1.

(a)

Gp register is initialized:

一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

The gp register plus 5064 to get the memory address of result

The memory address of procedure of sudan is 0x104a8

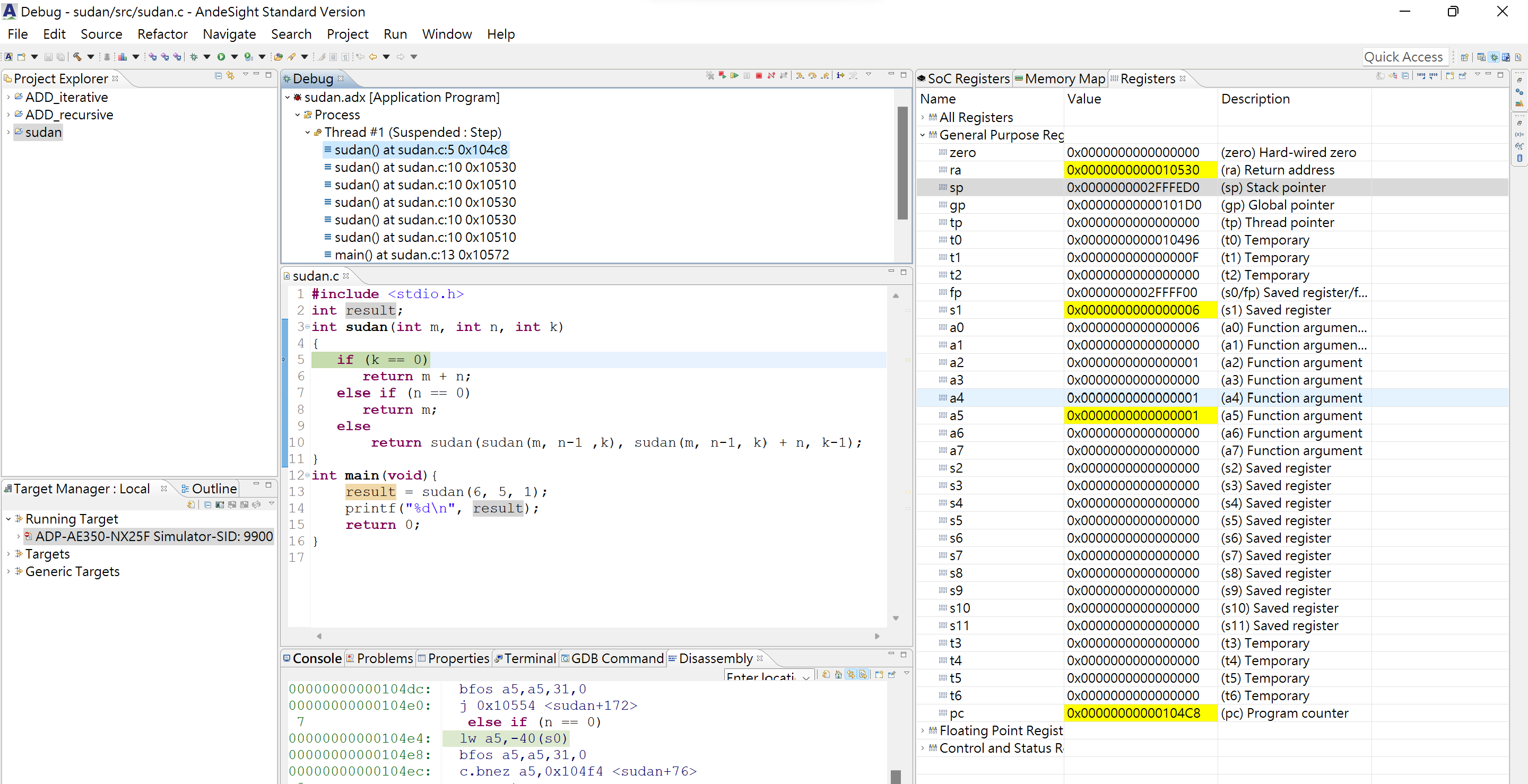
(b)

(1)

一張含有 桌 的圖片

自動產生的描述

(2)



0x0000000002FFFED0

(c)

一張含有 文字 的圖片

自動產生的描述

We have a0 = m, a1 = n, a2 = k

Makes a5 = -40(s0) = n, and n-1. Then make a4 = a5, a3 = -44(s0) = k ,

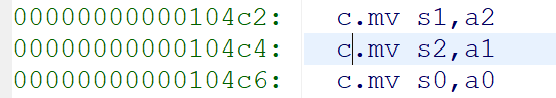
and a5 = -36(s0) = m

Then make a0, a1, a2 be replaced by a5, a4, a3(new arguments).

