

Assignment 2

Faculteit Industriële
Ingenieurswetenschappen
EA-ICT

Kris Myny

RISC-V[®]
ARCHITECTURE

COMPUTER
ARCHITECTURES



De opleiding industrieel ingenieur is een gezamenlijke
opleiding van UHasselt en KU Leuven

► UHASSELT

KU LEUVEN

Assignment 2

- You have to write assembler code for three different programs.
- Use sufficient # to explain the program.
- Generate a small report (*Word or Latex*) showing your code and output for the assignment,
- By when? Wednesday **October 23, 2024**
- Upload on Toledo!

- Which programs?
 1. String reverse
 2. Calculate median → cycle count via Ripes
 3. n'th element Fibonacci calculation → cycle count via Ripes
- Ripes → see last slides

String reverse

- Simple warming up exercise ☺
- Let's load a string and reverse it.
Save the reverse string!
- First calculate the length of the string, don't define this in data.

```
.data  
lab:  
.byte 'h'  
.byte 'e'  
.byte 'l'  
.byte 'l'  
.byte 'o'  
.byte 32  
.byte 'c'  
.byte 'o'  
.byte 'm'  
.byte 'a'  
.byte 'r'  
.byte '!'  
.byte 0
```

- First plot original string, followed by reversed string.

Median

- On the next slide, you can copy the array and array lengths for this lab.
- First, you perform the bubblesort algorithm to sort the array.
- Then, you calculate the median
 - In case of odd number of elements → middle element
 - In case of even number of elements → average of both middle elements (of course with integers and rounded numbers)
- Print the original array, the sorted array and the median. Make even and odd array sizes (min 7 elements); Add this code to the report.
- Remove printing of original array. Keep the printing of the median (for debugging) and count the cycles via Ripes.

Students that improve the cycle count can get a higher score.

This can be done by optimizing the code, or by using an improved sorting algorithm.
Obviously, you need to compare with the original Bubblesort algorithm.

Explain the amendments in your report.

```
void sort(int v[], size_t n)
{
    size_t i, j;
    for (i = n-2; i >= 0; i--){
        for (j = 0; j <= i; j++){
            if (v[j] > v[j+1]){
                swap(v, j);
            }
        }
    }
}
```

To evaluate whether a number is even:
 $\text{Value} = (n+1)\%2$

You can use the rem(ainder) instruction
`rem rd, rs, rs`

If `rd = 1` → even; if `rd = 0` → odd

Arrays

- ❑ length: .word 10
- ❑ array: .word 2, 4, 8, 5, 1, 9, 2, 8, 9, 25

- ❑ length: .word 11
- ❑ array: .word 2, 4, 8, 5, 1, 9, 2, 8, 9, 25, 0

- ❑ length: .word 100
- ❑ array: .word 1138, 533, 1465, 389, 492, 878, 1444, 120, 1127, 1383, 793, 1499, 395, 790, 927, 1375, 936, 306, 1459, 90, 75, 582, 259, 943, 1203, 1305, 596, 143, 581, 781, 1383, 1285, 191, 329, 980, 972, 74, 1203, 1224, 17, 48, 767, 229, 1291, 1099, 745, 679, 596, 1367, 738, 767, 1458, 863, 405, 136, 513, 1472, 1249, 589, 178, 1083, 1292, 1134, 1061, 1125, 173, 47, 127, 396, 1327, 1075, 499, 1496, 732, 1286, 294, 114, 1098, 872, 11, 1447, 125, 931, 1231, 874, 400, 1021, 1206, 1176, 954, 1349, 1358, 1298, 307, 709, 1209, 8, 1012, 866, 1261

- ❑ length: .word 101
- ❑ array: .word 1036, 783, 710, 939, 21, 332, 675, 1398, 1337, 28, 633, 1317, 863, 873, 1094, 886, 1171, 393, 142, 111, 638, 919, 211, 1355, 998, 1131, 821, 930, 195, 276, 704, 330, 337, 160, 133, 1199, 1250, 471, 500, 107, 1466, 774, 644, 1198, 1, 668, 45, 370, 531, 440, 928, 989, 1191, 1310, 13, 1323, 254, 1168, 853, 77, 855, 831, 969, 1435, 716, 861, 239, 1239, 644, 304, 909, 1271, 523, 1360, 183, 917, 1094, 479, 1084, 765, 74, 1438, 1021, 1100, 1494, 600, 1401, 560, 217, 1422, 1383, 505, 1371, 549, 766, 1396, 471, 14, 132, 1082, 400

