FINAL EXAM REVIEW

Kevin Tew

http://www.cs.utah.edu/~tewk/cs4400-final.pdf

Computer Engineering Senior Project Demo Day

and 3710 Project Demo day is Friday from 10-12

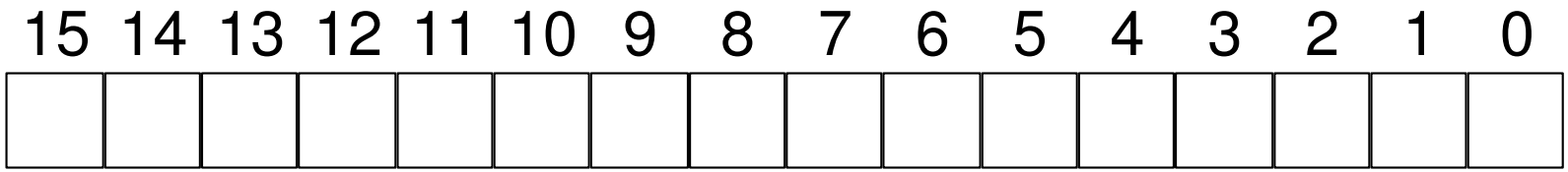
Problem 1-a

1. The following problem concerns the way virtual addresses are translated into physical addresses.

   - The memory is byte addressable.
   - Memory accesses are to **1-byte words** (not 4-byte words).
   - Virtual addresses are 16 bits wide.
   - Physical addresses are 13 bits wide.
   - The page size is 512 bytes.
   - The TLB is 8-way set associative with 16 total entries.
   - The cache is 2-way set associative, with a block size of 4 bytes and a capacity of 64 bytes.

   (a) The box below shows the format of a virtual address. Indicate (by labeling the diagram) the fields that would be used to determine the following.

VPO	The virtual page offset
VPN	The virtual page number
TLBI	The TLB index
TLBT	The TLB tag



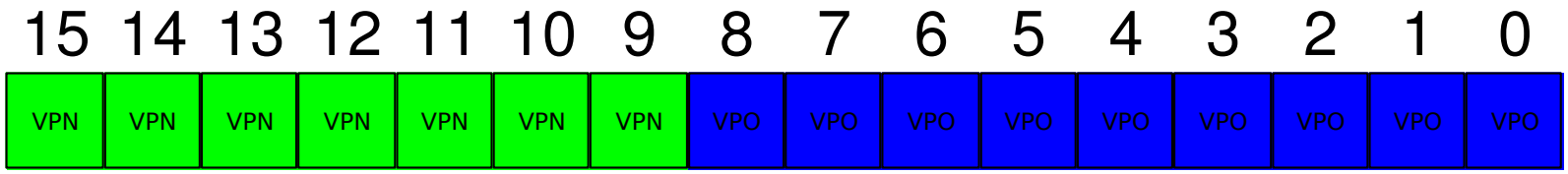
Problem 1-a

1. The following problem concerns the way virtual addresses are translated into physical addresses.

   - The memory is byte addressable.
   - Memory accesses are to **1-byte words** (not 4-byte words).
   - Virtual addresses are 16 bits wide.
   - Physical addresses are 13 bits wide.
   - The page size is 512 bytes.
   - The TLB is 8-way set associative with 16 total entries.
   - The cache is 2-way set associative, with a block size of 4 bytes and a capacity of 64 bytes.

   (a) The box below shows the format of a virtual address. Indicate (by labeling the diagram) the fields that would be used to determine the following.

VPO	The virtual page offset
VPN	The virtual page number
TLBI	The TLB index
TLBT	The TLB tag



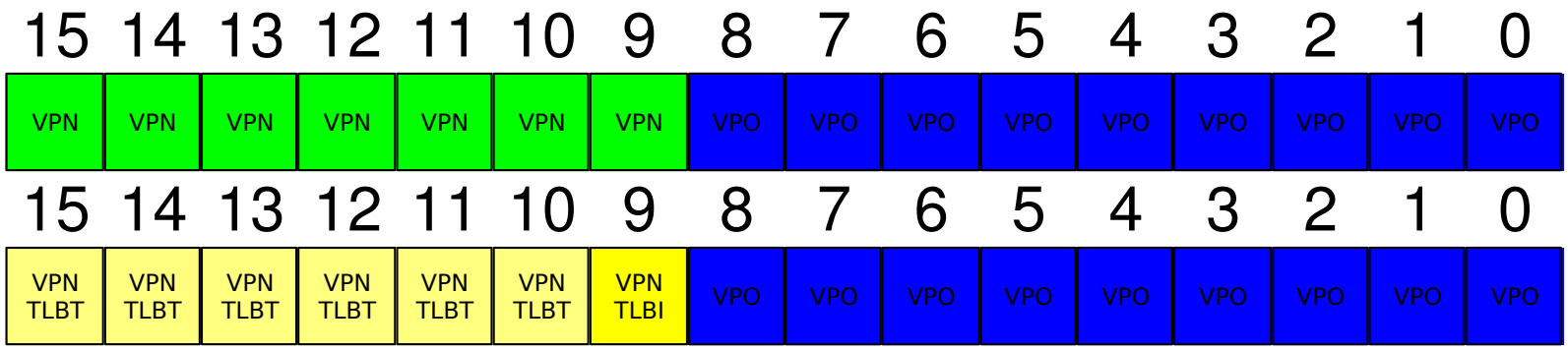
Problem 1-a

1. The following problem concerns the way virtual addresses are translated into physical addresses.

   - The memory is byte addressable.
   - Memory accesses are to **1-byte words** (not 4-byte words).
   - Virtual addresses are 16 bits wide.
   - Physical addresses are 13 bits wide.
   - The page size is 512 bytes.
   - The TLB is 8-way set associative with 16 total entries.
   - The cache is 2-way set associative, with a block size of 4 bytes and a capacity of 64 bytes.

   (a) The box below shows the format of a virtual address. Indicate (by labeling the diagram) the fields that would be used to determine the following.

VPO	The virtual page offset
VPN	The virtual page number
TLBI	The TLB index
TLBT	The TLB tag



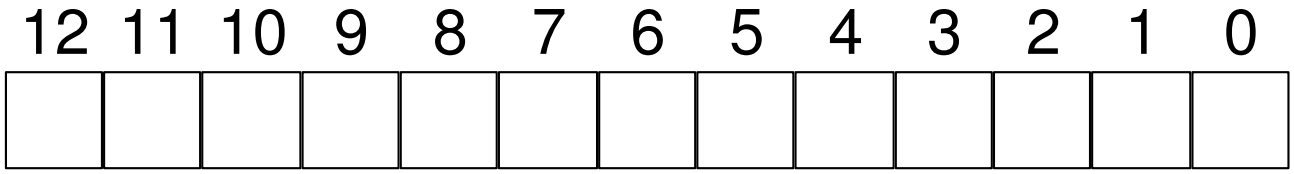
Problem 1-b

1. The following problem concerns the way virtual addresses are translated into physical addresses.

   - The memory is byte addressable.
   - Memory accesses are to **1-byte words** (not 4-byte words).
   - Virtual addresses are 16 bits wide.
   - Physical addresses are 13 bits wide.
   - The page size is 512 bytes.
   - The TLB is 8-way set associative with 16 total entries.
   - The cache is 2-way set associative, with a block size of 4 bytes and a capacity of 64 bytes.

   (b) The box below shows the format of a physical address. Indicate (by labeling the diagram) the fields that would be used to determine the following.

