Verificación pre-silicio Primavera 2022

lab3.C problem

professor:Gloria Castro Muñoz

Team 4:
José Alberto Gómez Díaz
Ricardo Bazaldúa Calderón
Perla Noemí Méndez Zavaleta
Christian Aaron Ortega Blanco

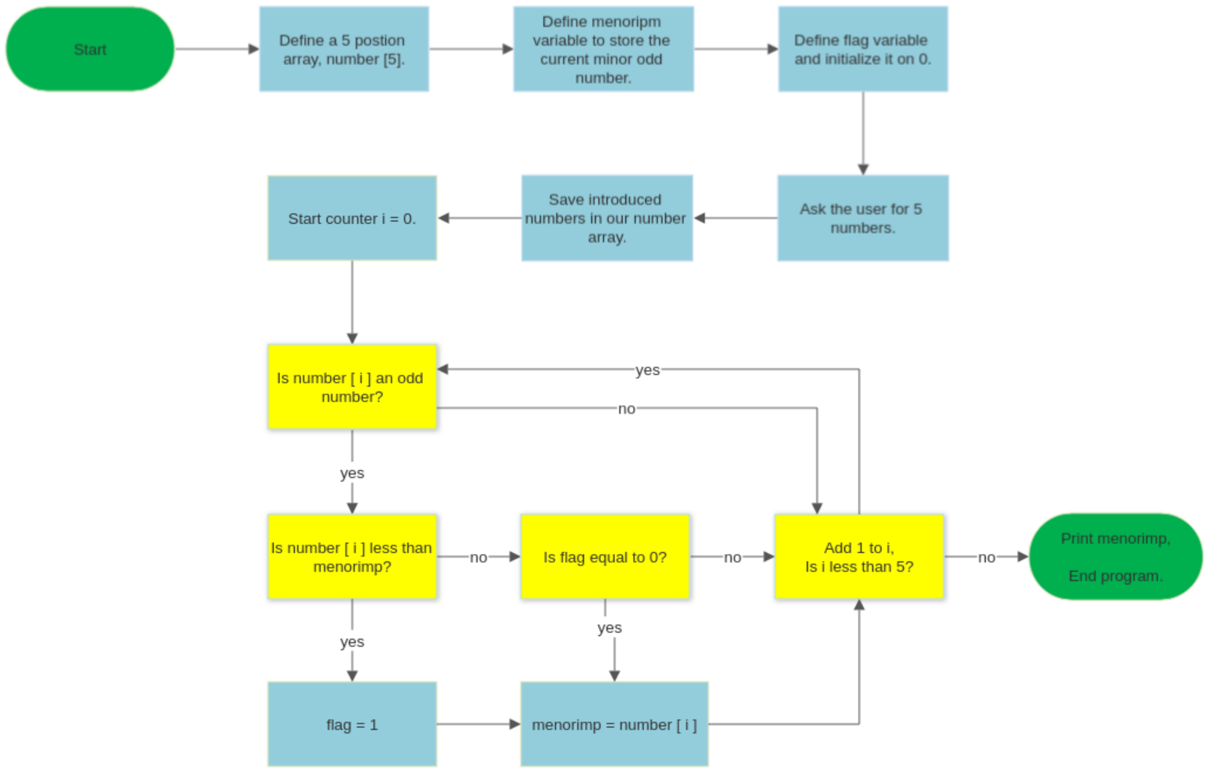
méxico, 26/05/2022

A. By the given numbers f,g,h,i,j and assuming that at least one of them is an odd number, make a C program that decides the smallest odd number.
B. Translate fromC program to MIPS assembler.
C. Translate from assembler to machine code.

note: using arrays to store the entry numbers.

index:

1. Flux Diagram



2. C Program

Variables *number, menorimp, flag* and *modulo* were initialized. Where *number* is a 5-bit vector, *menorimp* is the variable where the lowest odd number will be saved, *flag* is initialized in 0 as well as *modulo.*

```
int number[5];
int menorimp;
int flag=0;
int modulo=0;
```

Print the letters F, G, H, I, J on the screen to ask the user the 5 numbers to be saved in the variable *number*.

```c
    printf("F ");
    scanf("%d", &number[0]);
    printf("G ");
    scanf("%d", &number[1]);
    printf("H ");
    scanf("%d", &number[2]);
    printf("I ");
    scanf("%d", &number[3]);
    printf("J ");
    scanf("%d", &number[4]);
    // displays output
```

