選択課題プログラムとして、時計(2)をタイマ割り込み機能を用いて実装した。

ソフトウェア実験1  
個人レポート

C過程 S-15組 1TE20137W 2022/11/18

柳 鷹

作成したプログラムは以下である。Step9までのグループ課題であったエコーバックプログラムを編集しているため、修正箇所はハイライトしている。

|  |
| --- |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* 各種レジスタ定義  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* レジスタ群の先頭  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  .equ REGBASE, 0xFFF000 | DMAP を使用.  .equ IOBASE, 0x00d00000  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* 割り込み関係のレジスタ  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  .equ IVR, REGBASE+0x300 | 割り込みベクタレジスタ  .equ IMR, REGBASE+0x304 | 割り込みマスクレジスタ  .equ ISR, REGBASE+0x30c | 割り込みステータスレジスタ  .equ IPR, REGBASE+0x310 | 割り込みペンディングレジスタ  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* タイマ関係のレジスタ  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  .equ TCTL1, REGBASE+0x600 | タイマ1コントロールレジスタ  .equ TPRER1, REGBASE+0x602 | タイマ1プリスケーラレジスタ  .equ TCMP1, REGBASE+0x604 | タイマ1コンペアレジスタ  .equ TCN1, REGBASE+0x608 | タイマ1カウンタレジスタ  .equ TSTAT1, REGBASE+0x60a | タイマ1ステータスレジスタ  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* UART1 (送受信)関係のレジスタ  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  .equ USTCNT1, REGBASE+0x900 | UART1 ステータス / コントロールレジスタ  .equ UBAUD1, REGBASE+0x902 | UART1 ボーコントロールレジスタ  .equ URX1, REGBASE+0x904 | UART1 受信レジスタ  .equ UTX1, REGBASE+0x906 | UART1 送信レジスタ  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* LED  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  .equ LED7, IOBASE+0x000002f | ボード搭載のLED用レジスタ  .equ LED6, IOBASE+0x000002d | 使用法については付録 A.4.3.1  .equ LED5, IOBASE+0x000002b  .equ LED4, IOBASE+0x0000029  .equ LED3, IOBASE+0x000003f  .equ LED2, IOBASE+0x000003d  .equ LED1, IOBASE+0x000003b  .equ LED0, IOBASE+0x0000039  \*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*システムコール番号  \*\*\*\*\*\*\*\*\*\*\*\*\*\*  .equ SYSCALL\_NUM\_GETSTRING, 1  .equ SYSCALL\_NUM\_PUTSTRING, 2  .equ SYSCALL\_NUM\_RESET\_TIMER, 3  .equ SYSCALL\_NUM\_SET\_TIMER, 4  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* 初期化  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  .section .text  .even  boot:  \* スーパーバイザ & 各種設定を行っている最中の割込禁止  move.w #0x2000,%SR  lea.l SYS\_STK\_TOP, %SP | Set SSP  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* 割り込みコントローラの初期化  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  move.b #0x40, IVR | ユーザ割り込みベクタ番号を  | 0x40+level に設定.  move.l #0x00ff3ff9,IMR | 全割り込みマスクMUART=>0,MTMR1=>1  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* 送受信 (UART1) 関係の初期化 ( 割り込みレベルは 4 に固定されている )  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  