Fibonacci

- Search google what the Fibonacci sequence is, starting from 0 .. 1 ..
- Define a variable for the n^{th} element
- You can make this recursive or non-recursive
- Print only the result (final value)

Create a graph via data obtained in Ripes.

X-axis n^{th} element (from 1-15)

Y-axis cycle count (taken from Ripes)

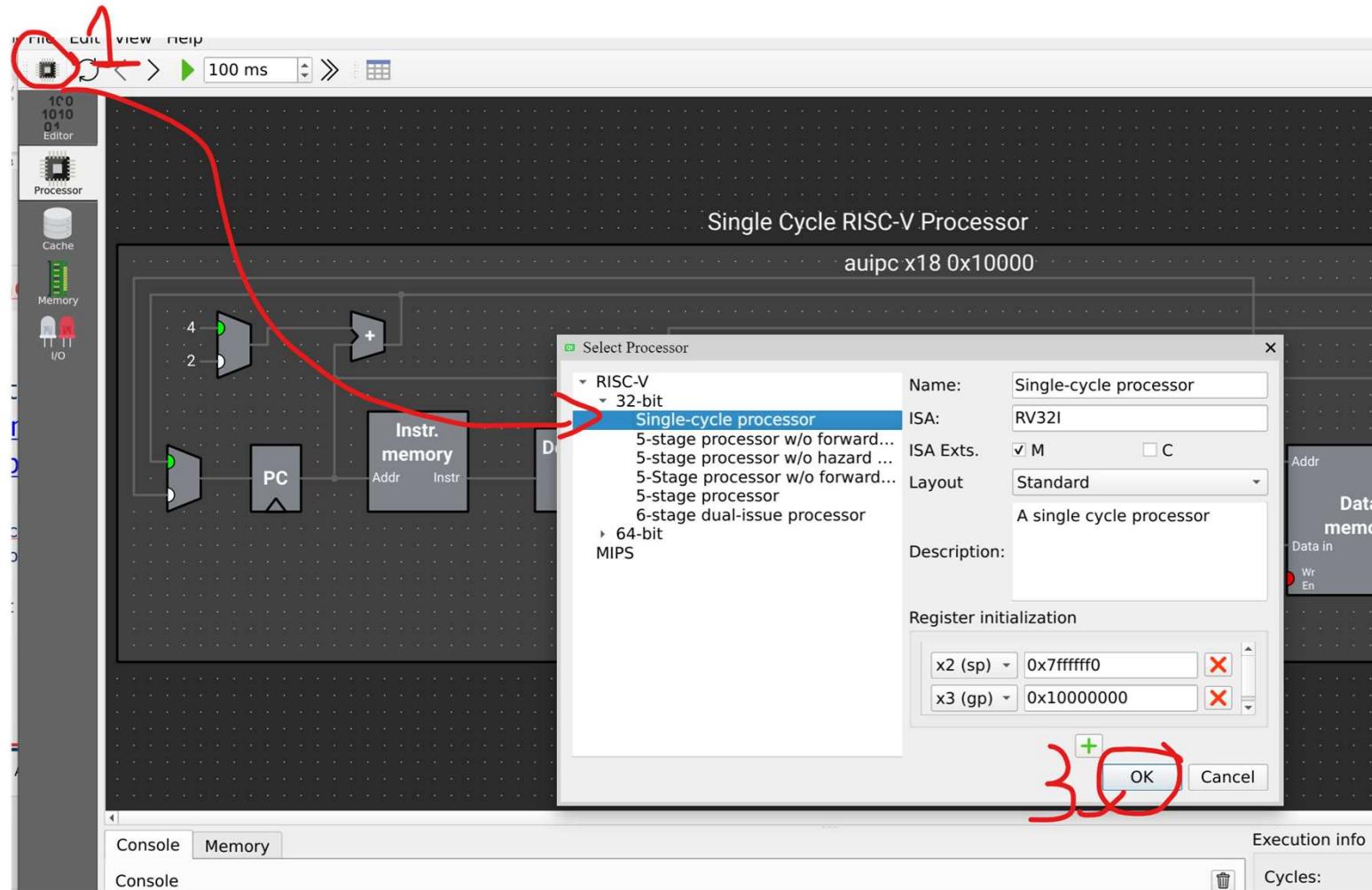
Improvements towards a better cycle count get a higher score.

