ言語プログラム
1	0000	; casl program
2	0000	; factrial
3	0000	start #1000
4	1000 1010 100a	ld gr1,n
5	1002 8000 100c	call fact
6	1004 1110 100b	st gr1,ans
7	1006 8000 1040	call print
8	1008 ff00 0000	halt
9	100a 0005	n dc 5
10	100b	ans ds 1
11	100c	;
12	100c	fact
13	100c 4110 1020	cpl gr1,h0001
14	100e 6200 1014	jnz fact1
15	1010 1210 0001	lea 1
16	1012 8100 0000	ret
17	1014	fact1
18	1014 7001 0000	push 0,gr1
19	1016 2110 1020	sub gr1,h0001
20	1018 8000 100c	call fact
21	101a 7120 0000	pop gr2
22	101c 8000 1021	call mul
23	101e 8100 0000	ret
24	1020 0001	h0001 dc 1
25	1021	; mul

図 2.1 CASL プログラム例(一部)

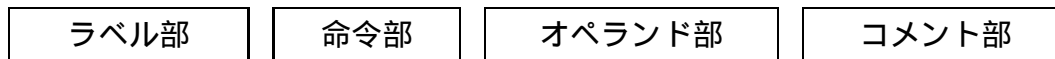


図 2.2 アセンブリ言語プログラムの記述法

このように機械語命令を人間が理解できる文字列で記述する方法をアセンブリ言語という。そして、これを機械語命令コードに変換するプログラムをアセンブラという。しかしながら、アセンブラに対して、機械語プログラムを格納する位置を指示すること (org)、メモリ領域を確保すること (ds)、定数値を設定すること (dc)、プログラムの終了を指示すること (end)、ラベルに値を定義すること (equ)などを指示する必要がある。この指示のことを疑似命令と呼び、図 2.2 に示す記述法と同じ方法を取る。

2.2 アセンブラの設計

アセンブラを設計・製作する場合、表 2.1 に示すラベルテーブルを作成する必要がある。アセンブラは一般的に 2 パスが用いられる。すなわち、オペランド部で初めてラベルが出

てきた場合、ラベルに値（またはアドレス）が定義されていないので、最初のパスでこのラベルテーブルを作成する。そして、ラベル部にそのラベルが現れると、その位置のアドレスをテーブルに格納する。このようにすると、次のパスではすべてのラベルに値（またはアドレス）が設定されるので（設定されていない場合は未定義ラベルということでエラーにする）、すべての機械語命令が定義されることになる。表 2.1 において、定義フラグが 1 のときラベルに値が定義された場合であり、0 のときラベルに値が定義されていない場合である。このようなアセンブラを設計・製作してみるのもよいプログラミング演習になる。

表 2.1 ラベルテーブル（22 行目のコンパイル時）

ラベル	定数值またはアドレス値	定義フラグ
n	(100a) ₁₆	1
ans	(100b) ₁₆	1
fact	(100c) ₁₆	1
fact1	(1014) ₁₆	1
h0001	(xxxx) ₁₆	0
mul	(xxxx) ₁₆	0
-----	-----	0
-----	-----	1

2.3 ROBOCUBE 用アセンブリ言語設計

ROBOCUBE は、第 1 章に示したように、パケットによって動作している。ここでは、このパケット生成のための命令・機能記述をアセンブリ言語風な記述法を考える。そこで、以下に表 1.1 ~ 表 1.8 に基づいてアセンブリ言語の記述法を詳細に説明する。

(1) Program Start

サブルーティンやメインルーティンの開始を表す。記述法は以下の通りである。

サブルーティン開始： Subroutine

メインルーティン開始： Mainroutine

この場合、生成されるパケット構成は以下のようになる。

サブルーティン開始： *1500 01 00 00 00 0000 00 00 ck _

メインルーティン開始： *1500 00 00 00 00 0000 00 00 ck _

ここで、ck はチェックサムである（以下同様）。

(2) Call Sub (or Call Sub Set Interrupt)

メインルーティンからサブルーティンへコールする場合に利用する命令である。記述法は以下の通りである。

```
CallSub label[,nodx,it,jp1]
```

```
CallSubSetInterrupt label[,nodx,it,jp1]
```

ここで、`jp1` はサブルーティンから戻って次に実行するアドレス `j1` である（以下同様）。また、`it` は割り込みが発生した場合の開始アドレスであり、`nodx` は図 1.8 に示すデータ構造である。さらに、`label` はサブルーティンの開始位置 `js` に記述されているラベルを表す。なお、記述法において、`[]` は省略可能を意味する（以下同様）。省略した場合、一般に `0` が入る。ただし、`jp1`（および `jp2`）においては、次に実行する命令のアドレスとなる（以下同様）。この場合、生成されるパケット構成は以下のようなになる。

```
*1500 00 no 00 js nodx it j1 ck _
```

ここで、`no` はメインルーティンまたはサブルーティンを格納するアドレスを表す（以下同様）。

(3) Call Sub And Test (or Interrupt)

メインルーティンからサブルーティンへコールし、サブルーティンからの結果をテストして分岐する命令である。また、`jp2` はテストの結果偽であるときの分岐アドレス `j2` である（以下同様）。この記述法は以下の通りである。

```
CallSubAndTest label[,jp2,jp1]
```

```
Interrupt label[,jp2,jp1]
```

ここで、`label` はサブルーティンの開始位置に記述されているラベルを表す。この場合、生成されるパケット構成は以下のようなになる。

```
*1500 00 no 01 js 0000 j2 j1 ck _
```

(4) Wait

待ち時間をとる命令である。記述法は以下の通りである。

```
Wait time[,jp1]
```

ここで、`time` は待ち時間（`m` 秒）を表す。この場合、生成されるパケット構成は以下のようなになる。

```
*1500 mp no 02 00 time 00 j1 ck _
```

ここで、`mp` は `00`（メインルーティン）、`01`（サブルーティン）または `02`（ダイレクトモード、`no=00`）を表す（以下同様）。

(5) Random Jump

分岐先 2 箇所または 3 箇所を乱数によってランダムに分岐する命令である。記述法は以下の通りである。

```
RandomJump [jp3],jp2[,jp1]
```

この場合、生成されるパケット構成は以下のようになる。

```
*1500 mp no 03 j3 0000 j2 j1 ck _
```

ここで、j3, j2 は分岐先である。

(6) Set Counter

10 個のカウンタレジスタの一つに値を格納する命令である。この記述法は以下の通りである。

```
SetCounter ct,data[,jp1]
```

ここで、ct はカウンタ番号であり、0~9 の値である。この場合、生成されるパケット構成は以下のようになる。

```
*1500 mp no 04 ct data 00 j1 ck _
```

(7) Dec Counter (Counter Judge)

指定されたカウンタレジスタ(0 ~ 10)の内容を1減じ、0であれば分岐する命令である。この記述法は以下の通りである。

```
DecCounter ct[,jp2,jp1]
```

ここで、ct はカウンタ番号であり、0~9 の値である。この場合、生成されるパケット構成は以下のようになる。

```
*1500 mp no 05 ct 0000 j2 j1 ck _
```

(8) Jump label

ジャンプ命令であり、記述法は以下の通りである。

```
Jump jp1
```

この場合、生成されるパケット構成は以下のようになる。

```
*1500 mp no 06 00 0000 00 j1 ck _
```

(9) Stop

プログラムの実行を停止する命令である。この記述法は以下の通りである。

```
Stop
```

この場合、生成されるパケット構成は以下のようになる。

```
*1500 mp no 06 00 0000 00 00 ck _
```

(10) Set Mode

各ブロックの動作を決定するための命令であり、以下のように記述する。

```
block IDL,dt,data[,jp1]
```

ここで、block は、Motor、Buzzer、Light、Ultrasonic、Optical、Touch、Gyro 等のように記述する。また、IDL はブロック番号 ID の下位であり、通常 0 である（以下同様）。ただし、バージョンアップセットのセンターブロックに内蔵されているブザーおよびライトの IDL には 10 がセットされている。dt には表 1.4 ~ 表 1.7 の機能に示す文字列（例えば、SET_OFF 等）を記述する。さらに、data 部には、表 1.3 および表 1.4 の機能に示すデータ値を記述する。この場合、生成されるパケット構成は以下ようになる。

```
*1500 mp no 07 dt data nd j1 ck _
```

ここで、nd は、IDL と block からバインド時に割り当てられたノード番号である（以下同様）。

(11) WaitTimer

待ち時間をとる命令である。記述法は以下の通りである。

```
WaitTimer time[,jp1]
```

この場合、生成されるパケット構成は以下ようになる。

```
*1500 mp no 08 00 time 00 j1 ck _
```

(12) WaitStatus

状態待ちの命令である。記述法は以下の通りである。

```
WaitStatus block,IDL,code[,jp1]
```

ここで、block は、前と同様、Motor、Buzzer、Light、Ultrasonic、Optical、Touch、Interface、Relay、Gyro 等のように記述する。この場合、生成されるパケット構成は以下ようになる。

```
*1500 mp no 09 cd 0000 nd j1 ck _
```

ここで、nd は、IDL（通常 0）と block からバインド時に割り当てられたノード番号である（以下同様）。

(13) Load Ax

Ax レジスタに値やブロックからのデータを取り込み、それをスタックに積む命令である。この場合、生成されるパケット構成は以下のように記述される。

値を格納する場合 LoadAx 0,data[,jp1]

ブロックから格納 LoadAx block,IDL,op3[,jp1]

ここで、op3 (o3) は 01（アナログデータ値）、02（デジタルデータ値）、03（ステータス値）の値をとる。この場合、生成されるパケット構成は以下ようになる。

値を格納する場合 *1500 mp no 0A 00 data 00 j1 ck _

ブロックから格納 *1500 mp no 0A o3 node 00 j1 ck _

ここで, node は, IDL (通常 0) と block からバインド時に割り当てられたノード番号である (以下同様)。

(14) Save Ax

データをブロックに格納する命令である。記述法は以下の通りである。

```
SaveAx block,IDL,op3,data[,jp1]
```

ここで, op3 (o3) は 01 (アナログデータ値), 02 (デジタルデータ値) の値をとる。この場合, 生成されるパケット構成は以下ようになる。

```
*1500 mp no 0B o3 data nd j1 ck _
```

(15) Pop One Data

スタックから Ax レジスタに値を取り出す命令である。記述法は以下の通りである。

```
PopOneData [jp1]
```

この場合, 生成されるパケット構成は以下ようになる。

```
*1500 mp no 0C 00 0000 00 j1 ck _
```

(16) Push One Data

Ax レジスタの内容をスタックに積む命令である。記述法は以下の通りである。

```
PushOneData [jp1]
```

この場合, 生成されるパケット構成は以下ようになる。

```
*1500 mp no 0D 00 0000 00 j1 ck _
```

(17) Compliment Ax

Ax レジスタの内容をビット反転する命令である。記述法は以下の通りである。

```
ComplimentAx [jp1]
```

この場合, 生成されるパケット構成は以下ようになる。

```
*1500 mp no 0E 00 0000 00 j1 ck _
```

(18) Test Ax

Ax レジスタの内容のビットテストし, その真(値 1)・偽(値 0)をスタックに積む命令である。また, 偽である場合 jp2 にジャンプする。この記述法は以下の通りである。

```
TestAx bit[,jp2,jp1]
```

