Module 3 - Problems:

Programming in assembly

Introduction to computers II

Juan Lanchares Dávila
Fernando Castro Rodríguez
Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid

1) Write a RISC-V assembly program that implements the following code. Use the **`.data`** section to assign the initial value of the variables.

```
int x = 10, y = 5;
if (x >= y) {
  x = x + 2;
  y = y - 2;
}
```

```
.global main                # "main" label is global
.data                       # Data with initial value
x: .word 10                 # Declares variables with initial value
y: .word 5

.text                       # Instructions
main:

        la t0,x             # pseudo-instruction t0=@x
        la t1,y             # pseudo-instruction t1=@y
        lw s1,0(t0)         # s1=10
        lw s2,0(t1)         # s2=5
        blt s1,s2,exit      # reverse condition s1<s2
        addi s1,s1,2        # x=x+2
        addi s2,s2,-2       # y=y-2
        sw s1,0(t0)
        sw s2,0(t1)
exit:
        j .                 # last instruction
.end                        # end of file directive
```

2) Write a RISC-V assembly program that implements the following code. Use the **`.data`** section to assign the initial value of the variables.

```
int x = 5, y = 10;
if (x >= y) {
  x = x + 2;
  y = y + 2;
}
else {
   x = x - 2;
   y = y - 2;
}
```

```
.global main

.data                       # Data with initial value
x: .word 5
y: .word 10

.text                       # Instruction
main:
        la t0,x             # pseudo-instruction t0=@x
        la t1,y             # pseudo-instruction t1=@y
        lw s1,0(t0)         # s1=5
        lw s2,0(t1)         # s2=10
        blt s1,s2, else     # reverse condition s1 <s2
        addi s1,s1,2        # x=x+2
        addi s2,s2,2        # y=y+2
        j endif
else:
        addi s1,s1,-2       # x = x -2;
        addi s2,s2,-2       # y = y -2
endif:
        sw s1,0(t0)
        sw s2,0(t1)
exit:
        j .
.end
```

3) Write a RISC-V assembly program that implements the following code. Use the **`.bss`** section to reserve memory space for the variables.

```
int a, b;
a = 81;
b = 18;
do {
  a = a - b;
} while (a > 0);
```

```
.global main
.bss
a: .space 4
b: .space 4

.text
main:
        la t1,a
        la t2,b
        li s1,81
        sw s1,0(t1)       # a=81
        li s2,18
        sw s2,0(t2)       # b=18
do:
        sub s1,s1,s2      # a=a-b
        sw  s1,0(t1)
while:
        bgt s1,zero,do    # direct condition
exit:
        j .
.end
```

4) Write a RISC-V assembly program that implements the following code. Use the **`.bss`** section to reserve memory space for the variables.

```
int n, fprev, f, i, faux;
n = 5;
fprev = 1;
f = 1;
i = 2;
while (i <= n) {
  faux = f;
  f = f + fprev;
  fprev = faux;
  i = i + 1;
}
```

```
.global main
.bss
n:      .space 4
fprev:  .space 4
f:      .space 4
i:      .space 4
faux:   .space 4

.text
main:
        la t1,f
        li s1,1          # f in s1
        sw s1,0(t1)
        la t0,i
        li s2,2          # i in s2
        sw s2,0(t0)
        la t0,fprev
        li s3,1          # fprev in s3
        sw s3,0(t0)
        la t0,n
        li s4,5          # n in s4
        sw s4,0(t0)
```

4) Write a RISC-V assembly program that implements the following code. Use the **`.bss`** section to reserve memory space for the variables.

```
int n, fprev, f, i, faux;
n = 5;
fprev = 1;
f = 1;
i = 2;
while (i <= n) {
  faux = f;
  f = f + fprev;
  fprev = faux;
  i = i + 1;
}
```

```
while:
        bgt s2,s4,exit
        mv t2,s1                    # t2 = f (=faux)
        la t3,faux
        sw t2,0(t3)
        add s1,s1,s3                # f = f + fprev
        sw s1,0(t1)                 # store f
        mv s3,t2                    # update fprev and store
        la t3,fprev
        sw s3,0(t3)
        addi s2,s2,1                # update i and store it
        la t3,i
        sw s2,0(t3)
        j while
exit:
        j .
.end
```

5) Write a RISC-V assembly program that implements the following code. Use the **`.data`** section to assign the initial value of variables f and n, and the **`.bss`** section to reserve memory space for variable i.

```
int f = 2, n = 5;
int i;
for (i = 2; i <= n; i++)
  f = f + f;
```

```
.global main
.equ n,5
.data
f: .word 2
n: .word 5
.bss
i: .space 4

.text
main :
        la t1,f          # t1=@f
        lw s1,0(t1)      # s1 is f
        la t2,n          # t2=@n
        lw s2,0(t2)      # s2 is n

        la t2,i
        li s3,2          # s3 is i
        sw s3,0(t2)
for:
        bgt s3,s2,exit   # reverse condition
        add s1,s1,s1
        sw s1,0(t1)
        addi s3,s3,1
        sw s3,0(t2)
        j for
exit:
        j .
.end
```

6) The following program calculates the greatest common divisor of two numbers a and b according to the Euclidean algorithm. Write a RISC-V assembly program that implements the following code. Use the **`.data`** section to assign the initial value of variables a and b, and the **`.bss`** section to reserve memory space for variable gcd.

```
int a=5, b=15, gcd;
while (a ≠ b) {
  if (a > b)
    a = a - b;
   else
     b = b - a;
}
gcd = a;
```

```
.global main
.data
a: .word 5
b: .word 15
gcd: .space 4

.text
main:
        la t1,a
        lw s1,0(t1)       # s1 is a
        la t2,b
        lw s2,0(t2)       # s2 is b
while:
        beq s1,s2,endwhile
        ble s1,s2,else
        sub s1,s1,s2      # a=a-b
        j endif
else:
        sub s2,s2,s1      # b=b-a
endif:
        j while
endwhile:
        la t3,gcd         # t3 = @gcd
        sw s1,0(t3)
exit:
        j .
.end
```

