CHAPTER 9

9-1.

```
a) LD    R1, A
   LD    R2, B
   LD    R3, C
   LD    R4, D
   LD    R5, E
   ADD   R1, R1, R2
   SUB   R1, R1, R3
   SUB   R4, R4, R5
   MUL   R1, R1, R4
   ST     X, R1
```

```
b) MOV   T1, A
   ADD   T1, B
   SUB   T1, C
   MOV   T2, D
   SUB   T2, E
   MUL   T1, T2
   MOV    X, T1
```

```
c) LD    A
   ADD   B
   SUB   C
   ST    T1
   LD    D
   SUB   E
   MUL   T1
   ST     X
```

9-2.*

```
a) LD    R1, E
   LD    R2, F
   MUL   R1, R1, R2
   LD    R2, D
   SUB   R1, R2, R1
   LD    R2, C
   DIV   R1, R2, R1
   LD    R2, A
   LD    R3, B
   ADD   R2, R2, R3
   MUL   R1, R1, R2
   ST     Y, R1
```

```
b) MOV   T1, A
   ADD   T1, B
   MUL   T1, C
   MOV   T2, E
   MUL   T2, F
   MOV   T3, D
   SUB   T3, T2
   DIV   T1, T3
   MOV   Y, T1
```

```
c) LD    E
   MUL   F
   ST    T1
   LD    D
   SUB   T1
   ST    T1
   LD    A
   ADD   B
   MUL   C
   DIV   T1
   ST    Y
```

9-3.*

a) $(A-B)\times(A+C)\times(B-D)=AB-AC+xBD-x$

b, c)

PUSH A	PUSH B	SUB	PUSH A	PUSH C	ADD
A	B	A−B	A	C	A+C
	A		A−B	A	A−B
				A−B	

MUL	PUSH B	PUSH D	SUB	MUL	POP X
(A−B) × (A+C)	B	D	B−D	(A−B) × (A+C) × (B−D)	
	(A−B) × (A+C)	B	(A−B) × (A+C)		
		(A−B) × (A+C)			

9-4.

a) $(((A\times B)+C)\times D\div(E-(A\times F)) = AB\times C+D\times EAF\times-\div Y$

b, c)

PUSH A	PUSH B	MUL	PUSH C	ADD	PUSH D
A	B	A × B	C	(A × B) +C	D
	A		A × B		(A × B)+C

MUL	PUSH E	PUSH A	PUSH F	MUL	SUB
((A × B)+C) × D	E	A	F	A × F	E − (A × F)
	((A × B)+C) × D	E	A	E	
		((A × B)+C) × D	E	((A × B)+C) × D	
			((A × B)+C) × D		

DIV		POP Y
(((A × B)+C) × D) ÷ (E − (A × F))		

9-5.

a) $Z = Y$ b) $Z = M[Y]$ c) $Z = Y + W + 2$ d) $Z = Y + X$

9-6.*

a) X = 195 – 208 – 1 = –14 b) X = 1111 1111 1111 0010

The number is negative because the branch is backwards. The – 1 assumes that the PC has been incremented to point to the address after that of the address word of the instruction.

9-7.

a) X = 1000 – 144 – 1 = 855 b) X = 0000 0011 0101 0111

The number is negative because the branch is backwards. The − 1 assumes that the PC has been incremented to point to the address after that of the address word of the instruction.

9-8.

a) Read Instruction
Read Address Field
Read Effective Address
Read Operand
Read Address Field
Read Effective Address
Read Operand
Write Result
8 Memory Accesses

b) Read Instruction
Read Address Field
Read Effective Address
Read Operand
Read Address Field
Read Effective Address
Write Result
7 Memory Accesses

9-9.

a) Direct 2410 b) Immediate 551 c) Relative 552 + 2410 = 2962
d) Indexed 2410 + 2310 = 4720

9-10.*

a) 3 Register Fields × 4 bits/Field = 12 bits. 32 bits − 12 bits = 20 bits. $2^{20} = 1048576$

b) 64 < 100 < 128 => 7 bits. 2 Register Fields × 4 bits/Field => 8 bits. 32 bits – 7 bits – 8 bits => 17 Address Bits

9-11.

```
LDI    R6, TOSAD    # Load R6 with address of the Top of Stack
```

Push and Pop can be implemented in one of two ways.