PPO	The physical page offset
PPN	The physical page number
CO	The block offset within the cache line
CI	The cache index
CT	The cache tag



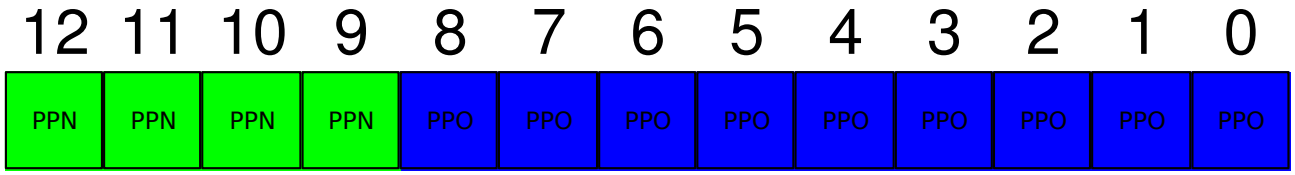
Problem 1-b

1. The following problem concerns the way virtual addresses are translated into physical addresses.

   - The memory is byte addressable.
   - Memory accesses are to **1-byte words** (not 4-byte words).
   - Virtual addresses are 16 bits wide.
   - Physical addresses are 13 bits wide.
   - The page size is 512 bytes.
   - The TLB is 8-way set associative with 16 total entries.
   - The cache is 2-way set associative, with a block size of 4 bytes and a capacity of 64 bytes.

   (b) The box below shows the format of a physical address. Indicate (by labeling the diagram) the fields that would be used to determine the following.

PPO	The physical page offset
PPN	The physical page number
CO	The block offset within the cache line
CI	The cache index
CT	The cache tag



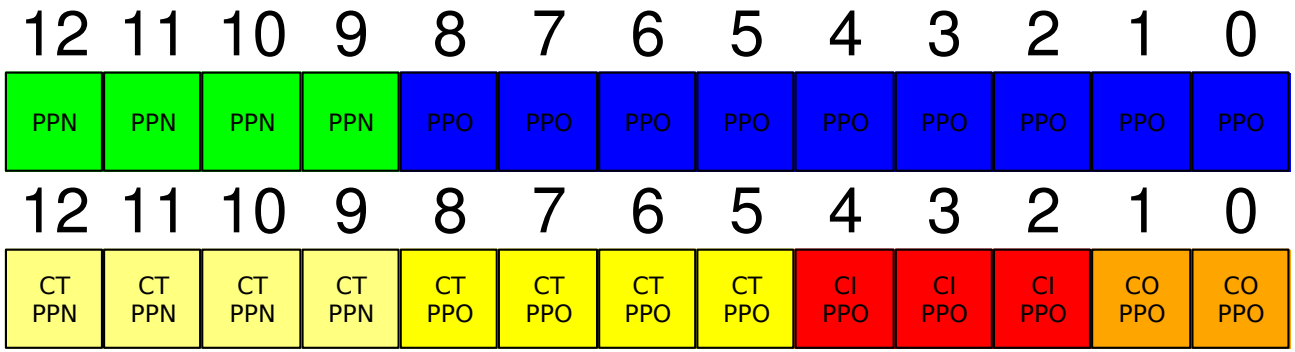
Problem 1-b

1. The following problem concerns the way virtual addresses are translated into physical addresses.

   - The memory is byte addressable.
   - Memory accesses are to **1-byte words** (not 4-byte words).
   - Virtual addresses are 16 bits wide.
   - Physical addresses are 13 bits wide.
   - The page size is 512 bytes.
   - The TLB is 8-way set associative with 16 total entries.
   - The cache is 2-way set associative, with a block size of 4 bytes and a capacity of 64 bytes.

   (b) The box below shows the format of a physical address. Indicate (by labeling the diagram) the fields that would be used to determine the following.

PPO	The physical page offset
PPN	The physical page number
CO	The block offset within the cache line
CI	The cache index
CT	The cache tag

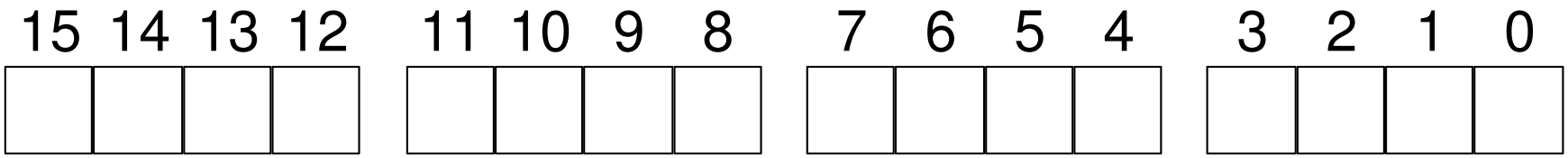


Problem 1-c

(c) Give the format of the virtual address.

Virtual address: `31DE`

Virtual address format (one bit per box)

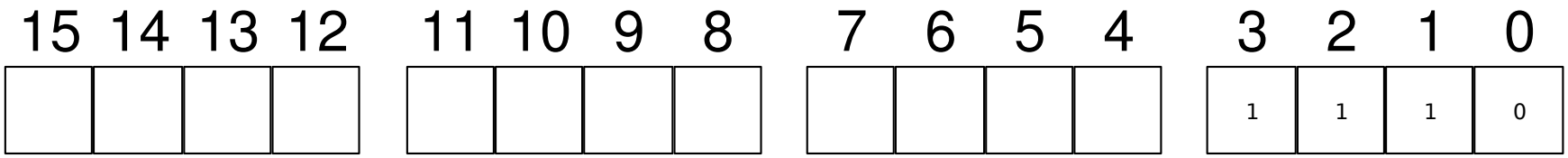


Problem 1-c

(c) Give the format of the virtual address.

Virtual address: `31DE`

Virtual address format (one bit per box)

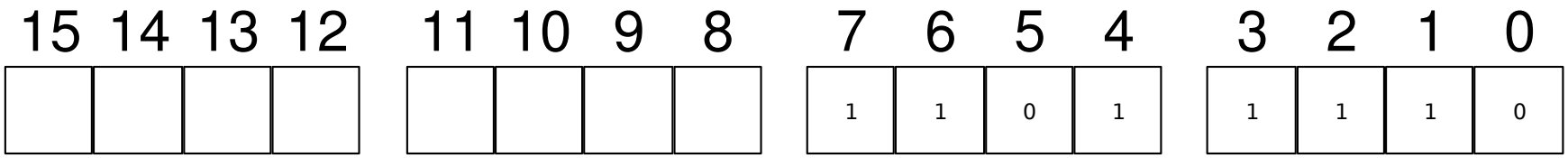


Problem 1-c

(c) Give the format of the virtual address.

Virtual address: `31DE`

Virtual address format (one bit per box)

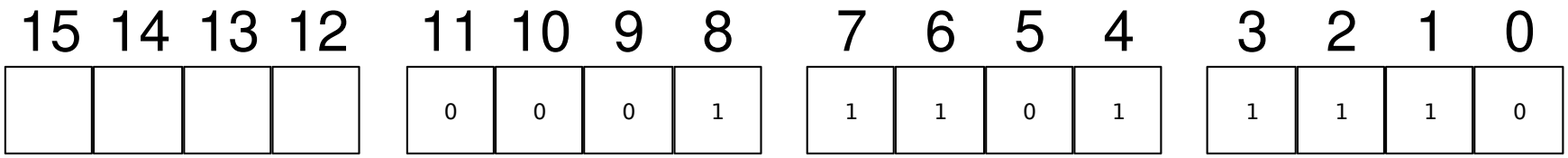


Problem 1-c

(c) Give the format of the virtual address.