7) Write a RISC-V assembly program that implements the following code:

```
int f, g, h, B[10]
f = g + h + B[4]
```

```
        .global main
        .equ n,4
        .data
        g: .word 2
        h: .word 3
        b: .word 0,1,2,3,4,5,6,7,8,9
        .bss
        f: .space 4

        .text
        main:
                la t2,g
                lw s2,0(t2)       # g is s2
                la t3,h
                lw s3,0(t3)       # h is s3
                la t4,b           # t4 is the base address of B
                li t5,n           # t5 is the index (4, in this case)
                slli t5,t5,2      # index*4, since the word size is 4 bytes
                add t5,t5,t4      # @= base_addr + index *4
                lw t5,0(t5)       # t5=B[4]
                add s2,s2,s3      # g+h
                add s2,s2,t5      # g+h+B[4]
                la t1,f
                sw s2,0(t1)
        exit:
                j .
        .end
```

8) The following code increments the components of a vector with 10 elements. Translate it into RISC-V assembly code.

```
#define N 10
int V[N] = {12, 1, -2,
15, -8, 4, -31, 8, 8, 25};
for (i = 0; i < N; i++)
  V[i] = V[i] + 1;
```

```
.global main
.equ n,10
.data
v: .word 12,1,-2,15,-8,4,-31,8,8,25

.text
main :
        li s1,n          # s1 is n
        mv s2,zero       # s2 is i
for:
        beq s2,s1,exit
        la t1,v          # t1= @base of v
        slli t3,s2,2     # i*4
        add t2,t1,t3     # t2= @effective of v[i]
        lw s3,0(t2)
        addi s3,s3,1
        sw s3,0(t2)
        addi s2,s2,1     # i=i+1
        j for
exit:
        j .
.end
```

9) The following code counts the number of components greater than 0 within a vector with 6 elements. Translate it into RISC-V assembly code.

```
#define N 6
int V[N] = {14, 1, -2,
7, -8, 4};
int count = 0;
for (i = 0; i < N; i++)
{
  if (V[i] > 0)
    count = count + 1;
}
```

```
.global main
.equ n,6
.data
v: .word 14,1,-2,7,-8,4
.bss
count : .space 4

.text
main :
        la t1,v          # t1 is the base address of v
        li t2,n          # t2=n
        li t3,0          # t3 is the index i
        li s2,0          # s2 = count = 0
for:
        bge t3,t2,endfor
        slli t5,t3,2     # t5=i*4
        add t5,t5,t1     # @=i*4+ @b
        lw s1,0(t5)      # s1=v[i]
        li t6,0          # t6=0
if:
        ble s1,t6,endif
        addi s2,s2,1
endif:
        addi t3,t3,1
        j for
endfor:
        la t1,count
        sw s2,0(t1)
exit:
        j .
.end
```

10) The following code calculates the Fibonacci sequence and stores it in a vector V with size N. This infinite sequence of natural numbers is defined as: V(0) = 0, V(1) = 1, V(i+2) = V(i+1) + V(i) with i=0, 1, 2... Translate it into RISC-V assembly code.

```
#define N 12
int V[N];
V[0] = 0;
V[1] = 1;
for (i = 0; i < N-2; i++)
  V[i+2] = V[i+1] + V[i];
```

```
.global main
.equ n,12
.bss
v: .space 48
.text
main :
        la t1,v          # t1= @base of v
        mv t0,zero
        sw t0,0(t1)      # V[0] = 0
        addi t0,t0,1     # i=1
        slli t0,t0,2     # i*4
        add t0,t0,t1     # t0= @address of i=1
        addi t2,zero,1   # t2=1
        sw t2,0(t0)      # v[1]=1
        mv t0,zero       # i=0
        li t3,n
        addi t3,t3,-2    # t3=N-2, i.e. t3=10
for:
        bge t0,t3,exit
        slli t2,t0,2     # i*4
        add t4,t1,t2
        lw t5,0(t4)
        addi t4,t4,4
        lw t6,0(t4)
        add t6,t6,t5
        addi t4,t4,4
        sw t6,0(t4)
        addi t0,t0,1
        j for
exit:
        j .
.end
```

11) The following program, given a vector A with 12 elements, generates another vector B such that B only contains the elements of A that are even numbers greater than 0. For example, if A = (0,1,2,7,-8,4,5,12,11,-2,6,3), then B = (2,4,12,6). Translate it into RISC-V assembly code.

```
#define N 12
int A[N] = {0, 1, 2, 7, -8, 4, 5,
12, 11, -2, 6, 3};
int B[N];
int countB = 0;
j = 0;
for (i = 0; i < N; i++) {
  if (A[i] > 0 && A[i] is even) {
    B[j] = A[i];
       j++;
  }
}
countB = j;
```

```
.global main
.equ n,12
.data
a: .word 0,1,2,7,-8,4,5,12,11,-2,6,3
.bss
b:      .space 48
countB: .space 4
.text
main:
        mv t1,zero          # t1 is j
        mv t2,zero          # t2 is i
        la s1,a             # s1 is @base of a
        la s2,b             # s2 is @base of b
        li s3,n             # s3 is n
for:
        bge t2,s3,endfor
        slli t3,t2,2        # i*4
        add t3,t3,s1        # @effective of i
        lw t4,0(t3)
        blez t4,endif
        andi t6,t4,1        # AND operation with 1
        bnez t6,endif
        slli t3,t1,2
        add t3,t3,s2
        sw t4,0(t3)         # B[j]=A[i]
        addi t1,t1,1
endif:
        addi t2,t2,1
        j for
endfor:
        la t3,countB
        sw t1,0(t3)         # countB =j
exit:
        j .
.end
```

