CompSys 2023

Machine arc

Assembler programming

Johan Topp

4 Machine architecture (about 24 %)

4.1 Assembler programming (about 14 %)

Consider the following program written in RISC-V assembler.

```
<start>:
    li   a5,1
    bge  a5,a0,L4
    addi a6,a1,4
    slli a0,a0,0x2
    add  t1,a1,a0
    li   a7,0
    li   a0,-1
    j    L2
L1:
    addi a4,a4,1
    slli a4,a4,0x2
    add  a4,a1,a4
    sw   a2,0(a4)
    addi a7,a7,1
    addi a6,a6,4
    beq  a6,t1,L4
L2:
    lw   a2,0(a6)
    mv   a4,a7
    mv   a5,a6
    bltz a7,L1
L3:
    lw   a3,-4(a5)
    bge  a2,a3,L1
    sw   a3,0(a5)
    addi a4,a4,-1
    addi a5,a5,-4
    bne  a4,a0,L3
    j    L1
L4:
    ret
```

Question 4.1.1: Identify the control structures of the program (e.g. loops, conditionals, and function calls).

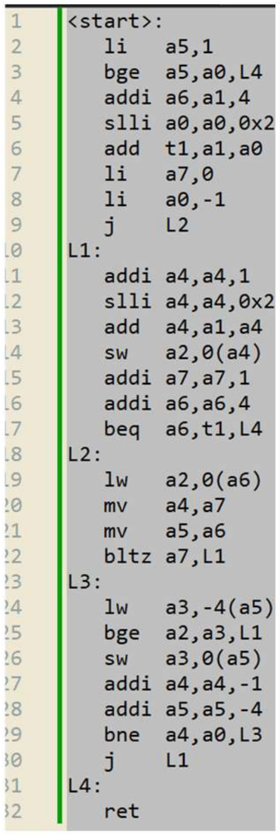
Question 4.1.2: Specify which registers are used for pointers. How can you see this? What are the operations?

Question 4.1.3: Which registers contains functions arguments and what are their type? Describe how you identified this.

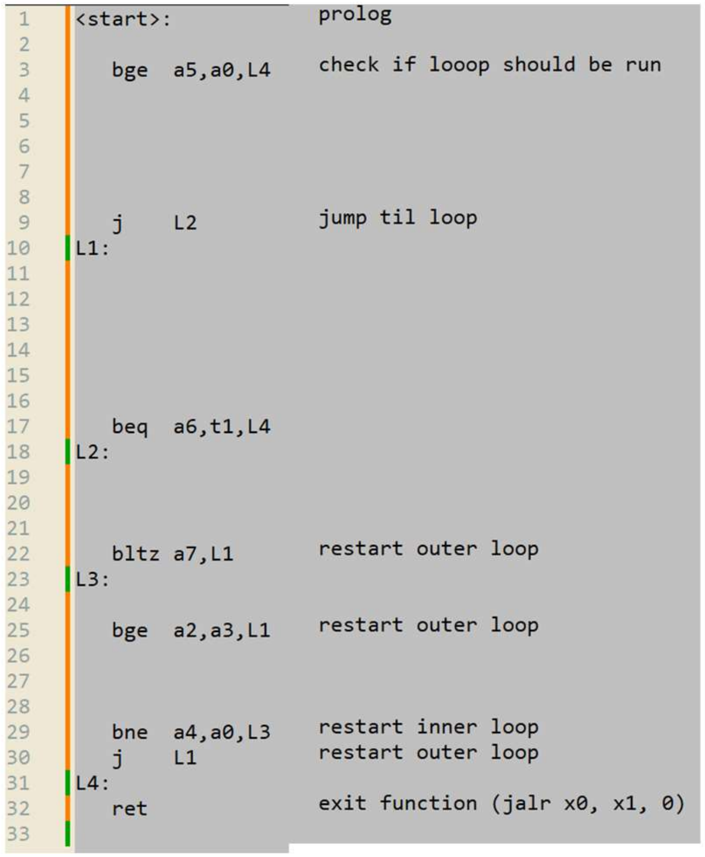
Question 4.1.4: Rewrite the above X86prime-assembler program to a C program. The resulting program must not have a goto-style and minor syntactical mistakes are acceptable.

Question 4.1.5: Descripe shortly the functionality of the program.

Question 4.1.1: Identify the control structures of the program (e.g. loops, conditionals, and function calls).

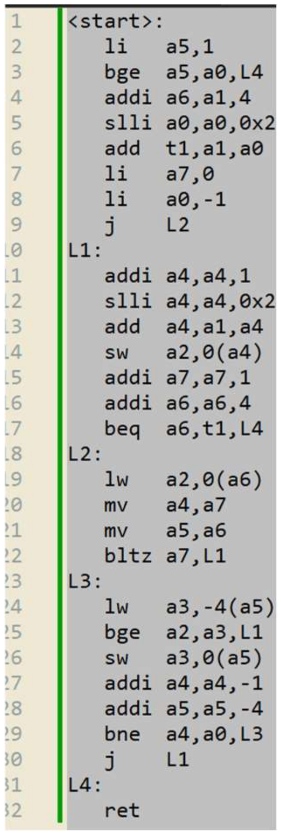


```
<start>:
   li   a5,1
   bge  a5,a0,L4
   addi a6,a1,4
   slli a0,a0,0x2
   add  t1,a1,a0
   li   a7,0
   li   a0,-1
   j    L2
L1:
   addi a4,a4,1
   slli a4,a4,0x2
   add  a4,a1,a4
   sw   a2,0(a4)
   addi a7,a7,1
   addi a6,a6,4
   beq  a6,t1,L4
L2:
   lw   a2,0(a6)
   mv   a4,a7
   mv   a5,a6
   bltz a7,L1
L3:
   lw   a3,-4(a5)
   bge  a2,a3,L1
   sw   a3,0(a5)
   addi a4,a4,-1
   addi a5,a5,-4
   bne  a4,a0,L3
   j    L1
L4:
   ret
```

