NATIONAL TSING HUA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE
CS 4100: Computer Architecture
Spring 2023, Mid-term Examination

1. (22%) Consider a specific computer called HAL running a program of 10 million instructions at a clock rate of 400 MHz. Among the instructions, 20% of them are INT instructions, 40% are FP instructions, 20% are load/store instructions, and the rest are branch instructions. Suppose that an INT instruction takes 1 cycle; an FP instruction takes 10 cycles; a load/store instruction takes 2 cycles; a branch instruction takes 3 cycles to execute.
   (a) (5%) What is the average CPI of this program running on HAL?
   (b) (7%) From (a), the architect wants to reduce the total time by improving the cycles of an FP instruction. Let n be the largest possible number of cycles for an FP instruction to reduce the total time by at least 50%, and n is an integer. What will n be?
   (c) (10%) From (a), to increase the cycle rate of HAL to 800 MHz, each branch instruction becomes 5 cycles. And 50% of the FP instructions must be executed in 14 cycles, whereas the rest remains the same. What is the overall speedup (i.e., the ratio of the enhanced performance over the original performance) after the enhancement?

2. (8%) The architect is evaluating two alternative processor implementations, P1 and P2, by using a benchmark set of two benchmark programs, B1 and B2. P1 operates at the clock rate of 800 MHz; P2 operates at 1.2 GHz. The execution cycles of the two benchmarks are summarized as follows:

Benchmark	Execution Cycles on P1	Execution Cycles on P2
B1	5M	12M
B2	10M	6M

   (a) (6%) What is the performance ratio of P1:P2, considering both benchmarks using geometric mean?
   (b) (2%) From (a), which processor implementation is faster? You must give the reason.

3. (25%) Modern computers are designed based on the stored-program concept, based on which programs are stored in and executed from the main memory, where data are also stored. One consequence is that program instructions and data may be manipulated in the same way. A good example to illustrate the idea is self-modifying code, which alters its own instructions while it is executing. Although most computers do not allow you to do it for robustness reasons, we assume here that you can do it, and the effect shows immediately. Consider the C statement shown below left. Assume that **a** and **b** have been loaded into registers **x28** and **x29**, respectively. One possible implementation of the C statement with self-modifying code is shown below right. Note that the instruction encodings of add and sub in RISC-V differ by only one bit:

add	`0000000 rs2 rs1 000 rd 0110011`
sub	`0100000 rs2 rs1 000 rd 0110011`

```
if a < b
   a = a - b;
else
   a = a + b;
```

`slt`	`x6,x28,x29`	x6 ← 1 if a < b else x6 ← 0
`slli`	`x6,x6,30`	
`auipc`	`x7,0`	
`lw`	`x30,16(x7)`	
`or`	`x30,x30,x6`	
`sw`	`x30,16(x7)`	

1. HAL: 10^7 instructions, clock rate: 400 MHz

20% INT (1cyc), 40% FP (10cyc), 20% load/store (2cyc), 20% branch (3cyc)

(a) Average CPI = $0.2\times1+0.4\times10+0.2\times2+0.2\times3=5.2$

(b) Improve FP. n 是至少 ↓50% time 的 max cycle 數

$\frac{5.2}{2}=2.6$, $5.2-4=1.2$, $2.6-1.2=1.4$, $0.4\times n\le1.4$

$n=3$ #

(c) rate → 800 MHz, branch → 5 cyc, FP → $\begin{cases}一半\ 14cyc\\ 一半\ 10cyc\end{cases}$

Avg CPI $=0.2\times1+0.2\times10+0.2\times14+0.2\times2+0.2\times5=6.4$

$$\text{Speedup}=\frac{5.2}{6.4\div2}=\frac{5.2}{3.2}=\frac{13}{8}$$

2. (a) $P1:P2=\sqrt{\frac{12}{1.2}\times\frac{6}{1.2}}:\sqrt{\frac{5}{0.8}\times\frac{10}{0.8}}=25\sqrt{2}:\frac{25\sqrt{2}}{4}=4:5$

(b) P2 is faster.

3. (a)(i) load 31~12 bit to x7, sign extend 63~32 bit, 11~0 bit = 0, add PC

x7 = PC

(ii) add x28, x28, x29; 0000000_11101_11100_000_11100_0110011

(7, 5, 5, 3, 5, 7)

(iii) if a<b, x30 [30]=1 other remain, if a≥b, x30 remains its value.

(sub ... add)

(iv) 0000000_11101_11100_000_11100_0110011

(7, 5, 5, 3, 5, 7)

1000 0004

```
add     x28,x28,x29
```

(a) (8%) Explain how the above RISC-V code implements the C statement:

(i) What is loaded into **x7** after executing **auipc**?

