Name: SOLUTIONS	UID: CS 33

1. Calculate the Size

(a) Given a 32-bit virtual address space and a 24-bit physical address, determine the number of bits in the VPN, VPO PPN, and PPO for the following page sizes:

Page Size	VPN bits	VPO bits	PPN bits	PPO bits
1 KiB	22	10	14	10
2 KiB	21	11	13	11
4 KiB	20	12	12	12
8 KiB	19	13	11	13

The number of VPO and PPO bits are the same, and is the log base 2 of the page size. VPN and PPN are the remaining bits from the virtual address size and the bhysical address size, respectively.

Thus, for 1KiB:

- VPO = PPO = $\log_2(1024) = 10$
- VPN = 32 (virtual address size) - 10 = 22
- PPN = 24 (physical address size) - 10 = 14

(b) For each of the following caches and a 16-bit physical address, fill in the table with the appropriate number of bits (assume that we are doing 1-byte cache accesses)

Cache	Tag bits	Index bits	Offset bits	Total Data in Cache
Direct-mapped cache with 32 sets and a cache line size of 4 bytes	9	5	2	128
2-way set associative cache with 128 sets and a cache line size of 32 bytes	4	7	5	8192
8-way set associative cache with 64 sets and a cache line size of 1 byte	10	6	0	512
16-way fully associative cache and a cache line size of 64 bytes	10	0	6	1024

Index bit = $\log_2$(# of sets)
Offset bits = $\log_2$(# of data items per cache line)
Tag Bits = Physical Address Bits - Index Bits - Offset Bits
Bytes in Cache = # of Cache Lines per Set * Cache Line Size * # of Sets
Note: Fully Associative Cache has only 1 set!

2. What's that Data?

You are given the following details about the memory system and the states of the TLB and caches.

- 16-bit physical address, 12-bit virtual address, 16-byte page size (2^4 bytes)
- 2-way set associative TLB with 16 entries
- 8-way set associative cache with 2 sets and a cache block size of 4 bytes
- The replacement policy for the TLB and cache is LRU, with most recently used on top

TLB			
Index	Tag	Valid	PPN
0	15	1	A80
0	01	1	408
1	0D	1	D6C
1	00	0	D8D
2	08	1	4C2
2	1F	0	5BE
3	0D	0	490
3	02	1	9C2
4	14	1	FA3
4	11	1	CA4
5	05	1	5AD
5	10	1	C9D
6	19	1	0CE
6	0F	1	E7C
7	10	1	087
7	01	0	D87

Page Table					
VPN	Valid	PPN	VPN	Valid	PPN
01	0	D8D	68	1	490
05	0	8F6	78	1	087
0D	1	A21	7B	0	823
0F	0	9FD	7F	0	99D
10	0	0EB	82	1	4F1
13	1	9C2	88	1	CA4
16	1	A80	8F	1	408
2F	0	CE8	94	0	821
30	0	899	A4	1	FA3
31	1	9D1	D0	1	80C
34	1	0CE	D4	1	D87
4E	0	B1B	D6	1	C9D
58	0	DDE	EC	0	BBC
5F	1	5AD	ED	1	49C
63	0	8C7	F2	1	187
69	1	D6C	F8	0	769

Cache						
Index	TAG	Valid	Data			
0	0B5A	1	85	7F	ED	E2
0	036F	1	6E	41	B4	ED
0	1442	1	99	25	DA	18
0	1F46	0	11	92	A1	78
0	019D	0	10	48	16	F9
0	1B0F	1	86	08	16	FD
0	0137	0	46	CF	FA	F7
0	0353	0	5D	46	F3	9E
1	010F	1	63	2A	BD	80
1	0939	1	F6	C3	ED	A9
1	0920	0	BC	54	DF	04
1	193B	1	B9	31	F2	3F
1	1AD9	1	29	80	4C	77
1	13A3	1	D4	52	B0	7A
1	1385	1	FE	F0	3D	89
1	118C	0	FB	40	47	C8

Fill in the following each of the following tables.

