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>

```
                    fast_power_recur:
00000000000000b0:     c.bnez  a1,0xb8 <fast_power_recur+8>
 7                          return 1;
00000000000000b2:     c.li    a0,1
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);
00000000000000c4:     sraiw   a1,a1,0x1
00000000000000c8:     jal     ra,0xb0 <fast_power_recur>
11                      if(pow & 1)
00000000000000cc:     bbs     s0,0,0xe0 <fast_power_recur+48>
15                      return m * m;
00000000000000d0:     mulw    a0,a0,a0
16                  }
00000000000000d4:     c.ldsp  ra,24(sp)
00000000000000d6:     c.ldsp  s0,16(sp)
00000000000000d8:     c.ldsp  s1,8(sp)
00000000000000da:     c.addi16sp      sp,32
00000000000000dc:     ret
13                          return x * m * m;
00000000000000e0:     mulw    s1,s1,a0
00000000000000e4:     mulw    a0,s1,a0
00000000000000e8:     j       0xd4 <fast_power_recur+36>
19                  {
```

```
20                      int power = fast_power_recur(6, 11);
00000000000000f0:     c.li    a1,11
00000000000000f2:     c.li    a0,6
00000000000000f4:     jal     ra,0xb0 <fast_power_recur>
```

```
                    fast_power_iter:
00000000000000b0:     c.mv     a5,a0
 6                      while(pow)
00000000000000b2:     c.li     a0,1
00000000000000b4:     j        0xc0 <fast_power_iter+16>
12                          x = x * x;
00000000000000b8:     mulw     a5,a5,a5
13                          pow >>= 1;
00000000000000bc:     sraiw    a1,a1,0x1
 6                      while(pow)
00000000000000c0:     c.beqz   a1,0xcc <fast_power_iter+28>
 8                          if(pow & 1)
00000000000000c2:     bbc      a1,0,0xb8 <fast_power_iter+8>
10                              res = res * x;
00000000000000c6:     mulw     a0,a5,a0
00000000000000ca:     c.j      0xb8 <fast_power_iter+8>
15                      return res;
00000000000000cc:     ret
19                  {
                    main:
```

```
                    main:
00000000000000d0:     c.addi   sp,-16
00000000000000d2:     c.sdsp   ra,8(sp)
20                      int power = fast_power_iter(6, 11);
00000000000000d4:     c.li     a1,11
00000000000000d6:     c.li     a0,6
00000000000000d8:     jal      ra,0xb0 <fast_power_iter>
```

(b)

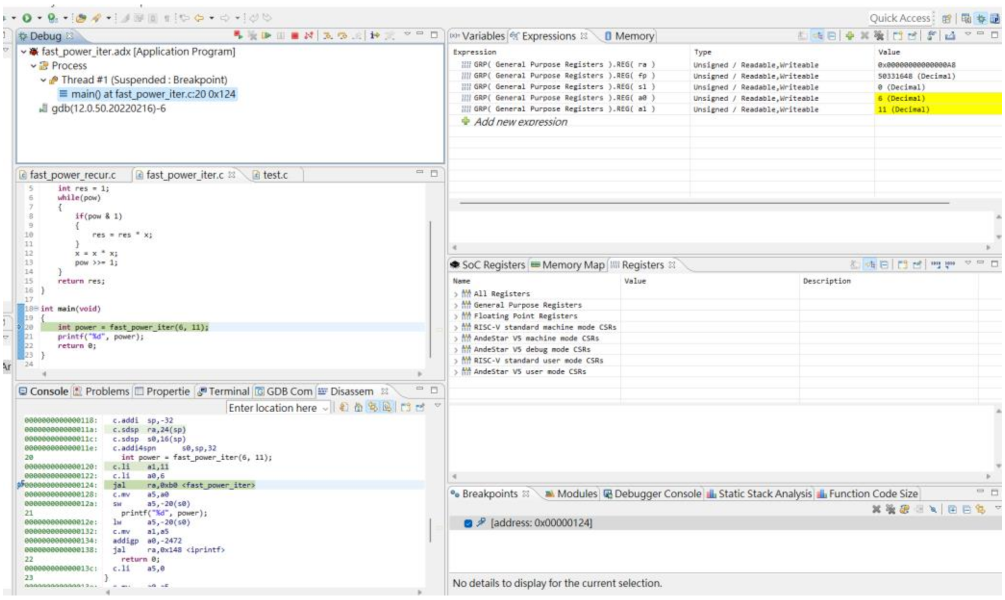
Code memory address	Instruction	Saved register	Stack offset
00000000000000be	c.sdsp s1,8(sp)	s1	8
00000000000000bc	c.sdsp s0,16(sp)	s0	16
00000000000000ba	c.sdsp ra,24(sp)	ra	24

```
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
```

