Laboratory of Electrical Engineering and Computer Science-II

Echo back program - Group 1

Group members:

1TE18224Y	Lee Jiseok
1TE19247Y	Loa Champ Bodhibaum
1TE19248G	Nimrawee Nattapat
1TE19249R	Liu Yiluo
1TE19250T	Sihanern Thitisan
1TE20257E	Nam Non

Submission date: 2022/11/18

```
********************
** System call numbers
********************
    .equ    SYSCALL_NUM_GETSTRING, 1
    .equ    SYSCALL_NUM_PUTSTRING, 2
    .equ    SYSCALL_NUM_RESET_TIMER, 3
    .equ    SYSCALL_NUM_SET_TIMER, 4

*******************************
** Head of the Register Group
*******************************
    .equ    REGBASE, 0xFFF000 | DMAP is used.
    .equ    IOBASE, 0x00d00000
*******************************
** Registers Related to Interrupts
*******************************
    .equ    IVR, REGBASE+0x300 | Interrupt Vector Register
    .equ    IMR, REGBASE+0x304 | Interrupt Mask Register
    .equ    ISR, REGBASE+0x30c | Interrupt Status Register
    .equ    IPR, REGBASE+0x310 | Interrupt Pending Register
*******************************
** Registers Related to the Timer
*******************************
    .equ    TCTL1, REGBASE+0x600    |Timer1 Control Register
    .equ    TPRER1, REGBASE+0x602   |Timer1 Prescaler Register
    .equ    TCMP1, REGBASE+0x604    |Timer1 Compare Register
    .equ    TCN1, REGBASE+0x608     |Timer1 Counter Register
    .equ    TSTAT1, REGBASE+0x60a   |Timer1 Status Register
*******************************
** Registers Related to UART1 (Transmitter and Receiver)
*******************************
    .equ    USTCNT1, REGBASE+0x900  |UART1 Status / Control Register
    .equ    UBAUD1, REGBASE+0x902   | UART 1 Baud Control Register
    .equ    URX1, REGBASE+0x904     | UART 1 Receiver register
    .equ    UTX1, REGBASE+0x906     | UART 1 Transmitter Register
*******************************
** LED
*******************************
    .equ    LED7, IOBASE+0x000002f  | Register for LED mounted on the board
    .equ    LED6, IOBASE+0x000002d  | Refer to Appendix A.4.3.1 for a way to use
    .equ    LED5, IOBASE+0x000002b
    .equ    LED4, IOBASE+0x0000029
    .equ    LED3, IOBASE+0x000003f
    .equ    LED2, IOBASE+0x000003d
    .equ    LED1, IOBASE+0x000003b
    .equ    LED0, IOBASE+0x0000039
    .equ    PUSHSW, 0xFFF419        | Register for Push Switch mounted on the board
*************************************************************
** Reservation of the stack region
*************************************************************
.section .bss
.even
SYS_STK:
    .ds.b   0x4000  | System stack region
    .even
SYS_STK_TOP:        | End of the system stack region
*************************************************************
** Initialization
```

```
** A specific value has been set to internal device registers.
** Refer to each register specification in Appendix B to know the above reason.
*****************************************************************
.section .text
.even
boot:
* Prohibit an interrupt into the supervisor and during performing various settings.
    move.w  #0x2700, %SR        | run at lv.0
    lea.l   SYS_STK_TOP, %SP    | Set SSP
*****************************
**Initialization of the interrupt controller
*****************************
    move.b  #0x40, IVR                  | Set the user interrupt vector number to 0x40+level.
