605.204 - Computer Organization
Module 9: Assignment

Nick Hinke

October 30, 2022

Brief Introduction

This assignment involves branching and looping within various assembly files in armv7l. All of my resulting code can be found at this *GitHub link* and can be cloned (along with pre-built binaries in a *bin/* folder) and viewed using the following commands:

```
git clone https://github.com/nhinke/computer-organization-repo.git
cd computer-organization-repo/assignments/module09/
```

The pre-built binaries can then be run using the following commands:

```
cd bin/
./checkCharLogical
./checkCharNoLogicals
./checkGrade
./maxInt
```

Note that each of the pre-built binaries will print out an example input-output sequence to the active terminal.

Problem 1

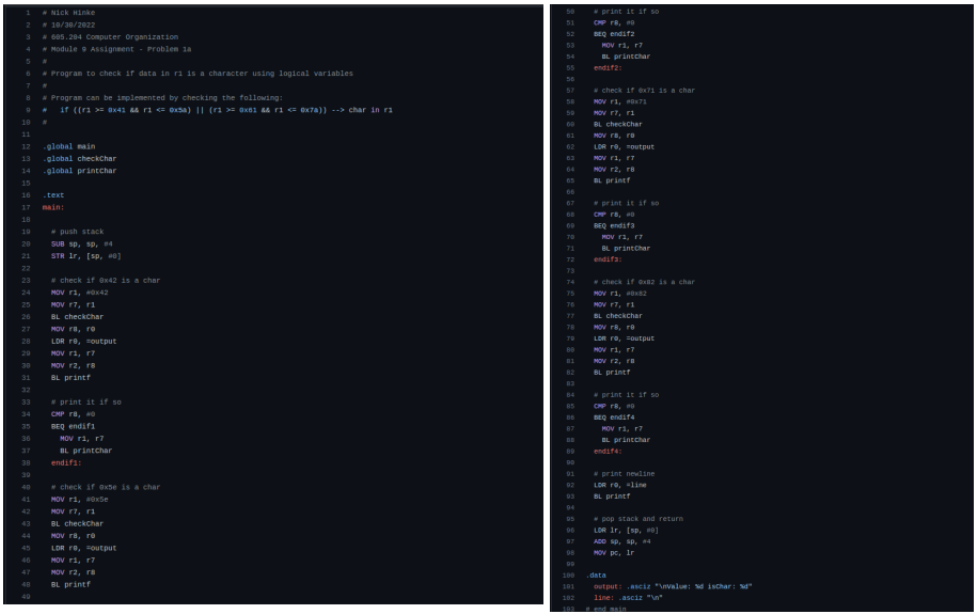
1. Write and test a function to check if a user input value is a character or not. Implement it in two ways:

a. As a logical variable.

b. Any way that does not use logical values.

Using Logical Vars

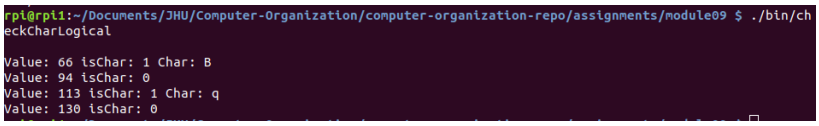
Program:



```
# Nick Hinke
# 10/30/2022
# 605.204 Computer Organization
# Module 9 Assignment - Problem 1a
#
# Program to check if data in r1 is a character using logical variables
#
# Program can be implemented by checking the following:
#   if ((r1 >= 0x41 && r1 <= 0x5a) || (r1 >= 0x61 && r1 <= 0x7a)) --> char in r1
#

.global main
.global checkChar
.global printChar

.text
main:

 # push stack
 SUB sp, sp, #4
 STR lr, [sp, #0]

 # check if 0x42 is a char
 MOV r1, #0x42
 MOV r7, r1
 BL checkChar
 MOV r8, r0
 LDR r0, =output
 MOV r1, r7
 MOV r2, r8
 BL printf

 # print it if so
 CMP r8, #0
 BEQ endif1
   MOV r1, r7
   BL printChar
 endif1:

 # check if 0x5e is a char
 MOV r1, #0x5e
 MOV r7, r1
 BL checkChar
 MOV r8, r0
 LDR r0, =output
 MOV r1, r7
 MOV r2, r8
 BL printf

 # print it if so
 CMP r8, #0
 BEQ endif2
   MOV r1, r7
   BL printChar
 endif2:

 # check if 0x71 is a char
 MOV r1, #0x71
 MOV r7, r1
 BL checkChar
 MOV r8, r0
 LDR r0, =output
 MOV r1, r7
 MOV r2, r8
 BL printf

 # print it if so
 CMP r8, #0
 BEQ endif3
   MOV r1, r7
   BL printChar
 endif3:

 # check if 0x82 is a char
 MOV r1, #0x82
 MOV r7, r1
 BL checkChar
 MOV r8, r0
 LDR r0, =output
 MOV r1, r7
 MOV r2, r8
 BL printf

 # print it if so
 CMP r8, #0
 BEQ endif4
   MOV r1, r7
   BL printChar
 endif4:

 # print newline
 LDR r0, =line
 BL printf

 # pop stack and return
 LDR lr, [sp, #0]
 ADD sp, sp, #4
 MOV pc, lr

.data
 output: .asciz "\nvalue: %d isChar: %d"
 line: .asciz "\n"
```

