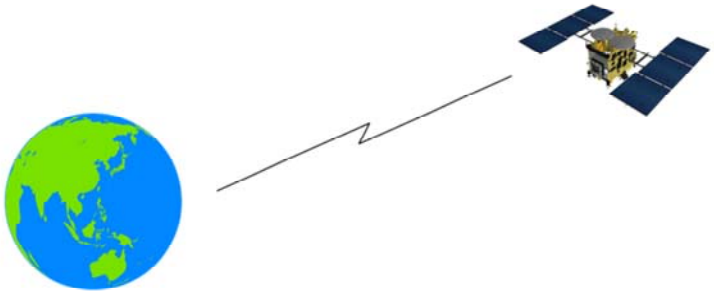
ソフトウェア作成の課題 4
Sレコード変換

v4.1

「はやぶさ」へのプログラム送信、データの受信にはSレコードが使われている

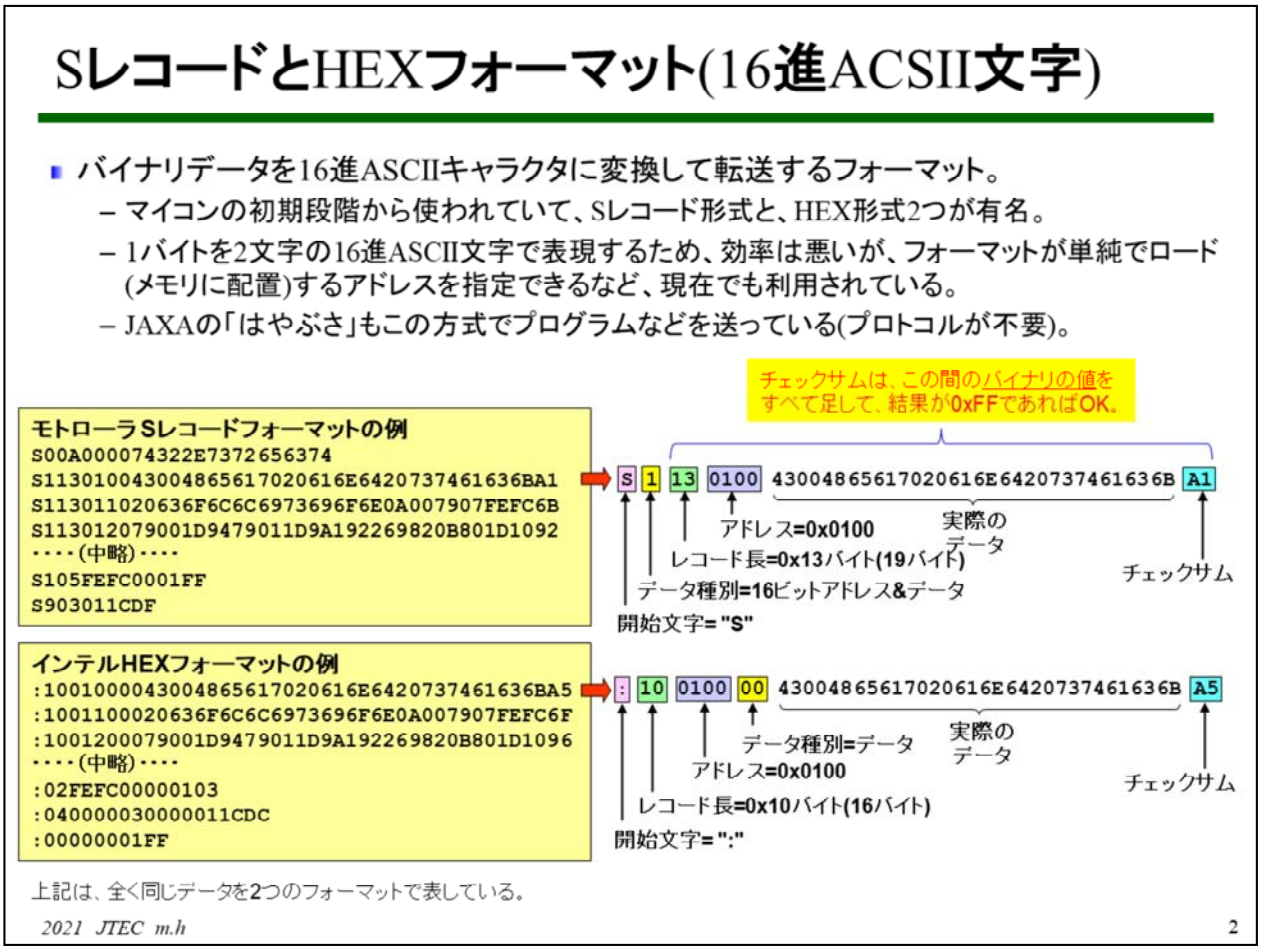


2021 JTEC m.h 1

C言語研修課題の5題目は、Sレコード変換です。

レコードとは、旧モトローラ社(現在はフリースケール・セミコンダクタ社)が開発した、バイナリデータを16進ASCIIキャラクタに変換して転送するフォーマットのことです。
同様のフォーマットとして、インテルのHEXフォーマットがありますが、方式としてはほとんど同じです。

Sレコードも、HEXフォーマットも、マイコンが登場した当時からあるものです。歴史は古いですが、現在でも、JAXAで「はやぶさ」とのプログラムやデータの送るのに使われています。



モトローラのSレコード形式とインテルHEXフォーマットは、バイナリデータを16進ASCII文字に変換して転送する方式で、マイコンの初期の段階から使われていました。ACSII文字を使用していて、7ビットのASCIIコードで転送できるのが特徴です。

1バイトを2文字の16進ASCII文字で表現するため、効率は悪いですが、フォーマットが単純でロード(メモリに配置)するアドレスを細かく指定できます。

プロトコル(通信相手とのやり取り)なしで、単純にシリアルでデータを転送できるため、USBやネットワークを持たないマイコンボードやROMライタにデータを送るときにも利用されています。

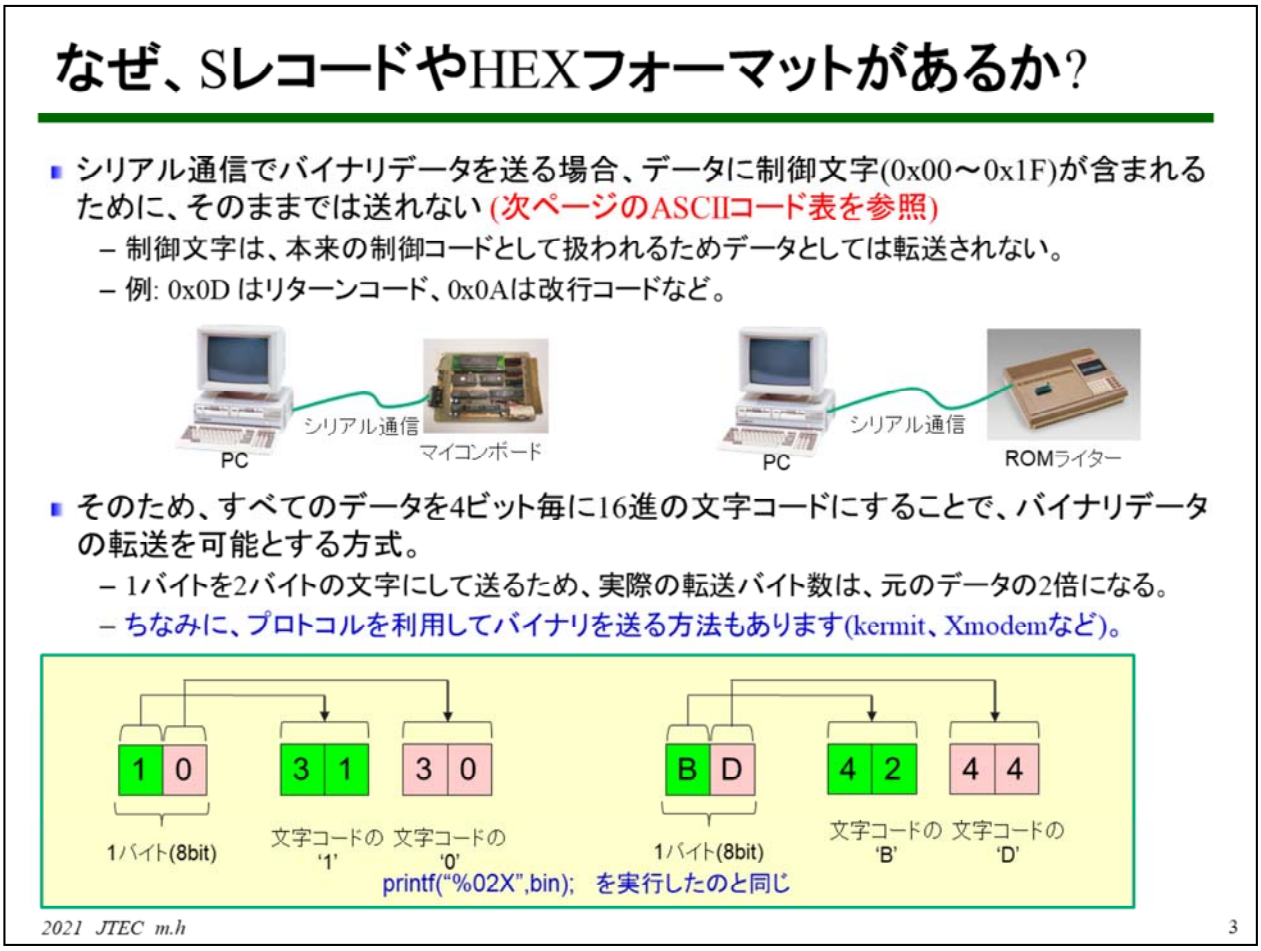
サンプルリターンに成功したJAXAの「はやぶさ」、「はやぶさ2」も、この方式でプログラムやデータを送信しています。小惑星の「イトカワ」や「リュウグウ」までは、電波の速度でも片道約20分もかかるので、当然プロトコルなどは使えません。

モトローラのSレコード方式もインテルのHEX方式も、フォーマットは単純で、開始コード、データの種別、レコード長、ロード開始アドレス、チェックサムなどから構成されています。

モトローラのSレコードは、“S”で始まり、「データ種別」、「レコード長」、「アドレス」、「データ本体」、「チェックサム」と続きます。

インテルのHEXレコードは、“:”で始まり、「レコード長」、「アドレス」、「データ種別」、「データ本体」、「チェックサム」と続きます。

C言語の課題4 (Sレコード変換のの作成)



なぜ、SレコードやHEXフォーマットがあるかというと、シリアル通信でバイナリデータを送る場合、データに制御文字(0x00～0x1F)が含まれているためです。

バイナリデータには、0x00～0xFFのすべての数値が含まれています。しかし、シリアル通信では、制御文字(0x00～0x1F)は、本来の制御コードとして扱われてしまいます。例えば、0x0D はリターンコード、0x0Aは改行コードなどがそうです。

