EXAMPLE SORT

OBJECTIVE
This example has the following objectives:

- Review the use of keystroke commands for controlling a process
- Introduce the concept of ‘multiple sort’ and its ‘sequential sort’ equivalent

PROGRAM EX_SORT
This program can be used for controlling a process through keystroke commands send from the terminal keyboard. Once the keystroke command is received, it is echoed to the terminal monitor for confirmation. Then, the keystroke in the command is compared with a set of command options, and action is appropriately taken. The program recognizes five keystroke commands:

i) Move forward    >
ii) Move backward    <
iii) Increase speed (decrease delay)    +
iv) Decrease speed (increase delay)    -
v) Stop program    S

The program use is intended for stepper motor control. A stepper motor is controlled by sending binary patterns (step sequences) to its controller board. Eight distinct binary patterns (S0 – S7) are recognized by the stepper motor controller board in your lab. When these patterns are hit in increasing order, one after the other, the motion is called ‘half speed forward’. If only the odd sequences are hit (i.e., every second pattern), the motion is ‘full speed forward’. If the patterns are hit in decreasing order, the motion is, respectively, ‘half-speed backward’ and ‘full speed backward’. The corresponding step sizes are, respectively, +1, +2, -1, -2 (Table 1). Using two’s complement 8-bit signed numbers convention, these values correspond to the variable STEP taking hex values $01, $02, $ff, $fe.

Table 1 Stepper motor speed definitions

<table>
<thead>
<tr>
<th>Step size</th>
<th>STEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Half speed forward</td>
<td>+1</td>
</tr>
<tr>
<td>Full speed forward</td>
<td>+2</td>
</tr>
<tr>
<td>Half-speed backward</td>
<td>-1</td>
</tr>
<tr>
<td>Full speed backward</td>
<td>-2</td>
</tr>
</tbody>
</table>

The "half-speed" and "full-speed" commands are part of the coarse speed control. Fine speed control can be attained by modifying the delay inserted between steps. A shorter delay ensures a higher speed, while a longer delay produces a slower speed. The shortest delay that can be physically accommodated by the stepper-motor dynamics is 5 ms. For a 2 MHz processor (0.5 \(\mu\)s/cycle), this corresponds to 10,000 ($2710) cycles. The maximum delay is, obviously, $ffff. In the program, we use the single precision variable, DELAY. When two numbers, $2710 and $ffff are rounded to 2 significant hex numbers, one gets $2800 and $ff00 (To maintain delay integrity, the first number was rounded up, the second number was rounded down.) Hence, the single precision variable, DELAY, can take values between $28 and $ff.
The following is the ‘big-picture’ of the program:

a) Define control variables: STEP, DELAY

b) Initialize program:
   i) Initialize X=REGBAS
   ii) Initialize SCI
   iii) Initialize control variables STEP=0, DELAY=$ff

c) Verify if new keystroke command has arrived. If N, jump directly to e)

d) "Echo" the command back to the terminal monitor for confirmation

e) Sort and process the keystroke command. Modify stepper motor controls based on the
   keystroke command:
   vi) > STEP=$01
   vii) <STEP=$ff
   viii) + Increase speed (decrease delay, but not below $28)
   ix) - Decrease speed (increase delay but not above $ff)
   x) S Stop program

f) Loop back to c)

In the initialization phase, the program enters the values of S0 – S7 given in Table 1 in the
appropriate memory locations. Then it sets STEP=$00, and POINTER=$00

FLOWCHART AND CODE

The program flowchart is show below. Two flowchart levels are presented: the big-picture and
the details. The big-picture is used to understand the overall architecture of the program. The
details are used to explain some of the blocks. (Details are given only for those blocks which are
somehow new, and have not been used in previous programs.) The essential code for this
program is shown to the right of the flowchart. The essential code was incorporated into the
standard template asm to generate the code file Ex_Sort.asm. Please refer to this file for more
coding details, and coding style.
Big-picture Flowchart

