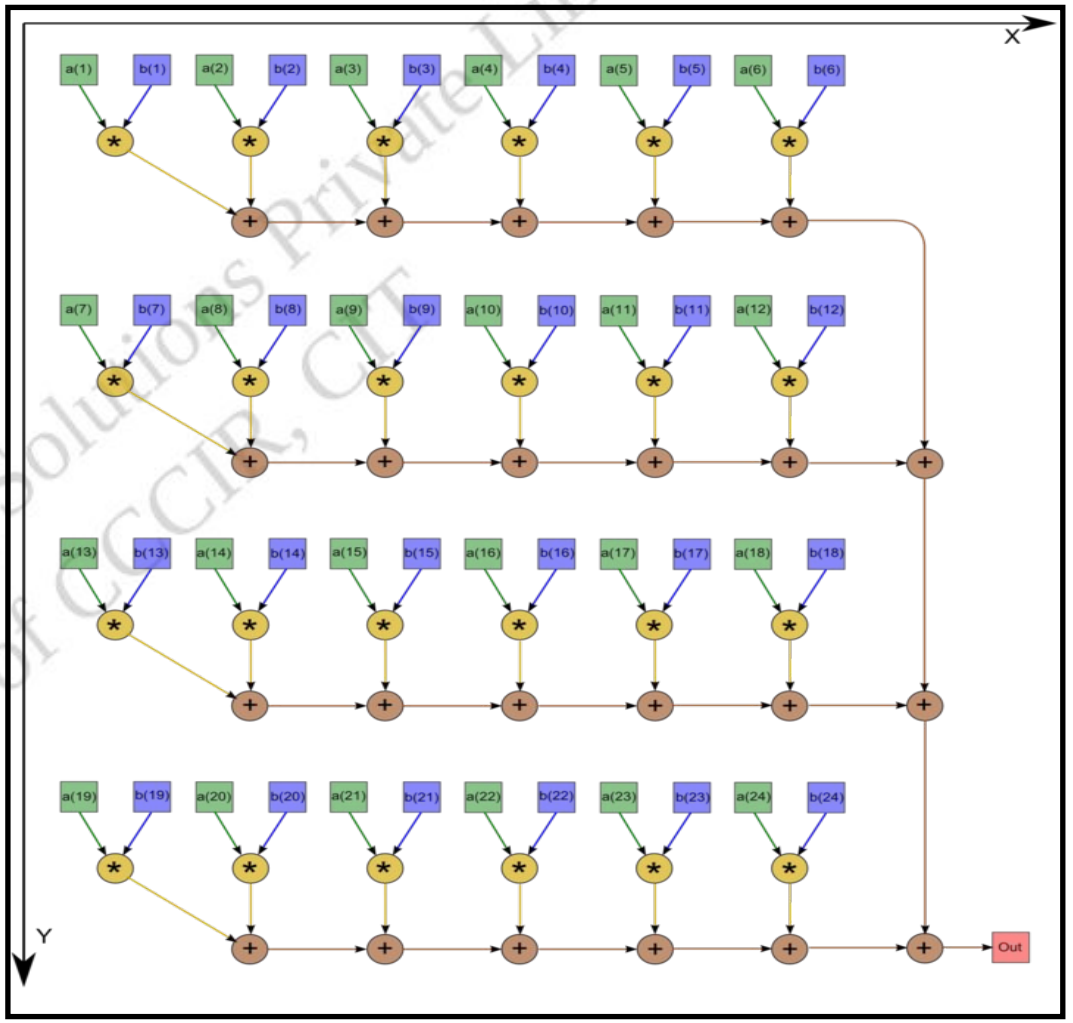
RISC V Assembly Programs

Dr. Girish H
Professor
Department of ECE
Cambridge Institute of Technology
&
Kavinesh
Research Staff
CCCIR
Cambridge Institute of Technology

