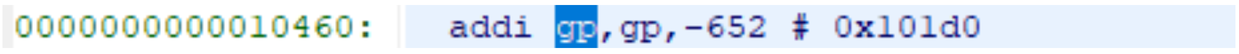
108062213 顏浩昀
Computer Architecture HW2
2022.03.31

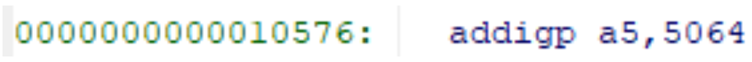
1.

a). First, show how the memory address of the gp register is initialized. Next, show how to get the memory address of the "result" variable by referencing the gp register in "main". Finally, find the memory address of the procedure "sudan".



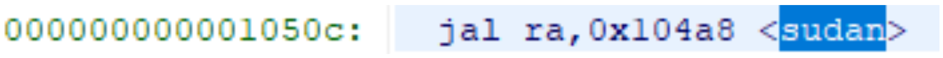
```
0000000000010460:    addi gp,gp,-652 # 0x101d0
```

By above line, I can get gp initial address is 0x101d0.



```
0000000000010576:    addigp a5,5064
```

By above line, I can get address of result which is stored in a5 by gp-5064. Hence, initial address of gp is 0x101d0, I can calculate memory address of result is 0x101d0+0x13C8(equal to 5064 in decimal) = 0x11598.

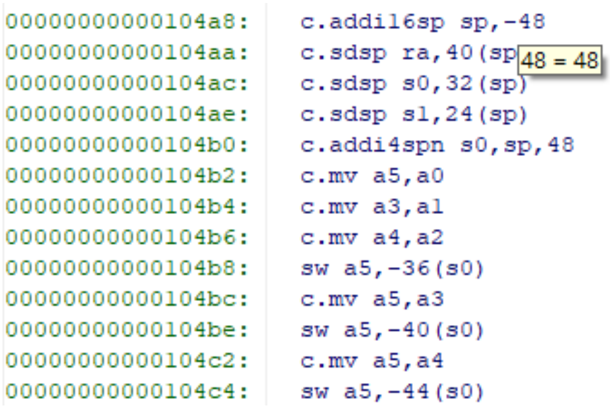


```
000000000001050c:    jal ra,0x104a8 <sudan>
```

By above line, I can get the address of sudan is 0x104a8.

b).

i. Record the assembly code from memory address 0x104a8 to 0x104c4. Fill in the stack block by register name and the corresponding memory offsets of the stack in the below figure.



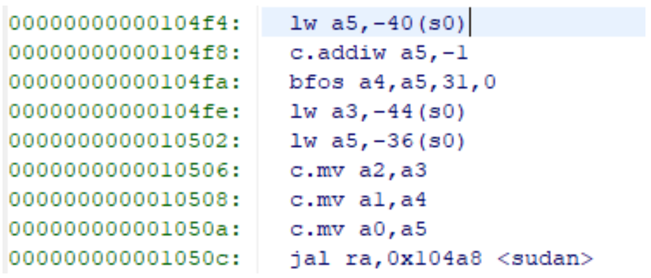
```
00000000000104a8:    c.addi16sp sp,-48
00000000000104aa:    c.sdsp ra,40(sp
00000000000104ac:    c.sdsp s0,32(sp)
00000000000104ae:    c.sdsp s1,24(sp)
00000000000104b0:    c.addi4spn s0,sp,48
00000000000104b2:    c.mv a5,a0
00000000000104b4:    c.mv a3,a1
00000000000104b6:    c.mv a4,a2
00000000000104b8:    sw a5,-36(s0)
00000000000104bc:    c.mv a5,a3
00000000000104be:    sw a5,-40(s0)
00000000000104c2:    c.mv a5,a4
00000000000104c4:    sw a5,-44(s0)
```

High memory address		
		0
		-4
	ra	-8
		-12
	s0	-16
		-20
	s1	-24
		-28
		-32
	a5	-36
	a5(a3)	-40
	a5(a4)	-44
		-48
Low memory address		

ii. During the execution of the sudan function, what is the lowest memory address that the stack pointer pointed to?

The lowest memory address is -48.

c). Record the assembly code from memory address 0x104f4 to 0x1050c, which correspond to part of the statement: return sudan(sudan(m,n-1,k),sudan(m,n-1,k)+n,k-1); Briefly explain what these instructions do.