move.l #UART1\_interrupt, 0x110 | 受信割り込みベクタをセット  move.w #0x0000, USTCNT1 | リセット  move.w #0xe108, USTCNT1 | 送受信可能 , パリティなし , 1 stop, 8 bit,  | 受信割り込み許可, 送信割り込み禁止  move.w #0x0038, UBAUD1 | baud rate = 230400 bps  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* タイマ関係の初期化 ( 割り込みレベルは 6 に固定されている )  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  move.w #0x0004, TCTL1 | restart, 割り込み不可 ,  | システムクロックの 1/16 を単位として計時,  | タイマ使用停止  move.l #COMPARE\_INTERPUT, 0x118 /\* level 6 \*/  move.l #SYS\_CALL, 0x080 /\*SYS\_CALLの割り込みベクタ設定\*/  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* キューの初期化  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  lea.l top0, %a2  lea.l top1, %a3  move.l %a2, out0  move.l %a3, out1  move.l %a2, in0  move.l %a3, in1  move.l #0, s0  move.l #0, s1    bra MAIN    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*プログラム領域  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  MAIN:  \*\*走行モードとレベルの設定(「ユーザモード」への移行処理)  move.w #0x0000, %SR | USER MODE, LEVEL 0  lea.l USR\_STK\_TOP,%SP | user stackの設定  \*\*システムコールによるRESET\_TIMERの起動  move.l #SYSCALL\_NUM\_RESET\_TIMER, %D0  trap #0  \*\*システムコールによるSET\_TIMERの起動  move.l #SYSCALL\_NUM\_SET\_TIMER, %D0  move.w #100, %D1  move.l #SELECT,%D2  trap #0    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* sys\_GETSTRING, sys\_PUTSTRINGのテスト  \*ターミナルの入力をエコーバックする  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  LOOP:  bra LOOP    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* 選択課題ゾーン  \*\* 時計(2)  \*\* プログラムを実行してからの時間を  \*\* ターミナルとLEDを"MM:SS.ff"という形で表示する  \*\* CS1, CS2 -> 小数第2位、第1位  \*\* SS1, SS1 -> 秒数1の位、10の位  \*\* MM1, MM2 -> 分数1の位、10の位  \*\* MMSSFF -> ターミナル表示用メモリ領域  \*\* BACK -> ターミナル表示用ヘッダ(\rTIME = )  \*\* COLON -> :(0x3a)  \*\* PERIOED -> .(0x2e)  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  SELECT:  movem.l %D0-%D7/%A0-%A6,-(%SP) | レジスタ退避  cmpi.b #0x3a,CS1 | CS1カウンタで10回実行したかどうか数える  beq CS1KILL | 0.10秒経っていたら繰り上げ  SECOND:  cmpi.b #0x3a,CS2 | CS2カウンタで10回実行したかどうか数える  beq CS2KILL | 1.00秒経っていたら繰り上げ  TENSECOND:  cmpi.b #0x3a,SS1 | SS1カウンタで10回実行したかどうか数える  beq SS1KILL | 10.00秒経っていたら繰り上げ  MINUTE:  cmpi.b #0x36,SS2 | SS2カウンタで6回実行したかどうか数える  beq SS2KILL | 60.00秒経っていたら繰り上げ  TENMINUTE:  cmpi.b #0x3a,MM1 | MM1カウンタで10回実行したかどうか数える  beq MM1KILL | 1.00分経っていたら繰り上げ  ENDCLOCK:  cmpi.b #0x36,MM2 | MM2カウンタで6回実行したかどうか数える  beq MM2KILL | 60.00分経っていたらタイマストップ  COUNT:  move.l #SYSCALL\_NUM\_PUTSTRING,%D0 | コール番号格納  move.l #0, %D1 | ch = 0  move.l #BACK, %D2 | p = #BACK  move.l #8, %D3 | size = 8  trap #0 | 呼び出し  lea.l MMSSFF, %a0 | a0レジスタにMMSSFFのメモリ番地  move.b MM2, (%a0)+ | 分数10の位代入  move.b MM1, (%a0)+ | 分数1の位代入  move.b COLON, (%a0)+ | :代入  move.b SS2, (%a0)+ | 秒数10の位代入  move.b SS1, (%a0)+ | 秒数1の位代入  move.b PERIOD, (%a0)+ |　.