CENG 3420 Midterm (2022 Spring)

Name: _____
ID: _____

Q0 (0 marks)

1. What is your last digit of your SID (0 is regarded as 10)? This value is defined as `NUM_1` in the whole question paper.
2. What is your last two digits of your SID (00 is regarded as 100)? This value is defined as `NUM_2` in the whole question paper.
3. What is your last three digits of your SID? This value is defined as `NUM_3` in the whole question paper.

Example: if your SID is 12345678, then `NUM_1` $= 8$, `NUM_2` $= 78$, `NUM_3` $= 678$.

Q1 (8%) Assume `a0=0xBA, a1=0xDC`. Find the value of `a2` after the following instructions, respectively. Please note that we treat `NUM_1%3+1` as a decimal value. The final result should be a hexadecimal value.

1.
```
xor a2, a0, a1
ori a2, a2, (NUM_1%3+1)
```

2.
```
slli a2, a0, (NUM_1%3+1)
andi a2, a2, 0x1E
```

A1 As shown below,

1. 0xBA = 0b10111010, 0xDC=0b11011100
   After the first instruction, a2 = 0b01100110 = 0x66
   - If `NUM_1%3+1` = 1, a2 = (0b01100110) or (0b00000001) = 0b01100111 = 0x67
   - If `NUM_1%3+1` = 2, a2 = (0b01100110) or (0b00000010) = 0b01100110 = 0x66
   - If `NUM_1%3+1` = 3, a2 = (0b01100110) or (0b00000011) = 0b01100111 = 0x67
2. 0x1E = 0b00011110
   - If `NUM_1%3+1` = 1, a2 = 0b01110100 after the first instruction.
     a2 = (0b01110100) and (0b00011110) = 0b00010100 = 0x14
   - If `NUM_1%3+1` = 2, a2 = 0b11101000 after the first instruction.
     a2 = (0b11101000) and (0b00011110) = 0b00001000 = 0x08
   - If `NUM_1%3+1` = 3, a2 = 0b11010000 after the first instruction.
     a2 = (0b11010000) and (0b00011110) = 0b00010000 = 0x10

Q2 (12%) Assume that the variables a, b, c, d, e, and f are assigned to registers `t0`, `t1`, `t2`, `t3`, `t4`, and `t5`, respectively. Given RISC-V assembly instructions:

```
addi t5, t5, 1
slli t2, t4, 2
bge t0, t2, label1
add t3, t1, t2
```

```
    jal x0, label2
label1:
    mul t3, t1, t2
label2:
    subi t3, t3, NUM_2
```

1. Translate the RISC-V assembly instructions above into the corresponding C statements. Please not that we treat `NUM_2` as a decimal value.
2. Branch instruction `bge` may cause a stall in pipeline. Please discuss the reason, and list potential solutions to avoid the delay.

A2 1.
```
f = f+1;
c = e * 4;
if (a >= c) {
d = b * c;
} else {
d = b + c;
}
d = d - NUM_2;
```

2. Reason: bge will cause control hazard.
Solution:
1. Stall (impacts CPI)
2. Move decision point as early in the pipeline as possible, thereby reducing the number of stall cycles
3. Delay decision (requires compiler support)
4. Branch Prediction.
Some students mention that there is data hazard between `slli` and `bge`, it's also reasonable.

Q3 (10%) Assume a 20 cm diameter wafer has a cost of `NUM_2`, contains `NUM_3` dies, and has 0.02 defects/cm^2.

1. Find the yield of this wafer.
2. Find the cost per die for this wafer.

A3 Following are sample answers when `NUM_2`=10 and `NUM_3`=100.

1.
$$\text{Die area} \approx \frac{\text{Wafer area}}{\text{Dies per wafer}} \tag{1}$$
According to Equation (1), Die area $\approx \frac{\pi \times (\frac{20}{2})^2}{100} \approx 3.14\text{cm}^2$
$$\text{Yield} = \frac{1}{(1 + \frac{\text{Defects per area} \times \text{Die area}}{2})^2} \tag{2}$$
According to Equation (2), Yield $= \frac{1}{(1+\frac{0.020 \times 3.14}{2})^2} = 94.0\%$.

