15-213: Final Exam Review

Jack, Nikhil, Raghav, Stan

Malloc final-f10 #1.17

- In malloclab, we provided code for an implicit list allocator (the naive implementation). Many students improved this code by creating an explicit linked list of free blocks. Which of the following reason(s) explain(s) why an explicit linked list implementation has better performance?

Malloc final-f10 #1.17

- (b) II only
- I. Immediate coalescing on free is faster
  - No; coalescing doesn’t depend on the list
- II. Implicit has all blocks vs just free blocks in explicit
  - **Yes** - significantly reduces search time, since there are fewer blocks to look at
- III. Insert into explicit list is faster
  - No; implicit list doesn’t have insertion operation at all, since all blocks are already in the implicit list!

Malloc - other things to know

- Fit algorithms - first/next/best/good fit
- Fragmentation
  - Internal - wasted space inside blocks
  - External - wasted space between blocks
- See the textbook for details

Linking exam2-s09-v1 #4

- (a) How many symbols does `main.c` generate in the executable program's symbol table?

Linking exam2-s09-v1 #4

- (a) How many symbols does `main.c` generate in the executable program's symbol table?
  - 5 symbols:
    - `long a = 1;`
    - `const long b = 2;`
    - `long c;`
    - `long d = -1;`
    - `int main(int argc, char *arg[])`
      `{...}`

Linking exam2-s09-v1 #4

- (b) What are the strong symbols from `main.c`, and what are the weak symbols from `main.c`?

Linking exam2-s09-v1 #4

- (b) What are the strong symbols from `main.c`, and what are the weak symbols from `main.c`?
  - `long a = 1;` **strong**
  - `const long b = 2;` **strong**
  - `long c;` **weak**
    - Does not have a defined value!
  - `long d = -1;` **strong**
  - `int main(int argc, char *arg[]) {...}` **strong**

Linking exam2-s09-v1 #4

- (c) Note the address of b. Why is it far removed from the addresses of the other variables?

Linking exam2-s09-v1 #4

- (c) Note the address of b. Why is it far removed from the addresses of the other variables?
  - Declared as `const long b = ...;`
  - Thus, it’s **r**ead-**o**nly
    - Placed in the `.`**`ro`**`data` section of binary
  - The other variables are NOT read-only
    - Placed in the `.data` section of the binary

Linking exam2-s09-v1 #4

- (d) Why is c located after d in memory, even though it’s before d in Harry’s program?

Linking exam2-s09-v1 #4

- (d) Why is `c` located after `d` in memory, even though it’s before `d` in Harry’s program?
  - `c` is **defined** in `data.c` **after** `main.c` is compiled

Linking exam2-s09-v1 #4

- (e) Note the output given by the final `printf`. Was Harry compiling and running the code on `x86` or `x86-64`? How do you know?
  - You don’t need to know the answer to this :)
  - If you still want to know, look up the address of the start of the `.data` section in `x86` vs. `x86-64`

Linking exam2-s09-v1 #4

- (f) Given that 4294967297 = 2^{32} + 1, what would be output by
  - `printf(“{%d, %d}”, c[0], c[1]);`
- if it were executed in `data.c`?

Linking exam2-s09-v1 #4

- (f) Given that 4294967297 = 2^{32} + 1, what would be output by
  - `printf("{%d, %d}", c[0], c[1]);`
- if it were executed in `data.c`?
  - `"{1, 1}" –` the line that prints `c` as a (64-bit) **`long`** in main prints 4294967297 = 2^{32} + 1 = 0x100000001
  - If we access `c` as an array of 2 (32-bit) `unsigned ints`, we just get the top and bottom 32 bits
  - Note: `c[0]` holds the bottom 32 bits due to little-endianness

Threads and Synchronization final-s11 #5

- What is the problem with the implementation?

- Starvation is a problem where one thread, or kind of thread (think reader or writer), is unable to acquire a resource. After fixing the previous problem, is starvation possible? How?

