**Appendix F**

**Selected Solutions**

# F.5 Chapter 5 Solutions

* 1. (a) ADD
     + operate
     + register addressing for destination and source 1
     + register or immediate addressing for source 2

1. JMP
   * + control
     + register addressing
2. LEA
   * + data movement
     + immediate addressing
3. NOT
   * + operate
     + register addressing

5.3 Sentinel. It is a special element which is not part of the set of allowable inputs and indicates the end of data.

5.5 (a) Addressing mode: mechanism for specifying where an operand is located.

1. An instruction’s operands are located as an immediate value, in a register, or in memory.
2. The 5 are: immediate, register, direct memory address, indirect memory address, base

+ offset address. An immediate operand is located in the instruction. A register operand is located in a register (R0 - R7). A direct memory address, indirect memory address and base + offset address all refer to operands locate in memory.

1. Add R2, R0, R1 => register addressing mode.

1

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

2 APPENDIX F. SELECTED SOLUTIONS

5.7 01111 (decimal 15)

5.9 (a) Add R1, R1, #0 => differs from a NOP in that it sets theCC’s.

1. BRnzp #1 => Unconditionally branches to one after the next address in the PC. There- fore no, this instruction is not the same as NOP.
2. Branch that is never taken. Yes same as NOP.

5.11 No. We cannot do it in a single instruction as the smallest representable integer with the 5 bits available for the immediate field in the ADD instruction is -16. However this could be done in two instructions.

5.13 (a) 0001 011 010 1 00000 (ADD R3, R2, #0 )

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| (b) | 1001 | 011 | 011 | 111111 | (NOT | R3, | R3 ) |
|  | 0001 | 011 | 011 | 1 00001 | (ADD | R3, | R3, #1 ) |
|  | 0001 | 001 | 010 | 0 00011 | (ADD | R1, | R2, R3 ) |

(c) 0001 001 001 1 00000 (ADD R1, R1, #0 )

or

0101 001 001 1 11111 (AND R1, R1, #-1)

1. Can’t happen. The condition where N=1, Z=1 and P=0 would require the contents of a register to be both negative and zero.

(e) 0101 010 010 1 00000 (AND R2, R2, #0)

5.15 1110 001 000100000 ( LEA R1, 0x20 ) R1 <- 0x3121

0010 010 000100000 ( LD R2, 0x20 ) R2 <- Mem[0x3122] = 0x4566

1010 011 000100001 ( LDI R3, 0x20 ) R3 <- Mem[Mem[0x3123]] = 0xabcd

0110 100 010 000001 ( LDR R4, R2, 0x1 ) R4 <- Mem[R2 + 0x1] = 0xabcd

1111 0000 0010 0101 ( TRAp 0x25 )

5.17 (a) LD: two, once to fetch the instruction, once to fetch the data.

1. LDI: three, once to fetch the instruction, once to fetch the data address, and once to fetch the data.
2. LEA: once, only to fetch the instruction.

5.19 PC-64 to PC+63. The PC value used here is the incremented PC value.

5.21 The Trap instruction provides 8 bits for a trap vector. That means there could be 28 = 256 trap routines.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5.23 | x30ff | 1110 | 0010 0000 0001 | | | (LEA R1, #1) R1 <- 0x3101 |
|  | x3100 | 0110 | 010 001 00 0010 | | | (LDR R2, R1, #2) R2 <- 0x1482 |
|  | x3101 | 1111 | 0000 | 0010 | 0101 | (TRAP 0x25) |
|  | x3102 | 0001 | 0100 | 0100 | 0001 |  |
|  | x3103 | 0001 | 0100 | 1000 | 0010 |  |

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

F.5. CHAPTER 5 SOLUTIONS 3

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 5.25 | 1001 | 100 | 011 | 111111 | ;(NOT R4, R3) | | |
|  | 0001 | 100 | 100 | 1 00001 | ;(ADD R4, R4, #1) | | |
| 0001  0000 | | 001  010 | 100 0 00 010 ;(ADD  000000101 ; (BRz | | | | R1, R4, R2)  Done) |
| 0000 | | 100 | 000000001 | | ; | (BRn | Reg3) |
| 0000 | | 001 | 000000010 | | ; | (BRp | Reg2) |
| 0001 | | 001 | 011 1 00000 | | ; | (Reg3 ADD R1, R3, #0) | |
| 0000 | | 111 | 000000001 | | ; | (BRnzp Done) | |
| 0101 | | 001 | 010 1 00000 | | ; | (Reg2 ADD R1, R2, #0) | |

1111 0000 0010 0101 ; (Done TRAP 0x25)

5.27 Four different values: xAAAA, x30F4, x0000, x0005

5.29 (a) LDR R2, R1, #0 ;load R2 with contents of location pointed to by R1 STR R2, R0, #0 ;store those contents into location pointed to by R0

(b) Theconstituentmicro-opsare: MAR < − SR

MDR < − Mem[MAR] MAR < − DR Mem[MAR] < − MDR

5.31 0x1000: 0001 101 000 1 11000

5.33 It can be inferred that R5 has exactly 5 of the lower 8 bits = 1.

5.35 The IR, SEXT unit, SR2MUX, Reg File and ALU implement the ADD instruction, alongwith NZP and the logic which goes with it.

5.37 Memory, MDR, MAR, IR, PC, Reg File, the SEXT unit connected to IR[8:0], ADDR2MUX, ADDR1MUX set to PC, alongwith the ADDER they connect to, and MAXMUX and GateMARMUX implement the LDI instruction, alongwith NZP and the logic which goes with it.

5.39 IR, PC, Reg File, the SEXT unit connected to IR[8:0], ADDR2MUX, ADDR1MUX set to PC, alongwith the ADDER they connect to, and MAXMUX and GateMARMUX implement the LEA instruction, alongwith NZP and the logic which goes with it.

5.41 (a) Y is the P Condition code.

(b) Yes. X should be one wherever the opcode field of the IR matches the opcodes which change the condition code registers. The problem is that X is 1 for the BR opcode (0000) and LEA (1110) in the given logic. They should be removed, and ADD opcode (0001) should be added.

5.43 Yes, there is difference. Instruction 1 (ADD) sets the Condition Codes while Instruction 2 (BR) doesn’t.

5.45 PC is put into MAR to fetch the next instruction that needs to be executed. PC is incremented to move to the next instruction to be fetched.