Dot Product

A dot product is a combination of multiplication and addition between two vectors. Given two vectors
A = [$aa\ bb\ cc$] and D = [$dd\ ee\ ff$]
The dot product A · D is defined by ,A · D = ad + be + cf

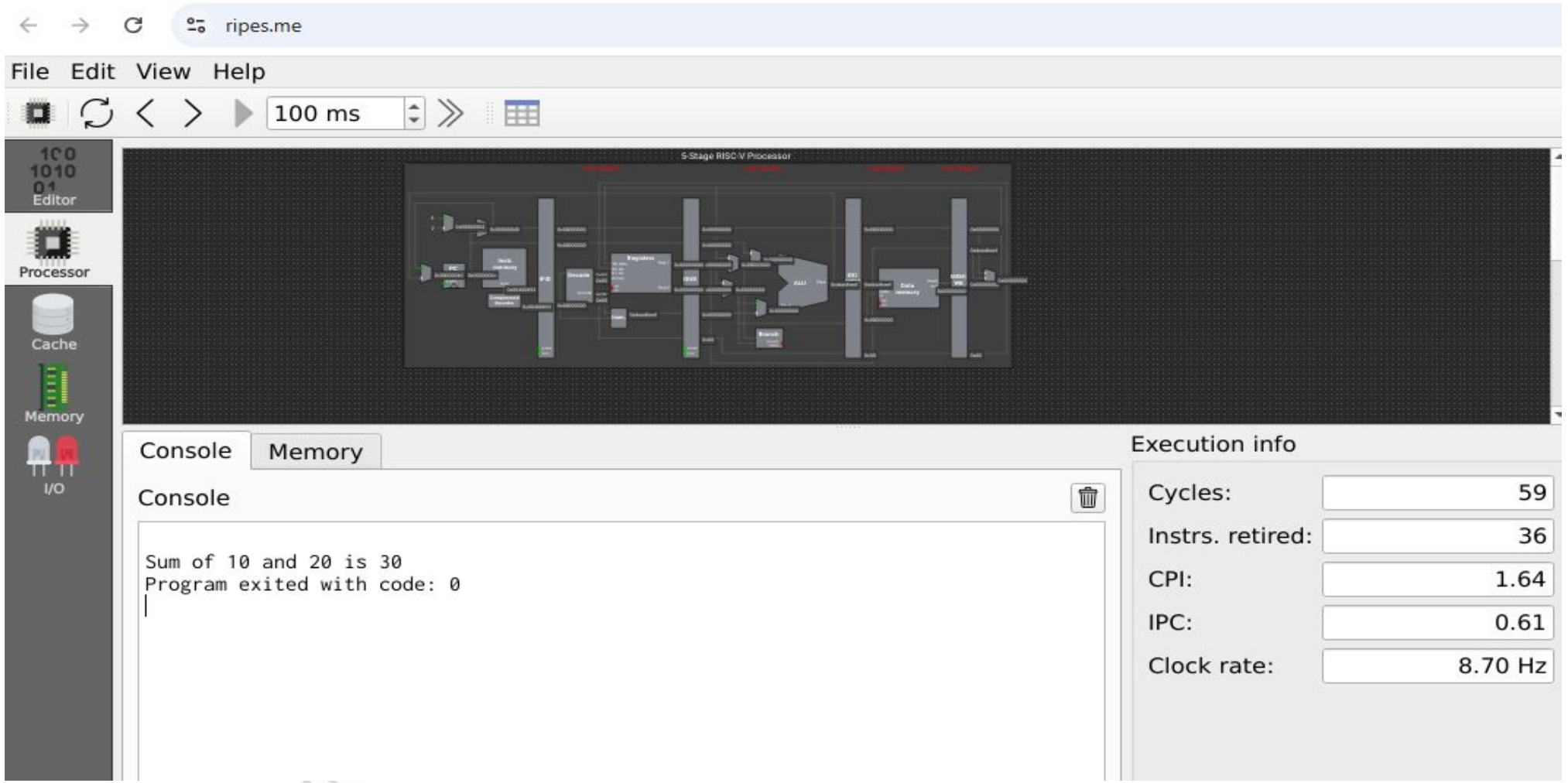


Dot product

```
#include <stdio.h>

int main()
{
    int a[3] = {1, 2, 3};
    int b[3] = {4, 5, 6};
    int sum = 0;
    for (int i = 0; i < 3; ++i)
    {
        sum += a[i] * b[i];
    }
    printf("The inner product of two vectors is %d", sum);
    return 0;
}
```