(ii) Where is **16(x7)** pointing to? What is loaded into **x30** after executing **lw**?

(iii) What is the effect of executing **or**, considering **a<b** and **a≥b**?

31 ~12

(iv) What instruction will be executed when the execution comes to **add**?

(b) (5%) Replace **auipc** with **lui** in the above RISC-V code, assuming that **add** is stored at memory location 0x0000 0000 1000 0004. This is an implementation that uses absolute addressing instead of PC-relative. Show all the instructions that are modified and your calculations of the addresses. You should not add other instructions.

(c) (7%) Note that the above RISC-V code does not use branch instructions, although the C statement is an if-then-else statement. Now, implement the C statement in RISC-V, particularly using the **blt** instruction to check the **a<b** condition. Show all the instructions. You may use labels, such as **EXIT**, to indicate locations of instructions.

(d) (5%) Give the encoding of the **blt** instruction according to your RISC-V code in (c).

blt	`imm[12|10:5] rs2 rs1 100 imm[4:1|11] 1100011`

less than

4. (10%) Translate the following C code into RISC-V assembly code. You can only use RV64I instructions and cannot use pseudoinstuctions. Note that 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." Also, the return address is stored in **x1** and the stack pointer is in **x2**.

```
long long int findGCD(long long int a, long long int b)
{
    if(a==b)
        return a;
    if(a>b)
        return findGCD(a-b,b);
    else
        return findGCD(a,b-a);
}
```

x10 x11

x10 8

x11 16

01100

x1 24

5. (10%) In class we have described a 64-bit ALU, which has two 64-bit data inputs $A = a_{63}a_{62}\cdots a_0$, $B = b_{63}b_{62}\cdots b_0$, and one 4-bit control input ALUop (composed of 1-bit Ainvert, 1-bit Bnegate, and 2-bit Operation from left to right). Now if you want to remove the slt (set-less-than) operation and change the specification of the ALU as follows, what will the new ALU look like for bit 0?

Operation	ALUop
A + B	1101
A − B	1001
A and B	1110
A or B	1100
A nor B	0010

12

+12

You can modify the following 1-bit ALU to show your answer.

01100

4321

0110

(b) auipc ... → lui x7, 0x0FFFF

(c)
```
     blt x28, x29, Sub
     add x28, x28, x29
     jal Exit
Sub: sub x28, x28, x29
Exit:
```

(d) rs1: 11100, rs2: 11101

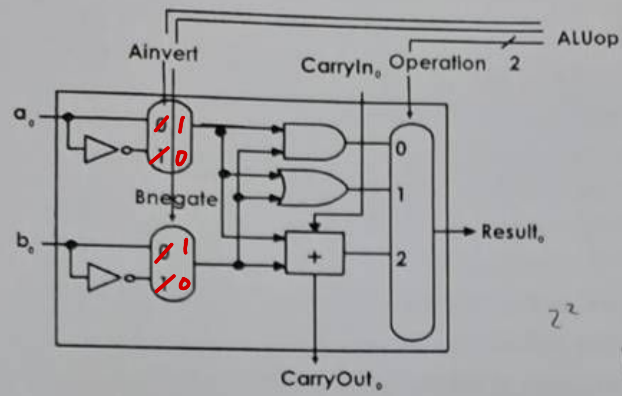
imm = 0 0000 0000 1100

imm[12, 10:5] = 0 00000

imm[4:1, 11] = 0110 0

4.

```
func: addi x2, -16
      sd   x1, 8(x2)
      bne  x10, x11, a<b
      addi x2, 16
      jalr x0, 0(x1)
a<b:  bge  x10, x11, a>b
      sub  x10, x10, x11
      jal  x1, func
      ld   x1, 8(x2)
      addi x2, 16
      jalr x0, 0(x1)
a>b:  sub  x11, x11, x10
      jal  x1, func
      ld   x1, 8(x2)
      addi x2, 16
      jalr x0, 0(x1)
```



