AVR200：乗除算ルーチン

要点

- 8ビットと16ビット実装
- 符号付きと符号なしのルーチン
- 速度とコード量の最適化ルーチン
- 実行可能な例プログラム
- ハードウェア乗除算に匹敵する速度
- 例: 8×8乗算2.1μs、16×16乗算6.25～7.25μs (16MHz)
- 極端に簡潔なコード

1. 序説

この応用記述は8ビットと16ビット、符号付きと符号なしの数値の乗除算用サブルーチンを一覧にします。核と成る性能特性とでの全実装の一覧が**表1-1.**で与えられます。

表1-1. 性能値要約

応用	コード量(語)	実行時間(周期)
符号なし,8×8=16ビット,(コード量最適化)	8	57
符号なし,8×8=16ビット,(速度最適化)	34	34
符号付き,8×8=16ビット,(コード量最適化)	10	73
符号なし,16×16=32ビット,(コード量最適化)	13	153～169
符号なし,16×16=32ビット,(速度最適化)	116	100～116
符号付き,16×16=32ビット,(コード量最適化)	16	194～226
符号なし,8÷8=8:8ビット,(コード量最適化)	10	65～73
符号なし,8÷8=8:8ビット,(速度最適化)	56	48～56
符号付き,8÷8=8:8ビット,(コード量最適化)	20	72～83
符号なし,16÷16=16:16ビット,(コード量最適化)	15	178～210
符号なし,16÷16=16:16ビット,(速度最適化)	177	145～177
符号付き,16÷16=16:16ビット,(コード量最適化)	39	187～234

応用記述一覧記録は次の2つのファイルから成ります。

- "avr200.asm" ：コード量最適化乗除算ルーチン
- "avr200b.asm" ：速度最適化乗除算ルーチン

(訳注) 最適化のため、本書に対応するプログラム例は原書のそれと若干異なります。同時にソース内のシンボル名も異なっていますので注意してください。

8-bit AVR® マイクロ コントローラ

応用記述

Rev. 0936C-05/06, 0936CJ3-12/13

2. 8×8=16 符号なし乗算 - ML8U

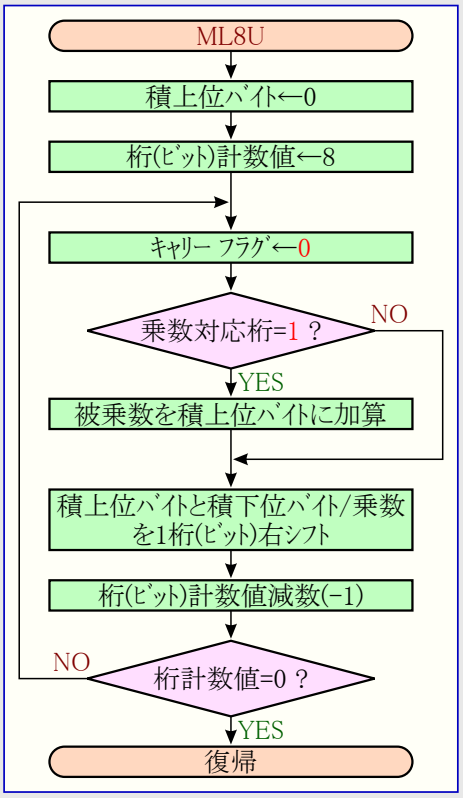
両方のファイルが符号なし8ビット乗算を実行する"ML8U"と呼ばれるルーチンを含みます。両実装は同じ算法を基にします。けれども、コード量最適化実装は繰り返しコードを使用し、それに反して速度最適化コードは一直線のコード実装です。図2-1.はコード量最適化版用の流れ図を示します。

2.1. 算法内容

コード量最適化用の算法は次の通りです。

1. 積上位バイトを空にします。
2. 桁(ビット)計数値に8を設定します。
3. キャリー フラグを解除(0)します。
4. 乗数の対応桁(ビット)=1なら、被乗数を積上位バイトに加算します。
5. 積上位バイトと積下位バイト/乗数を1桁(ビット)右にシフトします。
6. 桁(ビット)計数値を減数(-1)します。
7. 桁(ビット)計数値が0でなければ3.からを繰り返します。

図2-1. ML8U流れ図(コード量最適化実装)



2.2. 使用法

"ML8U"の使用法は両版に関して同じです。

1. 乗数と被乗数をレジスタ変数ML8XとML8Yに各々格納してください。
2. "ML8U"を呼び出してください。
3. 16ビットの積が2つのレジスタ変数ML8H(上位バイト)とML8L(下位バイト)で得られます。

使用レジスタ、コードと実行時間の最小化のために乗数と積下位バイトが同じレジスタを共有することに注目してください。

2.3. 性能

表2-1. "ML8U"使用レジスタ(コード量最適化実装)

レジスタ	入力	内部	出力
R16	ML8Y:被乗数		
R17	ML8X:乗数		ML8L:積下位バイト
R18			ML8H:積上位バイト
R19		CNT:桁(ビット)計数値	

