REPORT

LAB04 The Game of Nim

PB19010450 和泳毅

Requirements

Nim is a simple two-player game. There are many variations of this game with respect to the type of counters used (stones, matches, apples, etc.), the number of counters in each row, and the number of rows in the game board.

Rules

In our variation of Nim, the game board consists of three rows of rocks. **Row A** contains **3 rocks**, **Row B** contains **5 rocks**, and **Row C** contains **8 rocks**. The rules are as follows:

- Each player takes turns removing one or more rocks from a single row;
- A player cannot remove rocks from more than one row in a single turn;
- The game ends when a player removes the last rock from the game board. The player who removes the last rock loses.

What to do

1. At the beginning of the game you should display the initial state of the game board. Before each row of rocks you should output the name of the row, for instance "Row A:". You should use the **ASCII** character lowercase "o" (ASCII code **x006F**) to represent a rock. The initial state of the game board should look as follows:

$$\begin{aligned} &ROW\ A: \ ooo \\ &ROW\ B: \ ooooo \\ &ROW\ C: \ oooooooo \end{aligned}$$

2. Player 1 always goes first, and play alternates between Player 1 and Player 2. At the beginning of each turn you should output which players turn it is, and prompt the player for her move. For Player 1 this should look as follows:

$$Player\ 1, choose\ a\ row\ and\ number\ of\ rocks:$$

3. To specify which row and how many rocks to remove, the player should input a **letter** followed by a **number** (they do NOT need to press Enter after inputting a move). The letter (A, B, or C) specifies the row, and the number (from 1 to the number of rocks in the chosen row) specifies how many rocks to remove. Your program must make sure the players move has a valid row and number of rocks. If the players move is invalid, you should output an **error message** and prompt the same player for a move. For example, if it is Player 1s turn:

```
Player 1, choose a row and number of rocks: D4
Invalid move. Try again.

Player 1, choose a row and number of rocks: A9
Invalid move. Try again.

Player 1, choose a row and number of rocks: A*
Invalid move. Try again.

Player 1, choose a row and number of rocks: &4
Invalid move. Try again.

Player 1, choose a row and number of rocks:
```

4. Your program should **keep prompting** the player until a valid move is chosen. Be sure your program echoes the players move to the screen as they type it. After you have echoed the players move, you should output a **newline** character (ASCII code **x000A**) to move the cursor to the next line.
5. After a player has chosen a valid move, you should **check for a winner**. If there is one, display the appropriate banner declaring the winner. If there is no winner, your program should update the state of the game board to reflect the move, re-display the **updated game board**, and continue with the next players turn.
6. When a player has removed the last rock from the game board, the game is over. At this point, your program should display the winner and then halt. For example, if Player 2 removes the last rock, your program should output the following:

```
Player 1 Wins.
```

Notes and Suggestions:

1. Remember, all input and output functions use **ASCII** characters. You are responsible for making any conversions that are necessary.
2. For character input from the keyboard in this assignment, you should use **TRAP x20 (GETC).** To echo the characters onto the screen, you should follow each TRAP x20 with a **TRAP x21 (OUT)**. Recall that **TRAP x23** displays a banner to prompt the person at the keyboard to input a character. You do not need that banner since your program has its own style of prompt. Therefore you should use TRAP x20 which does the same as TRAP x23 except it does not print a banner on the screen to prompt for input.
3. You should use **subroutine**s where appropriate.
4. In each subroutine you write, you should save and **restore** any registers that you use. This will avoid a major headache during debugging
5. *A legitimate turn must contain the row, specified as A, B, or C (i.e., capital letter) followed by a number that is not larger than the number of rocks still remaining in that row.*