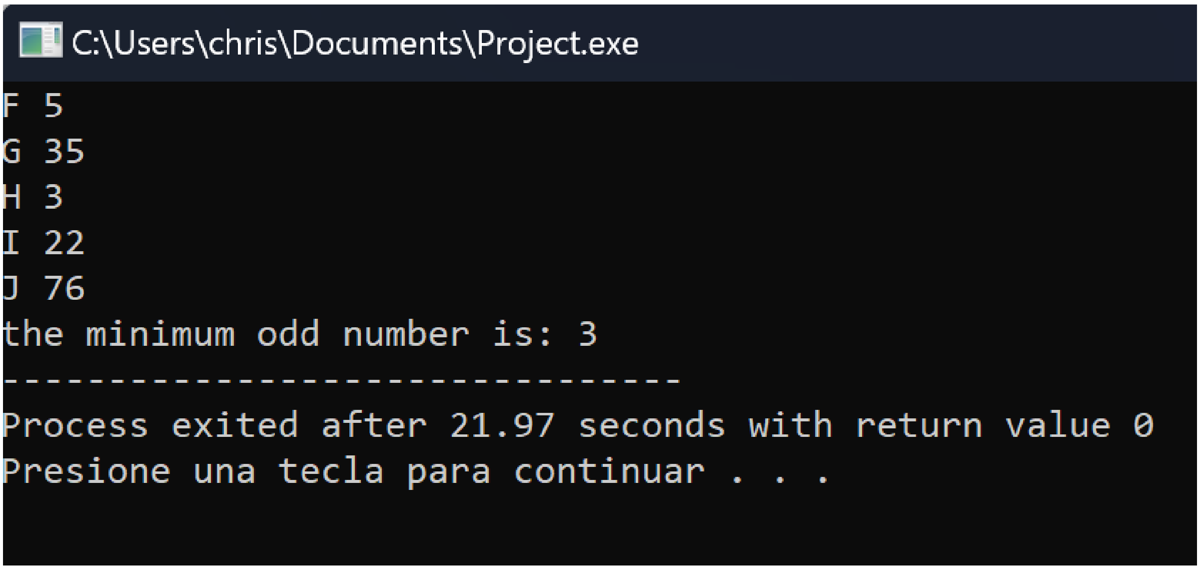
A *for* cycle was then used to identify which numbers are even. The *if* function helped us to separate even numbers from odd numbers with a flag, so we could only work with odd numbers and determine which is the smallest of them.

```c
for(int i=0;i<5;i++){
        modulo=number[i]/2;
        modulo=modulo*2;
        modulo=number[i]-modulo;

        if ( modulo == 1) {

            if(!flag){
                menorimp=number[i];
                flag=1;
            }
            else if(number[i]<menorimp) menorimp=number[i];


        }
}
```

Finally, the result was printed on the console with *printf*

```c
    printf("the minimum odd number is: %d", menorimp);
    return 0;
}
```

3. C Simulation



4. MIPS ASSEMBLER CODE

```
# Declare main as a global function
        .globl main
        number: .space 20 # 5 spaces in array

# The label 'main' represents the starting point
# number: .word 6,7,8,5,2
main:
        addi $s1,$0,0  #sum
        addi $s0,$0,0  #i
        addi $t0,$0,5  #t0=5
        #set the numbers
        addi $s2, $zero, 6
        addi $s3, $zero, 7
        addi $s4, $zero, 8
        addi $s5, $zero, 5
        addi $s6, $zero, 2
#index = $t1 array
                addi $t1, $zero, 0
        sw $s2, number($t1)
                addi $t1, $t1, 4
        sw $s3, number($t1)
                addi $t1, $t1, 4
        sw $s4, number($t1)
                addi $t1, $t1, 4
        sw $s5, number($t1)
                addi $t1, $t1, 4
        sw $s6, number($t1)
        addi $t1, $zero, 0 #clear $t1 to 0
        #variables for loop
        addi $s2, $zero, 0  # flag
        addi $s3, $zero, 0  # menorimp
loop: bne $s0,$t0,for #if i !=5 salta a for
                j end
```

```
for:
#finding module  1 odd, 0 even
            lw $t3, number($t1)
            div $s1,$t3,2
            mul $s1,$s1,2
            sub $s1, $t3, $s1  #$s1 module

            bne $s1, 1, else  #if module !=1 go else
                        bgtz $s2,findless #if flag > 0
                                    move $s3, $t3
                                    addi $s2, $zero, 1
                        findless:
                                    bgt $t3,$s3,else
                                    move $s3, $t3 #store result on S3
else:
            addi $t1, $t1, 4
            addi $s0, $s0,1 #i=i+1

            j loop
end:
            li $v0, 10   # Sets $v0 to "10" to select exit syscall
            syscall   # Exit
```

5. MIPS SIMULATION

QTSPIM it's a very simple compiler for MIPS assembler, once the .asm code is done just load the file using File/Reinitialize and load file, otherwise a register error could occur, then press the button to run the program.