表2-2. "ML8U"性能値(コード量最適化実装)

項目	値
コード容量(語)	8+RET
実行時間	57+RET
使用レジスタ	下位レジスタ=0, 上位レジスタ=4, ポインタ=0
使用割り込み	なし
使用周辺機能	なし

表2-3. "ML8U"使用レジスタ(速度最適化実装)

レジスタ	入力	内部	出力
R16	ML8Y:被乗数		
R17	ML8X:乗数		ML8L:積下位バイト
R18			ML8H:積上位バイト

表2-4. "ML8U"性能値(速度最適化実装)

項目	値
コード容量(語)	34+RET
実行時間	34+RET
使用レジスタ	下位レジスタ=0, 上位レジスタ=3, ポインタ=0
使用割り込み	なし
使用周辺機能	なし

3. 8×8=16 符号付き乗算 - ML8S

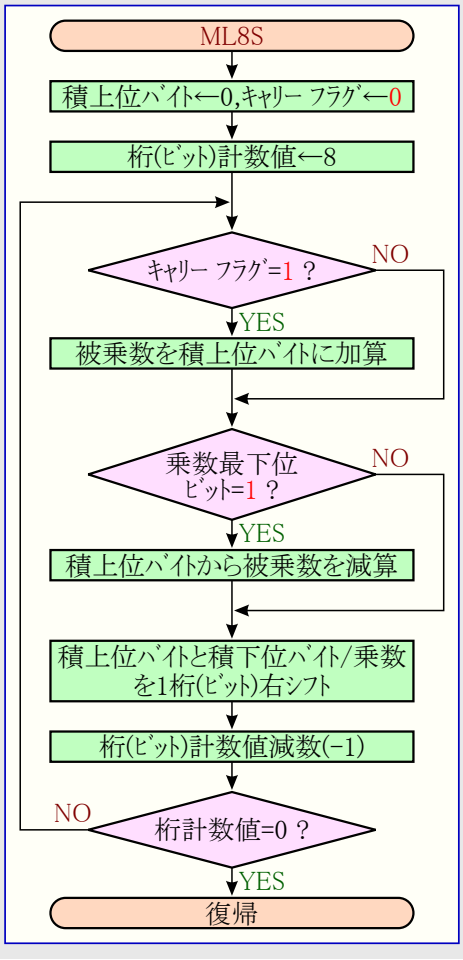
"avr200.asm"で得られるこのｻﾌﾞﾙｰﾁﾝは符号付き8×8乗算を実装します。負数は2の補数として表されます。この応用はﾌﾞｰｽの演算法(Booth's algorithm)の実装です。この演算法は小さく且つ速いｺｰﾄﾞを提供します。けれども、使用者が覚えているべき、"結果の全16ﾋﾞｯﾄが必要とされる場合、被乗数として最大負数(-128)での使用時にこの演算法は失敗する"、との1つの制限を持ちます。

3.1. 算法内容

符号付き8×8乗算用の算法は次の通りです。

1. 積上位ﾊﾞｲﾄを空にし、ｷｬﾘｰ ﾌﾗｸﾞを解除(0)します。
2. 桁(ﾋﾞｯﾄ)計数値に8を設定します。
3. ｷｬﾘｰ(ｼﾌﾄ前の乗数の最下位ﾋﾞｯﾄ=1)なら、被乗数を積上位ﾊﾞｲﾄに加算します。
4. 現在の乗数最下位ﾋﾞｯﾄ=1なら、積上位ﾊﾞｲﾄから被乗数を減算します。
5. 積上位ﾊﾞｲﾄと積下位ﾊﾞｲﾄ/乗数を1桁(ﾋﾞｯﾄ)右にｼﾌﾄします。
6. 桁(ﾋﾞｯﾄ)計数値を減数(-1)します。
7. 桁(ﾋﾞｯﾄ)計数値が0でなければ3.からを繰り返します。

図3-1. ML8S流れ図



3.2. 使用法

"ML8S"の使用法は次の通りです。

1. 乗数と被乗数をﾚｼﾞｽﾀ変数ML8XとML8Yに各々格納してください。
2. "ML8S"を呼び出してください。
3. 16ﾋﾞｯﾄの積が2つのﾚｼﾞｽﾀ変数ML8H(上位ﾊﾞｲﾄ)とML8L(下位ﾊﾞｲﾄ)で得られます。

使用ﾚｼﾞｽﾀ、ｺｰﾄﾞと実行時間の最小化のために乗数と積下位ﾊﾞｲﾄが同じﾚｼﾞｽﾀを共有することに注目してください。

3.3. 性能

表3-1. "ML8S"使用ﾚｼﾞｽﾀ

ﾚｼﾞｽﾀ	入力	内部	出力
R16	ML8Y:被乗数		
R17	ML8X:乗数		ML8L:積下位ﾊﾞｲﾄ
R18			ML8H:積上位ﾊﾞｲﾄ
R19		CNT:桁(ﾋﾞｯﾄ)計数値	