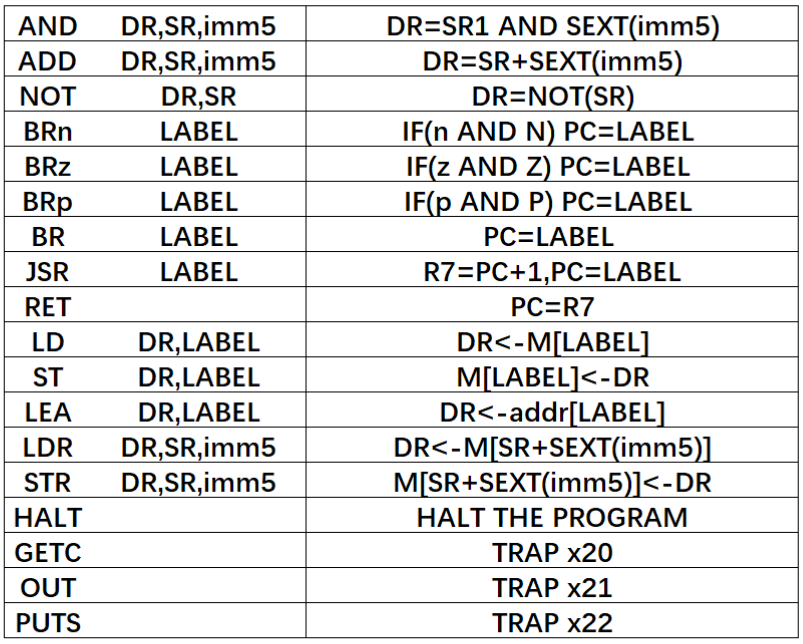
Design

1. Three counters are used to update the number of rocks in rows A, B and C at all times. And they are used to check the status of the game in progress after each time a player has chosen a valid move.

2. We know the **ASCII** code of "A" is **x0041**,so we can use the number **xFFBF** (-**x0041**) to check the Row A.The same to Row B and Row C.
3. We use a fixed register R1 to control the order of players. 0 means player 1 and 1 means player 2. Since player 1 starts the game first, we clear R1 first. After each player ends a valid move, the player is switched by taking the **NOT** of R1.
4. We know that the string ends with **x0000**. So we can apply this rule when we print the game status. For example, for the string "ROW A: ooo", we store the address of the last lowercase letter "o" and put **x0000** at that address. Printing again we will get "ROW A: oo".
5. The input we get through GETC is all characters. For example, if the player types "1", we get the character "1" instead of the integer value 1. However, it is easy to find that the decimal code of the character "1" is 49. So in order to get the integer value 1, we just need to let it subtract 48 .
6. We can use the string "\n" to replace the a **newline** character (ASCII code **x000A**) .
7. In each loop, we first check the game state to determine if the game is over. Then we check whether the player's input is valid or not. If it is invalid, a hint is performed. If it is valid, make a judgment of the row. Then we perform the calculation operation of the corresponding row. The computation operation consists of updating the number of rocks in each row and adjusting the terminator **x0000** of the string to be printed.

Code Writing

1. Instructions to be used



AND	DR,SR,imm5	DR=SR1 AND SEXT(imm5)
ADD	DR,SR,imm5	DR=SR+SEXT(imm5)
NOT	DR,SR	DR=NOT(SR)
BRn	LABEL	IF(n AND N) PC=LABEL
BRz	LABEL	IF(z AND Z) PC=LABEL
BRp	LABEL	IF(p AND P) PC=LABEL
BR	LABEL	PC=LABEL
JSR	LABEL	R7=PC+1,PC=LABEL
RET		PC=R7
LD	DR,LABEL	DR<-M[LABEL]
ST	DR,LABEL	M[LABEL]<-DR
LEA	DR,LABEL	DR<-addr[LABEL]
LDR	DR,SR,imm5	DR<-M[SR+SEXT(imm5)]
STR	DR,SR,imm5	M[SR+SEXT(imm5)]<-DR
HALT		HALT THE PROGRAM
GETC		TRAP x20
OUT		TRAP x21
PUTS		TRAP x22

2. Start at memory location **x3000**

```
        .ORIG   X3000
```

3. Initialization

```
INIT    AND     R1,R1,#0        ;Player 1 is 0,Player 2 is 1.
        LEA     R0,SHOW_A
        ADD     R0,R0,#12
        ST      R0,ADD_A
        LEA     R0,SHOW_B
        ADD     R0,R0,#13
        ST      R0,ADD_B
        LEA     R0,SHOW_C
        ADD     R0,R0,#15
        ADD     R0,R0,#1
        ST      R0,ADD_C
        LEA     R0,TITLE
        PUTS
```