12) Given two vectors A and B with 10 elements each, implement another vector, C, such that: C(i) = |A(i) + B(9-i)| with i = 0, 1, ... 9. Write a RISC-V assembly program that perform this calculation.

```
#define N 10
int A[n]= {1,2,3,4,5,6,7,8,9,10};
int B[n]= {10,9,8,7,6,5,4,3,2,1};
int C[n], aux;

for (i=0; i<N; i++) {
        aux = A[i]+B[N-1-i];
/* if aux is negative, change the sign */
        if (aux < 0)
                aux = 0-aux;
        C[i] = aux;
}
```

```
.global main
.equ n,10
.data
A: .word 1,2,3,4,5,6,7,8,9,10
B: .word 10,9,8,7,6,5,4,3,2,1
.bss
C: .space 40 #2,4,6,8,10,12,14,16,18,20
.text
main:
        li t1,n             # t1=n=10
        mv t2, zero         # t2 is i
for:
        bge t2,t1, endfor
        la t3,A             # t3 = @base of A
        slli t4,t2,2        # i*4
        add t4,t4,t3        # @effective of i
        lw s1,0(t4)         # s1=A[i]
        la t3,B             # t3 = @base of B
        li t4,9             # t4=9
        sub t4,t4,t2        # t4=9-i
        slli t4,t4,2
        add t4,t4,t3
        lw s2,0(t4)
        add s1,s1,s2
        bge s1,zero,store
        sub s1,zero,s1      # negative
store:
        la t3,C
        slli t4,t2,2        # i*4
        add t4,t4,t3        # @effective of i
        sw s1,0(t4)
        addi t2,t2,1
        j for
endfor:
        j .
.end
```

13) Translate the following program into RISC-V assembly code, considering that `swap(a,b)` swaps the values of variables a and b.

```
int a = 13, b = 16;
while (a > 10) {
  a=a-1;
  b=b+2;
}
if (a < b)
  swap(a, b);
else
  b = a - 1;
```

```
.global main
.data
a: .word 13
b: .word 16

.text
main:
        la t1,a
        lw s1,0(t1)
        la t2,b
        lw s2,0(t2)
        li t3,10
while:
        ble s1,t3,if
        addi s1,s1,-1
        addi s2,s2,2
        j while
if:
        bge s1,s2,else
        sw s1,0(t2)
        sw s2,0(t1)
        j exit
else:
        addi s2,s1,-1
        sw s1,0(t1)
        sw s2,0(t2)
exit:
        j .
.end
```

14) Write a RISC-V assembly program that calls a `swap` function that swaps the content of two memory positions. This function will receive the memory addresses of variables a and b as input parameters and will preserve the content of the appropriate registers according to the RISC-V function call convention.

Question: How can we avoid saving and restoring registers inside the function?

Answer: Using only temporary registers

```
.global main
.extern _stack # The stack initial addr is in the linker file

.data
a: .word 10
b: .word 15

.text
main:
        la sp,_stack      # stack initialization
        la a0,a           # passing the parameters
        la a1,b
        jal swap
exit:
        j .
swap:
        # prologue
        addi sp,sp,-8     # no need to push ra (leaf function)
        sw s1,0(sp)
        sw s2,4(sp)
        # body
        lw s1,0(a0)
        lw s2,0(a1)
        sw s1,0(a1)
        sw s2,0(a0)
        # epilogue
        lw s1,0(sp)
        lw s2,4(sp)
        addi sp,sp,8
        jr ra              # a.k.a ret
.end
```

15) Write a RISC-V assembly program to count the number of zeros in a vector with an arbitrary length. Use a function called `count0s` that receives all the needed information as input parameters.

```
.global main
.extern _stack
.equ n ,10
.data
V: .word 1,0,2,0,3,0,4,0,5,0
.bss
numzeros: .space 4

.text
main:
        la sp,_stack      # stack initialization
        la a0,V           # passing the parameters
        li a1,n
        call count0s      # calling the function
        la t1,numzeros
        sw a0,0(t1)
exit:
        j .

count0s:
        #prologue
        addi sp,sp,-24
        sw s1,0(sp)
        sw s2,4(sp)
        sw s3,8(sp)
        sw s4,12(sp)
        sw s5,16(sp)
        sw s6,20(sp)
```

15) Write a RISC-V assembly program to count the number of zeros in a vector with an arbitrary length. Use a function called `count0s` that receives all the needed information as input parameters.

```
        # body
        mv s1,zero          # s1=i
        mv s2,a1            # s2=n
        mv s3,a0            # s3= @base of v
        mv s6,zero          # s6 = count zeros
for:
        bge s1,s2,endfor
        slli s4,s1,2
        add s4,s4,s3
        lw s5,0(s4)
        bnez s5,endif
        addi s6,s6,1
endif:
        addi s1,s1,1
        j for
endfor:
        mv a0,s6            # returning the result

        #epilogue
        lw s1,0(sp)
        lw s2,4(sp)
        lw s3,8(sp)
        lw s4,12(sp)
        lw s5,16(sp)
        lw s6,20(sp)
        addi sp,sp,24
        ret
.end
```