そのため、すべてのデータを4ビット毎に16進数の文字コードにすることで、バイナリデータの転送を可能にしているのです。1バイトを2バイトの文字にして送るため、実際の転送バイト数は、元のデータの2倍になります。

例えば、1バイトの 0x10 のデータを送る場合、printf("02X", 0x10) で出力したのと同じです。

つまり、0x10の上位の4ビットの1は、文字の'1'(0x31)に、下位4ビットの0を'0'(0x30)にして出力します。

printf("02X") を使えば、簡単にバイナリをASCII文字コードに変換できます。今回は、使用しませんが、これをプログラムでやると下記のようになります(例では、bindata に変換するバイナリの値が入っている)。

```
char binH, binL;

binH = ((bindata & 0xF0) >> 4) + '0';  // 上位4ビットの取り出し、'0'(0x30)を足して文字コードにする
binL = (bindata & 0x0F) + '0';         // 下位4ビットの取り出し、'0'(0x30)を足して文字コードにする

if (binH > '9')                        // binHが '9'(0x39)より大きければ
    binH += ('A' - ':');               // 'A'と'9'の次の文字(':')との差分の7を加える

if (binL > '9')                        // binLが '9'(0x39)より大きければ
    binL += ('A' - ':');               // 'A'と'9'の次の文字(':')との差分の7を加える

printf("%c%c",binH, binL);             // binHとbinLを文字として出力
```

8ビット(8単位)ASCII文字コード表

	上位	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
下位		0x	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx
0000	x0	NUL	DLE	SP	0	@	P	`	p				ー	タ	ミ		
0001	x1	SOH	DC1	!	1	A	Q	a	q			。	ア	チ	ム		
0010	x2	STX	DC2	"	2	B	R	b	r			「	イ	ツ	メ		
0011	x3	ETX	DC3	#	3	C	S	c	s			」	ウ	テ	モ		
0100	x4	EOT	DC4	$	4	D	T	d	t			。	エ	ト	ヤ		
0101	x5	ENQ	NAK	%	5	E	U	e	u			・	オ	ナ	ユ		
0110	x6	ACK	SYN	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
0111	x7	BEL	ETB	'	7	G	W	g	w			ァ	キ	ヌ	ラ		
1000	x8	BS	CAN	(	8	H	X	h	x			ィ	ク	ネ	リ		
1001	x9	HT	EM	)	9	I	Y	i	y			ゥ	ケ	ノ	ル		
1010	xA	LF	EOF	*	:	J	Z	j	z			ェ	コ	ハ	レ		
1011	xB	VT	ESC	+	;	K	[	k	{			ォ	サ	ヒ	ロ		
1100	xC	FF	FS	,	<	L	¥	l	|			ャ	シ	フ	ワ		
1101	xD	CR	GS	-	=	M	]	m	}			ュ	ス	ヘ	ン		
1110	xE	SO	RS	.	>	N	^	n	~			ョ	セ	ホ	゛		
1111	xF	SI	US	/	?	O	_	o	DEL			ッ	ソ	マ	゜		

シリアル通信では、この部分のコードを、
文字として転送できない

Shift-JISコードの第1バイト目のコード
0x81～0x9F || 0xE0～0xEF

2021 JTEC m.h 4

この表は、ASCIIコード表です。最初のASCIIコードは、7ビットだったため左側半分だけでした。その後、日本で半角カタカナを追加して右側半分が追加されました。色々なところで登場する表です。
数字の'0'が0x30、英文字の'A'が0x41 程度は覚えておきましょう。

この表で、0番台、10番台のコード(灰色の2列)は文字コードでなく、制御コードとして定義されています。
これは、電話回線などで文字を送るときに使われるものです。
例えば、
0x02のSTXは、Start of TeXtで、これから文字を送りますとい意味です。
0x03のETXは、End of TeXtで、これで文字を終わりますという意味です。
というように使われています。

また、現在のPCでも使っている制御コードも多くあります。
[Enter]キーは、0x0AのLFコードです。[Back Space]キーは、0x08のBSコードです。
[Delete]キーは、0x7EのDELコードです。
その他、0x1BのESC(Escape)も使われています。
ちなみに、0x08はBEL(ベル)で、今でも、printf("%c",0x07); を実行するとPCから音がでます。

Sレコードからバイナリにするコマンド
srec2bin (仮称)

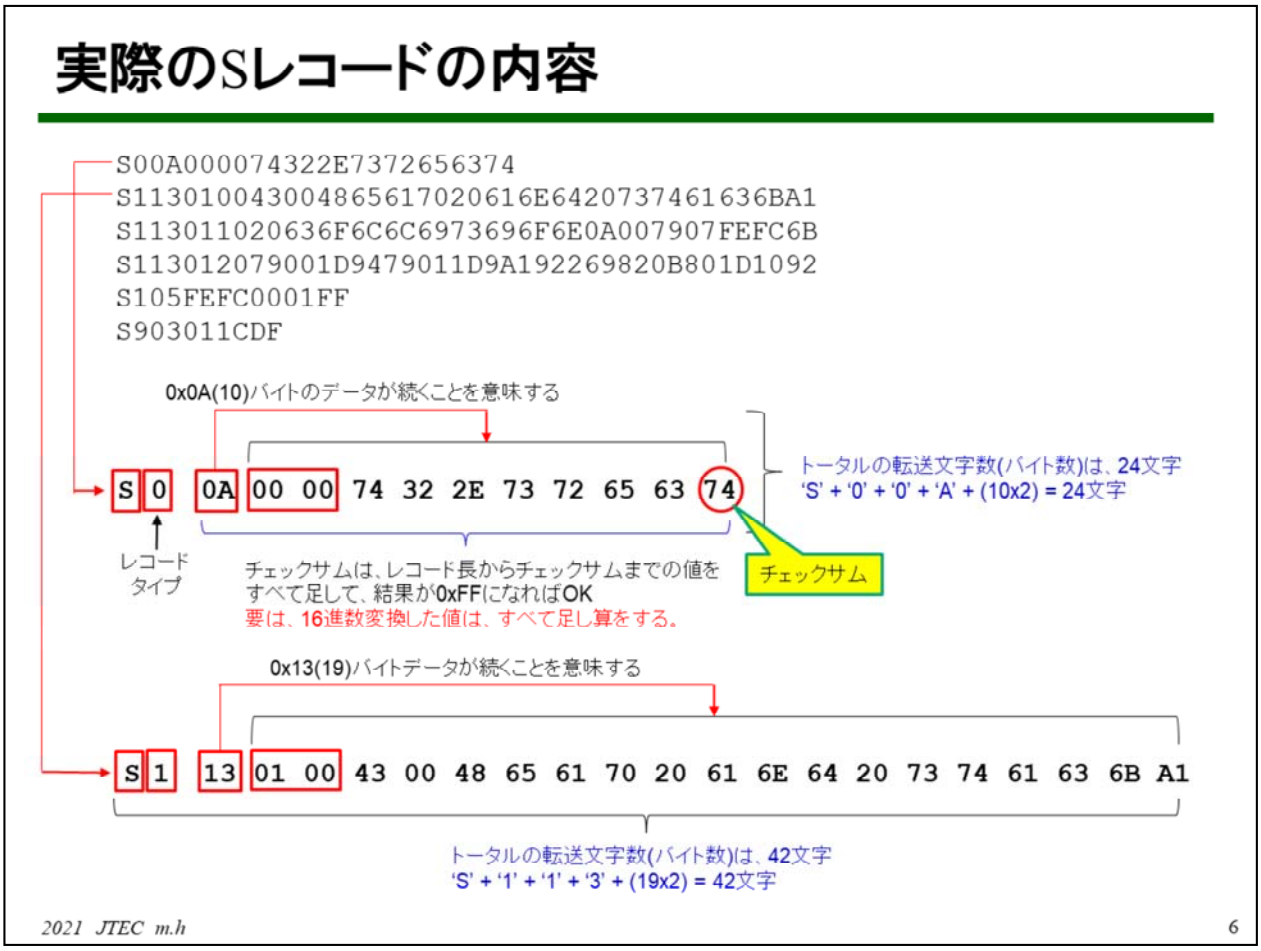
2021 JTEC m.h 5

Sレコードは、Sレコードからバイナリにするコマンドと、その逆にバイナリからSレコードを変換する2つがあります。両社はペアとして使います。

課題3でdump関数とdumpコマンドをやりました。dumpでは、”%02X”で16進数表示をしました。「バイナリからSレコード」は、dumpと同じように、”%02X”でSレコードを出力します。

まず、最初に、「Sレコードからバイナリ」にする srec2bin コマンドを作成します。コマンドですから、コマンド引数の解析は、dumpコマンドで作成した関数をコピーして使ってください。過去に作成したソースコードの再利用です。

また、「Sレコードからバイナリ」では、2文字の16進ASCII文字をバイナリに変換する関数を作成します。前々ページに示した、バイナリから16進文字変換の逆をすることになります。実際の変換方法は、次々ページに示します。



これは、実際のSレコードの内容を示したものです。一見、16進文字列の羅列で難しそうに思えますが、ひとつひとつを順番に紐解いていくと決して難しいものではありません。

まず、Sレコードは単なるテキストです。メモ帳でも開いてみることができます。そして、1行が1レコードとなります。

Sレコードですから、最初の文字は'S'で始まります。次にレコードタイプが1文字続きます。この最初の2文字はのみ文字として扱い、以後はすべて16進文字からバイナリに変換をしながら進めていきます。

この例で、最初のレコードは、「S00A000074322E7372656374」です。

これを、フィールドごとに分けてみると下記のようになります。

S　0　0A　0000　74322E73726563　74

①　②　③　④　⑤　⑥

① Sレコードの開始文字です。'S' で始まります。

② レコードのタイプです。0～9まで定義されていますが、レコードのタイプも文字です。
①と②は、唯一文字です。16進文字列からバイナリ変換はしません。

③ レコード長です。このあとに続くバイト数を意味します。ここからは、チェックサムまで、すべて16進文字列からバイナリ変換とチェックサムの計算が必要になります。

④ アドレス長です。アドレスのバイト数で、レコードタイプによって、2バイト、3バイト、4バイトがあります。

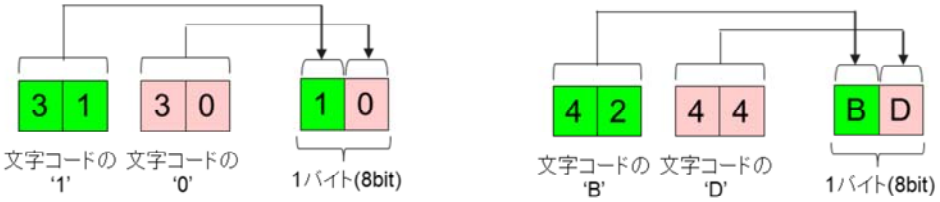
⑤ データです。データ長は、レコード長からアドレス長とチェックサムの1バイトを引いて求めます。
実際は、「データ長 = レコード長 -(アドレス長 + 1(チェックサム分))」で求まります。
その結果、データ長が 0 になることもあります。0はデータがないことを意味します。