表3-2. "ML8S"性能値

項目	値
ｺｰﾄﾞ容量(語)	10+RET
実行時間	73+RET
使用ﾚｼﾞｽﾀ	下位ﾚｼﾞｽﾀ=0, 上位ﾚｼﾞｽﾀ=4, ﾎﾟｲﾝﾀ=0
使用割り込み	なし
使用周辺機能	なし

4. 16×16=32 符号なし乗算 - ML16U

両方のファイルが符号なし16ビット乗算を実行する"ML16U"と呼ばれるルーチンを含みます。両実装は同じ算法を基にします。けれども、コード量最適化実装は繰り返しコードを使用し、それに反して速度最適化コードは一直線のコード実装です。図4-1.はコード量最適化(繰り返し)版用の流れ図を示します。

4.1. 算法内容

コード量最適化用の算法は次の通りです。

1. 積上位語(第3,4バイト)を空にします。
2. 桁(ビット)計数値に16を設定します。
3. 乗数を1桁(ビット)右にシフトします。
4. キャリー(シフト前の乗数の最下位ビット=1)なら、被乗数を積上位語に加算します。
5. 積上位語と積下位語/乗数を1桁(ビット)右にシフトします。
6. 桁(ビット)計数値を減数(-1)します。
7. 桁(ビット)計数値が0でなければ4.からを繰り返します。

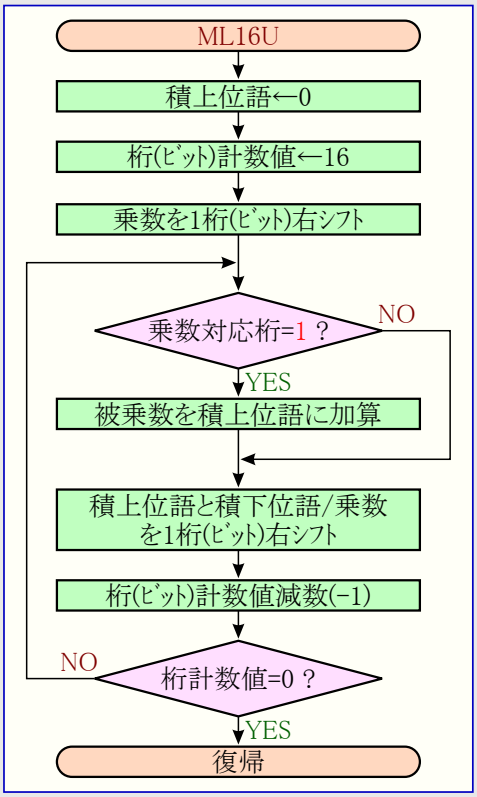
4.2. 使用法

"ML16U"の使用法は両版に関して同じです。

1. 乗数上位/下位バイトをレジスタ変数ML16XH/ML16XLに各々格納してください。
2. 被乗数上位/下位バイトをレジスタ変数ML16YH/ML16YLに各々格納してください。
3. "ML16U"を呼び出してください。
4. 32ビットの積が4つのレジスタ変数ML16_3:ML16_2:ML16_1:ML16_0で得られます。

使用レジスタ、コードと実行時間の最小化のために乗数と積下位語が同じレジスタを共有することに注目してください。

図4-1. ML16U流れ図(コード量最適化実装)



4.3. 性能

表4-1. "ML16U"使用レジスタ(コード量最適化実装)

レジスタ	入力	内部	出力
R16	ML16YL:被乗数下位バイト		
R17	ML16YH:被乗数上位バイト		
R18	ML16XL:乗数下位バイト		ML16_0:積最下位バイト
R19	ML16XH:乗数上位バイト		ML16_1:積第2バイト
R20			ML16_2:積第3バイト
R21			ML16_3:積最上位バイト
R22		CNT: 桁(ビット)計数値	

表4-2. "ML16U"性能値(コード量最適化実装)

項目	値
コード容量(語)	13+RET
実行時間	153～169+RET
使用レジスタ	下位レジスタ=0, 上位レジスタ=7, ポインタ=0
使用割り込み	なし
使用周辺機能	なし

表4-3. "ML16U"使用レジスタ(速度最適化実装)

レジスタ	入力	内部	出力
R16	ML16YL:被乗数下位バイト		
R17	ML16YH:被乗数上位バイト		
R18	ML16XL:乗数下位バイト		ML16_0:積最下位バイト
R19	ML16XH:乗数上位バイト		ML16_1:積第2バイト
R20			ML16_2:積第3バイト
R21			ML16_3:積最上位バイト

表4-4. "ML16U"性能値(速度最適化実装)

項目	値
コード容量(語)	116+RET
実行時間	100～116+RET
使用レジスタ	下位レジスタ=0, 上位レジスタ=6, ポインタ=0
使用割り込み	なし
使用周辺機能	なし