16) Write a RISC-V assembly program to implement a variant of the bubble sort algorithm. This variant sorts the elements of the vector according to the following code.

```
do {
  swapped = false
  for (i = 0; i <= N-2; i++){
  if (V[i] > V[i+1]){
    swap( V[i], V[i+1] )
    swapped = true
  }
} while swapped
```

```
.global main
.extern _stack
.equ n, 10
.data
V: .word 2,5,6,0,9,4,6,5,-10,-1
.text
main:
        la sp,_stack
        li s4,n             # s4=n
        addi s4,s4,-1
do:
        mv s3,zero          # s3=swapped=false
        mv s5,zero          # s5=i
for:
        bge s5,s4,endfor
        la t2,V             # t2=@base of v
        slli t3,s5,2        # i*4
        add a0,t3,t2        # @i
        lw s1,0(a0)         # V[i]
        addi a1,a0,4        # @i +1
        lw s2,0(a1)         # V[i+1]
if:
        ble s1,s2,endif
        call swap
        li s3,1             # swapped = true
endif:
        addi s5,s5,1
        j for
endfor:
        li t4,1
        beq s3,t4,do
exit:
j .
```

16) Write a RISC-V assembly program to implement a variant of the bubble sort algorithm. This variant sorts the elements of the vector according to the following code.

```
do {
  swapped = false
  for (i = 0; i <= N-2; i++){
  if (V[i] > V[i+1]){
    swap( V[i], V[i+1] )
    swapped = true
  }
} while swapped
```

```
#we will use a modified version of the swap function.
#the input parameters are the addresses of the values
#to swap

swap:
        # prologue
        addi sp,sp,-8
        sw s1,0(sp)
        sw s2,4(sp)

        # body
        lw s1,0(a0)
        lw s2,0(a1)
        sw s1,0(a1)
        sw s2,0(a0)

        # epilogue
        lw s1,0(sp)
        lw s2,4(sp)
        addi sp,sp,8
        jr ra               # ret
.end
```

17) Write a RISC-V assembly program to calculate the factorial of a non-negative number, n, using a loop that performs an iterative sequence of multiplications.

```
int fact(int n);
int n=6, rFact;

void main(){
        rFact = fact(n);
        while(1);
}

int fact(int num){
        int i,res;
        if (num > 1){
                res=num;
                for (i=num-1;i>1;i--)
                        res = res*i;
                }
        else
                res=1;
        return(res);
}
```

```
.global main
.extern _stack
.equ n,6
.bss
rFact: .space 4

.text
main:
        la sp,_stack
        li a0,n
        call fact
        la t1,rFact
        sw a0,0(t1)
fin:
        j .
```

17) Write a RISC-V assembly program to calculate the factorial of a non-negative number, n, using a loop that performs an iterative sequence of multiplications.

```
int fact(int n);
int n=6, rFact;

void main(){
        rFact = fact(n);
        while(1);
}

int fact(int num){
        int i,res;
        if (num > 1){
                res=num;
                for (i=num-1;i>1;i--)
                        res = res*i;
                }
        else
                res=1;
        return(res);
}
```

```
fact:
        # prologue
        addi sp,sp,-12    # this is a leaf function
        sw s1,0(sp)
        sw s2,4(sp)
        sw s3,8(sp)

        # body
        li s1,1           # s1=res
        mv s2,a0          # s2=i
        li s3,1
for:
        ble s2,s3,endfor
        mul s1,s1,s2
        addi s2,s2,-1
        j for
endfor:
        mv a0,s1

        # epilogue
        lw s1,0(sp)
        lw s2,4(sp)
        lw s3,8(sp)
        addi sp,sp,12
        jr ra             # ret
.end
```

18) (September 2015) Given two points P1(x1, y1) and P2(x2, y2), their Chebyshev distance can be calculated with t following algorithm:

```
int chebyshev(int x1, int y1, int x2, int y2)
{
  int d1, d2;
  d1 = abs(x1 - x2)
  d2 = abs(y1 - y2)
  if (d2 > d1)
    d1 = d2;
  return d1;
}
```

a) Write a RISC-V assembly function, `chebyshev(x1,x2,y1,y2),` which will receive the coordinates of two points P1 and P2 and will return their Chebyshev distance. This function will call another function that calculates the absolute value of a given number.