⑥ チェックサムです。バイナリに変換した、③のレコード長から、⑥のチェックサムまで、単純に符合なしで加算していきます。その結果、値が0xFFになれば正常で、0xFFにならなければ途中でエラーがあったことを意味します。

16進文字列からバイナリの変換

- Sレコードの文字列からバイナリーの変換は、下記のように行う。
- 2バイトの文字コードから、1バイトのバイナリを作る。
  - 16進変換を方法を理解するために、strtol()関数は使わずに、自分で変換関数を作成する。

- 文字コードから'0' (0x30)を引く
  - 引いた結果が、10より小さければ、0～9の範囲
  - 引いた結果が、10以上であれば、さらに7('A' – ':') を引く



変換のヒント

2文字を、上位4ビット(1文字目)、下位4ビット(2文字目)の2回に分けて変換。
最後に、「上位4ビット << 4 | 下位4ビット」を実行して1バイトのデータにする。

2021 JTEC m.h 7

16進文字列からバイナリに変換するには、dumpコマンドのオフセットを取得するときに使った strtol() 関数があります。しかし、strtol() は、Sレコードの変換に使うのは適切ではありません。

なぜなら、strtol() は、16進文字以外が来るまで変換をします。Sレコートはすべてが16進文字列なので、strtol() を使うには、2文字ずつ取り出して変換をする必要があり、かえって面倒になります。

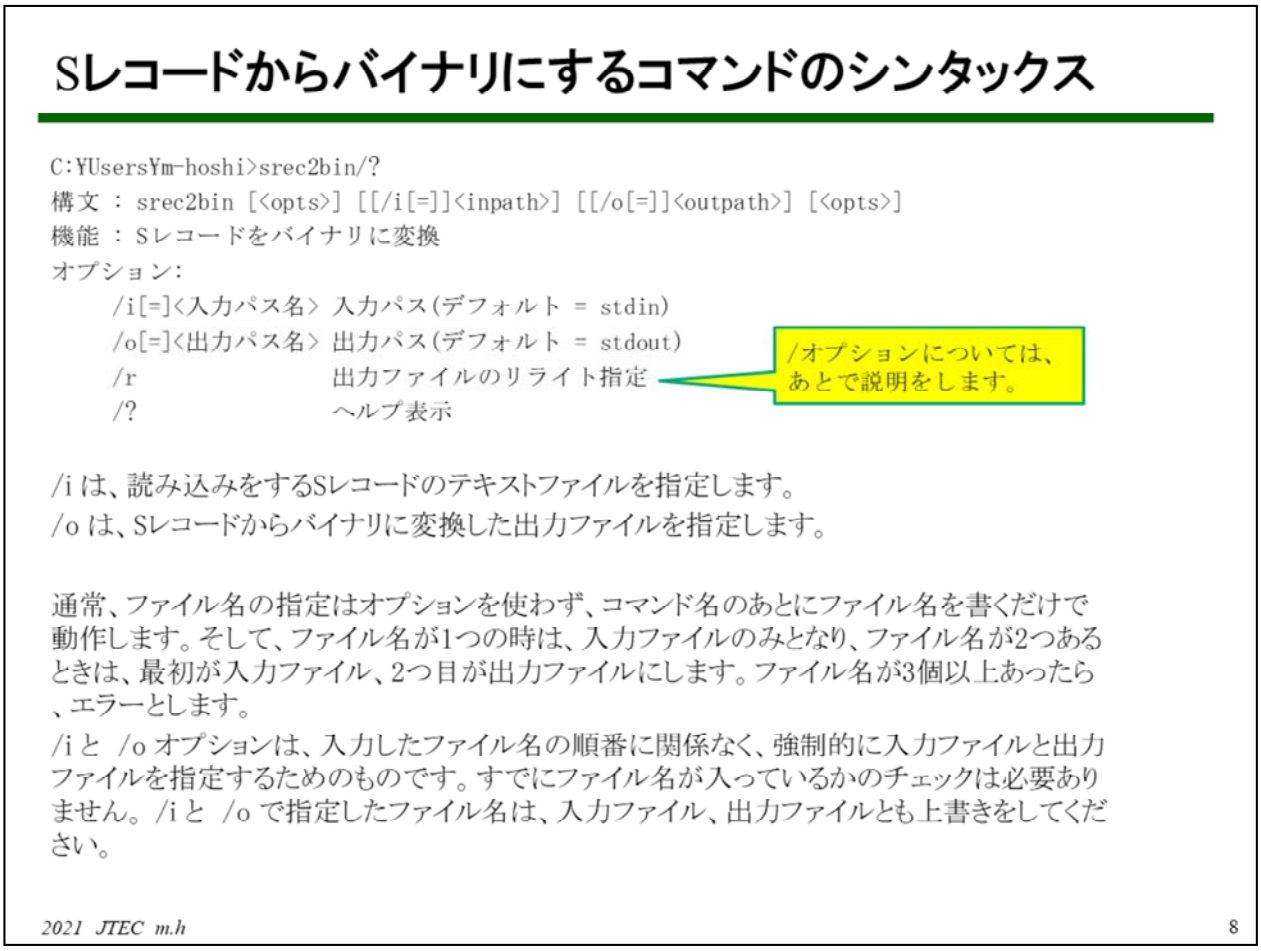
それで、今回、16進変換を方法を理解するためにも、自分で変換関数を作成します。

下記のような構成になります。

```
unsigned char hex2bin(char *hexstr){
    int binH;
    int binL;

    ここで、
    1文字目を変換して、binHに格納
    2文字目を変換して、binLに格納

    return (binH<<4 ) | binL;    ← 変換されたバイナリを返す
}
```



srec2binコマンドのヘルプメッセージの例です。
この中で、/i オプションと、/oオプションのついて詳しく説明をします。

/i オプションは、読み込みをするSレコードのテキストファイルを指定します。
/o オプションは、Sレコードからバイナリに変換した出力ファイルを指定します。

通常、ファイル名の指定はオプションを使わず、コマンド名のあとにファイル名を書くだけで動作します。そして、ファイル名が1つの時は、入力ファイルのみとなり、ファイル名が2つあるときは、最初が入力ファイル、2つ目が出力ファイルにします。ファイル名が3個以上あったら、エラーとします。

srec2bin txt-s1-record.txt ← ファイル名の1つ目は、入力ファイルにする。
コマンド名 入力ファイル名

srec2bin txt-s1-record.txt s1_record.txt ← 2つ目のファイル名は、出力ファイルにする。
コマンド名 入力ファイル名 出力ファイル名

/i と /o オプションは、入力したファイル名の順番に関係なく、強制的に入力ファイルと出力ファイルを指定するためのものです。すでにファイル名が入っているかのチェックは必要ありません。/i と /o で指定したファイル名は、入力ファイル、出力ファイルとも上書きをしてください。

srec2bin /o=s1_record.txt /i=txt-s1-record.txt ←順番に関係なく入力、出力ファイルを指定。
コマンド名 出力ファイル名 入力ファイル名

Sレコード変換プログラム作成のヒント

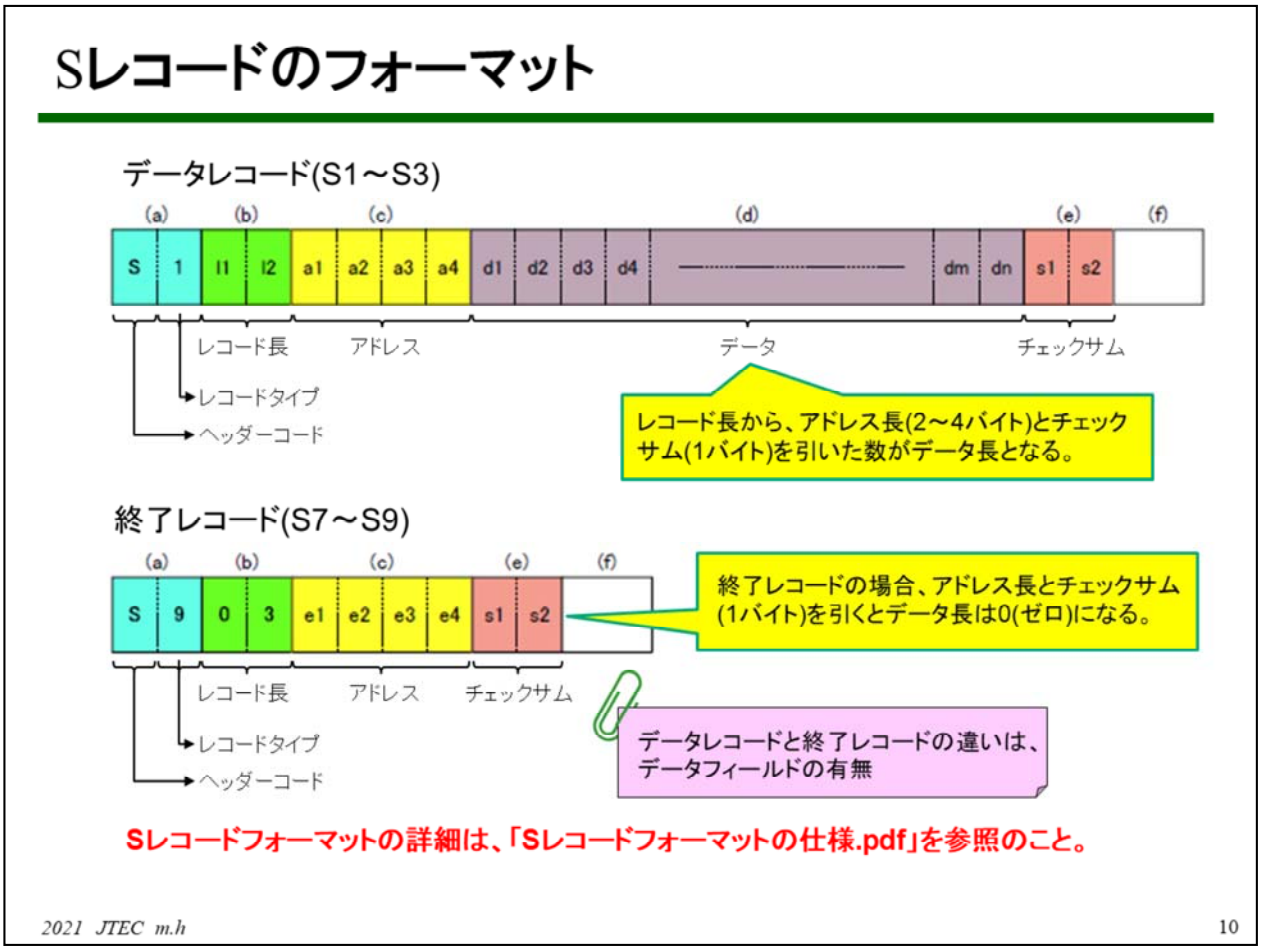
2021 JTEC m.h 9

ここで、Sレコード変換プログラム作成のヒントとして、Sレコードのフォーマットをおさらいしておきます。

このフォーマットを正しく理解すれば、Sレコード変換は決して難しいものではありません。

課題3のdumpコマンドでは、main関数で、while( )の中で、データを読込み、dump関数を呼んでいました。まったく、同じ構成になります。

ただ、Sレコードからバイナリ変換は、テキストファイルの読み込みになります。テキストデータの読み込みは、課題1のライフゲームでも使いました。



srec2bin(Sレコードからバイナリ変換)は、main( )で読込んだテキストの1行を、そのまま引数(char型のポインタ、あるいは1次元配列)として受け取り、テキストの最初から順番に調べていきます。

このとき、テキストの文字の位置を指し示す変数(例えば、int bytecnt)が必要になります。最初に bytecnt は 0 に初期化しておきます。以後、処理をするたびに、bytecntを、増加していきます、

char型ポインタは、1次元配列と見なすことができます。それで、Sレコードのテキストをアクセスするには、配列としてアクセスする場合と、ポインタとしてアクセスする方法の二通りがあります。

例えば、Sレコードのテキストを、char *srec_str で受け取った場合、

srec_str[bytecnt] ← 1次元配列としてアクセス

*(srec_str + bytecnt) ← ポインタでアクセス

C言語の基礎として、上記の2つのアクセスの結果が同じであるたことが分かりますか?