ここで, bit (bt) は 0~15 の値であり, テストしたいビット位置を表す。この場合, 生成されるパケット構成は以下ようになる。

```
*1500 mp no 0F bt 0000 j2 j1 ck _
```

(19) And Ax

Ax レジスタの内容と、即値またはブロックからの値と論理積を取り、その真(値 1)・偽(値 0)をスタックに積む命令である。また、偽である場合 jp2 にジャンプする。この記述法は以下の通りである。

即値の場合 AndAx 0,data[,jp2,jp1]

ブロック値 AndAx block,IDL,op3[,jp2,jp1]

ここで、op3 (o3) は 01 (アナログデータ値)、02 (デジタルデータ値) の値をとる。この場合、生成されるパケット構成は以下ようになる。

即値の場合 *1500 mp no 10 00 data j2 j1 ck -

ブロック値 *1500 mp no 10 o3 node j2 j1 ck -

(20) Or Ax

Ax レジスタの内容と、即値またはブロックからの値と論理和を取り、その真(値 1)・偽(値 0)をスタックに積む命令である。また、偽である場合 jp2 にジャンプする。この記述法は以下の通りである。

即値の場合 OrAx 0,data[,jp2,jp1]

ブロック値 OrAx block,IDL,op3[,jp2,jp1]

ここで、op3 (o3) は 01 (アナログデータ値)、02 (デジタルデータ値) の値をとる。この場合、生成されるパケット構成は以下ようになる。

即値の場合 *1500 mp no 11 00 data j2 j1 ck -

ブロック値 *1500 mp no 11 o3 node j2 j1 ck -

(21) Compare Ax

Ax レジスタの内容と、即値またはブロックからの値と論理比較を行い、その真(等しい場合値 1)・偽(等しくない場合値 0)をスタックに積む命令である。また、偽である場合 jp2 にジャンプする。この記述法は以下の通りである。

即値の場合 CompareAx 0,data[,jp2,jp1]

ブロック値 CompareAx block,IDL,op3[,jp2,jp1]

ここで、op3 (o3) は 01 (アナログデータ値)、02 (デジタルデータ値) の値をとる。この場合、生成されるパケット構成は以下ようになる。

即値の場合 *1500 mp no 12 00 data j2 j1 ck -

ブロック値 *1500 mp no 12 o3 node j2 j1 ck -

(22) Large Ax

Ax レジスタの内容と、即値またはブロックからの値と比較を行い、その真(大きい場

合値 1)・偽(それ以外の場合値 0)をスタックに積む命令である。また、偽である場合 jp2 にジャンプする。この記述法は以下の通りである。

即値の場合 LargeAx 0,data[,jp2,jp1]

ブロック値 LargeAx block,IDL,op3[,jp2,jp1]

ここで、op3 (o3) は 01 (アナログデータ値)、02 (デジタルデータ値) の値をとる。この場合、生成されるパケット構成は以下のようになる。

即値の場合 *1500 mp no 13 00 data j2 j1 ck -

ブロック値 *1500 mp no 13 o3 node j2 j1 ck -

(23) Small Ax

Ax レジスタの内容と、即値またはブロックからの値と比較を行い、その真(小さい場合値 1)・偽(それ以外の場合値 0)をスタックに積む命令である。また、偽である場合 jp2 にジャンプする。この記述法は以下の通りである。

即値の場合 SmallAx 0,data[,jp2,jp1]

ブロック値 SmallAx block,IDL,op3[,jp2,jp1]

ここで、op3 (o3) は 01 (アナログデータ値)、02 (デジタルデータ値) の値をとる。この場合、生成されるパケット構成は以下のようになる。

即値の場合 *1500 mp no 14 00 data j2 j1 ck -

ブロック値 *1500 mp no 14 o3 node j2 j1 ck -

(24) Update Test

この命令の記述法は以下の通りである。

UpdateTest [jp1]

この場合、生成されるパケット構成は以下のようになる。

*1500 mp no 15 00 0000 00 j1 ck -

(25) Return

サブルーティンからの戻り(タイル処理の終わり)の命令である。この記述法は以下の通りである。

Return

この場合、生成されるパケット構成は以下のようになる。

*1500 mp no 16 00 0000 00 00 ck -

(26) Pop Two Data

スタックから Bx レジスタおよび Ax レジスタの順にロードする命令である。この記

述法は以下の通りである。

```
PopTwoData [jp1]
```

この場合、生成されるパケット構成は以下のようになる。

```
*1500 mp no 25 00 0000 00 j1 ck _
```

(27) Not Ax

Ax レジスタの内容を反転し、結果をスタックに積む命令である。この記述法は以下の通りである。

```
NotAx [jp2,jp1]
```

この場合、生成されるパケット構成は以下のようになる。

```
*1500 mp no 26 00 0000 j2 j1 ck _
```

(28) Not Equal

この記述法は以下の通りである。

```
NotEqual [jp2,jp1]
```

この場合、生成されるパケット構成は以下のようになる。

```
*1500 mp no 27 00 0000 j2 j1 ck _
```

(29) Send Strobe

モータなど同時に動作させる場合にコントロールブロックから一斉に各ブロックに対して起動パケットを送出する命令である。この命令は以下のように記述する。

```
SendStrobe [jp1]
```

この場合、生成されるパケット構成は以下のようになる。

```
*1500 mp no 07 00 0000 FF j1 ck _
```

(30) Exection

ROBOCUBE を自走で動作させるための命令であり、以下のように記述する。

```
BindBlock
```

この場合、生成されるパケット構成は以下のようになる。

```
*1500 02 00 19 00 0000 00 00 D2 _
```

(31) Define Block

擬似命令であり、使用するブロックを定義する命令である。この記述法は以下の通りである。

```
DefineBlock IDL,block
```

ここで、IDL はブロック ID の下位であり、通常 0 である。同じブロックが複数ある場

合に 0 以外の値を用いる。また, block は, Motor , Buzzer , Light , Ultrasonic , Optical , Touch , Gyro 等のように記述する。ただし, 提供する基本ブロックのアセンブラ cubeassem5.c (バージョン 2.05) およびバージョンアップセットのアセンブラ cubeassem31.c (バージョン 3.01) では予め基本ブロックを登録しているので必要がない。ブロックを削除する場合やブロックを追加する場合は, アセンブラ (C 言語) ソース (有償) を変更する必要がある。

(32) Bind Block

DefineBlock で定義されたブロックの ID とノード番号 node (0 から順にふられた番号) と対応を取る命令であり, ブロックの数分のパケットが生成される。この記述法は以下の通りである。

```
BindBlock
```

ただし, 提供するアセンブラでは予め基本ブロックを登録してあるので, 先の擬似命令 Define Block を行わないで直接この命令を実行すればよい (プログラム例を参照)。

(33) equ

ラベルと数値が等価である擬似命令である。この記述法は以下の通りである。

```
label equ 数値
```

(34) end

プログラムの終わりを表す擬似命令である。

以上をまとめると表 2.2 および表 2.3 のようになる。この記述法を用いた ROBOCUBE 動作プログラム例は以下のようなになる。このプログラムは, システムワット社から提供されたタイル言語によるプログラム開発ツールによって小さな自走プログラムを作成し, それから ROBOCUBE へ転送されるパケットを解析し, ROBOCUBE のハードウェアの動作を推定して記述したものである。このため, プログラム記述に無理がある可能性がある。このプログラムの詳細説明については, 第 4 章で行う。なお, 作成したアセンブラには, 表 1.3 ~ 表 1.7 のパラメータは予約語としてアセンブラに登録してあるので, 改めて equ 文で定義する必要はない。また, 表 2.3 にブロックから状態を取り込むマクロ命令 GetState , および負論理で値を戻すマクロ命令 ReturnWithValue を追加した。

```

Line Addr  Packet                               Source Program
  1                                     * RoboCube Program with Interrupt
  2                                     * made by Y. Yoshioka
  3                                     * 2004.05.21   Ver.31
  4 0000 *150002040300000A0000E0_      BindBlock
      0001 *1500020403010A100000E2_
      0002 *1500020403020A110000E4_
      0003 *1500020403030B100000E5_
      0004 *1500020403040B1A0000F7_
      0005 *1500020403050C100000E8_
      0006 *1500020403060C1A0000FA_
      0007 *1500020403070D100000EB_
      0008 *1500020403080E100000ED_
      0009 *1500020403090F100000EF_
      000A *15000204030A11100000E3_
      000B *15000204020000000000CE_
  5 0000 *1500010000000000000000C7_      Subroutine
  6                                     *
  7                                     initialize
  8 0001 *15000101070000010002D2_      Motor    0,SET_MODE,VEL_MODE
  9 0002 *15000102070000010103D5_      Motor    1,SET_MODE,VEL_MODE
10 0003 *150001030701001E0004EC_      Motor    0,SET_LEVEL,30
11 0004 *150001040701001E0105EF_      Motor    1,SET_LEVEL,30
12 0005 *15000105070000080206E3_      Buzzer   0,SET_MODE,MODE8
13 0006 *15000106070100320207E3_      Buzzer   0,SET_LEVEL,50
14 0007 *15000107070500320208E9_      Buzzer   0,SET_DATA,50
15 0008 *15000108070200010209E4_      Buzzer   0,ACTION,ON
16 0009 *1500010907000008030AF3_      Buzzer   10,SET_MODE,MODE8
17 000A *1500010A07010032030BFA_      Buzzer   10,SET_LEVEL,50
18 000B *1500010B07050032030C00_      Buzzer   10,SET_DATA,50
19 000C *1500010C07020001030DFB_      Buzzer   10,ACTION,ON
20 000D *1500010D07000001080E00_      Optical  0,SET_MODE,REF_MODE
21 000E *1500010E070600FA080F2E_      Optical  0,SET_ALARM1,250
22 000F *1500010F070700FA08101B_      Optical  0,SET_ALARM2,250
23 0010 *15000110070000020711DA_      Ultrasonic 0,SET_MODE,ALM_MODE
24 0011 *15000111070100020712DD_      Ultrasonic 0,SET_LEVEL,MEDIUM
25 0012 *15000112070600640713EC_      Ultrasonic 0,SET_ALARM1,100
26 0013 *15000113070700C8071400_      Ultrasonic 0,SET_ALARM2,200
27 0014 *15000114160000000000D3_      Return
28                                     *
29                                     forward
30 0015 *15000115070600000016E1_      Motor    0,MOVE_CW
31 0016 *15000116070700000117E5_      Motor    1,MOVE_CCW
32 0017 *1500011707000000FF180B_      SendStrobe
33 0018 *15000118080003E8001902_      WaitTimer 1000
34 0019 *15000119160000000000D8_      Return
35                                     *
36                                     backward