b) Translate the following program into RISC-V assembly code, which stores (into a vector D) the Chebyshev distances of a point P to each of the points within a vector V with N elements. P, V y D will be global variables. Vector V will contain 2N integers such that the i-th point will have coordinates (x, y) = (V[2*i], V[2*i + 1])

```
#define N, ...
int Px, Py; // x , y coordinates of point P
int V[2N];  //Vector with N points V=[x0,y0,x1,y1,...]
int D[N];   //Vector with N distances
void main(void)
{
 int i;
 for (i = 0; i < N; i++)
   D[i] = chebyshev(Px, Py, V[2*i], V[2*i + 1]);
}
```

```c
int chebyshev(int x1, int y1, int x2, int y2)
{
  int d1, d2;
  d1 = abs(x1 - x2)
  d2 = abs(y1 - y2)
  if (d2 > d1)
    d1 = d2;
  return d1;
}
```

```c
#define N, ...

int Px, Py; // x , y coordinates of point P
int V[2N];  //Vector with N points V=[x0,y0,x1,y1,...]
int D[N];   //Vector with N distances

void main(void)
{
 int i;
 for (i = 0; i < N; i++)
   D[i] = chebyshev(Px, Py, V[2*i], V[2*i + 1]);
}
```

```
.global main
.extern _stack
.equ n,5  #number of points (2*n components)

.data
P: .word 4,5 # x , y coordinates of point P
V: .word 1,2,-3,4,5,9,17,-15,20,12 # Vector with N
points V=[x0,y0,x1,y1,...]
.bss
D: .space n*4

.text
main:
          la sp,_stack
          mv fp,zero
          mv s1,zero          # s1 is i
          li s2,n             # s2 is N
          la s3,V             # s3 is @base of V

for:      bge s1,s2,endfor
          la s6,P
          lw a0,0(s6)
          lw a1,4(s6)
          slli s4,s1,1
          slli s4,s4,2
          add s4,s4,s3
          lw a2,0(s4)
          lw a3,4(s4)
          call chebyshev
          la s5,D              # s5 is @base of D
          slli s4,s1,2
          add s4,s4,s5
          sw a0,0(s4)
          addi s1,s1,1
          j for
endfor:   j .
```

```c
int chebyshev(int x1, int y1, int x2, int y2)
{
  int d1, d2;
  d1 = abs(x1 - x2)
  d2 = abs(y1 - y2)
  if (d2 > d1)
    d1 = d2;
  return d1;
}
```

```c
#define N, ...

int Px, Py; // x , y coordinates of point P
int V[2N];  //Vector with N points V=[x0,y0,x1,y1,...]
int D[N];   //Vector with N distances

void main(void)
{
 int i;
 for (i = 0; i < N; i++)
   D[i] = chebyshev(Px, Py, V[2*i], V[2*i + 1]);
}
```

```
chebyshev:
        # prologue
        addi sp,sp,-12
        sw s1,0(sp)
        sw s2,4(sp)
        sw ra,8(sp)

        # body
d1:
        sub s1,a0,a2 #x1 -x2
        mv a0,s1
        call abs
        mv s1,a0
d2:
        sub s2,a1,a3 #y1 -y2
        mv a0,s2
        call abs
        mv s2,a0
if:
        ble s2,s1,endcall
        mv s1,s2
endcall:
        mv a0,s1

        # epilogue
        lw s1,0(sp)
        lw s2,4(sp)
        lw ra,8(sp)
        addi sp,sp,12
        ret
abs:    bgez a0,pos
        sub a0,zero,a0
pos:    ret
.end
```

19) (June 2016) Given a vector A with 3*N elements, we want to obtain a new vector B with N elements so that each element of B is the mod-32 addition of three consecutive elements of A:

$$B[0] = (A[0]+A[1]+A[2]) \bmod 32, \quad B[1] = (A[3]+A[4]+A[5]) \bmod 32, \text{ etc}$$

a) Write a RISC-V assembly program to implement the described calculation, according to the following C code:

```
#define N 4

int A[3*N] = {a list with 3*N values};
int B[N];
int i, j = 0;

void main(void)
{
  for (i = 0; i < N; i++){
    B[i] = sum_mod_32(A, j, 3);
    j = j+3;
  }
}
```

Where `sum_mod_32(V, p, m)` returns the mod-32 addition of m consecutive elements of vector V, starting at position p.

b) Write the RISC-V assembly code of the `sum_mod_32` function, according to the following C code:

```
sum_mod_32(int A[], int j, int len)
{
  int i, sum=0;
  for (i = 0; i < len; i++)
    sum = sum + A[j+i];
  sum = mod_power_of_2(sum, 5);
  return sum;
}
```

Where `mod_power_of_2(num, exp)`, returns the value of (num mod (2^exp)), being num a positive integer and exp and integer greater than 0 and less than 32. For example, if the function is called as `mod_power_of_2(34, 5)`, it will return: (34 mod (2^5)) = (34 mod 32) = 2.

a)

```
#define N 4

int A[3*N] = {a list with 3*N values};
int B[N];
int i, j = 0;

void main(void)
{
  for (i = 0; i < N; i++){
    B[i] = sum_mod_32(A, j, 3);
    j = j+3;
  }
}
```

```
.global main
.extern _stack
.equ n,4
.data
A: .word 10,11,12,10,11,12,10,11,12,10,11,12
.bss
B: .space 16

.text
main:
        la sp,_stack
        mv fp,zero
        la s1,A          # s1 is @base of A
        la s2,B          # s2 is @base of B
        mv s3,zero       # s3 is i
        mv s4,zero       # s4 is j
        li s5,n          # s5 is n
for:
        bge s3,s5,endfor
        la a0,A          # @base A is an argument
        mv a1,s4         # j is an argument
        li a2,3          # 3 is an argument
        call sum_mod     # function call
        la s6,B          # s6 is the @base of B
        slli t0,s3,2     # t0=i*4
        add t0,t0,s6     # @effective
        sw a0,0(t0)
        addi s4,s4,3     # j=j+3
        addi s3,s3,1     # i=i+1
        j for
endfor:
        j .
```

b)

```
sum_mod_32(int A[], int j, int len)
{
  int i, sum=0;
  for (i = 0; i < len; i++)
    sum = sum + A[j+i];
  sum = mod_power_of_2(sum, 5);
  return sum;
}
```

```
sum_mod: #prologue
        addi sp,sp,-20
        sw s0,0(sp)
        sw s1,4(sp)
        sw s2,8(sp)
        sw s3,12(sp)
        sw s4,16(sp)

        # body; a0 is @base of A; a1 is j; a2 is 3
        mv s0,zero        # s0 is the index of for2
        mv s4,zero        # initialize accumulator
for2:
        bge s0,a2,endfor2 # compare if index s0 =3
        slli s1,a1,2      # a1 es j => s1 is j*4
        add s2,s1,a0      # s2 = @effective =j*4 + @base
        lw s3,0(s2)
        add s4,s4,s3      # s4 is the accumulator
        addi s0,s0,1      # increment index
        addi a1,a1,1      # j+1
        j for2
endfor2:
        andi s4,s4,31     #this does the mod32 operation
        mv a0,s4
        #epilogue
        lw s0,0(sp)
        lw s1,4(sp)
        lw s2,8(sp)
        lw s3,12(sp)
        lw s4,16(sp)
        addi sp,sp,20
        ret
.end
```