Virtual address: `31DE`

Virtual address format (one bit per box)

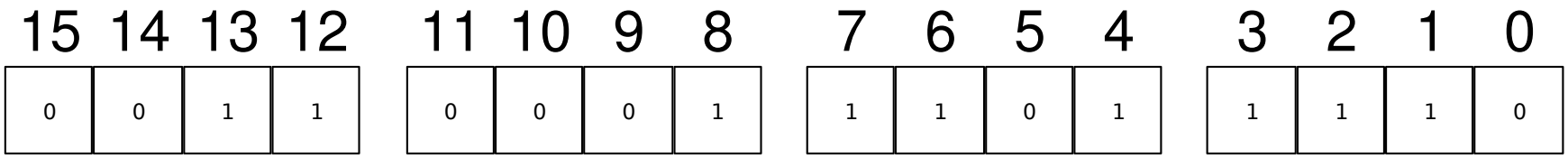


Problem 1-c

(c) Give the format of the virtual address.

Virtual address: `31DE`

Virtual address format (one bit per box)

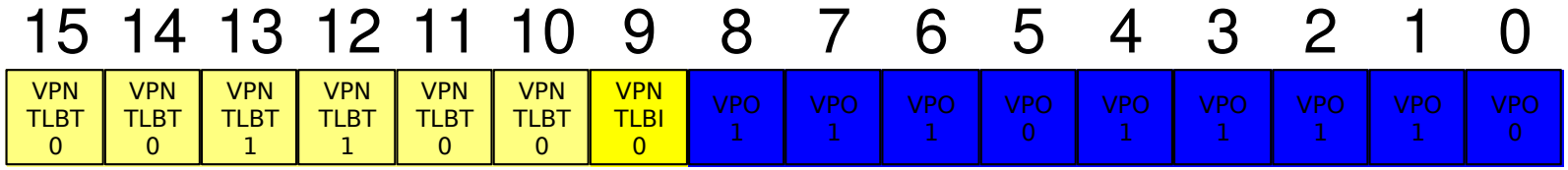


Problem 1-d

(d) For the given virtual address, fill in the following table. If there is a page fault, enter "–" for "PPN".

Address translation

Parameter	Value
VPN	0x
TLB Index	0x
TLB Tag	0x
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	0x

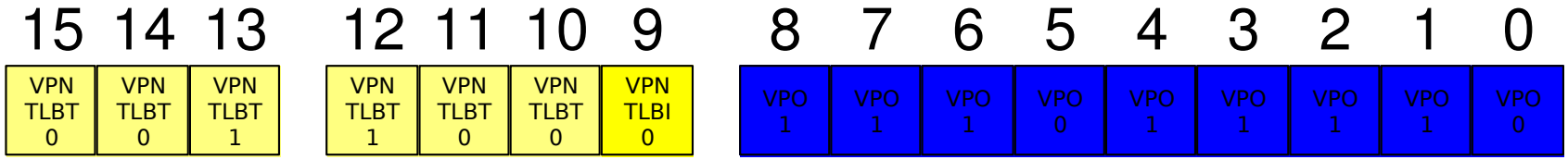


Problem 1-d

(d) For the given virtual address, fill in the following table. If there is a page fault, enter "–" for "PPN".

Address translation

Parameter	Value
VPN	0x
TLB Index	0x
TLB Tag	0x
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	0x

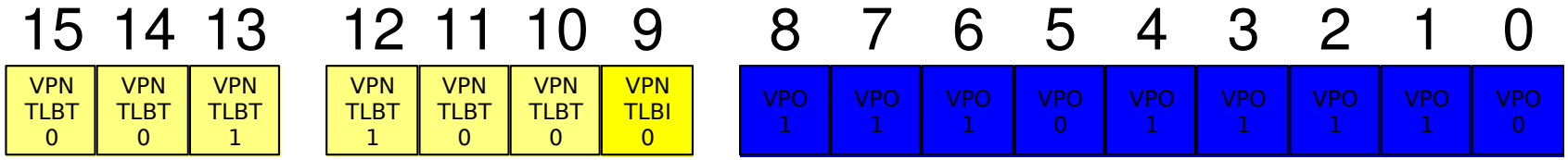


Problem 1-d

(d) For the given virtual address, fill in the following table. If there is a page fault, enter "–" for "PPN".

Address translation

Parameter	Value
VPN	0x 18
TLB Index	0x
TLB Tag	0x
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	0x

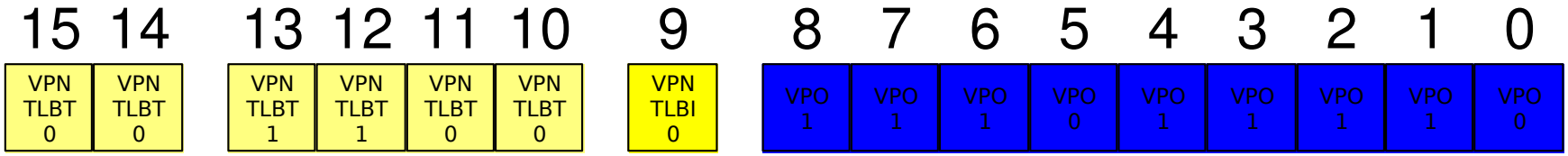


Problem 1-d

(d) For the given virtual address, fill in the following table. If there is a page fault, enter "–" for "PPN".

Address translation

Parameter	Value
VPN	0x 18
TLB Index	0x
TLB Tag	0x
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	0x

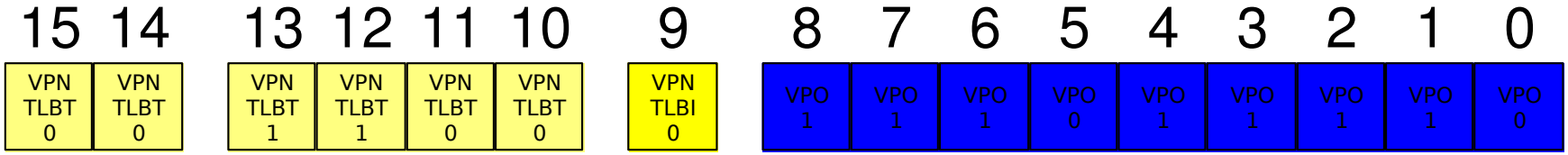


Problem 1-d

(d) For the given virtual address, fill in the following table. If there is a page fault, enter "–" for "PPN".

Address translation

Parameter	Value
VPN	0x 18
TLB Index	0x 0
TLB Tag	0x
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	0x

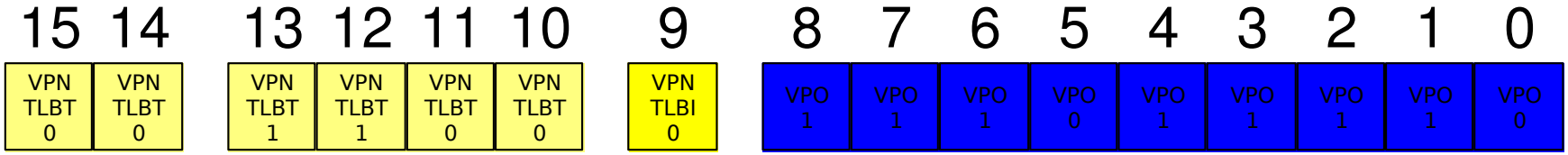


Problem 1-d

(d) For the given virtual address, fill in the following table. If there is a page fault, enter "–" for "PPN".