5. 16×16=32 符号付き乗算 - ML16S

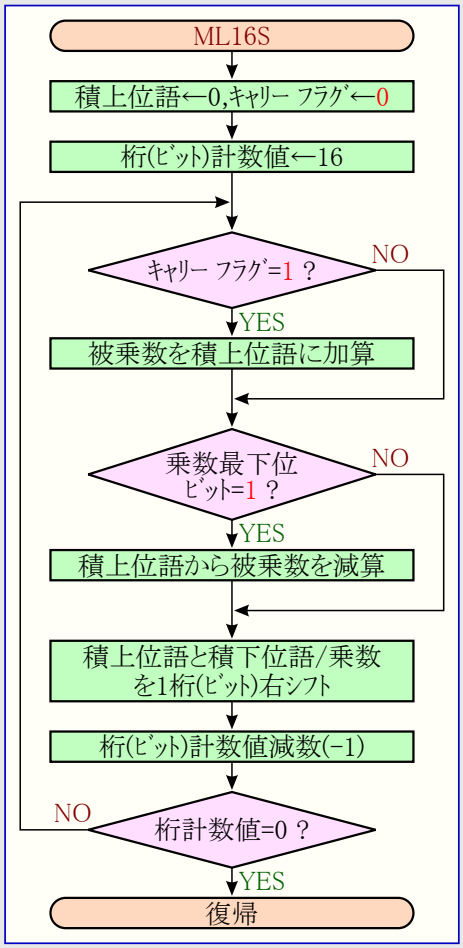
"avr200.asm"で得られるこのｻﾌﾞﾙｰﾁﾝは符号付き16×16乗算を実装します。負数は2の補数として表されます。この応用はﾌﾞｰｽの演算法(Booth's algorithm)の実装です。この演算法は小さく且つ速いｺｰﾄﾞを提供します。けれども、使用者が覚えているべき、"結果の全32ﾋﾞｯﾄが必要とされる場合、被乗数として最大負数(-32768)での使用時にこの演算法は失敗する"、との1つの制限を持ちます。

5.1. 算法内容

符号付き16×16乗算用の算法は次の通りです。

1. 積上位語(第3,4ﾊﾞｲﾄ)を空にし、ｷｬﾘｰ ﾌﾗｸﾞを解除(0)します。
2. 桁(ﾋﾞｯﾄ)計数値に16を設定します。
3. ｷｬﾘｰ(ｼﾌﾄ前の乗数の最下位ﾋﾞｯﾄ=1)なら、被乗数を積上位語に加算します。
4. 現在の乗数最下位ﾋﾞｯﾄ=1なら、積上位語から被乗数を減算します。
5. 積上位語と積下位語/乗数を1桁(ﾋﾞｯﾄ)右にｼﾌﾄします。
6. 桁(ﾋﾞｯﾄ)計数値を減数(-1)します。
7. 桁(ﾋﾞｯﾄ)計数値が0でなければ3.からを繰り返します。

図5-1. ML16S流れ図



5.2. 使用法

"ML16S"の使用法は次の通りです。

1. 乗数上位/下位ﾊﾞｲﾄをﾚｼﾞｽﾀ変数ML16XH/ML16XLに各々格納してください。
2. 被乗数上位/下位ﾊﾞｲﾄをﾚｼﾞｽﾀ変数ML16YH/ML16YLに各々格納してください。
3. "ML16S"を呼び出してください。
4. 32ﾋﾞｯﾄの積が4つのﾚｼﾞｽﾀ変数ML16_3:ML16_2:ML16_1:ML16_0で得られます。

使用ﾚｼﾞｽﾀ、ｺｰﾄﾞと実行時間の最小化のために乗数と積下位ﾊﾞｲﾄが同じﾚｼﾞｽﾀを共有することに注目してください。

5.3. 性能

表5-1. "ML16S"使用ﾚｼﾞｽﾀ

ﾚｼﾞｽﾀ	入力	内部	出力
R16	ML16YL:被乗数下位ﾊﾞｲﾄ		
R17	ML16YH:被乗数上位ﾊﾞｲﾄ		
R18	ML16XL:乗数下位ﾊﾞｲﾄ		ML16_0:積最下位ﾊﾞｲﾄ
R19	ML16XH:乗数上位ﾊﾞｲﾄ		ML16_1:積第2ﾊﾞｲﾄ
R20			ML16_2:積第3ﾊﾞｲﾄ
R21			ML16_3:積最上位ﾊﾞｲﾄ
R22		CNT:桁(ﾋﾞｯﾄ)計数値	

表5-2. "ML16S"性能値

項目	値
ｺｰﾄﾞ容量(語)	16+RET
実行時間	194～226+RET
使用ﾚｼﾞｽﾀ	下位ﾚｼﾞｽﾀ=0, 上位ﾚｼﾞｽﾀ=7, ﾎﾟｲﾝﾀ=0
使用割り込み	なし
使用周辺機能	なし

