

計算機結構 作業二

學號：111062107

姓名：鄧弘利

1.

(a)

Subroutine	Starting memory address	Ending memory address	Reference
fast_power_recur	00000000000000b0	00000000000000e8	jal ra,0xb0 <fast_power_recur>
fast_power_iter	00000000000000b0	00000000000000cc	jal ra,0xb0 <fast_power_iter>

```

00000000000000b0: fast_power_recur:
7      c.bnez a1,0xb8 <fast_power_recur+8>
00000000000000b2:      return 1;
16     c.li a0,1
00000000000000b4:      }
4      ret
00000000000000b8:      {
00000000000000ba:      c.addi sp,-32
00000000000000bc:      c.sdsp ra,24(sp)
00000000000000be:      c.sdsp s0,16(sp)
00000000000000c0:      c.sdsp s1,8(sp)
00000000000000c2:      c.mv s0,a1
10     c.mv s1,a0
00000000000000c4:      int m = fast_power_recur(x, pow >> 1);
00000000000000c8:      srailw a1,a1,0x1
11     jal ra,0xb0 <fast_power_recur>
00000000000000cc:      if(pow & 1)
15     bbs s0,0,0xe0 <fast_power_recur+48>
00000000000000d0:      return m * m;
16     mulw a0,a0,a0
00000000000000d4:      }
00000000000000d6:      c.ldsp ra,24(sp)
00000000000000d8:      c.ldsp s0,16(sp)
00000000000000da:      c.ldsp s1,8(sp)
00000000000000dc:      c.addi16sp sp,32
13     ret
00000000000000e0:      return x * m * m;
00000000000000e4:      mulw s1,s1,a0
00000000000000e8:      mulw a0,s1,a0
19     j 0xd4 <fast_power_recur+36>
{
20
00000000000000f0:      int power = fast_power_recur(6, 11);
00000000000000f2:      c.li a1,11
00000000000000f4:      c.li a0,6
      jal ra,0xb0 <fast_power_recur>

```

```

fast_power_iter:
c.mv    a5,a0
while(pow)
c.li    a0,1
j       0xc0 <fast_power_iter+16>
        x = x * x;
mulw    a5,a5,a5
        pow >>= 1;
sraiw   a1,a1,0x1
while(pow)
c.beqz   a1,0xcc <fast_power_iter+28>
        if(pow & 1)
bbc     a1,0,0xb8 <fast_power_iter+8>
        res = res * x;
mulw    a0,a5,a0
c.j      0xb8 <fast_power_iter+8>
        return res;
ret
{
main:

```

```

main:
c.addi   sp,-16
c.sdsp   ra,8(sp)
int power = fast_power_iter(6, 11);
c.li     a1,11
c.li     a0,6
jal      ra,0xb0 <fast_power_iter>

```

(b)

Code memory address	Instruction	Saved register	Stack offset
00000000000000be	c.sdsp s1,8(sp)	s1	8
00000000000000bc	c.sdsp s0,16(sp)	s0	16
00000000000000ba	c.sdsp ra,24(sp)	ra	24

```

16
00000000000000b4: } ret
4
00000000000000b8: { c.addi sp,-32
00000000000000ba: c.sdsp ra,24(sp)
00000000000000bc: c.sdsp s0,16(sp)
00000000000000be: c.sdsp s1,8(sp)
00000000000000c0: c.mv s0,a1
00000000000000c2: c.mv s1,a0
10 int m = fast_power_recur(x, pow >> 1);

```

(c)

Function	Parameter x		Parameter pow		Return value	
	Register	Value	Register	Value	Register	Value
fast_power_iter	a0	6	a1	11	a0	362797056
fast_power_recur	a0	6	a1	11	a0	362797056

iter before

The screenshot shows the GDB debugger interface. The console window displays the assembly code for the `fast_power_recur` function. The registers panel shows the values of the registers `a0` and `a1`. The `a0` register contains the value 6, and the `a1` register contains the value 11. The `main` function is also visible in the console window.

iter after

The screenshot shows the GDB debugger interface after the `fast_power_iter` function call. The console window displays the assembly code for the `fast_power_iter` function. The registers panel shows the updated values of the registers `a0` and `a1`. The `a0` register now contains the value 362797056, and the `a1` register contains the value 11. The `main` function is also visible in the console window.

recur before

fast_power_recur.adx [Application Program]

Thread #1 (Suspended: Breakpoint)

main() at fast_power_recur.c:20 0x140

gdb(12.0.50.20220216)-2

```
5 if(pow == 0)
6 {
7     return 1;
8 }
9
10 int m = fast_power_recur(x, pow >> 1);
11 if(pow & 1)
12 {
13     return x * m * m;
14 }
15 return m * m;
16 }
17
18 int main(void)
19 {
20     int power = fast_power_recur(6, 11);
21     printf("%d", power);
22     return 0;
23 }
24
```