実際の処理ですが、最初の文字は、"S"(bytecntは0)で、次に続く文字(bytecntは1)がレコードタイプになります。この最初の2バイトだけは文字として扱います。そして、このレコードタイプは、ローカル変数(例えば、char stype)に保存しておきます。

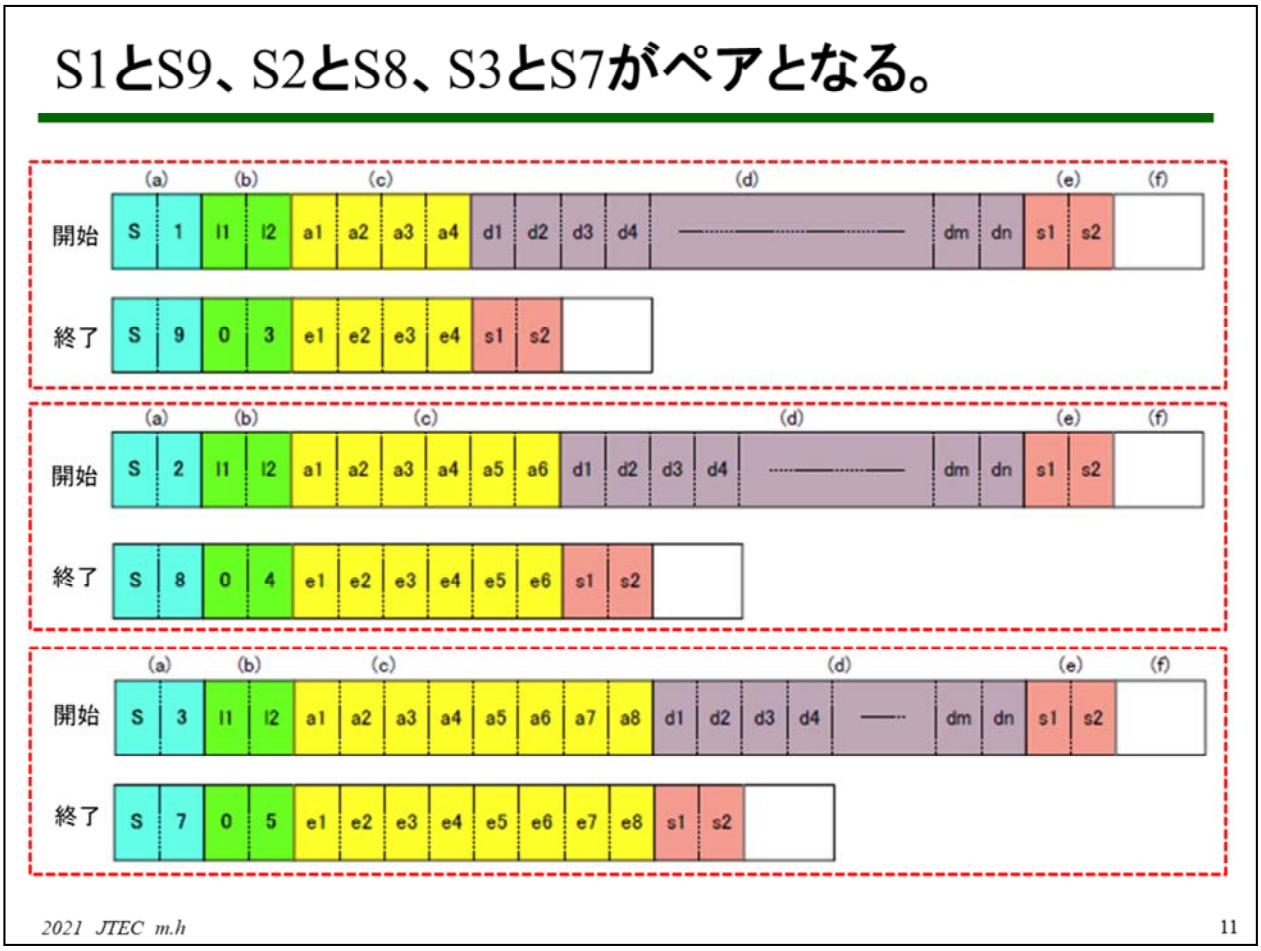
次の文字になる、レコード長(bytecntは2)からは、すべて2バイトずつ、バイナリに変換します。そして、変換が終わったら、に、bytecntを、+2ずつ増加します。

recsize = hex2bin(srec_str + bytecnt); あるいは、 recsize = hex2bin(&srec_str[bytecnt]);

と変換したあと、

bytecnt +=2; ← テキストの位置を更新

chksum += recsize; ← チェックサムに加算



レコード長が取得できたら、次は、switch-case文で、最初に保存したレコードタイプに応じて、アドレス長を設定していきます。レコードタイプが、

'0'、'1'、'9'の場合、アドレス長は2バイト(16ビット)

'2'、'8'の場合、アドレス長は3バイト(24ビット)

'3'、'7' の場合、アドレス長は4バイト(32ビット)

になります。

この段階で、bytecntは、アドレスデータの先頭になっていますので、アドレス長の分だけ、for文でループをして、アドレスを変換していきます、ここで、厳密には、アドレス長に応じて、アドレスを生成するのですが、研修では、アドレスは使用しないので生成は必要ありません。ただ、チェックサムの計算があるので、変換してチェックサムに加算する処理は必要になります。

アドレス部の処理が終わる、bytecnt は、データの先頭になっています。Sレコードのレコードタイプの違いは、アドレス長だけです。アドレスの変換が終われば、どのレコードタイプてあっても、bytecntすべてデータの先頭になっています。

ここで、レコード長からアドレス長とチェックサムの1バイトを引くと、データ長が求まります。データ長が求まれば、for文でループをし、変換したデータを出力していきます。

データ部の処理が終わると、bytecnt は、チェックサムの位置になっています。

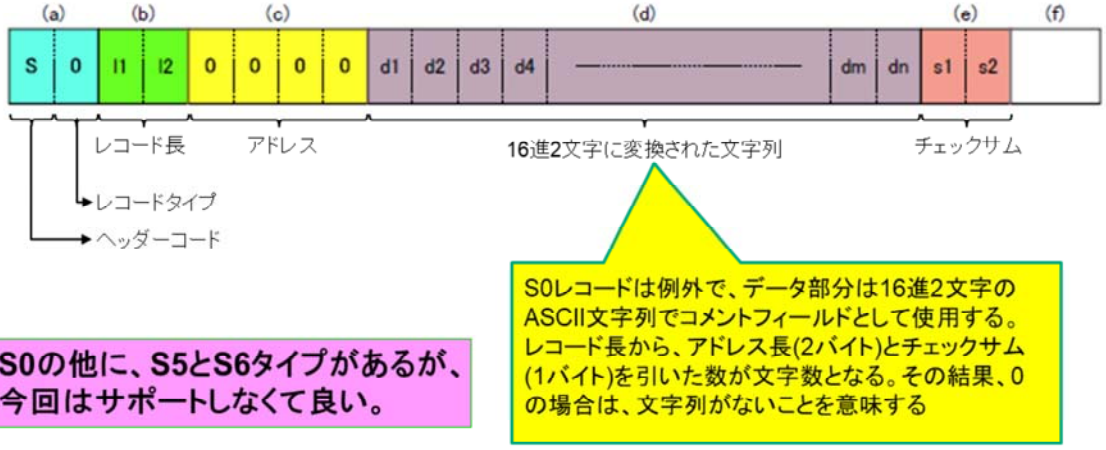
最後に、チェックサムを変換し、チェックサムに足したら結果が、0xFF になれば正解です。もし、0xFFにならなければチェックサムエラーとします。

この時、行番号をカウントしておいて、行番号と読み取ったSレコードの文字列を表示させると親切です。

正常なデータで、0xFF にならない場合は、とこかで計算が間違っています。

S0のヘッダレコード

S0レコードは、ヘッダレコードで、データ部分は、ASCII文字列となっている。
文字列を含むかどうかは、任意である。



S0の他に、S5とS6タイプがあるが、
今回はサポートしなくて良い。

S0レコードは例外で、データ部分は16進2文字の
ASCII文字列でコメントフィールドとして使用する。
レコード長から、アドレス長(2バイト)とチェックサム
(1バイト)を引いた数が文字数となる。その結果、0
の場合は、文字列がないことを意味する

2021 JTEC m.h 12

レコードタイプのS0は、ヘッダレコードとして処理します。アドレス長は、2バイトなのでS1、S9と同じ処理になります。

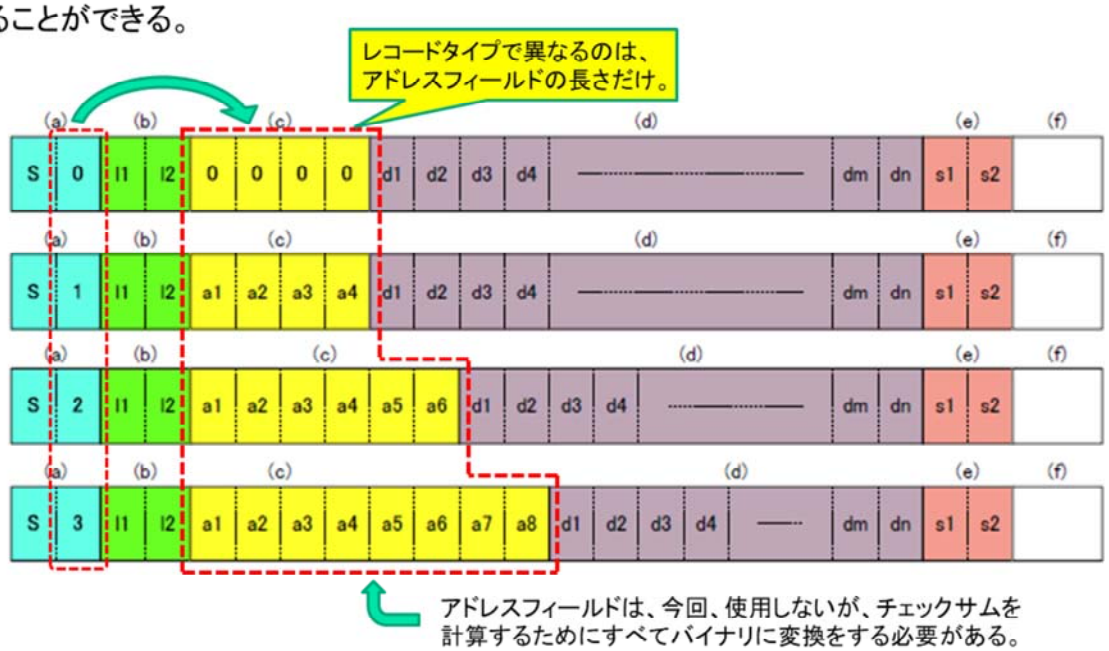
ただし、ヘッダレコードにあるデータ部は、転送するデータではなく、文字列が入っているタイトルになります。そのため、変換したあと、データとして出力してはいけません。