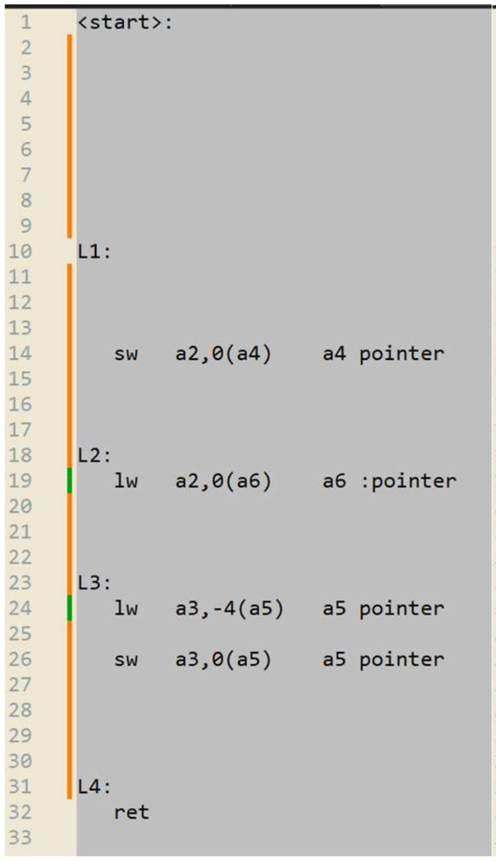


```
<start>:                prolog

   bge  a5,a0,L4        check if looop should be run





   j    L2              jump til loop
L1:






   beq  a6,t1,L4
L2:



   bltz a7,L1           restart outer loop
L3:

   bge  a2,a3,L1        restart outer loop



   bne  a4,a0,L3        restart inner loop
   j    L1              restart outer loop
L4:
   ret                  exit function (jalr x0, x1, 0)

```

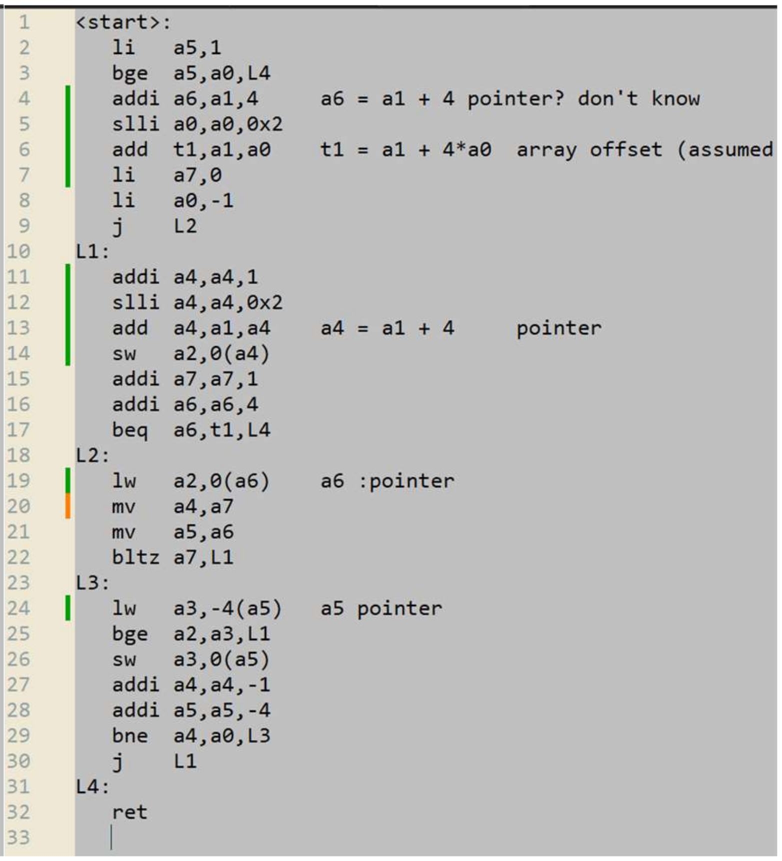
Question 4.1.2: Specify which registers are used for pointers. How can you see this? What are the operations?



```
<start>:
   li   a5,1
   bge  a5,a0,L4
   addi a6,a1,4
   slli a0,a0,0x2
   add  t1,a1,a0
   li   a7,0
   li   a0,-1
   j    L2
L1:
   addi a4,a4,1
   slli a4,a4,0x2
   add  a4,a1,a4
   sw   a2,0(a4)
   addi a7,a7,1
   addi a6,a6,4
   beq  a6,t1,L4
L2:
   lw   a2,0(a6)
   mv   a4,a7
   mv   a5,a6
   bltz a7,L1
L3:
   lw   a3,-4(a5)
   bge  a2,a3,L1
   sw   a3,0(a5)
   addi a4,a4,-1
   addi a5,a5,-4
   bne  a4,a0,L3
   j    L1
L4:
   ret
```