2.
$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{yield}} \tag{3}$$
According to Equation (3), Cost per die $= \frac{10}{100 \times 0.94} = 0.106$.

Q4 (10%) Some RISC-V assembly instructions are shown below. Assume that the variables f, g are assigned to registers `x5, x6`, respectively. Assume that the base address of the arrays A and B are in registers `x10` and `x11`, respectively.

```
slli x30, x5, 2 // x30 = f*4
add x30, x10, x30 // x30 = &A[f]
slli x31, x6, 2 // x31 = g*4
add x31, x11, x31 // x31 = &B[g]
lw x5, 0(x30) // x5 = A[f]
addi x12, x30, 8
lw x30, 0(x12)
sub x30, x30, x5
sw x30, 0(x31)
```

1. What's the meaning of the last four instructions.
2. What is corresponding C statement?

A4

1.
```
addi x12, x30, 8 // x12 = &A[f]+8 (i.e.  &A[f+2])
lw x30, 0(x12) // x30 = A[f+2]
sub x30, x30, x5 // x30 = A[f+2] - A[f]
sw x30, 0(x31) // B[g] = x30 (i.e.  A[f+2] - A[f])
```
2. The corresponding C code is:
```
B[g]= A[f+2] - A[f]
```

Q5 (10%) IEEE 754-2008 contains a half precision that is only 16 bits wide. The leftmost bit is still the sign bit, the exponent is 5 bits wide and has a bias of 15, and the mantissa is 10 bits long. A hidden 1 is assumed. Write down the bit pattern to represent 4.36525×10^1

Q5

1. $4.35625 \times 10^1 = 43.5625$
2. 43.5625 to binary 101011.1010011100... which is $1.0101110100111... \times 2^5$
3. fraction 0.0101110100111..., 11-12 bits are "11", mantissa 0101110101
4. exponent $5 + 15 = 20$ is 10100, sign bit is 0
5. final result 0101000101110101

Q6 (15%) Consider a RISC-V code snippet, and assume that the code is running on a 5-stage RISC-V core. Assuming the design has not implemented any forwarding or hazard detection.

```
sub a2, a1, a3
and a12, a2, a5
or a13, a6, a2
add a14, a2, a2
sd a15, 100(a2)
```

1. Find all data dependences in this instructioin sequence.
2. The hazard conditions can classified into 4 different types:
   - 1a. EX/MEM.RegisterRd =ID/EX.RegisterRs1
   - 1b. EX/MEM.RegisterRd =ID/EX.RegisterRs2
   - 2a. MEM/WB.RegisterRd =ID/EX.RegisterRs1

- 2b. MEM/WB.RegisterRd =ID/EX.RegisterRs2

Classify the dependences in this sequence.

3. If both 1a type and 2a type happen to the same instruction, how can we resolve the data dependance issue? Please draw a pipeline dragram to illustrate your idea.

Q6 1. The last four instructions are all dependent on the result in register x2 of the first instruction.

2.
- the sub-and is a type 1a hazard: EX/MEM.RegisterRd =ID/EX.RegisterRs1 =x2
- The sub-or is a type 2b hazard: MEM/WB.RegisterRd =ID/EX.RegisterRs2 =x2
- he two dependences on sub-add are not hazards because the register file supplies the proper data during the ID stage of add.
- There is no data hazard between sub and sd because sd reads x2 the clock cycle after sub writes x2.

Q7 (10%) Consider two different designs for conditional branch instructions:

(1) CPU_A: First set the condition code by a comparison instruction, and then test the condition code to branch.

(2) CPU_B: Include the comparison process in the branch instruction.

In both CPUs, a conditional branch instruction takes `NUM_1`$\%2+2$ clock cycles, while all other instructions take `NUM_1`$\%2+1$ clock cycle. **For** CPU_A, branch instructions account for 20% of the executed instructions and since a comparison instruction is required before each branch instruction, the comparison instruction also accounts for 20%. **For** CPU_B, no comparison instruction is needed since the comparison process is already included in the branch instruction. And assume the clock cycle time of CPU_B is 1.25 times the clock cycle time of CPU_A, which is $clock\_cycle_B = 1.25 \times clock\_cycle_A$.