S0には、データがない場合が多いですか、もしあったら場合は、stderr(標準エラー)に出力します。

Sレコードのファイルが複数ある場合、送信側はファイル名で区別ができますが、受信側では分かりません。それで、S0 レコードの中に、文字を入れれば受信側でも判定ができます。タイトルとして入れる文字は自由です。ファイル名やバージョン番号であったりします。要は、S0レコードとは、タイトルを入れた特別なレコードということになります。

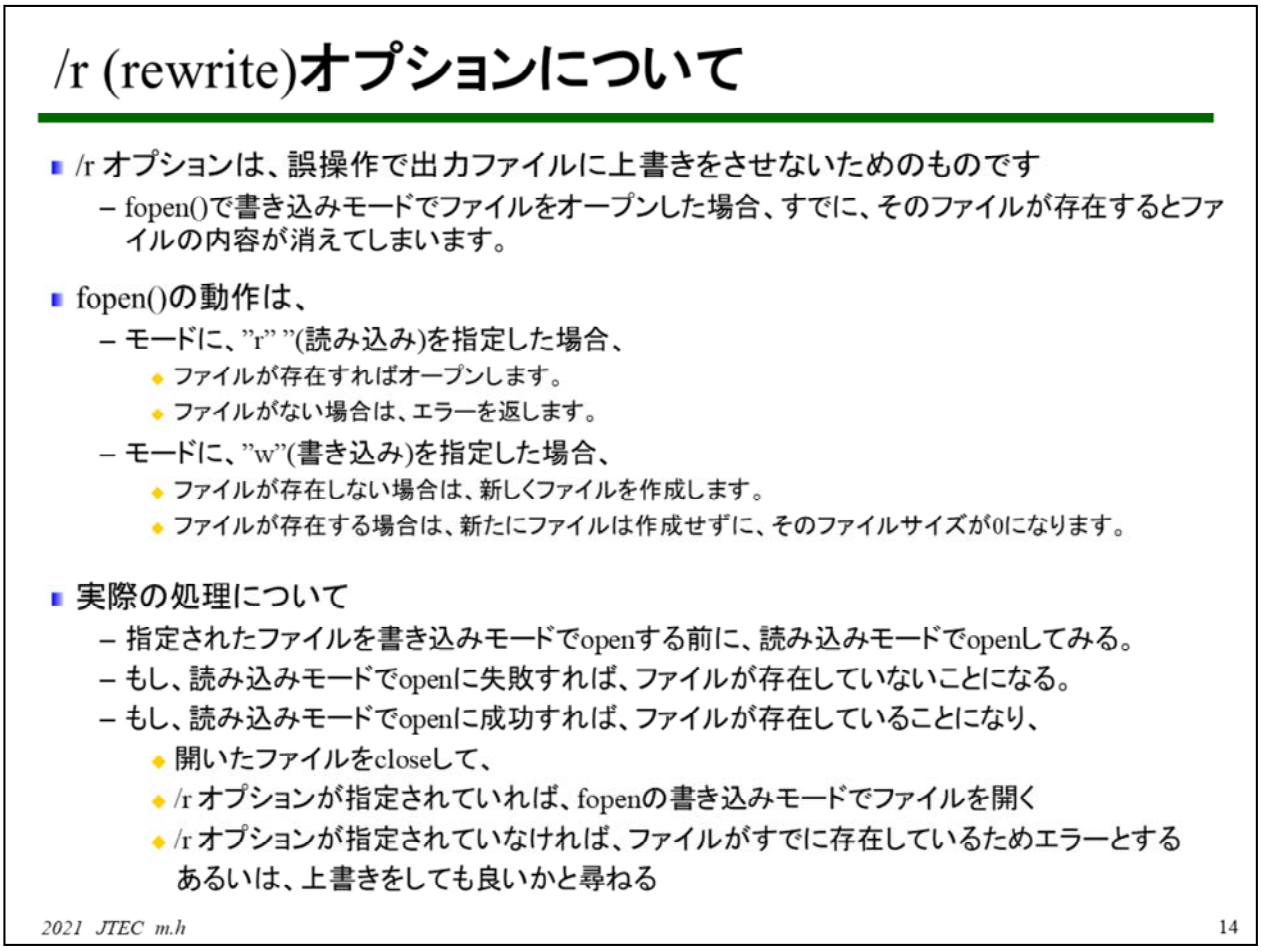
変換の考え方

Sレコード変換のアルゴリズムは、比較的簡単である。
S0～S9のタイプがあるが、違いはアドレスフィールドのバイト数だけである。
そのため、各タイプによって、アドレス長を調整すれば、以後の処理はすべて共通にすることができる。



2021 JTEC m.h 13

ここまで、説明をしてきましたが、わかりましたでしょうか?
Sレコード変換のアルゴリズムは、比較的簡単です。
S0～S9のタイプがあるが、違いはアドレスフィールドのバイト数だけです。
そのため、各タイプによって、アドレス長をさえ調整できれば、以後の処理はすべて共通になります。



/r オプションは、誤操作で出力ファイルに上書きをさせないためのものです。

fopen()で書き込みモードでファイルをオープンした場合、すでに、そのファイルが存在するとファイルの内容が消えてしまいます。

fopen()の動作は、

モードに、”r” ”(読み込み)を指定した場合、

　ファイルが存在すればオープンします。

　ファイルがない場合は、エラーを返します。

モードに、”w”(書き込み)を指定した場合、

　ファイルが存在しない場合は、新しくファイルを作成します。

　ファイルが存在する場合は、新たにファイルは作成せずに、そのファイルサイズが0になります。

実際の処理について

(1) 指定されたファイルを書き込みモードでopenする前に、読み込みモードでopenしてみる。

(2) もし、読み込みモードでopenに失敗すれば、ファイルが存在していないことになる。

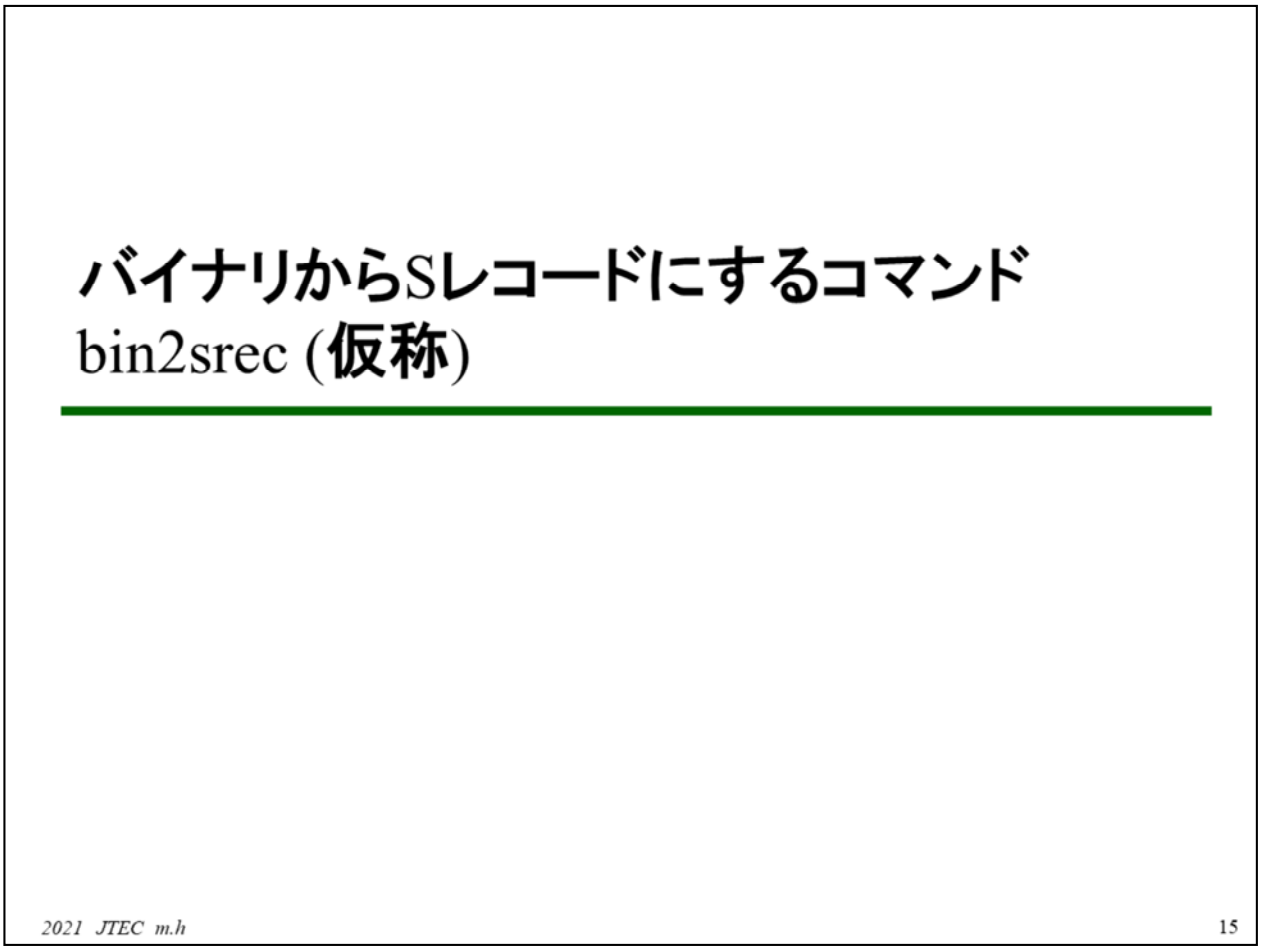
(3) もし、読み込みモードでopenに成功すれば、ファイルが存在していることになり、