Dot product

```
.data
arr1: .word 1, 2, 3 # a[3] = {2, 4, 6}
arr2: .word 4, 5, 6 # b[3] = {8, 10, 12}
len:  .word 3       # array length = 3
str:  .string "The inner product of two vectors is "
.text
main:
    la  s1, arr1      # s1 = a
    la  s2, arr2      # s2 = b
    lw  s3, len       # s3 = 3
    add s4, x0, x0    # sum = 0
    add t0, x0, x0    # i = 0
    jal ra, loop      # jump to for loop
    jal ra, print     # jump to for loop
    li  a7, 10        # end program
    ecall
loop:
    lw   t1, 0(s1)    # t1 = a[i]
    addi s1, s1, 4    # ++a (address move forward)
    lw   t2, 0(s2)    # t2 = b[i]
    addi s2, s2, 4    # ++b
    mul  t3, t1, t2   # t3 = a[i] * b[i]
    add  s4, s4, t3   # sum += a[i] * b[i]
    addi t0, t0, 1    # ++i
    blt  t0, s3, loop # if i < 3, go to loop
    ret               # else, return to main
print:
    la  a0, str       # load string
    li  a7, 4         # print string
    ecall
    add a0, s4, x0    # load result
    li  a7, 1         # print integer
    ecall
    ret               # go back to main
```