4. Check and print the game state

```
LOOP    LD      R0,ROCKS_A
        BRnp    PRINT
        LD      R0,ROCKS_B
        BRnp    PRINT
        LD      R0,ROCKS_C
        BRz     CHECK_3         ;Game over and someone wins.
PRINT   LEA     R0,SHOW_A
        PUTS
        LEA     R0,SHOW_B
        PUTS
        LEA     R0,SHOW_C
;
CHECK_3 ADD     R1,R1,#0        ;Check the winner.
        BRnp    P2WINS
P1WINS  LEA     R0,WIN_1
        PUTS
        BR      END
P2WINS  LEA     R0,WIN_2
        PUTS
END     LEA     R0,OVER
        PUTS
        HALT
```

5. Check the player.

```
CHECK_1 ADD     R1,R1,#0        ;Check the player.
        BRnp    PL2
PL1     LEA     R0,SHOW_1
        PUTS
        BR      INPUT
PL2     LEA     R0,SHOW_2
        PUTS
```

6. Check the Row

```
CHECK_2 LD      R3,A            ;Check the rows.
        LD      R4,ROW
        ADD     R3,R3,R4        ;Check if the row is A
        BRz     INPUTA
        LD      R3,B
        ADD     R3,R3,R4        ;Check if the row is B
        BRz     INPUTB
        LD      R3,C
        ADD     R3,R3,R4        ;Check if the row is B
        BRz     INPUTC
        BRnp    INV             ;Else invalid input
;
INPUTA  LD      R2,ROCKS_A
        LD      R3,ADD_A
        JSR     CAL
        ST      R2,ROCKS_A
        ST      R3,ADD_A
        NOT     R1,R1           ;Change the player.
        BR      LOOP
;
INPUTB  LD      R2,ROCKS_B
        LD      R3,ADD_B
        JSR     CAL
        ST      R2,ROCKS_B
        ST      R3,ADD_B
        NOT     R1,R1
        BR      LOOP
;
INPUTC  LD      R2,ROCKS_C
        LD      R3,ADD_C
        JSR     CAL
        ST      R2,ROCKS_C
        ST      R3,ADD_C
        NOT     R1,R1
        BR      LOOP
;
INV     LEA     R0,INVALID      ;Invalid input
        PUTS
        BR      CHECK_1
```

7. Calculate

```
CAL     LD      R4,ROCKS        ;CALCULATE
        LD      R5,N48
        ADD     R4,R4,R5        ;Char to int.
        BRnz    INV
        ADD     R2,R4,R2
        BRp     INV
        NOT     R4,R4
        ADD     R4,R4,#1        ;Negative number.
        ADD     R3,R3,R4
        AND     R5,R5,#0
        STR     R5,R3,#0
        RET
```

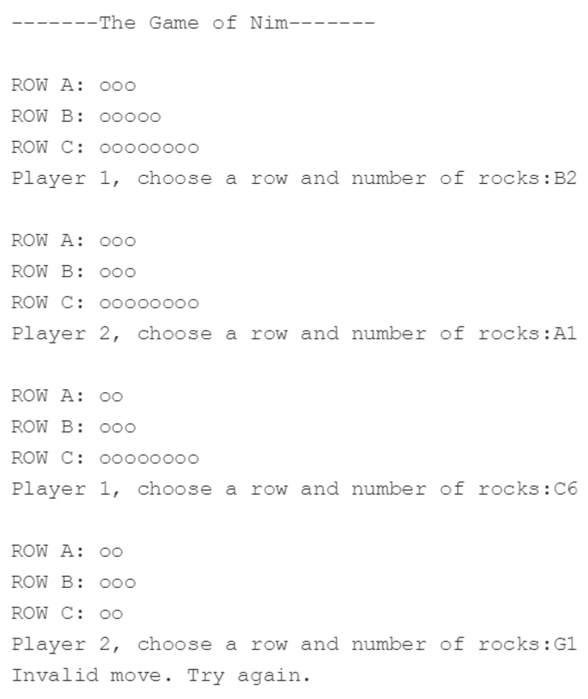
8. Non-code section

```
ROW     .FILL   x0000
ROCKS   .FILL   x0000
A       .FILL   xFFBF       ;-65,A is 65 in decimal.
B       .FILL   xFFBE       ;-66
C       .FILL   xFFBD       ;-67
ROCKS_A .FILL   xFFFD       ;-3,Rocks in row A now.
ROCKS_B .FILL   xFFFB       ;-5
ROCKS_C .FILL   xFFF8       ;-8
N48     .FILL   xFFD0       ;-48
```

```
ADD_A   .FILL   x0000
ADD_B   .FILL   x0000
ADD_C   .FILL   x0000
TITLE   .STRINGZ "-------The Game of Nim-------"
SHOW_A  .STRINGZ "\n\nROW A: ooo"
SHOW_B  .STRINGZ "\nROW B: ooooo"
SHOW_C  .STRINGZ "\nROW C: oooooooo"
SHOW_1  .STRINGZ "\nPlayer 1, choose a row and number of rocks:"
SHOW_2  .STRINGZ "\nPlayer 2, choose a row and number of rocks:"
WIN_1   .STRINGZ "\n\nPlayer 1 Wins.\n"
WIN_2   .STRINGZ "\n\nPlayer 2 Wins.\n"
INVALID .STRINGZ "\nInvalid move. Try again."
OVER    .STRINGZ "-------Game Over-------"
```