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

5.47

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **11** | **10** | **01** | **00** |
| **Mux 1 D[15:12]** | 3 | 2 | 1 | 0 |
| **Mux 2 D[11:8]** | 0 | 1 | 2 | 3 |
| **Mux 3 D[7:4]** | 3 | 1 | 2 | 0 |
| **Mux 4 D[3:0]** | 2 | 3 | 1 | 0 |

|  |  |  |
| --- | --- | --- |
| **R/W** | **MDR** | **MAR** |
| W | x72A3 | 00 |
| W | x8FAF | 11 |
| R | x72A3 | 00 |
| R | xFFFF | 10 |
| W | x732D | 11 |
| R | xFFFF | 01 |
| W | x37A3 | 01 |
| R | x37A3 | 01 |
| R | x732D | 11 |

5.49 If R6 is zero, then R0 is used as a base register

Contents after accesses: 00: x72A3

01: x37A3

10: xFFFF

11: x732D

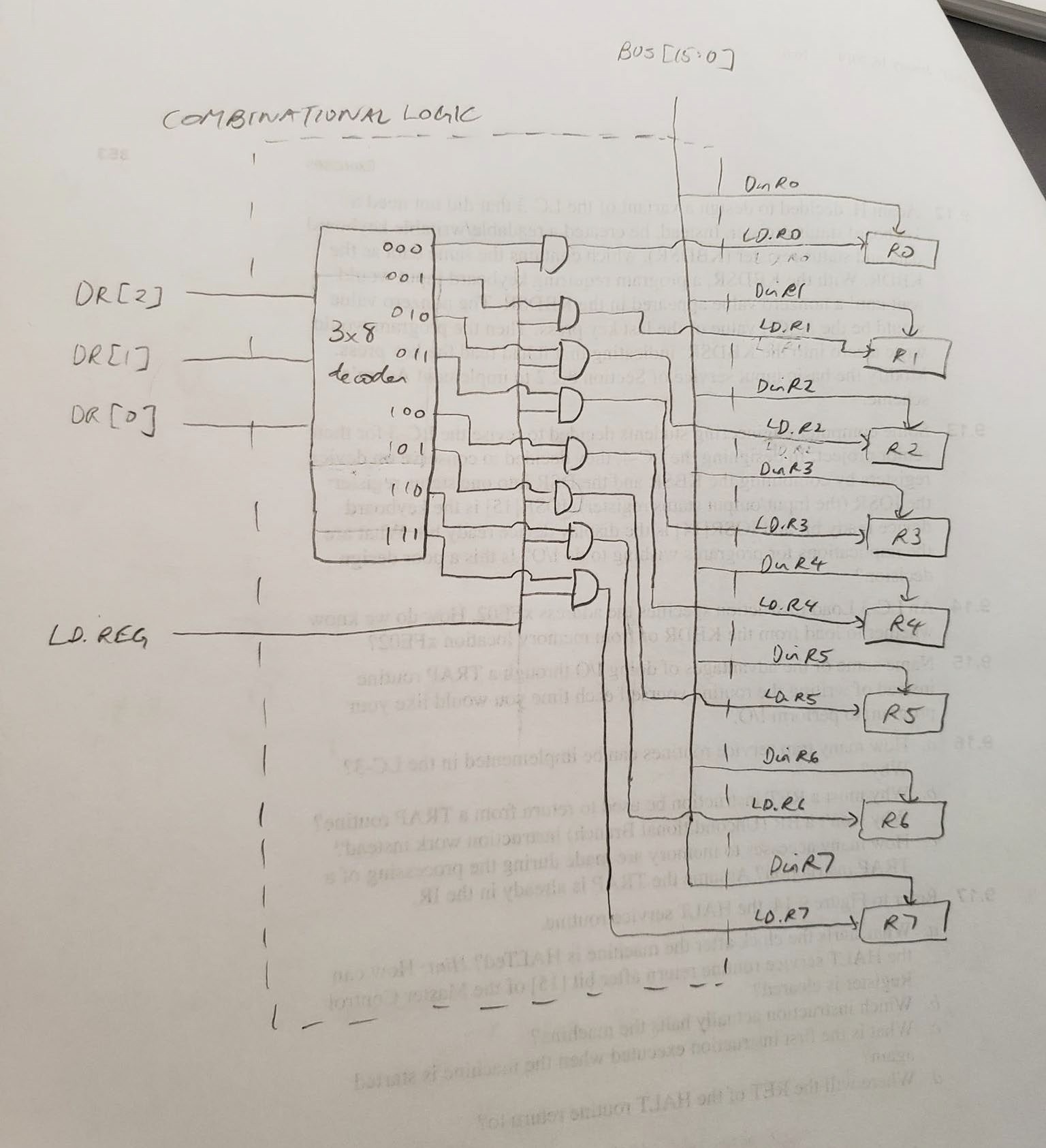
5.51 State 13: State A: State B:

ALUK = NOT SR2SEL = REGISTER\_A; ALUK = ADD;

Gate\_ALU = 1 LD.REG = 1; LD.CC = 1

5.53

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*



*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

5.55 (Same answer as Exercise 6.27. Refer to Chapter 6 solutions)

5.57 1. ST, STR, STI 2. IR[5]

3. TRAP

5.59 ADD: 9

AND: 9

LD: 15

LEA: 9

LDI: 21

NOT: 9

BRnzp: 10

TRAP: 15 + 21 = 36

5.61

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Inst # | Clock Cycle | Bus | State | Gate PC | Gate MDR | Gate ALU | Gate MARMUX | LD PC | LD MDR | LD MAR | LD CC | LD REG | DR MUX | MIO EN | R W | PC MUX |
| Inst 1 | T | x3010 | 18 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | - | - | - | PC+1 |
| T + 6 | xF0AB | 35 | x | x | x | x | x | x | x | x | x | x | x | x | x |
| T + 8 | x00AB | 15 | x | x | x | x | x | x | x | x | x | x | x | x | x |
| T + 9 | x1510 | 28 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | R7 | 1 | R | - |
| T + 10 | x1510 | 28 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | R7 | 1 | R | - |
| T + 11 | x1510 | 28 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | R7 | 1 | R | - |
| T + 12 | x1510 | 28 | x | x | x | x | x | x | x | x | x | x | x | x | x |
| T + 13 | x1510 | 28 | x | x | x | x | x | x | x | x | x | x | x | x | x |
| T + 14 | x1510 | 30 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | - | - | - | BUS |
| Inst 2 | T + 15 | x1510 | 18 | x | x | x | x | x | x | x | x | x | x | x | x | x |
| T + 21 | x2219 | 35 | x | x | x | x | x | x | x | x | x | x | x | x | x |
| T + 23 | x152A | 2 | x | x | x | x | x | x | x | x | x | x | x | x | x |
| T + 29 | x8001 | 27 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 11.9 | - | - | - |
| Inst 3 | T + 30 | x1511 | 18 | x | x | x | x | x | x | x | x | x | x | x | x | x |
| T + 36 | x0804 | 35 | x | x | x | x | x | x | x | x | x | x | x | x | x |
| Inst 4 | T + 37 | x1516 | 18 | x | x | x | x | x | x | x | x | x | x | x | x | x |
| x | x1200 | x | x | x | x | x | x | x | x | x | x | x | x | x | x |
| x | x0000 | x | x | x | x | x | x | x | x | x | x | x | x | x | x |

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

# Chapter 6 Solutions

6.1 Yes, for example, an iterative block where the test condition remains true for each iteration. This procedure will never end and is therefore not finite and not an algorithm. The following is an example of a procedure that isn’t an algorithm:

x3000 0101 000 000 1 00000 ( LOOP AND R0, R0, #0 )

x3001 0000 010 111111110 ( BRz LOOP )

This is not an algorithm because the branch instruction is always taken and the program loops indefinitely.