    move.l  #0x00ff3ff9, IMR            | Allow UART1 and timer interrupts
    move.l  #SYSCALL, 0x080             | Set the interrupt for system call TRAP #0
    move.l  #UART1_INTERRUPT, 0x110     | Set the interrupt subroutine for level 4 interrupt
    move.l  #TIMER_INTERRUPT, 0x118     | Set the interrupt subroutine for level 6 interrupt
*****************************
** Initialization related to the transmitter and the receiver (UART1)
** (The interrupt level has been fixed to 4.)
*****************************
    move.w  #0x0000, USTCNT1 | Reset
    move.w  #0xe10c, USTCNT1 | Transmission and reception possible - no parity, 1 stop, 8 bit,
         allow only tranmission interrupt
    move.w  #0x0038, UBAUD1  | baud rate = 230400 bps
************************
** Initialization related to the timer (The interrupt level has been fixed to 6.)
************************
    move.w  #0x0004, TCTL1  | Restart, an interrupt impossible
                            | Count the time with the 1/16 of the system clock
                            | as a unit
                            | Stop the timer use
    jsr     INIT
    bra     MAIN
*****************************************************************
**    Program region
*****************************************************************
MAIN:
    ** Set the running mode and the level (The process to move to 'the user mode')
    move.w  #0x0000, %SR        /*USER MODE, LEVEL 0*/
    lea.l   USR_STK_TOP, %SP    /*set user stack*/

    ** Start up RESET_TIMER by the system call
    move.l  #SYSCALL_NUM_RESET_TIMER, %d0

    trap    #0
    ** Start up SET_TIMER by the system call
    move.l  #SYSCALL_NUM_SET_TIMER, %d0
    move.w  #50000, %d1
    move.l  #TT, %d2
    trap    #0


***********************************
*    Test of sys_GETSTRING and sys_PUTSTRING
*    Echo-back the input from a terminal
***********************************

LOOP:
```

```
117     move.l  #SYSCALL_NUM_GETSTRING, %d0
118     move.l  #0, %d1         /*ch = 0*/
119     move.l  #BUF, %d2       /*p = #BUF*/
120     move.l  #256, %d3       /*size = 256*/
121     trap    #0
122     move.l  %d0, %d3        /*size = %d0 (The length of a given string)*/
123     move.l  #SYSCALL_NUM_PUTSTRING, %d0
124     move.l  #0, %d1         /*ch = 0*/
125     move.l  #BUF, %d2       /*p = #BUF*/
126     trap    #0
127     bra     LOOP
128
129 ************************************
130 *    Test of the timer
131 *    Display  '' ****** and CRLF (Carriage Return, Line Feed) five times
132 *    Do RESET_TIMER after five times of the execution
133 ************************************
134
135 TT:
136     movem.l %d0-%d7/%a0-%a6, -(%sp)
137     cmpi.w  #5, TTC         /*Count with the counter TTC whether five times of the execution
            have been performed*/
138     beq     TTKILL          /*Stop the timer after five times of the execution*/
139     move.l  #SYSCALL_NUM_PUTSTRING, %d0
140     move.l  #0, %d1         /*ch = 0*/
141     move.l  #TMSG, %d2      /*p = #TMSG*/
142     move.l  #8, %d3         /*size = 8*/
143     trap    #0
144     addi.w  #1, TTC         /*Increment TTC counter by 1 and return*/
145     bra     TTEND
146
147 TTKILL:
148     move.l  #SYSCALL_NUM_RESET_TIMER, %d0
149     trap    #0
150
151 TTEND:
152     movem.l (%sp)+, %d0-%d7/%a0-%a6
153     rts
154