Flowchart Details

Code

```assembly
ORG DATA
STEP RMB 1
DELAY RMB 1

START
LDX #REGBAS
LDAA #%00110000
STAA BAUD,X
LDAA #%00000000
STAA SCI
LDAA #%00001100
STAA SCCR2,X
LDAA #0
STAA STEP
LDAA #$ff
STAA DELAY

LABEL0 NOP
LABEL1 LDAA SCSR,X
ANDA #%00100000
BEQ LABEL9
LDAA SCDR,X
LABEL2 LDAA SCSR,X
ANDA #%10000000
BEQ LABEL2
STAB SCDR,X
CMPB #'>'
BEQ FORWD
CMPB #'<
BEQ BACKWD
CMPB #'+'
BEQ FASTER
CMPB #'-
BEQ SLOWER
CMPB #'S'
BEQ STOP
JMP LABEL9

FORWD LDAA STEP
CMPA #+1
BGT LABEL10
INCA
LABEL10 STAA STEP
JMP LABEL9

BACKWD LDAA STEP
CMPA #-1
BLT LABEL11
DECA
LABEL11 STAA STEP
JMP LABEL9

FASTER LDAA DELAY
CMPA #$28
BLS LABEL6

LABEL6 STAA DELAY
JMP LABEL9

SLOWER LDAA DELAY
CMPA #$ff
BHS LABEL7

LABEL7 STAA DELAY
JMP LABEL9

STOP SWI
LABEL9 BRA LABEL0
```
EXECUTION
Open THRSim11. Close the Commands window. Open and assemble Ex_Sort.asm. View CPU registers, serial registers SCSR (set to binary) and SCDR, timer registers TCNT through TOC1, memory list for variables STEP and DELAY, serial transmitter, serial receiver. Arrange windows for maximum benefit. Reset registers. Set standard labels (Label/Set Standard Labels). Press the RESET button. Your screen should look like this:

a) Set breakpoints at the following lines:
   - LDAB SCDR,X
   - STAB SCDR,X
   - SWI

b) Step through the program up to LABEL0. The program should initialize X=$1000, SCI control registers, and STEP=$00, DELAY=$ff.

c) Let the program run. It should loop continuously through the body of the program and back to LABEL0, etc.
i) **CHECK THE KEYSTROKE COMMAND > (FORWARD)**

d) Type > in serial transmitter, and send. After some time, you will see the RDRF bit in SCSR (bit 5) getting set, and your program should stop at the breakpoint. At this point, SCDR-$3e$, i.e., ASCII code for >. Your program looks like this:
e) Run. The program clears the RDRF flag, and stops at the next breakpoint, STAB SCDR,X.

f) Step through your program. You will see
   • The program enters the sort sequence
   • The program branches to label FORWARD
   • The BGT condition is not satisfied, and the program increments STEP from $00 to $01. You have set the STEP to the value +1, for 'slow-forward'
   • The program branches to LABEL9 and then to LABEL0. Your screen looks like this:

g) Let your program run. You will see the program looping again in the big loop.

h) Send the character > again. The sequence of events described above repeats itself. At the end of the process, STEP=2. You have achieved ‘fast-forward’.
i) Let your program run. You will see the program looping again in the big loop. Send the character > again and repeat the sequence of events. Observe carefully how the segment FORWD is executed. You will notice that the condition BGT is now satisfied, and the program branches over the INCA instruction. Thus, your STEP does no longer increase, and stays at its maximum value, STEP=$02 (decimal, +2). Your screen looks like this:
ii) **CHECK THE KEystROKE COMMAND < (BACKWARD)**

j) Run. While the program is looping on LABEL0, type the character < into the serial transmitter, and send. The program should stop at the first breakpoint. Run again. The program stops at second breakpoint.

