Assessment3

Due Jul 23 at 11:57am **Points** 100 **Questions** 28
Available Jul 23 at 10:59am - Jul 23 at 11:59am about 1 hour **Time Limit** 56 Minutes

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	56 minutes	76.25 out of 100 *

* Some questions not yet graded

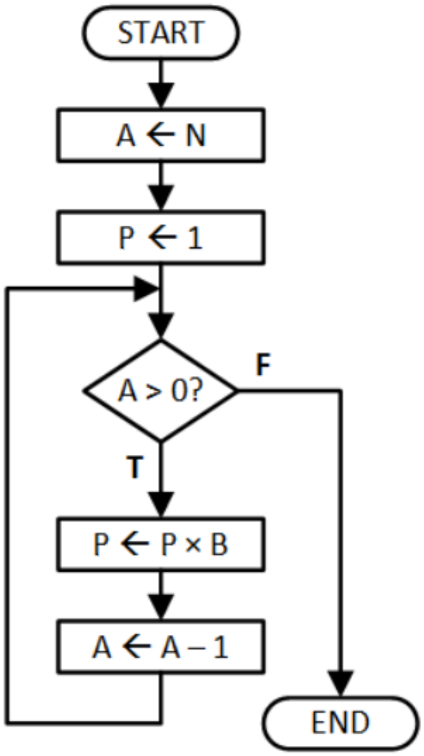
Correct answers are hidden.

Score for this quiz: **76.25** out of 100 *
Submitted Jul 23 at 11:55am
This attempt took 56 minutes.

This assessment is to be completed on an **individual** basis, no collaboration. This assessment is timed. You have 55min to complete it. It is recommended you take the questions in order. If you jump around you are likely to miss a question. It is worth 13.5% of your grade. Just relax and answer the questions deliberately and thoughtfully.

Question 1

3 / 3 pts



Simulate in your head for different small values of **N** & **B**

If **N**=0 the result in **P** is 1

If **N**=2 the **B**=3 the result in **P** is 9

if **N**=5 and **B**=2 the result in **P** is 32 *(you can simulate or realize what it is calculating)*

Answer 1:

1

Answer 2:

9

Answer 3:

32

Question 2

3 / 3 pts

When a **BRzp** instruction is executed, the branch will be taken if either the z or p flags are set

The **BRzp** branch will be taken if the last value written to a register was ≥0

If one leaves the flag specifiers off a **BRx** instruction (i.e. just a **BR** instruction) the branch is always taken

Answer 1:

if either the z or p flags are set

Answer 2:

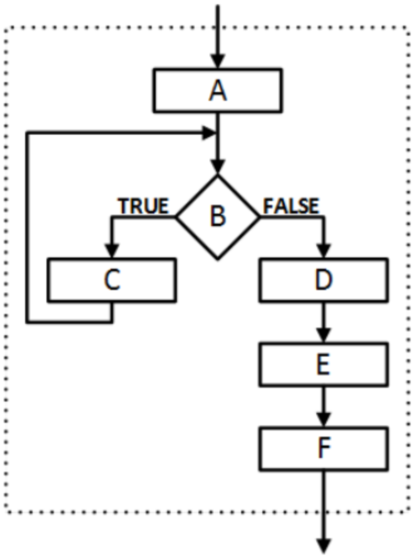
≥0

Answer 3:

branch is always taken

Question 3

3 / 3 pts



Simulate **different** scenarios in your head.

Then give the best answer to the following:

Shape A is a task that is executed once

Shape B is a condition check that is executed one or more times

Shape C is a task that is executed zero or more times

Shape E is a task that is executed once

Answer 1:

a task that is executed once

Answer 2:

a condition check that is executed one or more times

Answer 3:

a task that is executed zero or more times

Answer 4:

a task that is executed once

Question 4

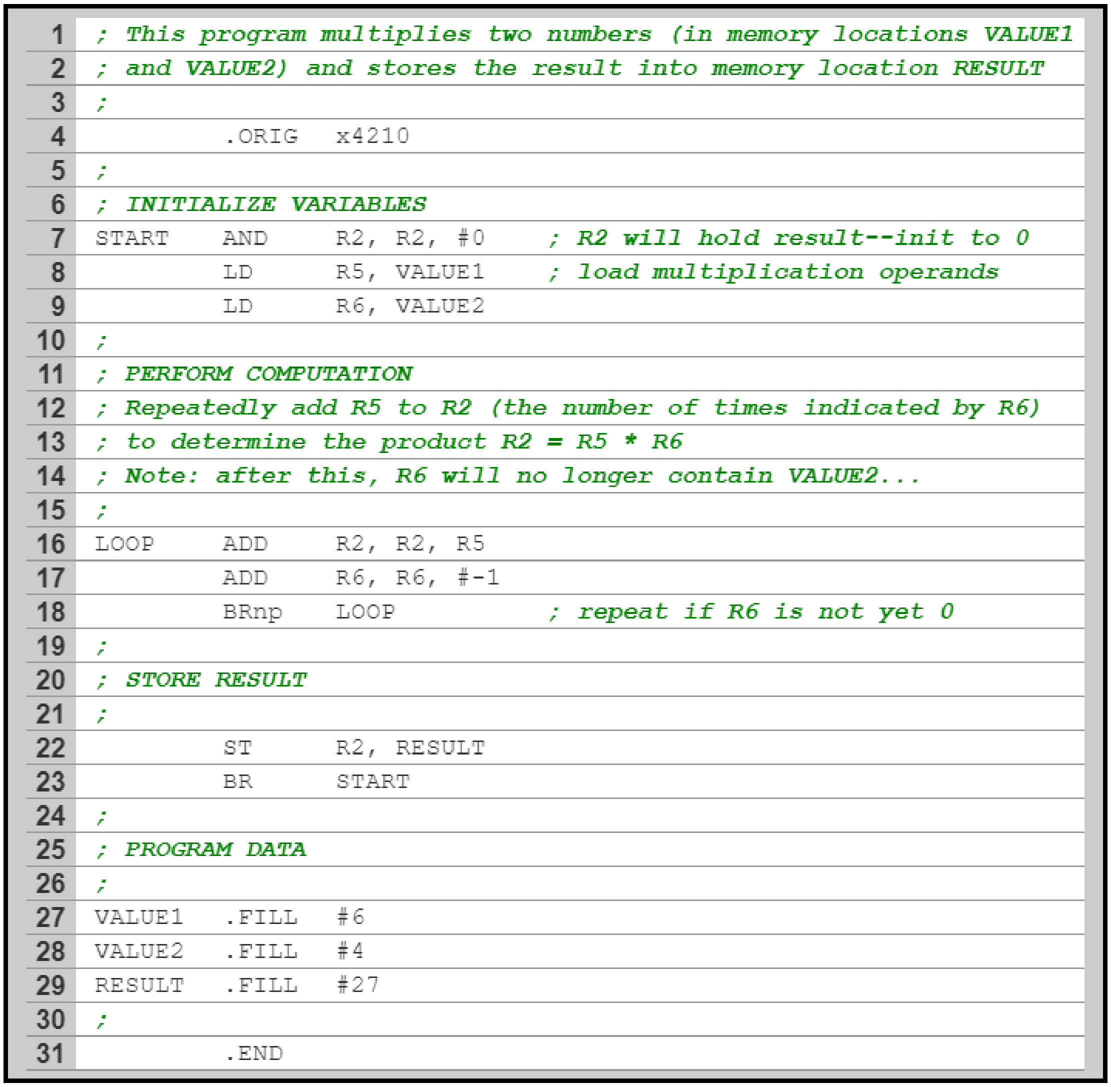
3 / 3 pts

Match the description to the instruction

Loads the PC with a value from a register	JMP
Based on tested condition code(s) it load the PC with value from a register	No instruction like that
Equivalent to: JMP R7	RET
Unconditionally branches to PC+ + sext(immediate) while saving current PC+ value	JSR label

Question 5

3 / 3 pts



```
1  ; This program multiplies two numbers (in memory locations VALUE1
2  ; and VALUE2) and stores the result into memory location RESULT
3  ;
4          .ORIG   x4210
5  ;
6  ; INITIALIZE VARIABLES
7  START   AND     R2, R2, #0     ; R2 will hold result--init to 0
8          LD      R5, VALUE1     ; load multiplication operands
9          LD      R6, VALUE2
10 ;
11 ; PERFORM COMPUTATION
12 ; Repeatedly add R5 to R2 (the number of times indicated by R6)
13 ; to determine the product R2 = R5 * R6
14 ; Note: after this, R6 will no longer contain VALUE2...
15 ;
16 LOOP    ADD     R2, R2, R5
17         ADD     R6, R6, #-1
18         BRnp    LOOP           ; repeat if R6 is not yet 0
19 ;
20 ; STORE RESULT
21 ;
22         ST      R2, RESULT
23         BR      START
24 ;
25 ; PROGRAM DATA
26 ;
27 VALUE1  .FILL   #6
28 VALUE2  .FILL   #4
29 RESULT  .FILL   #27
30 ;
31         .END
```

At what **4-digit hex** address is the label **LOOP** defined 4213

What offset (answer as a decimal number) would be encoded into the **BRnp** instruction -3

True/False: *(simply type true or false in the box)* the program would operate the same if line **29** was:

RESULT .BLKW 1 true

Answer 1:

4213

Answer 2:

-3

Answer 3:

True

Question 6

3 / 3 pts

Choose the **best** description of the **LEA** instruction as used in LC-3 assembly

○ It adds the already incremented PC to an offset and puts that in the destination register

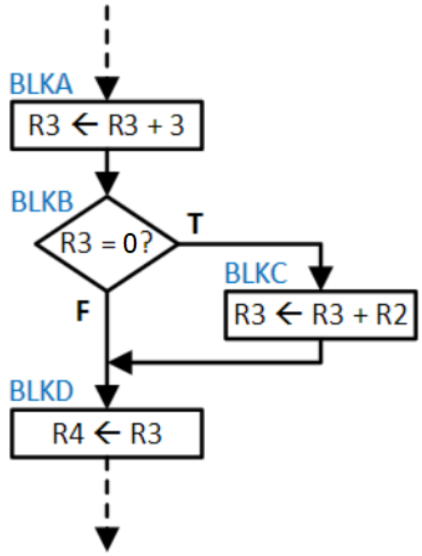
○ It gives one the value stored in a memory location

● It gives one (stores it in a label) the address of a label

○ It gives one the contents of memory at a label

Question 7

3 / 3 pts



Implement the flow chart

```
BLKA        ADD R3, R3, #3
               BRnp BLKD
BLKC       ADD R3, R3, R2
BLKD       AND R4, R3, R3
```

Answer 1:

BRnp

Answer 2:

BLKD

Answer 3:

ADD R3, R3, R2

Answer 4:

AND R4, R3, R3

Question 8

3 / 3 pts

; Routine takes max of two arguments (R0,R1) returns in R0

```
; R0 = Argument 1

; R1 = Argument 2

;  Computes the max of R0 and R1 and places it in R0

MAX  NOT R2, R1                    ; negate R1 into R2

        ADD R2, R2, #1

        ADD R2, R2, R1                        ; compare R2 to R1

        BRzp DONE                        ; decide based on comparison

        ADD R0, R1, #0

DONE  RET
```

As a sub-routint this routine is: faulty because it changes R1 and R2

Answer 1:

NOT R2, R1

Answer 2:

ADD R2, R2, #1

Answer 3:

ADD R2, R2, R1

Answer 4:

BRzp DONE

Answer 5:

ADD R0, R1, #0

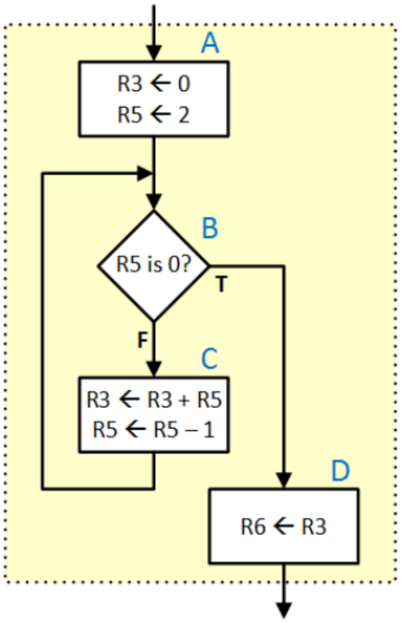
Answer 6:

faulty because it changes R1 and R2

Partial

Question 9

2.25 / 3 pts



Complete the drop downs to implement block **A** and **B** of this flowchart in most logical way

AND R3, R3, #0

ADD R5, R5, #2

BRz BLOCK_D

Answer 1:

AND R3,

Answer 2:

R3,

Answer 3:

#0

Answer 4:

ADD R5,

Answer 5:

R5,

Answer 6:

#2

Answer 7:

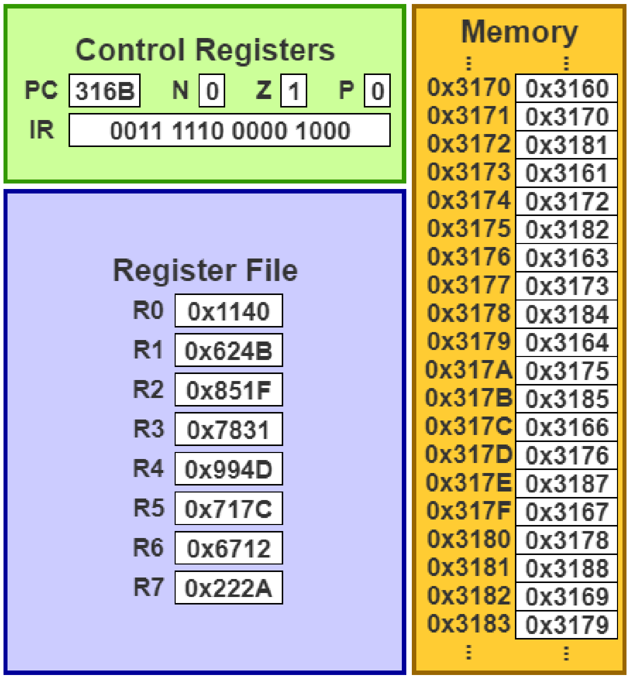
BRz

Answer 8:

BLOCK_D

Question 10

2.5 / 2.5 pts



Control Registers

PC 316B N 0 Z 1 P 0

IR 0011 1110 0000 1000

Register File

R0	0x1140
R1	0x624B
R2	0x851F
R3	0x7831
R4	0x994D
R5	0x717C
R6	0x6712
R7	0x222A

Memory

⋮	⋮
0x3170	0x3160
0x3171	0x3170
0x3172	0x3181
0x3173	0x3161
0x3174	0x3172
0x3175	0x3182
0x3176	0x3163
0x3177	0x3173
0x3178	0x3184
0x3179	0x3164
0x317A	0x3175
0x317B	0x3185
0x317C	0x3166
0x317D	0x3176
0x317E	0x3187
0x317F	0x3167
0x3180	0x3178
0x3181	0x3188
0x3182	0x3169
0x3183	0x3179
⋮	⋮

The **destination** of this instruction is (type a **specific** register (i.e **R0**) or type **memory** if it is memory) memory

The **address** in memory being accessed is (enter **4-digit hex** number)

3173

Answer 1:

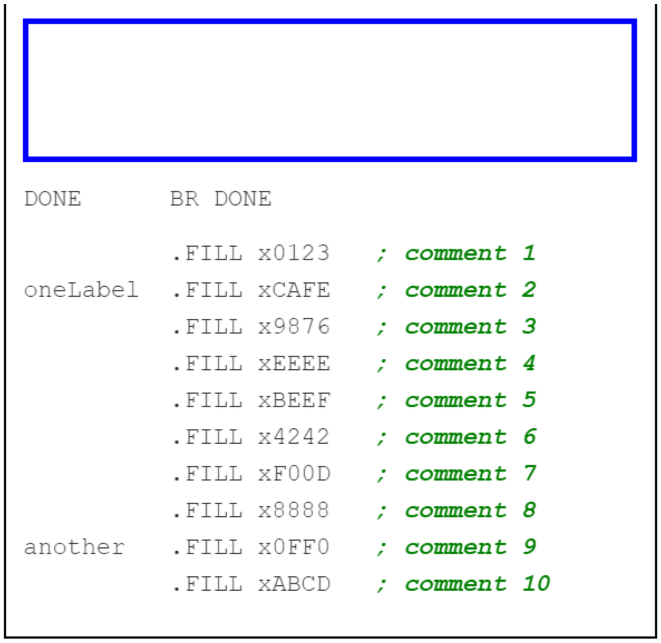
memory

Answer 2:

3173

Question 11

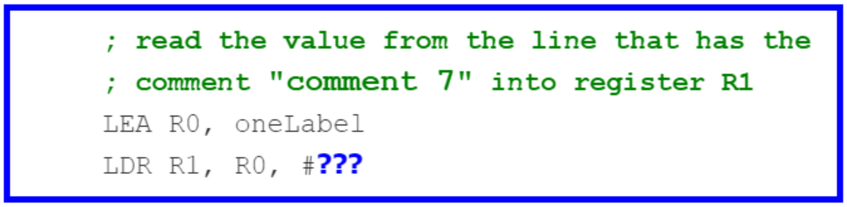
2.5 / 2.5 pts



```
DONE        BR DONE

            .FILL x0123    ; comment 1
oneLabel    .FILL xCAFE    ; comment 2
            .FILL x9876    ; comment 3
            .FILL xEEEE    ; comment 4
            .FILL xBEEF    ; comment 5
            .FILL x4242    ; comment 6
            .FILL xF00D    ; comment 7
            .FILL x8888    ; comment 8
another     .FILL x0FF0    ; comment 9
            .FILL xABCD    ; comment 10
```

Code exists in blue box, but is not shown

If the **DONE** label exists at **address** 0x0282 then what is the **address** of **oneLabel** (enter 4-digit hex) 0284



```
; read the value from the line that has the
; comment "comment 7" into register R1
LEA R0, oneLabel
LDR R1, R0, #???
```

The code in the blue box is now shown above. What value must ??? be to make the comments correct (answer as a decimal number) 5

Answer 1:

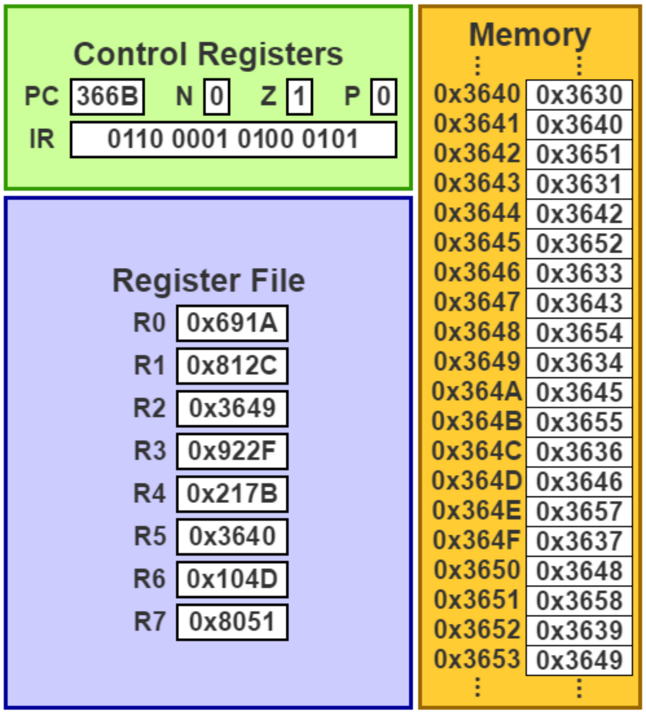
0284

Answer 2:

5

Question 12

2.5 / 2.5 pts



Control Registers

PC	366B	N	0	Z	1	P	0
IR	0110 0001 0100 0101						

Register File

R0	0x691A
R1	0x812C
R2	0x3649
R3	0x922F
R4	0x217B
R5	0x3640
R6	0x104D
R7	0x8051

Memory

⋮	⋮
0x3640	0x3630
0x3641	0x3640
0x3642	0x3651
0x3643	0x3631
0x3644	0x3642
0x3645	0x3652
0x3646	0x3633
0x3647	0x3643
0x3648	0x3654
0x3649	0x3634
0x364A	0x3645
0x364B	0x3655
0x364C	0x3636
0x364D	0x3646
0x364E	0x3657
0x364F	0x3637
0x3650	0x3648
0x3651	0x3658
0x3652	0x3639
0x3653	0x3649
⋮	⋮

Given this state of an LC-3

The **destination** of this instruction is (type a **specific** register (i.e **R0**) or type **memory** if it is memory) R0

The **address** in memory being accessed is (enter **4-digit hex** number)

3645

Answer 1:

R0

Answer 2:

3645

Question 13

2.5 / 2.5 pts

The LC-3 is executing a LDR instruction. The base register contains **0xFFFF** and the 6-bit offset is **0x08**. What is the **address** of the memory location to be read? Enter in as **4-digit hex** number