Result Test

1. The example



```
-------The Game of Nim-------

ROW A: ooo
ROW B: ooooo
ROW C: oooooooo
Player 1, choose a row and number of rocks:B2

ROW A: ooo
ROW B: ooo
ROW C: oooooooo
Player 2, choose a row and number of rocks:A1

ROW A: oo
ROW B: ooo
ROW C: oooooooo
Player 1, choose a row and number of rocks:C6

ROW A: oo
ROW B: ooo
ROW C: oo
Player 2, choose a row and number of rocks:G1
Invalid move. Try again.
```

```
Player 2, choose a row and number of rocks:B3

ROW A: oo
ROW B:
ROW C: oo
Player 1, choose a row and number of rocks:A3
Invalid move. Try again.
Player 1, choose a row and number of rocks:C2

ROW A: oo
ROW B:
ROW C:
Player 2, choose a row and number of rocks:A1

ROW A: o
ROW B:
ROW C:
Player 1, choose a row and number of rocks:A*
Invalid move. Try again.
Player 1, choose a row and number of rocks:&4
Invalid move. Try again.
Player 1, choose a row and number of rocks:A1

Player 2 Wins.
```

2.

```
-------The Game of Nim-------

ROW A: ooo
ROW B: ooooo
ROW C: oooooooo
Player 1, choose a row and number of rocks:C9
Invalid move. Try again.
Player 1, choose a row and number of rocks:C8

ROW A: ooo
ROW B: ooooo
ROW C:
Player 2, choose a row and number of rocks:A2

ROW A: o
ROW B: ooooo
ROW C:
Player 1, choose a row and number of rocks:B3

ROW A: o
ROW B: oo
ROW C:
Player 2, choose a row and number of rocks:B1

ROW A: o
ROW B: o
ROW C:
Player 1, choose a row and number of rocks:C1
Invalid move. Try again.
Player 1, choose a row and number of rocks:B1

ROW A: o
ROW B:
ROW C:
Player 2, choose a row and number of rocks:A1

Player 1 Wins.
```

Thinking

1. Functions that are used repeatedly can be written as sub-code for invocation；
2. The string terminator x0000 can be used flexibly to better control the printing of strings；
3. The string is occupying one more memory than the number of characters in the string.