```

```

37 001A *1500011A07070000001BFA_ Motor 0,MOVE_CCW
38 001B *1500011B070600000011CFC_ Motor 1,MOVE_CW
39 001C *1500011C07000000FF1D23_ SendStrobe
40 001D *1500011D080003E8001E1A_ WaitTimer 1000
41 001E *1500011E160000000000E4_ Return
42
43
44 001F *1500011F070600000020ED_ Motor 0,MOVE_CW
45 0020 *150001200706000000121DA_ Motor 1,MOVE_CW
46 0021 *1500012107000000FF2201_ SendStrobe
47 0022 *15000122080003E80023F8_ WaitTimer 1000
48 0023 *15000123160000000000D3_ Return
49
50
51 0024 *15000124070700000025E2_ Motor 0,MOVE_CCW
52 0025 *150001250707000000126E5_ Motor 1,MOVE_CCW
53 0026 *1500012607000000FF270B_ SendStrobe
54 0027 *15000127080003E8002802_ WaitTimer 1000
55 0028 *15000128160000000000D8_ Return
56
57
58 0029 *1500012907030000002AEF_ Motor 0,STOP
59 002A *1500012A070300000012BF9_ Motor 1,STOP
60 002B *1500012B07000000FF2C23_ SendStrobe
61 002C *1500012C080003E8002D1A_ WaitTimer 1000
62 002D *1500012D160000000000E4_ Return
63
64
65 002E *1500012E070000000042F01_ Light 0,SET_MODE,OFF
66 002F *1500012F070500FF04301E_ Light 0,SET_RED,255
67 0030 *150001300706000000431DF_ Light 0,SET_GREEN,0
68 0031 *150001310707000000432E2_ Light 0,SET_BLUE,0
69 0032 *15000132070200030433E2_ Light 0,ACTION,DIM_ON
70 0033 *15000133070200010434E2_ Light 0,ACTION,ON
71 0034 *150001340700000000535E2_ Light 10,SET_MODE,OFF
72 0035 *15000135070500FF053615_ Light 10,SET_RED,255
73 0036 *150001360706000000537EC_ Light 10,SET_GREEN,0
74 0037 *150001370707000000538EF_ Light 10,SET_BLUE,0
75 0038 *15000138070200030539EF_ Light 10,ACTION,DIM_ON
76 0039 *1500013907020001053AF6_ Light 10,ACTION,ON
77 003A *1500013A160000000000E2_ Return
78
79
80 003B *1500013B070000000043CFD_ Light 0,SET_MODE,OFF
81 003C *1500013C070500000043D04_ Light 0,SET_RED,0
82 003D *1500013D070600FF043E33_ Light 0,SET_GREEN,255
83 003E *1500013E070700000043F0A_ Light 0,SET_BLUE,0
84 003F *1500013F070200030440F4_ Light 0,ACTION,DIM_ON

```

```

85 0040 *15000140070200010441DE_ Light 0,ACTION,ON
86 0041 *15000141070000000542DE_ Light 10,SET_MODE,OFF
87 0042 *15000142070500000543E5_ Light 10,SET_RED,0
88 0043 *15000143070600FF054414_ Light 10,SET_GREEN,255
89 0044 *15000144070700000545EB_ Light 10,SET_BLUE,0
90 0045 *15000145070200030546EB_ Light 10,ACTION,DIM_ON
91 0046 *15000146070200010547EB_ Light 10,ACTION,ON
92 0047 *15000147160000000000D9_ Return
93 *
94 blue
95 0048 *15000148070000000449EB_ Light 0,SET_MODE,OFF
96 0049 *1500014907050000044AF9_ Light 0,SET_RED,0
97 004A *1500014A07060000044B03_ Light 0,SET_GREEN,0
98 004B *1500014B070700FF044C32_ Light 0,SET_BLUE,255
99 004C *1500014C07020003044D06_ Light 0,ACTION,DIM_ON
100 004D *1500014D07020001044E06_ Light 0,ACTION,ON
101 004E *1500014E07000000054F06_ Light 10,SET_MODE,OFF
102 004F *1500014F070500000550F7_ Light 10,SET_RED,0
103 0050 *15000150070600000551E4_ Light 10,SET_GREEN,0
104 0051 *15000151070700FF055213_ Light 10,SET_BLUE,255
105 0052 *15000152070200030553E7_ Light 10,ACTION,DIM_ON
106 0053 *15000153070200010554E7_ Light 10,ACTION,ON
107 0054 *15000154160000000000D7_ Return
108 *
109 greenblue
110 0055 *15000155070000000456E7_ Light 0,SET_MODE,OFF
111 0056 *15000156070500000457EE_ Light 0,SET_RED,0
112 0057 *15000157070600FF04581D_ Light 0,SET_GREEN,255
113 0058 *15000158070700FF045920_ Light 0,SET_BLUE,255
114 0059 *1500015907020003045AFB_ Light 0,ACTION,DIM_ON
115 005A *1500015A07020001045B02_ Light 0,ACTION,ON
116 005B *1500015B07000000055C02_ Light 10,SET_MODE,OFF
117 005C *1500015C07050000055D09_ Light 10,SET_RED,0
118 005D *1500015D070600FF055E38_ Light 10,SET_GREEN,255
119 005E *1500015E070700FF055F3B_ Light 10,SET_BLUE,255
120 005F *1500015F070200030560F9_ Light 10,ACTION,DIM_ON
121 0060 *15000160070200010561E3_ Light 10,ACTION,ON
122 0061 *15000161160000000000D5_ Return
123 *
124 redgreen
125 0062 *15000162070000000463E3_ Light 0,SET_MODE,OFF
126 0063 *15000163070500FF046416_ Light 0,SET_RED,255
127 0064 *15000164070600FF046519_ Light 0,SET_GREEN,255
128 0065 *15000165070700000466F0_ Light 0,SET_BLUE,0
129 0066 *15000166070200030467F0_ Light 0,ACTION,DIM_ON
130 0067 *15000167070200010468F0_ Light 0,ACTION,ON
131 0068 *15000168070000000569F0_ Light 10,SET_MODE,OFF
132 0069 *15000169070500FF056A2A_ Light 10,SET_RED,255

```

```

133 006A *1500016A070600FF056B34_ Light 10,SET_GREEN,255
134 006B *1500016B07070000056C0B_ Light 10,SET_BLUE,0
135 006C *1500016C07020003056D0B_ Light 10,ACTION,DIM_ON
136 006D *1500016D07020001056E0B_ Light 10,ACTION,ON
137 006E *1500016E160000000000E9_ Return
138
139 *
redblue
140 006F *1500016F070000000470F5_ Light 0,SET_MODE,OFF
141 0070 *15000170070500FF047112_ Light 0,SET_RED,255
142 0071 *15000171070600000472E9_ Light 0,SET_GREEN,0
143 0072 *15000172070700FF047318_ Light 0,SET_BLUE,255
144 0073 *15000173070200030474EC_ Light 0,ACTION,DIM_ON
145 0074 *15000174070200010475EC_ Light 0,ACTION,ON
146 0075 *15000175070000000576EC_ Light 10,SET_MODE,OFF
147 0076 *15000176070500FF05771F_ Light 10,SET_RED,255
148 0077 *15000177070600000578F6_ Light 10,SET_GREEN,0
149 0078 *15000178070700FF057925_ Light 10,SET_BLUE,255
150 0079 *1500017907020003057A00_ Light 10,ACTION,DIM_ON
151 007A *1500017A07020001057B07_ Light 10,ACTION,ON
152 007B *1500017B160000000000E7_ Return
153
154 *
white
155 007C *1500017C07000000047D07_ Light 0,SET_MODE,OFF
156 007D *1500017D070500FF047E3A_ Light 0,SET_RED,255
157 007E *1500017E070600FF047F3D_ Light 0,SET_GREEN,255
158 007F *1500017F070700FF04802A_ Light 0,SET_BLUE,255
159 0080 *15000180070200030481E8_ Light 0,ACTION,DIM_ON
160 0081 *15000181070200010482E8_ Light 0,ACTION,ON
161 0082 *15000182070000000583E8_ Light 10,SET_MODE,OFF
162 0083 *15000183070500FF05841B_ Light 10,SET_RED,255
163 0084 *15000184070600FF05851E_ Light 10,SET_GREEN,255
164 0085 *15000185070700FF058621_ Light 10,SET_BLUE,255
165 0086 *15000186070200030587F5_ Light 10,ACTION,DIM_ON
166 0087 *15000187070200010588F5_ Light 10,ACTION,ON
167 0088 *15000188160000000000DE_ Return
168
169 *
optical1
170 0089 *150001890A020008008A0C_ LoadAx OPTICAL,0,2
171 008A *1500018A0C000000008B0D_ PopOneData
172 008B *1500018B0F00000008C8C2D_ TestAx 0
173 008C *1500018C0C000000008D11_ PopOneData
174 008D *1500018D26000000008E08_ NotAx 0
175 008E *1500018E0CFF0000008F41_ PopOneData
176 008F *1500018F160000000000EC_ Return
177
178 *
optical2
179 0090 *150001900A0200080091F5_ LoadAx OPTICAL,0,2
180 0091 *150001910C0000000092EF_ PopOneData

```

```

181 0092 *150001920F010000939301_ TestAx 1
182 0093 *150001930C0000000094F3_ PopOneData
183 0094 *15000194260000000095EA_ NotAx 0
184 0095 *150001950CFF0000009623_ PopOneData
185 0096 *1500019616000000000DD_ Return
186
187
188 0097 *150001970A020006009801_ LoadAx TOUCH,0,2
189 0098 *150001980C0000000099FD_ PopOneData
190 0099 *150001990F0200009A9A25_ TestAx 2
191 009A *1500019A0C000000009B0F_ PopOneData
192 009B *1500019B26000000009C06_ NotAx 0
193 009C *1500019C0CFF0000009D3F_ PopOneData
194 009D *1500019D16000000000EB_ Return
195
196
197 009E *1500019E0A020006009F1D_ LoadAx TOUCH,0,2
198 009F *1500019F0C00000000A00A_ PopOneData
199 00A0 *150001A00F060000A1A118_ TestAx 6
200 00A1 *150001A10C00000000A2FF_ PopOneData
201 00A2 *150001A22600000000A3F6_ NotAx 0
202 00A3 *150001A30CFF000000A42F_ PopOneData
203 00A4 *150001A416000000000E3_ Return
204
205
206 00A5 *150001A50A02000600A60D_ LoadAx TOUCH,0,2
207 00A6 *150001A60C00000000A709_ PopOneData
208 00A7 *150001A70F030000A8A82A_ TestAx 3
209 00A8 *150001A80C00000000A90D_ PopOneData
210 00A9 *150001A92600000000AA0B_ NotAx 0
211 00AA *150001AA0CFF000000AB4B_ PopOneData
212 00AB *150001AB16000000000F1_ Return
213
214
215 00AC *150001AC0A02000600AD29_ LoadAx TOUCH,0,2
216 00AD *150001AD0C00000000AE25_ PopOneData
217 00AE *150001AE0F070000AF58_ TestAx 7
218 00AF *150001AF0C00000000B013_ PopOneData
219 00B0 *150001B02600000000B1F4_ NotAx 0
220 00B1 *150001B10CFF000000B22D_ PopOneData
221 00B2 *150001B216000000000E2_ Return
222
223
224 00B3 *150001B30A02000600B40B_ LoadAx TOUCH,0,2
225 00B4 *150001B40C00000000B507_ PopOneData
226 00B5 *150001B50F010000B6B625_ TestAx 1
227 00B6 *150001B60C00000000B70B_ PopOneData
228 00B7 *150001B72600000000B802_ NotAx 0