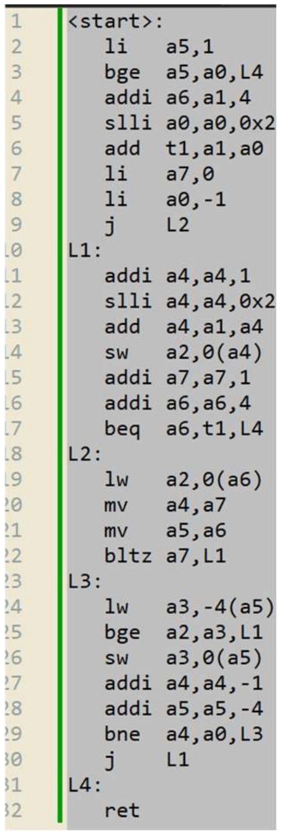


```
<start>:








L1:



   sw    a2,0(a4)     a4 pointer




L2:
   lw    a2,0(a6)     a6 :pointer



L3:
   lw    a3,-4(a5)    a5 pointer

   sw    a3,0(a5)     a5 pointer




L4:
   ret

```

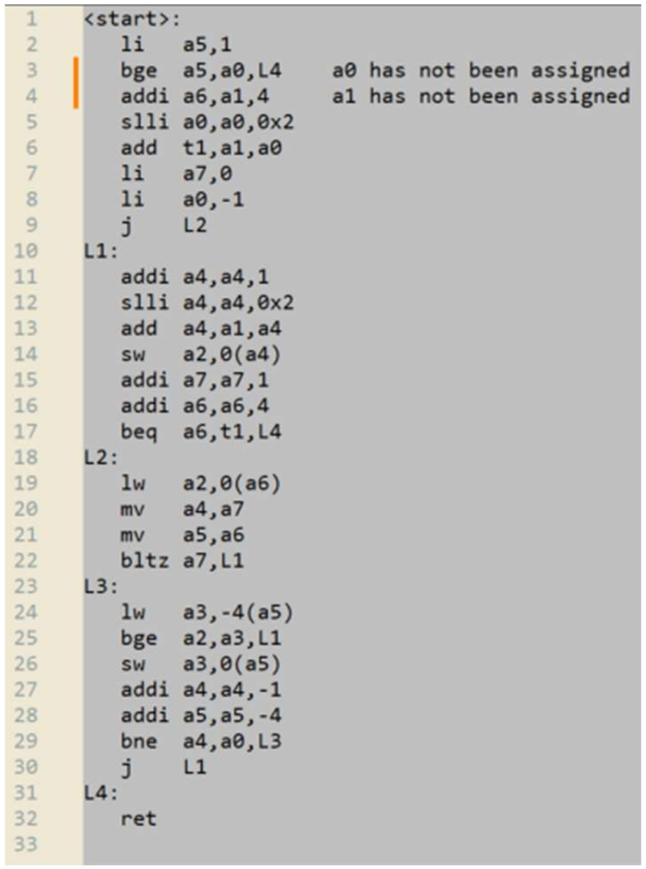


```
<start>:
   li   a5,1
   bge  a5,a0,L4
   addi a6,a1,4       a6 = a1 + 4 pointer? don't know
   slli a0,a0,0x2
   add  t1,a1,a0      t1 = a1 + 4*a0   array offset (assumed
   li   a7,0
   li   a0,-1
   j    L2
L1:
   addi a4,a4,1
   slli a4,a4,0x2
   add  a4,a1,a4      a4 = a1 + 4      pointer
   sw   a2,0(a4)
   addi a7,a7,1
   addi a6,a6,4
   beq  a6,t1,L4
L2:
   lw   a2,0(a6)      a6 :pointer
   mv   a4,a7
   mv   a5,a6
   bltz a7,L1
L3:
   lw   a3,-4(a5)     a5 pointer
   bge  a2,a3,L1
   sw   a3,0(a5)
   addi a4,a4,-1
   addi a5,a5,-4
   bne  a4,a0,L3
   j    L1
L4:
   ret

```

Question 4.1.3: Which registers contains functions arguments and what are their type? Describe how you identified this.



```
<start>:
   li   a5,1
   bge  a5,a0,L4
   addi a6,a1,4
   slli a0,a0,0x2
   add  t1,a1,a0
   li   a7,0
   li   a0,-1
   j    L2
L1:
   addi a4,a4,1
   slli a4,a4,0x2
   add  a4,a1,a4
   sw   a2,0(a4)
   addi a7,a7,1
   addi a6,a6,4
   beq  a6,t1,L4
L2:
   lw   a2,0(a6)
   mv   a4,a7
   mv   a5,a6
   bltz a7,L1
L3:
   lw   a3,-4(a5)
   bge  a2,a3,L1
   sw   a3,0(a5)
   addi a4,a4,-1
   addi a5,a5,-4
   bne  a4,a0,L3
   j    L1
L4:
   ret
```