1. Which CPU is faster?
2. If CPU_B's clock cycle time is only 1.1 times that of CPUA, which CPU is faster?

A7 These are suggested solutions. Assume `NUM_1` $= 0$, then

1. The average CPI of CPU_A is calculated as :

$$CPI_A = 0.2 \times 2 + 0.8 \times 1 = 1.2$$

Then the CPU time on CPU_A is:

$$\begin{aligned} CPU\ time_A &= IC_A \times CPI_A \times clock\_cycle_A \\ &= 1.2 \times IC_A \times clock\_cycle_A \end{aligned}$$

For CPU_B, since there is no comparison instructions, then the instruction count of CPU_B is only 80% of that of CPU_A, that's $IC_B = 0.8 \times IC_A$. And also, the proportion of branch instructions on CPU_B is :

$$20\%/80\% = 25\%$$

Then the average CPI of CPU_B is calculated as :

$$CPI_B = 0.25 \times 2 + 0.75 \times 1 = 1.25$$

Then the CPU time on CPU_B is:

$$\begin{aligned} CPU\ time_B &= IC_B \times CPI_B \times clock\_cycle_B \\ &= 0.8 \times IC_A \times 1.25 \times (1.25 \times clock\_cycle_A) \quad = 1.25 \times IC_A \times clock\_cycle_A \end{aligned}$$

Obviously, CPU_A is faster since it has smaller CPU time.

2. If CPU_B's clock cycle time is only 1.1 times that of CPUA, then we have:

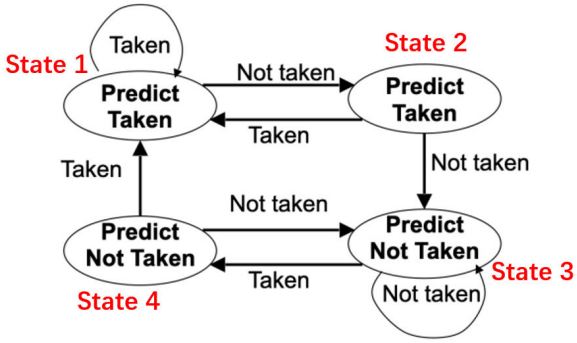
$$CPU\ time'_B = IC_B \times CPI_B \times clock\_cycle_B$$
$$= 0.8 \times IC_A \times 1.25 \times (1.1 \times clock\_cycle_A) \quad = 1.1 \times IC_A \times clock\_cycle_A$$

Under this condition, we can see that CPU_B is faster than CPU_A

Q8 (10%)

This exercise examines the accuracy of various branch predictors for the following repeating pattern (e.g., in a loop) of branch outcomes: NT, NT, T, T, NT. (T means 'Taken' and NT means 'Not taken')

1. What is the accuracy of always-taken and always-not-taken predictors for this sequence of branch outcomes?
2. What is the accuracy of the 2-bit predictor if this pattern is repeated forever, assuming that the predictor starts off from State `NUM_1`%4 + 1 of the FSM? The states of FSM is marked in a clock-wise order, starting from the top left, as shown below:



A8 Assume `NUM_1` = 1, then starts from State 2:

1. the accuracy of always-taken predictor is $2/5 * 100\% = 40\%$, and the accuracy of always-not-taken predictor is $3/5 * 100\% = 60\%$.
2. Assume that the states of FSM are numbered in clockwise order as 1,2,3,4, where top left state is numbered 1. Let's first list the conditions for the first two iteration of repeating pattern as follows:

	Iteration1					**Iteration2**				
Branch output	NT	NT	T	T	NT	NT	NT	T	T	NT
State	2	3	3	4	1	2	3	3	4	1
Predict	T	NT	NT	NT	T	T	NT	NT	NT	T
Change state or not	yes	no	yes	yes	yes	yes	yes	yes	no	yes
Accurate or not	×	✓	×	×	×	×	✓	×	×	×