```

```
229 00B8 *150001B80CFF000000B93B_ PopOneData
230 00B9 *150001B9160000000000E9_ Return
231 *
232 touch5
233 00BA *150001BA0A02000600BB27_ LoadAx TOUCH,0,2
234 00BB *150001BB0C00000000BC23_ PopOneData
235 00BC *150001BC0F050000BDBD53_ TestAx 5
236 00BD *150001BD0C00000000BE27_ PopOneData
237 00BE *150001BE2600000000BF1E_ NotAx 0
238 00BF *150001BF0CFF000000C041_ PopOneData
239 00C0 *150001C0160000000000E1_ Return
240 *
241 touch0
242 00C1 *150001C10A02000600C209_ LoadAx TOUCH,0,2
243 00C2 *150001C20C00000000C305_ PopOneData
244 00C3 *150001C30F000000C4C421_ TestAx 0
245 00C4 *150001C40C00000000C509_ PopOneData
246 00C5 *150001C52600000000C600_ NotAx 0
247 00C6 *150001C60CFF000000C739_ PopOneData
248 00C7 *150001C7160000000000E8_ Return
249 *
250 touch4
251 00C8 *150001C80A02000600C917_ LoadAx TOUCH,0,2
252 00C9 *150001C90C00000000CA1A_ PopOneData
253 00CA *150001CA0F040000CBCB4F_ TestAx 4
254 00CB *150001CB0C00000000CC25_ PopOneData
255 00CC *150001CC2600000000CD1C_ NotAx 0
256 00CD *150001CD0CFF000000CE55_ PopOneData
257 00CE *150001CE160000000000F6_ Return
258 *
259 ultra0
260 00CF *150001CF0A02000700D01E_ LoadAx ULTRASONIC,0,2
261 00D0 *150001D00C00000000D103_ PopOneData
262 00D1 *150001D10F000000D2D21E_ TestAx 0
263 00D2 *150001D20C00000000D307_ PopOneData
264 00D3 *150001D32600000000D4FE_ NotAx 0
265 00D4 *150001D40CFF000000D537_ PopOneData
266 00D5 *150001D5160000000000E7_ Return
267 *
268 ultra1
269 00D6 *150001D60A02000700D716_ LoadAx ULTRASONIC,0,2
270 00D7 *150001D70C00000000D811_ PopOneData
271 00D8 *150001D80F010000D9D934_ TestAx 1
272 00D9 *150001D90C00000000DA1C_ PopOneData
273 00DA *150001DA2600000000DB1A_ NotAx 0
274 00DB *150001DB0CFF000000DC53_ PopOneData
275 00DC *150001DC160000000000F5_ Return
276 *
```


表 2.2 ROBOCUBE 用アセンブリ言語設計の一例 (1/2)

命令記述	*1500 mp no op dt data it jp ck _
BindBlock	*1500 02 04 03 00 0008 00 00 D7 _ *1500 02 04 03 01 0A10 00 00 E2 _ *1500 02 04 03 02 0A11 00 00 E4 _ *1500 02 04 03 03 0B10 00 00 E5 _ *1500 02 04 03 04 0C10 00 00 E7 _ *1500 02 04 03 05 0D10 00 00 E9 _ *1500 02 04 03 06 0E10 00 00 EB _ *1500 02 04 03 07 0F10 00 00 ED _ *1500 02 04 03 08 1110 00 00 DA _ *1500 02 04 02 00 0000 00 00 CE _
Mainroutine	*1500 00 00 00 00 0000 00 00 C6 _
Subroutine	*1500 01 00 00 00 0000 00 00 C7 _
Nop	*1500 00 no 06 00 0000 00 j1 ck _
CallSub label[,nodx,it,jp1]	*1500 00 no 00 js nodx it j1 ck _
CallSubAndTest label[,jp2,jp1]	*1500 00 no 01 js 0000 j2 j1 ck _
Wait time[,jp1]	*1500 mp no 02 00 time 00 j1 ck _
RandomJump [jp3],jp2[,jp1]	*1500 mp no 03 j3 0000 j2 j1 ck _
SetCounter ct,data[,jp1]	*1500 mp no 04 ct data 00 j1 ck _
DecCounter ct[,jp2,jp1]	*1500 mp no 05 ct 0000 j2 j1 ck _
Jump jp1	*1500 mp no 06 00 0000 00 j1 ck _
Stop	*1500 mp no 06 00 0000 00 00 ck _
block IDL,dt,data[,jp1]	*1500 mp no 07 dt data nd j1 ck _
WaitTimer data[,jp1]	*1500 mp no 08 00 time 00 j1 ck _
WaitStatus block,IDL,code[,jp1]	*1500 mp no 09 cd 0000 nd j1 ck _
LoadAx 0,data[,jp1]	*1500 mp no 0A 00 data 00 j1 ck _
LoadAx block,IDL,op3[,jp1]	*1500 mp no 0A o3 node 00 j1 ck _
SaveAx block,IDL,op3,data[,jp1]	*1500 mp no 0B o3 data nd j1 ck _
PopOneData [jp1]	*1500 mp no 0C 00 0000 00 j1 ck _
PushOneData [jp1]	*1500 mp no 0D 00 0000 00 j1 ck _
ComplimentAx [jp1]	*1500 mp no 0E 00 0000 00 j1 ck _
TestAx bit[,jp2,jp1]	*1500 mp no 0F bt 0000 j2 j1 ck _
AndAx 0,data[,jp2,jp1]	*1500 mp no 10 00 data j2 j1 ck _
AndAx block,IDL,op3[,jp2,jp1]	*1500 mp no 10 o3 node j2 j1 ck _
OrAx 0,data[,jp2,jp1]	*1500 mp no 11 00 data j2 j1 ck _
OrAx block,IDL,op3[,jp2,jp1]	*1500 mp no 11 o3 node j2 j1 ck _
CompareAx 0,data[,jp2,jp1]	*1500 mp no 12 00 data j2 j1 ck _
CompareAx block,IDL,op3[,jp2,jp1]	*1500 mp no 12 o3 node j2 j1 ck _
LargeAx 0,data[,jp2,jp1]	*1500 mp no 13 00 data j2 j1 ck _
LargeAx block,IDL,op3[,jp2,jp1]	*1500 mp no 13 o3 node j2 j1 ck _
SmallAx 0,data[,jp2,jp1]	*1500 mp no 14 00 data j2 j1 ck _
SmallAx block,IDL,op3[,jp2,jp1]	*1500 mp no 14 o3 node j2 j1 ck _

表 2.3 ROBOCUBE 用アセンブリ言語設計の一例 (2/2)

命令記述	*1500 mp no op dt data it jp ck _
UpdateTest [jp1]	*1500 mp no 15 00 0000 00 j1 ck _
Return	*1500 mp no 16 00 0000 00 00 ck _
PopTwoData [jp1]	*1500 mp no 25 00 0000 00 j1 ck _
NotAx [jp2,jp1]	*1500 mp no 26 00 0000 j2 j1 ck _
NotEqual [jp2,jp1]	*1500 mp no 27 00 0000 j2 j1 ck _
CallSubSetInterrupt label[,nodx,it,jp1]	*1500 00 no 00 js nodx it j1 ck _
Interrupt label[,jp2,jp1]	*1500 00 no 01 js 0000 j2 j1 ck _
SendStrobe [jp1]	*1500 00 no 07 00 0000 FF j1 ck _
GetState block,IDL,op3[,jp2,jp1] (ブロックから状態を取り込むマクロ命令)	LoadAx block,IDL,op3 PopOneData TestAx op3[,jp2,jp1]
ModeSetMaster [jp1]	*1500 00 no 0C ad 0000 00 j1 ck _
NodeBlockControl op1,data[,jp2,jp1]	*1500 00 no 07 o1 data j1 j1 ck _
ReturnWithValue (負論理で値をメインへ戻るマクロ命令)	PopOneData NotAx 0 PopOneData Return
Execution	*1500 02 00 19 00 0000 00 00 D2 _

2.4 まとめ

以上，アセンブリ言語およびそのコンパイラ（アセンブラ）の特徴を示してきた。そして，ROBOCUBE 用のアセンブリ言語を設計し，そのアセンブラを作成した（バージョン 1.02）。提供するアセンブラは，基本ブロックのアセンブラ `cubeassem5.c`（バージョン 2.05）およびバージョンアップセットのアセンブラ `cubeassem31.c`（バージョン 3.01）の実行形式である（なお，ソースプログラムは有償で提供する）。このようなアセンブリ言語を利用すれば，大きなプログラム開発や複雑な動作を行うプログラム開発が容易になったといえる。しかし，このアセンブリ言語記述では，ROBOCUBE のハードウェア仕様をかなり理解していないと，プログラミングが非常に困難である。æ

Chapter 3

ROBOCUBE 用マクロ命令

ROBOCUBE はパケットによって動作していることは、前でも述べた。第 2 章に示したように、1 パケットに対応する文字列で記述できるようなアセンブリ言語設計にすると、どのような組み合わせでプログラミングを行えばよいか分かりづらい。また、第 2 章で示した ROBOCUBE 動作プログラム例に示すように、動作設定パケットの組み合わせは決まっている。そこで、動作を表す文字列を定義するマクロ命令で行う記述法を示す。