6.3 The following program uses DeMorgan’s Law to set the appropriate bits of the machine busy register.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| x3000 | 1010000000001110 | ( | LDI | R0, | S ) |  |  |
| x3001 | 1010001000001110 | ( | LDI | R1, | I ) |  |  |
| x3002 | 0101010010100000 | ( | AND | R2, | R2, | #0 ) | |
| x3003 | 0001010010100001 | ( | ADD | R2, | R2, | #1 ) | |
| x3004 | 0001001001111111 | ( L | ADD | R1, | R1, | #-1 ) | |
| x3005 | 0000100000000010 | ( | BRn | D ) |  |  |  |
| x3006 | 0001010010000010 | ( | ADD | R2, | R2, | R2 ) | |
| x3007 | 0000111111111100 | ( | BRnzp L | | ) |  |  |
| x3008 | 0001001010100000 | ( D | ADD | R1, | R2, | #0 | ) |
| x3009 | 1001000000111111 | ( | NOT | R0, | R0 ) | |  |
| x300a | 1001001001111111 | ( | NOT | R1, | R1 ) | |  |
| x300b | 0101000000000001 | ( | AND | R0, | R0, | R1 | ) |
| x300c | 1001000000111111 | ( | NOT | R0, | R0 ) | |  |
| x300d | 1011000000000001 | ( | STI | R0, | S ) | |  |

x300e 1111000000100101 ( TRAP x25 )

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

x300e 0100000000000001 ( S .FILL x4001 ) x300f 0100000000000000 ( I .FILL x4000 )

6.5 The three additions of 88 + 88 + 88 requires fewer steps to complete than the eighty eight additions of 3 + 3 + ... + 3. Because 88 + 88 + 88 requires fewer instructions to complete, it is faster and therefore preferable.

6.7 This program adds together the corresponding elements of two lists of numbers (vector ad- dition). One list starts at address x300e and the other starts at address x3013. The program finds the length of the two lists at memory location x3018. The first element of the first list is added to the first element of the second list and the result is stored back to the first element of the first list; the second element of the first list is added to the second element of the second list and the result is stored back to the second element of the first list; and so on.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 6.9 x3100 | 0010 | 000 0 0000 0101 | ( LD R0, Z | ) |
| x3101 | 0010 | 001 0 0000 0101 | ( LD R1, C | ) |
| x3102 | 1111 | 0000 0010 0001 | ( L TRAP x21 | ) |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| x3103 | 0001 | 001 001 1 11111 | | | ( ADD R1, R1, #-1 ) | | |
| x3104 | 0000 | 001 1 1111 1101 | | | ( BRp L ) | | |
| x3105 | 1111 | 0000 | 0010 | 0101 | ( | TRAP x25 | ) |
| x3106 | 0000 | 0000 | 0101 | 1010 | ( Z | .FILL x5A | ) |
| x3107 | 0000 | 0000 | 0110 | 0100 | ( C | .FILL #100 | ) |

6.11 This program increments each number in the list of numbers starting at address A and ending at address B. The program tells when it’s done by comparing the last address it loaded data from with the address B. When the two addresses are equal, the program stops incrementing data values.

|  |  |  |  |
| --- | --- | --- | --- |
| x3000 | 0010 | 000 011111111 | ( LD R0, x3100 ) |
| x3001 | 0010 | 001 011111111 | ( LD R1, x3101 ) |
| x3002 | 0001 | 001 001 1 00001 | ( ADD R1, R1, #1 ) |
| x3003 | 1001 | 001 001 111111 | ( NOT R1, R1 ) |
| x3004 | 0001 | 001 001 1 00001 | ( ADD R1, R1 #1 ) |
| x3005 | 0001 | 011 000 0 00 001 | ( l ADD R3, R0, R1 ) |
| x3006 | 0000 | 010 000000101 | ( BRz Done ) |
| x3007 | 0110 | 010 000 000000 | ( LDR R2, R0, #0 ) |
| x3008 | 0001 | 010010100001 | ( ADD R2, R2, #1 ) |
| x3009 | 0111 | 010000000000 | ( STR R2, R0, #0 ) |
| x300a | 0001 | 000000100001 | ( ADD R0, R0, #1 ) |
| x300b | 0000 | 111111111001 | ( BRnzp l ) |
| x300c | 1111 | 000000100101 | ( Done HALT ) |

6.13 Memory location x3011 holds the number to be right shifted. The strategy here is to im- plement a one bit right shift by shifting to the left 15 bits. The most significant bit must be carried back to the least significant bit when it’s shifted out (a circular left shift). The data to be shifted is stored at x3013. R1 is a counter to keep track of how many left shifts remain to be done.

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

|  |  |  |
| --- | --- | --- |
| x3000 | 0010000000010010 | ( LD R0, NUM ) |
| x3001 | 0101001001100000 | ( AND R1, R1, #0 ) |
| x3002 | 0001001001101111 | ( ADD R1, R1, #15 ) |
| x3003 | 0001000000100000 | ( LOOP ADD R0, R0, #0 ) |
| x3004 | 0000100000000001 | ( BRn NEG ) |
| x3005 | 0000001000000101 | ( BRp POS ) |
| x3006 | 0001000000000000 | ( NEG ADD R0, R0, R0 ) |
| x3007 | 0001000000100001 | ( ADD R0, R0, #1 ) |
| x3008 | 0001001001111111 | ( ADD R1, R1, #-1 ) |
| x3009 | 0000110000000101 | ( BRnz DONE ) |
| x300a | 0000111111111000 | ( BRnzp LOOP ) |
| x300b | 0001000000000000 | ( POS ADD R0, R0, R0 ) |
| x300c | 0001001001111111 | ( ADD R1, R1, #-1 ) |
| x300d | 0000110000000001 | ( BRnz DONE ) |
| x300e | 0000111111110100 | ( BRnzp LOOP ) |
| x300f | 0010001000000100 | (DONE LD R1, MASK ) |
| x3010 | 0101000000000001 | ( AND R0, R0, R1 ) |
| x3011 | 0011000000000001 | ( ST R0, NUM ) |
| x3012 | 1111000000100101 | ( HALT ) |
| x3013 | 1000010000100001 | ( NUM .FILL x8421 ) |
| x3014 | 0111111111111111 | ( MASK .FILL x7FFF ) |