6. 8÷8=8(8) 符号なし除算 - DV8U

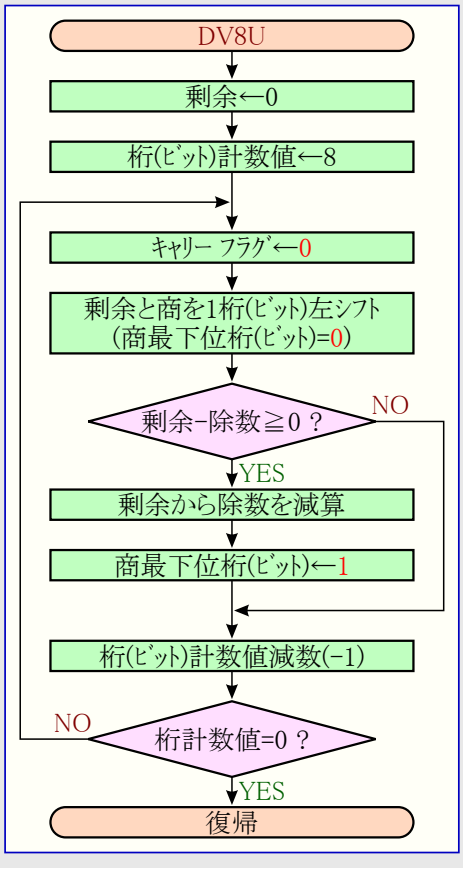
両方のファイルが符号なし8ビット除算を実行する"DV8U"と呼ばれるルーチンを含みます。両実装は同じ算法を基にします。けれども、コード量最適化実装は繰り返しコードを使用し、それに反して速度最適化コードは一直線のコード実装です。図6-1.はコード量最適化版用の流れ図を示します。

6.1. 算法内容

符号なし8÷8除算(コード量最適化)用の算法は次の通りです。

1. 剰余を空にし、キャリー フラグを解除(0)します。
2. 桁(ビット)計数値に8を設定します。
3. 現桁の商を0に仮設定し、被除数と商を1桁(ビット)左にシフトします。
4. 現在の剰余から除数が減算可能ならば減算し、現桁の商を1にします。
5. 桁(ビット)計数値を減数(-1)します。
6. 桁(ビット)計数値が0でなければ3.からを繰り返します。

図6-1. DV8U流れ図(コード量最適化実装)



6.2. 使用法

"DV8U"の使用法は両版に関して同じで、以下の手続きで記述されます。

1. 被除数(割られる数)をレジスタ変数DV8Yに格納してください。
2. 除数(割る数)をレジスタ変数DV8Xに格納してください。
3. "DV8U"を呼び出してください。
4. 商はDV8Rで、DV8Mで剰余が得られます。

6.3. 性能

表6-1. "DV8U"使用レジスタ(コード量最適化実装)

レジスタ	入力	内部	出力
R16	DV8Y:被除数		DV8R:商
R17			DV8M:剰余
R18	DV8X:除数		
R19		CNT:桁(ビット)計数値	

表6-2. "DV8U"性能値(コード量最適化実装)

項目	値
コード容量(語)	10+RET
実行時間	65～73+RET
使用レジスタ	下位レジスタ=0, 上位レジスタ=4, ポインタ=0
使用割り込み	なし
使用周辺機能	なし

表6-3. "DV8U"使用レジスタ(速度最適化実装)

レジスタ	入力	内部	出力
R16	DV8Y:被除数		DV8R:商
R17			DV8M:剰余
R18	DV8X:除数		

表6-4. "DV8U"性能値(速度最適化実装)

項目	値
コード容量(語)	56+RET
実行時間	48～56+RET
使用レジスタ	下位レジスタ=0, 上位レジスタ=3, ポインタ=0
使用割り込み	なし
使用周辺機能	なし

7. 8÷8=8(8) 符号付き除算 - DV8S

このｻﾌﾞﾙｰﾁﾝは符号付き8ﾋﾞｯﾄ除算を実装します。この実装はｺｰﾄﾞ量最適化です。負の場合、入力値は2の補数形式で表されるでしょう。

7.1. 算法内容

符号付き8×8乗算用の算法は次の通りです。

1. 除数と被除数の符号から商の符号を作成(XOR)して保存します。
2. 除数が負数なら、絶対値にします。(2の補数)。
3. 被除数が負数なら、絶対値にします。(2の補数)。
4. 剰余を空に、ｷｬﾘｰ ﾌﾗｸﾞを解除(0)します。
5. 桁(ﾋﾞｯﾄ)計数値に8を設定します。
6. 現桁の商を0に仮設定し、被除数と商を1桁(ﾋﾞｯﾄ)左にｼﾌﾄします。
7. 現在の剰余から除数が減算可能ならば減算し、現桁の商を1にします。
8. 桁(ﾋﾞｯﾄ)計数値を減数(-1)します。
9. 桁(ﾋﾞｯﾄ)計数値が0でなければ6.からを繰り返します。
10. 商の符号が負なら、商と剰余を負数(2の補数)に変換します。