	Method 1			Method 2	
PUSH RX:	ST	RX, (R6)+	or	ST	RX, –(R6)
POP RX:	LD	RX,–(R6)		LD	RX, (R6)+

9-12.

PSHR and POPR work as follows:

PSHR:		POPR:	
	$M[SP] \leftarrow R0$		$SP \leftarrow SP + 1$
	$SP \leftarrow SP - 1$		$R7 \leftarrow M[SP]$
	$M[SP] \leftarrow R1$		$SP \leftarrow SP + 1$
	$SP \leftarrow SP - 1$		$R6 \leftarrow M[SP]$
	.		.
	.		.
	.		.
	$M[SP] \leftarrow R7$		$SP \leftarrow SP + 1$
	$SP \leftarrow SP - 1$		$R0 \leftarrow M[SP]$

9-13.

LD R[DR] ADRS
ST ADRS R[SB]
where ADRS is a memory address.

9-14.*

a)		b)		
ADD	R0, R4	$R0 \leftarrow 7B + 4B$,	R0 = C6,	C = 0
ADC	R1, R5	$R1 \leftarrow 24 + ED + 0$,	R1 = 11,	C = 1
ADC	R2, R6	$R2 \leftarrow C6 + 57 + 1$,	R2 = 1E,	C = 1
ADC	R3, R7	$R3 \leftarrow 1F + 00 + 1$	R3 = 20,	C = 0

9-15.

a)	AND:	0100 1000	OR:	1110 1111	XOR:	1010 0111
b)	AND:	0001 0001	OR:	0111 1001	XOR:	0110 1000

9-16.

a) OR with $(FF00_{16}$ b) XOR with $(AAAA)_{16}$ c) AND with $(A\ A\ A\ A)_{16}$

9-17.*

OP	Result Register	C
	0110 1001	1
SHR	0011 0100	1
SHL	0110 1000	0
SHRA	0011 0100	0
SHLA	0110 1000	0
ROR	0011 0100	0
ROL	0110 1000	0
RORC	0011 0100	0
ROLC	0110 1000	0

9-18.

$$\begin{array}{l} (-)\ 0.123450000 \times 10^5 \\ \underline{+(+)\ 0.0000000071234 \times 10^5} \\ =\ \ (-)\ 0.123449993 \times 10^5 \text{ Result} = -0.123449993 \times 10^5 \end{array}$$

9-19.*

Smallest Number	=	0.5	×	2^{-255}
Largest Number	=	$(1 - 2^{-26})$	×	2^{+255}

9-20.*

E	e	$(e)_2$
+8	15	1111
+7	14	1110
+6	13	1101
+5	12	1100
+4	11	1011
+3	10	1010
+2	9	1001
+1	8	1000
0	7	0111
–1	6	0110
–2	5	0101
–3	4	0100
–4	3	0011
–5	2	0010
–6	1	0001
–7	0	0000

9-21.

a) The largest quantity is 0 1111 11111 = $2^7 \times (0.111111)_2 = (1111110)_2 = 126$. The smallest quantity is 0 0000 00001 = $2^{-8} \times (0.100001)_2 = (0.00000000100001)_2$ = 0.00201416015625

b) 5.675 = $(101.101)_2$ to six binary digits, which is equal to $(0.101101)_2 \times 2^3$. The 4-bit excess 8 exponent would be 1011, and the normalized fraction would be stored as 01101. So using this floating point format, -5.675 = 1 1011 01101.

9-22.

a) $(-1)^{\text{sign}} \times 2^{\text{exponent}-1023} \times (1.\text{fraction})$

b)

–1022	=	$(000\ 0000\ 0001)_2$
–1	=	$(011\ 1111\ 1110)_2$
0	=	$(011\ 1111\ 1111)_2$
+1	=	$(100\ 0000\ 0000)_2$
+1023	=	$(111\ 1111\ 1110)_2$

c)

Largest	=	$(2 - 2^{-52}) \times 2^{+1023}$
Smallest	=	1×2^{-1022}

9-23.