6.15 0111 010 100 000111 ( STR R2, R4, #7 )

6.17 JSR #15

* 1. The bugs are:
     1. The instruction at x3000 should be 0010 0000 0000 1010
     2. The instruction at x3004 should be 0001 0100 1010 0100
     3. The instruction at x3008 should be 0000 1111 1111 1001
     4. The instruction at x3009 should be 0111 0100 0100 0000

6.21

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x3003 | 0001 | 0010 | 0011 | 0001 | R1 <- R0 - 15 |
| x3009 | 0001 | 0000 | 0010 | 0001 | R0 <- R0 + 1 |

6.23

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **PC** | **MAR** | **MDR** | **IR** | **R0** | **R1** |
| Before execution starts | x3000 | - | - | - | x0000 | x0000 |
| After 1st | x3001 | x3006 | xB333 | x2005 | xB333 | x0000 |
| After 2nd | x3002 | x3001 | x0601 | 0x601 | xB333 | x0000 |
| After 3rd | x3003 | x3002 | x1261 | x1261 | xB333 | x0001 |
| After 4th | x3004 | x3003 | x1000 | x1000 | x6666 | x0001 |
| After 5th | x3001 | x3004 | x0BFC | x0BFC | x6666 | x0001 |

Final values: R0 **= x9800** R1 = **6**

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

6.25 Yes, the subtract executes correctly since ADD R3, R3, #1 is 0x16E1 in machine code.

6.27

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Registers** | **Initial Value** | **After 1st** | **After 2nd** | **After 3rd** | **After 4th** | **After 5th** | **After 6th** |
| **R0** | x4006 | x4050 | x4050 | x4050 | x4050 | x4050 | x4050 |
| **R1** | x5009 | x5009 | x5009 | x5009 | x5009 | x5009 | x5009 |
| **R2** | x4008 | x4008 | x4008 | x4008 | x4008 | x4008 | xC055 |
| **R3** | x4002 | ***x4002*** | ***x8005*** | x8005 | x8005 | x8005 | x8005 |
| **R4** | x4003 | x4003 | x4003 | x4003 | x4003 | ***x4003*** | x4003 |
| **R5** | x400D | x400D | ***x400D*** | ***x400D*** | x400D | x400D | x400D |
| **R6** | x400C | x400C | x400C | x400C | x400C | x400C | x400C |
| **R7** | x6001 | x6001 | x6001 | x6001 | ***x400E*** | ***x400E*** | x400E |

|  |
| --- |
| **PC Trace** |
| *x3000* |
| *x3001* |
| *x3002* |
| x400D |
| *x5009* |
| x400E |

|  |  |
| --- | --- |
| **MAR Trace** | **MDR Trace** |
| *x3000* | *xA009* |
| *x300A* | *x3025* |
| x3025 | *x4050* |
| *x3001* | x1703 |
| *x3002* | *xC100* |
| *x400D* | x4040 |
| *x5009* | *xC1C0* |
| x400E | x1403 |

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

# Chapter 7 Solutions

7.1 0xA7FE

7.3 Using an instruction as a label confuses the assembler because it treats the label as the opcode itself so the label AND will not be entered into the symbol table. Instead the assembler will give an error in the second pass.

7.5 (a) The program calculates the product of values at addresses M0 and M1. The product is stored at address RESULT.

mem[RESULT] = mem[M0] \* mem[M1]

(b) x200C

7.7 The assembly language program is:

|  |  |  |
| --- | --- | --- |
|  | .ORIG  AND | x3000  R5, R5, #0 |
|  | ADD | R5, R5, #1 ;R5 will act as a mask to |
|  |  | ;mask out the unneeded bit |
|  | AND | R1, R1, #0 ;zero out the result register |
|  | AND | R2, R2, #0 ;R2 will act as a counter |
|  | LD | R3, NegSixt |
| MskLoop | AND | R4, R0, R5 ;mask off the bit |
|  | BRz | NotOne ;if bit is zero then don’t |
|  |  | ;increment the result |
|  | ADD | R1, R1, #1 ;if bit is one increment |
|  |  | ;the result |
| NotOne | ADD | R5, R5, R5 ;shift the mask one bit left |
|  | ADD | R2, R2, #1 ;increment counter (tells us |
|  |  | ;where we are in bit pattern) |

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

ADD R6, R2, R3

BRn MskLoop ;not done yet go back and

;check other bits

HALT

NegSixt .FILL #-16

.END

* 1. The .END pseudo-op tells the assembler where the program ends. Any string that occurs after that will be disregarded and not processed by the assembler. It is different from HALT instruction in very fundamental aspects:
     1. It is not an instruction, it can never be executed.
     2. Therefore it does not stop the machine.
     3. It is just a marker that helps the assembler to know where to stop assembling.

7.11 ; Prog 7.11

; This code does not perform error checking

; It accepts 3 characters as input

; The first one is either x or #

; The next two is the number.

.ORIG x3000

IN ; input the first char - either x or # AND R3, R3, #0

ADD R3, R3, #9 ; R3 = 9 if we are working

; with a decimal or 16 if hex

LD R4, NASCIID

LD R5, NHEXDIF

LD R1, NCONSD

ADD R1, R1, R0

BRz GETNUMS

LD R1, NCONSX

ADD R1, R1, R0

BRnp FAIL

ADD R3, R3, #6 ; R3 = 15

GETNUMS IN

ST R0, CHAR1 IN

ST R0, CHAR2

LEA R6, CHAR1

AND R2, R2, #0

ADD R2, R2, #2 ; Loop twice

; Using R2, R3, R4, R5, R6 here

AND R0, R0, #0 ; Result

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| LOOP | ADD | R1, | R3, | #0 |
|  | ADD | R7, | R0, | #0 |
| LPCUR | ADD | R0, | R0, | R7 |
|  | ADD | R1, | R1, | #-1 |