7.2. 使用法

"DV8S"の使用法は以下の手続きに従ってください。

1. 被除数(割られる数)をﾚｼﾞｽﾀ変数DV8Yに格納してください。
2. 除数(割る数)をﾚｼﾞｽﾀ変数DV8Xに格納してください。
3. "DV8S"を呼び出してください。
4. 商はDV8Rで、DV8Mで剰余が得られます。

7.3. 性能

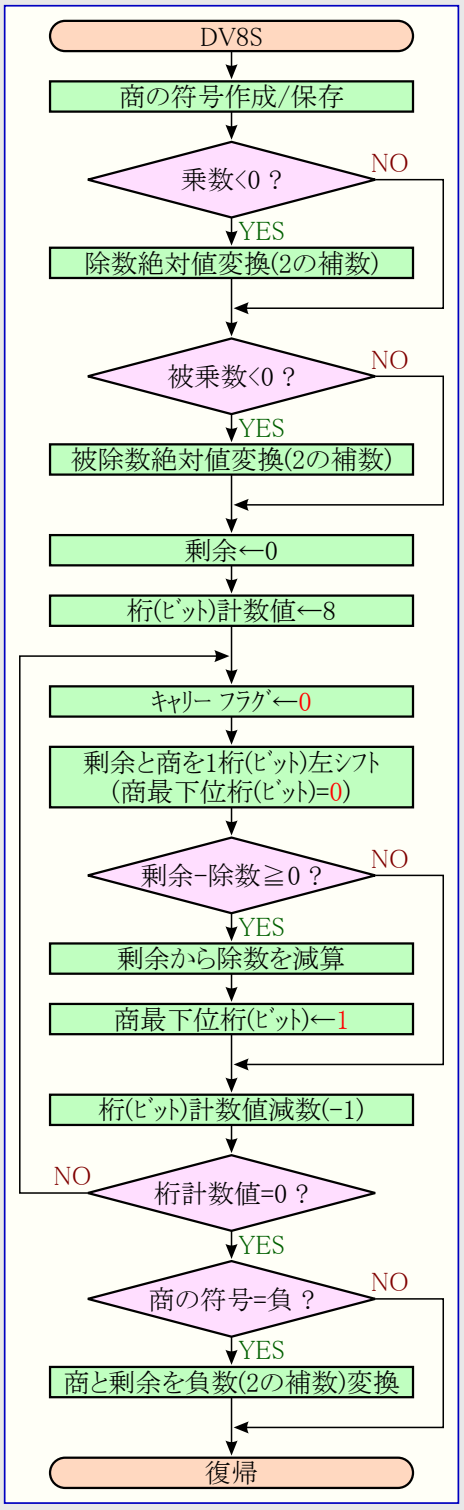
表7-1. "DV8S"使用ﾚｼﾞｽﾀ

ﾚｼﾞｽﾀ	入力	内部	出力
R15		D8S:商符号	
R16	DV8Y:被除数		DV8L:商
R17			DV8H:剰余
R18	DV8X:除数		
R19		CNT:桁(ﾋﾞｯﾄ)計数値	

表7-2. "DV8S"性能値

項目	値
ｺｰﾄﾞ容量(語)	20+RET
実行時間	72～83+RET
使用ﾚｼﾞｽﾀ	下位ﾚｼﾞｽﾀ=1, 上位ﾚｼﾞｽﾀ=4, ﾎﾟｲﾝﾀ=0
使用割り込み	なし
使用周辺機能	なし

図7-1. DV8S流れ図



8. 16÷16=16(16) 符号なし除算 - DV16U

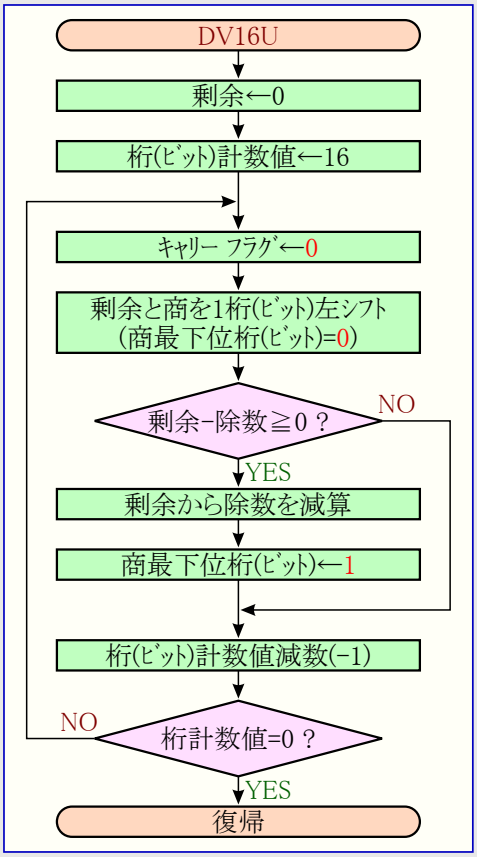
両方のファイルが符号なし16ビット除算を実行する"DV16U"と呼ばれるルーチンを含みます。両実装は同じ算法を基にします。けれども、コード量最適化実装は繰り返しコードを使用し、それに反して速度最適化コードは一直線のコード実装です。図8-1.はコード量最適化版用の流れ図を示します。