Address translation

Parameter	Value
VPN	0x 18
TLB Index	0x 0
TLB Tag	0x 0C
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	0x



Problem 1-d

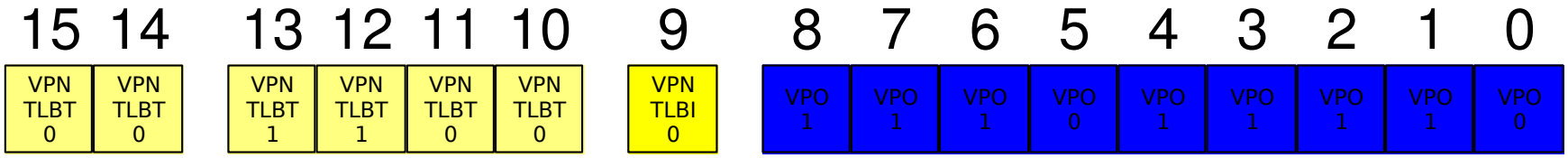
TLB			
Index	Tag	PPN	Valid
0	09	4	1
	12	2	1
	10	0	1
	08	5	1
	05	7	1
	13	1	0
	10	3	0
	18	3	0
1	04	1	0
	0C	1	0
	12	0	0
	08	1	0
	06	7	0
	03	1	0
	07	5	0
	02	2	0

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	6	1	10	0	1
01	5	0	11	5	0
02	3	1	12	2	1
03	4	1	13	4	0
04	2	0	14	6	0
05	7	1	15	2	0
06	1	0	16	4	0
07	3	0	17	6	0
08	5	1	18	1	1
09	4	0	19	2	0
0A	3	0	1A	5	0
0B	2	0	1B	7	0
0C	5	0	1C	6	0
0D	6	0	1D	2	0
0E	1	1	1E	3	0
0F	0	0	1F	1	0

(d) For the given virtual address, fill in the following table. If there is a page fault, enter "–" for "PPN".

Address translation

Parameter	Value
VPN	0x 18
TLB Index	0x 0
TLB Tag	0x 0C
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	0x



Problem 1-d

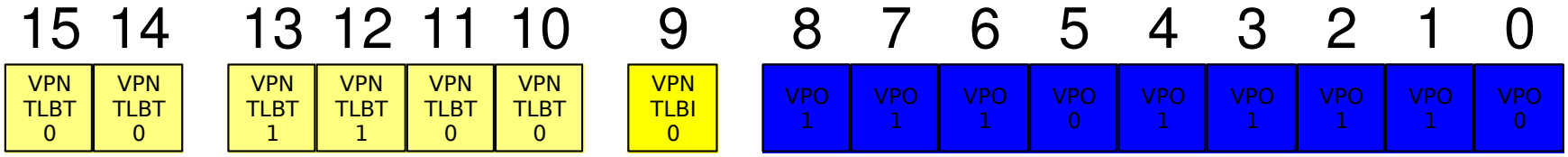
TLB			
Index	Tag	PPN	Valid
0	09	4	1
	12	2	1
	10	0	1
	08	5	1
	05	7	1
	13	1	0
	10	3	0
	18	3	0
1	04	1	0
	0C	1	0
	12	0	0
	08	1	0
	06	7	0
	03	1	0
	07	5	0
	02	2	0

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	6	1	10	0	1
01	5	0	11	5	0
02	3	1	12	2	1
03	4	1	13	4	0
04	2	0	14	6	0
05	7	1	15	2	0
06	1	0	16	4	0
07	3	0	17	6	0
08	5	1	18	1	1
09	4	0	19	2	0
0A	3	0	1A	5	0
0B	2	0	1B	7	0
0C	5	0	1C	6	0
0D	6	0	1D	2	0
0E	1	1	1E	3	0
0F	0	0	1F	1	0

(d) For the given virtual address, fill in the following table. If there is a page fault, enter "–" for "PPN".

Address translation

Parameter	Value
VPN	0x 18
TLB Index	0x 0
TLB Tag	0x 0C
TLB Hit? (Y/N)	N
Page Fault? (Y/N)	
PPN	0x



Problem 1-d

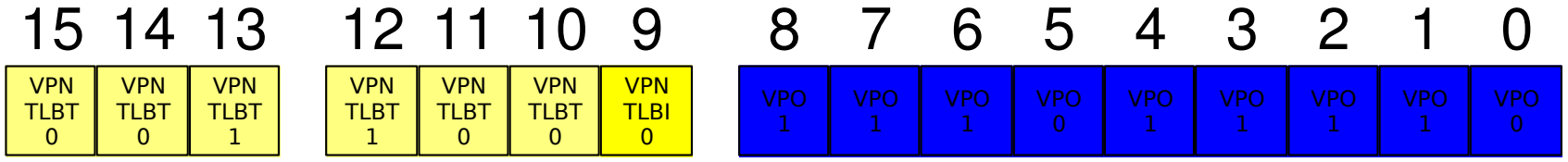
TLB			
Index	Tag	PPN	Valid
0	09	4	1
	12	2	1
	10	0	1
	08	5	1
	05	7	1
	13	1	0
	10	3	0
	18	3	0
1	04	1	0
	0C	1	0
	12	0	0
	08	1	0
	06	7	0
	03	1	0
	07	5	0
	02	2	0

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	6	1	10	0	1
01	5	0	11	5	0
02	3	1	12	2	1
03	4	1	13	4	0
04	2	0	14	6	0
05	7	1	15	2	0
06	1	0	16	4	0
07	3	0	17	6	0
08	5	1	18	1	1
09	4	0	19	2	0
0A	3	0	1A	5	0
0B	2	0	1B	7	0
0C	5	0	1C	6	0
0D	6	0	1D	2	0
0E	1	1	1E	3	0
0F	0	0	1F	1	0

(d) For the given virtual address, fill in the following table. If there is a page fault, enter "–" for "PPN".

Address translation

Parameter	Value
VPN	0x 18
TLB Index	0x 0
TLB Tag	0x 0C
TLB Hit? (Y/N)	N
Page Fault? (Y/N)	
PPN	0x



Problem 1-d

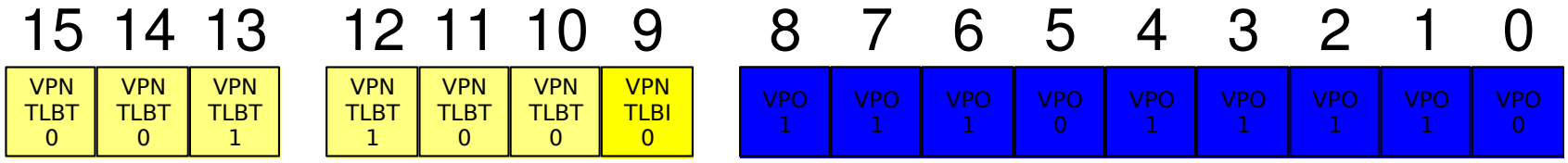
TLB			
Index	Tag	PPN	Valid
0	09	4	1
	12	2	1
	10	0	1
	08	5	1
	05	7	1
	13	1	0
	10	3	0
	18	3	0
1	04	1	0
	0C	1	0
	12	0	0
	08	1	0
	06	7	0
	03	1	0
	07	5	0
	02	2	0

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	6	1	10	0	1
01	5	0	11	5	0
02	3	1	12	2	1
03	4	1	13	4	0
04	2	0	14	6	0
05	7	1	15	2	0
06	1	0	16	4	0
07	3	0	17	6	0
08	5	1	18	1	1
09	4	0	19	2	0
0A	3	0	1A	5	0
0B	2	0	1B	7	0
0C	5	0	1C	6	0
0D	6	0	1D	2	0
0E	1	1	1E	3	0
0F	0	0	1F	1	0

(d) For the given virtual address, fill in the following table. If there is a page fault, enter "–" for "PPN".