Console

```
000000000000013c: c.li a1,11
000000000000013e: c.li a0,6
0000000000000140: jal ra,0x0000000000000144 <fast_power_recur>
0000000000000144: c.mv a5,a0
0000000000000146: sw a5,-20(s0)
0000000000000148: printf("%d", power);
000000000000014a: lw a5,-20(s0)
000000000000014c: c.mv a1,a5
000000000000014e: addiip a0,-2472
0000000000000150: jal ra,0x164 <printf>
```

Variables

Expression	Type	Value
GRP(General Purpose Registers).REG(ra)	Unsigned / Readable,Writeable	0x0000000000000000
GRP(General Purpose Registers).REG(fp)	Unsigned / Readable,Writeable	58331648 (Decimal)
GRP(General Purpose Registers).REG(s1)	Unsigned / Readable,Writeable	0 (Decimal)
GRP(General Purpose Registers).REG(a0)	Unsigned / Readable,Writeable	6 (Decimal)
GRP(General Purpose Registers).REG(a1)	Unsigned / Readable,Writeable	11 (Decimal)

Registers

Name	Value	Description
All Registers		
General Purpose Registers		
Floating Point Registers		
RISC-V standard machine mode CSRs		
AndeStar V5 machine mode CSRs		
AndeStar V5 debug mode CSRs		
RISC-V standard user mode CSRs		
AndeStar V5 user mode CSRs		

Breakpoints Modules Debugger Console Static Stack Analysis Function Code Size

recur after

fast_power_recur.adx [Application Program]

Thread #1 (Suspended: Breakpoint)

fast_power_recur() at fast_power_recur.c:16 0x130

main() at fast_power_recur.c:20 0x144

gdb(12.0.50.20220216)-11

```
5 if(pow == 0)
6 {
7     return 1;
8 }
9
10 int m = fast_power_recur(x, pow >> 1);
11 if(pow & 1)
12 {
13     return x * m * m;
14 }
15 return m * m;
16 }
17
18 int main(void)
19 {
20     int power = fast_power_recur(6, 11);
21     printf("%d", power);
22     return 0;
23 }
24
```

Console

```
000000000000012c: c.lidp s0,32(sp)
000000000000012e: c.addi16sp sp,40
0000000000000130: ret
0000000000000132:
0000000000000134: c.addi sp,-32
0000000000000136: c.sdsp ra,24(sp)
0000000000000138: c.sdsp s0,16(sp)
000000000000013a: c.addi4spn s0,sp,32
000000000000013c: int power = fast_power_recur(6, 11);
000000000000013e: c.li a0,6
0000000000000140: jal ra,0x0000000000000144 <fast_power_recur>
0000000000000144: c.mv a5,a0
0000000000000146: sw a5,-20(s0)
0000000000000148: printf("%d", power);
000000000000014a: lw a5,-20(s0)
000000000000014c: c.mv a1,a5
000000000000014e: addiip a0,-2472
```

Variables

Expression	Type	Value
GRP(General Purpose Registers).REG(ra)	Unsigned / Readable,Writeable	0x0000000000000144
GRP(General Purpose Registers).REG(fp)	Unsigned / Readable,Writeable	58331648 (Decimal)
GRP(General Purpose Registers).REG(s1)	Unsigned / Readable,Writeable	0 (Decimal)
GRP(General Purpose Registers).REG(a0)	Unsigned / Readable,Writeable	362797956 (Decimal)
GRP(General Purpose Registers).REG(a1)	Unsigned / Readable,Writeable	0 (Decimal)

Registers

Name	Value	Description
All Registers		
General Purpose Registers		
Floating Point Registers		
RISC-V standard machine mode CSRs		
AndeStar V5 machine mode CSRs		
AndeStar V5 debug mode CSRs		
RISC-V standard user mode CSRs		
AndeStar V5 user mode CSRs		

Breakpoints Modules Debugger Console Static Stack Analysis Function Code Size

address: 0x00000130

address: 0x00000144

No details to display for the current selection.

(d)

<pre> 6 F000000000000000b2: F000000000000000b4: 12 F000000000000000b8: 13 F000000000000000bc: 6 F000000000000000c0: 8 F000000000000000c2: 10 F000000000000000c6: F000000000000000ca: 15 F000000000000000cc: 19 </pre>	<pre> while(pow) c.li a0,1 j 0xc0 <fast_power_iter+16> x = x * x; mulw a5,a5,a5 pow >>= 1; sraiw a1,a1,0x1 while(pow) c.beqz a1,0xcc <fast_power_iter+28> if(pow & 1) bbc a1,0,0xb8 <fast_power_iter+8> res = res * x; mulw a0,a5,a0 c.j 0xb8 <fast_power_iter+8> return res; ret { main: </pre>
---	--

My breakpoints

Simulator Performance Meter					
Mode	InsC	CycC	I\$Miss	D\$Miss	Branch Misprediction
D	113	174	0	0	8
D	7	9	0	0	0
D	1	1	0	0	0
D	1	4	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	4	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	4	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	1	0	0	0
D	1	4	0	0	0
D	714	1,384	0	0	12

After calculating the cycle counts for CoreA,B,C, I find that

CoreA has total of 4 cycle counts , CoreB has 4 cycle counts , and CoreC has 24 cycle counts.The total cycle counts is 32.