Appendix

Complete code:

LC-3：

```
        .ORIG x3000
INIT    AND     R1,R1,#0        ;Player 1 is 0,Player 2 is 1.
        LEA     R0,SHOW_A
        ADD     R0,R0,#12
        ST      R0,ADD_A
        LEA     R0,SHOW_B
        ADD     R0,R0,#13
        ST      R0,ADD_B
        LEA     R0,SHOW_C
        ADD     R0,R0,#15
        ADD     R0,R0,#1
        ST      R0,ADD_C
        LEA     R0,TITLE
        PUTS
;
LOOP    LD      R0,ROCKS_A
        BRnp    PRINT
        LD      R0,ROCKS_B
        BRnp    PRINT
        LD      R0,ROCKS_C
        BRz     CHECK_3         ;Game over and someone wins.
;
PRINT   LEA     R0,SHOW_A
        PUTS
        LEA     R0,SHOW_B
        PUTS
        LEA     R0,SHOW_C
        PUTS
;
CHECK_1 ADD     R1,R1,#0        ;Check the player.
        BRnp    PL2
PL1     LEA     R0,SHOW_1
        PUTS
        BR      INPUT
PL2     LEA     R0,SHOW_2
        PUTS
;
INPUT   AND     R0,R0,#0
        GETC
        ST      R0,ROW
        OUT
        GETC
        ST      R0,ROCKS
        OUT
CHECK_2 LD      R3,A            ;Check the rows.
        LD      R4,ROW
        ADD     R3,R3,R4        ;Check if the row is A
        BRz     INPUTA
        LD      R3,B
        ADD     R3,R3,R4        ;Check if the row is B
        BRz     INPUTB
        LD      R3,C
        ADD     R3,R3,R4        ;Check if the row is B
        BRz     INPUTC
        BRnp    INV             ;Else invalid input
;
INPUTA  LD      R2,ROCKS_A
        LD      R3,ADD_A
        JSR     CAL
        ST      R2,ROCKS_A
        ST      R3,ADD_A
        NOT     R1,R1           ;Change the player.
        BR      LOOP
;
INPUTB  LD      R2,ROCKS_B
        LD      R3,ADD_B
        JSR     CAL
        ST      R2,ROCKS_B
        ST      R3,ADD_B
        NOT     R1,R1
        BR      LOOP
;
INPUTC  LD      R2,ROCKS_C
        LD      R3,ADD_C
        JSR     CAL
        ST      R2,ROCKS_C
        ST      R3,ADD_C
        NOT     R1,R1
        BR      LOOP
;
INV     LEA     R0,INVALID      ;Invalid input
        PUTS
        BR      CHECK_1
;
CHECK_3 ADD     R1,R1,#0        ;Check the winner.
        BRnp    P2WINS
P1WINS  LEA     R0,WIN_1
        PUTS
        BR      END
P2WINS  LEA     R0,WIN_2
        PUTS
END     LEA     R0,OVER
        PUTS
        HALT                    ;Game Over.
;
CAL     LD      R4,ROCKS        ;CALCULATE
        LD      R5,N48
        ADD     R4,R4,R5        ;Char to int.
        BRnz    INV
        ADD     R2,R4,R2
        BRp     INV
        NOT     R4,R4
        ADD     R4,R4,#1        ;Negative number.
        ADD     R3,R3,R4
        AND     R5,R5,#0
        STR     R5,R3,#0
        RET
;
ROW     .FILL   x0000
ROCKS   .FILL   x0000
A       .FILL   xFFBF       ;-65,A is 65 in decimal.
B       .FILL   xFFBE       ;-66
C       .FILL   xFFBD       ;-67
ROCKS_A .FILL   xFFFD       ;-3,Rocks in row A now.
ROCKS_B .FILL   xFFFB       ;-5
ROCKS_C .FILL   xFFF8       ;-8
N48     .FILL   xFFD0       ;-48
ADD_A   .FILL   x0000
ADD_B   .FILL   x0000
ADD_C   .FILL   x0000
TITLE   .STRINGZ "-------The Game of Nim-------"
SHOW_A  .STRINGZ "\n\nROW A: ooo"
SHOW_B  .STRINGZ "\nROW B: ooooo"
SHOW_C  .STRINGZ "\nROW C: oooooooo"
SHOW_1  .STRINGZ "\nPlayer 1, choose a row and number of rocks:"
SHOW_2  .STRINGZ "\nPlayer 2, choose a row and number of rocks:"
WIN_1   .STRINGZ "\n\nPlayer 1 Wins.\n"
WIN_2   .STRINGZ "\n\nPlayer 2 Wins.\n"
INVALID .STRINGZ "\nInvalid move. Try again."
OVER    .STRINGZ "-------Game Over-------"
        .END
```