0007

Question 14

3 / 3 pts

Complete code that takes a branch if the 2-LSBs of **R0** (i.e. **R0[1:0]**) are zero.

```
TEST_00     AND R1, R0, #3
              BRz SKIP
             ; code that is skipped if R0[1:0] == 00
SKIP
```

Answer 1:

AND R1,

Answer 2:

R0,

Answer 3:

#3

Answer 4:

BRz

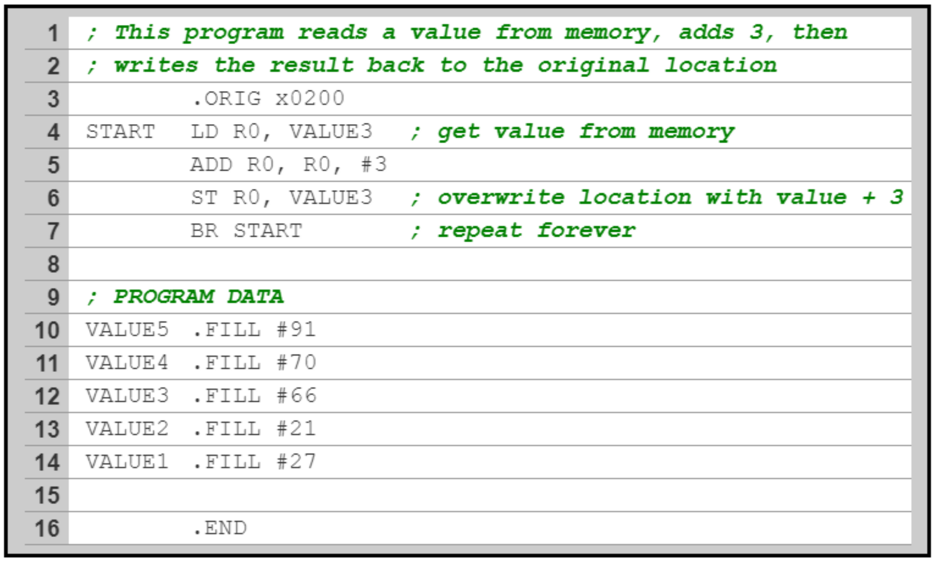
Answer 5:

SKIP

Partial

Question 15

2 / 3 pts



```
 1  ; This program reads a value from memory, adds 3, then
 2  ; writes the result back to the original location
 3          .ORIG x0200
 4  START   LD R0, VALUE3    ; get value from memory
 5          ADD R0, R0, #3
 6          ST R0, VALUE3    ; overwrite location with value + 3
 7          BR START         ; repeat forever
 8
 9  ; PROGRAM DATA
10  VALUE5  .FILL #91
11  VALUE4  .FILL #70
12  VALUE3  .FILL #66
13  VALUE2  .FILL #21
14  VALUE1  .FILL #27
15
16          .END
```

Note code line numbers in grey

After the program is assembled the contents of **line 4** will be located at what memory **address** (enter **4-digit hex**) 0200

After the program is assembled the contents of **line 11** will be located at what memory **address** (enter **4-digit** hex) 0205

Finish off the encoding of line 7 (enter last 9 **binary** digits) 0000111 000000100

Answer 1:

0200

Answer 2:

0205

Answer 3:

000000100

Question 16

3 / 3 pts

Complete the symbol table for the following code:

```
         .ORIG   x3370
;
; INITIALIZE VARIABLES
START    AND     R3, R3, #0       ; R3 will hold result--init to 0
         LD      R5, VALUE1       ; load multiplication operands
         LD      R2, VALUE2
;
; PERFORM COMPUTATION
; Repeatedly add R5 to R3 (the number of times indicated by R2)
; to determine the product R3 = R5 * R2
; Note: after this, R2 will no longer contain VALUE2...
;
LOOP     ADD     R3, R3, R5
         ADD     R2, R2, #-1
         BRnp    LOOP             ; repeat if R2 is not yet 0
;
; STORE RESULT
;
         ST      R3, RESULT
;
; PROGRAM DATA
;
VALUE1   .FILL   #4
VALUE2   .FILL   #7
```

Label:	Address:
START	3370
LOOP	3373
VALUE1	3377
VALUE2	3378

Answer 1:

3370

Answer 2:

LOOP

Answer 3:

3373

Answer 4:

VALUE1

Answer 5:

3377

Answer 6:

VALUE2

Answer 7:

3378

Question 17

3 / 3 pts

MyString .STRINGZ "34"

The above assembler directive reserves 3 words of memory. The words are allocated/initialized as x0033 x0034 x0000 not allocated

Answer 1:

3

Answer 2:

x0033

Answer 3:

x0034

Answer 4:

x0000

Answer 5:

not allocated

Question 18

Not yet graded / 6 pts

NOTE: This problem is manually graded, therefore will show as zero till graded. (max auto graded score is 79)

Download this template for an ABS routine. Look at the comments for what it should perform and complete the code.

When you are satisfied with your code copy and paste it into the "essay" box below.