BRp LPCUR

LDR R1, R6, #0

ADD R1, R1, R4

ADD R0, R0, R1

DONECUR

ADD R1, R1, R5

BRn DONECUR

ADD R0, R0, #-7 ; for hex numbers

ADD R6, R6, #1

ADD R2, R2, #-1

|  |  |  |
| --- | --- | --- |
|  | BRp | LOOP |
| ; R0 | has number at this point |
| AND | R2, R2, #0 |
| ADD | R2, R2, #8 |
| LEA | R3, RESEND |
| LD | R4, ASCNUM |
| AND | R5, R5, #0 |
| ADD | R5, R5, #1 |
| STLP | AND | R1, R0, R5 |
|  | BRp | ONENUM |
|  | ADD | R1, R4, #0 |
|  | BRnzp | STORCH |
| ONENUM | ADD | R1, R4, #1 |
| STORCH | ADD | R5, R5, R5 |
|  | STR | R1, R3, #-1 |
|  | ADD | R3, R3, #-1 |
|  | ADD | R2, R2, #-1 |
|  | BRp | STLP |
|  | LEA | R0, RES |
|  | PUTS |  |
| FAIL | HALT |  |
| CHAR1 | .FILL | x0 |
| CHAR2 | .FILL | x0 |

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ASCNUM | .FILL | x30 |  |  |
| NHEXDIF | .FILL | xFFEF | ; | -x11 |
| NASCIID | .FILL | xFFD0 | ; | -x30 |
| NCONSX | .FILL | xFF88 | ; | -x78 |
| NCONSD | .FILL | xFFDD | ; | -x23 |

RES .BLKW 8 RESEND .FILL x0

.END

7.13 Error 1:

Line 8: ST R1, SUM

SUM is an undefined label. This error will be detected at assembly time.

Error 2:

Line 3: ADD R1, R1, R0

R1 was not initialized before it was used; therefore, the result of this ADD instruction may not be correct. This error will be detected at run time.

7.15 This program doubles all the positive numbers and leaves the negative numbers unchanged.

7.17 There is not a problem in using the same label in separate modules assuming the programmer expected the label to refer to different addresses, one within each module. This is not a problem because each module has its own symbol table associated with it. It is an error on the otherhand if the programmer expected each label AGAIN to refer to the same address.

7.19 The instruction labeled LOOP executes 4 times.

7.21 Correction: Please use the following LC-3 assembly language program for this problem:

|  |  |  |  |
| --- | --- | --- | --- |
|  | .ORIG  AND | x3000  R0, | R0, #0 |
| ADD | R2, | R0, #10 |
| LD | R1, | MASK |
| LD | R3, | PTR1 |
| LOOP | LDR | R4, | R3, #0 |
|  | AND  BRz | R4, NEXT | R4, R1 |
|  | ADD | R0, | R0, #1 |
| NEXT | ADD | R3, | R3, #1 |
|  | ADD | R2, | R2, #-1 |
|  | BRp | LOOP |  |
|  | STI | R0, | PTR2 |
|  | HALT |  |  |

|  |  |  |
| --- | --- | --- |
| MASK | .FILL | x8000 |
| PTR1 | .FILL | x4000 |
| PTR2 | .FILL | x5000 |

Solution:

The assembled program:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0101 | 0000 | 0010 | 0000 | ( AND R0, R0, #0 ) |
| 0001 | 0100 | 0010 | 1010 | ( ADD R2, R0, #10 ) |
| 0010 | 0010 | 0000 | 1010 | ( LD R1, MASK ) |
| 0010 | 0110 | 0000 | 1010 | ( LD R3, PTR1 ) |
| 0110 | 1000 | 1100 | 0000 | ( LDR R4, R3, #0 ) |
| 0101 | 1001 | 0000 | 0001 | ( AND R4, R4, R1 ) |
| 0000 | 0100 | 0000 | 0001 | ( BRz NEXT ) |
| 0001 | 0000 | 0010 | 0001 | ( ADD R0, R0, #1 ) |
| 0001 | 0110 | 1110 | 0001 | ( ADD R3, R3, #1 ) |
| 0001 | 0100 | 1011 | 1111 | ( ADD R2, R2, #-1 ) |
| 0000 | 0011 | 1111 | 1001 | ( BRp LOOP ) |
| 1011 | 0000 | 0000 | 0011 | ( STI R0, PTR2 ) |
| 1111 | 0000 | 0010 | 0101 | ( HALT ) |
| 1000 | 0000 | 0000 | 0000 |  |
| 0100 | 0000 | 0000 | 0000 |  |
| 0101 | 0000 | 0000 | 0000 |  |

This program counts the number of negative values in memory locations 0x4000 - 0x4009 and stores the result in memory location 0x5000.

7.23 (a) ADD R1, R1, #-1

1. LDR R4, R1, #0
2. ADD R0, R0, #1
3. ADD R1, R1, #-1
4. BR LOOP

7.25 his is an assembler error. The number 0xFF004 does not fit in one LC-3 memory location and therefore this .FILL cannot be assembled.

7.27 The program logical right-shifts the number in R0 by the number in R1 and puts it in RESULT.

R0 holds the input number to right-shift. Range = [x0000 to xFFFF] R1 holds the amount to right-shift. Range = [1 to 15]

R6 holds the right-shifted output. Range = [x0000 to x7FFF]

7.29 A = x1801 F = 0x1800

B = xEA67 G = x1867

C = x1867 H = x1803

D = x1802 I = x0FFD

E = x3BFE J = x1867

Instructions are: LEA R5, x67; ST R5, #-2; BRnzp #-3; ADD R4, R1, #7

7.31 The program counts the number of odd integers in the array

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

* 1. Memory access = 3 cycles

|  |  |  |
| --- | --- | --- |
| Cycle Number | State Number | Information |
| 11 | 27 | LD.REG = 1; DRMUX = 000; GateMDR = 1;  LD.CC = 1; GateALU = 0; GatePC = 0 |
| 16 | 35 | LD.MDR = 0; LD IR = 1; MDR = x2209; IR = x2009 |
| 50 | 1 | LD.REG = 1; BUS = 0x0001; MDR = x14A1; DRMUX = 010; GateMDR = 0 |
| 57 | 1 | PC = x3007; BUS = x0003; IR = x1040;  GateALU = 1; GatePC = 0 |
| 65 | 22 | ADDR1MUX = 0; ADDR2MUX = 10; LD.PC = 1; PC = x3008; PCMUX = ADDER |

* + 1. ADD R2, R2, #1
    2. ADD R0, R1, R0
    3. B .FILL #2

The student was trying to divide the value at A by the value and B and store the quotient at C. To fix the program, the *BRnzp AGAIN* should be changed to *BRp AGAIN*