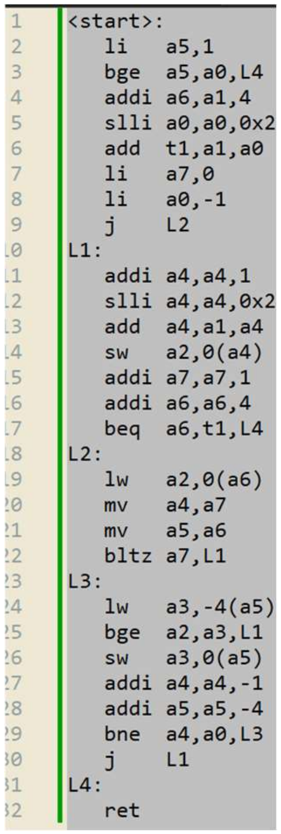


```
<start>:
   li   a5,1
   bge  a5,a0,L4       a0 has not been assigned
   addi a6,a1,4        a1 has not been assigned
   slli a0,a0,0x2
   add  t1,a1,a0
   li   a7,0
   li   a0,-1
   j    L2
L1:
   addi a4,a4,1
   slli a4,a4,0x2
   add  a4,a1,a4
   sw   a2,0(a4)
   addi a7,a7,1
   addi a6,a6,4
   beq  a6,t1,L4
L2:
   lw   a2,0(a6)
   mv   a4,a7
   mv   a5,a6
   bltz a7,L1
L3:
   lw   a3,-4(a5)
   bge  a2,a3,L1
   sw   a3,0(a5)
   addi a4,a4,-1
   addi a5,a5,-4
   bne  a4,a0,L3
   j    L1
L4:
   ret
```

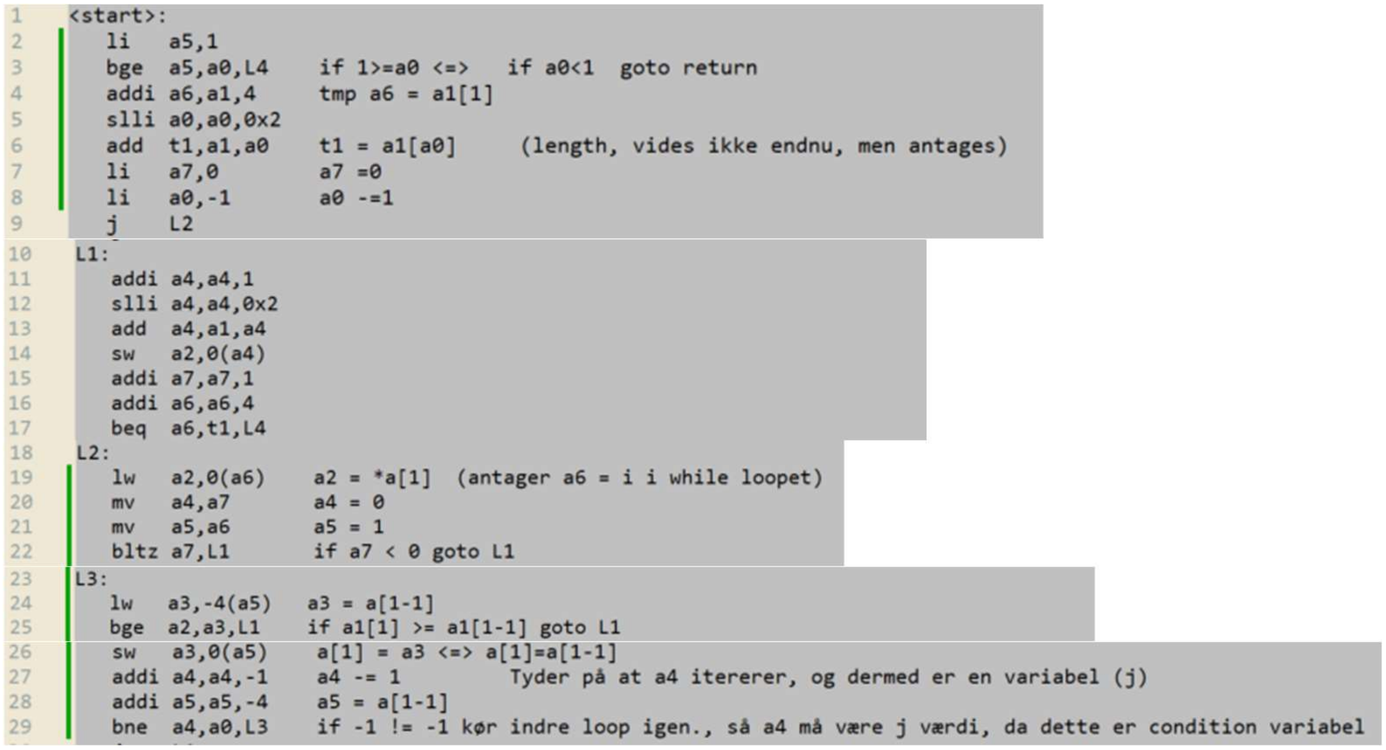
Start fra <start> og se hvad der ikke er assigned, før det bliver brugt.

Note:
Husk at følge jumps/branches, ikke bare se oppe fra og ned. L2 kunne benytte et argument der ikke var assigned, men som efterfølgende blev assigned i L1

Question 4.1.4: Rewrite the above X86prime-assembler program to a C program. The resulting program must not have a goto-style and minor syntactical mistakes are acceptable.