8.1. 算法内容

コード量最適化用の算法は次の通りです。

1. 剰余を空に、キャリー フラグを解除(0)します。
2. 桁(ビット)計数値に16を設定します。
3. 現桁の商を0に仮設定し、被除数と商を1桁(ビット)左にシフトします。
4. 現在の剰余から除数が減算可能ならば減算し、現桁の商を1にします。
5. 桁(ビット)計数値を減数(-1)します。
6. 桁(ビット)計数値が0でなければ3.からを繰り返します。

図8-1. DV16U流れ図(コード量最適化実装)



8.2. 使用法

"DV16U"の使用法は両実装に関して同じで、以下の手続きで記述されます。

1. 被除数(割られる数)を16ビット レジスタ変数DV16YH:DV16YLに格納してください。
2. 除数(割る数)を16ビット レジスタ変数DV16XH:DV16XLに格納してください。
3. "DV16U"を呼び出してください。
4. 商はDV16RH:DV16RLで、DV16MH:DV16MLで剰余が得られます。

8.3. 性能

表8-1. "DV16U"使用レジスタ(コード量最適化実装)

レジスタ	入力	内部	出力
R16	DV16YL:被除数下位バイト		DV16RL:商下位バイト
R17	DV16YH:被除数上位バイト		DV16RH:商上位バイト
R18			DV16ML:剰余下位バイト
R19			DV16MH:剰余上位バイト
R20	DV16XL:除数下位バイト		
R21	DV16XH:除数上位バイト		
R22		CNT: 桁(ビット)計数値	

表8-2. "DV16U"性能値 (コード量最適化実装)

項目	値
コード容量(語)	15+RET
実行時間	178～210+RET
使用レジスタ	下位レジスタ=0, 上位レジスタ=7, ポインタ=0
使用割り込み	なし
使用周辺機能	なし

表8-3. "DV16U"使用レジスタ(速度最適化実装)

レジスタ	入力	内部	出力
R16	DV16YL:被除数下位バイト		DV16RL:商下位バイト
R17	DV16YH:被除数上位バイト		DV16RH:商上位バイト
R18			DV16ML:剰余下位バイト
R19			DV16MH:剰余上位バイト
R20	DV16XL:除数下位バイト		
R21	DV16XH:除数上位バイト		

表8-4. "DV16U"性能値 (速度最適化実装)

項目	値
コード容量(語)	177+RET
実行時間	145～177+RET
使用レジスタ	下位レジスタ=0, 上位レジスタ=6, ポインタ=0
使用割り込み	なし
使用周辺機能	なし

9. 16÷16=16(16) 符号付き除算 - DV16S

このｻﾌﾞﾙｰﾁﾝは符号付き16ﾋﾞｯﾄ除算を実装します。この実装はｺｰﾄﾞ量最適化です。負の場合、入力値は2の補数形式で表されるでしょう。

9.1. 算法内容

符号付き16÷16除算用の算法は次の通りです。

1. 除数と被除数の符号から商の符号を作成(XOR)して保存します。
2. 除数が負数なら、絶対値にします。(2の補数)。
3. 被除数が負数なら、絶対値にします。(2の補数)。
4. 剰余を空に、ｷｬﾘｰ ﾌﾗｸﾞを解除(0)します。
5. 桁(ﾋﾞｯﾄ)計数値に16を設定します。
6. 現桁の商を0に仮設定し、被除数と商を1桁(ﾋﾞｯﾄ)左にｼﾌﾄします。
7. 現在の剰余から除数が減算可能ならば減算し、現桁の商を1にします。
8. 桁(ﾋﾞｯﾄ)計数値を減数(-1)します。
9. 桁(ﾋﾞｯﾄ)計数値が0でなければ6.からを繰り返します。
10. 商の符号が負なら、商と剰余を負数(2の補数)に変換します。

9.2. 使用法

"DV16S"の使用法は以下の手続きで記述されます。

1. 被除数(割られる数)を16ﾋﾞｯﾄ ﾚｼﾞｽﾀ変数DV16YH:DV16YLに格納してください。
2. 除数(割る数)を16ﾋﾞｯﾄ ﾚｼﾞｽﾀ変数DV16XH:DV16XLに格納してください。
3. "DV16S"を呼び出してください。
4. 商はDV16RH:DV16RLで、DV16MH:DV16MLで剰余が得られます。