7.35

|  |  |  |
| --- | --- | --- |
| Address | Content | Assembly |
| x3000 | 0101 001 001 1 00000 | AND R1, R1, #0 |
| x3001 | 0010 000 0 1111 1110 | LD R0, x3100 |
| x3002 | 0000 110 000000011 | BRnz x3006 |
| x3003 | 0001 001 001 0 00000 | ADD R1, R1, R0 |
| x3004 | 0001 000 000 1 11111 | ADD R0, R0, #-1 |
| x3005 | 0000 111 111111100 | BRnzp x3002 |
| x3006 | 0011 001 0 1111 1010 | ST R1, x3101 |
| x3007 | 1111 0000 0010 0101 | HALT |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Instruction # | PC | MAR | MDR | R0 | R1 |
| Initial | x3000 | xxxx | xxxx | xxxx | xxxx |
| 1 | x3001 | xxxx | xxxx | xxxx | x0000 |
| 2 | x3002 | x3100 | x0003 | x0003 | x0000 |
| 3 | x3003 | xxxx | xxxx | x0003 | x0000 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 | x3004 | x3003 | x1240 | x0003 | x0003 |
| 5 | x3005 | xxxx | xxxx | x0002 | x0003 |
| 9 | x3005 | xxxx | xxxx | x0001 | x0005 |
| 13 | x3005 | xxxx | xxxx | x0000 | x0006 |
| 14 | x3002 | xxxx | xxxx | x0000 | x0006 |
| 15 | x3006 | xxxx | xxxx | x0000 | x0006 |
| 16 | x3007 | x3101 | x0006 | x0000 | x0006 |
| 17 | xxxx | xxxx | xxxx | x0000 | x0006 |

7.37

|  |  |
| --- | --- |
| - | BUS |
| 1 | x3000 |
| 2 | x1263 |
| 3 | x009A |
| 4 | x3001 |
| 5 | xA000 |
| 6 | x3002 |
| 7 | x3000 |
| 8 | x1263 |
| 9 | x3002 |
| 10 | x3000 |
| 11 | x3003 |
| 12 | x1263 |
| 13 | x3003 |
| 14 | x1263 |
| 15 | x009D |

Instructions executed: ADD R1, R1, #3

LDI R0, #0

ST R0, #0

ADD R1, R1, #3

Contents after execution: R0 = 0x1263

R1 = 0x009D

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

# F.10 Chapter 8 Solutions

8.1 The defining characteristic of a stack is the unique specification of how it is to be accessed. Stack is a LIFO (Last in First Out) structure. This means that the last thing that is put in the stack will be the first one to get out from the stack.

8.3 (a) PUSH R1

1. POP R0
2. PUSH R3
3. POP R7

8.5 One way to check for overflow and underflow conditions is to keep track of a pointer that tracks the bottom of the stack. This pointer can be compared with the address of the first and last addresses of the space allocated for the stack.

;

; Subroutines for carrying out the PUSH and POP functions. This

; program works with a stack consisting of memory locations x3FFF

; (BASE) through x3FFB (MAX). R6 is the bottom of the stack.

;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| POP | ST | R1, | Save1 | ; | are needed by POP. |
|  | ST | R2, | Save2 |  |  |
|  | ST | R3, | Save3 |  |  |

LD R1, NBASE ; BASE contains -x3FFF. ADD R1, R1, #-1 ; R1 contains -x4000.

ADD R2, R6, R1 ; Compare bottom of stack to x4000 BRz fail\_exit ; Branch if stack is empty.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| LD | R1, | BASE |  | ;Iterate from the top of |
|  |  |  |  | ;the stack |
| LDI | R0, | BASE |  | ;Load the value from the |
| NOT | R3, | R6 |  | ;top of stack |
| ADD | R3, | R3, | #1 | ;Generate the |
|  |  |  |  | ;negative of the |
|  |  |  |  | ;bottom-of-stack pointer |
| ADD | R6, | R6, | #1 | ;Increment the |
|  |  |  |  | ;bottom-of-stack |
|  |  |  |  | ;pointer |

pop\_loop ADD R2, R1, R3 ;Compare iterating

;pointer to

;bottom-of-stack pointer BRz success\_exit;Branch if no more

;entries to shift

LDR R2, R1, #-1 ;Load the entry to shift STR R2, R1, #0 ;Shift the entry

ADD R1, R1, #-1 ;Increment the

;iterating pointer

BRnzp pop\_loop

PUSH ST R1, Save1 ; Save registers that ST R2, Save2 ; are needed by PUSH. ST R3, Save3

LD R1, MAX ; MAX contains -x3FFB

ADD R2, R6, R1 ; Compare stack pointer to -x3FFB BRz fail\_exit ; Branch if stack is full.

ADD R1, R6, #0 ;Iterate from the bottom

;of stack

LD R3, NBASE ;NBASE contains

;-x3FFF

ADD R3, R3, #-1 ; R3 = -x4000

push\_loop ADD R2, R1, R3 ;Compare iterating

;pointer to

;bottom-of-stack pointer BRz push\_entry ;Branch if no more

;entries to shift

LDR R2, R1, #0 ;Load the entry to shift STR R2, R1, #-1 ;Shift the entry

ADD R1, R1, #1 ;Decrement the

;iterating pointer

BRnzp push\_loop

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

push\_entry ADD R6, R6, #-1 ;Increment the

;bottom-of-stack pointer STI R0, BASE ;Push a value onto stack BRnzp success\_exit

success\_exit LD R1, Save1 ;Restore original

LD R2, Save2 ;register values LD R3, Save3

AND R5, R5, #0 ;R5 <--- success RET

fail\_exit LD R1, Save1 ;Restore original

LD R2, Save2 ;register values LD R3, Save3

AND R5, R5, #0

ADD R5, R5, #1 ;R5 <--- failure RET

BASE .FILL x3FFF

NBASE .FILL xC001 ; NBASE contains -x3FFF.

MAX .FILL xC005

Save1 .FILL x0000

Save2 .FILL x0000

Save3 .FILL x0000

8.7 ; Subroutines for carrying out the PUSH and POP functions. This

; program works with a stack consisting of memory locations x3FFF

; (BASE) through x3FFB (MAX). R6 is the stack pointer. R3 contains

; the size of the stack element. R4 is a pointer specifying the

; location of the element to PUSH from or the space to POP to

;

POP ST R2, Save2 ; are needed by POP. ST R1, Save1

ST R0, Save0

LD R1, BASE ; BASE contains -x3FFF. ADD R1, R1, #-1 ; R1 contains -x4000.

ADD R2, R6, R1 ; Compare stack pointer to x4000 BRz fail\_exit ; Branch if stack is empty.