Threads and Synchronization final-s11 #5

- What is the problem with the implementation?

- Starvation is a problem where one thread, or kind of thread (think reader or writer), is unable to acquire a resource. After fixing the previous problem, is starvation possible? How?

Threads and Synchronization final-s11 #5

- What is the problem with the implementation?

```
void readlock(struct rwlock *lock) {
   while(1) {
      sem_wait(lock->sem);
      if(lock->writers == 0) {
         lock->readers++;
         break;
      } sem_post(lock->sem);
   }
}
```

Threads and Synchronization final-s11 #5

- What is the problem with the implementation?

```
void readlock(struct rwlock *lock) {
    while(1) {
        sem_wait(lock->sem);
        if(lock->writers == 0) {
            lock->readers++;
            break; // same goes for writelock!
        } sem_post(lock->sem);
    }
}
```

Threads and Synchronization final-s11 #5

- What is the problem with the implementation?
  - When either a read or write lock is acquired, the function returns without calling `sem_post`

```
readlock:
while true:
 –  sem_wait(sem)
 –  if writers == 0
     –  readers++; break;
 –  sem_post(sem)
```

```
unlock:
sem_wait(sem)
if lock->readers > 0
 –  lock->readers--
else
 –  lock->writers--
sem_post(sem)
```

Threads and Synchronization final-s11 #5

- What is the problem with the implementation?
  - When either a read or write lock is acquired, the function returns without calling `sem_post`

```
readlock:
while true: (1)
 –  sem_wait(sem)
 –  if writers == 0
(2) –  readers++; break;
 –  sem_post(sem)
```

```
unlock:
sem_wait(sem) (3) … :(
if lock->readers > 0
 –  lock->readers--
else
 –  lock->writers--
sem_post(sem)
```

Threads and Synchronization final-s11 #5

- Starvation is a problem where one thread, or kind of thread (think reader or writer), is unable to acquire a resource. After fixing the previous problem, is starvation possible? How?

Threads and Synchronization final-s11 #5

- Starvation is a problem where one thread, or kind of thread (think reader or writer), is unable to acquire a resource. After fixing the previous problem, is starvation possible? How?
  - Yes. Writers can be starved as long as one reader remains in the critical section at all times.

Threads and Synchronization final-s11 #5

- Starvation is a problem where one thread, or kind of thread (think reader or writer), is unable to acquire a resource. After fixing the previous problem, is starvation possible? How?

```
readlock:
while true:
 -  sem_wait(sem)
 -  if writers == 0
     -  readers++; break;
 -  sem_post(sem)
sem_post(sem)
```

```
writelock:
while true:
 -  sem_wait(sem)
 -  if writers == 0 and readers == 0
     -  writers = 1; break;
 -  sem_post(sem)
sem_post(sem)
```

Signals final-f10 #10

- A form of inter-process communication
- Sending and receiving a signal:
  a. “Sending” process tells kernel to send signal to target process

Signals final-f10 #10

- A form of inter-process communication
- Sending and receiving a signal:
  a. "Sending" process tells kernel to send signal to target process
  b. Kernel updates the pending signals mask to show that a signal has arrived

Signals final-f10 #10

- A form of inter-process communication
- Sending and receiving a signal:
  a. "Sending" process tells kernel to send signal to target process
  b. Kernel updates the pending signals mask to show that a signal has arrived
  c. Target process handles delivered (i.e. not blocked) signal when it jumps from kernel mode to user mode (getting context switched to, returning from syscall, etc.)