3.1 マクロ命令による命令設計法

マクロ命令とは、複数の命令を 1 つの文字列で表現する記述法である。特に、複数の命令の実行順序が決まっている場合や、分りにくい命令が並ぶ場合等にマクロ命令が利用される。第 2 章で示した ROBOCUBE 動作プログラム例を見ると、決まった命令列があることが分かる。そこで、動作を表す文字列（マクロ命令）を定義して、その動作を設定する複数のパケットを生成するようにする。例えば、前進を Forward、後退を Backward などのように、動作で表す方法である。このようなマクロ命令を定義し、そのマクロ命令が表 2.2 および表 2.3 に示す命令列とどのように対応しているかを示すと表 3.1 および表 3.2 のようになる。表 2.2 および表 2.3 に示す命令に表 3.1 および表 3.2 のマクロ命令を加えたアセンブラ（バージョン 2）を作成した。したがって、第 2 章で示した ROBOCUBE 動作プログラムもコンパイルすることができる。このため、マクロ命令で定義されていない命令については、表 2.2 および表 2.3 に示す命令を利用してプログラミングすることになる。また、バージョンアップセットのセンターブロック内には、ライト、ブザー、カレンダーが内蔵されている。ライトおよびブザーには IDL に 10 が割り当てられている。なお、このアセンブラでは、使用するブロックを予め登録しておく方法を取った。このアセンブラについては別媒体で提供する。

表 3.1 マイクロ命令による命令設計例 (1/3)

命令記述	表 2.2 及び 表 2.3 の命令記述	説明
MotorSetMode mode,speed1,speed2[,jp1]	Motor 0,SET_MODE,VEL_MODE Motor 1,SET_MODE,VEL_MODE Motor 0,SET_LEVEL,speed1 Motor 1,SET_LEVEL,speed2[,jp1]	モード設定
Forward [jp1]	Motor 0,MOVE_CW Motor 1,MOVE_CCW SendStrobe	前進
Backward [jp1]	Motor 0,MOVE_CCW Motor 1,MOVE_CW SendStrobe	後退
RotateRight [jp1]	Motor 0,MOVE_CW Motor 1,MOVE_CW SendStrobe	右回転
RotateLeft [jp1]	Motor 0,MOVE_CCW Motor 1,MOVE_CCW SendStrobe	左回転
TurnRight [jp1]	Motor 0,MOVE_CW Motor 1,STOP SendStrobe	右向きへ
TurnLeft [jp1]	Motor 1,MOVE_CCW Motor 0,STOP SendStrobe	左向きへ
MotorStop [jp1]	Motor 0,STOP Motor 1,STOP SendStrobe	モータ停止
BuzzerOn mode,level,temp [,jp1]	Buzzer 0,SET_MODE,mode Buzzer 0,SET_LEVEL,level Buzzer 0,SET_DATA,temp Buzzer 0,ACTION,ON[,jp1]	ブザーオン
BuzzerOff [jp1]	Buzzer 0,ACTION,OFF[,jp1]	ブザーオフ
LightOn mode,red,green,blue[,jp1]	Light 0,SET_MODE,mode Light 0,SET_RED,red Light 0,SET_GREEN,green Light 0,SET_BLUE,blue Light 0,ACTION,DIM_ON Light 0,ACTION,ON[,jp1]	ライトオン 赤の強さ 緑の強さ 青の強さ
LightRedOn mode,red[,jp1]	Light 0,SET_MODE,mode Light 0,SET_RED,red Light 0,ACTION,DIM_ON Light 0,ACTION,ON[,jp1]	ライトオン 赤の強さ
LightGreenOn mode,green[,jp1]	Light 0,SET_MODE,mode Light 0,SET_GREEN,green Light 0,ACTION,DIM_ON Light 0,ACTION,ON[,jp1]	ライトオン 緑の強さ

表 3.2 マイクロ命令による命令設計例 (2/3)

命令記述	表 2.2 及び 表 2.3 の命令記述	説明
LightBlueOn mode,blue[,jp1]	Light 0,SET_MODE,mode Light 0,SET_BLUE,blue Light 0,ACTION,DIM_ON Light 0,ACTION,ON[,jp1]	ライトオン 青の強さ
LightOff	Light 0,SET_MODE,OFF[,jp1]	ライトオフ
BuzzerOn2 mode,level,temp [,jp1] (センターブロック内ブザー)	Buzzer 0,SET_MODE,mode Buzzer 0,SET_LEVEL,level Buzzer 0,SET_DATA,temp Buzzer 0,ACTION,ON[,jp1]	ブザーオン
BuzzerOff2 [jp1] (センターブロック内ブザー)	Buzzer 0,ACTION,OFF[,jp1]	ブザーオフ
LightOn2 mode,red,green,blue[,jp1] (センターブロック内ライト)	Light 0,SET_MODE,mode Light 0,SET_RED,red Light 0,SET_GREEN,green Light 0,SET_BLUE,blue Light 0,ACTION,DIM_ON Light 0,ACTION,ON[,jp1]	ライトオン 赤の強さ 緑の強さ 青の強さ
LightRedOn2 mode,red[,jp1] (センターブロック内ライト)	Light 0,SET_MODE,mode Light 0,SET_RED,red Light 0,ACTION,DIM_ON Light 0,ACTION,ON[,jp1]	ライトオン 赤の強さ
LightGreenOn2 mode,green[,jp1] (センターブロック内ライト)	Light 0,SET_MODE,mode Light 0,SET_GREEN,green Light 0,ACTION,DIM_ON Light 0,ACTION,ON[,jp1]	ライトオン 緑の強さ
LightBlueOn2 mode,blue[,jp1] (センターブロック内ライト)	Light 0,SET_MODE,mode Light 0,SET_BLUE,blue Light 0,ACTION,DIM_ON Light 0,ACTION,ON[,jp1]	ライトオン 青の強さ
LightOff2 (センターブロック内ライト)	Light 0,SET_MODE,OFF[,jp1]	ライトオフ
UltrasonicSetMode mode,level,alarm1,alarm2[,jp1]	Ultrasonic 0,SET_MODE,ALM_MODE Ultrasonic 0,SET_LEVEL,level Ultrasonic 0,SET_ALARM1,alarm1 Ultrasonic 0,SET_ALARM2,alarm2[,jp1]	超音波 センサー 設定
UltrasonicGetState op1[,jp2,jp1]	LoadAx ULTRASONIC,0,2 PopOneData TestAx op1[,jp2,jp1]	状態取り込み
OpticalSetMode mode,level1,level2[,jp1]	Optical 0,SET_MODE,REF_MODE Optical 0,SET_ALARM1,level1 Optical 0,SET_ALARM2,level2[,jp1]	モード設定 レベル 1 レベル 2
OpticalGetState op1[,jp2,jp1]	LoadAx OPTICAL,0,2 PopOneData TestAx op1[,jp2,jp1]	状態取り込み

表 3.3 マイクロ命令による命令設計例 (3/3)

命令記述	表 2.2 及び 表 2.3 の命令記述	説明
TouchGetState op1,[jp2,jp1]	LoadAx TOUCH,0,2 PopOneData TestAx op1[,jp2,jp1]	状態取り込み
LogicOR [jp2,jp1]	PopTwoData OrAx 0[,jp2,jp1]	論理和
LogicAND [jp2,jp1]	PopTwoData AndAx 0[,jp2,jp1]	論理積
LogicNOT [jp2,jp1]	PopOneData NotAx [jp2,jp1]	論理否定
ReturnWithValue (負論理で値をメインへ)	PopOneData NotAx 0 PopOneData Return	値を伴う戻り

3.2 マクロ命令によるプログラミング

表 3.1 および表 3.2 で定義したマクロ命令を用いて、第 2 章に示した ROBOCUBE 動作プログラム例とほぼ同じ動作を行うソースプログラムは、以下のようなになる。このプログラムにおいて、BuzzerOn2 や LightOn2 などは、センターブロック内蔵のブザーおよびライトへの命令である。

```
* test program with interrupt
* made by Yoshio Yoshioka
* 2004.05.21    Ver.03
*
speed1      equ    30
speed2      equ    30
buzlevel    equ    50
buztempo    equ    30
red0        equ    0
red255      equ    255
green0      equ    0
green255    equ    255
blue0       equ    0
blue255     equ    255
optright    equ    250
optleft     equ    250
salarm1     equ    100
salarm2     equ    200
time        equ    500
*
  BindBlock
  Subroutine
* motor initialize
```

```
initialize
  MotorSetMode      VEL_MODE,speed1,speed2
  BuzzerOn          MODE6,buzlevel,buztempo
  BuzzerOn2         MODE6,buzlevel,buztempo
  OpticalSetMode    REF_MODE,optleft,optright
  UltrasonicSetMode ALM_MODE,MEDIUM,salarm1,salarm2
  Return
*
forward
  LightOn  CONTINUOUS,red255,green0,blue0
  LightOn2 CONTINUOUS,red255,green0,blue0
  Forward
  WaitTimer      time
  Return
*
backward
  LightOn  CONTINUOUS,red0,green255,blue0
  LightOn2 CONTINUOUS,red0,green255,blue0
  Backward
  WaitTimer      time
  Return
*
rotateright
  LightOn  CONTINUOUS,red0,green0,blue255
  LightOn2 CONTINUOUS,red0,green0,blue255
  RotateRight
  WaitTimer      time
  Return
*
rotateleft
  LightOn  CONTINUOUS,red0,green255,blue255
  LightOn2 CONTINUOUS,red0,green255,blue255
  RotateLeft
  WaitTimer      time
  Return
*
stop
  LightOn  CONTINUOUS,red255,green255,blue255
  LightOn2 CONTINUOUS,red255,green255,blue255
  MotorStop
  WaitTimer      time
  Return
*
opticalgetleft
  OpticalGetState LEFT
  ReturnWithValue
*
opticalgetright
```

```
    OpticalGetState  RIGHT
    ReturnWithValue
*
touchgettopR
    TouchGetState    R.TOP
    ReturnWithValue
*
touchgettopL
    TouchGetState    L.TOP
    ReturnWithValue
*
touchgetleftR
    TouchGetState    R.LEFT
    ReturnWithValue
*
touchgetleftL
    TouchGetState    L.LEFT
    ReturnWithValue
*
touchgetrightR
    TouchGetState    R.RIGHT
    ReturnWithValue
*
touchgetrightL
    TouchGetState    L.RIGHT
    ReturnWithValue
*
touchgetbottomR
    TouchGetState    R.BOTTOM
    ReturnWithValue
*
touchgetbottomL
    TouchGetState    L.BOTTOM
    ReturnWithValue
*
ultrasonicget0
    UltrasonicGetState  0
    ReturnWithValue
*
ultrasonicget1
    UltrasonicGetState  1
    ReturnWithValue
*
    Mainroutine
    CallSub            initialize
loopback
    CallSub            forward,$ff,int
    Jump              loopback
```

```

*
int
  CallSubAndTest  touchgettopR,next04
  CallSubAndTest  touchgetleftR,next04
  CallSubAndTest  touchgetrightR,next04
  CallSubAndTest  touchgetbottomR,next04
  CallSubAndTest  touchgettopL,next04
  CallSubAndTest  touchgetleftL,next04
  CallSubAndTest  touchgetrightL,next04
  CallSubAndTest  touchgetbottomL,next04
  CallSubAndTest  ultrasonicget0,next01
  CallSubAndTest  ultrasonicget1,next01
  CallSubAndTest  opticalgetleft,next02
  CallSubAndTest  opticalgetright,next03
  Jump           loopback
next01
  CallSub        backward
  Jump           int
next02
  CallSub        rotateleft
  Jump           loopback
next03
  CallSub        stop
  Jump           loopback
next04
  CallSub        rotateright
  Jump           int
Execute
end