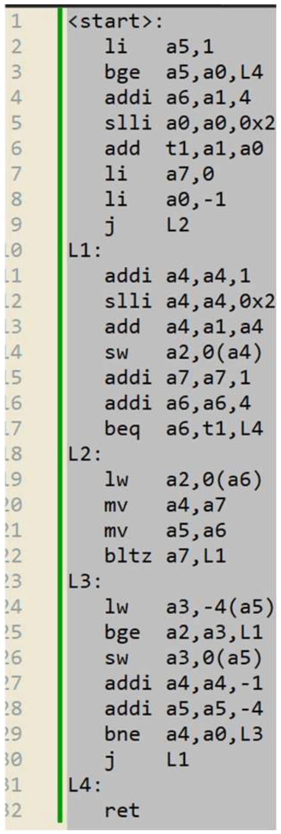
```
1   <start>:
2      li   a5,1
3      bge  a5,a0,L4
4      addi a6,a1,4
5      slli a0,a0,0x2
6      add  t1,a1,a0
7      li   a7,0
8      li   a0,-1
9      j    L2
10  L1:
11     addi a4,a4,1
12     slli a4,a4,0x2
13     add  a4,a1,a4
14     sw   a2,0(a4)
15     addi a7,a7,1
16     addi a6,a6,4
17     beq  a6,t1,L4
18  L2:
19     lw   a2,0(a6)
20     mv   a4,a7
21     mv   a5,a6
22     bltz a7,L1
23  L3:
24     lw   a3,-4(a5)
25     bge  a2,a3,L1
26     sw   a3,0(a5)
27     addi a4,a4,-1
28     addi a5,a5,-4
29     bne  a4,a0,L3
30     j    L1
31  L4:
32     ret
```



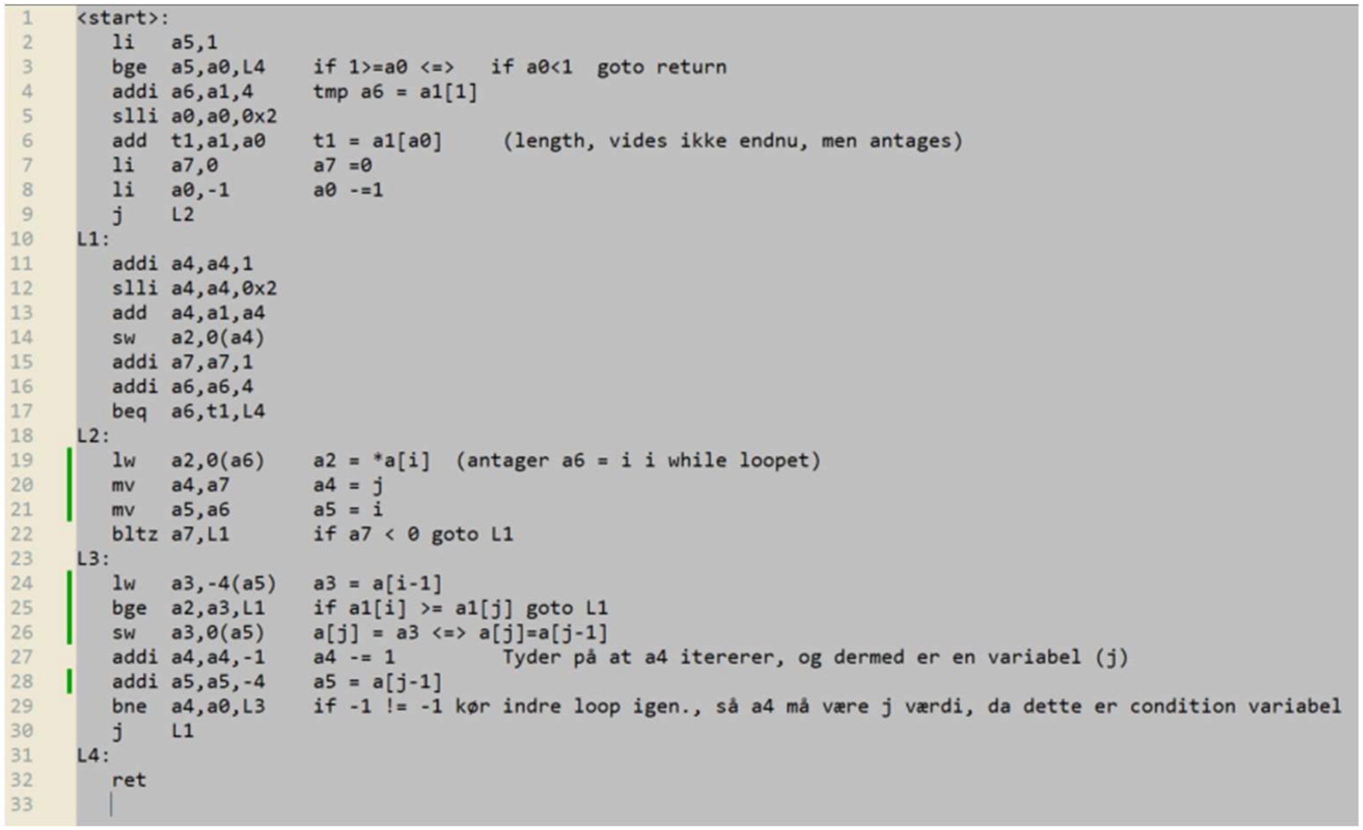
```
1   <start>:
2      li   a5,1
3      bge  a5,a0,L4     if 1>=a0 <=>   if a0<1  goto return
4      addi a6,a1,4      tmp a6 = a1[1]
5      slli a0,a0,0x2
6      add  t1,a1,a0     t1 = a1[a0]     (length, vides ikke endnu, men antages)
7      li   a7,0         a7 =0
8      li   a0,-1        a0 -=1
9      j    L2
10  L1:
11     addi a4,a4,1
12     slli a4,a4,0x2
13     add  a4,a1,a4
14     sw   a2,0(a4)
15     addi a7,a7,1
16     addi a6,a6,4
17     beq  a6,t1,L4
18  L2:
19     lw   a2,0(a6)     a2 = *a[1]  (antager a6 = i i while loopet)
20     mv   a4,a7        a4 = 0
21     mv   a5,a6        a5 = 1
22     bltz a7,L1        if a7 < 0 goto L1
23  L3:
24     lw   a3,-4(a5)   a3 = a[1-1]
25     bge  a2,a3,L1    if a1[1] >= a1[1-1] goto L1
26     sw   a3,0(a5)     a[1] = a3 <=> a[1]=a[1-1]
27     addi a4,a4,-1     a4 -= 1         Tyder på at a4 itererer, og dermed er en variabel (j)
28     addi a5,a5,-4     a5 = a[1-1]
29     bne  a4,a0,L3     if -1 != -1 kør indre loop igen., så a4 må være j værdi, da dette er condition variabel
```