Signals final-f10 #10

Strategy: **Process graphs!**

Snippet 1 outcome	Possible?
Nothing is printed	
“A” is printed	
“B is printed	
“Ab” is printed	
“Ba” is printed	
A process does not terminate	

Signals final-f10 #10

Strategy: **Process graphs!**

Snippet 1 outcome	Possible?
Nothing is printed	Yes
“A” is printed	
“B is printed	
“Ab” is printed	
“Ba” is printed	
A process does not terminate	

Signals final-f10 #10

Strategy: **Process graphs!**

Snippet 1 outcome	Possible?
Nothing is printed	Yes
“A” is printed	Yes
“B is printed	
“Ab” is printed	
“Ba” is printed	
A process does not terminate	

Signals final-f10 #10

Strategy: **Process graphs!**

Snippet 1 outcome	Possible?
Nothing is printed	Yes
“A” is printed	Yes
“B is printed	Yes
“Ab” is printed	
“Ba” is printed	
A process does not terminate	

Signals final-f10 #10

Strategy: **Process graphs!**

Snippet 1 outcome	Possible?
Nothing is printed	Yes
“A” is printed	Yes
“B is printed	Yes
“Ab” is printed	Yes
“Ba” is printed	
A process does not terminate	

Signals final-f10 #10

Strategy: **Process graphs!**

Snippet 1 outcome	Possible?
Nothing is printed	Yes
“A” is printed	Yes
“B is printed	Yes
“Ab” is printed	Yes
“Ba” is printed	Yes
A process does not terminate	

Signals final-f10 #10

Strategy: **Process graphs!**

Snippet 1 outcome	Possible?
Nothing is printed	Yes
“A” is printed	Yes
“B is printed	Yes
“Ab” is printed	Yes
“Ba” is printed	Yes
A process does not terminate	No

Signals final-f10 #10

Strategy: **Process graphs!**

Snippet 2 outcome	Possible?
Nothing is printed	
“ba” is printed	
“abc is printed	
“bac” is printed	
“bca” is printed	
A process does not terminate	

Signals final-f10 #10

Strategy: **Process graphs!**

Snippet 2 outcome	Possible?
Nothing is printed	No
“ba” is printed	
“abc is printed	
“bac” is printed	
“bca” is printed	
A process does not terminate	

Signals final-f10 #10

Strategy: **Process graphs!**

Snippet 2 outcome	Possible?
Nothing is printed	No
“ba” is printed	No
“abc is printed	
“bac” is printed	
“bca” is printed	
A process does not terminate	

Signals final-f10 #10

Strategy: **Process graphs!**

Snippet 2 outcome	Possible?
Nothing is printed	No
“ba” is printed	No
“abc is printed	No
“bac” is printed	
“bca” is printed	
A process does not terminate	

Signals final-f10 #10

Strategy: **Process graphs!**

Snippet 2 outcome	Possible?
Nothing is printed	No
“ba” is printed	No
“abc is printed	No
“bac” is printed	Yes
“bca” is printed	
A process does not terminate	

Signals final-f10 #10

Strategy: **Process graphs!**

Snippet 2 outcome	Possible?
Nothing is printed	No
“ba” is printed	No
“abc is printed	No
“bac” is printed	Yes
“bca” is printed	Yes
A process does not terminate	

Signals final-f10 #10

Strategy: **Process graphs!**

Snippet 2 outcome	Possible?
Nothing is printed	No
“ba” is printed	No
“abc is printed	No
“bac” is printed	Yes
“bca” is printed	Yes
A process does not terminate	Yes