代入  move.b CS2, (%a0)+ | 小数第1位代入  move.b CS1, (%a0) | 小数第2位代入  move.l #SYSCALL\_NUM\_PUTSTRING,%D0 | コール番号格納  move.l #0, %D1 | ch = 0  move.l #MMSSFF, %D2 | p = #MMSSFF  move.l #8, %D3 | size = 8  trap #0 | システム呼び出し  /\* c秒 \*/  move.b CS1, %d7 | CS1をd7レジスタへ代入  move.b %d7, LED0 | LED0を変化  /\* 10c秒 \*/  move.b CS2, %d7 | CS2をd7レジスタへ代入  move.b %d7, LED1 | LED1を変化  /\* 秒 \*/  move.b SS1, %d7 | SS1をd7レジスタへ代入  move.b %d7, LED3 | LED3を変化  /\* 10秒 \*/  move.b SS2, %d7 | SS2をd7レジスタへ代入  move.b %d7, LED4 | LED4を変化  /\* 分 \*/  move.b MM1, %d7 | MM1をd7レジスタへ代入  move.b %d7, LED6 | LED6を変化  /\* 10分 \*/  move.b MM2, %d7 | CS1をd7レジスタへ代入  move.b %d7, LED7 | MM2を変化  /\* カウントアップ \*/  addi.b #1,CS1 | CS1カウンタを1つ増やして  bra SELECTEND | サブルーチン終了へ  CS1KILL:  move.b #0x30, CS1 | 小数第2位を0に  add.b #0x01, CS2 | 小数第1位に繰り上げ  bra SECOND  CS2KILL:  move.b #0x30, CS2 | 小数第1位を0に  add.b #0x01, SS1 | 秒数1の位に繰り上げ  bra TENSECOND  SS1KILL:  move.b #0x30, SS1 | 秒数1の位を0に  add.b #0x01, SS2 | 秒数10の位に繰り上げ  bra MINUTE  SS2KILL:  move.b #0x30, SS2 | 秒数10の位を0に  add.b #0x01, MM1 | 分数1の位に繰り上げ  bra TENMINUTE  MM1KILL:  move.b #0x30, MM1 | 分数1の位を0に  add.b #0x01, MM2 | 分数10の位に繰り上げ  bra ENDCLOCK  MM2KILL:  move.b #0x30, CS1 | 小数第2位を0に  move.b #0x30, CS2 | 小数第1位を0に  move.b #0x30, SS1 | 秒数1の位を0に  move.b #0x30, SS2 | 秒数10の位を0に  move.b #0x30, MM1 | 分数1の位を0に  move.b #0x30, MM2 | 分数10の位を0に  move.b #'M', LED7 | LED7をMに点灯  move.b #'M', LED6 | LED6をMに点灯  move.b #':', LED5 | LED5を:に点灯  move.b #'S', LED4 | LED4をSに点灯  move.b #'S', LED3 | LED3をSに点灯  move.b #'.', LED2 | LED2を.に点灯  move.b #'f', LED1 | LED1をfに点灯  move.b #'f', LED0 | LED0をfに点灯  move.l #SYSCALL\_NUM\_RESET\_TIMER,%D0 | タイマリセット  trap #0 　 | システムコール呼び出し  SELECTEND:  movem.l (%SP)+,%D0-%D7/%A0-%A6 | レジスタ復帰  rts | サブルーチン呼び出し前へ戻る  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*タイマのテスト  \* ’\*\*\*\*\*\*’を表示し改行する．  \*５回実行すると，RESET\_TIMERをする．  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  TT:  movem.l %D0-%D7/%A0-%A6,-(%SP)  cmpi.w #5,TTC | TTCカウンタで5回実行したかどうか数える  beq TTKILL | 5回実行したら，タイマを止める  move.l #SYSCALL\_NUM\_PUTSTRING,%D0  move.l #0, %D1 | ch = 0  move.l #TMSG, %D2 | p = #TMSG  move.l #8, %D3 | size = 8  trap #0  addi.w #1,TTC | TTCカウンタを1つ増やして  bra TTEND |そのまま戻る  TTKILL:  move.l #SYSCALL\_NUM\_RESET\_TIMER,%D0  trap #0  TTEND:  movem.l (%SP)+,%D0-%D7/%A0-%A6  rts  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* COMPARE\_INTERPUT: タイマ用のハードウェア割り込みインターフェース  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  