Address translation

Parameter	Value
VPN	0x 18
TLB Index	0x 0
TLB Tag	0x 0C
TLB Hit? (Y/N)	N
Page Fault? (Y/N)	N
PPN	0x



Problem 1-d

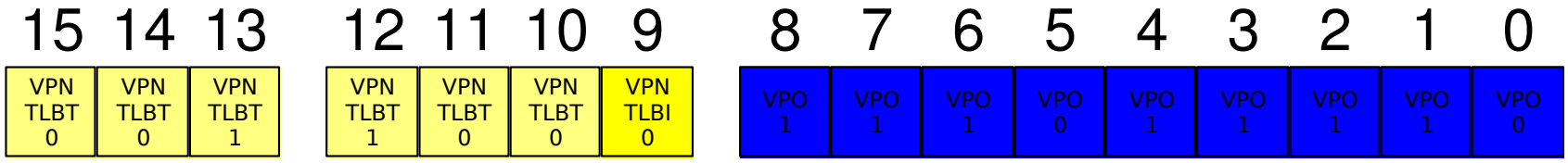
TLB			
Index	Tag	PPN	Valid
0	09	4	1
	12	2	1
	10	0	1
	08	5	1
	05	7	1
	13	1	0
	10	3	0
	18	3	0
1	04	1	0
	0C	1	0
	12	0	0
	08	1	0
	06	7	0
	03	1	0
	07	5	0
	02	2	0

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	6	1	10	0	1
01	5	0	11	5	0
02	3	1	12	2	1
03	4	1	13	4	0
04	2	0	14	6	0
05	7	1	15	2	0
06	1	0	16	4	0
07	3	0	17	6	0
08	5	1	18	1	1
09	4	0	19	2	0
0A	3	0	1A	5	0
0B	2	0	1B	7	0
0C	5	0	1C	6	0
0D	6	0	1D	2	0
0E	1	1	1E	3	0
0F	0	0	1F	1	0

(d) For the given virtual address, fill in the following table. If there is a page fault, enter "–" for "PPN".

Address translation

Parameter	Value
VPN	0x 18
TLB Index	0x 0
TLB Tag	0x 0C
TLB Hit? (Y/N)	N
Page Fault? (Y/N)	N
PPN	0x 1

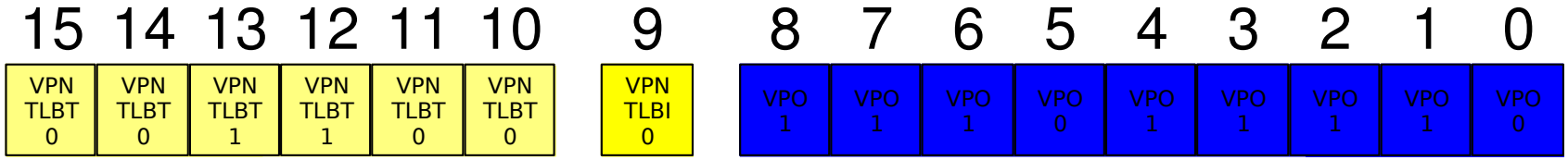


Problem 1-e

(d) For the given virtual address, fill in the following table. If there is a page fault, enter "–" for "PPN".

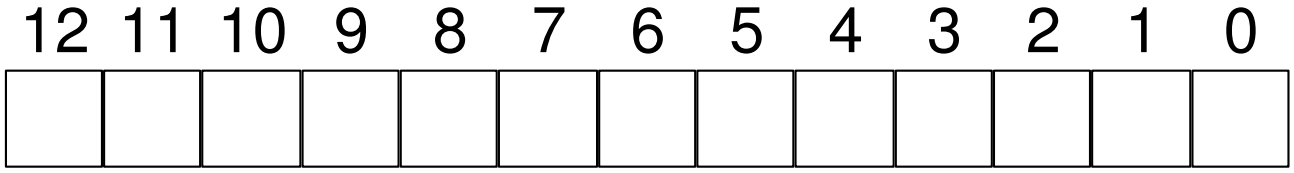
Address translation

Parameter	Value
VPN	0x 18
TLB Index	0x 0
TLB Tag	0x 0C
TLB Hit? (Y/N)	N
Page Fault? (Y/N)	N
PPN	0x 1



(e) If there is a page fault, leave this part blank. Otherwise, indicate the format of the physical address.

Physical address format (one bit per box)

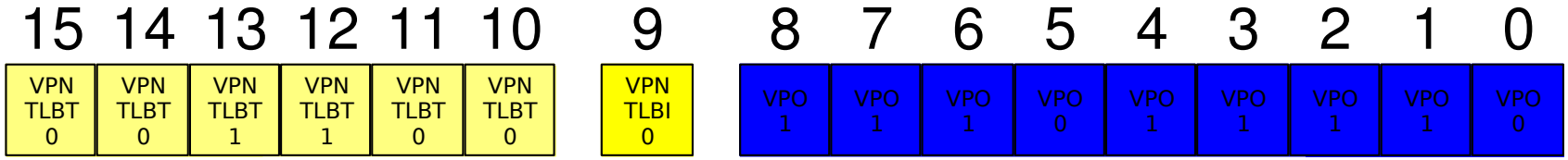


Problem 1-e

(d) For the given virtual address, fill in the following table. If there is a page fault, enter "–" for "PPN".

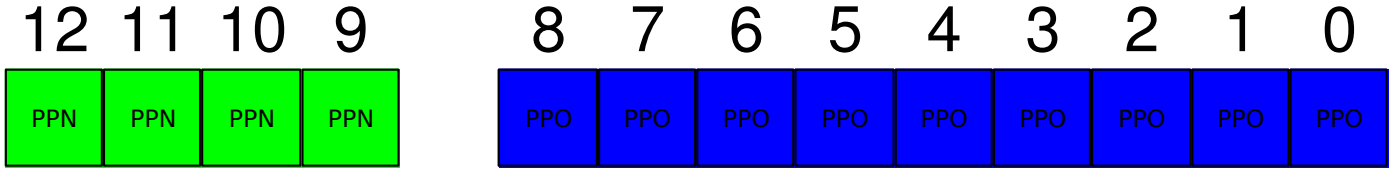
Address translation

Parameter	Value
VPN	0x 18
TLB Index	0x 0
TLB Tag	0x 0C
TLB Hit? (Y/N)	N
Page Fault? (Y/N)	N
PPN	0x 1



(e) If there is a page fault, leave this part blank. Otherwise, indicate the format of the physical address.

Physical address format (one bit per box)

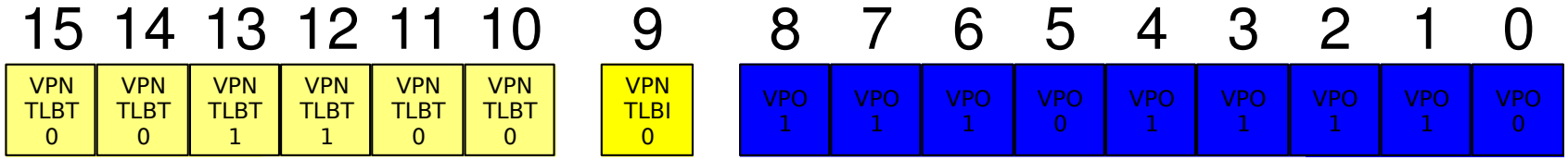


Problem 1-e

(d) For the given virtual address, fill in the following table. If there is a page fault, enter "–" for "PPN".

