Problem 1: Y86-Assembly (24’)

1.

```
30f4 0003 0000 0000 0000

mrmovq (%rdi), %rax

6002

0x0078

3301 0000 0000 0000

.pos 0x300
```

2.

%rsi	0x0
%rdx	0x4566
%rbp	0x300
CC.ZF	1
CC.SF	0
CC.OF	0

Problem 2: Y86-HCL (8’)

```
1. bool NAE = !(a && b && c) && !(!a && !b && !c);

2. word range = [
       a < b && b < c    : c - a;
       a < c && c < b    : b - a;
       b < a && a < c    : c - b;
       b < c && c < a    : a - b;
       c < a && a < b    : b - c;
       1                 : a - c
   ];
```

Problem3: Storage Hierarchy(18’)

1. No. Because Hard disk need to seek and rotate before writing data; Because write granularity of SSD is larger than read granularity. If we do random write on SSD, it will cause write amplification to some extent.
2. 1 2 5
3.

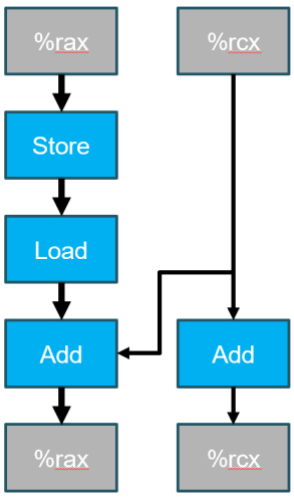
Fronted cache for backend database	**DRAM**

Video player	Hard disk
Long-loading games	SSD
Persistent data structures	PM

Problem 4: Optimization (20’)

1.
```
void populate_order(order_t *order, double *prices)
{
    double sum1 = 0, sum2 = 0;
    double *v = order->item_price;
    int n = number_of_items(order);
    int limit = n - 1;
    if (order->item_prices == NULL)
        order->item_prices = new double[n];
    int i = 0 for (; i < limit; i += 2)
    {
        v[i] = prices[i];
        v[i + 1] = prices[i + 1];
        sum1 += v[i];
        sum2 += v[i + 1];
    }
    for (; i < n; i++)
    {
        v[i] = prices[i];
        sum1 += v[i];
    }
    order->total = sum1 + sum2;
}
```

2.



3.

3.1 2+2+3+1=8 (the store instructions write to the same address in all loop iterations, so each store instruction takes 2 cycles to complete, resulting in 2+2=4 cycles; the remaining 3 cycles account for two parallel load, and 1 cycle for add)
3.2 3+3+3+1=10 (the first and second store always write to different address, each store instruction takes 3 cycles to complete)

Problem 5: Processor (30')

1 [1] D_icode [2] D_valC

2 [3] E_icode == IDJXX && !e_Cnd

[4] B [5] --

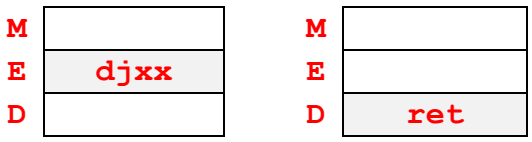
3 When an mispredicted djxx enters M-stage, we need to select the PC value of the instruction following $Instr_d$. However, M_valA is the PC of $Instr_d$. There are two ways to correct it.

There are correct solutions. Here we give some possible solutions.

Solution1: Add a new E-register: E_valP which simply fetches its value from D_valP, and select E_valP when mispredicted djxx enters M-state.

Solution2: Let d_valA of djxx be f_valP (i.e., forward f_valP to djxx'x d_valA).

4 a)



b) [6] S/-- [7] B [8] --

5 (Not the only solution) Replace Line 5 (jle Nops) with djle2 Nops. Move Line 8 and Line 9 before Line 6(iaddq $1, %rax).

Since djxx2 delays jumping until the next 2 instructions are executed, even if djxx2 is mispredicted, we do not need to "cancel" any instruction (by inserting bubbles to the pipeline).