COMPARE\_INTERPUT:  movem.l %d0, -(%sp) /\* d0退避 \*/  move.w TSTAT1, %d0 /\* TSTATの値をd0へ \*/  btst #0, %d0 /\* タイマ1ステータスレジスタの0ビット目が0か、否か \*/  beq COMPARE\_END /\* 0ならコンペアイベントなし、つまりカウンタ値とコンペアレジスタ値が異なる \*/  move.w #0x0000, TSTAT1 /\* タイマ1ステータスレジスタを0クリア \*/  jsr CALL\_RP /\* CALL\_RPを呼び出す \*/  COMPARE\_END:  movem.l (%sp)+, %d0 /\* d0復帰 \*/  rte  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* UART1\_interrupt  \*\* 送受信割り込みを扱うインターフェース  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  /\* btst : 指定データの指定ビットが0であるか判断し、0であればCCRのZをセット \*/  UART1\_interrupt:  movem.l %d1-%d3, -(%SP)  /\* 受信FIFOが空でないとき(URX[13]==1)受信割り込みであると判断 \*/  /\* URX[13] -> 0: 受信FIFOが空, 1: 受信FIFOが空でない \*/  move.w URX1, %d3  move.b %d3, %d2 | %d3.wの下位8bitを%d2.bにコピー  btst.l #13, %d3 | 13ビット目は受信FIFOにデータが存在するか  beq CALL\_INTERPUT | if URX1[13] == 0 (受信FIFOが空のとき)  move.l #0, %d1 | ch = 0 を明示  jsr INTERGET | 受信割り込み時処理  bra END\_interrupt  CALL\_INTERPUT:  /\* 送信FIFOがに空のとき(UTX[15]==1)送信割り込みであると判断 \*/  /\* UTX[15] -> 0: 送信FIFOが空でない, 1: 送信FIFOが空 \*/  btst.l #15, UTX1 | 15ビット目は送信FIFOが空であるか  beq END\_interrupt | if UTX1[15] == 0 (送信FIFOが空でないとき終了)  move.l #0, %d1 | ch = 0 を明示  jsr INTERPUT | 送信割り込み時処理  END\_interrupt:  movem.l (%SP)+, %d1-%d3  rte  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* INTERGET 受信割り込みルーチン  \*\* 引数 : %d1.l = チャネル(ch)  \*\* %d2.b = 受信データdata  \*\*戻り値　なし  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  INTERGET:  move.l %d0, -(%SP)  cmp #0, %d1 | ch = 0 であるか確認  bne END\_INTERGET  move.l #0, %d0 | %d0 = 受信キュー  move.b %d2, %d1 | %d1 = 受信したデータ  jsr IN\_Q | %d0 <= 成功したか否か  END\_INTERGET:  move.l (%SP)+, %d0  rts  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* INTERPUT : 送信割り込み時の処理  \*\* 引数 : %d1.l = チャネル(ch)  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  INTERPUT:  move.l %d1, -(%SP)  move.w #0x2700, %SR | 走行レベルを７に設定  cmpi.l #0, %d1 | ch = 0 を確認  bne END\_INTERPUT | if ch != 0 => 復帰  move.l #1, %d0  jsr OUT\_Q | %d1.b = data  cmpi #0, %d0 | %d0(OUTQの戻り値) == 0(失敗)  bne TX\_DATA | if so => 送信割り込みをマスク(真下)  move.w #0xe108, USTCNT1  bra END\_INTERPUT  TX\_DATA:  add.w #0x0800, %d1 | ヘッダを付与  move.w %d1, UTX1  END\_INTERPUT:  move.l (%SP)+, %d1  rts  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* INQ  \*\*入力キュー番号,d0.l 書き込むデータ,d1.b  \*\*出力 d0,成功1, 失敗0  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  IN\_Q:  cmp.b #0x00, %d0 |受信キュー、送信キューの判別  bne i\_loop1  jsr INQ0  rts  i\_loop1:  jsr INQ1  rts  INQ0:  move.w %sr, -(%sp) |レジスタ退避  movem.l %a0-%a4,-(%sp)  move.w #0x2700, %SR |走行レベルを7に設定  move.l s0, %d0 |s=256 => %d0=0:失敗  sub.l #0x100, %d0  