```

このソースプログラムをみると、非常に分かりやすいし、プログラミングし易くなったことが分かる。このソースプログラム（ファイル名 `acube_int03.asm`）をアセンブラにかけると以下のコンパイルリスト（ファイル名 `assemble.lst`）が出力される。なお、このプログラムの詳細説明については、第4章で示す。

Line	Addr	Packet	Source	Program
1			*	test program with interrupt
2			*	made by Yoshio Yoshioka
3			*	2004.05.21 Ver.03
4			*	
5			speed1	equ 30
6			speed2	equ 30
7			buzlevel	equ 50
8			buztempo	equ 30
9			red0	equ 0
10			red255	equ 255

```

11          green0      equ      0
12          green255    equ      255
13          blue0       equ      0
14          blue255     equ      255
15          optright    equ      250
16          optleft     equ      250
17          salarm1     equ      100
18          salarm2     equ      200
19          time        equ      500
20          *
21 0000 *150002040300000A0000E0_  BindBlock
    0001 *1500020403010A100000E2_
    0002 *1500020403020A110000E4_
    0003 *1500020403030B100000E5_
    0004 *1500020403040B1A0000F7_
    0005 *1500020403050C100000E8_
    0006 *1500020403060C1A0000FA_
    0007 *1500020403070D100000EB_
    0008 *1500020403080E100000ED_
    0009 *1500020403090F100000EF_
    000A *15000204030A11100000E3_
    000B *15000204020000000000CE_
22 0000 *15000100000000000000C7_  Subroutine
23          * motor initialize
24          initialize
25 0001 *15000101070000010002D2_  MotorSetMode      VEL_MODE,speed1,speed2
    0002 *15000102070000010103D5_
    0003 *150001030701001E0004EC_
    0004 *150001040701001E0105EF_
26 0005 *15000105070000060206E1_  BuzzerOn          MODE6,buzlevel,buztempo
    0006 *15000106070100320207E3_
    0007 *150001070705001E0208FA_
    0008 *15000108070200010209E4_
27 0009 *1500010907000006030AF1_  BuzzerOn2         MODE6,buzlevel,buztempo
    000A *1500010A07010032030BFA_
    000B *1500010B0705001E030C11_
    000C *1500010C07020001030DFB_
28 000D *1500010D07000001080E00_  OpticalSetMode    REF_MODE,optleft,opttright
    000E *1500010E070600FA080F2E_
    000F *1500010F070700FA08101B_
29 0010 *15000110070000020711DA_  UltrasonicSetMode ALM_MODE,MEDIUM,salarm1,salarm2
    0011 *15000111070100020712DD_
    0012 *15000112070600640713EC_
    0013 *15000113070700C8071400_
30 0014 *15000114160000000000D3_  Return
31          *
32          forward
33 0015 *15000115070000000416DF_  LightOn           CONTINUOUS,red255,green0,blue0

```

```

0016 *15000116070500FF041712_
0017 *15000117070600000418E9_
0018 *15000118070700000419EC_
0019 *1500011907020003041AF3_
001A *1500011A07020001041BFA_
34 001B *1500011B07000000051CFA_   LightOn2   CONTINUOUS,red255,green0,blue0
001C *1500011C070500FF051D2D_
001D *1500011D07060000051E04_
001E *1500011E07070000051F07_
001F *1500011F070200030520F1_
0020 *15000120070200010521DB_
35 0021 *15000121070600000022DB_   Forward
0022 *15000122070700000123DF_
0023 *1500012307000000FF2405_
36 0024 *15000124080001F40025F7_   WaitTimer   time
37 0025 *15000125160000000000D5_   Return
38
39
40 0026 *15000126070000000427E3_   LightOn     CONTINUOUS,red0,green255,blue0
0027 *15000127070500000428EA_
0028 *15000128070600FF042919_
0029 *1500012907070000042AF7_
002A *1500012A07020003042BFE_
002B *1500012B07020001042CFE_
41 002C *1500012C07000000052DFE_   LightOn2   CONTINUOUS,red0,green255,blue0
002D *1500012D07050000052E05_
002E *1500012E070600FF052F34_
002F *1500012F070700000530F5_
0030 *15000130070200030531DF_
0031 *15000131070200010532DF_
42 0032 *15000132070700000033E0_   Backward
0033 *15000133070600000134E2_
0034 *1500013407000000FF3509_
43 0035 *15000135080001F40036FB_   WaitTimer   time
44 0036 *15000136160000000000D7_   Return
45
46
47 0037 *15000137070000000438E7_   LightOn     CONTINUOUS,red0,green0,blue255
0038 *15000138070500000439EE_
0039 *1500013907060000043AF8_
003A *1500013A070700FF043B2E_
003B *1500013B07020003043C02_
003C *1500013C07020001043D02_
48 003D *1500013D07000000053E02_   LightOn2   CONTINUOUS,red0,green0,blue255
003E *1500013E07050000053F09_
003F *1500013F070600000540F6_
0040 *15000140070700FF05410F_
0041 *15000141070200030542E3_

```

```

0042 *15000142070200010543E3_
49 0043 *15000143070600000044E3_ RotateRight
0044 *15000144070600000145E6_
0045 *1500014507000000FF460D_
50 0046 *15000146080001F40047FF_ WaitTimer time
51 0047 *15000147160000000000D9_ Return
52 *
53 rotateleft
54 0048 *15000148070000000449EB_ LightOn CONTINUOUS,red0,green255,blue255
0049 *1500014907050000044AF9_
004A *1500014A070600FF044B2F_
004B *1500014B070700FF044C32_
004C *1500014C07020003044D06_
004D *1500014D07020001044E06_
55 004E *1500014E07000000054F06_ LightOn2 CONTINUOUS,red0,green255,blue255
004F *1500014F070500000550F7_
0050 *15000150070600FF055110_
0051 *15000151070700FF055213_
0052 *15000152070200030553E7_
0053 *15000153070200010554E7_
56 0054 *15000154070700000055E8_ RotateLeft
0055 *15000155070700000156EB_
0056 *1500015607000000FF5711_
57 0057 *15000157080001F4005803_ WaitTimer time
58 0058 *15000158160000000000DB_ Return
59 *
60 stop
61 0059 *1500015907000000045AF6_ LightOn CONTINUOUS,red255,green255,blue255
005A *1500015A070500FF045B30_
005B *1500015B070600FF045C33_
005C *1500015C070700FF045D36_
005D *1500015D07020003045E0A_
005E *1500015E07020001045FOA_
62 005F *1500015F070000000560F4_ LightOn2 CONTINUOUS,red255,green255,blue255
0060 *15000160070500FF056111_
0061 *15000161070600FF056214_
0062 *15000162070700FF056317_
0063 *15000163070200030564EB_
0064 *15000164070200010565EB_
63 0065 *15000165070300000066E8_ MotorStop
0066 *15000166070300000167EB_
0067 *1500016707000000FF6815_
64 0068 *15000168080001F4006907_ WaitTimer time
65 0069 *15000169160000000000DD_ Return
66 *
67 opticalgetleft
68 006A *1500016A0A020008006B11_ OpticalGetState LEFT
006B *1500016B0C000000006COB_

```

```
006C *1500016C0F0000006D6D2A_
69 006D *1500016D0CFF0000006E3B_   ReturnWithValue
006E *1500016E26000000006F06_
006F *1500016F0CFD0000007027_
0070 *15000170160000000000D5_
70
71                                     *
72 0071 *150001710A0200080072F3_   OpticalGetState   RIGHT
0072 *150001720C0000000073ED_
0073 *150001730F0100007474FE_
73 0074 *150001740CFF000000751D_   ReturnWithValue
0075 *15000175260000000076E8_
0076 *150001760CFD000000771F_
0077 *15000177160000000000DC_
74
75                                     *
76 0078 *150001780A0200060079FF_   TouchGetState     R.TOP
0079 *150001790C000000007A02_
007A *1500017A0F0600007B7B2D_
77 007B *1500017B0CFF0000007C39_   ReturnWithValue
007C *1500017C26000000007D04_
007D *1500017D0CFD0000007E3B_
007E *1500017E160000000000EA_
78
79                                     *
80 007F *1500017F0A020006008005_   TouchGetState     L.TOP
0080 *150001800C0000000081EB_
0081 *150001810F0200008282FC_
81 0082 *150001820CFF000000831B_   ReturnWithValue
0083 *15000183260000000084E6_
0084 *150001840CFD000000851D_
0085 *15000185160000000000DB_
82
83                                     *
84 0086 *150001860A0200060087FD_   TouchGetState     R.LEFT
0087 *150001870C0000000088F9_
0088 *150001880F070000898916_
85 0089 *150001890CFF0000008A30_   ReturnWithValue
008A *1500018A26000000008B02_
008B *1500018B0CFD0000008C39_
008C *1500018C160000000000E9_
86
87                                     *
88 008D *1500018D0A020006008E19_   TouchGetState     L.LEFT
008E *1500018E0C000000008F15_
008F *1500018F0F030000909010_
89 0090 *150001900CFF0000009119_   ReturnWithValue
0091 *15000191260000000092E4_
```

```

0092 *150001920CFD000000931B_
0093 *15000193160000000000DA_
90 *
91 touchgetrightR
92 0094 *150001940A0200060095FB_ TouchGetState R.RIGHT
0095 *150001950C0000000096F7_
0096 *150001960F050000979711_
93 0097 *150001970CFF0000009827_ ReturnWithValue
0098 *15000198260000000099F2_
0099 *150001990CFD0000009A30_
009A *1500019A160000000000E8_
94 *
95 touchgetrightL
96 009B *1500019B0A020006009C17_ TouchGetState L.RIGHT
009C *1500019C0C000000009D13_
009D *1500019D0F0100009E9E37_
97 009E *1500019E0CFF0000009F43_ ReturnWithValue
009F *1500019F2600000000A0FF_
00A0 *150001A00CFD000000A127_
00A1 *150001A1160000000000E0_
98 *
99 touchgetbottomR
100 00A2 *150001A20A02000600A307_ TouchGetState R.BOTTOM
00A3 *150001A30C00000000A403_
00A4 *150001A40F040000A5A522_
101 00A5 *150001A50CFF000000A633_ ReturnWithValue
00A6 *150001A62600000000A7FE_
00A7 *150001A70CFD000000A835_
00A8 *150001A8160000000000E7_
102 *
103 touchgetbottomL
104 00A9 *150001A90A02000600AA1C_ TouchGetState L.BOTTOM
00AA *150001AA0C00000000AB1F_
00AB *150001AB0F000000ACAC48_
105 00AC *150001AC0CFF000000AD4F_ ReturnWithValue
00AD *150001AD2600000000AE1A_
00AE *150001AE0CFD000000AF51_
00AF *150001AF160000000000F5_
106 *
107 ultrasonicget0
108 00B0 *150001B00A02000700B106_ UltrasonicGetState 0
00B1 *150001B10C00000000B201_
00B2 *150001B20F000000B3B31B_
109 00B3 *150001B30CFF000000B431_ ReturnWithValue
00B4 *150001B42600000000B5FC_
00B5 *150001B50CFD000000B633_
00B6 *150001B6160000000000E6_
110 *