Lav konklusioner. Bne I linje 29 er indre loop af j. Og (j--(ud fra linje 27))

Question 4.1.4: Rewrite the above X86prime-assembler program to a C program. The resulting program must not have a goto-style and minor syntactical mistakes are acceptable.

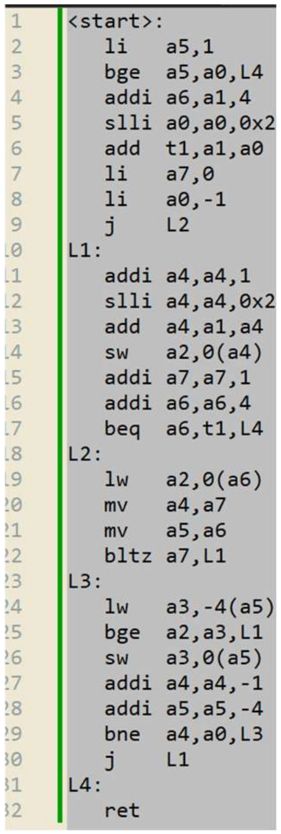


```
<start>:
   li   a5,1
   bge  a5,a0,L4
   addi a6,a1,4
   slli a0,a0,0x2
   add  t1,a1,a0
   li   a7,0
   li   a0,-1
   j    L2
L1:
   addi a4,a4,1
   slli a4,a4,0x2
   add  a4,a1,a4
   sw   a2,0(a4)
   addi a7,a7,1
   addi a6,a6,4
   beq  a6,t1,L4
L2:
   lw   a2,0(a6)
   mv   a4,a7
   mv   a5,a6
   bltz a7,L1
L3:
   lw   a3,-4(a5)
   bge  a2,a3,L1
   sw   a3,0(a5)
   addi a4,a4,-1
   addi a5,a5,-4
   bne  a4,a0,L3
   j    L1
L4:
   ret
```

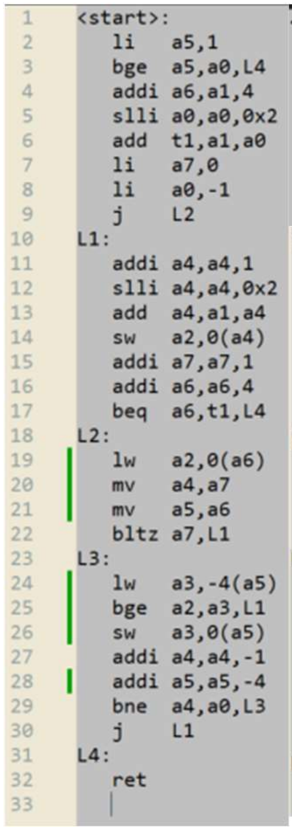


```
<start>:
   li   a5,1
   bge  a5,a0,L4     if 1>=a0 <=>   if a0<1  goto return
   addi a6,a1,4      tmp a6 = a1[1]
   slli a0,a0,0x2
   add  t1,a1,a0     t1 = a1[a0]     (length, vides ikke endnu, men antages)
   li   a7,0         a7 =0
   li   a0,-1        a0 -=1
   j    L2
L1:
   addi a4,a4,1
   slli a4,a4,0x2
   add  a4,a1,a4
   sw   a2,0(a4)
   addi a7,a7,1
   addi a6,a6,4
   beq  a6,t1,L4
L2:
   lw   a2,0(a6)     a2 = *a[i]  (antager a6 = i i while loopet)
   mv   a4,a7        a4 = j
   mv   a5,a6        a5 = i
   bltz a7,L1        if a7 < 0 goto L1
L3:
   lw   a3,-4(a5)    a3 = a[i-1]
   bge  a2,a3,L1     if a1[i] >= a1[j] goto L1
   sw   a3,0(a5)     a[j] = a3 <=> a[j]=a[j-1]
   addi a4,a4,-1     a4 -= 1         Tyder på at a4 itererer, og dermed er en variabel (j)
   addi a5,a5,-4     a5 = a[j-1]
   bne  a4,a0,L3     if -1 != -1 kør indre loop igen., så a4 må være j værdi, da dette er condition variabel
   j    L1
L4:
   ret
```