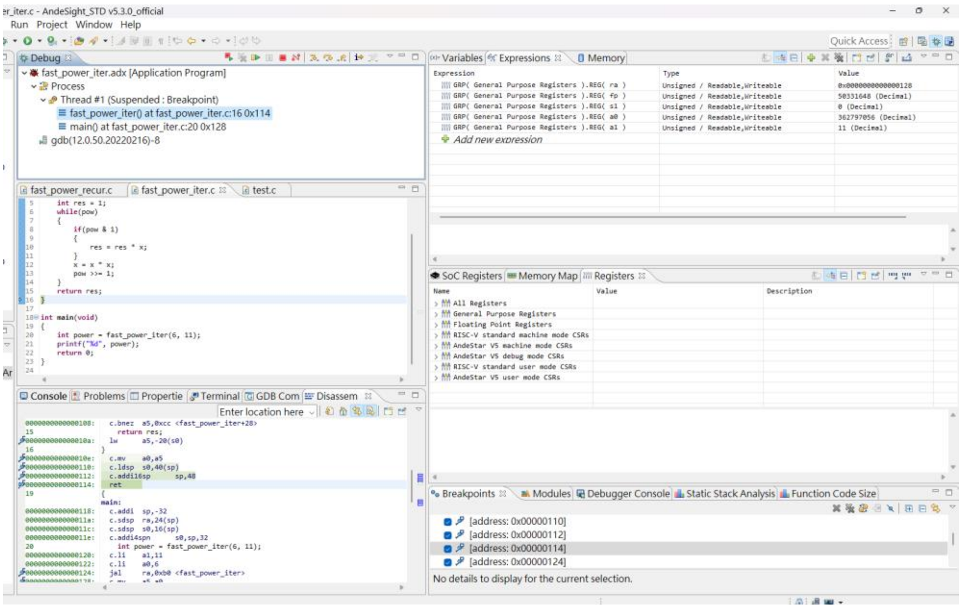
(c)

