2023 秋 - 计算机组成原理书面作业参考答案

1 第一单元

(1) B。256 = 2^6 * 4, 因此相当于要支持 31+4=35 条 3 地址指令。而 35 条指令相当于需要 6 位 opcode，因此共需要 6 + 6 * 3 = 24 位的指令字长。因此选择 B。

(2) C。反码为原码在保持符号位的基础上，其余位置全部取反。因此反码的 0 与原码一样有 +0 和 -0 两种表示。因此 C 选项错误。

(3) BD。B 选项 110000…000 的原码和补码表示相同。D 选项，对于 x = 10000…000（如 int 中的 -2147483648），有 -x = x，D 项结论不一定成立。因此选择 BD。

(4)（1）中，当 exp=0, fraction=0 时表示 0，根据符号位判断正负，因此为 1_00000_00000_00000。（2）中，非规格化数的 exp=0。而最大的非规格化数即为正数，而且尾数最大的。即 0_00000_11111_11111。

(5)（1）0; 1 （2）P4P1P2D1P3D2D3D4=00110011; P4P1P2D1P3D2D3D4=11000011

(6) 不可以，虽然 ret 指令可以由 jalr zero, ra, 0 替代，但 call 指令至少需要一条指令获取 PC，进行相对跳转；或者一条指令将函数的地址放到寄存器中，不能够只使用 jalr 指令完成。

(7) -3 = (111101)_2；10 = (001010)_2

-3 * 10 = `111101 * (0*1 - 2^1 + 2^2 - 2^3 + 2^4 + 0*2^5)`

因此：

```
      退位    退位
- 1    1  1    0 1 0
+ 1    1  0    1 0 0
- 1    0  1    0 0 0
+ 0    1  0    0 0 0
---------------------
  1    0  0    0 1 0
```

若全部换做加法，则有：

```
     0    0    0 1 1 0
+    1    1    0 1 0 0
+    0    1    1 0 0 0
+    0    1    0 0 0 0
```

```
  进2位 进位 进位
-----------------------
     1    0    0 0 1 0
```

(100010)_2 = -30。

若使用 ppt 上乘法器的方式，则有：

取 X = -3 = (111101)_2；Y = 10 = (001010)_2

```
步骤    状态       部分积          附加位
0       INIT     000000|001010    0
1       R SHIFT  0000000|00101    0
2       -x       0000110|00101    0
        R SHIFT  00000110|0010    1
3       +x       11111010|0010    1
        R SHIFT  111111010|001    0
4       -x       000010010|001    0
        R SHIFT  0000010010|00    1
5       +x       1111100010|00    1
        R SHIFT  11111100010|0    0
6       R SHIFT  111111100010|    0
```

有 (111111_100010)_2 = -30。

2 第二单元

(1) C。C 选项有明显错误，如果先处理 IF 段请求，则 MEM 仍会导致 IF/ID/EXE 段阻塞，因此应该先处理 MEM 段，让指令更早地发射。

(2) AB。C 选项，BTB 可以减少控制冲突。D 选项，本题由于不涉及读内存，因此数据旁路可以解决所有数据冲突。

(3) 错误，CPU 的性能需要综合考虑 CPI 和主频，若 CPI 高，但是主频低，最后执行程序的性能仍然不好。

(4) 错误，在 RISCV 标准 5 段流水线中下面的指令的数据冲突无法解决：

```
lw s0, 0(x0)
add s0, s0, s0
```

答出如上的 load-relate 等因为数据单纯没取出来的冲突即可，注意 `lw,sw` 的组合是可以解决的，这个不行。

(5) 可以，在 CPU 处理第一个中断时会关闭中断（xstatus.xIE=0），此时可以在内存中保存现场（如寄存器，关键 CSR），然后软件打开中断处理嵌套中断。