20) (September 2016) Given a vector A with N 32-bit unsigned elements, we want to calculate its Cyclic Redundancy Check (CRC) according to the Fletcher's checksum. This algorithm outputs two unsigned integers in order to check data integrity. The 64-bit version of this algorithm (Fletcher64) can be implemented with the following code:

```
void Fletcher64(
      unsigned int data[], int length, unsigned int crc[] )
{
  unsigned int sum1 = 0;
  unsigned int sum2 = 0;
  int index;
  for (index = 0; index < length; index++) {
    sum1 = sum_mod64(sum1, data[index]);
    sum2 = sum_mod64(sum1, sum2);
  }
  crc[0] = sum1;
  crc[1] = sum2;
}
```

Where the first argument is the data vector, the second one is the length vector and the third one is the vector that stores the result (i.e., the two unsigned integers that form the CRC). Assume that function `unsigned int sum_mod64(unsigned int A, unsigned int B)` is already implemented.

a) Write the RISC-V assembly code that implements the `Fletcher64` function.

b) Write a RISC-V assembly program that calls the `Fletcher64` function and calculates the CRC of the following vectors:

V={0x12340000, 0x00005678},

W={0xAB000000, 0x00CD0000, 0x0000EF00, 0x00000011}

The `Fletcher64` function must be called twice. The first call will return the CRC of V and the second one the CRC of W.

a)

```
   void Fletcher64(unsigned int
data[], int length, unsigned int
crc[])

   {
     unsigned int sum1 = 0;
     unsigned int sum2 = 0;
     int index;
     for (index = 0; index < length;
index++) {
       sum1 = sum_mod64(sum1,
data[index]);
       sum2 = sum_mod64(sum1, sum2);
     }
     crc[0] = sum1;
     crc[1] = sum2;
   }
```

```
Fletcher64:
        #prologue
        addi sp,sp,-36
        sw s0,0(sp)
        sw s1,4(sp)
        sw s2,8(sp)
        sw s3,12(sp)
        sw s4,16(sp)
        sw s5,20(sp)
        sw s6,24(sp)
        sw s7,28(sp)
        sw ra,32(sp)

        # body
        mv s1,zero        # s1 is sum1
        mv s2,zero        # s2 is sum2
        mv s0,a0          # s0 is @base of the array
        mv s3,a1          # s3 is the length
        mv s4,a2          # s4 is the @base of the result vector
        mv s5,zero        # s5 is the index for vector A
        mv s6,zero        # s6 is the @effective of A
```

a)

```
   void Fletcher64(unsigned int
data[], int length, unsigned int
crc[])

   {
     unsigned int sum1 = 0;
     unsigned int sum2 = 0;
     int index;
     for (index = 0; index < length;
index++) {
       sum1 = sum_mod64(sum1,
data[index]);
       sum2 = sum_mod64(sum1, sum2);
     }
     crc[0] = sum1;
     crc[1] = sum2;
   }
```

```
for:
        bge s5,s3,endfor
        slli s6,s5,2        # s6=s5*4
        add s6,s6,s0        # @effective
        lw s7,0(s6)         # element of A as an argument
        mv a0,s1
        mv a1,s7
        call sum_mod64      # function call
        mv s1,a0
        mv a1,s2
        call sum_mod64
        mv s2,a0
        addi s5,s5,1        # i=i+1
        j for
endfor: mv a0,s1
        mv a1,s2
        sw a0,0(s4)
        sw a1,4(s4)

        #epilogue
        lw s0,0(sp)
        lw s1,4(sp)
        lw s2,8(sp)
        lw s3,12(sp)
        lw s4,16(sp)
        lw s5,20(sp)
        lw s6,24(sp)
        lw s7,28(sp)
        lw ra,32(sp)
        addi sp,sp,36
        ret
.end
```

b)

```
   void Fletcher64(unsigned int
data[], int length, unsigned int
crc[])

   {
     unsigned int sum1 = 0;
     unsigned int sum2 = 0;
     int index;
     for (index = 0; index < length;
index++) {
       sum1 = sum_mod64(sum1,
data[index]);
       sum2 = sum_mod64(sum1, sum2);
     }
     crc[0] = sum1;
     crc[1] = sum2;
   }
```

```
.global main
.extern _stack
.equ Nv,64
.equ Nw,128

.data
V: .word 0x12340000, 0x00005678
W: .word 0xAB000000, 0x00CD0000, 0x0000EF00, 0x00000011

.bss
crc_v: .space 2*4
crc_w: .space 2*4

.text
main:
        la sp,_stack
        mv fp,zero
        la a0,V          # a0 = 1st argument
        li a1,Nv         # a1 = 2nd argument
        la a2,crc_v      # a2 = 3rd argument
        call Fletcher64

        la a0,W          # a0 = 1st argument
        li a1,Nw         # a1 = 2nd argument
        la a2,crc_w      # a2 = 3rd argument
        call Fletcher64

        j .
```

21) (June 2013). Assume that we define a number as "nice" if its value is less than one hundred thousand and it can be obtained as the sum of natural numbers as 1+2+3+4+5+…

a) Write a RISC-V assembly program that receives a natural number N and decides whether it is nice or not. The program will write a 1 in variable B if N is nice and a 0 otherwise.