Virtual Address	**0x31A**
VPN	0x31
VPO/PPO	0xA
TLB Index	0x1
TLB Tag	0x06
TLB Hit? (Y/N)	N
Page Fault? (Y/N)	N
PPN	0x9D1
Physical Address	0x9D1A
Cache Offset	0x2
Cache Index	0x0
Cache Tag	0x13A3
Data (1-byte access)	Miss

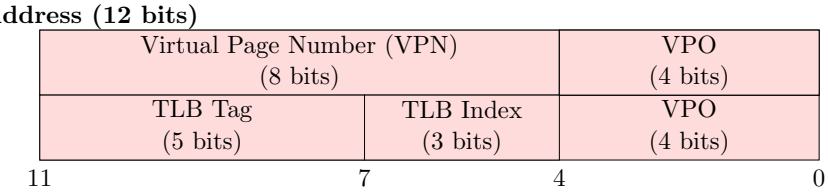
Virtual Address	**0x0D3**
VPN	0x0D
VPO/PPO	0x3
TLB Index	0x5
TLB Tag	0x01
TLB Hit? (Y/N)	N
Page Fault? (Y/N)	N
PPN	A21
Physical Address	A213
Cache Offset	0x3
Cache Index	0x0
Cache Tag	0x1442
Data (1-byte access)	0x18

Virtual Address	**0x0F4**
VPN	0x0F
VPO/PPO	0x4
TLB Index	0x7
TLB Tag	0x01
TLB Hit? (Y/N)	N
Page Fault? (Y/N)	Y
PPN	
Physical Address	
Cache Offset	
Cache Index	
Cache Tag	
Data (1-byte access)	

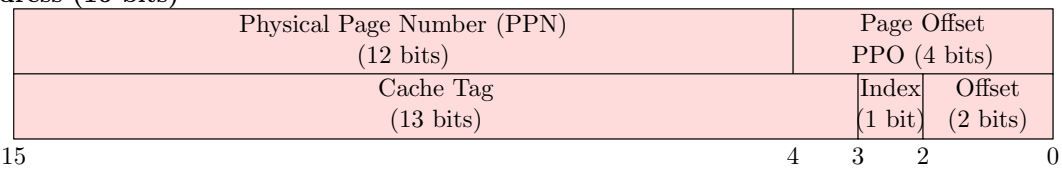
Virtual Address	**0xA40**
VPN	0xA4
VPO/PPO	0x0
TLB Index	0x4
TLB Tag	0x14
TLB Hit? (Y/N)	Y
Page Fault? (Y/N)	N
PPN	0xFA3
Physical Address	0xFA30
Cache Offset	0x0
Cache Index	0x0
Cache Tag	0x1F46
Data (1-byte access)	Miss

We refer to the following table to see how the address is split up:

Virtual Address (12 bits)



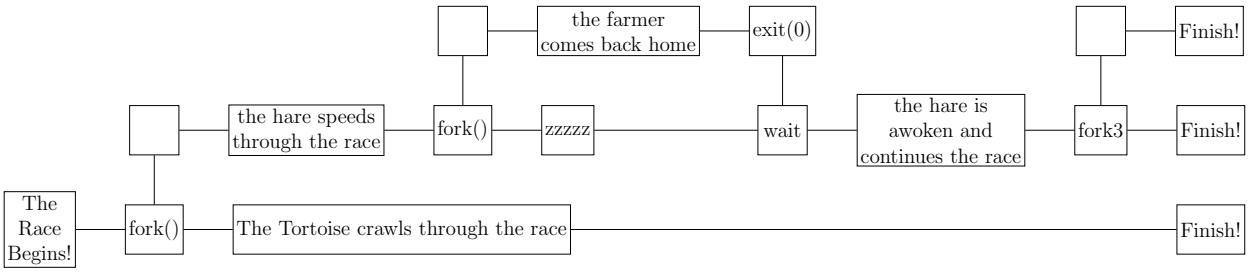
Physical Address (16 bits)



3. Who wins the Race?

For the following code, draw the process graph and provide a possible output.

```
void Tortoise_and_Hare() {
    int child_status;
    printf("The Race Begins!\n");
    if (fork() == 0) {
        printf("The hare speeds through the race!\n");
        // The hare enters the barn to take a nap
        if (fork() == 0) {
            printf("The farmer comes back home!\n");
            exit(0)
        } else {
            printf("zzzz...\n");
            wait(&child_status);
            printf("The hare is awoken and continues the race!\n");
        }
        fork(); // the hare races to the finish with their friends
    } else {
        printf("The Tortoise crawls through the race!\n");
    }
    printf("Finish!\n");
}
```



The process table is given above

A possible output (several are possible):

The Race Begins!
The Tortoise crawls through the race!
Finish!
The hare speeds through the race!
The farmer comes back home!
zzz....
The hare is awoken and continues the race
Finish
Finish