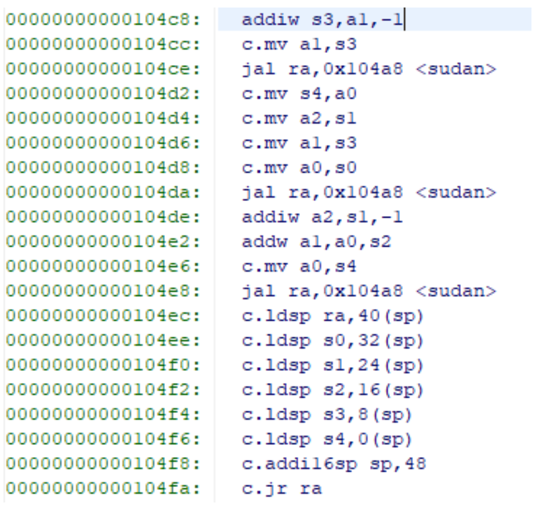
```
00000000000104f4:   lw a5,-40(s0)
00000000000104f8:   c.addiw a5,-1
00000000000104fa:   bfos a4,a5,31,0
00000000000104fe:   lw a3,-44(s0)
0000000000010502:   lw a5,-36(s0)
0000000000010506:   c.mv a2,a3
0000000000010508:   c.mv a1,a4
000000000001050a:   c.mv a0,a5
000000000001050c:   jal ra,0x104a8 <sudan>
```

1. Load data from memory address s0 with offset -40 to reg a5.
2. Add -1 immeidately to reg a5.
3. a4, a5 bit field operation.
4. Load data from memory address s0 with offset -44 to reg a3, with offset -36 to reg a5
5. Copy a3 value to a2, copy a4 value to a1, copy a5 value to a0.
6. Jump to 0x104a8(<sudan>)

d).

i. Change the optimization level to -Og, and do the same as (c) but with the whole statement: return sudan(sudan(m,n-1,k),sudan(m,n-1,k)+n,k -1).

-Og:



```
00000000000104c8:   addiw s3,a1,-1
00000000000104cc:   c.mv a1,s3
00000000000104ce:   jal ra,0x104a8 <sudan>
00000000000104d2:   c.mv s4,a0
00000000000104d4:   c.mv a2,s1
00000000000104d6:   c.mv a1,s3
00000000000104d8:   c.mv a0,s0
00000000000104da:   jal ra,0x104a8 <sudan>
00000000000104de:   addiw a2,s1,-1
00000000000104e2:   addw a1,a0,s2
00000000000104e6:   c.mv a0,s4
00000000000104e8:   jal ra,0x104a8 <sudan>
00000000000104ec:   c.ldsp ra,40(sp)
00000000000104ee:   c.ldsp s0,32(sp)
00000000000104f0:   c.ldsp s1,24(sp)
00000000000104f2:   c.ldsp s2,16(sp)
00000000000104f4:   c.ldsp s3,8(sp)
00000000000104f6:   c.ldsp s4,0(sp)
00000000000104f8:   c.addi16sp sp,48
00000000000104fa:   c.jr ra
```

1. Add -1 to a1 and put the value to s3.
2. Copy value in s3 to a1.
3. Jump to 0x104a8(sudan)
4. Copy value (a0->s4, s1->a2, s3->a1, s0->a0)
5. Jump to 0x104a8(sudan)
6. Add -1 to s1 and put value to a2
7. Add value in a0 and a2 and put value to a1
8. Copy value in s4 to a0
9. Jump to 0x104a8(sudan)
10. Restore ra, s0~s4 from stack
11. Return space on stack
12. Return to caller

ii. Compare the difference of the assembly codes according to the return statements that are generated by the different optimization levels -Og and -O0.
When using -Og level, more register are be used repeatedly, so it could take less moving steps. There are significantly fewer amount of instruction than -O0 level.