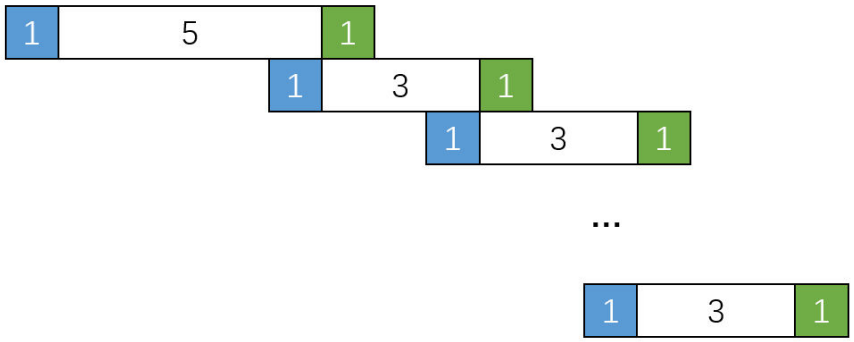
We can see that the condition of the first branch of iteration 2 is just the same as that of iteration 1, which means the following iterations keep the same as iteration 1. So we can use the result of iteration 1 to evaluate the accuracy of the 2-bit predictor, and that is $1/5 * 100\% = 20\%$.

Q9 (15%) This exercise examines memory module interleaving. To transfer a **8-byte** block, assume it takes one clock to send address to DRAM memory and one clock to send data back. DRAM has `NUM_1`%2 + 5 cycle latency for first byte, and `NUM_1`%2 + 3 cycles for each of subsequent bytes in the block.

1. What is the total cycle number if we use a **non-interleaved** system? Please **draw the diagram** to illustrate the calculation.
2. How many interleaved modules (banks) do we need **in minimal** if we want to **reduce the total cycle number to** $\leq 50\%$ of the cycle number in a non-interleaved system? Please **draw the diagram** to illustrate the calculation.
3. Please explain the main idea of interleaving.
4. Does interleaving takes advantages of spatial locality or temporal locality?
5. From your opinion, what's the difference against using multiple main memory cards?

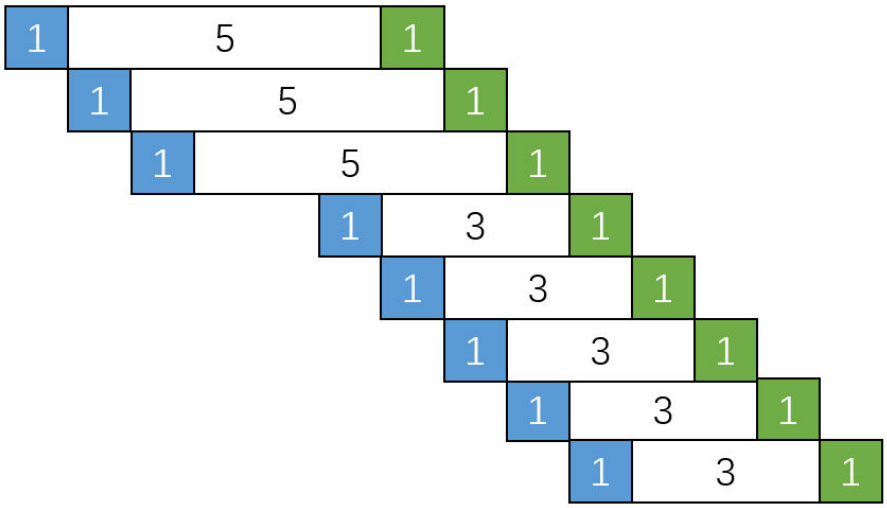
Q9 Assume `NUM_1` = 2, then

1. The situation is shown as below:



Then we have total cycle = $1 + 5 + 7 \times 3 + 1 = 28$

2. The target total cycle = $28 \times (1 - 50\%) = 14$. By enumerating systems with different numbers of interleaved modules, we can get the answer is 3, as shown below:



where total cycle = $1 + 5 + 7 \times 1 + 1 = 14$

3. The main idea of interleaving is trying to hide access latency by interleaving memory accesses across several memory modules. With the help of memory interleaving, we can access multiple modules at the same time thus achieving parallelism.
4. No. The transfered blocks will not be kept, so it does not take advantage of temporal locality. On the other hand, consecutive data is allocated to different memory banks (thus different locations). Therefore, there is no reliance on spatial locality.
5. Both multiple memory cards and memory interleaving can improve parallelism. The difference is that interleaving can improve parallelism at a smaller cost. In memory interleaving different banks can shared the same bus, while with multiple memory cards, each card need an individual bus.