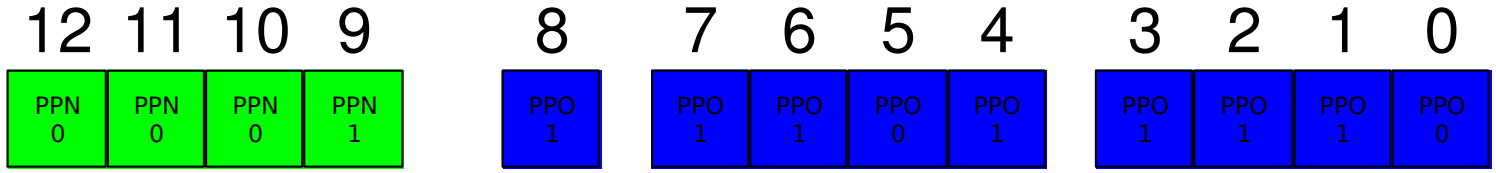
Address translation

Parameter	Value
VPN	0x 18
TLB Index	0x 0
TLB Tag	0x 0C
TLB Hit? (Y/N)	N
Page Fault? (Y/N)	N
PPN	0x 1



(e) If there is a page fault, leave this part blank. Otherwise, indicate the format of the physical address.

Physical address format (one bit per box)



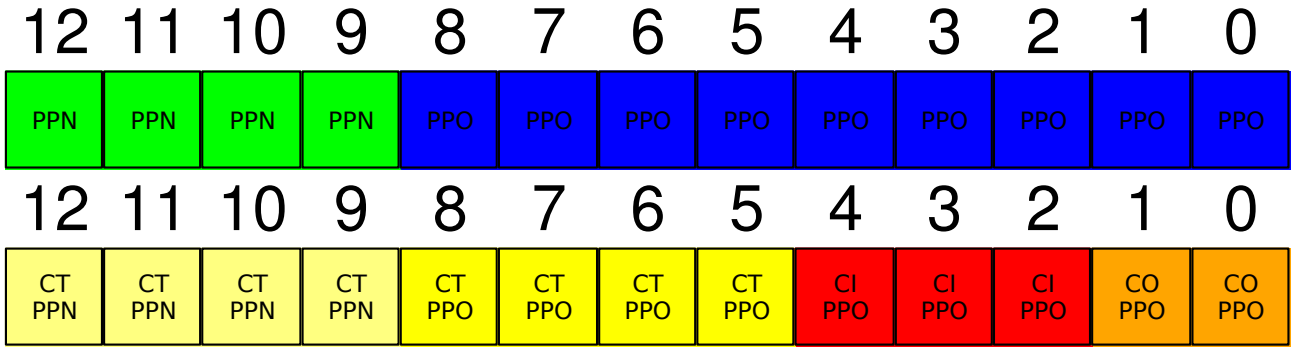
Problem 1-f

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	19	1	99	11	23	11	00	0	99	11	23	11
1	15	0	4F	22	EC	11	2F	1	55	59	0B	41
2	1B	1	00	02	04	08	0B	1	01	03	05	07
3	06	0	84	06	B2	9C	12	0	84	06	B2	9C
4	07	0	43	6D	8F	09	05	0	43	6D	8F	09
5	0D	1	36	32	00	78	1E	1	A1	B2	C4	DE
6	11	0	A2	37	68	31	00	1	BB	77	33	00
7	16	1	11	C2	11	33	1E	1	00	C0	0F	00

(f) If there is a page fault, leave this part blank. Otherwise, fill in the following table. If there is a cache miss, enter "–" for "Cache Byte returned".

Physical memory reference

Parameter	Value
Byte offset	0x
Cache Index	0x
Cache Tag	0x
Cache Hit? (Y/N)	
Cache Byte returned	0x



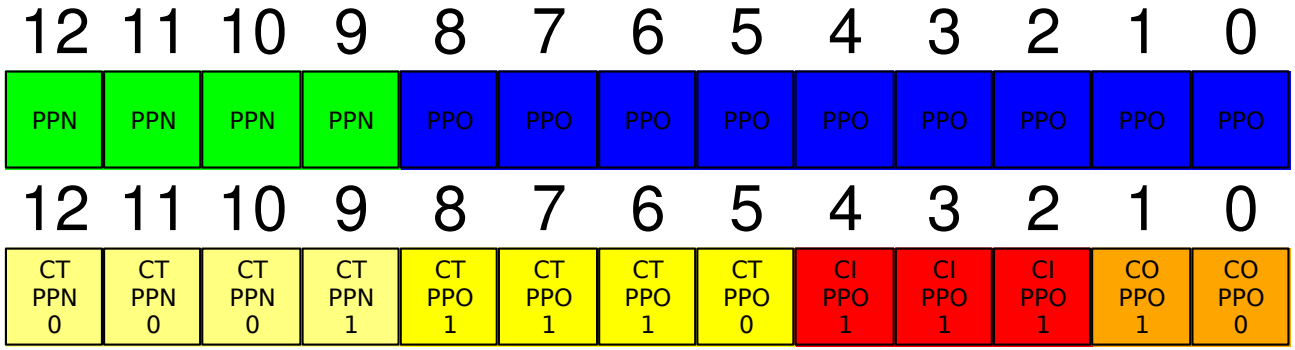
Problem 1-f

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	19	1	99	11	23	11	00	0	99	11	23	11
1	15	0	4F	22	EC	11	2F	1	55	59	0B	41
2	1B	1	00	02	04	08	0B	1	01	03	05	07
3	06	0	84	06	B2	9C	12	0	84	06	B2	9C
4	07	0	43	6D	8F	09	05	0	43	6D	8F	09
5	0D	1	36	32	00	78	1E	1	A1	B2	C4	DE
6	11	0	A2	37	68	31	00	1	BB	77	33	00
7	16	1	11	C2	11	33	1E	1	00	C0	0F	00

(f) If there is a page fault, leave this part blank. Otherwise, fill in the following table. If there is a cache miss, enter "–" for "Cache Byte returned".

Physical memory reference

Parameter	Value
Byte offset	0x
Cache Index	0x
Cache Tag	0x
Cache Hit? (Y/N)	
Cache Byte returned	0x



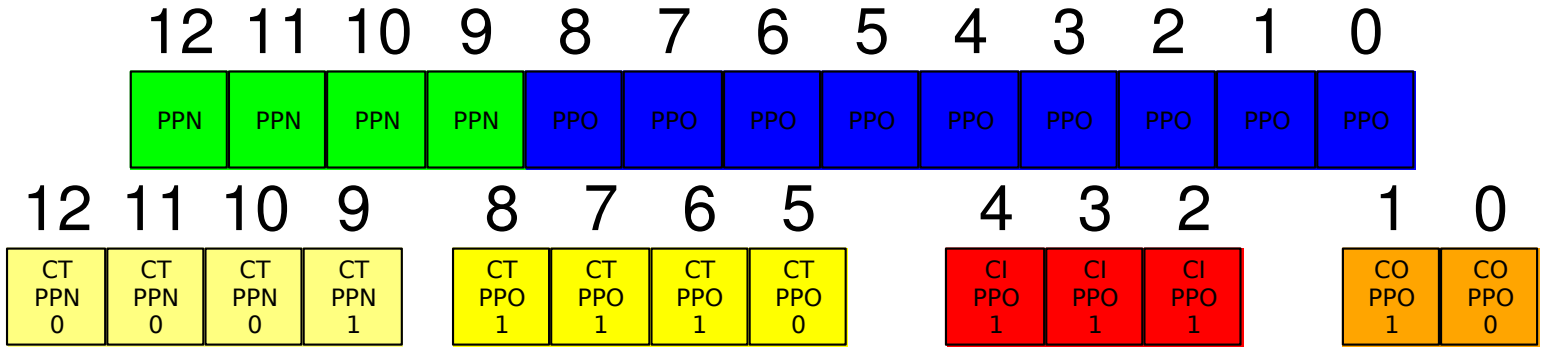
Problem 1-f

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	19	1	99	11	23	11	00	0	99	11	23	11
1	15	0	4F	22	EC	11	2F	1	55	59	0B	41
2	1B	1	00	02	04	08	0B	1	01	03	05	07
3	06	0	84	06	B2	9C	12	0	84	06	B2	9C
4	07	0	43	6D	8F	09	05	0	43	6D	8F	09
5	0D	1	36	32	00	78	1E	1	A1	B2	C4	DE
6	11	0	A2	37	68	31	00	1	BB	77	33	00
7	16	1	11	C2	11	33	1E	1	00	C0	0F	00