ADD R0, R4, #0

ADD R1, R3, #0

ADD R5, R6, R3

ADD R5, R5, #-1

ADD R6, R6, R3

|  |  |  |
| --- | --- | --- |
| pop\_loop | LDR  STR | R2, R5, #0  R2, R0, #0 |
|  | ADD | R0, R0, #1 |
|  | ADD | R5, R5, #-1 |
|  | ADD  BRp BRnzp | R1, R1, #-1  pop\_loop success\_exit |
| PUSH | ST | R2, Save2 ; Save registers that |
|  | ST | R1, Save1 ; are needed by PUSH. |
|  | ST | R0, Save0 |
|  | LD | R1,MAX ; MAX contains -x3FFB |
|  | ADD  BRz ADD ADD ADD | R2,R6,R1 ; Compare stack pointer to -x3FFB fail\_exit ; Branch if stack is full.  R0, R4, #0  R1, R3, #0  R5, R6, #-1 |
|  | NOT | R2, R3 |
|  | ADD | R2, R2, #1 |
|  | ADD | R6, R6, R2 |
| push\_loop | LDR | R2, R0, #0 |
|  | STR | R2, R5, #0 |
|  | ADD | R0, R0, #1 |
|  | ADD | R5, R5, #-1 |
|  | ADD  BRp | R1, R1, #-1  push\_loop |
| success\_exit | LD | R0, Save0 |
|  | LD | R1, Save1 ; Restore original |
|  | LD | R2, Save2 ; register values. |
| fail\_exit | AND RET  LD | R5, R5, #0 ; R5 <-- success.  R0, Save0 |
|  | LD | R1, Save1 ; Restore original |
|  | LD | R2, Save2 ; register values. |
|  | AND | R5, R5, #0 |
| BASE | ADD RET  .FILL | R5, R5, #1 ; R5 <-- failure.  xC001 ; BASE contains -x3FFF. |
| MAX | .FILL | xC005 |
| Save0 | .FILL | x0000 |

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

|  |  |  |
| --- | --- | --- |
| Save1 | .FILL | x0000 |
| Save2 | .FILL | x0000 |

8.9 (a) BDECJKIHLG

* 1. Push Z

Push Y Pop Y Push X Pop X Push W Push V Pop V Push U Pop U

Pop W

Pop Z Push T Push S Pop S Push R Pop R

Pop T

* 1. 14 different output streams.

8.11 16 memory locations are needed for the assembled program. Address of C = **x400F**

After execution, C contains the **average** of the four consecutive values starting at memory location specified in B.

8.13 FACT ST R1, SAVE\_R1

ADD R1, R0, #0

## BRnp SKIP

**ADD R1, R1, #1**

**BRnzp DONE**

**SKIP** ADD R0, R0, #-1 BRz DONE

AGAIN MUL R1, R1, R0

ADD R0, R0, #-1 BRnp AGAIN

DONE ADD R0, R1, #0

LD R1, SAVER1 RET

SAVE\_R1 .BLKW 1

8.15 NOTE: There is an error in the statement of this problem. See Errata Sheet for question. Additionally, this problem would belong in Chapter 9 rather than Chapter 8.

|  |  |
| --- | --- |
| **MAR** | **MDR** |
| **x400E** | x5020 |
| **x400F** | xF0F0 |
| **x1FFF** | **x8002** |
| **x1FFE** | x4010 |
| **x00F0** | **x2000** |
| x2000 | x71BF |
| x1FFD | **x0000** |
| **x2001** | x8000 |
| **x1FFE** | **x4010** |
| **x1FFF** | **x8002** |
| **x4010** | xF025 |

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

# 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.

1. 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.
2. 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.

* 1. 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.

* 1. STI R0, MCR This instruction clears the most significant bit of the machine control register, stopping the clock.
  2. LD R1, SaveR1
  3. 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

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

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 us- ing 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 ser- vice 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

1. TRAP x25
2. ADD R0, R0, #5
3. BRzp K

9.33 (a) The keyboard interrupt is enabled, and the digit 2 is repeatedly written to the screen.

* 1. The character typed is echoed twice to the screen.
  2. The digit 2 some number of times, followed by the digit typed twice or three times, followed by the digit 2 continually thereafter.
  3. 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 LDR0, 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

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

x6311

PSR for device B x6203

PSR of Program A x3007

xxxxx

* 1. 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 ADD R1, 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.

* 1. The three errors that arose in the ﬁrst student’s program are:
     1. The stack is left unbalanced.

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

* + 1. The privilege mode and condition codes are not restored.
    2. 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

1. ADD R2, R2, R2
2. ADD R2, R2, R2
3. 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 SaveR1.BLKW 1

(d)

.BLKW 1 (e)

***SOLUTION***:

1. ST R0, SaveR0
2. ADD R0, R0, R1
3. LD R0, SaveR0
4. .FILL x-20
5. SaveR0

9.49 (a) LEA R0, INPUT

1. ADD R6, R6, R0
2. LDR R0, R6, #2
3. .FILL S0
4. .FILL S1
5. .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 |

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

# F.10 Chapter 10 Solutions

10.1 The Multiply step works by adding the multiplicand a number of times to an accumulator. The number of times to add is determined by the multiplier. The number of instructions executed to perform the Multiply step = 3 + 3\*n, where n is the value of the multiplier. We will in general do better if we replace the core of the Multiply routine (lines 17 through 19 of Figure 10.14) with the following, doing the Multiply as a series of shifts and adds:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | AND | R0, | R0, | #0 |  |
|  | ADD | R4, | R0, | #1 | ;R4 contains the bit mask (x0001) |
| Again | AND  BRz ADD | R5, R2,  BitZero R0, R0, | | R4  R1 | ;Is corresponding  ;bit of multiplier=1  ;Multiplier bit=1  ;--> add  ;shifted multiplicand  ;Product has already  ;exceeded range  ;Shift the |
|  | BRn | Restore2 | |  |
| BitZero | ADD | R1, R1, | | R1 |
|  | BRn  ADD | Check  R4, R4, | | R4 | ;multiplicand bits  ;Mcand too big  ;--> check if any  ;higher mpy bits = 1  ;Set multiplier bit to  ;next bit position |
|  | BRn BRnzp | DoRangeCheck Again | | | ;We have shifted mpy  ;bit into bit 15  ;-->done. |
| Check  DoRangeCheck | AND  BRp ADD  BRp | R5, R2, R4  Restore2 R4, R4, R4  Check | | |  |