So if we use pipeline , the total cycle counts is 24. And since $32/24 < 2$,so it's impossible to achieve speedup of 2.

2.

(a)

Instruction ID	Hexadecimal Encoded instruction	Decoded instruction and brief explanation
5	0x0040 8067	<p>jalr x0, 4(x1) $PC = x1 + 4$ Program counter 設為 x1 儲存的位置加 4 的地方</p> <p>000000000100 00001 000 00000 1100111</p> <p>From the last 7 digits , “1100111” , we can know that this is jalr instruction. (I-format)</p> <p>From the rs1 field , “00001” , we can know that it is x1 register.</p> <p>From the beginning 12 bits , “000000000100” , we can know it means 4.</p> <p>From rd field , “00000” ,we can know it’s x0 register.</p>
6	0xFF84 3283	<p>ld x5, -8(x8) x5 register = Mem[x8-8] (from the memory address x8-8 load double word to x5) 將 x8 儲存的位置-8 之後，得到一個新的位置，並將在 memory 該位置儲存的值，存到 x5</p> <p>111111111000 01000 011 00101 0000011</p> <p>From opcode field and funct3 field, “0000011” and “011”, we can know it’s ld instruction. (I-format)</p> <p>From the rs1 field , “01000” , we can know that it is x8 register.</p> <p>From the beginning 12 bits , “111111111000” , we can know it means -8.</p> <p>From rd field , “00101” ,we can know it’s x5 register.</p>
8	0x4142 D293	<p>srai x5, x5, 20 x5 = x5 register shift right for 20 bits. 將 x5 的值 arithmetic shift right 20 位後存回 x5</p>

		<p>010000010100 00101 101 00101 0010011</p> <p>From the imm,funct3 and opcode fields, “0100000”, “101” and ” 0010011”,we can know it’s srai instruction.</p> <p>From the rest imm field, “10100” , we can know it means 20.</p> <p>From the rs1 field , “00101” , we can know it’s about x5 register.</p> <p>From the rd field,” 00101” ,we can know it means x5 register.</p>
--	--	---

(b)

Instruction ID	Updated register	Updated memory
1	$x7 \leftarrow x5 - x6 = 0x0000\ 0000\ 0000\ 0004$	
2		MEM[0x0000 003E FF20 13F0] $\leftarrow 0xFF20\ 13E0$ MEM[0x0000 003E FF20 13F4] $\leftarrow 0x0000\ 003E$
3	$x1 \leftarrow PC+4 = 0x0000\ 0000\ 0001\ 00BC$	
6	$x5 \leftarrow \{MEM[0x0000\ 003E\ FF20\ 13FC], MEM[0x0000\ 003E\ FF20\ 13F8]\}$ $= 0x0A0B\ 0130\ 0041\ 0000$	
7	$x5 \leftarrow x30 \& x5 = 0x0000\ 0000\ 00F0\ 0000$ $\& 0x0A0B\ 0130\ 0041\ 0000 =$ $0x0000\ 0000\ 0040\ 0000$	
8	$x5 \leftarrow x5 \gg 20 = 0x0000\ 0000\ 0040\ 0000$ $0000 \gg 20 = 0x0000\ 0000\ 0000\ 0004$	
9		
10	$x28 \leftarrow x2 = 0x0000\ 003E\ FF20\ 13E0$	
11	$x7 \leftarrow 0xFFFF\ FFFF\ FFFF\ FF91$	
12		
13	$x1 \leftarrow 0x0000\ 0000\ 0001\ 00E4$	
4	$x6 \leftarrow MEM[0x0000\ 003E\ FF20\ 13E4]$ $= 0xFFFF\ FFFF\ A800\ 3F10$	
5		

15	$x7 \leftarrow x6 \oplus x7 = 0x0000\ 0000\ 57FF\ C081$	
16	$x7 \leftarrow x7 \gg 16 = 0x0000\ 0000\ 57FF\ C081 \gg 16 = 0x0000\ 0000\ 0000\ 57FF$	
17	$x31 \leftarrow x6 + 1000 = 0xFFFF\ FFFF\ A800\ 42F8$	
18	$x31 \leftarrow x31 \gg 16 = 0xFFFF\ FFFF\ A800\ 42F8 \gg 16 = 0xFFFF\ FFFF\ FFFF\ A800$	
19		MEM[0x0000 003E FF20 1400 - 8] = MEM[0x0000 003E FF20 13F8] = 0x0041 0004
20		MEM[0x0000 003E FF20 1400 - 24] = MEM[0x0000 003E FF20 13E8] = 0xFFFF A800 MEM[0x0000 003E FF20 13EC] = 0xFFFF FFFF

(c)

2, 6, 11, 19, 4, 20 instructions and 19 executed code addresses(PC).