(6)（1）

Offset = -5 * 4 = -20

-20 = ~(0 0000 0001 0100)_2+1 = (1 1111 1110 1100)_2

因此指令二进制为：

```
1    111111    01100  01111  001   0110    1    1100011
imm[12|10:5]    rs2    rs1   BEQ   imm[4:1|11]   opcode

1111 1110 1100 0111 1001 0110 1110 0011
  F    E    C    7    9    6    E    3
```

因此答案为 0xFEC796E3

（2）

1 0000 0000 0000 0000 = -2^12 Byte

0 1111 1111 1111 1110 = 2^12 -2 Byte

所以跳转范围是 PC - 2^12 Byte -> PC + 2^12 - 2 Byte

（3）

```
fibo[0] = fibo[1] = 1;
for(int i = 2; i < 10; i++)
    fibo[i] = fibo[i - 1] + fibo[i - 2];
```

结束时 a4 为 55。

（4）

下面使用 reg, (line, cmd)->(line, cmd) 指代数据冲突。

```
a5, (1, lui) -> (2, addi)
a5, (1, lui) -> (4, addi)
a4, (2, addi) -> (5, sw)
a3, (3, li) -> (5, sw)
a3, (3, li) -> (6, sw)
a5, (4, addi) -> (7, addi)
a4, (8, li) -> (14, add)*
a4, (14,add) -> (15, sw)
a5, (16, addi) -> (17, bne)
a5, (16, addi) -> (11, lw)*
a5, (16,addi) -> (12, lw)*
a4, (11, lw) -> (14, add)
```

其中带 * 号的为考虑分支预测之后会产生的数据冲突。

(7)（1）a)

所有可能的数据冲突如下：

```
a1, ori, MEM -> slli, EXE    => ALU@EXE/MEM->ALU
t0, slli, MEM -> add, EXE    => ALU@EXE/MEM->ALU
t0, add, MEM -> lw, EXE      => ALU@EXE/MEM->ALU
t0, add, WB -> lw, EXE       => ALU@MEM/WB->ALU
                                和 ALU@EXE/MEM->RF
t0, add, WB -> sw, ID        => ALU@MEM/WB->RF
t2, lw, WB -> sw, MEM        => DM @MEM/WB->DM
```

因此写出最后三种情况的任意一种即可，而且只能出现其中一种。

剩下的内容是可以出现的：

```
DM@MEM/WB->ALU
DM@MEM/WB->RF
xx@xxxxxx->BC
```

注意，这题的前传是数据已经计算好放到流水线寄存器里面的，因此不能写 ALU@EXE -> RF 这种前传网络。

(I) 若 a) 答案为 ALU@MEM/WB->ALU 和 ALU@EXE/MEM->RF，则一种可行答案为：

```
...
add t0, a0, t0
lw t1, 0(t0)
nop (i.e. add zero, zero, zero)
lw t2, 4(t0)
...
```

若 a) 答案为 ALU@MEM/WB->RF，则一种可行答案为：

```
...
add t0, a0, t0
lw t1, 0(t0)
lw t2, 4(t0)
nop
sw t2, 0(t0)
...
```

若 a) 答案为 DM@MEM/WB->DM，则一种可行答案为：

```
...
add t0, a0, t0
lw t1, 0(t0)
lw t2, 4(t0)
sw t1, 4(t0)
sw t2, 0(t0)
```

（2）a)

IF	ID	EXE	MEM	WB
nop	sw	bne	add t1, t1, t0	addi t3, t3, -1

(I) 写出所有可能的数据冲突：

```
t3, addi, WB -> addi, ID      ALU@MEM/WB->RF
t3, addi, WB -> bne, EXE      ALU@MEM/WB->BC 或
                              ALU@EXE/MEM->RF
```

再写出所有可能的数据旁路：

```
ALU@EXE/MEM -> RF,  ALU*, BC*
ALU@MEM/WB  -> RF,  ALU*, BC,  DM*
DM@MEM/WB   -> RF*, ALU*, BC*, DM*
```