Virtual Memory

Virtual Address - 18 Bits

Physical Address - 12 Bits

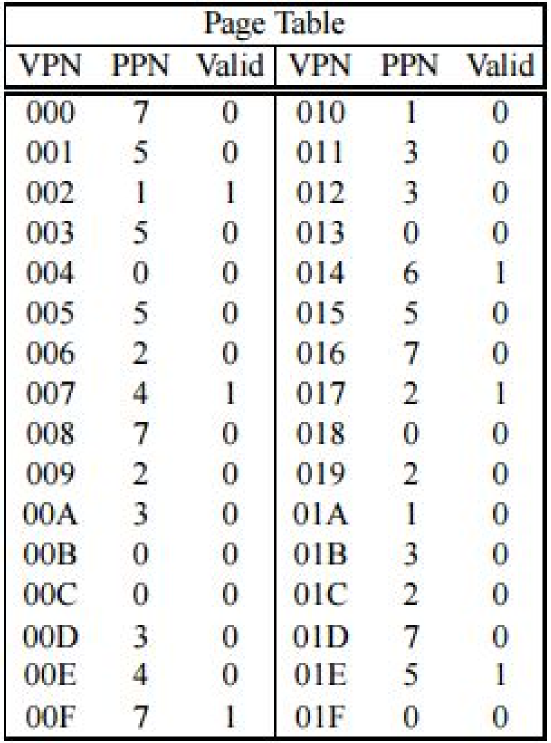
Page Size - 512 Bytes

TLB is 8-way set associative

Cache is 2-way set associative

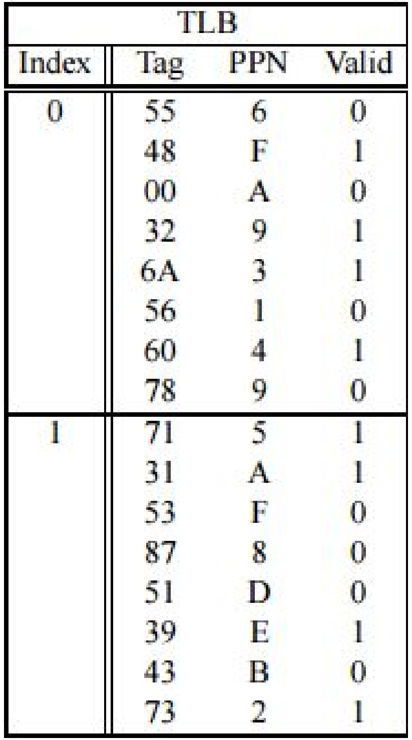
Final S-02 (#5)

Lecture 18: VM - Systems



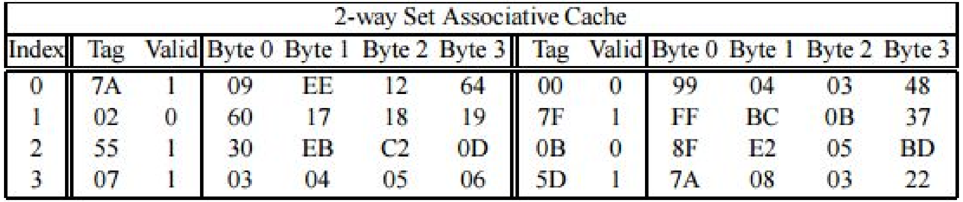
Page Table

VPN	PPN	Valid	VPN	PPN	Valid
000	7	0	010	1	0
001	5	0	011	3	0
002	1	1	012	3	0
003	5	0	013	0	0
004	0	0	014	6	1
005	5	0	015	5	0
006	2	0	016	7	0
007	4	1	017	2	1
008	7	0	018	0	0
009	2	0	019	2	0
00A	3	0	01A	1	0
00B	0	0	01B	3	0
00C	0	0	01C	2	0
00D	3	0	01D	7	0
00E	4	0	01E	5	1
00F	7	1	01F	0	0



TLB

Index	Tag	PPN	Valid
0	55	6	0
	48	F	1
	00	A	0
	32	9	1
	6A	3	1
	56	1	0
	60	4	1
	78	9	0
1	71	5	1
	31	A	1
	53	F	0
	87	8	0
	51	D	0
	39	E	1
	43	B	0
	73	2	1