b) Convert the previous program in a function that receives a natural number N and returns a 1 if N is nice and a 0 otherwise. Write a RISC-V assembly program that calls this function. This program will receive a vector A with M natural numbers and will find out how many nice numbers there are in such vector. The program will store the amount of nice numbers in a `count_nice` variable.

a) Write a RISC-V assembly program that receives a natural number N and decides whether it is nice or not. The program will write a 1 in variable B if N is nice and a 0 otherwise.

```
.global main
.equ N,21
.bss
B: .space 4

.text
main :
        mv s0,zero        # nice = false
        li s1,N
        li s2,100000
        mv s3,zero        # s3 is the accumulator
        mv s4,zero        # s4 is the generator of natural num.
do:
        add s3,s3,s4
        addi s4,s4,1
        bge s3,s2,exit    # accumulator >=100000
        bgt s3,s1,exit    # accumulator >N
        beq s3,s1,nicetrue
        j do

nicetrue:
        addi s0,zero,1 # nice = true

exit:   la t2,B
        sw s0,0(t2)
        j .
.end
```

b) Convert the previous program in a function that receives a natural number N and returns a 1 if N is nice and a 0 otherwise. Write a RISC-V assembly program that calls this function. This program will receive a vector A with M natural numbers and will find out how many nice numbers there are in such vector. The program will store the amount of nice numbers in a `count_nice` variable.

```
.global main
.extern _stack
.equ N,5
.data
A: .word 3,5,6,15,13          # sample array
M: .word 5                    # sample array length
.bss
count_nice: .space 4
.text
main:
        la sp,_stack
        la s1,M
        lw s1,0(s1)           # s1=M
        la s2,A               # s2= base@ of A
        mv s3,zero            # s3 is index i of vector A
        mv s4,zero            # s4 is the count of nice numbers
        li s5,1               # s5 stores a "1"
        mv s6,zero            # s6 is the @effective of A
for:
        bge s3,s1,endfor
        slli s6,s3,2          # s6=s3 *4
        add s6,s6,s2          # @effective
        lw a0,0(s6)           # element of A is an argument
        call nice             # function call
        addi s3,s3,1          # i=i+1
        beq a0,s5,exit
        j for
exit:
        addi s4,s4,1          # nice count increased
        j for
endfor:
        la s1,count_nice
        sw s4,0(s1)
        j .
```

b) Convert the previous program in a function that receives a natural number N and returns a 1 if N is nice and a 0 otherwise. Write a RISC-V assembly program that calls this function. This program will receive a vector A with M natural numbers and will find out how many nice numbers there are in such vector. The program will store the amount of nice numbers in a `count_nice` variable.

```
nice:    #prologue
         addi sp,sp,-20
         sw s0,0(sp)
         sw s1,4(sp)
         sw s2,8(sp)
         sw s3,12(sp)
         sw s4,16(sp)

         # body
         mv s0,zero          # nice = false
         li s2,100000
         mv s3,zero          # s3 is the accumulator
         mv s4,zero          # s4 is the generator of natural num.
do:
         add s3,s3,s4
         addi s4,s4,1
         bge s3,s2,exit2     # accumulator >=100000
         bgt s3,a0,exit2     # accumulator >N
         beq s3,a0,nicetrue
         j do

nicetrue:
         addi s0,zero,1 # nice = true

exit2:   mv a0,s0
         #epilogue
         lw s0,0(sp)
         lw s1,4(sp)
         lw s2,8(sp)
         lw s3,12(sp)
         lw s4,16(sp)
         addi sp,sp,20
         ret
.end
```

22) (June 2014) Given a vector V with N elements, we define it as Melchioriform if it has at least one Blonde element. An element V[i] is Blonde if it meets the following expression:

$$\sum_{j=0}^{N-1} v[j] = 2 * v[i]$$

a) Write a RISC-V assembly function, `AddVector(V, N)`, which adds the N elements of vector V. This function must follow the RISC-V function call convention.

b) Write a RISC-V assembly program that, given a vector V and its size N, decides whether V is Melchioriform or not, using the `AddVector` function.

a)

```
addvector:
        #prologue
        addi sp,sp,-24
        sw s0,0(sp)
        sw s1,4(sp)
        sw s2,8(sp)
        sw s3,12(sp)
        sw s4,16(sp)
        sw s5,20(sp)
        # body
        mv s0,a0            #s0 is the @base of V
        mv s1,a1            #s1 is N
        mv s2,zero          #s2 is i
        mv s5,zero          #s5 is the accumulator
for:
        bge s2,s1,endfor
        slli s3,s2,2
        add s3,s3,s0
        lw s4,0(s3)
        add s5,s5,s4
        addi s2,s2,1
        j for
endfor:
        mv a0,s5
        #epilogue
        lw s0,0(sp)
        lw s1,4(sp)
        lw s2,8(sp)
        lw s3,12(sp)
        lw s4,16(sp)
        lw s5,20(sp)
        addi sp,sp,24
        ret
.end
```

22) (June 2014) Given a vector V with N elements, we define it as Melchioriform if it has at least one Blonde element. An element V[i] is Blonde if it meets the following expression:

$$\sum_{j=0}^{N-1} v[j] = 2 * v[i]$$

a) Write a RISC-V assembly function, `AddVector(V, N)`, which adds the N elements of vector V. This function must follow the RISC-V function call convention.

b) Write a RISC-V assembly program that, given a vector V and its size N, decides whether V is Melchioriform or not, using the `AddVector` function.