Your Answer:

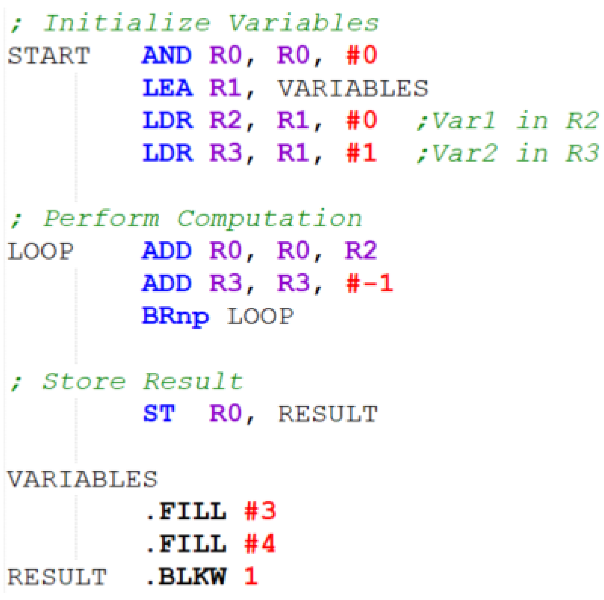
LD R0, SIGNED_DATA
; if negative, negate it
BRzp WRITE
NOT R0, R0
ADD R0, R0, #1

WRITE ; write the (now) positive value to ABS_DATA
ST R0, ABS_DATA

Partial

Question 19

3 / 4 pts



```
; Initialize Variables
START    AND R0, R0, #0
         LEA R1, VARIABLES
         LDR R2, R1, #0   ;Var1 in R2
         LDR R3, R1, #1   ;Var2 in R3

; Perform Computation
LOOP     ADD R0, R0, R2
         ADD R3, R3, #-1
         BRnp LOOP

; Store Result
         ST  R0, RESULT

VARIABLES
         .FILL #3
         .FILL #4
RESULT   .BLKW 1
```

Simulate this code in your head

Register **R1**'s purpose is best described as an address pointer

Register **R0**'s purpose is best described as an accumulator

The final result in register **R3** is #0

This algorithm works when Var2 is positive

Answer 1:

an address pointer

Answer 2:

an accumulator

Answer 3:

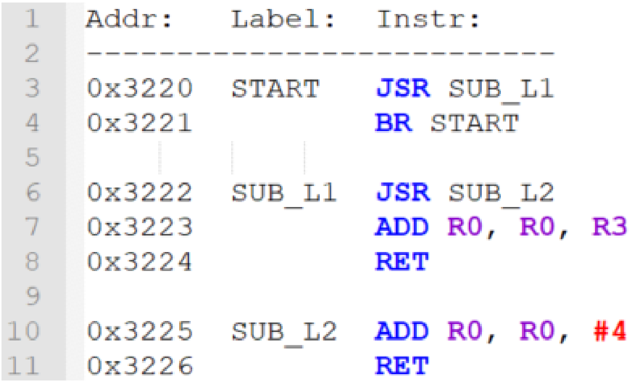
#0

Answer 4:

Var2 is positive

Question 20

3 / 3 pts



```
1   Addr:    Label:   Instr:
2   --------------------------
3   0x3220   START    JSR SUB_L1
4   0x3221            BR START
5
6   0x3222   SUB_L1   JSR SUB_L2
7   0x3223            ADD R0, R0, R3
8   0x3224            RET
9
10  0x3225   SUB_L2   ADD R0, R0, #4
11  0x3226            RET
```

Trace through the execution of this code in your head

What is the **address** of the instruction that will be executed after the **RET** on line 11 (**RET** of **SUB_L2**) (answer as **4-digit hex**) 3223

What is the address of the instruction that will be executed after the **RET** on line 8 (**RET** of **SUB_L1**) (answer as **4-digit hex**) 3223

Answer 1:

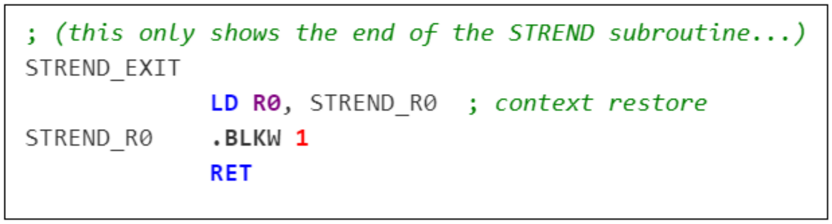
3223

Answer 2:

3223

Question 21

3 / 3 pts



```
; (this only shows the end of the STREND subroutine...)
STREND_EXIT
            LD R0, STREND_R0   ; context restore
STREND_R0   .BLKW 1
            RET
```

Context restore of **R0** at end of the STREND routine

This code has a bug the .BLKW directive will be executed as if it was an instruction this will result in unknown behavior

Answer 1:

has a bug

Answer 2:

.BLKW directive

Answer 3:

executed as if it was an instruction

Answer 4:

unknown behavior

Question 22

3 / 3 pts

To return from a subroutine one places a RET instruction at the end of the subroutine. This instruction is equivalent to:

JMP R7

Answer 1:

RET

Answer 2:

JMP

Answer 3:

R7

Question 23

Not yet graded / 7 pts

NOTE: This problem is manually graded, therefore will show as zero till graded. (max auto graded score is 79)

Download **this template** for a swap routine. Flesh out the code. It is to be coded as a subroutine, not a stand alone algorithm.

When you are satisfied with your code copy and paste it into the "essay" box below.

