F.9 Chapter 9 Solutions

9.1 (a) A device register is a register (or memory location) that is used for data transfer to/from an input/output device. It provides a means of communication between the processor and the input/output device. The processor can poll this register to find out whether it has received an input or it can send an output from/to the specific device that the de- vice register belongs to. In memory mapped I/O device registers are dedicated memory locations for each I/O device. There may be more than one device register (dedicated memory location) for one device.

(b) A device data register is a device register (a dedicated memory location in memory-mapped I/O) that holds the data that is to be input/output.

(c) A device status register is a device register (a dedicated memory location in memory-mapped I/O) that indicates the status of the input/output. It allows for the processor to know whether or not input/output of the value in the device data register has occurred. Basically it is an important step to achieve synchronization in an asynchronous I/O system.

9.3 The processor can accept a character every clock cycle at its maximum rate. This means that a 300 MHz processor can accept a character each 1/(300M) seconds.That is this processor can accept a character every 3.333 nanoseconds which corresponds to a rate of 18 billion characters per one minute. This is the maximum rate it can accept input in one minute. If the typist would have to type 3 billion words per minute to synchronize with this maximum rate, then a word must be 18/3 = 6 characters long. (This, of course, counts the *space* between words as one of the characters in the word!)

9.5 Bit [15] of the KBSR is the ready bit. This is used as the synchronization mechanism to let the processor know that input has occurred. If KBSR[15] is 0, no key has been struck and the value in KBDR is not valid. If KBSR[15] is 1, the value in the KBDR is the ASCII code corresponding to the last key struck.

9.7 Memory mapped and polling. The system is memory-mapped because KBSR and KBDR device registers have assigned addresses in the memory address space of the ISA. The system is polling because the Ready bit is tested to see if a key has been struck.

9.9 If KBSR[15] is 1, the data contained in the KBDR has not been read by the processor. Thus, if the keyboard hardware does not check the KBSR before writing to the KBDR, user input could be lost.

9.11 Interrupt-driven I/O is more efficient than polling. Because, in polling, the processor needs to check a specific register (or memory location) regularly to see if anything is being input or output. This consumes unnecessary processing power because the processor checks the register periodically (stopping all other jobs) even when nothing is being input or output. (Most of the time the register will not be inputting or outputting anything unless it is a really I/O-intensive program). However, in interrupt-driven I/O, when something is input or output by a device, the device sends a signal to the processor. Only when the processor receives that signal, it stops all other jobs and does the I/O. Hence, processing power is used for I/O only when it is necessary to do so.

9.13 Suppose the LC-3 datapath allows combining the two registers into one. Using separate registers, the test to see if the Ready bit is set simply involves checking bit 15 of the status register. This is performed using a branch instruction that tests if the value of the register is negative. If the

KBSR and DSR are combined, the test to see if the Display device Ready bit is set involves masking out bit 14 and testing if the bit is set or cleared. Doing it this way requires more instructions than the first method.

9.15 The most important advantage of doing I/O through a trap routine is the fact that it is not necessary for the programmer to know the gory low-level details of the specific hardware's input/output mechanism. These details include:

- the hardware data registers for the input and output devices
- the hardware status registers for the input and output devices
- the asynchronous nature of the input relative to the executing program

Besides, these details may change from computer to computer. The programmer would have to know these details for the computer she's working on in order to be able to do input/output. Using a trap routine requires no hardware-specific knowledge on part of the programmer and saves time.

9.17 (a) Some external mechanism is the only way to start the clock (hence, the computer) af- ter it is halted. The Halt service routine can never return after bit 15 of the machine control register is cleared because the clock has stopped, which means that instruction processing has stopped.

(b) STI R0, MCR This instruction clears the most significant bit of the machine control register, stopping the clock.

(c) LD R1, SaveR1

(d) The RET of the HALT routine will bring program control back to the program that executed the HALT instruction. The PC will point to the address following the HALT instruction.

9.19 Note: This problem should be corrected to read as follows:

```
                .ORIG  x3000
                LEA    R0,
                LABEL
                STR    R1, R0, #3
                TRAP   x22
                TRAP   x25
LABEL           .STRINGZ "FUNKY"
LABEL2          .STRINGZ "HELLO
WORLD"
                .END
```

Answer: FUN

9.21 If the value in A is a prime number, 1 is stored in memory location RESULT; otherwise, 0 is stored in RESULT.

9.23 Since the LC-3 ISA allows for an 8-bit trap vector, 256 service routines can be created using the current semantics of the LC-3 ISA. However, if the address specified by the TRAP instruction contained the first instruction in the service routine, the number of possible service routines would be greatly reduced. If each service routine required 16 locations, then the number of possible service routines would only be 16 (256/16=16). The semantics of the TRAP instruction could be modified as follows: Change the trap vector to 4 bits (instead of 8); zero-extend the trap vector and shift it to the left by 4 to get the starting address of the service routine.

9.25 The final values at DATA are the sorted version of the initial values at DATA in ascending order.

9.27 If the RUN latch is later set (manually), the service routine will restore the values in R0,R1, and R7 and return to the calling program. This use of the TRAP x25 instruction can be a useful tool in troubleshooting and debugging.