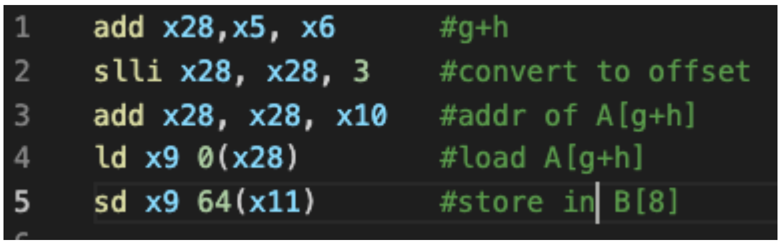
2. Respectively show how the value 14A7CF9E$_{hex}$ would be arranged in the memory of a little-endian machine and a big-endian machine. Assume that the machines are byte-addressable and the data are stored starting at address 0x00000000.

Little-endian	
0x00000000	9E
0x00000001	CF
0x00000002	A7
0x00000003	14

Big-endian	
0x00000000	14
0x00000001	A7
0x00000002	CF
0x00000003	9E

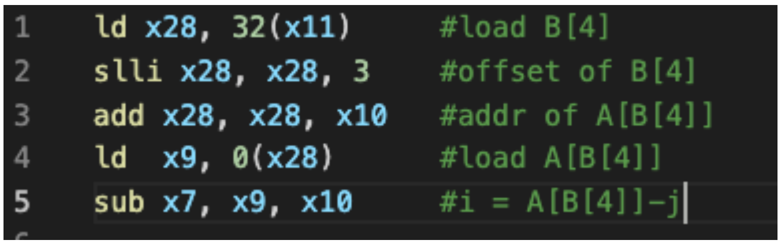
3. For each of the following C statements, write the corresponding RISC-V assembly code. Assume that the base addresses of arrays A and B are in registers x10 and x11, respectively. Each element of A or B is 8 bytes, and the variables g, h, i, j are assigned to registers x5, x6, x7 and x8, respectively.

a). `B[8] = A[g + h];`



```
add x28,x5, x6      #g+h
slli x28, x28, 3    #convert to offset
add x28, x28, x10   #addr of A[g+h]
ld x9 0(x28)        #load A[g+h]
sd x9 64(x11)       #store in B[8]
```

b). `i = A[B[4]] - j;`



```
ld x28, 32(x11)     #load B[4]
slli x28, x28, 3    #offset of B[4]
add x28, x28, x10   #addr of A[B[4]]
ld  x9, 0(x28)      #load A[B[4]]
sub x7, x9, x10     #i = A[B[4]]-j
```

4. Consider the following code sequence, assuming that LOOP is at memory location 1024. What is the binary representation for the 4th instruction (beq) and the 8th instruction (jal)?

4th instruction (beq x9, x24, EXIT):

imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[0]	opcode
0	000000	11000	01001	000	1010	0	1100011

8th instruction (jal x0, LOOP)

imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode
1	1111110010	1	11111111	00000	1101111

5.

a). Give the instruction type and assembly instruction for the following RISC-V machine instruction: 0100 0001 0111 1001 0101 0100 0011 0011

First check opcode=0110011 : R-type.
So I go to find funct7=0100000, funct3=101 : sra.
Next, I find rs2=10111, rs1=10010, rd=01000.
I can get the instruction is : sra, x8, x18, x23

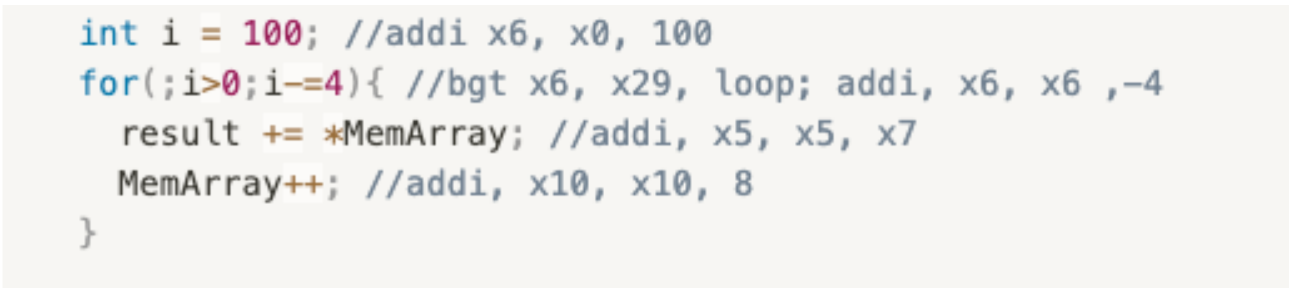
b). Give the instruction type and hexadecimal representation of the following RISC-V assembly instruction: sd x6, 80(x26)

sd is S-type

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
0000010	00110	11010	011	10000	0100011