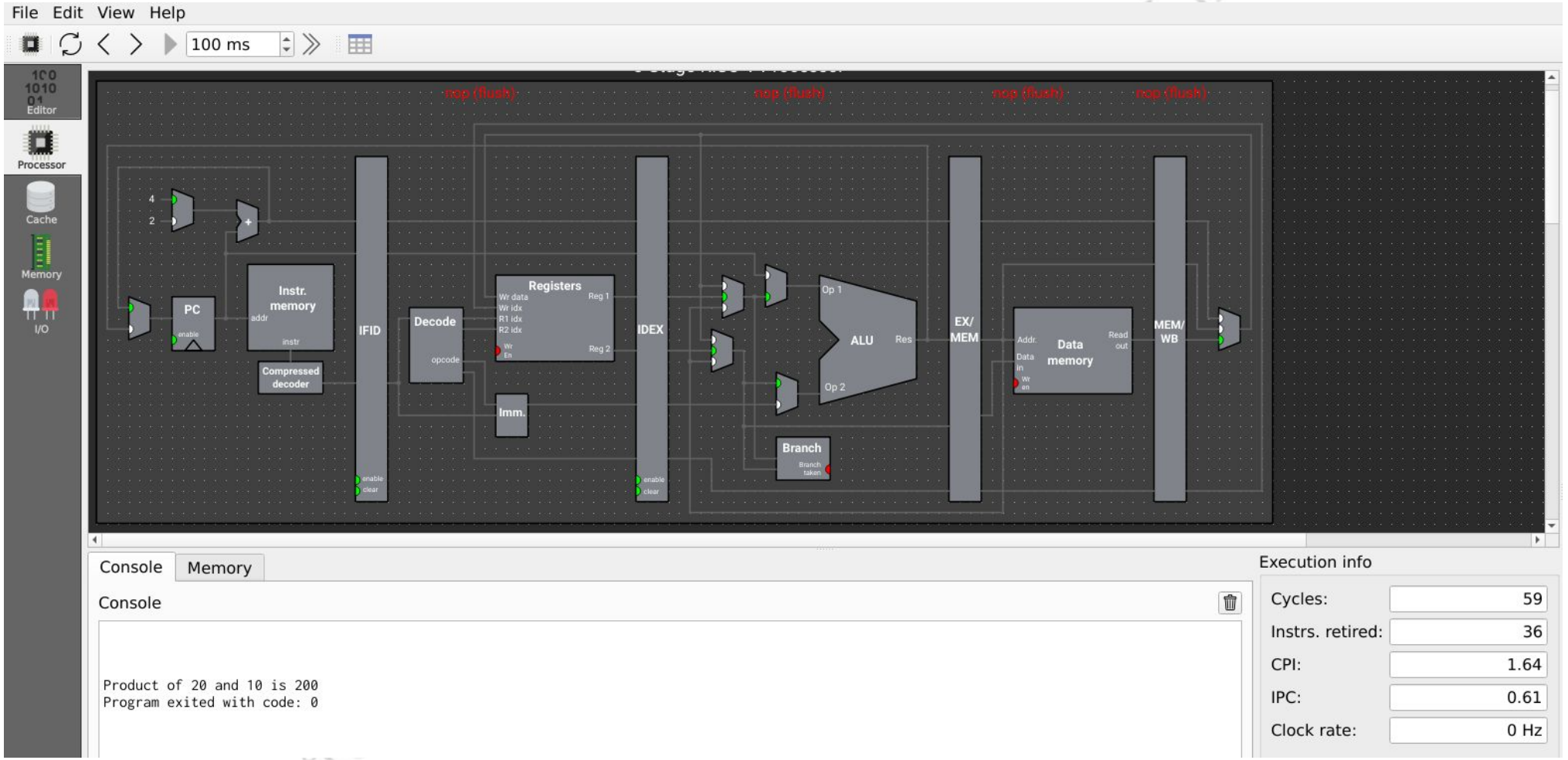


Length of last string

```
#include <stdio.h>
#include <string.h>
int main() {

const char* str1 = "Hello World";
const char* str2 = "Length of the last word is ";

    printf("%s\n", str1);
    int len = strlen(str1);
    int count = 0;
    int i = len - 1;

    while (i >= 0 && str1[i] == ' ') {
    i--;
    }

    while (i >= 0 && str1[i] != ' ') {
    count++;
    i--;
    }

    printf("%s%d\n", str2, count);

    return 0;
}
```

length of string

```
.data
str1:   .string "Hello World"
str2:   .string "Length of the last word is "
space:    .string " "
a:      .string "a"
.text
# s1 = str1 address
# s2 = space
# s3 = counter
# t1 = str1[i]
main:
        #function:lengthOfLastWord(char * s)
        la      s1, str1
        lb      s2, space
        lb      s4, 5(s1)
        add             s3, x0, x0
        jal     ra, loop1

        lb      t0, 0(s1)
        jal     ra, loop2
        jal     ra, loop3
        jal     ra, print
        li      a7, 10
        ecall
loop1:
        addi            s1, s1, 1 #find the last position of the string
        lb      t0, 0(s1)
        bne             t0, x0, loop1
        ret
loop2:
        addi            s1, s1, -1      #s--
        lb      t0, 0(s1)       #t0=char[i](s), i=length(s)-1
        beq             t0, s2, loop2 #while(char[i]==" ")
        ret
loop3:
        addi            s1, s1, -1      #s--
        lb      t0, 0(s1)       #t1=char[i](s), i=last word
        addi            s3, s3, 1       #length++
        bne             t0, s2, loop3 #while(char[i]!=" ")
        ret
print:
        la      a0, str2
        addi            a7, x0, 4
        ecall
        add             a0, s3, x0
        li      a7, 1
        ecall
        ret
```