　開いたファイルをcloseして、

　(4) /r オプションが指定されていれば、fopenり書き込みモードでファイルを開く

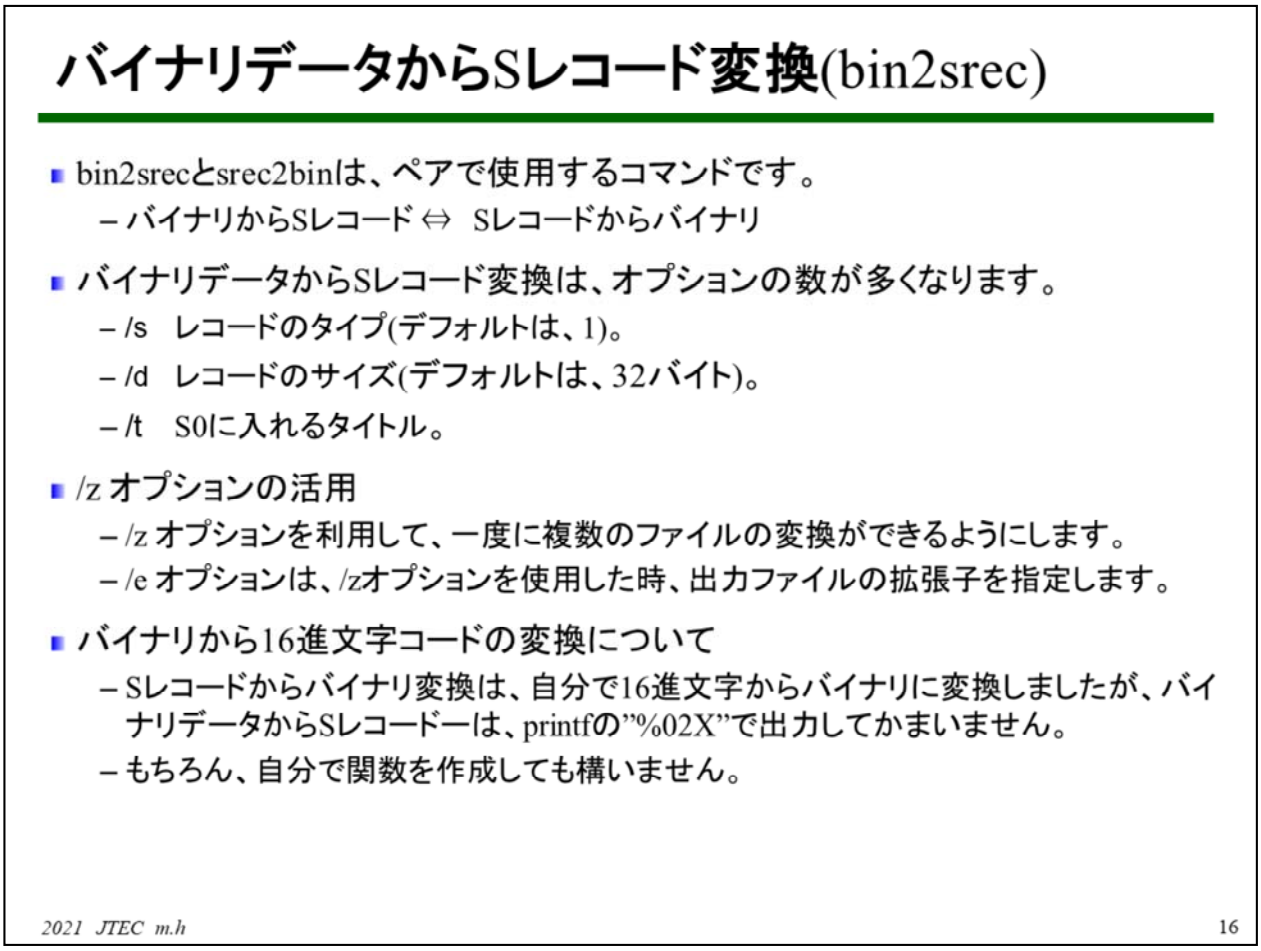
　(5) /r オプションが指定されていなければ、ファイルがすでに存在しているためエラーとする

　　あるいは、上書きをしても良いかを尋ねる。



Sレコードからバイナリ変換(srec2bin)が完成したら、次は、その逆のバイナリからSレコード変換する(bin2srec)になります。

bin2srecとsrec2binは、ペアで使用するコマンドです。



bin2srec(バイナリからSレコード変換)は、オプションの数が多くなります。srec2binにあったオプションに加え、

/s　　レコードのタイプ(デフォルトは、1)。

/d　　レコードのサイズ(デフォルトは、32バイト)。

/t　　S0に入れるタイトルの指定。

が必要になります。

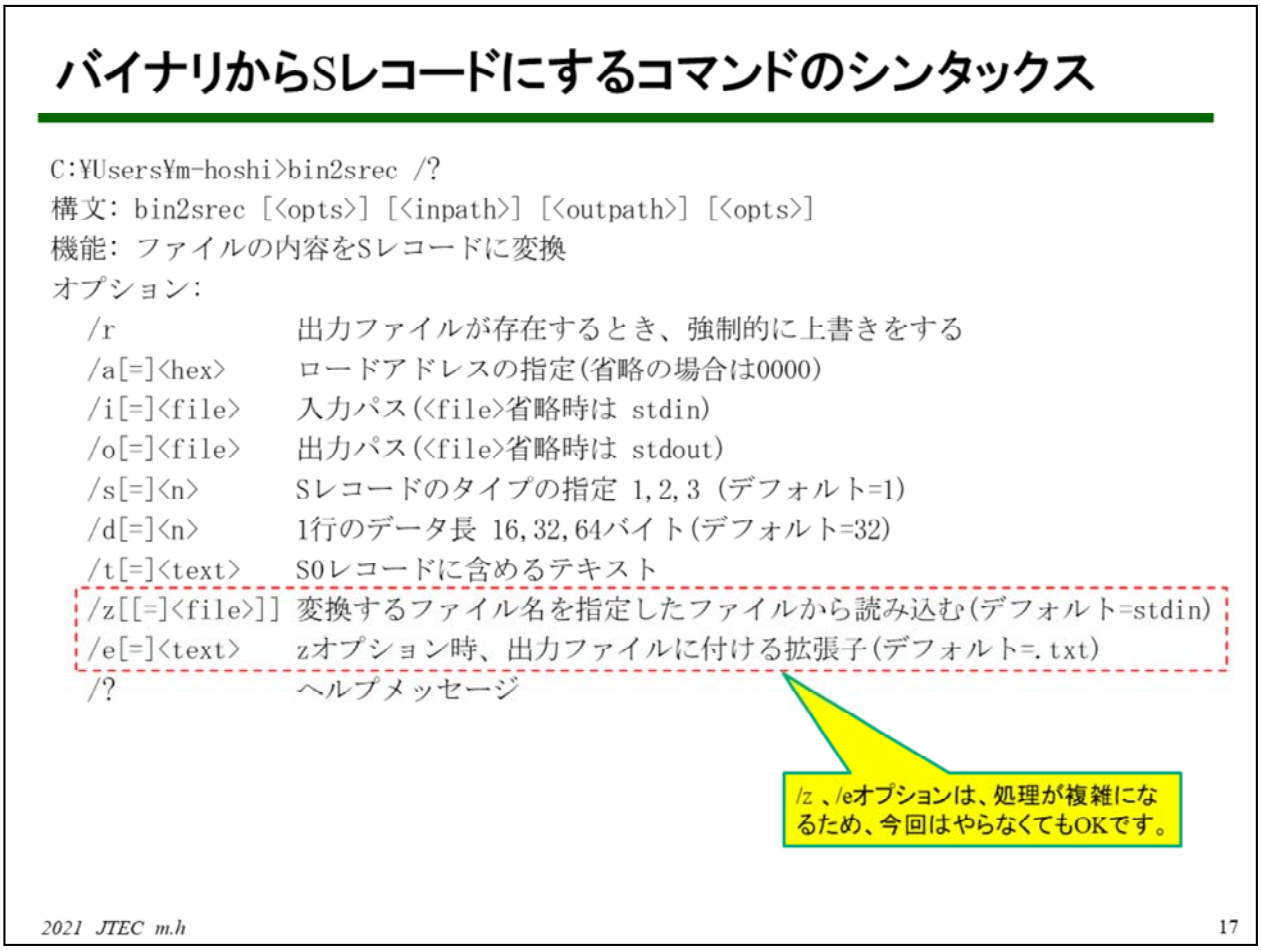
/z オプションの活用

変換するファイル名を、/zで指定されたファイルのリストから読み込みます。/z オプションを利用すると、一度に複数のファイルの変換ができるようになります。便利な機能ですが、ちょっと処理が面倒になります。ので、今回はサポートをしなくても良いです。

/e オプションは、/zオプションを使用した時、出力ファイルの拡張子を指定します(これは、入力ファイル名と出力ファイル名が同じにならないようにするためです)。

Sレコードからバイナリ変換は、自分で16進文字からバイナリに変換しましたが、バイナリデータからSレコードーは、printfの"%02X"で出力してかまいません。このとき、16進数は、大文字で出力します。

もちろん、自分で関数を作成しても構いません。



bin2srec(バイナリからSレコード)のヘルプメッセージです。Sレコードからバイナリに比べると、オプションの数が多くなります。

/r、/i、/o は、bin2srec(Sレコードからバイナリ)と同じなので説明は省略します。

/a[=]<hex>　ロードアドレスの指定(省略の場合は0000)

ロードアドレスと表現していますが、アドレスの初期値のことです。本来のSレコードは、受信をしたとき、このロードアドレスのメモリからデータを書き込んでいきます。

/s[=]<n>　Sレコードのタイプの指定 1,2,3 (デフォルト=1)

Sレコードのタイプを指定します。タイプは、1、2、3のいずれかになります。

/d[=]<n>　1行のデータ長 16,32,64バイト(デフォルト=32)

1行のデータ長になります。このサイズ分をfread( )で読込みます。

/t[=]<text>　S0レコードに含めるテキスト

S0レコードに入れるテキストになります。ここに書いた文字列がS0レコートのデータとなります。

/z[[=]<file>]]　変換するファイル名を指定したファイルから読み込む (デフォルト=stdin)

Sレコードに変換するファイル名を、zオプションで指定したファイルから読み込んで処理します。

ファイル名の指定がない場合は、stdinからファイル名を入力します。

/e[=]<text>　zオプション時、出力ファイルに付ける拡張子(デフォルト=_s.txt)

zオプションでファイル名を読込んだとき、出力ファイルに付ける拡張子です。

省略時は、"_s.txt" が付加されます。

なお、研修では、/zオプションと/e/オプションはサポートをしなくても良いです。余裕のある人は、挑戦してください。

バイナリからSレコード変換のヒント

- main関数でコマンド引数の解析が終わったら、switch-case文で、レコードタイプを調べて、アドレス長と、終了レコードのタイプを設定しておく
  - タイプが「1」なら終了レコードは、「9」
  - タイプが「2」なら終了レコードは、「8」
  - タイプが「3」なら終了レコードは、「7」
- そうしてから、whileループで、ファイルを読込む
  - バイナリの読み込みなので、dumpコマンドと同じfreadを使う
  - このとき、1回のfreadで読込むバイト数は、データ長で指定したサイズになる
  - そして、読込んだデータとサイズを引数にして、バイナリからSレコード変換の関数を呼ぶ
- バイナリからSレコード変換で、ちょっと面倒なのがアドレスの出力
  - 単にアドレスだけの出力であれば、"%04X"、"%06X"、"%08X"で澄む
  - しかし、アドレスも1バイトごとチェックサムに足しこむ必要がある
- データの出力は簡単で、読込んだデータサイズ分をforループで出力
  - 単純に、"%02X"で出力
- 最後は、チェックサムの出力
  - チェックサムは、値をビット反転(~)出力します。ビット反転とは1の補数にする
  - ビット反転をすると、これは、0xFFから引き算をしたと同じ値になる