(f) If there is a page fault, leave this part blank. Otherwise, fill in the following table. If there is a cache miss, enter "–" for "Cache Byte returned".

Physical memory reference

Parameter	Value
Byte offset	0x
Cache Index	0x
Cache Tag	0x
Cache Hit? (Y/N)	
Cache Byte returned	0x



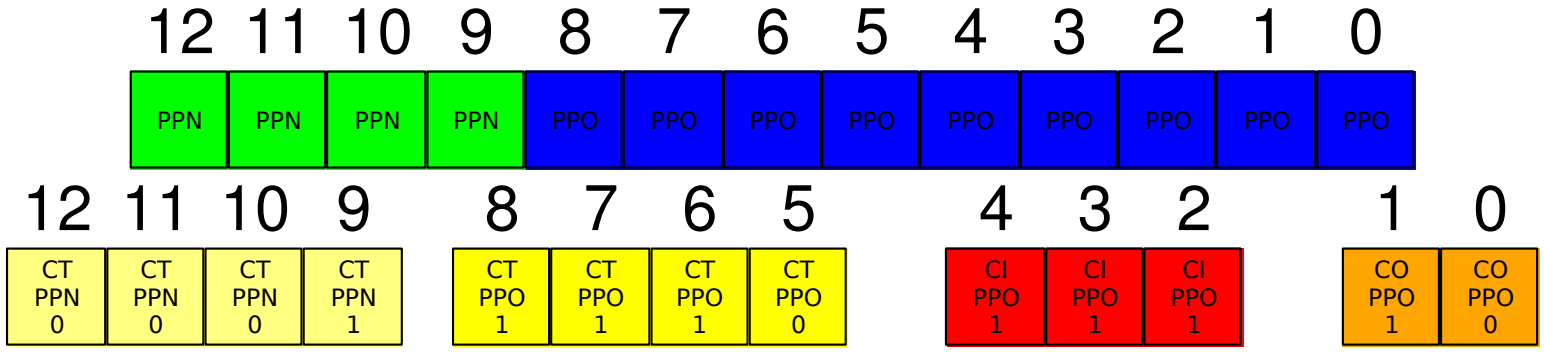
Problem 1-f

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	19	1	99	11	23	11	00	0	99	11	23	11
1	15	0	4F	22	EC	11	2F	1	55	59	0B	41
2	1B	1	00	02	04	08	0B	1	01	03	05	07
3	06	0	84	06	B2	9C	12	0	84	06	B2	9C
4	07	0	43	6D	8F	09	05	0	43	6D	8F	09
5	0D	1	36	32	00	78	1E	1	A1	B2	C4	DE
6	11	0	A2	37	68	31	00	1	BB	77	33	00
7	16	1	11	C2	11	33	1E	1	00	C0	0F	00

(f) If there is a page fault, leave this part blank. Otherwise, fill in the following table. If there is a cache miss, enter "–" for "Cache Byte returned".

Physical memory reference

Parameter	Value
Byte offset	0x2
Cache Index	0x
Cache Tag	0x
Cache Hit? (Y/N)	
Cache Byte returned	0x



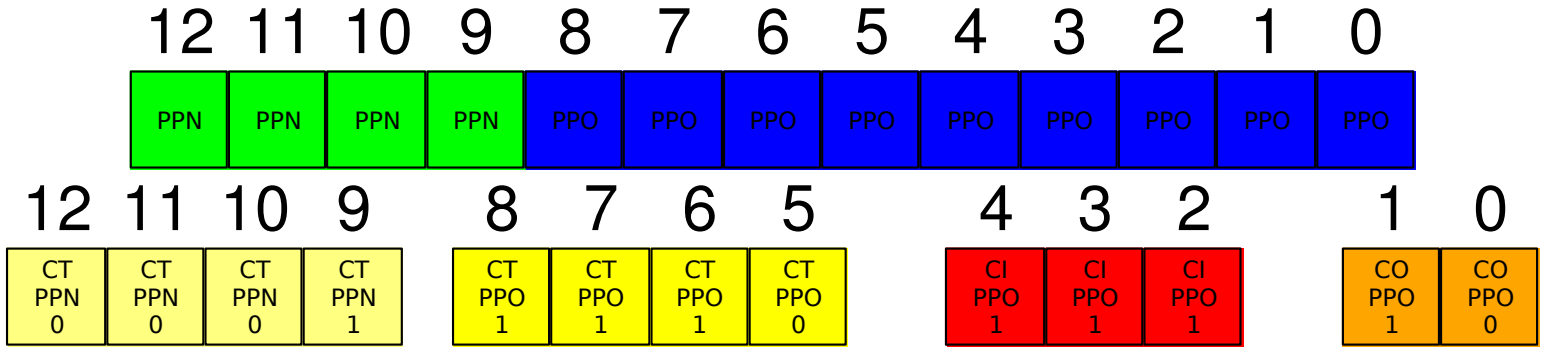
Problem 1-f

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	19	1	99	11	23	11	00	0	99	11	23	11
1	15	0	4F	22	EC	11	2F	1	55	59	0B	41
2	1B	1	00	02	04	08	0B	1	01	03	05	07
3	06	0	84	06	B2	9C	12	0	84	06	B2	9C
4	07	0	43	6D	8F	09	05	0	43	6D	8F	09
5	0D	1	36	32	00	78	1E	1	A1	B2	C4	DE
6	11	0	A2	37	68	31	00	1	BB	77	33	00
7	16	1	11	C2	11	33	1E	1	00	C0	0F	00

(f) If there is a page fault, leave this part blank. Otherwise, fill in the following table. If there is a cache miss, enter "–" for "Cache Byte returned".