Sum of 1d array

```
#include <stdlib.h>
#include <stdio.h>
int* runningSum(int* nums, int numsSize){
int* result = (int*)malloc(sizeof(int)*numsSize);

    for(int i=0; i<numsSize; i++){
    if(i==0)
        result[0] = nums[0];
    else
        result[i] = result[i-1] + nums[i];
    }
    return result;
}
int main(int argc, char *argv[]){
    int nums1[5] = {1,2,3,4,5};
    int nums2[6] = {1,3,5,7,9,11};
    int nums3[7] = {0,2,4,6,8,10,12};
    int* result1 = runningSum(nums1, 5);
    int* result2 = runningSum(nums2, 6);
    int* result3 = runningSum(nums3, 7);
    for(int j=0; j<5; j++){
    printf("%d ", result1[j]);
    }
    printf("\n");
    for(int j=0; j<6; j++){
    printf("%d ", result2[j]);
    }
    printf("\n");
    for(int j=0; j<7; j++){
    printf("%d ", result3[j]);
    }
    printf("\n");
}
```

Sum of 1d array

```
   .data
nums1:    .word 1,2,3,4,5
nums2:    .word 1,3,5,7,9,11
nums3:    .word 0,2,4,6,8,10,12
res1:     .word 0,0,0,0,0,
res2:     .word 0,0,0,0,0,0
res3:     .word 0,0,0,0,0,0,0
space:    .string " "
nl:    .string "\n"
.text
main:
la    t2, nums1  # load arr base to t2
la    t3, res1   # load res base to t3
li    t4, 5      # load the numSize to t4
jal   ra, runningSum
la    t2, nums2  # load arr base to t2
la    t3, res2   # load res base to t3
li    t4, 6      # load the numSize to t4
jal   ra, runningSum
la    t2, nums3  # load arr base to t2
la    t3, res3   # load res base to t3
li    t4, 7      # load the numSize to t4
jal   ra, runningSum
j     exit
runningSum:
add  t6, t3, zero
li    t5, 1      # load i with 1
lw    a3, 0(t2)  # load nums[0] to a3
sw    a3, 0(t6)  # save num[0] to result[0]
loop:
addi t2, t2, 4   # nums[i] addr
lw    a2, 0(t2)  # nums[i]
add  a3, a3, a2 # add result[i-1] +
nums[i]
addi t6, t6, 4   # result[i]
addi t5, t5, 1   # i = i + 1
sw    a3, 0(t6)  # save result[0] +
nums[1] to result[1]
      blt    t5, t4, loop  # if i < numsSize
then branch to loop
print:
      lw     a0, 0(t3)  # load result out
      li     a7, 1      # pint a0
      ecall
      la     a0, space      # load space
      li     a7, 4      # print string
      ecall
      addi t3, t3,  4
      addi t5, t5, -1  # i--
      blt    zero, t5, print
      la     a0, nl     # load next line
      li     a7, 4      # print string
      ecall
      jr     ra
exit:
      li     a7, 10     # exit
      ecall
```

Program for factorial in RISC-V Inline assembly

```
#include<stdio.h>
void factorial_asm(void);
static int data = 5;
static int factorial;
int main()
{
factorial_asm();
printf("factorial is %d", factorial);
}

void factorial_asm()
{
        asm volatile(
"la x12, data\n\t"
"lw x13, 0(x12)\n\t"
"addi x14, x0, 1\n\t"
"loop:\n\t"
"mul x14, x14, x13\n\t"
"addi x13, x13, -1\n\t"
"blt x0, x13, loop\n\t"
"la x15,factorial\n\t"
"sw x14,0(x15)\n\t"
);
}
```

Palindrome

```
include <stdio.h>

void swap_asm(void);
static int a=121;
static int b;

int main()
{
 swap_asm();
if (a==b)
 printf("number is a palindrome %d = %d\n", a, b);
else
 printf("not a palindrome\n");
}

void swap_asm()
{
asm volatile(
"la x10, a\n\t"
"ld x11, 0(x10)\n\t"
"addi x5, x11, 0\n\t"
"addi x6, x0, 0\n\t"
"loop:\n\t"
"addi x7, x0, 10\n\t"
"mul x6, x6, x7\n\t"
"rem x8, x5, x7\n\t"
"add x6, x6, x8\n\t"
"div x5, x5, x7\n\t"
"bne x5, x0, loop\n\t"
"la x12, b\n\t"
"sw x6, 0(x12)\n\t"
          );
}
```