So total number of memory accesses performed throughout the code is 25.

(d)

Code address	Assembly instruction	Hexadecimal encoded instruction	Taken?
0x0000 0000 0001 0100	blt x31, x7, BEGIN	0XFC7F C2E3	Yes

3.

```
1  i = 0; //addi x10, x0, 0
2  int *temp = MemArray; //addi x28, x13, 0
3  //LOOPI:
4  while(m < i) { //bge x10, x3, ENDI
5      j = 0; //addi x11, x0, 0
6      total = 0; //addi x12, x0, 0
7      int value = *temp; //lw x29, 0(x28)
8      //addi x30, x0, 32
9      //LOOPJ:
10     while(32 < j) { //bge x11, x30, ENDJ
11         //srl x31, x29, x11 ((value >> j))
12         //andi x31, x31, 1 ((value >> j) & 1)
13         total += (value >> j) & 1; //add x12, x12, x31
14         j++; //addi x11, x11, 1
15     } //jal x0, LOOPJ
16     //ENDJ:
17     *temp = total; //sw x12, 0(x28)
18     i++; //addi x10, x10, 1
19     temp++; //addi x28, x28, 4
20 } //jal x0, LOOPI
21 //ENDI:
```

4.

(a)

slli x28, x7, 2 # $x28 = i * 4$

addi x28, x28, 1 # $x28 = i * 4 + 1$

slli x28, x28, 2 # $x28 = (i * 4 + 1) * 4$

add x29, x5, x28 # $x29 = \text{address of } A[i * 4 + 1]$

lw x29, 0(x29) # $x29 = A[i * 4 + 1]$

slli x29, x29, 2 # $x29 = A[i * 4 + 1] * 4$

add x29, x6, x29 # $x29 = \text{address of } B[A[i * 4 + 1]]$

lw x29, 0(x29) # $x29 = B[A[i * 4 + 1]]$

slli x30, x7, 2 # $x30 = i * 4$

add x30, x6, x30 # $x30 = \text{address of } B[i]$

lw x30, 0(x30) # $x30 = B[i]$

add x11, x29, x30 # $j = x29 + x30 = B[A[i * 4 + 1]] + B[i]$

(b)

slli x28, x7, 2 # $x28 = i * 4$

addi x28, x28, 1 # $x28 = i * 4 + 1$

slli x28, x28, 3 # $x28 = (i * 4 + 1) * 8$

add x29, x5, x28 # $x29 = \text{address of } A[i * 4 + 1]$

lw x29, 0(x29) # $x29 = A[i * 4 + 1]$

slli x29, x29, 3 # $x29 = A[i * 4 + 1] * 8$

add x29, x6, x29 # $x29 = \text{address of } B[A[i * 4 + 1]]$

lw x29, 0(x29) # $x29 = B[A[i * 4 + 1]]$

slli x30, x7, 3 # $x30 = i * 8$

add x30, x6, x30 # $x30 = \text{address of } B[i]$

lw x30, 0(x30) # $x30 = B[i]$

add x11, x29, x30 # $j = x29 + x30 = B[A[i * 4 + 1]] + B[i]$

5.

```
12 Func:
13     addi sp,sp,-32      # make space on stack
14     sd   ra,16(sp)     # save return address in x1 onto stack
15     sd   a0,0(sp)      # save argument in x10 onto stack
16     beq  a0,zero,Exit  # if n == 0 ,go to Exit
17     addi t0,a0,1       # t0 = 1
18     andi t0,a0,t0      # t0 = n & t0 = n & 1
19     beq  t0,zero,Else  # if (n & 1) == 0,go to Else
20     srli a0,a0,1       # n = n >> 1
21     jal  ra,Func       # call Func(n >> 1)
22 Else
23     srli a0,a0,1       # n = n >> 1
24     jal  ra,Func       # call Func(n >> 1)
25
26     addi t1,a0,0       # move return value of Func(n >> 1) to t1
27     ld   a0,0(sp)      # restore caller's n
28     ld   ra,16(sp)     # restore return address
29     addi sp,sp,32      # return space on stack
30     beq  a0,zero,Exit  # if n == 0 ,go to Exit
31     addi a0,a0,t1      # return n + Func(n >> 1)
32 Exit:
33     jalr zero, 0(ra)   # return
```