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

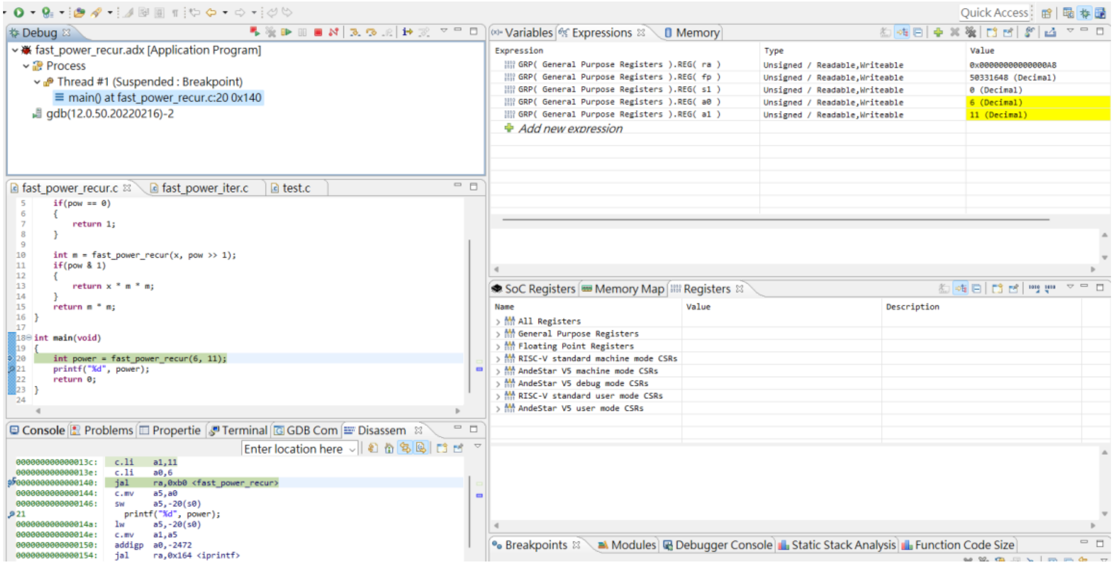
iter before



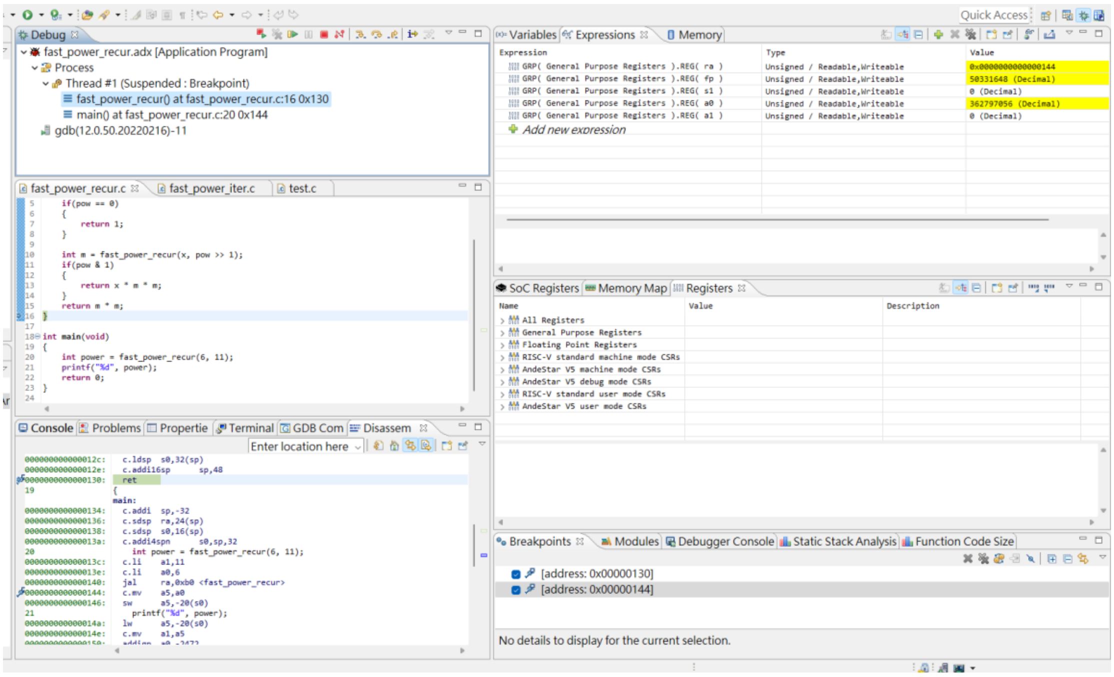
iter after



recur before



recur after



(d)

```
6            while(pow)
00000000000000b2:   c.li    a0,1
00000000000000b4:   j       0xc0 <fast_power_iter+16>
12                    x = x * x;
00000000000000b8:   mulw    a5,a5,a5
13                    pow >>= 1;
00000000000000bc:   sraiw   a1,a1,0x1
6            while(pow)
00000000000000c0:   c.beqz  a1,0xcc <fast_power_iter+28>
8                     if(pow & 1)
00000000000000c2:   bbc     a1,0,0xb8 <fast_power_iter+8>
10                      res = res * x;
00000000000000c6:   mulw    a0,a5,a0
00000000000000ca:   c.j     0xb8 <fast_power_iter+8>
15           return res;
00000000000000cc:   ret
19         {
           main:
```