If $2^x = 10^y$ then:

$x \log_{10} 2 = y \log_{10} 10$ $\qquad \log_{10} 2 = 0.3, \log_{10} 10 = 1$

0.3 x = y

Largest is $2^{127} \times (2 - 2^{-23}) = 2^{128} - 2^{104} = 10^{0.3\times128} = 10^{0.3\times104} = 2.5\times10^{38}$

Smallest is $2^{-126} = 10^{-0.3\times126} = 10^{-37.8} = 1.58\times10^{-38}$

9-24.

a) -9.359375 is stored as C115C000 in IEEE single-precision floating point format.
b) 41CBA000 is the IEEE single-precision floating point representation of 25.453125.

9-25.*

TEST	$(0001)_{16}$, R	(AND Immediate 1 with Register R)
BNZ	ADRS	(Branch to ADRS if Z = 0)

9-26.

			Unsigned	Signed
a)	A =	1011 0110	182	− 74
	B =	<u>0011 0111</u>	<u>55</u>	<u>55</u>
b)	A + B	= 1110 1101	237	− 19
c)	C=0, Z=0, N=0, V=0		C=0, Z=0, N=1, V=0	
d)	BNZ, BNC, BNN, BNV		BNZ, BNC, BN, BNV	

9-27.*

a)	A =	0101 1101	93
	B =	<u>0101 1100</u>	<u>−92</u>
	A – B =	0000 0001	1

b) C (borrow) = 0, Z = 0
c) BA, BAE, BNE

9-28.

a)	A =	1101 1010	–38
	B =	<u>0111 0110</u>	<u>–(118)</u>
	A – B =	00010000	100

b) N = 0, Z = 0, V = 1
c) BL, BLE, BNE

9-29.

a)	A =	1010 0100	–92
	B =	<u>1010 1001</u>	<u>–(-87)</u>
	A – B =	1111 1011	-5

b) N = 1, Z = 0, V = 0
c) BL, BLE, BNE

9-30.*

	PC	SP	TOS
a) Initially	2000	4000	5000
b) After Call	0502	4001	2002
c) After Return	2002	4000	5000

9-31.

a) $R7 \leftarrow PC$	PC points to the next instruction in sequence
$PC \leftarrow M[PC-1]$	PC Gets Effective address

b) R7 must be saved to memory.

9-32.

Branch Instruction:	The processor jumps to a new location without the ability to return on its own.
Call Instruction:	A jump with a way for the processor to return to the instruction after the one that caused the jump.
Program Interrupt:	A jump initiated by an event, not an instruction. A way to return to the first uncompleted instruction is provided.

9-33.*

External Interrupts:	Internal Interrupts:
1) Hard Drive	1) Overflow
2) Mouse	2) Divide by zero
3) Keyboard	3) Invalid opcode
4) Modem	4) Memory stack overflow
5) Printer	5) Protection violation

A software interrupt provides a way to call the interrupt routines normally associated with external or internal interrupts by inserting an instruction into the code. Privileged system calls for example must be executed through interrupts in order to switch from user to system mode. Procedure calls do not allow this change.

9-34.

a) $SP \leftarrow SP - 1$
$M[SP] \leftarrow PC$
$SP \leftarrow SP - 1$
$M[SP] \leftarrow PSR$
$EI \leftarrow 0$
$PC \leftarrow IVAD$
$PSR \leftarrow M[PC]$
$PC \leftarrow PC + 1$
Begin Fetch of Interrupt Service Routine

b) $PSR \leftarrow M[SP]$
$SP \leftarrow SP + 1$
$PC \leftarrow M[SP]$
$SP \leftarrow SP + 1$
Begin fetch of next instruction following the one for which the interrupt occurred.

9-35.

a)
```
LD R0, PARAMETER1
LD R1, PARAMETER2
CALL PART_A_PROCEDURE
```

b)
```
PART_B_PROCEDURE:
    PUSH R3
    PUSH R4
    /* … other instructions in the procedure…if a value is returned by the procedure, the value will be
    stored in R2 just before the next two instructions… */
    POP R4
    POP R3
    RET
```