2021 JTEC m.h 18

main関数で、コマンド引数の解析が終わったら、switch-case文で、レコードタイプを調べて、アドレス長と、終了レコードのタイプを設定しておきます。

タイプが「1」なら終了レコードは「9」、タイプが「2」なら「8」を、タイプが「3」なら「7」を設定しておきます。

その後、実際のデータを読込む前に、S0レコードだけを出力します。

そうしてから、srec2binと同様に、whileループで、ファイルを読込みます。

バイナリの読み込みなので、dumpコマンドと同じfreadを使います。

このとき、1回のfreadで読込むバイト数は、データ長で指定したサイズになります。

freadは、読込んだバイト数を返してくるので、読込んだデータとサイズを引数にして、バイナリからSレコード変換の関数を呼びます。

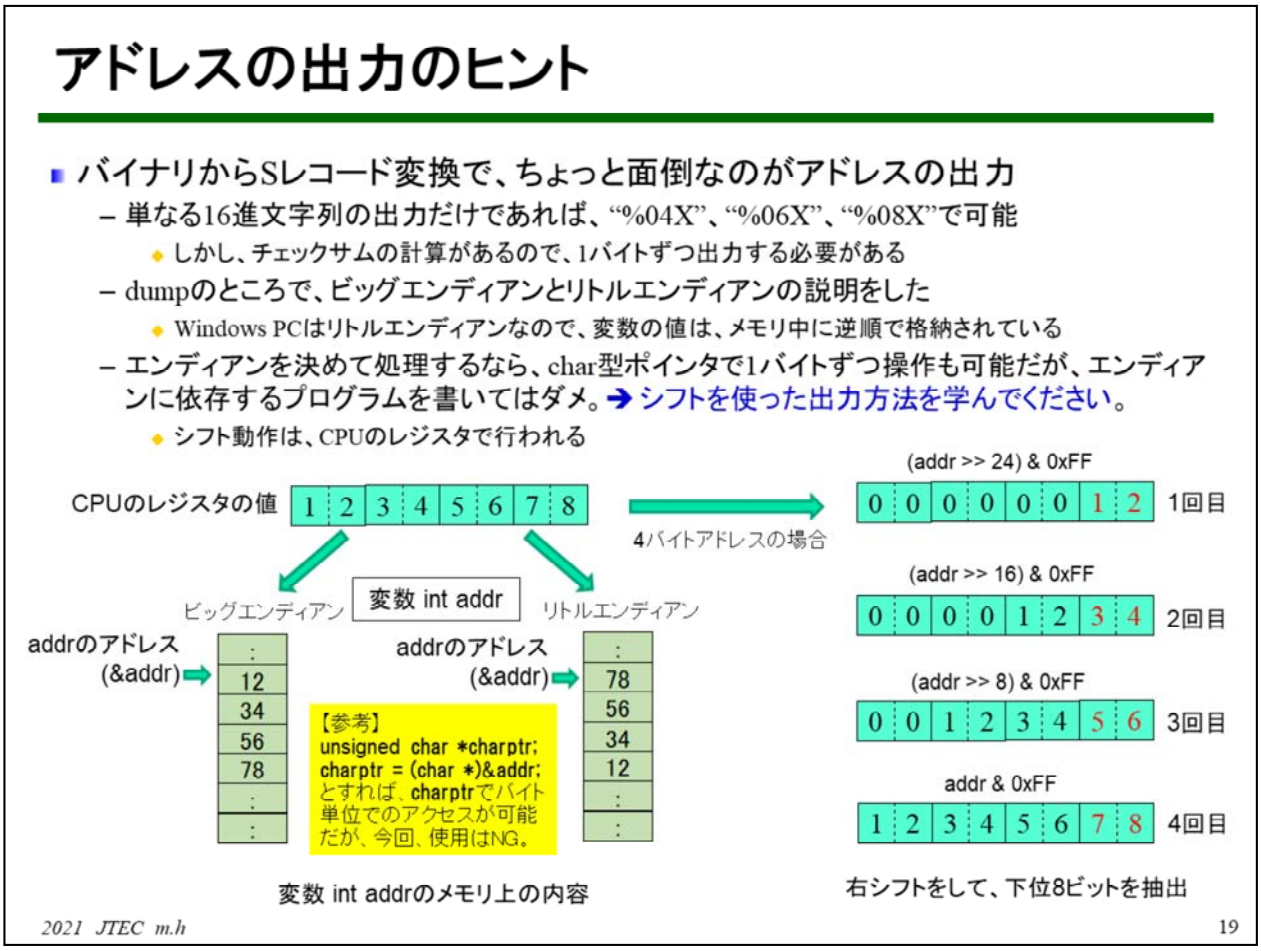
データの読み込みが終了してwhileループを抜けたら、最後に終了レコードを出力して終了です。

bin2srec の関数では、最初に'S'とレコードタイプの2文字を出力します。そして、次のレコード長は、実際に読込んだバイト数に、アドレス長、チェックサムの1バイトを足した値を出力します。構造体の中で設定されているデータ長ではないので注意をしてください。

次にアドレスの出力です。バイナリからSレコード変換で、ちょっと面倒なのが、このアドレスの出力になります。単にアドレスだけの出力であれば、"%04X"、"%06X"、"%08X"で済みます。しかし、アドレスも1バイトごとチェックサムに足しこむ必要があります。そのため、アドレスの内容を、8ビットずつ右シフトしてサイズ分を出力する必要があります。

データの出力は簡単で、読込んだデータサイズ分をforループで出力します。このとき、ループする回数は、データ長ではなく、実際に読込んだデータサイズです。単純に、"%02X"で出力します。

最後は、チェックサムの出力です。チェックサムは、値をビット反転(~演算子を使用して)出力します。ビット反転とは1の補数にすることです。これは、0xFFから引き算をしたと同じになります。



バイナリからSレコード変換で、ちょっと面倒なのがアドレスの出力です。

単なる16進文字列の出力だけであれば、“%04X”、“%06X”、“%08X”で可能ですが、チェックサムの計算があるので、1バイトずつ出力する必要があります。

dumpのところで、ビッグエンディアンとリトルエンディアンの説明をしました。

Windows PCはリトルエンディアンなので、変数の値は、メモリ中に逆順で格納されています。

ただ、チェックサムの計算は加算だけなので、char型ポインタで1バイトずつ操作でも可能ですが、研修なので、シフトを使った出力方法を学んでください。

シフト動作は、変数がCPU内のレジスタに読込まれて行われます。

そのため、変数を右に8ビットずつシフトをして、下位8ビットを取り出します。

```
unsigned char charval;
charval = (addr >> 8 ) & 0xFF;  // シフトして下位8ビットを取り出す
chksum += charval;              // チェックサムに足す
printf("%02X", charval);        // charvalを出力
```

上記のような処理を、アドレス長に応じてやります。

2バイトアドレスの場合は、8ビット、シフトなし(2回)

3バイトアドレスの場合は、16ビット、8ビット、シフトなし(3回)

4バイトアドレスの場合は、24ビット、16ビット、8ビット、シフトなし(4回)

なお、int型の変数を、char型のポインタにすれば1バイト単位でアクセスはできますが、ビッグエンディアン、リトルエンディアンによってアクセスする方向を逆にする必要があります。

エンディアンに依存するプログラムを書いてはダメです。

/z オプションの説明

- /z オプションは、変換するファイル名を/zで指定されたファイル名リストから読み込みます。
- ファイル名を与えず /z だけにすると標準入力(stdin)から読み込みます
- /zがない場合は、stdinからファイル名を入力します。

例えば、filelist.txt に下記の内容になっているとします。
fileA
fileB
fileC
:
fileX

そのとき、コマンドラインから、
bin2srec /z= filelist.txt と入力すると、Sレコードに変換するファイル名をfilelist.txtから読み込んで処理します。

/zオプションで、ファイル名リストのファイルの指定がない場合は、stdinからファイル名を入力します。
この方法を応用すると、パイプラインを使って、例えば、

dir /b | bin2srec /z ← dir /b の/bオプションは、ファイル名のみを出力する

のように入力すると、dirで出力されたファイルのすべてがSレコードに変換されます。

2021 JTEC m.h 20

zオプションは、複数のファイルを連続して変換するとき使います。
変換するファイル名を/zで指定されたファイル名リストから読み込みます。
「/z=ファイル名」として、ファイル名の入っているファイルをオープンして、その中に書かれているファイル名を変換するファイルとして処理をします。つまり、変換するファイル名の間接指定となります。

例えば、filelist.txt が下記の内容になっているとします。
fileA
fileB
fileC
:
fileX
bin2srec /z= filelist.txt と入力すると、Sレコードに変換するファイル名を、filelist.txtから読み込んで処理します。

zオプションで、ファイル名リストの指定がない場合は、stdinからファイル名を入力します。
この方法を応用すると、パイプラインを使って、例えば、
dir /b | bin2srec /z とすると、dirで出力されるファイルがすべてSレコードに変換されます。