Add this graph in the report, including some explanations,

Ripes

- Visual simulator of RISC-V
- <https://ripes.me/>
- <https://github.com/mortbopet/Ripes>
- Procedure
 - Define the single cycle processor
 - Load your code from Visual Studio (Venus)
 - Run the code
 - Take a screenshot of the cycle number

Ripes - Define the single cycle processor



Ripes - Load your code from Visual Studio (Venus)

RUN

The screenshot shows the Ripes debugger interface. On the left, there's a sidebar with icons for Processor, Cache, Memory, and I/O. A red circle highlights the 'Processor' icon. Above the processor icon is a red number '1'. In the center, there's a large red circle with the text 'Copy-paste your code here' inside it, pointing towards the source code editor. The source code editor contains assembly code for calculating K = ((a+b) - (c+d)). The assembly code is shown in the middle column, and the disassembly view is selected. The right side of the interface shows a memory dump with hex values and their corresponding memory locations.

Source code

Input type: Assembly Executable code

View mode: Binary Disassembled

1 # Calculate K = ((a+b) - (c+d))
2 .data
3 a: .word 0x19
4 b: .word 0x000F
5 c: .word 0x00000046
6 d: .word 0x0032
7
8 text
9
10 calculate_k:
11 # Load words
12 lw x18, a
13 lw x19, b
14 lw x20, c
15 lw x21, d
16
17 ## NOT NEEDED for this, however, for other exercises!
18 la x22, a # load address of a in X22
19
20 # Do operations
21 add x5, x18, x19 # Temp x5 = a + b
22 add x6, x20, x21 # Temp x6 = c + d
23 sub x5, x6, x5 # Temp x5 = x5 - x6
24
25 # this is how you save d on its location
26 sw x21, 0(x22)
27
28 # Print solution
29 mv a1, x5
30 li a0, 34
31 ecall
32
33 # End simulation
34 li a0, 10
35 ecall
36
37 # csrr t0, 0xC00

00000000 <calculate_k>:
0: 10000917 auipc x18 0x10000
4: 00092903 lw x18 0 x18
8: 10000997 auipc x19 0x10000
C: ffc9a983 lw x19 -4 x19
10: 10000a17 auipc x20 0x10000
14: ff8a2a03 lw x20 -8 x20
18: 10000a97 auipc x21 0x10000
1c: ff4aaa83 lw x21 -12 x21
20: 10000b17 auipc x22 0x10000
24: fe0b0b13 addi x22 x22 -32
28: 013902b3 add x5 x18 x19
2c: 015a0333 add x6 x20 x21
30: 405302b3 sub x5 x6 x5
34: 015b2023 sw x21 0 x22
38: 00028593 addi x11 x5 0
3c: 02200513 addi x10 x0 34
40: 00000073 ecall
44: 00a00513 addi x10 x0 10
48: 00000073 ecall

Pay attention: ripes does not support .byte

Pay attention 2: ecalls are different: reg a7 and a0

Supported system calls

Func. (a7)	Name
1	PrintInt
2	PrintFloat
4	PrintString
10	Exit
11	PrintChar
17	GetCWD
30	Time_msec
31	Cycles
34	PrintIntHex
35	PrintIntBinary
36	PrintIntSigned

Arguments:

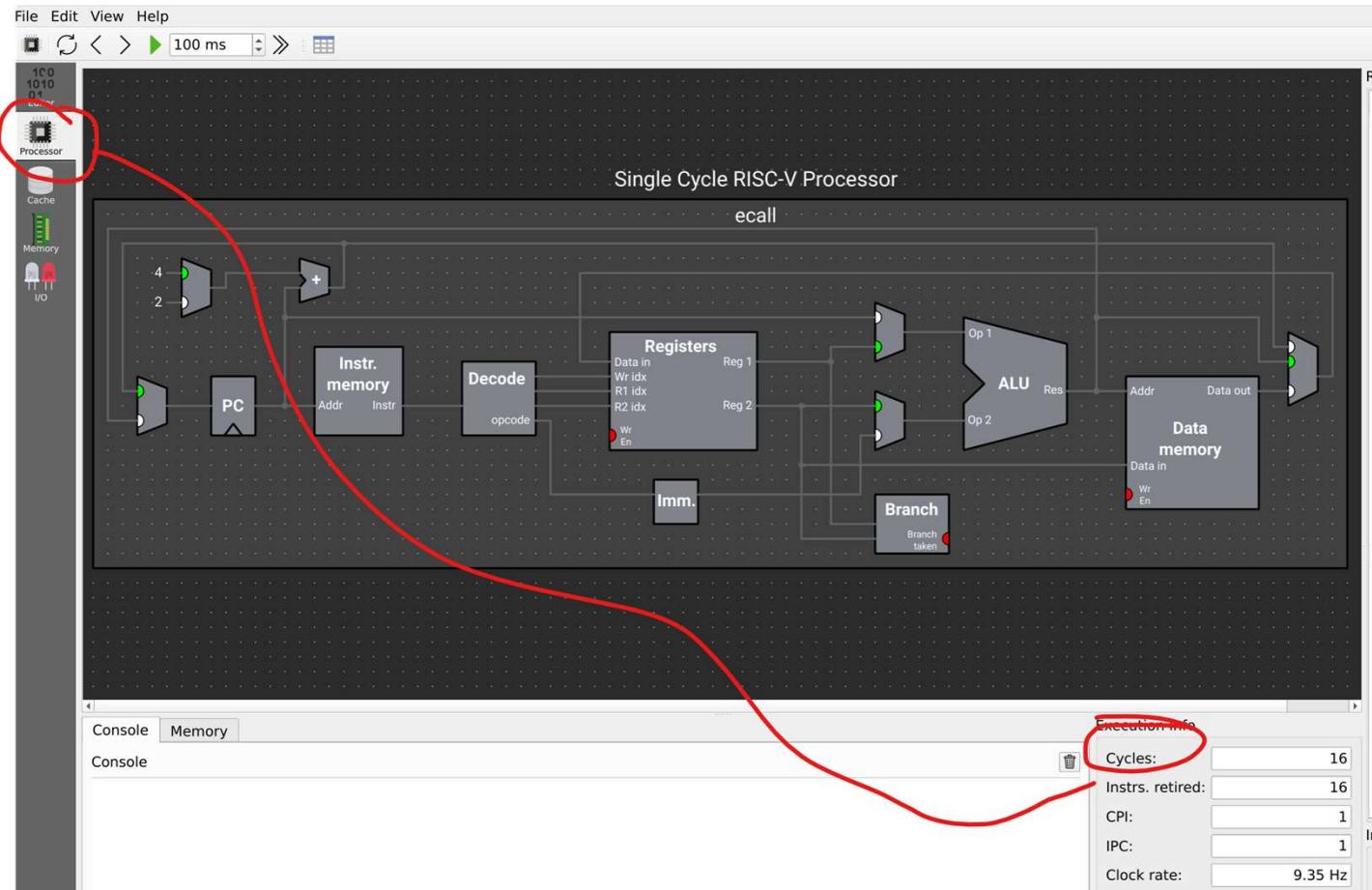
Arg. # Reg.	Description
0	a0 integer to print

Returns:

Ret. # Reg.	Description
-------------	-------------

Menu help → ecalls in Ripes

Ripes - Take a screenshot of the cycle number



This screenshot

Cycles:	16
Instrs. retired:	16
CPI:	1
IPC:	1
Clock rate:	9.35 Hz