Program to Swap two numbers

```
#include <stdio.h>
void swap_asm(void);
static int a=5;
static int b=7;
int main()
{
 swap_asm();
 printf("after swap a = %d, b= %d", a, b);
}

void swap_asm()
{
 asm volatile(
          "la x10, a\n\t"
          "lw x11, 0(x10)\n\t"
          "la x12, b\n\t"
          "lw x13, 0(x12)\n\t"
          "sw x13, 0(x10)\n\t"
          "sw x11, 0(x12)\n\t"
         );
}
```

Swap two number and profile result using csr

```
#include <stdio.h>
void swap_asm(void);
static int a=5;
static int b=7;

nsigned long read_cycles(void)
{
  unsigned long cycles;
  asm volatile ("rdcycle %0" : "=r" (cycles));
  return cycles;
}
int main()
{
unsigned long start, stop;
    start = read_cycles();
 swap_asm();
stop = read_cycles();
    printf(" cycle :%ld\n", stop - start);
 printf("after swap a = %d, b= %d", a, b);
}
```

```
void swap_asm()
{
 asm volatile(
          "la x10, a\n\t"
          "lw x11, 0(x10)\n\t"
          "la x12, b\n\t"
          "lw x13, 0(x12)\n\t"
          "sw x13, 0(x10)\n\t"
          "sw x11, 0(x12)\n\t"
         );
}
```