Figure 1: Screenshot of program to check if data is a char using logicals

Example:



```
rpi@rpi1:~/Documents/JHU/Computer-Organization/computer-organization-repo/assignments/module09 $ ./bin/ch
eckCharLogical

Value: 66 isChar: 1 Char: B
Value: 94 isChar: 0
Value: 113 isChar: 1 Char: q
Value: 130 isChar: 0
```

Figure 2: Screenshot of program output

Without Using Logical Vars

Program:

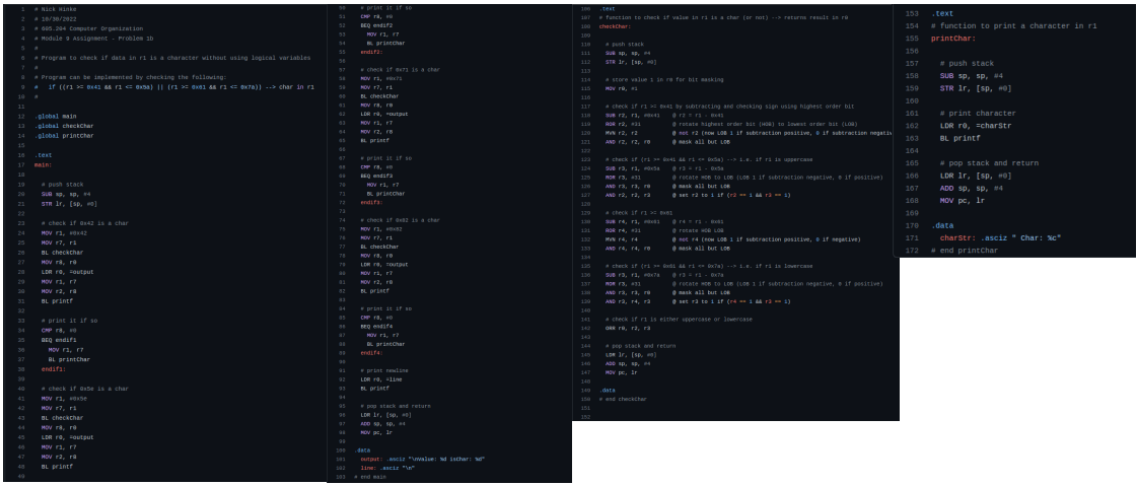


Figure 3: Screenshot of program to check if data is a char *without* using logicals

Example:

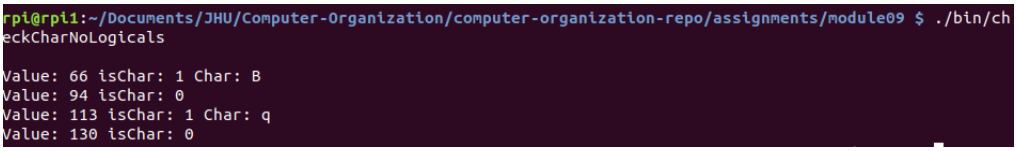


Figure 4: Screenshot of program output

Problem 2

2. Implement a grading program as follows. It should follow the proper style for assembly.

1. *Prompt for a name and an average.*
2. *If the average is not on range [0,100], print an error.*
3. *Calculate a grade as 90-100 as A, 80-90 as B, 70-80 as C, else F.*
4. *Print out the student's name and grade.*