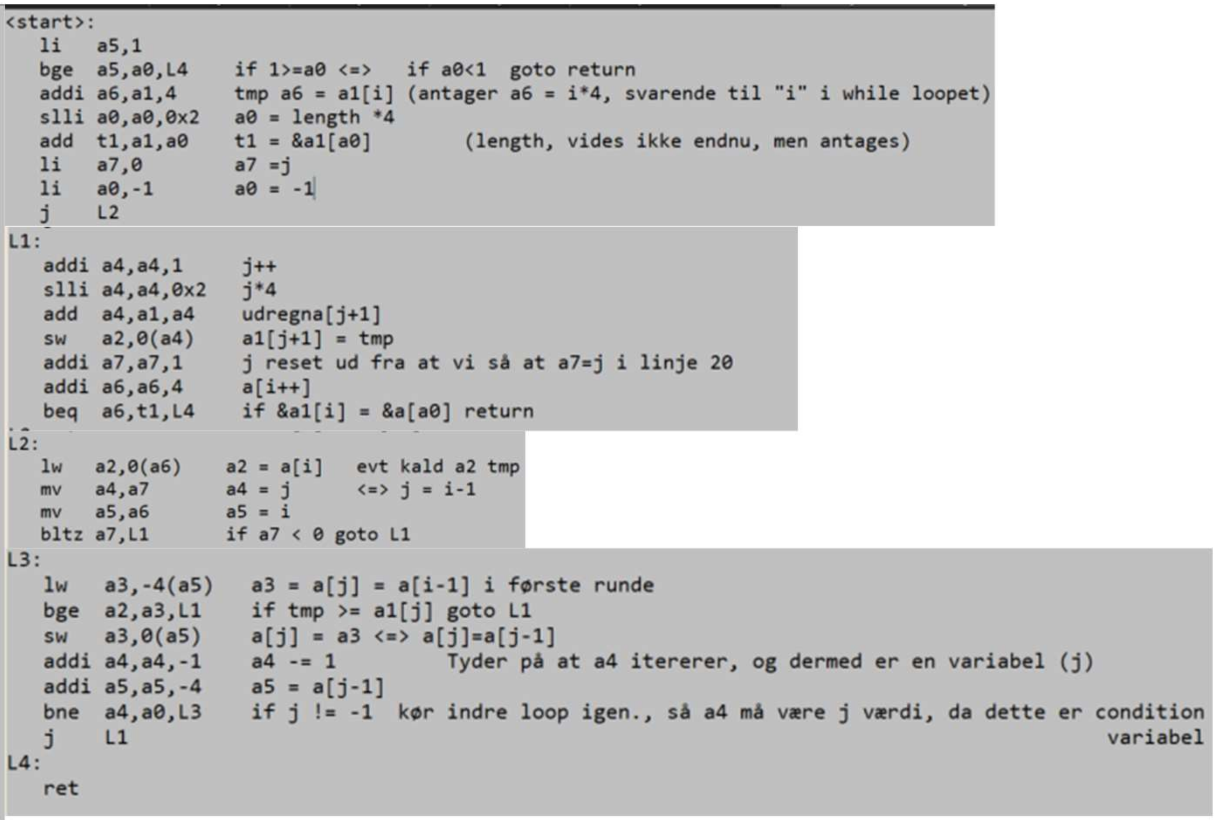
Question 4.1.4: Rewrite the above X86prime-assembler program to a C program. The resulting program must not have a goto-style and minor syntactical mistakes are acceptable.



```
<start>:
   li   a5,1
   bge  a5,a0,L4
   addi a6,a1,4
   slli a0,a0,0x2
   add  t1,a1,a0
   li   a7,0
   li   a0,-1
   j    L2
L1:
   addi a4,a4,1
   slli a4,a4,0x2
   add  a4,a1,a4
   sw   a2,0(a4)
   addi a7,a7,1
   addi a6,a6,4
   beq  a6,t1,L4
L2:
   lw   a2,0(a6)
   mv   a4,a7
   mv   a5,a6
   bltz a7,L1
L3:
   lw   a3,-4(a5)
   bge  a2,a3,L1
   sw   a3,0(a5)
   addi a4,a4,-1
   addi a5,a5,-4
   bne  a4,a0,L3
   j    L1
L4:
   ret
```