このとき、出力ファイルには、/eオプションで指定された拡張子が付加されます。/eオプションを省略した場合は、"_s.txt"が付加されます。

シリアル通信の参考資料

2021 JTEC m.h 21

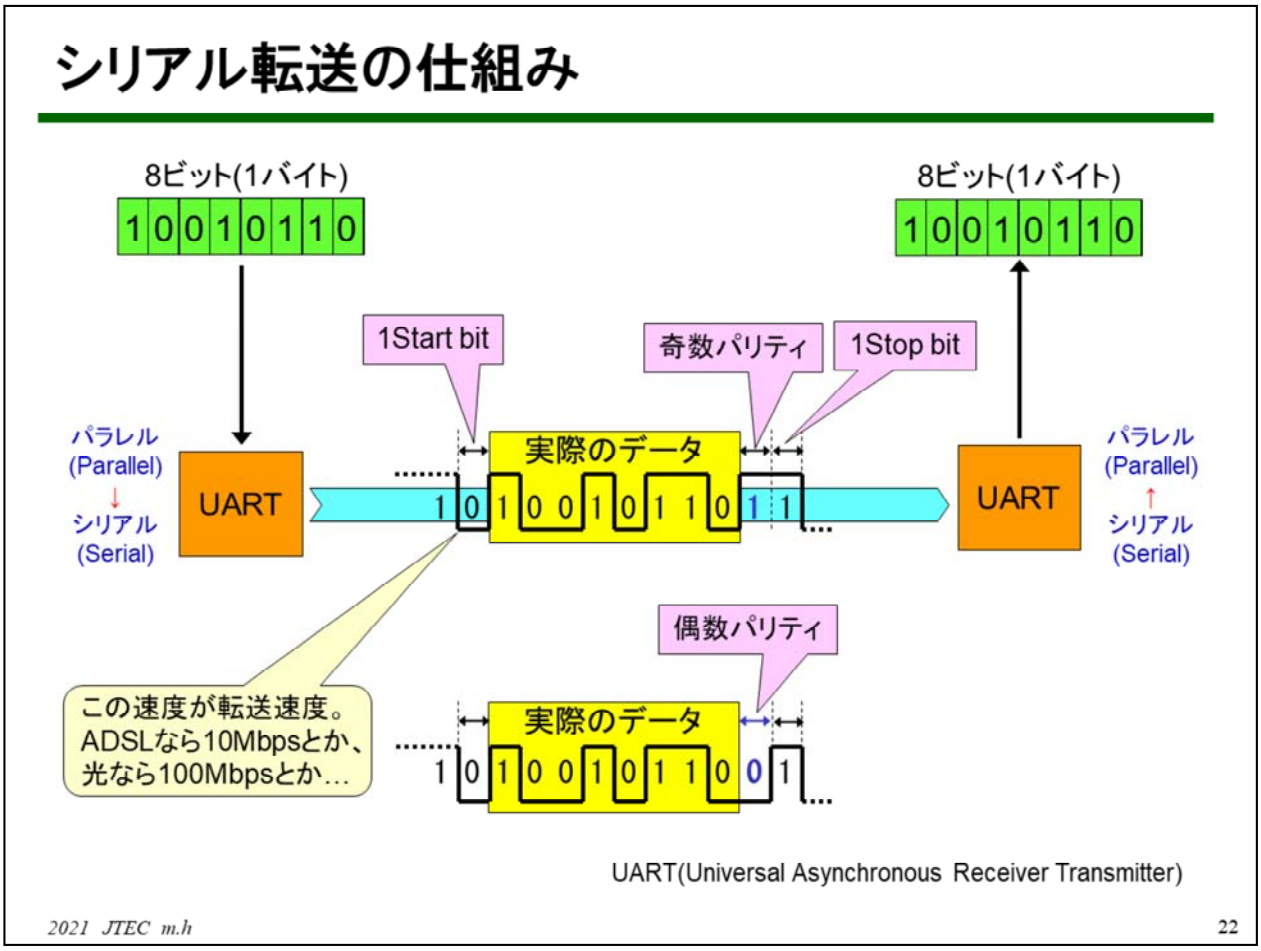
シリアル通信とは、データを1ビットずつ転送する通信です。昔、プリンタやハードディスクなどは、パラレ転送と言って、8ビット単位でデータを転送していました。つまり、8ビットを同時に転送できるわけです。その代わり、機器と繋ぐ線は、データ線が8本と、制御線、グランド線も含めて12本程度繋いでいました。

現在では、USBの発達により、PCにパラレルで機器をつなぐことはほとんどなくなりました。

USBは、Universal Serial Busの略で、データをシリアルで転送する規格の名前です。

当然、8ビットを同時に転送するパラレル転送のほうが、1ビットずつ送るよりも、転送速度は速いです。しかし、転送速度が超高速になった今、パラレル転送だとノイズの関係などから速くすることが難しくなりました。それに伴い、シリアル通信の技術が発展し、今ではシリアルのほうが高速に転送できるようになりました。

現在は、シリアル通信が主流となっています。皆さんが使っている、PCのUSBにつながる機器は、すべてシリアル通信をしています。電話線、光ケーブル、ネットワークのLANケーブルも、すべてシリアル転送で行われています。



コンピュータは、2進数をパラレル(並列)で処理をする。その一度に処理ができるビットによってCPUの能力が決まってきます。

世界で最初のマイクロプロセッサは、4ビットであった。それが、8ビット、16ビット、32ビットと進化をしてきました。

それに伴い、メモリをアクセスするためにはそのビット幅の線が必要になる。実際、32ビットのCPUでは、32本のデータ線が必要です。

しかし、コンピュータ内部では、ビット幅に合わせて配線ができるが、通信でそのデータを送る場合バス幅に合わせて伝送するのは不可能です。

そこで、パラレルの信号線をシリアル(直列)にして送るようにしています。

そのパラレルのデータも、8ビット、すなわち1バイト単位で転送されます。

8ビットのデータは、1ビットづつ順番に出力され、受け側は逆に1ビットの信号から8ビットのデータに変換します。

通信方法としては、同期通信、非同期通信とがあるが、組込みシステムでは非同期通信が多く使われています。

その代表が、UART(Universal Asynchronous Receiver Transmitter)と言われ、基本的な通信方法として最も多く利用されています。

非同期歩調式のシリアル通信

- RS-232C (Recommended Standard 232 version C)とは
  - シリアルポートのインターフェース規格
- UART (Universal Asynchronous Receiver Transmitter)とは
  - シリアル通信を実現する回路を表す
- RS-232CもUARTも同じ意味で使われる
  - UART: 一般的なシリアル通信全般を示す。
  - RS-232C: 電気信号などの物理的な規格を規定
- RS-232Cの規格では、信号の電圧レベルを±5V～±15Vで規定
  - 標準では、0＝+5V～+12V、1=-5V～-12V
- COMポートとは
  - RS-232Cシリアル通信デバイスのWindowsが定めた名称

2021 JTEC m.h 23

シリアル通信には、同期式と非同期式があります。同期式とは、データ線の他に、クロック線があり、そのクロックに同期してデータを送る方法です。USBなどは、その方式です。

ただ、安価に使うシリアル通信は、古くから使われている非同期歩調式です。非同期歩調式とは、最初にスタートビット(0)が1ビット来て、そのあとは、自分の持っているクロックで一定間隔ごとに、0か1かを読み取る方式です。

通信速度は、150bpsから倍々で使われます。一般的に9600が標準で使われていましたが、現在は、19200、38400,76,800など多く使われています。

そのシリアル通信として、もっとも古くから使われていたのが、RS-232Cです

RS-232C (Recommended Standard 232 version C)とは

　シリアルポートのインターフェース規格

UART (Universal Asynchronous Receiver Transmitter)とは

　シリアル通信を実現する回路を表す

RS-232CもUARTも同じ意味で使われています。

UART: 一般的なシリアル通信全般を示す。

RS-232C: 電気信号などの物理的な規格を規定

RS-232Cの規格では、信号の電圧レベルを±5V～±15Vで規定

標準では、0 = +5V～+12V、1= -5V～-12V

COMポートとは、

シリアル通信デバイスとして、マイクロソフトが定めた名称です。COMは、Communication の略です。

COM(RS-232C)ポートのピン配置と信号名

ピンーオスDTE配列の機器
（PCなど一般的な機器の配列）

ピン番号	信号名	方向	備考
1	CD	←	通常使用されない
2	RXD	←	
3	TXD	→	
4	DTR	→	
5	GND	―	
6	DSR	←	
7	RTS	→	
8	CTS	←	
9	RI	←	通常使用されない

ピンーメスDCE配列の機器
（モデムなどの配列）

ピン番号	信号名	方向	備考
1	CD	→	通常使用されない
2	RXD	→	
3	TXD	←	
4	DTR	←	
5	GND	―	
6	DSR	→	
7	RTS	←	
8	CTS	→	
9	RI	→	通常使用されない

市販されているUSBシリアル変換ケーブル

2021 JTEC m.h 24

USBの普及で、最近はCOMポートと言われる、シリアル通信用のコネクタがあるPCがほとんどなくなりました。専用のコネクタがなくても、USBから変換ができるようになっているからです。

もともと、USBは、Universal Serial Busの略で、高速でシリアル転送をするための規格です、

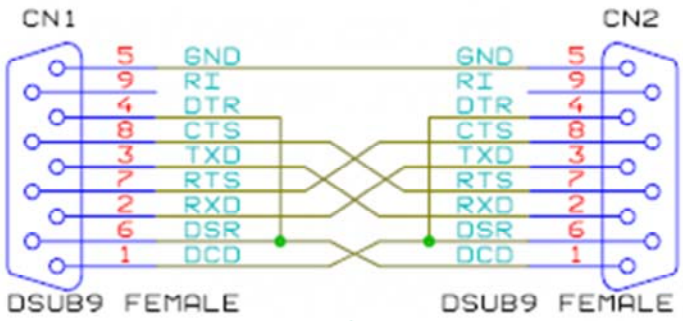
写真のようなUSBをシリアルに変換するケーブルが1000~2000円程度で市販されています。

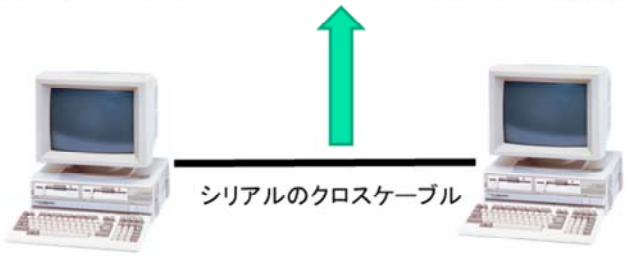
RS-232Cケーブルのクロス接続

PCとPCの1対1の接続には、クロスケーブルが必要

ピンNo.	信号名	入出力	内容
1	DCD	IN	キャリア検出
2	RxD	IN	受信データ
3	TxD	OUT	送信データ
4	DTR	OUT	データ端末レディ
5	GND	-	グランド
6	DSR	IN	データセットレディ
7	RTS	OUT	送信リクエスト
8	CTS	IN	送信可
9	RI	IN	被呼表示

PC側には、メスのコネクタが付いている





2021 JTEC m.h 25

写真は、UARTのコネクタが付いているPCです。PC側には、同じピン配置のメスのコネクタが付きます。

そのため、PC同士でシリアルケーブルをつなげて通信をするときは、クロスケーブルというものが必要になります。つまり、PCのコネクタのピン配置は同じため、単純にケーブルをつなげると、受信データのピン同士、送信データのピン同士をつなぐことになり、通信ができません。

そのようなときは、受信データと送信データとの線を入れ替えた(クロス)した、クロスケーブルが必要になります。実際は、受信データと送信データだけでなく、その他の信号線も入れ替える必要があります。

なお、クロスケーブルに対して、線の入れ替えをしないケーブルを、ストレートケーブルと言います。

LANケーブルも同様に、PC同士を直接接続する場合はクロスケーブルが必要になります。ハブを経由すれば、クロスケーブルを使う必要はありません。

ただ、最近のルータやパブは、ストレートケーブルであっても、クロスケーブルであっても、自動的に検出して動作をします。クロスケーブルが必要にケースは、PC同士を直接つなぐ時だけになりました。