b)

```
.global main
.extern _stack
.equ n,6
.data
V: .word 2,1,10,2,2,3
.bss
melch : .zero 4  # result is 0 by default
.text
main :
        la sp,_stack
        mv fp,zero
        la a0,V
        li a1,n
        call addvector
        mv s1,a0          #s1 is the vector sum
        la t1,V           #t1 is the @base of V
        mv t2,zero        #t2 is i
        li t3,n           #t3 is n
do:
        slli t3,t2,2
        add t3,t3,t1
        lw s2,0(t3)
        slli s2,s2,1
        beq s2,s1,melctrue
        addi t2,t2,1 #i++
        bge t2,t3,exit
        j do
melctrue:
        addi s1,zero,1     # result is 1
        la t1,melch
        sw s1,0(t1)
exit:
        j .
```

23) (September 2014) A vector V with N natural elements is Noelian if it is a monotonic increasing sequence and its elements add up to 45. For example: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} is Noelian because 0≤1≤2≤3≤4≤5≤6≤7≤8≤9 and 1+2+3+4+5+6+7+8+9=45. Also {3, 5, 5, 7, 10, 15} is Noelian, because 3≤5≤5≤7≤10≤15 and 3+5+5+7+10+15=45.

a) Write a RISC-V assembly function, `Sum45(A, N)`, which receives the initial address of a vector A as its first parameter, the number of elements N as its second parameter and returns 1 if its sum is 45 (0 otherwise). This function must be programmed following the RISC-V function call convention.

b) Write a RISC-V assembly program that, using the previous function, determines whether an input vector is Noelian or not.

a)

```
sum45:
        #prologue
        addi sp,sp,-20
        sw s0,0(sp)
        sw s1,4(sp)
        sw s3,8(sp)
        sw s4,12(sp)
        sw s5,16(sp)

        # body
        mv s0,a0        # s0 is the @address of v
        mv s1,a1        # s1 is n
        mv s3,zero      # s3 is i
        mv s5,zero      # s5 is the accumulator
        # vector sum
for:
        bge s3,s1,endfor
        slli s4,s3,2
        add s4,s4,s0
        lw s4,0(s4)
        add s5,s5,s4
        addi s3,s3,1
        j for
```

23) (September 2014) A vector V with N natural elements is Noelian if it is a monotonic increasing sequence and its elements add up to 45. For example: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} is Noelian because 0≤1≤2≤3≤4≤5≤6≤7≤8≤9 and 1+2+3+4+5+6+7+8+9=45. Also {3, 5, 5, 7, 10, 15} is Noelian, because 3≤5≤5≤7≤10≤15 and 3+5+5+7+10+15=45.

a) Write a RISC-V assembly function, `Sum45(A, N)`, which receives the initial address of a vector A as its first parameter, the number of elements N as its second parameter and returns 1 if its sum is 45 (0 otherwise). This function must be programmed following the RISC-V function call convention.

b) Write a RISC-V assembly program that, using the previous function, determines whether an input vector is Noelian or not.

a)

```
endfor:
        addi s3,zero,45
        bne s5,s3,else
        addi s4,zero,1
        mv a0,s4            # a0=1 if sum=45
        j exit2
else:
        mv a0,zero

        #epilogue
exit2:
        lw s0,0(sp)
        lw s1,4(sp)
        lw s3,8(sp)
        lw s4,12(sp)
        lw s5,16(sp)
        addi sp,sp,20
        ret
.end
```

23) (September 2014) A vector V with N natural elements is Noelian if it is a monotonic increasing sequence and its elements add up to 45. For example: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} is Noelian because 0≤1≤2≤3≤4≤5≤6≤7≤8≤9 and 1+2+3+4+5+6+7+8+9=45. Also {3, 5, 5, 7, 10, 15} is Noelian, because 3≤5≤5≤7≤10≤15 and 3+5+5+7+10+15=45.

a) Write a RISC-V assembly function, `Sum45(A, N)`, which receives the initial address of a vector A as its first parameter, the number of elements N as its second parameter and returns 1 if its sum is 45 (0 otherwise). This function must be programmed following the RISC-V function call convention.

b) Write a RISC-V assembly program that, using the previous function, determines whether an input vector is Noelian or not.

b)

```
.global main
.extern _stack
.equ n,6
.data
V: .word 3,5,5,7,10,15
.bss
noelian: .space 4
.text
main:
        la sp,_stack
        la a0,V
        li a1,n
        call sum45
        mv s0,a0            # s0 is the sum45 result
        addi s3,zero,1      # s3 is 1
        bne s0,s3,false     # sum is not 45
        la t0,V             # t0 is the @base of V
        li t3,n-1           # t3=n-1 (n-1 comparisons)
        mv t1,zero          # t1=i=0
do:
        slli t2,t1,2        # i *4
        add t2,t2,t0        # t2 = @i
        lw s4,0(t2)
        lw s5,4(t2)
        bgt s4,s5,false
        addi t1,t1,1        # i++
        blt t1,t3,do        # i<n
        addi s3,zero,1      # noelian = true
        la t4,noelian
        sw s3,0(t4)
        j exit
false:
        la t4,noelian
        sw zero,0(t4)
exit:
        j .
```

About *Creative Commons*

- CC license (Creative Commons)

  - This license enables reusers to distribute, remix, adapt, and build upon the material in any medium or format for noncommercial purposes only, and only so long as attribution is given to the creator. If you remix, adapt, or build upon the material, you must license the modified material under identical terms:

Attribution:
Credit must be given to the creator.

Non commercial:
Only noncommercial uses of the work are permitted.

Share alike:
Adaptations must be shared under the same terms.

More information: https://creativecommons.org/licenses/by-nc-sa/4.0/