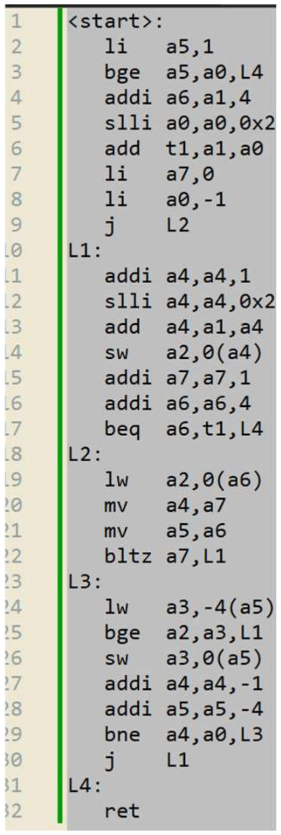


```
1   <start>:
2      li   a5,1
3      bge  a5,a0,L4
4      addi a6,a1,4
5      slli a0,a0,0x2
6      add  t1,a1,a0
7      li   a7,0
8      li   a0,-1
9      j    L2
10  L1:
11     addi a4,a4,1
12     slli a4,a4,0x2
13     add  a4,a1,a4
14     sw   a2,0(a4)
15     addi a7,a7,1
16     addi a6,a6,4
17     beq  a6,t1,L4
18  L2:
19     lw   a2,0(a6)
20     mv   a4,a7
21     mv   a5,a6
22     bltz a7,L1
23  L3:
24     lw   a3,-4(a5)
25     bge  a2,a3,L1
26     sw   a3,0(a5)
27     addi a4,a4,-1
28     addi a5,a5,-4
29     bne  a4,a0,L3
30     j    L1
31  L4:
32     ret
33
```



```
<start>:
   li   a5,1
   bge  a5,a0,L4    if 1>=a0 <=>   if a0<1  goto return
   addi a6,a1,4     tmp a6 = a1[i] (antager a6 = i*4, svarende til "i" i while loopet)
   slli a0,a0,0x2   a0 = length *4
   add  t1,a1,a0    t1 = &a1[a0]        (length, vides ikke endnu, men antages)
   li   a7,0        a7 =j
   li   a0,-1       a0 = -1
   j    L2
L1:
   addi a4,a4,1     j++
   slli a4,a4,0x2   j*4
   add  a4,a1,a4    udregna[j+1]
   sw   a2,0(a4)    a1[j+1] = tmp
   addi a7,a7,1     j reset ud fra at vi så at a7=j i linje 20
   addi a6,a6,4     a[i++]
   beq  a6,t1,L4    if &a1[i] = &a[a0] return
L2:
   lw   a2,0(a6)    a2 = a[i]   evt kald a2 tmp
   mv   a4,a7       a4 = j      <=> j = i-1
   mv   a5,a6       a5 = i
   bltz a7,L1       if a7 < 0 goto L1
L3:
   lw   a3,-4(a5)   a3 = a[j] = a[i-1] i første runde
   bge  a2,a3,L1    if tmp >= a1[j] goto L1
   sw   a3,0(a5)    a[j] = a3 <=> a[j]=a[j-1]
   addi a4,a4,-1    a4 -= 1          Tyder på at a4 itererer, og dermed er en variabel (j)
   addi a5,a5,-4    a5 = a[j-1]
   bne  a4,a0,L3    if j != -1  kør indre loop igen., så a4 må være j værdi, da dette er condition
   j    L1                                                                              variabel
L4:
   ret
```

Question 4.1.4: Rewrite the above X86prime-assembler program to a C program. The resulting program must not have a goto-style and minor syntactical mistakes are acceptable.



```
<start>:
   li   a5,1
   bge  a5,a0,L4
   addi a6,a1,4
   slli a0,a0,0x2
   add  t1,a1,a0
   li   a7,0
   li   a0,-1
   j    L2
L1:
   addi a4,a4,1
   slli a4,a4,0x2
   add  a4,a1,a4
   sw   a2,0(a4)
   addi a7,a7,1
   addi a6,a6,4
   beq  a6,t1,L4
L2:
   lw   a2,0(a6)
   mv   a4,a7
   mv   a5,a6
   bltz a7,L1
L3:
   lw   a3,-4(a5)
   bge  a2,a3,L1
   sw   a3,0(a5)
   addi a4,a4,-1
   addi a5,a5,-4
   bne  a4,a0,L3
   j    L1
L4:
   ret
```

Tid til at oversætte til C kode:

```
void sort(int length, int array[]) {
    int i = 1;
    int lastelm = array + length;
        while (&array[i] < lastelm) {
                int j = i-1;
                tmp = array[i];
                while (j>=0 && array[j]> tmp) {
                    array[j+1] = array[j];
                    j--;
                }
                array[j+1] = tmp;
        }
}
```

C kode:

```
void sort(int length, int array[]) {
        int i = 1;
        int lastelm = array + length;
        while (&array[i] < lastelm) {
                int j = i-1;
                tmp = array[i];
                while (j>=0 && array[j]> tmp) {
                        array[j+1] = array[j];
                        j--;
                }
                array[j+1] = tmp;
        }
}
```

```
void sort(int length, int array[]) {
        int i = 1;
        if (length >= i)
                int lastelm = array + length;
                int jbase = 0;
                int j = jbase;
                while (&array[i] < lastelm) {
                        int jbase ++;
                        j = jbase;
                        tmp = array[i];
                        while (j>=0 && array[j]> tmp) {
                                array[j+1] = array[j];
                                j--;
                        }
                        array[j+1] = tmp;
        }
}
```

Question 4.1.5: Descripe shortly the functionality of the program.

Funktionen er insertionsort.

Den leverede tabel sorteres gradvist. Den ydre løkke gennemløber alle elementerne et for et og udvider den sorterede del af tabellen. Den indre løkke placerer hvert element det rigtige sted blandt de sorterede elementer.