beq i0\_Finish |s=256 => 復帰  movea.l in0, %a1 |書き込み先アドレス=%a1  move.b %d1, (%a1)+ |データをキューへ入れる,書き込み先アドレスを更新  lea.l bottom0, %a2 |次回書き込みアドレスa1<キューデータ領域の末尾アドレスa2=>step1  cmp.l %a2, %a1  bls i0\_STEP1  lea.l top0, %a3 |in=top  move.l %a3, %a1  i0\_STEP1:  move.l %a1, in0 |in更新  add.l #1, s0 |s+1  move.l #1, %d0 |d0=1 =>成功  i0\_Finish:  movem.l (%sp)+, %a0-%a4 |レジスタ復帰  move.w (%sp)+, %sr  rts |サブルーチン復帰  INQ1:  move.w %sr,-(%sp) |レジスタ退避  movem.l %a0-%a4,-(%sp)  move.w #0x2700, %SR |走行レベルを7に設定  move.l s1, %d0 |s=256 => %d0=0:失敗  sub.l #0x100, %d0  beq i1\_Finish |s=256 => 復帰  movea.l in1, %a1 |書き込み先アドレス=%a1  move.b %d1, (%a1)+ |データをキューへ入れる,書き込み先アドレスを更新  lea.l bottom1, %a2 |次回書き込みアドレスa1<キューデータ領域の末尾アドレスa2=>step1  cmp.l %a2, %a1  bls i1\_STEP1  lea.l top1, %a3 |in=top  move.l %a3, %a1  i1\_STEP1:  move.l %a1, in1 |in更新  add.l #1, s1 |s+1  move.l #1, %d0 |d0=1 =>成功  i1\_Finish:  movem.l (%sp)+, %a0-%a4 |レジスタ復帰  move.w (%sp)+, %sr  rts |サブルーチン復帰  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* OUTQ  \*\*入力:キュー番号:d0.l  \*\*出力:d0:0失敗, d0:1成功  \*\*取り出したデータ:d1.b  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  OUT\_Q:  cmp.b #0x00, %d0 |受信キュー、送信キューの判別  bne o\_loop1  jsr OUTQ0  rts  o\_loop1:  jsr OUTQ1  rts  OUTQ0:  move.w %sr,-(%sp) |レジスタ退避  movem.l %a0-%a4,-(%sp)  move.w #0x2700, %SR |走行レベルを7に設定  move.l s0, %d0 |s=0 => %d0=0:失敗  cmp.l #0x00, %d0  beq o0\_Finish |s=0 => 復帰  movea.l out0, %a1 |取り出し先アドレス=%a1  move.b (%a1)+, %d1 |キューからデータを取り出し(%d1),取り出し先アドレスを更新  lea.l bottom0, %a2 |次回取り出すアドレスa1<キューデータ領域の末尾アドレスa2=>step1  cmp.l %a2, %a1  bls o0\_STEP1  lea.l top0, %a3 |out=top  move.l %a3, %a1  o0\_STEP1:  move.l %a1, out0 |out更新  sub.l #1, s0 |s--  move.l #1, %d0 |d0=1 =>成功  o0\_Finish:  movem.l (%sp)+, %a0-%a4 |レジスタ復帰  move.w (%sp)+, %sr  rts |サブルーチン復帰  OUTQ1:  move.w %sr,-(%sp)  movem.l %a0-%a4,-(%sp) |レジスタ退避  move.w #0x2700, %SR |走行レベルを7に設定  move.l s1, %d0 |s=0 => %d0=0:失敗  cmp.l #0x00, %d0  beq o1\_Finish |s=0 => 復帰  movea.l out1, %a1 |取り出し先アドレス=%a1  move.b (%a1)+, %d1 |キューからデータを取り出し(%d1),取り出し先アドレスを更新  lea.l bottom1, %a2 |次回取り出すアドレスa1<キューデータ領域の末尾アドレスa2=>step1  cmp.l %a2, %a1  bls o1\_STEP1  lea.l top1, %a3 |out=top  move.l %a3, %a1  o1\_STEP1:  move.l %a1, out1 |out更新  sub.l #1, s1 |s--  move.l #1, %d0 |d0=1 =>成功  o1\_Finish:  movem.l (%sp)+, %a0-%a4 |レジスタ復帰  move.w (%sp)+, %sr  rts |サブルーチン復帰  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* PUTSTRING 送信割り込みの処理  \*\* 引数 : %d1.l = チャネル(ch)  \*\* %d2.l = データ読み込み先の先頭アドレスp いったんa6にさせて  \*\* %d3.l = 送信するデータ数size  \*\* 出力 : %d0.l = 取り出した要素数  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  PUTSTRING:  cmp #0, %d1 | ch = 0 であるか確認  bne END\_PUTSTRING | そうでなければ復帰  move.w #0, %d4 | sz = 0 (取った要素数)  move.l %d2, %a5 | i = %d2 = 読み込み先 address  cmp #0, %d3 | 取り出すべきサイズが０であるか確認  beq END2\_PUTSTRING | 0であれば復帰  LOOP1\_PUTSTRING:  cmp %d4, %d3 | 取るべき要素数と取った要素数を比較  beq END3\_PUTSTRING | 同等であれば復帰  move.b #1, %d0 | %d0 = 1 (キューの番号：送信キュー)  move.b (%a5), %d1 | %d1 = 読み込んだ値  jsr IN\_Q  cmp #0, %d0 | IN\_Qの復帰値が成功（１）であるか確認  beq END3\_PUTSTRING | 失敗ならば復帰  add #1, %d4 | sz ++  add #1, %a5 | i ++  bra LOOP1\_PUTSTRING  END3\_PUTSTRING:    move.w #0xe10c, USTCNT1 | 送信割り込みを許可（アンマスク）  END2\_PUTSTRING:  move %d4, %d0 | 返り値　%d0 = sz (取った要素数)  END\_PUTSTRING:  rts    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* GETSTRING 受信割り込みの処理  \*\* 引数 : %d1.l = チャネル(ch)  \*\* %d2.l = データ書き込み先の先頭アドレスp  \*\* %d3.l = 取り出すデータ数size  \*\* 出力 :　%d0.l = 実際に取り出したデータ数  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  GETSTRING:  cmp #0, %d1 | ch = 0 であるか確認  bne END\_GETSTRING | そうでなければ復帰  move.w #0, %d4 | sz = 0 (取った要素数)  move.l %d2, %a5 | i = %d2 = 書き込み先 address  LOOP1\_GETSTRING:  cmp %d4, %d3 | 取るべき要素数と取り出した要素数を比較  beq END2\_GETSTRING | 同等であれば復帰  move.l #0, %d0 | %d0 = 0 (キューの番号：受信キュー)  jsr OUT\_Q | OUT\_Q ==> %d0: success?, %d1: 取り出したデータ  cmp #0, %d0 | OUT\_Qの復帰値が成功(1)であるか確認  beq END2\_GETSTRING | 失敗ならば復帰  move.b %d1, (%a5)+ | 書き込み先にデータを書き込み  add #1, %d4 | sz ++  bra LOOP1\_GETSTRING  END2\_GETSTRING:  move %d4, %d0 | 返り値 %d0 = sz (実際に取り出したデータ数)  END\_GETSTRING:  rts  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* RESET\_TIMER(): タイマ割り込み→不可、タイマ→停止  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  RESET\_TIMER:  move.w #0x0004, TCTL1 /\* タイマ1コントロールレジスタに0x0004を設定→割り込み不可、(SYSCLK/16選択)、タイマ禁止 \*/  rts    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* SET\_TIMER(t,p): タイマ割り込み時に呼び出すルーチン設定  \*\* タイマ割り込み周期tを設定（t \* 0.1 msec毎に割り込み発生）  \*\* タイマ使用およびタイマ割り込み  \*\* 引数 : t→%d1.w: タイマの発生周期  \*\* p→%d2.l 割り込み時に起動するルーチンの先頭アドレス  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  SET\_TIMER:  move.l %d2, task\_p /\* 割り込み時に起動するルーチンの先頭アドレスpを大域変数task\_pへ \*/  move.w #0206, TPRER1 /\* 0.1msec進むとカウンタが1増えるようにする \*/  move.w %d1, TCMP1 /\* t秒周期に設定 \*/  move.w #0x0015, TCTL1 /\* タイマ1コントロールレジスタに0x0015を設定→割り込み許可、(SYSCLK/16選択)、タイマ許可 \*/  move.b #'M', LED7  move.b #'M', LED6  move.b #':', LED5  move.b #'S', LED4  move.b #'S', LED3  move.b #'.', LED2  move.b #'f', LED1  move.b #'f', LED0  rts  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* CALL\_RP(): タイマ割り込み時に処理すべきルーチン呼び出し  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  CALL\_RP:  movem.l %a0, -(%sp)  movea.l task\_p, %a0  jsr (%a0)  