The smallest odd number is stored on register s3.

```
BadVAddr = 0
Status   = 805371664

HI       = 0
LO       = 2

R0  [r0] = 0
R1  [at] = 1
R2  [v0] = 10
R3  [v1] = 0
R4  [a0] = 1
R5  [a1] = 2147480020
R6  [a2] = 2147480028
R7  [a3] = 0
R8  [t0] = 5
R9  [t1] = 20
R10 [t2] = 0
R11 [t3] = 2
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 5
R17 [s1] = 0
R18 [s2] = 1
R19 [s3] = 5
R20 [s4] = 8
R21 [s5] = 5
R22 [s6] = 2
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
R28 [gp] = 268468224
R29 [sp] = 2147480016
R30 [s8] = 0
R31 [ra] = 4194328
```

6. Machine code

In order to convert an assembly program to machine code we had to look at the structure of every instruction to realize what kind of instruction it is. Every instruction has an opcode so we can realize what kind of instruction it belongs to. After that we had to identify the value for all the registers used in the instruction. and place it in the correct position.

I type has 4 sections. These sections are opcode, rs,rt and an immediate in that order.

I Type			
op	rs	rt	imm
001000	00000	10001	0000000000000000

R type has 5 sections. These sections are opcode,rs,rt,rd,shamt and function.

R Type					
op	rs	rt	rd	shamt	funct
000000	00000	01011	10011	00000	100001

J Type has 2 sections. These sections are opcode and address.

J Type	
op	addr
000010	00000100000000000000000000

Finally we got this code

```
00100000000100010000000000000000
00100000000100000000000000000000
00100000000010000000000000000101
00100000000100100000000000000110
00100000000100110000000000000111
00100000000101000000000000001000
00100000000101010000000000000101
00100000000101100000000000000010
00100000000010010000000000000000
00111100000000010000000001000000
00000000001010010000100000100001
10101100001100100000000000100100
00100001001010010000000000000100
00111100000000010000000001000000
00000000001010010000100000100001
10101100001100110000000000100100
00100001001010010000000000000100
00111100000000010000000001000000
```

```
00000000001010010000100000100001
10101100001101000000000000100100
00100001001010010000000000000100
00111100000000010000000001000000
00000000001010010000100000100001
10101100001101010000000000100100
00100001001010010000000000000100
00111100000000010000000001000000
00000000001010010000100000100001
10101100001101100000000000100100
00100000000010010000000000000000
00100000000100100000000000000000
00100000000100110000000000000000
00010110000010000000000000000010
00001000000100000000000001000011
00111100000000010000000001000000
00000000001010010000100000100001
10001100001010110000000000100100
00110100000000010000000000000010
00000001011000010000000000011010
00000000000000001000100000010010
00110100000000010000000000000010
01110000010110001000000000000010
00000001011100011000100000100010
00110100000000010000000000000001
00010100001100010000000000000111
00011110010000000000000000000011
00000000000010111001100000100001
00100000000100100000000000000001
00000010011010110000100000101010
00010100001000000000000000000010
00000000000010111001100000100001
00100001001010010000000000000100
00100010000100000000000000000001
00001000000100000000000000101101
00110100000000100000000000001010
00000000000000000000000000001100
```

7. Conclusions

As shown on the simulation results, the behavior of the output it's pretty much alike on both programs. visualizing that the assembler code as a different structure, the C program had to pass through different ways of thinking to ensure that the result is going to be almost directly converted to MIPS. contemplating that, the assembler programing was very step forward.

8. References

- James Larus. (2018). SPIM: A MIPS32 Simulator. 26/05/2022, de Morgan Kaufmann Sitio web: http://spimsimulator.sourceforge.net/
- Amell Peralta. (2015). MIPS Tutorial 28 Printing an Array with a While Loop. 26/05/2022, de youtube Sitio web: https://www.youtube.com/watch?v=Vb8kuvxc4NE