9.29 Error 1: The line VALUE .FILL X30000 will generate an assembly error because 0x30000 does not fit in one LC-3 memory location.

only one error in current problem statement

9.31 (a) ADD R1, R1, #1
(b) TRAP x25
(c) ADD R0, R0, #5
(d) BRzp K

9.33 (a) The keyboard interrupt is enabled, and the digit 2 is repeatedly written to the screen.
(b) The character typed is echoed twice to the screen.
(c) The digit 2 some number of times, followed by the digit typed twice or three times, followed by the digit 2 continually thereafter.
(d) The digit typed will be displayed to the screen twice or three times, depending on when the typed character interrupted the program. If the program was interrupted immediately after LD R0, B , the character typed would appear on the screen three times

9.35 For example, lets take the following program which adds 10 numbers starting at memory location x4000 and stores the result at x5000

```
.ORIG x3000
LD R1, PTR
AND R0, R0, #0
LD R2, COUNT
LOOP LDR R3, R1, #0
ADD R0, R0, R3
ADD R2, R2, #-1
BRp LOOP
STI R0, PRES
HALT
PTR .FILL x4000
PRES .FILL x5000
COUNT .FILL #10
```

If the condition codes were not saved as part of initiation of the interrupt service routine, we could end up with incorrect results. In this program, take the case when an interrupt occurred during the processing of the intruction at location x3005 and condition codes were not saved. Let R2 = 5 and hence the condition codes be P=1, N=0, Z=0 before servicing the interrupt. When control is returned to the instruction at location x3006, the BR instruction, the condition codes depend on the processing within the interrupt service routine. If they are P=0, N=0, Z=1, then the BR is not taken. This means that result stored is just the sum of the first five values, not all ten.

9.37 a) PC = x3006 Stack:

___

___

xxxxx - Saved SSP

(b) PC = x6200 Stack:

___

___

PSR of Program A - R6
x3007

xxxxx

(c) PC = x6300

Stack:

___

___

PSR for device B - R6
x6203

PSR of Program A
x3007

xxxxx

(d) PC = x6400
Stack:

___

___

PSR for device C - R6
x6311

PSR for device B
x6203

PSR of Program A
x3007

xxxxx

(e) PC = x6311
Stack:

___

___

PSR for device C
x6311

PSR for device B - R6
x6203

PSR of Program A
x3007

xxxxx

(f) PC = x6203
Stack:

___

___

PSR for device C
x6311

PSR for device B
x6203

PSR of Program A - R6
x3007

xxxxx

(g) PC = x3007
Stack:

___

___

PSR for device C
x6311

PSR for device B
x6203

PSR of Program A
x3007

xxxxx - Saved.SSP

9.39 Correction - If the buffer is full, a character has been stored in 0x40FE.

```
LDI     R0, KBDR
LDI     R1, PENDBF
LD      R2, NEGEND
ADD     R2, R1, R2
BRz     ERR         ; Buffer is full
STR     R0, R1, #0 ; Store the
character ADDR1, R1, #1
STI     R1, PENDBF ; Update next available empty
BRnzp DONE
ERR     LEA R0, MSG
PUTS
DONE RTI
KBDR    .FILL xFE02
PBUF    .FILL x4000
PNUMCH  .FILL x40FD
PENDBF  .FILL x40FF
NEGEND  .FILL xBF04 ; xBF04 = -(x40FC)
MSG     .STRINGZ   "Character cannot be accepted; input
buffer full."
```

9.41 Correction - Consider the modified interrupt handler of Exercise 10.15.

The variable “number of characters in the buffer” is shared between both the interrupt handler which is adding numbers to the buffer and the program that is removing characters. So now if the program has just loaded the number of characters in the buffers value into a register when an interrupt occurs, the value in the register is going to be stale after the interrupt is serviced. Hence when the program writes this value back to x40FD, it is writing a wrong value.

9.43 The three errors that arose in the first student’s program are:

1. The stack is left unbalanced.
2. The privilege mode and condition codes are not restored.
3. Since the value in R7 is used for the return address instead of the value that was saved on the stack, the program will most likely not return to the correct place.

9.45 (a) AND R3, R1, R2
(b) ADD R2, R2, R2
(c) ADD R2, R2, R2
(d) LDI R0, R0 ,#0

9.47 The statement of this problem is **incorrectly stated**. The correct statement is:

```
        .ORIG x2055
        ST R1, SaveR1
        ________ (a)
        TRAP x20
        LD R1, A
        ________ (b)
        TRAP x21
        ________ (c)
        LD R1, SaveR1
        RTI
A       .FILL _____ (d)
SaveR1      .BLKW 1
______      .BLKW 1   (e)
```

SOLUTION:
a) ST R0, SaveR0
b) ADD R0, R0, R1
c) LD R0, SaveR0
d) .FILL x-20
e) SaveR0

9.49 (a) LEA R0, INPUT
(b) ADD R6, R6, R0
(c) LDR R0, R6, #2
(d) .FILL S0
(e) .FILL S1
(f) .FILL x0063

9.51 Outputs 4 to the screen since cc has Z bit set before branch

9.53

Memory Address	Content
x0150	x1000

x160	x2000

Address	After cycle 100
x2FFA	x0001
x2FFB	x0010
x2FFC	x1002
x2FFD	x0404
x2FFE	x3003
x2FFF	x8201
x3000	x5020
x3001	x1025
x3002	x2207
Stack Pointer	x2FFC

9.55

MAR	**MDR**
x2000	x8000
x2C00	x1050
x2C01	x0004
x1050	xBCAE
x10FF	x2800
x2800	x2C04
x1051	x1DA6
x1052	x3C4D
x10A0	x2C0A