155 ************************************************************
156 **  System Call Interface:
157 **  Maker: Sihanern Thitisan
158 **  Reviewer: Loa Champ, Nimrawee Nattapat
159 ************************************************************
160
161 SYSCALL:
162     cmpi.l  #SYSCALL_NUM_GETSTRING,%d0      | if %d0 == 1
163     beq     JUMP_GETSTRING                  | Jump to the subroutine for GETSTRING
164     cmpi.l  #SYSCALL_NUM_PUTSTRING,%d0      | if %d0 == 2
165     beq     JUMP_PUTSTRING                  | Jump to the subroutine for PUTSTRING
166     cmpi.l  #SYSCALL_NUM_RESET_TIMER,%d0    | if %d0 == 3
167     beq     JUMP_RESET_TIMER                | Jump to the subroutine for RESET_TIMER
168     cmpi.l  #SYSCALL_NUM_SET_TIMER,%d0      | if %d0 == 4
169     beq     JUMP_SET_TIMER                  | Jump to the subroutine for SET_TIMER
170     rte
171
172 JUMP_GETSTRING:
173     jsr    GETSTRING
174     rte
```

```
JUMP_PUTSTRING:
    jsr    PUTSTRING
    rte
JUMP_RESET_TIMER:
    jsr    RESET_TIMER
    rte
JUMP_SET_TIMER:
    jsr    SET_TIMER
    rte


*************************************************************
**  Timer interrupt
**  Maker: Nimrawee Nattapat, Loa Champ
**  Reviewer: Sihanern Thitisan, Nam Non
*************************************************************
TIMER_INTERRUPT:
    movem.l %a0, -(%sp)         | Save the registers
    cmp.w   #1, TSTAT1          | Check 0th bit of TSTAT1 to see if the cycle count has reached compare value
    beq     TSTAT1_reset        | If last bit = 1 and timeer interrupt is occuring, jump to TSTAT1_reset
    jmp     Go_back

RESET_TIMER:
    move.w  #0x0004,    TCTL1   | Restart, an interrupt impossible |Count the time with the 1/16 of the system clock |as a unit |Stop the timer use
    rts

SET_TIMER:
    move.w  #0x0ce, TPRER1      | Set TPRER1 to a value that allows one cycle to be 0.1 ms
    move.w  %d1, TCMP1          | Move compare value input in d1 to TCMP1
    move.w  #0x0015, TCTL1      | Enable timer
    move.l  %d2, task_p         | move head address of interupt task to a variable called task_p
    rts

Go_back:
    movem.l (%sp)+, %a0         | Restore the registers
    rte

TSTAT1_reset:
    move.w  #0, TSTAT1          | Reset TSTAT1 back to 0
    jsr     Call_rp             | Jump to Call_rp to perform task_p
    jmp     Go_back

Call_rp:
    move.l  (task_p), %a0
    jsr     (%a0)               | jump to the address in task_p
    rts

*************************************************************
**  UART1 Interrupt
**  Maker: Sihanern Thitisan, Lee Jiseok
**  Reviewer: Loa Champ, Nimrawee Nattapat
*************************************************************
UART1_INTERRUPT:
    movem.l %d1-%d4, -(%sp)
    move.w  URX1, %d3
```

```
230     btst.l  #13, %d3                | Check if the 13th bit of URX1 is 1
231     bne     RECEIVER_INTERRUPT      | If the 13th bit is 1, it is a receiver interrupt
232     move.w  UTX1, %d3
233     btst.l  #15, %d3                | Check if the 15th bit of UTX1 is 1
234     bne     TRANSMITTER_INTERRUPT
235     bra     UART1_INTERRUPT_END
236
237 TRANSMITTER_INTERRUPT:
238     move.l  #0, %d1                 | Move 0 to %d1
239     jsr     INTERPUT                | Jump to INTERPUT subroutine
240     bra     UART1_INTERRUPT_END
241 RECEIVER_INTERRUPT:
242     move.l  #0, %d1