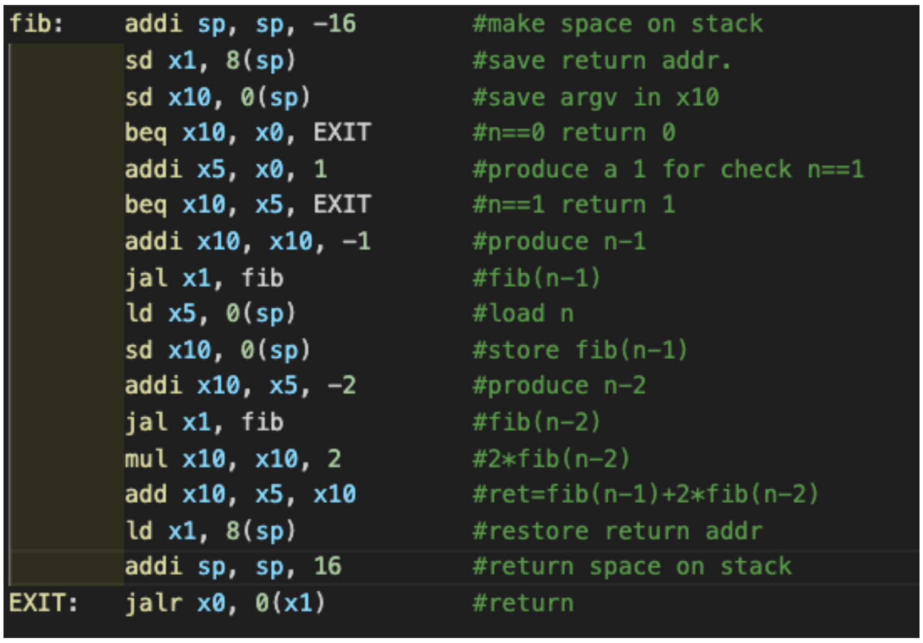
hex representation is 0x46D3823

6. Translate the following RISC-V code into C code. Assume that the C-code integer result is held in register x5, x6 holds the C-code integer i, and x10 holds the base address of the integer array MemArray.



```
int i = 100; //addi x6, x0, 100
for(;i>0;i-=4){ //bgt x6, x29, loop; addi, x6, x6 ,-4
  result += *MemArray; //addi, x5, x5, x7
  MemArray++; //addi, x10, x10, 8
}
```

7. Implement the following C code in RISC-V assembly. Note: According to RISC-V spec, "In the standard RISC-V calling convention, the stack grows downward and the stack pointer is always kept 16-byte aligned.”.



```
fib:    addi sp, sp, -16          #make space on stack
        sd x1, 8(sp)              #save return addr.
        sd x10, 0(sp)             #save argv in x10
        beq x10, x0, EXIT         #n==0 return 0
        addi x5, x0, 1            #produce a 1 for check n==1
        beq x10, x5, EXIT         #n==1 return 1
        addi x10, x10, -1         #produce n-1
        jal x1, fib               #fib(n-1)
        ld x5, 0(sp)              #load n
        sd x10, 0(sp)             #store fib(n-1)
        addi x10, x5, -2          #produce n-2
        jal x1, fib               #fib(n-2)
        mul x10, x10, 2           #2*fib(n-2)
        add x10, x5, x10          #ret=fib(n-1)+2*fib(n-2)
        ld x1, 8(sp)              #restore return addr
        addi sp, sp, 16           #return space on stack
EXIT:   jalr x0, 0(x1)            #return
```

8. We would like to expand the RISC-V register file to 128 registers and expand the instruction set to contain four times as many instructions.

a). How would this affect the size of each of the bit fields in the R-type instructions?

rs1, rs2 , rd should be 7 bits since it can produce #0~127 register.
Opcode would expand 9 bits (7 bits original and expand 2 more bits)

b). How could each of the two proposed changes along decrease the size of a RISC-V assembly program? On the other hand, how could the two proposed changes together increase the size of an RISC-V assembly program?

Increasing the size of field might lead to a longer instruction, which causes the code size become larger. Increasing number of registers would reduce register reuse, decreasing load/store instruction use, thus decreasing the code size.