Program *main* label:

```
.global main
.global checkGradeValid
.global printLetterGrade


.text
main:

  # push stack
  SUB sp, sp, #4
  STR lr, [sp, #0]

  # prompt and read stduent name from user
  LDR r0, =promptName
  BL printf
  LDR r0, =formatName
  LDR r1, =name
  BL scanf

  # prompt and read number grade for student from user
  LDR r0, =promptGrade
  LDR r1, =name
  BL printf
  LDR r0, =formatGrade
  LDR r1, =grade
  BL scanf

  # check number grade valid using checkGradeValid function (result returned in r0)
  LDR r0, =grade
  LDR r0, [r0, #0]
  BL checkGradeValid

  # if number grade valid, print letter grade using printLetterGrade function
  CMP r0, #1
  BNE endMain
    LDR r0, =name
    LDR r1, =grade
    LDR r1, [r1, #0]
    BL printLetterGrade

  endMain:

  # pop stack and return
  LDR lr, [sp, #0]
  ADD sp, sp, #4
  MOV pc, lr

.data
  promptName:  .asciz "Please enter the student's name: "
  promptGrade: .asciz "Please enter an integer grade on [0,100] for %s: "
  formatName:  .asciz "%s"
  formatGrade: .asciz "%d"
  name: .space 32
  grade: .word 0
```

Figure 5: Screenshot of main label within grade conversion program

Visit *this link* to see the rest of the program (it is too long to paste screenshots in a meaningful way).

Example:

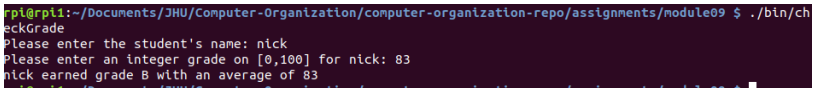


Figure 6: Screenshot of sample program output

Problem 3

3. Implement a function to find the largest of 3 values. The function signature is "findMaxOf3(int val1, int val2, int val3)". The function should compare the 3 values, and return the largest. Be sure to use the ABI standards for all arguments and return values. Write a program to prompt for 3 values, call this function to the find the maximum value, and print out the maximum value. Follow proper style for assembly.

Program *main* and *findMaxOf3* labels:

```
# Nick Hinke
# 10/23/2022
# 605.204 Computer Organization
# Module 9 Assignment - Problem 3
#
# Program to find the largest of three integer values
#

.global main
.global promptForInteger
.global findMaxOf3

.text
main:

  # push the stack
  SUB sp, sp, #4
  STR lr, [sp, #0]

  # print welcome string
  LDR r0, =numIntsStr
  BL printf

  # prompt user for three integers and store in r5-r7
  BL promptForInteger
  MOV r5, r0
  BL promptForInteger
  MOV r6, r0
  BL promptForInteger
  MOV r7, r0

  # move integers into r0-r2 and branch to comparison function
  MOV r0, r5
  MOV r1, r6
  MOV r2, r7
  BL findMaxOf3

  # print result stotred in r0
  MOV r1, r0
  LDR r0, =endLoopStr
  BL printf

  # pop stack and return
  LDR lr, [sp, #0]
  ADD sp, sp, #4
  MOV pc, lr

.data
  numIntsStr: .asciz "Hello! Program ready to compare 3 integers...\n"
  endLoopStr: .asciz "\nMaximum integer found: %d\n"
# end main
```

```
findMaxOf3:

  # push the stack
  SUB sp, sp, #4
  STR lr, [sp, #0]

  # use r3 to contain maximum --> initially set to minimum possible int
  MOV r3, #1
  ROR r3, #1
  B CompareInt1

  # compare first integer to current maximum
  CompareInt1:
    CMP r0, r3
    BLE CompareInt2
      MOV r3, r0
      B CompareInt2

  # compare second integer to current maximum
  CompareInt2:
    CMP r1, r3
    BLE CompareInt3
      MOV r3, r1
      B CompareInt3

  # compare third integer to current maximum
  CompareInt3:
    CMP r2, r3
    BLE endComparisons
      MOV r3, r2
      B endComparisons

  # store result in r0
  endComparisons:
    MOV r0, r3

  # pop stack and return
  LDR lr, [sp, #0]
  ADD sp, sp, #4
  MOV pc, lr

.data
# end findMaxOf3
```

Figure 7: Screenshot of main and findMaxOf3 labels within program

Visit *this link* to see the rest of the program (it is too long to paste screenshots in a meaningful way).

Example:

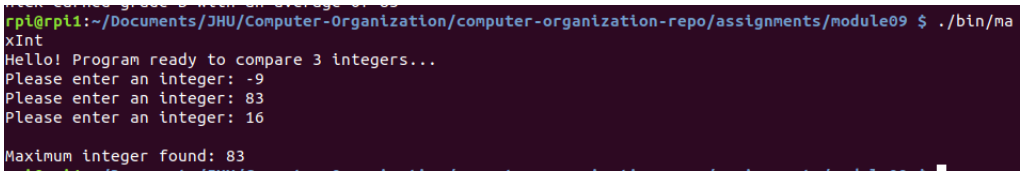


Figure 8: Screenshot of sample program output