Pascal Triangle

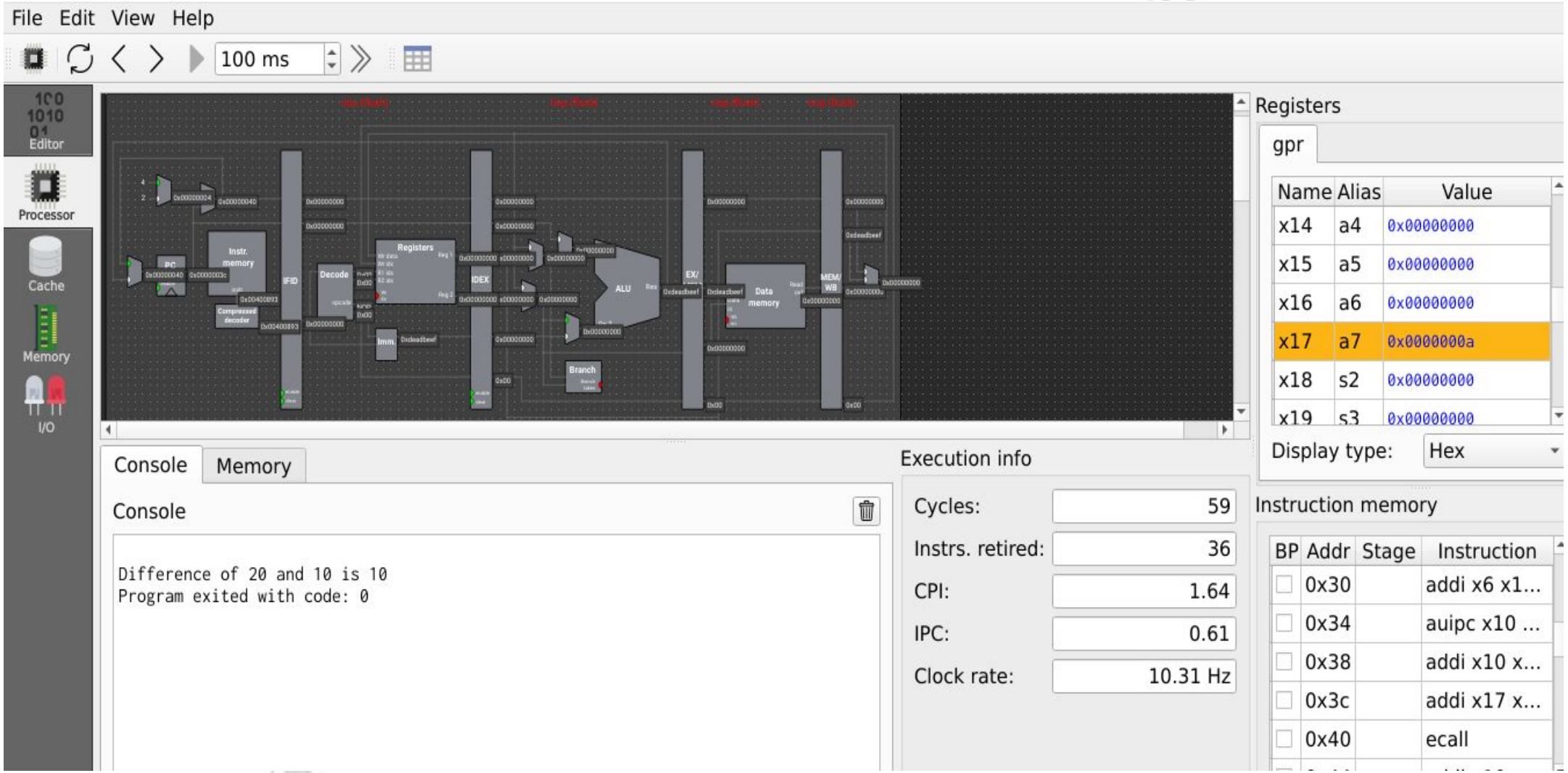
```
#include<stdio.h>
void swap(int res[34], int temp[34]) {
  int a ;
  for ( int i = 0 ; i < 34 ; i ++ ) {
    a = res[i] ;
    res[i] = temp[i] ;
    temp[i] = a ;
  } // for
} // swap

int main() {
  int result[34] = {0} ;
  int temp[34] = {0} ;
  int rowIndex = 5 ;
  int limit = 33 ;
  if ( rowIndex < 0 || rowIndex > 33 ) { // error
    printf("Input is out of range!\n") ;
    return ;    }
  else if ( rowIndex == 0 ) {
        result[0] = 1 ;
  } // else if
else if ( rowIndex == 1 ) {
        result[0] = 1 ;
        result[1] = 1 ;
  } // else if
  else {  // rowIndex > 1
        result[0] = 1 ;
        result[1] = 1 ;
        for ( int i = 2 ; i <= rowIndex ; i ++ ) {
        temp[0] = 1 ;
        for ( int j = 1 ; j < i ; j ++ ) {
        temp[j] = result[j-1] + result[j] ;
        } // for
        temp[i] = 1 ;
        swap(result, temp) ;
        } // for
  } // else
  printf("[%d", result[0]) ;
  for ( int k = 1 ; k <= rowIndex ; k ++ ) {
        printf(",%d", result[k]) ;
  } // for
  printf("]\n") ;
}
```