k) Step through the program. The program will go through the sorting sequence and end up at the BACKWD segment. The BLT condition is not satisfied, and the program decrements STEP from $02 to $01. You are back to ‘slow-forward’ At the end of this process, your screen looks like this:
l) Repeat and get the variable STEP to get the value STEP=$00. This corresponds to ‘stand still’.

m) Repeat. This time the variable STEP will be decremented to the value STEP=$ff (decimal –1). This corresponds to ‘slow-backward’.

n) Repeat. This time the variable STEP will be decremented to the value STEP=$fe (decimal –2). This corresponds to ‘fast-backward’. Your screen should look like this:
iii) **CHECK THE KEYSTROKE COMMAND + (INCREASE SPEED)**

- We want to test the keystroke command +. This command should produce an increase in speed, i.e., a decrease in DELAY from $ff to $fe. Run the program. While the program is looping on LABEL0, type the character. After the breakpoints, you will see your program sorting at the program segment FASTER. While going through this segment, the condition for BLS is not satisfied, and program decrements DELAY to the value DELAY=$fe. At the end of the process, your screen should look like this:
p) Repeat two times. The variable DELAY should become DELAY=$fd, then DELAY=$fc. Your program should look like this:
q) Test now the logic at the lower limit of DELAY, i.e., that it does not go below $28. Make \( \text{DELAY} = 28 \). Then, run. Send + again. After the second breakpoint, step through it. The program gets again to the program segment FASTER, but, this time, the condition for BLS is satisfied and the program branches over the operation DECA. Hence, the variable delay is not decremented any further. It stays at \( \text{DELAY} = 28 \). The logic works as expected. Your screen should look like this:
iv) **CHECK THE KEYSTROKE COMMAND -(DECREASE SPEED)**

r) We want to test the keystroke command -. This command should produce a decrease in speed, i.e., an increase in DELAY from $28$ to $29$. Run the program. While the program is looping on LABEL0, type the character – and send. After the breakpoints, you will see your program sorting at the program segment SLOWER. While going through this segment, the condition for BHS is not satisfied, and program increments DELAY from $28$ to $29$. At the end of the process, your screen should look like this:

s) Repeat two times. The variable DELAY should become DELAY=$2a$, then DELAY=$2b$. Your program should look like this:
t) Test now the logic at upper limit of DELAY, i.e., that it does not go over $ff. Make DELAY=$ff. Then, run. Send - again. After the second breakpoint, step through it. The program gets again to the program segment SLOWER, but, this time, the condition for BHS is satisfied and the program branches over the operation INCA. Hence, the variable delay is not incremented any further. It stays at DELAY=$ff. The logic works as expected. Your screen should look like this:
v) **CHECK THE KEYSTROKE COMMAND S (STOP)**

u) We want to test the keystroke command +. This command should produce the program to stop. Run the program. While the program is looping on LABEL0, type the character S. After the breakpoints, you will see your program sorting at the program label STOP. The operation at this label is SWI. Your program successfully stops. (Do not step manually through SWI, since it will throw you outside the program area, and give a disassemble error.)

vi) **FURTHER CHECKS**

v) Remove the breakpoint from LDAB SCDR,X and STAB SCDR, X. Leave only the breakpoint at SWI.

w) Run your program and test the keystroke commands >, <, +, -, S again, until you are satisfied that you understood how they work.

**WHAT YOU HAVE LEARNED**

In this example, you have learned:

- The use of keystroke commands for controlling a process. The following keystroke commands have been used:
  - i) Move forward >
  - ii) Move backward <
  - iii) Increase speed (decrease delay) +
  - iv) Decrease speed (increase delay) -
  - v) Stop program S

- Introduce the concept of ‘multiple sort’ and its ‘sequential sort’ equivalent. The multiple sort was needed to go to one of the five possible outcomes according to the keystroke value. Since a multiple sort command is not available in the MCU programming language, an equivalent ‘sequential sort’ was used instead. The sequential sort, checks the keystroke against one possibility at a time; if a coincidence is found, it branches out. Otherwise, it continues to check.