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	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	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	jalr x0, 4(x1) PC = x1 + 4 Program counter 設為 x1 儲存的位置加 4 的地方 000000000100 00001 000 00000 1100111 From the last 7 digits , “1100111” , we can know that this is jalr instruction. (I-format) From the rs1 field , “00001” , we can know that it is x1 register. From the beginning 12 bits , “000000000100” , we can know it means 4. From rd field , “00000” ,we can know it’s x0 register.
6	0xFF84 3283	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 111111111000 01000 011 00101 0000011 From opcode field and funct3 field, “0000011” and “011”, we can know it’s ld instruction. (I-format) From the rs1 field , “01000” , we can know that it is x8 register. From the beginning 12 bits , “111111111000” , we can know it means -8. From rd field , “00101” ,we can know it’s x5 register.
8	0x4142 D293	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

		010000010100 00101 101 00101 0010011 From the imm,funct3 and opcode fields, "0100000" , "101" and " 0010011",we can know it's srai instruction. From the rest imm field, "10100" , we can know it means 20. From the rs1 field , "00101" , we can know it's about x5 register. From the rd field," 00101" ,we can know it means x5 register.

(b)

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

15	x7 $\leftarrow$ x6 $\oplus$ x7 = 0x0000 0000 57FF C081	
16	x7 $\leftarrow$ x7 >> 16 = 0x0000 0000 57FF C081 >> 16 = 0x0000 0000 0000 57FF	
17	x31 $\leftarrow$ x6+1000 = 0xFFFF FFFF A800 42F8	
18	x31 $\leftarrow$ x31 >> 16 = 0xFFFF FFFF A800 42F8 >> 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.

```
i = 0; //addi x10, x0, 0
int *temp = MemArray; //addi x28, x13, 0
//LOOPI:
while(m < i) { //bge x10, x3, ENDI
    j = 0; //addi x11, x0, 0
    total = 0; //addi x12, x0, 0
    int value = *temp; //lw x29, 0(x28)
    //addi x30, x0, 32
    //LOOPJ:
    while(32 < j) { //bge x11, x30, ENDJ
        //srl x31, x29, x11 ((value >> j))
        //andi x31, x31, 1 ((value >> j) & 1)
        total += (value >> j) & 1; //add x12, x12, x31
        j++; //addi x11, x11, 1
    } //jal x0, LOOPJ
    //ENDJ:
    *temp = total; //sw x12, 0(x28)
    i++; //addi x10, x10, 1
    temp++; //addi x28, x28, 4
} //jal x0, LOOPI
//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 = 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 = 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 = 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 = 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 = 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 = 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.

```
Func:
    addi sp,sp,-32      # make space on stack
    sd   ra,16(sp)      # save return address in x1 onto stack
    sd   a0,0(sp)       # save argument in x10 onto stack
    beq  a0,zero,Exit   # if n == 0 ,go to Exit
    addi t0,a0,1        # t0 = 1
    andi t0,a0,t0       # t0 = n & t0 = n & 1
    beq  t0,zero,Else   # if  (n & 1) == 0,go to Else
    srli a0,a0,1        # n = n >> 1
    jal  ra,Func        # call Func(n >> 1)
Else
    srli a0,a0,1        # n = n >> 1
    jal  ra,Func        # call Func(n >> 1)

    addi t1,a0,0        # move return value of Func(n >> 1) to t1
    ld   a0,0(sp)       # restore caller's n
    ld   ra,16(sp)      # restore return address
    addi sp,sp,32       # return space on stack
    beq  a0,zero,Exit     # if n == 0 ,go to Exit
    addi a0,a0,t1       # return n + Func(n >> 1)
Exit:
    jalr zero, 0(ra)      # return
```