Pascal triangle

```
.data
str1:    .string "["
str2:    .string ","
str3:    .string "]\n"
str4:    .string "Input is out of range!\n"
        .text
main:
        addi sp, sp, -272      # initialize the space of the two
arrays
        addi s1, sp, 136       # address of result array
        add  s2, sp, zero      # address of temporary array
        addi s3, zero, 5       # rowIndex = 5
        li   s0, 33            # input limit is 33
        blt  s3, zero, error   # if ( rowIndex < 0 ) goto error
        bgt  s3, s0, error     # if ( rowIndex > 33 ) goto
        add  t0, zero, zero    # t0 = 0
        beq  s3, t0, input0    # if ( rowIndex == 0 )
        addi t0, zero, 1       # t0 = 1
        beq  s3, t0, input1    # if ( rowIndex == 1 )
        j    other             # rowIndex > 1
input0:
        addi t0, zero, 1       # t0 = 1
        sw   t0, 0(s1)         # result array = {1}
        j    printArr
input1:
        addi t0, zero, 1       # t0 = 1
        sw   t0, 0(s1)         # result array = {1}
        sw   t0, 4(s1)         # result array = {1,1}
        j    printArr
```

```
other:
        addi t0, zero, 1        # t0 = 1
        sw   t0, 0(s1)          # result array = {1}
        sw   t0, 4(s1)          # result array = {1,1}
        li   s4, 2              # s4 = i, initialize to 2
outerLoop:
        bgt  s4, s3, printArr
        sw   t0, 0(s2)          # temporary array = {1}
        add  t1, zero, t0
        add  s5, zero, t0       # s5 = j, initialize to 1
innerLoop:
        bge  s5, s4, outerIncre
        slli t2, t1, 2          # t2 is the offset to access temporary
        add  t2, t2, s2         # t2 is the address of temp[j]

        addi t3, s5, -1         # t3 = j - 1
        slli t3, t3, 2          # t3 is the offset
        add  t3, t3, s1         # t3 is the address of result[j - 1]
        lw   t3, 0(t3)          # t3 = result[j - 1]
        add  t4, s5, zero       # t4 = j
        slli t4, t4, 2          # t4 is the offset
        add  t4, t4, s1         # t4 is the address of result[j]
        lw   t4, 0(t4)          # t4 = result[j]
        add  t3, t3, t4         # t3 = result[j - 1] + result[j]
        sw   t3, 0(t2)          # temp[j] = result[j - 1] + result[j]
        addi t1, t1, 1          # t1 = t1 + 1, t1 is index of temporary
        addi s5, s5, 1          # j = j + 1
        j    innerLoop          # goto innerLoop
outerIncre:
        add  t2, s4, zero       # t2 = i
        slli t2, t2, 2          # t2 is the offset
        add  t2, t2, s2         # t2 is the address of temp[i]
        sw   t0, 0(t2)          # temp[i] = 1
        add  t2, s1, zero       # exchange the addresses of
        add  s1, s2, zero       # result array and temporary array
        add  s2, t2, zero

        addi s4, s4, 1          # i = i + 1
```

```
        j    outerLoop          # goto outerLoop
printArr:
        add  t0, zero, zero     # k = 0, index of print loop
        la   a0, str1           # load label str1, which is "["
        li   a7, 4              # a7 = 4, which means ecall will print a
string
        ecall
        slli t1, t0, 2          # t1 is the offset
        add  t1, t1, s1         # t1 is the address of result[0]
        lw   t1, 0(t1)          # t1 = result[0]
        add  a0, t1, zero       # a0 = result[0]
        li   a7, 1              # a7 = 1, which means ecall will print
        ecall
        addi t0, t0, 1          # k = 1
printLoop:
        bgt  t0, s3, printStr3
        la   a0, str2           # load label str2, which is ","
        li   a7, 4              # a7 = 4, which means ecall will print a
        ecall
        slli t1, t0, 2          # t1 is the offset
        add  t1, t1, s1         # t1 is the address of result[k]
        lw   t1, 0(t1)          # t1 = result[k]
        add  a0, t1, zero       # a0 = result[k]
        li   a7, 1              # a7 = 1, which means ecall will print a
        ecall
        addi t0, t0, 1          # k = k + 1
        j    printLoop
printStr3:
        la   a0, str3           # load label str3, which is "]\n"
        li   a7, 4              # a7 = 4, which means ecall will print a
string
        ecall
        j    exit               # goto exit
error:
        la   a0, str4           # load label str4, which is the error
message
        li   a7, 4              # a7 = 4, which means ecall will print a
string
        ecall
exit:
        addi sp, sp, 272        # release stack space
```



Thank you