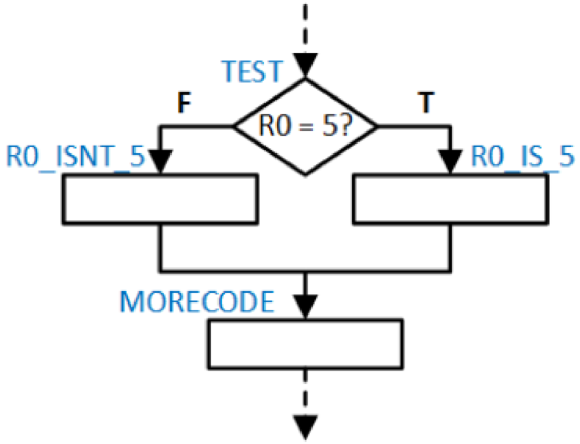
Your Answer:

```
ST R2, SAVE_R2 ; save R2 (context save)
AND R2, R2, #0 ; set to 0 as temp var
ADD R2, R2, R0 ; set R2 to R0

; set R0 to R1
AND R0, R0, #0
ADD R0, R0, R1
; set R1 to R0 (R2 due to R0 already being changed)
AND R1, R1, #0
```

```
ADD R1, R1, R2


LD R2, SAVE_R2 ; restore the value of R2


RET ; return from routine

; context save?

SAVE_R2 .BLKW 1
```

Question 24

4 / 4 pts



Implement in optimal manner

```
TEST            ADD R1, R0, #-5

                  BRz R0_IS_5

R0_ISNT_5 ; code if R0 ≠5

                ...

                  BR MORECODE

R0_IS_5       ;  code if R0 = 5

                 ...

                not needed

MORECODE

                ; executes after joining

                ; for both cases
```

Answer 1:

ADD

Answer 2:

R0, #-5

Answer 3:

BRz

Answer 4:

R0_IS_5

Answer 5:

BR

Answer 6:

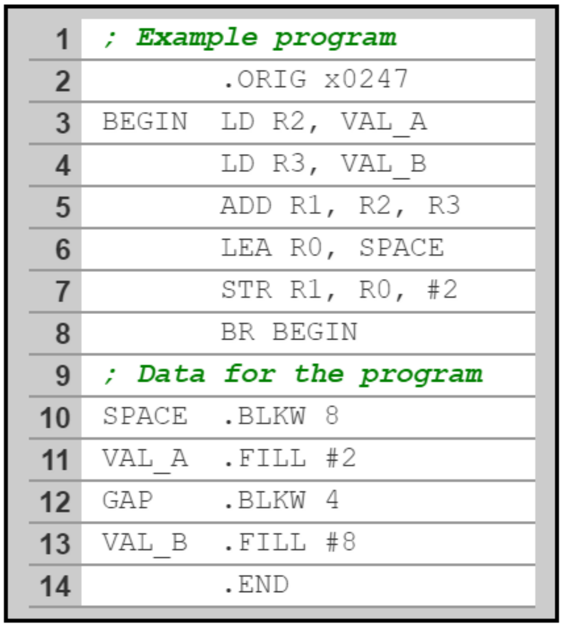
MORECODE

Answer 7:

not needed

Question 25

4 / 4 pts



```
; Example program
        .ORIG x0247
BEGIN   LD R2, VAL_A
        LD R3, VAL_B
        ADD R1, R2, R3
        LEA R0, SPACE
        STR R1, R0, #2
        BR BEGIN
; Data for the program
SPACE   .BLKW 8
VAL_A   .FILL #2
GAP     .BLKW 4
VAL_B   .FILL #8
        .END
```

How many words are allocated **for data** in this program 14

How many words allocated for data are **uninitialized** 12

Within the memory space how many words contain **uninitialized** data after execution 11

How many entries does the assembler create in the symbol table for this code 5

The effective **address** for the **STR** instruction is (i.e. to what address will it store R1) 024F

Answer 1:

14

Answer 2:

12

Answer 3:

11

Answer 4:

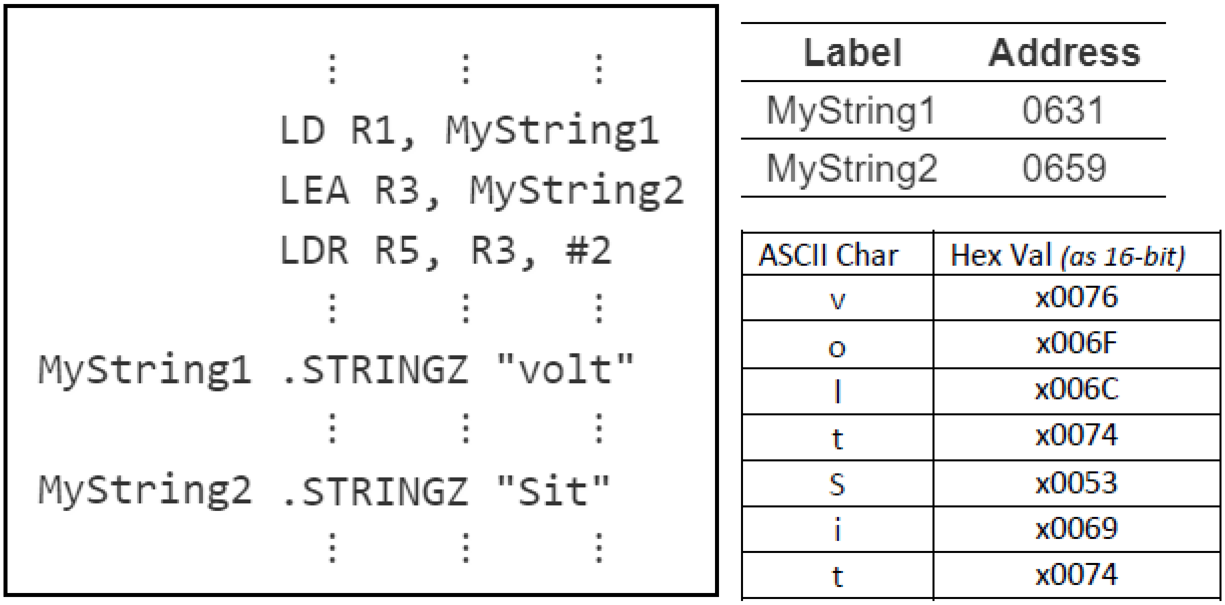
5

Answer 5:

024F

Question 26

4 / 4 pts



```
              ⋮      ⋮      ⋮
            LD R1, MyString1
            LEA R3, MyString2
            LDR R5, R3, #2
              ⋮      ⋮      ⋮
MyString1 .STRINGZ "volt"
              ⋮      ⋮      ⋮
MyString2 .STRINGZ "Sit"
              ⋮      ⋮      ⋮
```

Label	Address
MyString1	0631
MyString2	0659

ASCII Char	Hex Val *(as 16-bit)*
v	x0076
o	x006F
l	x006C
t	x0074
S	x0053
i	x0069
t	x0074

After the **LD** instruction is executed, what is the value in register **R1**? (answer as **4-digit hex**) 0076

After the **LEA** instruction is executed, what is the value in register **R3**? (answer as **4-digit hex**) 0659

After the **LDR** instruction is executed, what is the value in register **R5**? (answer as **4-digit hex**) 0074

Answer 1:

0076

Answer 2:

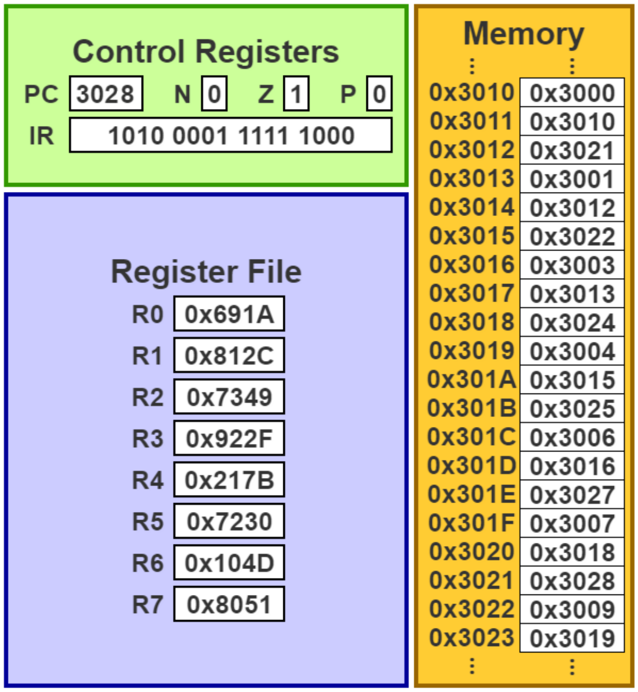
0659

Answer 3:

0074

Question 27

5 / 5 pts



Control Registers

PC 3028 N 0 Z 1 P 0

IR 1010 0001 1111 1000

Register File

R0	0x691A
R1	0x812C
R2	0x7349
R3	0x922F
R4	0x217B
R5	0x7230
R6	0x104D
R7	0x8051

Memory

⋮	⋮
0x3010	0x3000
0x3011	0x3010
0x3012	0x3021
0x3013	0x3001
0x3014	0x3012
0x3015	0x3022
0x3016	0x3003
0x3017	0x3013
0x3018	0x3024
0x3019	0x3004
0x301A	0x3015
0x301B	0x3025
0x301C	0x3006
0x301D	0x3016
0x301E	0x3027
0x301F	0x3007
0x3020	0x3018
0x3021	0x3028
0x3022	0x3009
0x3023	0x3019
⋮	⋮

State of LC-3 after PC incremented but before instruction executed.

This instruction involves 2 memory reads and 0 memory writes.

The first memory read is to **address** 0x3020

The second memory read is to **address** 0x3018

The first memory write is to **address** no writes performed

Answer 1:

2

Answer 2:

0

Answer 3:

0x3020

Answer 4:

0x3018

Answer 5:

no writes performed

Question 28

Not yet graded / 8 pts

NOTE: This is a manually graded question, therefore it will show as zero till graded. (max score of autograded portion is 79)

Download this template for a conditional swap routine (could be used in a bubble sort). Flesh out the code. It is to be coded as a subroutine, not a stand alone algorithm.

It is intended to employ the SWAP routine you wrote earlier in this exam (i.e call it). Do not, however, bother repeating your earlier SWAP routine code here.

When you are satisfied with your code copy and paste it into the "essay" box below.

Your Answer:

```
ST R2, CONDSWAP_R2 ; context save R2

; negate R0 into R2
NOT R2, R0
ADD R2, R2, #1

; compare R0 and R1
ADD R2, R1, R2
BRnz NOSWAP
ST R7, SAVE_R7
JSR SWAP
LD R7, SAVE_R7

NOSWAP
```

```
LD R2, CONDSWAP_R2
RET ; return from routine

; context save allocation ?

SAVE_R7 .BLKW 1
CONDSWAP_R2 .BLKW 1
```

Quiz Score: **76.25** out of 100