2^{24}

1

24個 0

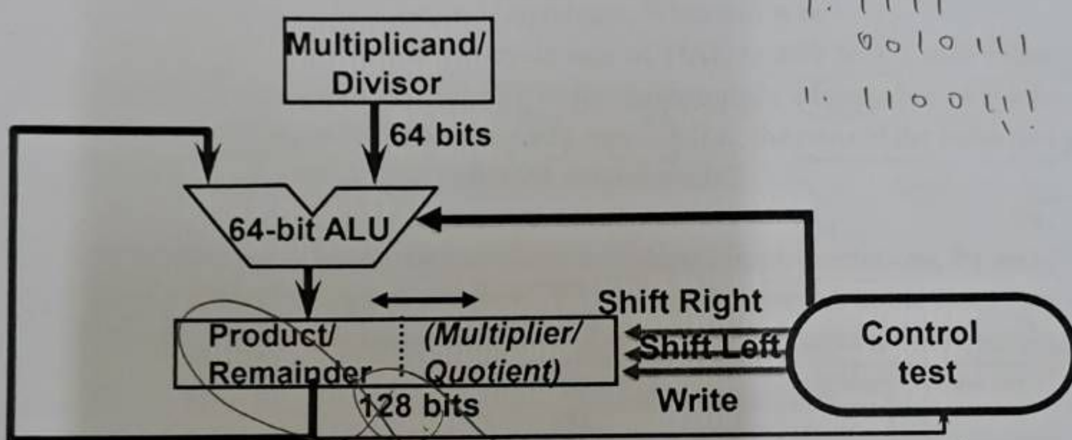
2^2

100

24個 1

6. (10%) The following hardware has been introduced in class for performing 64-bit unsigned integer multiplication and division.

1. 1111
0010111
1. 1100111



12 12 5 64 112 13

Let M=1100 and N=0101 be 4-bit unsigned integers. Perform each of the following operations using a 4-bit version of the hardware, respectively: 1000 0011

(i) $M \times N$ 0111 1101

(ii) $M \div N$ 1 +12+ 112

For each operation, you only need to show (1) the initial values of Multiplicand/Divisor register and Product/Remainder register, and (2) the updated value of Product/Remainder register at the end of each iteration. Besides, you also need to show the final value of Product/Remainder register for (ii).

7. (15%) Consider the IEEE 754 floating-point standard and its arithmetic operations.

(a) (3%) Does 0x7F839C5A represent a single precision normalized number? Why?

(b) (3%) Suppose the largest single precision denormalized number represents the decimal number $(A - 2^{-23}) \times 2^{-126}$. What is A?

(c) (6%) Consider the following two single precision floating-point numbers:

B = 0011 1111 0011 1000 0000 0000 0000 0000

C = 1100 0000 1111 1000 0000 0000 0000 0000

Show all the work to perform $B + C$, and write the result in the single precision format.

(d) (3%) Let the decimal number, $(2^{24} - 1) \times 2^D$, denote the largest even integer that can be exactly represented by the single precision format. What is D?

1. 110001
0010111
1111001

1 2 4 8 16 32 64

1. 1111 112
0010111
110001

1. 111+ 112
0010111
1.110

6. $M = \overset{12}{1100}$, $N = \overset{5}{0101}$ unsigned

(i) $M \times N$

	iter.	Product	Mcand
init	0.	0000 0101	1100
	1.	0110 0010	1100
	2.	0011 0001	1100
	3.	0111 1000	1100
	4.	0011 1100	1100

$M \times N = 00111100_2 = 60_{10}$

(ii) $M \div N$

	iter.	Rem	Divisor
init	0.	0000 1100	0101
	1.	0011 0000	0101
	2.	0110 0000	0101
	3.	0010 0001	0101
	4.	0100 0010	0101
	5.	$\underbrace{0010}_{\text{Rem}}$ $\underbrace{0010}_{\text{Q}}$	0101

$M \div N = 0010_2 = 2_{10} \cdots 2_{10}$

7. (a) 0x 7F839C5A = 0_1111 0000_ 011 ...

normalized $\because$ exponent = 1111 0000 $\neq$ 0000 0000

(b) $(2^{23}-1)\times 2^{-23}\times 10^{-126} = (1-2^{-23})\times 10^{-126}$, A=1

(c) B = 0011 1111 0011 1000 0000 0000 0000 0000

C = 1100 0000 1111 1000 0000 0000 0000 0000

ExpB ⇒ $\overset{+3}{1}$00 0000 1

0. 001 0111 0000 ....

−1. 111 1000 0000 ....

−1. 110 0001 0000 ....

Exp ⇒ 100 0000 1 , frac ⇒ 110 0001 0000 0000 0000 0000

sign bit ⇒ 1

⇒ B+C = 1100 0000 1110 0001 0000 0000 0000 0000_2

(d) $(2^{24}-1)\times 2^{-23}\times 2^{127} = (2^{24}-1)\times 2^{104}$, D = 104

$1.\overset{1}{1}11\cdots$ (23)

1111 1111 → keep

1111 1110 = 254

254 − 127 = 127