Finally, jump to 0x104a8 and redo sudan.

(d)

(1)



S1 = k, s2 = n, s0 = m

一張含有 文字 的圖片

自動產生的描述

The value of s3 = n-1, and make a1 = s3 = n-1, then do sudan by a0=m, a1=(n-1), and a2=k call sudan(m, n-1, k)

Then move m to a0, then call sudan again (sudan(m,n-1,k) )

Then make a2 = k-1, a1 = a0(the return value of sudan = sudan(m, n-1, k))+s2(n), and a0 = sudan(…)

And call again sudan(sudan(…), sudan(…)+n, k-1)

(2)

In og use more saved register so can reduce instructions to move values in registers.

In O1 will load value from stacks to argument registers so makes it slower than Og .

2.

**Little-Endian System:** the bytes in the low significant digit will store in earlier memory address

00000000:    9E

00000001: CF

00000002: A7

00000003: 14

**Big-Endian System:** the bytes in the most significant digit will store in earlier memory address

00000000:    14

00000001: A7

00000002: CF

00000003: 9E

3.

(a)

x10 = base address of array A  
x11 = base address of array B  
x5 = g  
x6 = h  
x7 = i

x8 = j

B[8] = A[g + h]  
**RISC code**  
add x30 , x5 , x6 //x30 = g + h  
sll x30 , x30 , 3 //x30 = 8\*(g + h)  
add x30 , x30 , x10 //x30 = address of A[g + h]  
ld x30 , 0(x30) //x30 = A[g + h]  
sd x30 , 64(x11) //B[8] = A[g + h]

(b)

i = A[B[4]] - j;

**RISC code**

ld x30, 32(x11) //x30 = B[4]

sll x30, x30, 3 //x30 = B[4]\*8

add x30, x10, x30 //x30 = address of A[B[4]]

ld x30, 0(x30) //x30 = A[B[4]]

sub x7, x30, x8 //i = A[B[4]]-j

4.

The RISC-V instruction has 32 bits

(beq)

4\*5 = 20

Immediate data = 20’s binary = 00000001010 0

Rs2 = 11000, rs1 = 01001, funct3=000, op code = 1100011

So BEQ in binary is 0000 0001 1000 0100 1000 1010 0110 0011

(jal)

Immediate data =-7\*4 = -28 = 11111…11110010 0

Rd=00000, op code = 1101111

So JAL in binary is 1111 1110 0101 1111 1111 0000 0110 1111

5.

(a) sra x8, x18 ,x23 (r-type)

(b) 0000010 00110 11010 011 10000 0100011 = 046D3823 (s-type)

6.

int i = 100;

for(int j = 0; i>j;){

result += \*MemArray;

MemArray += 1;

i = i – 4;

}

7.

fib:

ble a0, s0, L1 # if(n <= 1)

addi sp, sp, -12 # push the stack

sw ra, 8(sp) # store return address

sw a0, 4(sp) # store argument n

addi a0, a0, -1 # argument = n - 1

jal ra, fib # call fib(n - 1)

sw a0, 0(sp) # store return value of fib(n - 1)

lw a0, 4(sp) # load argument n

addi a0, a0, -2 # argument = n - 2

jal ra, fib # call fib(n - 2)

lw t0, 0(sp) # load return value of fib(n - 1)

add t0, t0, t0 #t0 = 2\*t0

add a0, a0, t0 # fib(n - 1) +2\* fib(n - 2)

lw ra, 8(sp) # load return address

addi sp, sp, 12 # pop the stack

ret # return

L1:

ret # return

8.

(a)

Need 7 bits to represent op, so num of instruction is >64 and <128.

Expand instruction to four times => num of instructions is >256 and <512

So we need 9 bits to represent instructions

Need 7 bits to represent 128 registers, so rs2, rs1 and rd will each require 7 bits

Therefore, 9 +3\*7 = 30 bits are used, and remains 2 bits to represent funt3 and funct7.

We can represent them by (1 bit and 1 bit). So there is one possibility.

(b)

Programs may decrease in size because more registers means you can hold onto more data in the processor without going out to memory and by allowing more complex operations to be implemented in one instruction vs requiring multiple instructions.

Programs increase in size because they could increase the size of a program by requiring adding bits to the opcode and register fields, and this increase memory width.