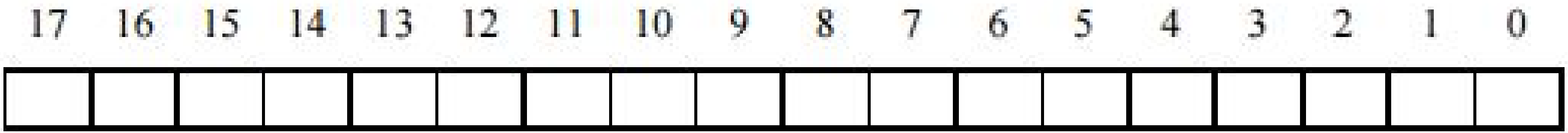
2-way Set Associative Cache

Index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	7A	1	09	EE	12	64	00	0	99	04	03	48
1	02	0	60	17	18	19	7F	1	FF	BC	0B	37
2	55	1	30	EB	C2	0D	0B	0	8F	E2	05	BD
3	07	1	03	04	05	06	5D	1	7A	08	03	22

Virtual Memory

Label the following:

(A) *VPO:* Virtual Page Offset
(B) *VPN:* Virtual Page Number
(C) *TLBI:* TLB Index
(D) *TLBT:* TLB Tag



17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Virtual Memory

Label the following:

(A) *VPO:* Virtual Page Offset - Location in the page

Page Size = 512 Bytes = 2^9 → Need 9 bits

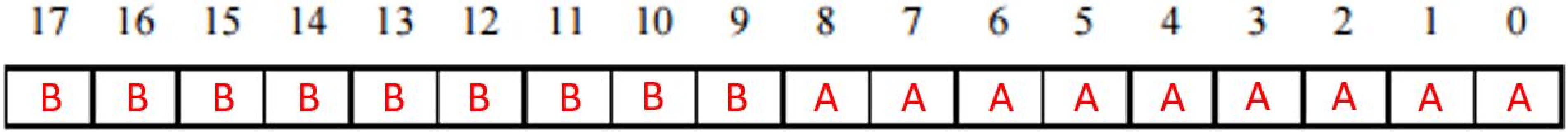
17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
									A	A	A	A	A	A	A	A	A

Virtual Memory

Label the following:

(A) *VPO:* Virtual Page Offset

(B) *VPN:* Virtual Page Number - Everything Else



17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	B	B	B	B	B	B	B	B	A	A	A	A	A	A	A	A	A

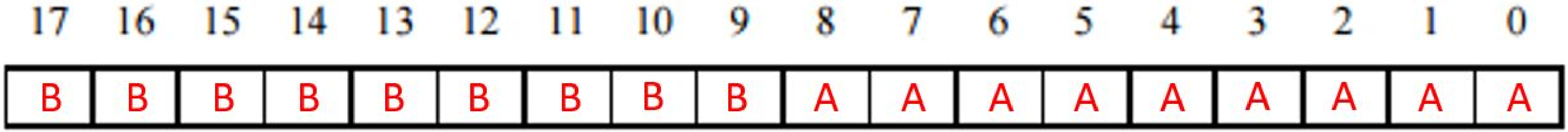
Virtual Memory

Label the following:

(A) *VPO:* Virtual Page Offset

(B) *VPN:* Virtual Page Number

(C) *TLBI:* TLB Index - Location in the TLB Cache



17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	B	B	B	B	B	B	B	B	A	A	A	A	A	A	A	A	A

Virtual Memory

Label the following:

(A) *VPO:* Virtual Page Offset

(B) *VPN:* Virtual Page Number

(C) *TLBI:* TLB Index - Location in the TLB Cache

2 Indices → 1 Bit

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	B	B	B	B	B	B	B	B	A	A	A	A	A	A	A	A	A
								TLBI									

Virtual Memory

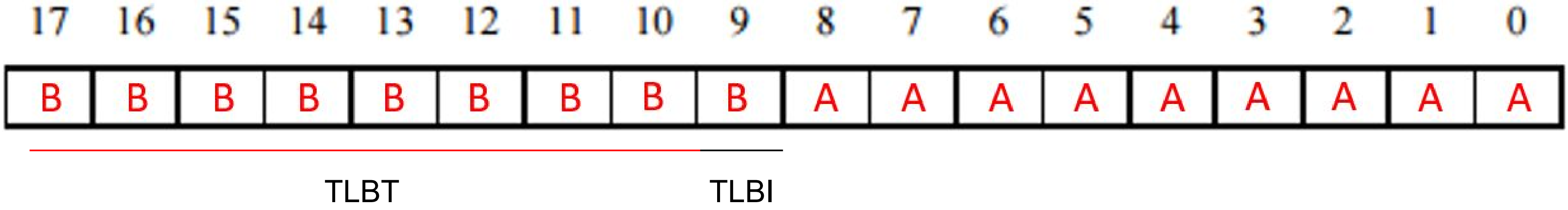
Label the following:

(A) *VPO:* Virtual Page Offset

(B) *VPN:* Virtual Page Number

(C) *TLBI:* TLB Index

(D) TLBT: TLB Tag - Everything Else



17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	B	B	B	B	B	B	B	B	A	A	A	A	A	A	A	A	A

Virtual Memory

Label the following:

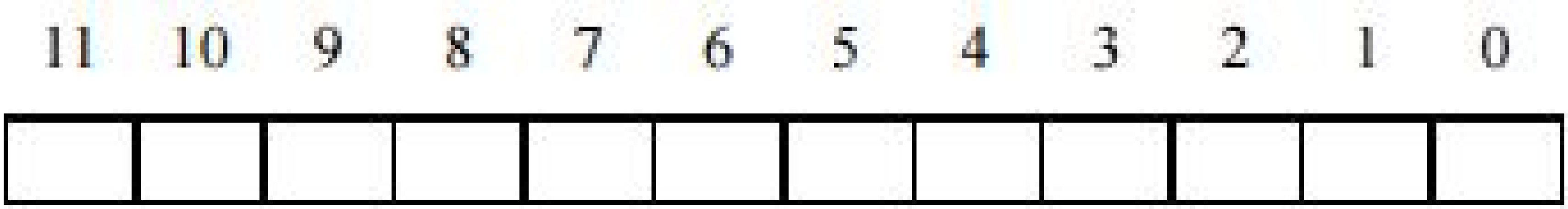
(A) *PPO:* Physical Page Offset
(B) *PPN:* Physical Page Number
(C) *CO:* Cache Offset
(D) *CI:* Cache Index
(E) *CT:* Cache Tag

11	10	9	8	7	6	5	4	3	2	1	0

Virtual Memory

Label the following:

(A) *PPO:* Physical Page Offset



11	10	9	8	7	6	5	4	3	2	1	0

Virtual Memory

Label the following:

(A) *PPO:* Physical Page Offset - Same as VPO

11	10	9	8	7	6	5	4	3	2	1	0
			A	A	A	A	A	A	A	A	A

Virtual Memory

Label the following:

(A) *PPO:* Physical Page Offset - Same as VPO

(B) *PPN:* Physical Page Number - Everything Else

11	10	9	8	7	6	5	4	3	2	1	0
B	B	B	A	A	A	A	A	A	A	A	A

Virtual Memory

Label the following:

(A) *PPO:* Physical Page Offset - Same as VPO

(B) *PPN:* Physical Page Number - Everything Else

(C) *CO:* Cache Offset - Offset in Block

11	10	9	8	7	6	5	4	3	2	1	0
B	B	B	A	A	A	A	A	A	A	A	A

Virtual Memory

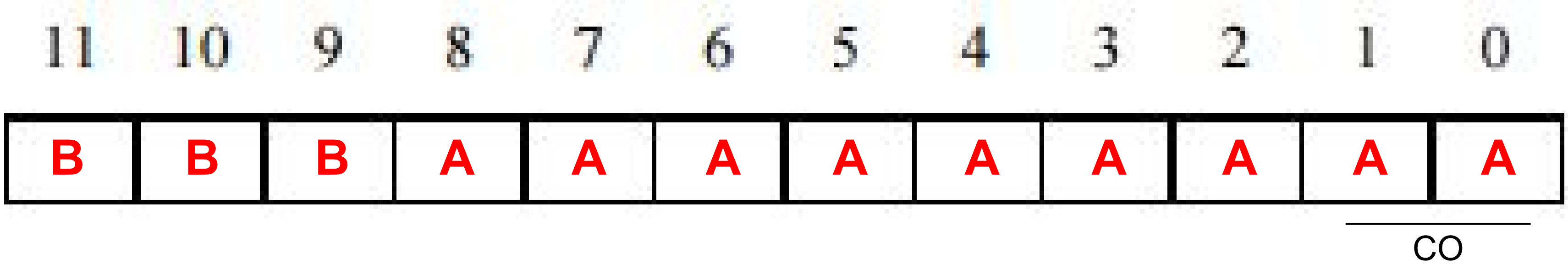
Label the following:

(A) *PPO:* Physical Page Offset - Same as VPO

(B) *PPN:* Physical Page Number - Everything Else

(C) *CO:* Cache Offset - Offset in Block

4 Byte Blocks → 2 Bits



11	10	9	8	7	6	5	4	3	2	1	0
B	B	B	A	A	A	A	A	A	A	A	A

Virtual Memory

Label the following:

(A) *PPO:* Physical Page Offset - Same as VPO

(B) *PPN:* Physical Page Number - Everything Else

(C) *CO:* Cache Offset - Offset in Block

(D) *CI:* Cache Index

11	10	9	8	7	6	5	4	3	2	1	0
B	B	B	A	A	A	A	A	A	A	A	A
										CO	

Virtual Memory

Label the following:

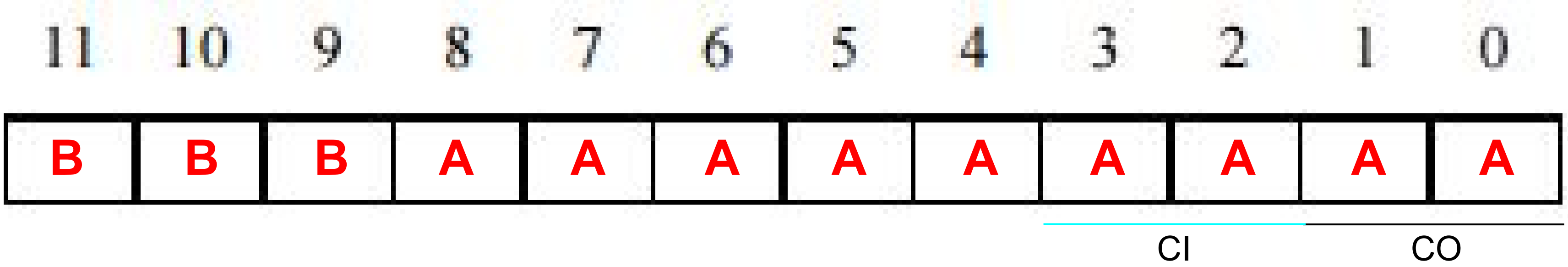
(A) *PPO:* Physical Page Offset - Same as VPO

(B) *PPN:* Physical Page Number - Everything Else

(C) *CO:* Cache Offset - Offset in Block

(D) *CI:* Cache Index

4 Indices → 2 Bits



11	10	9	8	7	6	5	4	3	2	1	0
B	B	B	A	A	A	A	A	A	A	A	A

Virtual Memory

Label the following:

(A) *PPO:* Physical Page Offset - Same as VPO

(B) *PPN:* Physical Page Number - Everything Else

(C) *CO:* Cache Offset - Offset in Block

(D) *CI:* Cache Index

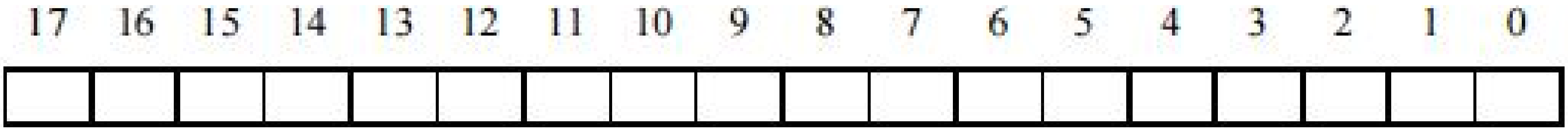
(E) *CT:* Cache Tag - Everything Else

11	10	9	8	7	6	5	4	3	2	1	0
B	B	B	A	A	A	A	A	A	A	A	A
Cache Tag								CI		CO	

Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**



17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation

   1 = 0001 A = 1010 9 = 1001 F = 1111 4 = 0100

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	0	0	1	1	1	1	1	0	1	0	0

Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation
2. Extract Information:

VPN: 0x?? TLBI: 0x?? TLBT: 0x??

TLB Hit: Y/N? Page Fault: Y/N? PPN: 0x??

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	0	0	1	1	1	1	1	0	1	0	0

Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation
2. Extract Information:

VPN: 0xD4 TLBI: 0x?? TLBT: 0x??

TLB Hit: Y/N? Page Fault: Y/N? PPN: 0x??

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	0	0	1	1	1	1	1	0	1	0	0

Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation
2. Extract Information:

VPN: 0xD4 TLBI: 0x00 TLBT: 0x??

TLB Hit: Y/N? Page Fault: Y/N? PPN: 0x??

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	0	0	1	1	1	1	1	0	1	0	0

Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation
2. Extract Information:

VPN: 0xD4 TLBI: 0x00 TLBT: 0x6A

TLB Hit: Y/N? Page Fault: Y/N? PPN: 0x??

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	0	0	1	1	1	1	1	0	1	0	0

Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation
2. Extract Information:

VPN: 0xD4 TLBI: 0x00 TLBT: 0x6A

TLB Hit: Y! Page Fault: Y/N? PPN: 0x??

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	0	0	1	1	1	1	1	0	1	0	0

Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation
2. Extract Information:

VPN: 0xD4 TLBI: 0x00 TLBT: 0x6A

TLB Hit: Y! Page Fault: N! PPN: 0x??

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	0	0	1	1	1	1	1	0	1	0	0

Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation
2. Extract Information:

VPN: 0xD4 TLBI: 0x00 TLBT: 0x6A

TLB Hit: Y! Page Fault: N! PPN: 0x3

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	0	0	1	1	1	1	1	0	1	0	0

Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation
2. Extract Information
3. Put it all together: PPN: 0x3, PPO = 0x??

11	10	9	8	7	6	5	4	3	2	1	0
0	1	1									

Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation
2. Extract Information
3. Put it all together: PPN: 0x3, PPO = VPO = 0x1F4

11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	0	1	0	0

Virtual Memory

Q) **What is the value of the address?**

CO: 0x?? CI: 0x?? CT: 0x?? Cache Hit: Y/N? Value:0x??

11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	0	1	0	0

Virtual Memory

Q) **What is the value of the address?**

1. Extract more information

CO: 0x00 CI: 0x?? CT: 0x?? Cache Hit: Y/N? Value:0x??

11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	0	1	0	0

Virtual Memory

Q) **What is the value of the address?**

1. Extract more information

CO: 0x00 CI: 0x01 CT: 0x?? Cache Hit: Y/N? Value:0x??

11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	0	1	0	0

Virtual Memory

Q) **What is the value of the address?**

1. Extract more information

CO: 0x00 CI: 0x01 CT: 0x7F Cache Hit: Y/N? Value:0x??

11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	0	1	0	0

Virtual Memory

Q) **What is the value of the address?**

1. Extract more information
2. Go to Cache Table

CO: 0x00 CI: 0x01 CT: 0x7F Cache Hit: Y Value:0x??

11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	0	1	0	0

Virtual Memory

Q) **What is the value of the address?**

1. Extract more information
2. Go to Cache Table

CO: 0x00 CI: 0x01 CT: 0x7F Cache Hit: Y Value:0xFF

11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	0	1	0	0

Good luck!