```

```

111                                     ultrasonicget1
112 00B7 *150001B70A02000700B814_   UltrasonicGetState   1
    00B8 *150001B80C00000000B90F_
    00B9 *150001B90F010000BABA3F_
113 00BA *150001BA0CFF000000BB4D_   ReturnWithValue
    00BB *150001BB2600000000BC18_
    00BC *150001BC0CFD000000BD4F_
    00BD *150001BD160000000000F4_
114                                     *
115 0000 *1500000000000000000000C6_   Mainroutine
116 0001 *15000001000100000002CA_   CallSub               initialize
117                                     loopback
118 0002 *15000002001500FF040301_   CallSub               forward,$ff,int
119 0003 *15000003060000000002D1_   Jump                  loopback
120                                     *
121                                     int
122 0004 *15000004017800001705E7_   CallSubAndTest       touchgettopR,next04
123 0005 *15000005018600001706E8_   CallSubAndTest       touchgetleftR,next04
124 0006 *15000006019400001707E9_   CallSubAndTest       touchgetrightR,next04
125 0007 *1500000701A200001708F1_   CallSubAndTest       touchgetbottomR,next04
126 0008 *15000008017F00001709FD_   CallSubAndTest       touchgettopL,next04
127 0009 *15000009018D0000170A05_   CallSubAndTest       touchgetleftL,next04
128 000A *1500000A019B0000170B0D_   CallSubAndTest       touchgetrightL,next04
129 000B *1500000B01A90000170C0E_   CallSubAndTest       touchgetbottomL,next04
130 000C *1500000C01B00000110D02_   CallSubAndTest       ultrasonicget0,next01
131 000D *1500000D01B70000110E0B_   CallSubAndTest       ultrasonicget1,next01
132 000E *1500000E016A0000130F0D_   CallSubAndTest       opticalgetleft,next02
133 000F *1500000F017100001510EC_   CallSubAndTest       opticalgetright,next03
134 0010 *15000010060000000002CF_   Jump                  loopback
135                                     next01
136 0011 *15000011002600000012D3_   CallSub               backward
137 0012 *15000012060000000004D3_   Jump                  int
138                                     next02
139 0013 *15000013004800000014DB_   CallSub               rotateleft
140 0014 *15000014060000000002D3_   Jump                  loopback
141                                     next03
142 0015 *15000015005900000016E1_   CallSub               stop
143 0016 *15000016060000000002D5_   Jump                  loopback
144                                     next04
145 0017 *15000017003700000018E1_   CallSub               rotateright
146 0018 *15000018060000000004D9_   Jump                  int
147 0019 *15000200190000000000D2_   Execute
148                                     end

```

3.3 まとめ

以上,ROBOCUBE 用のアセンブリ言語を設計し,そのアセンブラ(バージョン 2.05,バージョンアップセットに関してはバージョン 3.01)を作成した。このようなアセンブリ言語を利用すれば,大きなプログラム開発や複雑な動作を行うプログラム開発が容易になったといえる。しかしながら,アセンブリ言語は一般的にハードウェアの動作を十分理解していないと,よいプログラムを作成することができない。逆に言うならば,アセンブリ言語でプログラミングを行うと,ハードウェアの動作をよく理解できるということになる。

Chapter 4

プログラム記述

第 2 章および第 3 章に示した ROBOCUBE 自走プログラム例は、各センサーからの入力を検知し、モータを回転すると同時にライトを点灯するプログラム例である。これらのプログラムは、システムワット社から提供されたタイル言語によるプログラム開発ツールによって小さな自走プログラムを作成し、このプログラムから ROBOCUBE へ転送されるパケットを解析し、ROBOCUBE のハードウェアの動作を推定して記述したものである。このため、仕様書に書かれていない隠れた仕様がいくつか見られた。そこで、本章では、タイルに相当するサブルーティンの記述法、メインプログラムの記述法、隠れた仕様による問題点などについて述べる。なお、本章で示すプログラミング記述は、第 3 章で示したマクロ命令による命令記述を用いる。

4.1 サブルーティンのプログラミング

まず、ROBOCUBE のプログラム領域は、図 4.1 に示すように、mp=00 および no=01 ~ FF がメインルーティン格納領域、mp=01 および no=00 ~ FF がサブルーティン格納領域となっている。図 1.5 に示すように、サブルーティンが最初に記述され、サブルーティン領域に格納される。次に、メインルーティンが記述され、メインルーティン領域に格納される。そこで、ここでは、第 3 章に示す ROBOCUBE 動作プログラム例のサブルーティンについて、以下に説明を加える。

初期設定

初期設定の各行の意味は以下ようになる。

```
* motor initialize
initialize
  MotorSetMode      VEL_MODE, speed1, speed2      モータの設定
```

BuzzerOn	MODE6,buzlevel,buztempo	ブザーの初期設定
OpticalSetMode	REF_MODE,optleft,optright	光学センサーの初期設定
UltrasonicSetMode	ALM_MODE,MEDIUM,salarm1,salarm2	超音波センサーの初期設定
Return		メインルーティンへ

ここで、タッチセンサーについての初期設定は必要がない。また、VEL_MODE、MODE6、REF_MODE、ALM_MODE は、表 1.6 の引数名および値である。MEDIUM は表 1.7 の引数名および値である。speed1 および speed2 の値は 0~100 の範囲を、buzlevel の値は 0~255 の範囲を、buztemp の値は 8~250 の範囲を、optleft および optright の値は 0~255 の範囲を、salarm1 および salarm2 の値は 0~255 の範囲を入れる。これらの変数名については equ によって定義されているが、直接数値を記述することも可能である。



図 4.1 ROBOCUBE のプログラム領域

ライトオンとモータ制御

ライトオンとモータ制御の各行の意味は以下のようなになる。

forward		ラベル
LightOn	CONTINUOUS,red255,green0,blue0	ライトオン
Forward		モータ前進
WaitTimer	time	タイマー待ち
Return		メインルーティンへ

ここで、CONTINUOUS は表 1.6 に示す引数名および値である。また、red255、green0、blue0 の部分は 0~255 の範囲の値であり、直接記述することができる。

タッチセンサーからの入力

タッチセンサー入力ルーティンの各行の意味は以下ようになる。

touchgettopR		ラベル
TouchGetState	R.TOP	右タッチセンサー上オン
ReturnWithValue		上の信号値を伴ってメインへ

上記の他, L.TOP (左タッチセンサー上オン), R.LEFT (右タッチセンサー左オン), L.LEFT (左タッチセンサー左オン), R.RIGHT (右タッチセンサー右オン), L.RIGHT (左タッチセンサー右オン), R.BOTTOM (右タッチセンサー下オン) L.BOTTOM (左タッチセンサー下オン) がある。これらの文字列は予約語となっている。

光学センサーからの入力

光学センサー入力ルーティンの各行の意味は以下ようになる。

opticalgetleft		ラベル
OpticalGetState	LEFT	左光学センサーからの入力
ReturnWithValue		上の信号値を伴ってメインへ

上記の他, RIGHT がある。

超音波センサーからの入力

超音波センサー入力ルーティンの各行の意味は以下ようになる。

ultrasonicget		ラベル
UltrasonicGetState		超音波センサーからの入力
ReturnWithValue		上の信号値を伴ってメインへ

論理和を利用するサブルーティン

2 つのタッチセンサーのどちらかがオンになった場合に, 0 (負論理の真) の値を返すプログラムは以下ようになる。

touchgettop		
TouchGetState	R.TOP	右タッチセンサーの上オン
TouchGetState	L.TOP	左タッチセンサーの上オン
LogicOR		論理和
ReturnWithValue		値を伴ってメインへ戻る

なお, 論理和 LogicOR の代わりに論理積 LogicAND にすると, 2 つのタッチセンサーが同時にオンになった場合に, 0 (負論理の真) の値を返すプログラムとなる。

以上, サブルーティンの記述法について示してきた。このサブルーティンはタイル言語でのタイルに相当する。

4.2 メインプログラム

図 4.1 に示すメインプログラム格納領域の命令は、表 2.2 に示すほとんどのものが利用できる。これらの命令を利用して、プログラムを作成すれば以下ようになる。

センサーからの入力によって前進したり、回転したりするプログラム：

Mainroutine		メインルーティン
CallSub	initialize	初期化サブルーティンへ
loopback		
CallSubAndTest	ultrasonicget,next01	超音波センサー入力へ
CallSubAndTest	opticalgetleft,next02	光センサー入力(左)へ
CallSubAndTest	opticalgetright,next03	光センサー入力(右)へ
CallSubAndTest	touchgettopR,next04	タッチセンサー入力(右上)へ
CallSubAndTest	touchgetleftR,next04	タッチセンサー入力(右左)へ
CallSubAndTest	touchgetrightR,next04	タッチセンサー入力(右右)へ
CallSubAndTest	touchgetbottomR,next04	タッチセンサー入力(右下)へ
CallSubAndTest	touchgettopL,next04	タッチセンサー入力(左上)へ
CallSubAndTest	touchgetleftL,next04	タッチセンサー入力(左左)へ
CallSubAndTest	touchgetrightL,next04	タッチセンサー入力(左右)へ
CallSubAndTest	touchgetbottomL,next04	タッチセンサー入力(左下)へ
CallSub	forward	前進サブルーティンへ
Jump	loopback	
next01		
CallSub	backward	後退サブルーティンへ
Jump	loopback	
next02		
CallSub	rotateleft	左回転サブルーティンへ
Jump	loopback	
next03		
CallSub	stop	停止サブルーティンへ
Jump	loopback	
next04		
CallSub	rotateright	右回転サブルーティンへ
Jump	loopback	
Execute		プログラム実行
end		

ここで、サブルーティンから値を伴う戻り ReturnWithValue は、真が値 0、偽が値 1 となっている。いわゆる負論理である。このため、サブルーティンコール CallSubAndTest sub,jp2[,jp1] は真の値が戻ってくるときに jp2 にジャンプすることになる。なお、サブルーティンから正論理で値を戻すことも可能であるが、この場合 jp1 にジャンプするようにしなければならない。

有限回のループを実行し、停止するプログラム：

Mainroutine		メインルーティンの開始
CallSub	initialize	初期設定へ
SetCounter	0,5	カウンタ 0 に値 5 をセット
loopback		
CallSub	forward	前進のサブルーティン
CallSub	backward	後退のサブルーティン
DecCounter	0,stop,loopback	カウンタ 0 を 1 減じ, 0 であれば Stop へ
stop		
Stop		
Execute		
end		