9.3. 性能

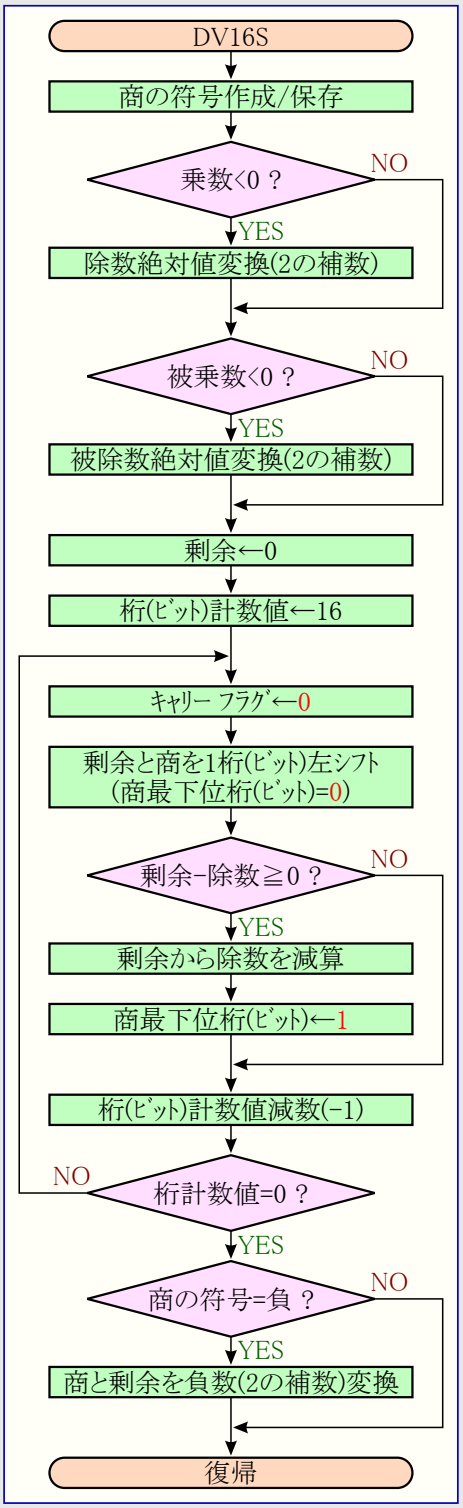
表9-1. "DV16S"使用ﾚｼﾞｽﾀ

ﾚｼﾞｽﾀ	入力	内部	出力
R15		D8S:商符号	
R16	DV16YL:被除数下位ﾊﾞｲﾄ		DV16RL:商下位ﾊﾞｲﾄ
R17	DV16YH:被除数上位ﾊﾞｲﾄ		DV16RH:商上位ﾊﾞｲﾄ
R18			DV16ML:剰余下位ﾊﾞｲﾄ
R19			DV16MH:剰余上位ﾊﾞｲﾄ
R20	DV16XL:除数下位ﾊﾞｲﾄ		
R21	DV16XH:除数上位ﾊﾞｲﾄ		
R22		CNT:桁(ﾋﾞｯﾄ)計数値	

表9-2. "DV16S"性能値

項目	値
ｺｰﾄﾞ容量(語)	39+RET
実行時間	187～234+RET
使用ﾚｼﾞｽﾀ	下位ﾚｼﾞｽﾀ=1, 上位ﾚｼﾞｽﾀ=7, ﾎﾟｲﾝﾀ=0
使用割り込み	なし
使用周辺機能	なし

図9-1. DV16S流れ図



本社

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131, USA
TEL 1(408) 441-0311
FAX 1(408) 487-2600

国外営業拠点

Atmel Asia
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
TEL (852) 2245-6100
FAX (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-Yvelines
Cedex
France
TEL (33) 1-30-60-70-00
FAX (33) 1-30-60-71-11

Atmel Japan
104-0033 東京都中央区
新川1-24-8
東熱新川ﾋﾞﾙ 9F
ｱﾄﾒﾙ ｼﾞｬﾊﾟﾝ株式会社
TEL (81) 03-3523-3551
FAX (81) 03-3523-7581

製造拠点

Memory
2325 Orchard Parkway
San Jose, CA 95131, USA
TEL 1(408) 441-0311
FAX 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131, USA
TEL 1(408) 441-0311
FAX 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3
France
TEL (33) 2-40-18-18-18
FAX (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex
France
TEL (33) 4-42-53-60-00
FAX (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
TEL 1(719) 576-3300
FAX 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR
Scotland
TEL (44) 1355-803-000
FAX (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn
Germany
TEL (49) 71-31-67-0
FAX (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
TEL 1(719) 576-3300
FAX 1(719) 540-1759

Biometrics
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex
France
TEL (33) 4-76-58-47-50
FAX (33) 4-76-58-47-60

文献請求
www.atmel.com/literature

本応用記述はATMELのAVR200応用記述(doc0936.pdf Rev.0936C-05/06)の翻訳日本語版です。日本語では不自然となる重複する形容表現は省略されている場合があります。日本語では難解となる表現は大幅に意訳されている部分もあります。必要に応じて一部加筆されています。頁割の変更により、原本より頁数が少なくなっています。

必要と思われる部分には( )内に英語表記や略称などを残す形で表記しています。

青字の部分はﾘﾝｸとなっています。一般的に赤字の0,1は論理0,1を表します。その他の赤字は重要な部分を表します。