243     move.b  %d3, %d2                | Prepare arguments for INTERGET
244     jsr     INTERGET                | Jump to INTERGET subroutine
245 UART1_INTERRUPT_END:
246     movem.l (%sp)+, %d1-%d4
247     rte
248
249 ****************************************************************
250 **  INTERGET
251 **  Maker: Liu Yiluo, Nam Non
252 **  Reviewer: Lee Jiseok
253 ****************************************************************
254 INTERGET:
255     cmpi.l  #0, %d1                     | compare the ch, it should be 0
256     beq     INTERGET_INQ
257     rts
258
259 INTERGET_INQ:
260     movem.l %d0-%d2, -(%sp)
261     move.l  #0, %d0                     | 0 means the first queue, the reception queue
262     move.b  %d2, %d1
263     jsr     INPUT_QUEUE                 | jump to the INQ
264     movem.l (%sp)+, %d0-%d2
265     rts
266
267 ****************************************************************
268 **  INTERPUT
269 **  Maker: Lee Jiseok
270 **  Reviewer: Liu Yiluo
271 ****************************************************************
272 INTERPUT:
273     movem.l %d0-%d1/%a1, -(%sp)
274     move.w  %sr, -(%sp)
275     move.w  #0x2700, %sr        /* runlevel->7 */
276     cmpi    #0, %d1
277     bne     INTERPUT_END        /* chならば，何もせずに復帰!=0 */
278 INTERPUT_START:
279     moveq.l #1, %d0             /* to use Queue for transmission */
280     jsr     OUTPUT_QUEUE        /* d0:success of fail, d1:que's output */
281
282     cmpi    #0, %d0
283     beq     INTERPUT_FAIL
284
285 INTERPUT_SUCCESS:
286     move.w  #0x0800, %d2
287     move.b  %d1, %d2
288     move.w  %d2, UTX1
```

```
289     bra     INTERPUT_END
290 INTERPUT_FAIL:
291     andi    #0xfff8, USTCNT1    /* if outq failed */
292 INTERPUT_END:
293     move.w  (%sp)+, %sr
294     movem.l (%sp)+, %d0-%d1/%a1
295     rts
296 
297 ************************************************************
298 **  PUTSTRING
299 **  Maker: Liu Yiluo, Champ Loa, Nimrawee Nattapat
300 **  Reviewer: Lee Jiseok
301 ************************************************************
302 PUTSTRING:
303     cmpi.l  #0, %d1         /*≠ifch0,return without doing anything.*/
304     beq PUTSTRING_INIT
305     rts
306 
307 PUTSTRING_INIT:
308     movem.l %d1/%d7/%a1-%a6, -(%sp)
309     /*sz->0,i->p*/
310     move.l  #0, size_put        | init
311     move.l  size_put, %d7
312     move.l  %d2, ptr_put        | head address p
313     cmp     #0, %d3
314     beq     PUTSTRING_LOOP
315 
316 PUTSTRING_LOOP:
317     cmp     %d3, %d7            | sz = size?
318     beq     PUTSTRING_UNMASK
319 
320     moveq.l #1, %d0         /*Execute INQ(1,p[i]) and write in data at the address i into the
        transmitting queue*/
321     move.l  ptr_put, %a6
322     move.b  (%a6), %d1
323     jsr     INPUT_QUEUE
324     cmp     #0, %d0         /*if it failed or full go to unmask*/
325     beq     PUTSTRING_UNMASK
326 
327     /* sz++,i++ */
328     addq.l  #1, %d7             | update
329     addq.l  #1, %a6             | update
330     move.l  %d7, size_put
331     move.l  %a6, ptr_put
332     bra     PUTSTRING_LOOP
333 
334 PUTSTRING_UNMASK:
335     ori.w   #0xe107, USTCNT1    /*Permit the transmitter interrupt (unmask) manipulating
        USTCNT1.*/
336     bra     PUTSTRING_RETURN
337 
338 PUTSTRING_RETURN:
339     move.l  size_put, %d0       /*%D0 <-sz*/