10.3 This program assumes that hex digits are all capitalized.

LD R3, NEGASCII

LD R5, NEGHEX

TRAP x23

ADD R1, R0, R3 ;Remove ASCII template LD R4, HEXTEST ;Check if digit is hex ADD R0, R1, R4

BRnz NEXT1

ADD R1, R1, R5 ;Remove extra

;offset for hex

|  |  |  |
| --- | --- | --- |
| NEXT1 TRA | P x23 |  |
| ADD | R0, R0 | , R3 ;Remove ASCII template |
| ADD  BRn ADD | R2, R0  z NEXT2  R0, R0 | , R4 ;Check if digit is hex  , R5 ;Remove extra |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | ;offset for hex |
| NEXT2 | ADD | R0, R1, R0 | ;Add the numbers |
|  | ADD  BRnz LD ADD | R1, R0, R4 NEXT3  R2, HEX R0, R0, R2 | ;Check if digit > 9  ;Add offset for hex digits |
| NEXT3 | LD | R2, ASCII |  |
|  | ADD | R0, R0, R2 | ;Add the ASCII template |
| DONE | TRAP | x21 |  |
|  | TRAP | x25 |  |

|  |  |  |
| --- | --- | --- |
| ASCII | .FILL | x0030 |
| NEGASCII | .FILL | x-0030 |
| HEXTEST | .FILL | #-9 |
| HEX | .FILL | x0007 |
| NEGHEX | .FILL | x-7 |

10.5 ;

; R1 contains the number of digits including ’x’. Hex

; digits must be in CAPS.

ASCIItoBinary AND R0, R0, #0 ; R0 will be used for our result ADD R1, R1, #0 ; Test number of digits.

BRz DoneAtoB ; There are no digits

;

LD R3, NegASCIIOffset ; R3 gets xFFD0, i.e., -x0030 LEA R2, ASCIIBUFF

LD R6, NegXCheck LDR R4, R2, #0 ADD R6, R4, R6

BRz DoHexToBin

ADD R2, R2,R1

ADD R2, R2, #-1 ; R2 now points to "ones" digit

;

LDR R4, R2, #0 ; R4 <-- "ones" digit

ADD R4, R4, R3 ; Strip off the ASCII template

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

|  |  |
| --- | --- |
| ADD | R0, R0, R4 ; Add ones contribution |
| ; |  |
| ADD | R1, R1, #-1 |
| BRz | DoneAtoB ; The original number had one digit |
| ADD | R2, R2, #-1 ; R2 now points to "tens" digit |
| ; |  |
| LDR | R4, R2, #0 ; R4 <-- "tens" digit |
| ADD | R4, R4, R3 ; Strip off ASCII template |
| LEA | R5, LookUp10 ; LookUp10 is BASE of tens values |
| ADD | R5, R5, R4 ; R5 points to the right tens value |
| LDR | R4, R5, #0 |
| ADD | R0, R0, R4 ; Add tens contribution to total |
| ; |  |
| ADD | R1, R1, #-1 |
| BRz | DoneAtoB ; The original number had two digits |
| ADD | R2, R2, #-1 ; R2 now points to "hundreds" digit |
| ; |  |
| LDR | R4, R2, #0 ; R4 <-- "hundreds" digit |
| ADD | R4, R4, R3 ; Strip off ASCII template |
| LEA | R5, LookUp100 ; LookUp100 is hundreds BASE |
| ADD | R5, R5, R4 ; R5 points to hundreds value |
| LDR | R4, R5, #0 |
| ADD | R0, R0, R4 ; Add hundreds contribution to total |
| RET |  |

DoHexToBin ; R3 = NegASCIIOffset

; R2 = Buffer Pointer

; R1 = Num of digits + x

;

ST R7, SaveR7

LD R6, NumCheck ADD R1, R1, #-1

ADD R2, R2,R1

;

LDR R4, R2, #0 ; R4 <-- "ones" digit

ADD R4, R4, R3 ; Strip off the ASCII template ADD R7, R4, R6

BRnz Cont1

LD R7, NHexDiff

|  |  |  |
| --- | --- | --- |
|  | ADD | R4, R4, R7 |
| Cont1 | ADD | R0, R0, R4 ; Add ones contribution |
| ; | ADD | R1, R1, #-1 |

BRz DoneAtoB ; The original number had one digit ADD R2, R2, #-1 ; R2 now points to "tens" digit

;

LDR R4, R2, #0 ; R4 <-- "tens" digit

ADD R4, R4, R3 ; Strip off ASCII template ADD R7, R4, R6

BRnz Cont2

LD R7, NHexDiff ADD R4, R4, R7

|  |  |  |  |
| --- | --- | --- | --- |
| Cont2 | LEA  ADD | R5,  R5, | LookUp16  R5, R4 |
|  | LDR | R4, | R5, #0 |
|  | ADD | R0, | R0, R4 |
| ; |  |  |  |
|  | ADD | R1, | R1, #-1 |

BRz DoneAtoB ; The original number had two digits ADD R2, R2, #-1 ; R2 now points to "hundreds" digit

;

LDR R4, R2, #0

ADD R4, R4, R3 ; Strip off ASCII template ADD R7, R4, R6

BRnz Cont3

LD R7, NHexDiff ADD R4, R4, R7

Cont3 LEA R5, LookUp256 ADD R5, R5, R4 LDR R4, R5, #0 ADD R0, R0, R4

;

DoneAtoB LD R7, SaveR7 RET

NegASCIIOffset .FILL xFFD0 NumCheck .FILL #-9

NHexDiff .FILL #-7

NegXCheck .FILL xFF88

SaveR7 .FILL x0000

ASCIIBUFF .BLKW 4

LookUp10 .FILL #0

.FILL #10

.FILL #20

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*

|  |  |  |
| --- | --- | --- |
| ; | .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL | #30  #40  #50  #60  #70  #80  #90 |
| LookUp100  LookUp16 | .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL | #0  #100  #200  #300  #400  #500  #600  #700  #800  #900  #0  #16  #32  #48  #64  #80  #96  #112  #128  #144  #160  #176  #192  #208  #224  #240 |
| LookUp256 | .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL  .FILL | #0  #256  #512  #768  #1024  #1280  #1536  #1792  #2048  #2304 |

|  |  |
| --- | --- |
| .FILL | #2560 |
| .FILL | #2816 |
| .FILL | #3072 |
| .FILL | #3328 |
| .FILL | #3584 |
| .FILL | #3840 |

10.7 This program reverses the input string. For example, given an input of “Howdy”, the output is “ydwoH”.

10.9 NOTE: This question is redundant. The PUSH\_VALUE routine is already robust in that is does test to be sure that each character typed is a decimal digit. No further work needs to be done.

*Copyright 2020 © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.*