割り込みを利用したプログラム：

まず、割り込み処理の流れを示すと、図 4.2 に示すように、サブルーティンコール CallSub SubA,\$ff,jp2 を行う際、割り込みを受け付けるメインルーティンのアドレス int (jp2) を設定する。ここで、\$ff は図 1.8 に示すビット構成であり、割り込み受付ブロックを指定する。通常すべてのブロックからの割り込みを受け付けるように\$ff とする。このような割り込みを利用するプログラム例は以下のようなになる。

Mainroutine		メインルーティン
CallSub	initialize	初期化サブルーティンへ
loopback		
CallSub	forward,\$ff,int	割り込み設定と前進へ
Jump	loopback	
*		
int		
CallSubAndTest	ultrasonicget,next01	超音波センサー入力へ
CallSubAndTest	opticalgetleft,next02	光センサー入力(左)へ
CallSubAndTest	opticalgetright,next03	光センサー入力(右)へ
CallSubAndTest	touchgettopR,next04	タッチセンサー入力(右上)へ
CallSubAndTest	touchgetleftR,next04	タッチセンサー入力(右左)へ
CallSubAndTest	touchgetrightR,next04	タッチセンサー入力(右右)へ
CallSubAndTest	touchgetbottomR,next04	タッチセンサー入力(右下)へ
CallSubAndTest	touchgettopL,next04	タッチセンサー入力(左上)へ
CallSubAndTest	touchgetleftL,next04	タッチセンサー入力(左左)へ
CallSubAndTest	touchgetrightL,next04	タッチセンサー入力(左右)へ
CallSubAndTest	touchgetbottomL,next04	タッチセンサー入力(左下)へ
Jump	loopback	
next01		
CallSub	backward	後退サブルーティンへ
Jump	int	
next02		
CallSub	rotateleft	左回転サブルーティンへ
Jump	int	

```

next03
  CallSub      stop          停止サブルーティンへ
  Jump        int
next04
  CallSub      rotateright   右回転サブルーティンへ
  Jump        int
Execute
end

```

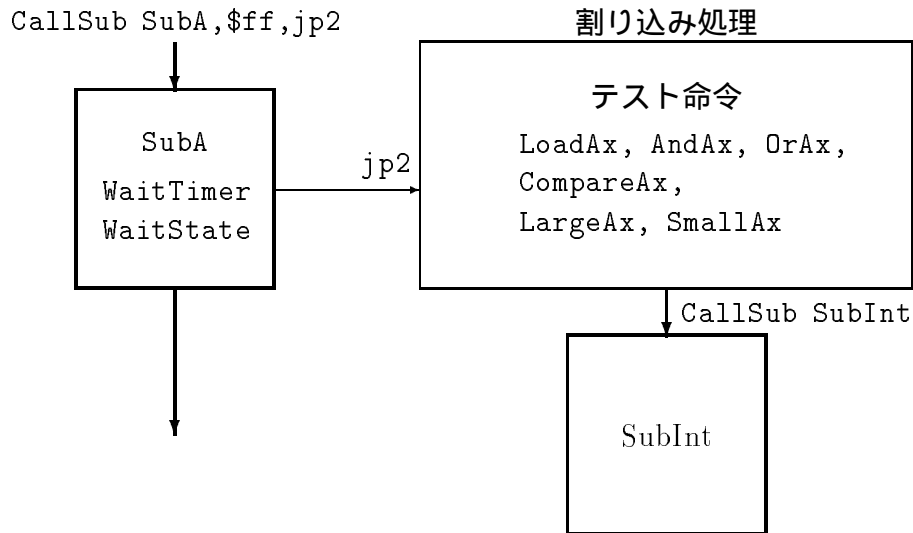


図 4.2 割り込み処理の流れ

ここで、割り込みを受け付ける命令 `CallSub forward,$ff,int` から呼び出すサブルーティン `forward` には `WaitTimer` または `WaitStatus` 命令が含まれていなければならない (図 4.2 参照)。

以上、これらを駆使してメインプログラムを記述すれば、種々のセンサーからの信号入力によって、種々の動作 (複雑な動作) をさせることが可能となる。例えば、トレースプログラム、迷路脱出プログラムなどを作ってみるのもよいと思う。

4.3 プログラミングの注意点

ROBOCUBE 用アセンブリ言語とそのコンパイラの開発を行い、動作プログラムを作成すると、仕様書に記載されていないいくつかの問題点 (隠れた仕様) が明らかとなった。これらについて以下に記載する。

モータ制御における問題点

第 2 章における前進のプログラム（サブルーティン）は以下のようになっている。

```

25                                     forward
26 0011 *15000111070600000012D9_   Motor    0,MOVE_CW
27 0013 *15000113070700000114DF_   Motor    1,MOVE_CCW
28 0014 *1500011407000000FF1505_   SendStrobe
29 0015 *15000115080003E80016FC_   WaitTimer 1000
30 0016 *15000116160000000000D5_   Return

```

このプログラムにおいて、モータブロックが向かい合って結合されているので、モータ 1 を前進、モータ 2 を後退にすることによって、全体として前進することになる。第 2 章に示すアセンブリ言語でプログラミングを行う場合、注意する必要がある。また、SendStrobe 命令を入れてあるが、これを取り除くと動作しなくなる。すなわち、2 つのモータを同時に動作させる命令である。しかし、この命令で生成されるパケット構成は仕様書にないものである。モータ制御を行う場合、この SendStrobe 命令を入れるようにする。

ブロックからの状態取り込み命令の問題

第 2 章における光センサーの状態取り込みプログラム（サブルーティン）は以下のようになっている。

```

128                                     optical1
129 0060 *150001600A0200060061ED_   LoadAx  OPTICAL,0,2
130 0061 *150001610C0000000062E9_   PopOneData
131 0062 *150001620F00000006363F7_   TestAx   0
132 0063 *150001630C0000000064ED_   PopOneData
133 0064 *15000164260000000065E4_   NotAx    0
134 0065 *150001650CFF000000661D_   PopOneData
135 0066 *15000166160000000000DA_   Return

```

このプログラムにおいて、LoadAx 命令は第 2 章に示すとおり（仕様書のとおり）、ブロックの状態を Ax レジスタに取り込む命令である。従って、この命令の後の PopOneData を行う必要はないはずである。しかし、この命令を取り除くと動作しなくなる。LoadAx 命令はブロックの状態を取り込みスタックに積む動作をしているものと考えられる。

論理否定の問題

上の光センサーの状態取り込みプログラム（サブルーティン）において、NotAx 命令の it 領域が 00 以外のとき、正常に動作しない。そこで、正論理で値を戻すため、次のようにもう一回論理否定を行う。

```

131 005B *1500015B0C000000005C09_   PopOneData
132 005C *1500015C2600000005D5D19_   NotAx
133 005D *1500015D0C000000005E0D_   PopOneData
134 005E *1500015E26000000005F1F_   NotAx      0
135 005F *1500015F0C0000000060FB_   PopOneData
136 0060 *15000160160000000000D4_   Return

```

この場合、最初の NotAx 命令の it 領域には j1 領域と同じ値が入っていても同じように動作する。さらに、NotAx を 2 回行う場合、PopOneData 命令を実行する必要がないように思われるが、これを取り除くとやはり動作しなくなる。すなわち、論理命令 AndAx, OrAx, CompareAx, LargeAx, SmallAx, テスト命令 TestAx などの命令を実行する場合、演算結果をレジスタに格納するのではなく、スタックに積む動作を行っているとは推察される。従って、これらの演算命令を実行する場合、スタックからレジスタに値を取り込む必要があるといえる。この考えは、一般のマイクロプロセッサの動作と異なるので、注意が必要である。

論理和および論理積の問題

第 2 章に示す論理積 LogicAND または論理和 LogicOR の動作では、即値 (dt=00 の場合) と論理積または論理和を行って、その真偽をスタックに積むとなっている (仕様書)。しかし、表 3.2 に示すように、論理積 LogicAND および論理和 LogicOR の命令は、PopTwoData 命令を実行した後に行われるようになっている。いわゆる、即値が 0 である場合、レジスタ AX と BX の論理積または論理和を行って、その真偽をスタックに積む動作となっている。

4.4 アセンブラの実行

表 2.2 および表 2.3, 表 3.1 および表 3.2 に示す命令をコンパイルする 8 ブロック用アセンブラ (バージョン 2.05, ファイル名 cubeasm5.c), およびバージョンアップセット用アセンブラ (バージョン 3.01, ファイル名 cubeasm31.c) を C 言語で作成した。そして、Windows 下で動作し、ROBOCUBE と接続するシリアルポートは COM1 に設定してある。このアセンブラの実行形式 (ファイル名 cubeasm5.exe および cubeasm31.exe) をユーザに提供する。このアセンブラを実行するとコマンドプロンプトウィンドウが開き、file name = を出力するので、ROBOCUBE のアセンブリ言語プログラム (第 3 章のソースプログラム) が格納されているファイル名を入力する。このとき、このアセンブリ言語プログラムをコンパイルするとともにパケットを生成して ROBOCUBE へ転送する。そして、ファイル名 assemble.lst にコンパイル結果を、ファイル名 packet.lst に ROBOCUBE との間でのパケット交換結果を格納する。

4.5 まとめ

本章では、ROBOCUBEの自走プログラムを作成するためのアセンブリ言語の開発とその記述法を示してきた。ここで開発したアセンブリ言語は、ソースプログラムを作成するだけでよい。また、プログラムの変更が容易である、大きな自走プログラムの作成ができる、複雑な動作を行う自走プログラムの作成が可能である、など利点が多い。これによって、工業高校、専門学校や大学学部等（高学年用）において、ROBOCUBEを利用した学生実験が可能になったと思う。すなわち、冒頭にも述べたが、現在のマイクロプロセッサの機能が高く、アセンブリ言語によるプログラミングが事実上不可能である。ここで示したROBOCUBE用アセンブリ言語は、命令数も少なく分かり易い。さらに、ロボットを制御するので、学生にとって興味ある実験テーマとなることが明らかである。また同時に、ROBOCUBEは機能分散型ロボットであるため、共通バス方式のネットワーク接続やパケット交換方式の概念をも学ぶことができるという利点がある。

また、アセンブリ言語でプログラミングを行うことは、ハードウェアの動作を十分理解しておかなければできない。その意味において、専門学科においてはアセンブリ言語によるプログラミングの必要がある。従って、本書で示したROBOCUBE用アセンブリ言語が学生実験用として利用されれば幸いである。

最後に、本書の内容等の利用については、下記の許可を得ていただきたい。

開発者（著者）： 吉岡 良雄 （弘前大学理工学部電子情報システム工学科）

e-mail slyoshi@si.hirosaki-u.ac.jp

権利者： （株）システムワット

e-mail robocube@watt.co.jp