movem.l (%sp)+, %a0  rts  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* システムコールインターフェース  \*\* 入力：  \*\*　　　　システムコール番号 : %d0.l  \*\*　　　　システムコールの引数 : %d1以降  \*\* 出力：  \*\*　　　　システムコール呼び出しの結果 : %d0.l  \*\* ========================================  \*\* +---+---------------+  \*\* | 1 | GETSTRING |  \*\* | 2 | PUTSTRING |  \*\* | 3 | RESET\_TIMER |  \*\* | 4 | SET\_TIMER |  \*\* +---+---------------+  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  SYS\_CALL:    CALL\_1:  cmpi.l #1, %d0 | コール番号の確認(no.1)  bne CALL\_2 | 異なれば他のコール番号の確認  jsr GETSTRING | 対応するシステムコールを呼び出し  bra END\_SYS\_CALL  CALL\_2:  cmpi.l #2, %d0  bne CALL\_3  jsr PUTSTRING  bra END\_SYS\_CALL  CALL\_3:  cmpi.l #3, %d0  bne CALL\_4  jsr RESET\_TIMER  bra END\_SYS\_CALL  CALL\_4:  cmpi.l #4, %d0  bne END\_SYS\_CALL  jsr SET\_TIMER  END\_SYS\_CALL:  rte  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* スタック領域の確保  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  .section .bss  .even  SYS\_STK:  .ds.b 0x4000 | システムスタック領域  .even  SYS\_STK\_TOP: | システムスタック領域の最後尾  task\_p: .ds.l 1 | タイマ割り込み時に起動するルーチン先頭アドレス代入用  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*初期値のあるデータ領域\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  .section .data  TMSG:  .ascii "\*\*\*\*\*\*\r\n" | \r:行頭へ(キャリッジリターン)  .even | \n:次の行へ(ラインフィード)  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* 選択課題ゾーン  \*\* CS1, CS2 -> 小数第2位、第1位  \*\* SS1, SS1 -> 秒数1の位、10の位  \*\* MM1, MM2 -> 分数1の位、10の位  \*\* MMSSFF -> ターミナル表示用メモリ領域  \*\* BACK -> ターミナル表示用ヘッダ(\rTIME = )  \*\* COLON -> :(0x3a)  \*\* PERIOED -> .(0x2e)  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  BACK:  .ascii "\rTIME = "  .even  MMSSFF:  .ascii "12:34.56"  .even  TTC:  .dc.w 0x30  .even  MM2:  .dc.b 0x30  .even  MM1:  .dc.b 0x30  .even  COLON:  .dc.b 0x3a  .even  SS2:  .dc.b 0x30  .even  SS1:  .dc.b 0x30  .even  PERIOD:  .dc.b 0x2e  .even  CS2:  .dc.b 0x30  .even  CS1:  .dc.b 0x30  .even  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\*\*初期値の無いデータ領域\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  .section .bss  BUF:  .ds.b 256 |BUF[256]  .even  USR\_STK:  .ds.b 0x4000 |ユーザスタック領域  .even  USR\_STK\_TOP: |ユーザスタック領域の最後尾  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*\* キュー用のメモリ領域確保  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  .section .data  .equ B\_SIZE, 256  top0:  .ds.b B\_SIZE-1  bottom0:  .ds.b 1  top1:  .ds.b B\_SIZE-1  bottom1:  .ds.b 1  out0:  .ds.l 1  out1:  .ds.l 1  in0:  .ds.l 1  in1:  .ds.l 1  s0:  .ds.l 1  s1:  .ds.l 1  .end |

**プログラムの内容及び工夫した点**

本プログラムではタイマ割り込み機能を用いて課題7「時計(2)」を実装した。始めの方針としては