340     movem.l (%sp)+, %d1/%d7/%a1-%a6
341     rts
342 
343 ************************************************************
344 **  GETSTRING
345 **  Maker: Liu Yiluo
```

```
346 **  Reviewer: Lee Jiseok
347 *************************************************************
348 GETSTRING:
349     cmpi.l  #0, %d1                             | compare the ch with 0
350     beq     GETSTRING_INIT
351     rts
352
353 GETSTRING_INIT:
354     movem.l %d1/%d7/%a1-%a6, -(%sp)
355     move.l  #0, size_get                        | init the sz
356     move.l  size_get, %d7
357     move.l  %d2, ptr_get
358     move.l  ptr_get, %a6                        | store the p at a6, the head address
359     bra     GETSTRING_LOOP
360
361 GETSTRING_LOOP:
362     cmp     %d3, %d7                            | compare whether the sz reaches the size
363     beq     GETSTRING_RETURN
364
365     moveq.l #0, %d0                             | 0 is the first queue
366     jsr     OUTPUT_QUEUE
367     cmp     #0, %d0                             | to check the output of OUTQ
368     beq     GETSTRING_RETURN
369
370     move.b  %d1, (%a6)                          | copy the data to the address
371     addq.l  #1, %d7
372     addq.l  #1, %a6
373     move.l  %d7, size_get                       | update
374     move.l  %a6, ptr_get                        | update
375     bra     GETSTRING_LOOP
376
377 GETSTRING_RETURN:
378     move.l  size_get, %d0
379     movem.l (%sp)+, %d1/%d7/%a1-%a6
380     rts
381
382 *************************************************************
383 ** Queues
384 **  Maker: Liu Yiluo, Lee Jiseok
385 **  Reviewer: Lee Jiseok, Liu Yiluo
386 *************************************************************
387 INIT:
388     movem.l %a1, -(%sp)
389     lea.l   top_0, %a1      /*top address is a1*/
390     move.l  %a1, in_0
391     move.l  %a1, out_0
392     move.w  #0, s_0
393
394     lea.l   top_1, %a1      /*tomove.w %sr, -(%sp)p address is a1*/
395     move.l  %a1, in_1
396     move.l  %a1, out_1
397     move.w  #0, s_1
398     movem.l (%sp)+, %a1
399     rts
400
401 INPUT_QUEUE:
402     movem.l %d3-%d5/%a1-%a6,-(%sp)
403     move.w  %SR, %d5
404     move.w  #0x2700, %SR    /*runlevel->7*/
```

```
405
406     /*check the no of the que*/
407     cmpi.b  #0, %d0
408     bne     INPUT_Q1
409     bra     INPUT_Q0
410
411 INPUT_Q0:               /*routine for the Queue no.0*/
412     lea.l   top_0, %a1      /*top address is a1*/
413     lea.l   bottom_0, %a2       /*bottom address is a2*/
414     move.l  in_0, %a3       /*pointer in->a3*/
415     move.l  out_0, %a4      /*pointer out->a4*/
416     move.w  s_0, %d3        /*s->d3*/
417     jsr     INQ
418
419     /*after the subroutine update the varibles*/
420     move.w  %d3, s_0
421     move.l  %a3, in_0
422     move.w  %d5, %SR
423     movem.l (%sp)+,%d3-%d5/%a1-%a6
424     rts
425
426 INPUT_Q1:
427     lea.l   top_1, %a1      /*top address is a1*/
428     lea.l   bottom_1, %a2       /*bottom address is a2*/
429     move.l  in_1, %a3       /*pointer in->a3*/
430     move.l  out_1, %a4      /*pointer out->a4*/
431     move.w  s_1, %d3        /*s->d3*/
432     jsr     INQ
433
434     /*after the subroutine update the varibles*/
435     move.w  %d3, s_1
436     move.l  %a3, in_1
437     move.w  %d5, %SR
438     movem.l (%sp)+,%d3-%d5/%a1-%a6