标 * 号的没有被用到。

（3）方案二不满足，例子如下：

```
j LABEL
RV.X
```

注意到 RV.X 指令在顺序执行的情况下不可达（会跳转）（即只能跳转过来触发）。

而当 j 指令在 EXE 段执行跳转时，流水线如下：

IF	ID	EXE	MEM	WB
*	RV.X	J LABEL	*	*

此时会触发异常，这是不正确的，不符合精确异常的定义。

3 第三单元

(1) B。平均旋转时间：0.5r / (10000r/min) * 60 * 10^3ms/min = 3ms。读取数据的时间 4KB / (20MB/s) / (1024KB/MB) * (1000ms/s) = 0.1953ms。因此总延迟为：6ms+3ms+0.1953ms+0.2ms = 9.4ms。

(2) B。虚拟内存会导致每次访存需要的访存次数增加，因此不能提高存储访问性能，剩下两项均正确，因此选择 B。

(3) 错误，考虑如下情况：

二路组相联和全连接，均 4 项，访问序列为 0，1，2，3，5，0，1，2，3，5，…

	0	1	2	3	5	0	1	2	3	5	命中率
二路组相联	×	×	×	×	×	√	×	√	×	×	40%
全相联	×	×	×	×	×	×	×	×	×	×	0%

(4) 错误，RAID1 相比 RAID0 只有读性能提高两倍，但有写冗余，因此写效率会下降。

(5) (450 * 100ns + (5000 - 450) * 20 ns) / 5000 = 27.2ns。

(6) 512GB；512 * (6 / 4) = 768 GB。

(7)（1）256B = 2^8 Bytes，因此 Offset 有 8 位。故页号有 12-8 = 4 位。

而 TLB 为 2 路组相联，8 项，因此 index 有 log(8/2) = 2 位，其余 2 位为 tag。因此

11: 10	9: 8	7: 0
TLBT	TLBI	VPO

11: 8	7: 0
VPN	VPO

（2）PPO = VPO = 8 位；PPN = 12 - 8 = 4 位

CO = log（4）= 2 位，CI = log(8/2) = 2 位

CT = 12 - 2 - 2 = 8 位

11: 4	3: 2	1: 0
CT	CI	CO

11: 8	7: 0
PPN	PPO

（3）

0x48a => tag = 1, index = 0, VPN = 4；0xa8a => PPN = A

0x1ea => tag = 0, index = 1, VPN = 1；0xeea => PPN = E

0xE0F => tag = 3, index = 2, VPN = E => PPN = 3

0xB5A => tag = 2, index = 3；VPN = B => PPN = 4

0xF5A => 0xB5a, VPN = F => PPN = B

0x88A => VPN = 8

0x56F => VPN = 5

VPN = A => tag = 2, index = 2 => PPN = F

VPN = C => tag = 3, index = 0 => PPN = 0

VPN = D => tag = 3, index = 1 => PPN = 2

VPN	PPN	Valid	VPN	PPN	Valid
0	1	1	8	/	0
1	E	1	9	5	1
2	5	0	A	F	1
4	A	1	C	0	1
5	/	0	D	2	1
7	C	1	F	B	1

Index	Tag	PPN	Valid
0	1	A	1
1	0	E	1
2	3	3	1
3	2	4	1

（4）0xA8A => Tag = 0xA8, index = 2

每个 index 有 2 路，一共 4 个 index，index = 2 的其中一路 tag 为 0xA8

访问过程：取出地址的 3:2 位，作为 index 访问 Cache。将地址的 11:4 位与取出的 cache 项的 tag 比较，若相等且 valid，则命中并返回数据；若两项的 tag 都不匹配则返回 miss。