439     rts
440
441 INQ:
442     cmp.w   #256, %d3
443     bne     INQ_SUCC        /*if s not equals to 256*/
444     bra     INQ_FAIL        /*if s==256*/
445
446 INQ_SUCC:
447     move.b  %d1, (%a3)
448
449     cmp     %a2, %a3
450     beq     INQ_BACK        /*reach the bottom*/
451     bra     INQ_NEXT
452
453 INQ_NEXT:                   | move to the next address
454     addq    #1, %a3
455     addi.w  #1, %d3
456     move.l  #1, %d0
457     rts
458
459 INQ_BACK:                   | go back to the start of the queue
460     addi.w  #1, %d3
461     move.l  %a1, %a3
462     move.l  #1, %d0
463     rts
```

```
464
465 INQ_FAIL:
466     move.l  #0, %d0
467     rts
468
469
470
471
472 OUTPUT_QUEUE:
473     movem.l %d3-%d5/%a1-%a6,-(%sp)
474     move.w  %SR, %d5
475     move.w  #0x2700, %SR    /*runlevel->7*/
476     cmpi.b  #0, %d0         /*check the no of the que*/
477     bne     OUTPUT_Q1
478     bra     OUTPUT_Q0
479
480 OUTPUT_Q0:
481     lea.l   top_0, %a1      /*top address is a1*/
482     lea.l   bottom_0, %a2       /*bottom address is a2*/
483     move.l  out_0, %a4      /*pointer out<-a4*/
484     move.w  s_0, %d3
485     jsr     OUTQ
486
487     /*after the subroutine update the varibles*/
488     move.w  %d3, s_0
489     move.l  %a4, out_0
490
491     move.w  %d5, %SR
492     movem.l (%sp)+,%d3-%d5/%a1-%a6
493     rts
494
495 OUTPUT_Q1:
496     lea.l   top_1, %a1      /*top address is a1*/
497     lea.l   bottom_1, %a2       /*bottom address is a2*/
498     move.l  out_1, %a4      /*pointer out<-a4*/
499     move.w  s_1, %d3
500     jsr     OUTQ
501
502     /*after the subroutine update the varibles*/
503     move.w  %d3, s_1
504     move.l  %a4, out_1
505
506     move.w  %d5, %SR
507     movem.l (%sp)+,%d3-%d5/%a1-%a6
508     rts
509
510 OUTQ:
511     cmp.w   #0, %d3
512     bne     OUTQ_SUCC
513     bra     OUTQ_FAIL
514
515 OUTQ_SUCC:
516     move.b  (%a4), %d1
517
518     cmp     %a2, %a4
519     beq     OUTQ_BACK       /*reach the bottom*/
520     bra     OUTQ_NEXT
521
522 OUTQ_NEXT:              | move to the next address
```

```
    addq    #1, %a4
    subi.w  #1, %d3
    move.l  #1, %d0
    rts

OUTQ_BACK:              | back to the start address
    move.l  %a1, %a4
    subi.w  #1, %d3
    move.l  #1, %d0
    rts

OUTQ_FAIL:
    move.l  #0, %d0
    rts

.section .data
    .equ    SIZE_of_QUEUE,  256

.section .bss
top_0:      .ds.b   SIZE_of_QUEUE-1
bottom_0:   .ds.b   1
in_0:       .ds.l   1
out_0:      .ds.l   1
s_0:        .ds.w   1

top_1:      .ds.b   SIZE_of_QUEUE-1
bottom_1:   .ds.b   1
s_1:        .ds.w   1
in_1:       .ds.l   1
out_1:      .ds.l   1

size_put:   .ds.l   1
ptr_put:    .ds.l   1
size_get:   .ds.l   1
ptr_get:    .ds.l   1

task_p:     .ds.l 1
            .even

*****************************************************************
**  Data region with an initial value
*****************************************************************
.section .data
TMSG:       .ascii  "******\r\n"
            .even
TTC:        .dc.w   0
            .even

*****************************************************************
**  Data region without an initial value
*****************************************************************
.section .bss
BUF:        .ds.b   256
            .even
USR_STK:
            .ds.b   0x4000
            .even
USR_STK_TOP:
```