（5）略

(8)（1）64B 的 Cache Line Size => Cache Offset 为 6 位。

32 KB，8 路 => 共 4*1024/64=64 组 => Cache Index 为 6 位。

Cache Tag	Cache Index	Cache Offset
31:12	11:6	5:0

（2）考虑第一行的 3 次是在哪里命中的：

15, 8193, 16384, 63, 4096, 8194, 64, 16385

15，8193，16384，4096 不可能命中，因此只能在 63，8194，64，16385 中选择 3 次命中。

注意到如果 64 命中（0 带进来，128B Cache Line）则 63 必然命中。此时 0，8192，16384，4096 必然在一组中。因为 63 命中，所以路数大于等于 4。此时 63，8194，64，16385 必然全部命中，不符题意。

因此 64 是没有命中的。此时可以确定 Cache Line 大小为 64B，至少有 4 路。

考虑此时 Cache Index 0 的情况：0，8192，16384，4096。

进入第二行，

32768，0，129，1024，3072，8192，260，513

32768 必须将 0 换出，否则 0 将命中，不符题意。

此时的 Cache index 0 的情况为：

16384，4096，32768，0

因此确认缓存为 4 路。

此时考虑缓存的大小：

若为 4KB，则 cache index 为 0~15。执行完第二行后，Cache Index 0 的情况是：

0，1024，3072，8192

这时第 3 行的 4096 不可能命中。因此判断缓存大小为 8KB，进行验证。

若为 8KB，则 cache index 为 0~31

4096，32768，0，8192

第三行的 0，4，8，4096 均命中

64 和 128 由于没有人将其换出，分别在第一行 64 和第二行 129 时进入缓存，因此全部命中，满足题目要求。

综上，缓存大小为 8KB，4 路，缓存行大小为 64B。

（3）将循环顺序换为 ikj 或 kij 均可，即要求 k 层循环在 j 层循环外。

下面的计算以如下的程序为例，并认为缓存的替换策略为 LRU：

```
for (int i = 0; i < n; i ++)
   for (int k = 0; k < n; k ++) {
      int tmp = M[i][k]; // in register
      for (int j = 0; j < n; j ++)
         Q[i][j] = Q[i][j] + a * N[k][j];
   }
```

由于 `N[k][j]` 和 `Q[i][j]` 在最内层循环稠密访问，故这两个数组在内层循环必然会将 `M[i][k]` 换出缓存，即 M 数组不可能命中。

由于 A 缓存为 8 路，即使 `Q[i][j]` 和 `N[k][j]` 在缓存中的 index 相同，它们也不会将对方一直换出缓存。

分析 Q 的命中率：缓存行大小为 64B，双精度浮点数为 8B。故每 2*8 次访问会出现 1 次 miss。

N 的命中率类似，每 8 次访问会出现 1 次 miss。

故总体的命中率为：

$$\lim_{n\to+\infty}\frac{15*n^3+7*n^3+0}{16*n^3+8*n^3+n^2}=\frac{11}{12}\approx 91.67\%$$

4 第四单元

(1) C。I1 被屏蔽，根据优先级响应 I3。

(2) D。DMA 设备可以独占内存总线。

(3) C。总线仲裁由内部的仲裁器完成。

(4) 增加总线宽度、分离地址和数据总线、使用成组传输。

(5) 错误，RISC-V 将外设映射到地址空间上，使用 load/store 指令访问外设。

(6)（1）指令的执行时间：200 instructions / (0.75 instructions / clocks) / (1G clocks / s) = 200s / (0.75 * 10^9) 。

因此数据传输速率为 2Byte / (200s / (0.75 * 10^9)) = 7.5 * 10^6 B / s。

（2）200 instructions -> 500 instructions 因此速率应变为 0.4 倍。

所以传输率为 3 * 10^6 B / s。

（3）传输需要的时间如下：

100 clocks / (1G clocks / sec) + 300 instructions / (0.75 instructions / clocks) / (1G clocks / s) = 100 * 10^(-9)s + 300 / 0.75 * 10^(-9)s = 500 * 10^(-9)s

传输速率为 4KB / (500 * 10^(-9)s) = 8*10^6 KB/s。