Physical memory reference

Parameter	Value
Byte offset	0x2
Cache Index	0x7
Cache Tag	0x
Cache Hit? (Y/N)	
Cache Byte returned	0x



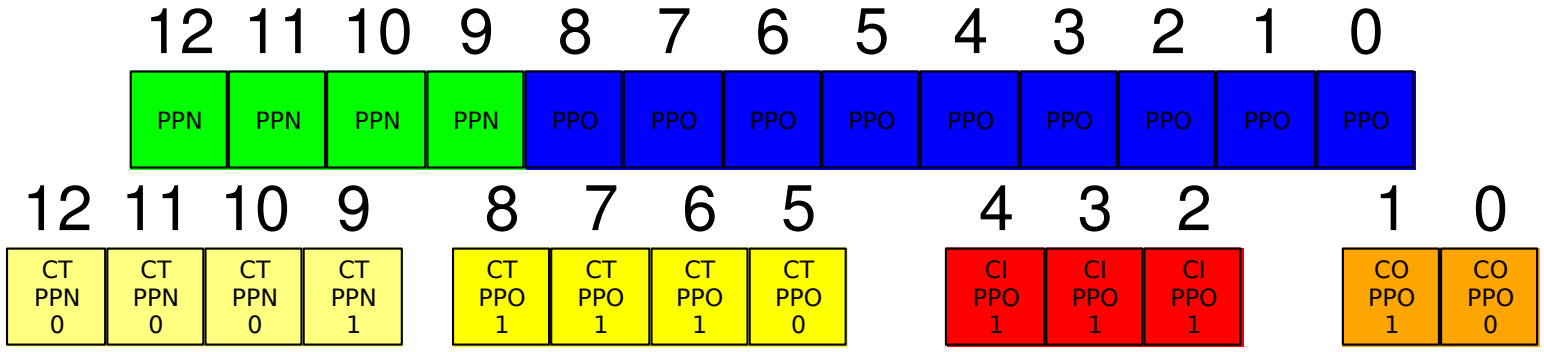
Problem 1-f

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	19	1	99	11	23	11	00	0	99	11	23	11
1	15	0	4F	22	EC	11	2F	1	55	59	0B	41
2	1B	1	00	02	04	08	0B	1	01	03	05	07
3	06	0	84	06	B2	9C	12	0	84	06	B2	9C
4	07	0	43	6D	8F	09	05	0	43	6D	8F	09
5	0D	1	36	32	00	78	1E	1	A1	B2	C4	DE
6	11	0	A2	37	68	31	00	1	BB	77	33	00
7	16	1	11	C2	11	33	1E	1	00	C0	0F	00

(f) If there is a page fault, leave this part blank. Otherwise, fill in the following table. If there is a cache miss, enter "–" for "Cache Byte returned".

Physical memory reference

Parameter	Value
Byte offset	0x2
Cache Index	0x7
Cache Tag	0x1E
Cache Hit? (Y/N)	
Cache Byte returned	0x



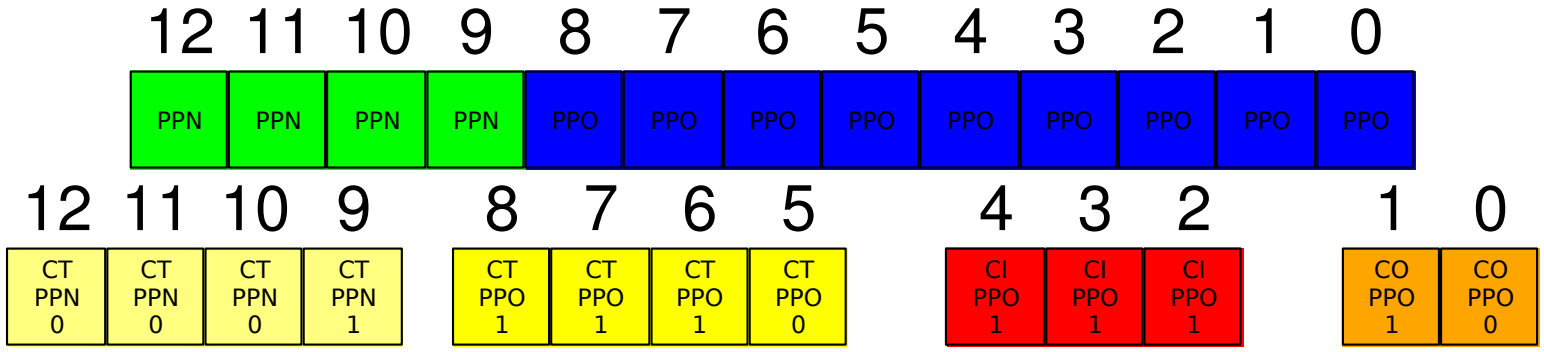
Problem 1-f

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	19	1	99	11	23	11	00	0	99	11	23	11
1	15	0	4F	22	EC	11	2F	1	55	59	0B	41
2	1B	1	00	02	04	08	0B	1	01	03	05	07
3	06	0	84	06	B2	9C	12	0	84	06	B2	9C
4	07	0	43	6D	8F	09	05	0	43	6D	8F	09
5	0D	1	36	32	00	78	1E	1	A1	B2	C4	DE
6	11	0	A2	37	68	31	00	1	BB	77	33	00
7	16	1	11	C2	11	33	1E	1	00	C0	0F	00

(f) If there is a page fault, leave this part blank. Otherwise, fill in the following table. If there is a cache miss, enter "–" for "Cache Byte returned".

Physical memory reference

Parameter	Value
Byte offset	0x2
Cache Index	0x7
Cache Tag	0x1E
Cache Hit? (Y/N)	Y
Cache Byte returned	0x



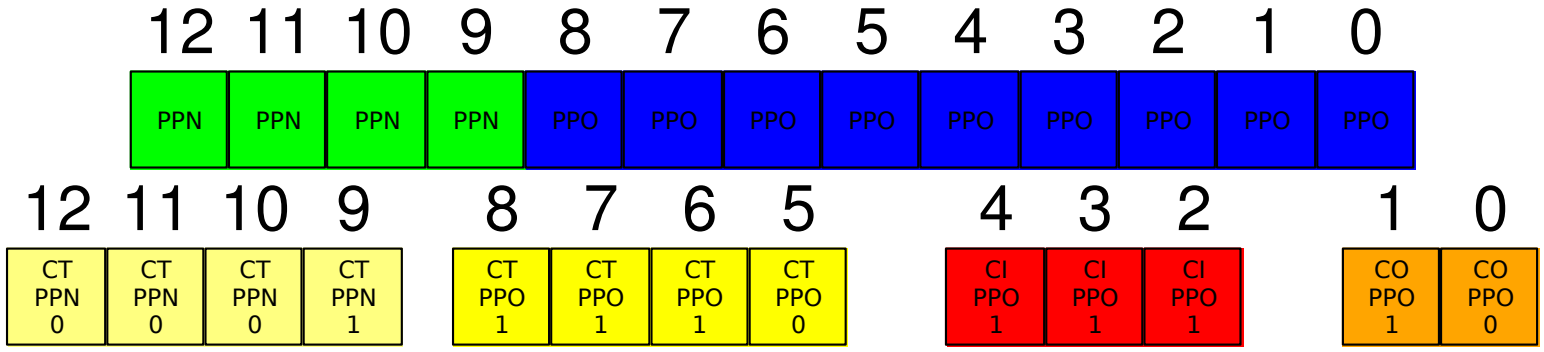
Problem 1-f

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	19	1	99	11	23	11	00	0	99	11	23	11
1	15	0	4F	22	EC	11	2F	1	55	59	0B	41
2	1B	1	00	02	04	08	0B	1	01	03	05	07
3	06	0	84	06	B2	9C	12	0	84	06	B2	9C
4	07	0	43	6D	8F	09	05	0	43	6D	8F	09
5	0D	1	36	32	00	78	1E	1	A1	B2	C4	DE
6	11	0	A2	37	68	31	00	1	BB	77	33	00
7	16	1	11	C2	11	33	1E	1	00	C0	0F	00

(f) If there is a page fault, leave this part blank. Otherwise, fill in the following table. If there is a cache miss, enter "–" for "Cache Byte returned".

Physical memory reference

Parameter	Value
Byte offset	0x2
Cache Index	0x7
Cache Tag	0x1E
Cache Hit? (Y/N)	Y
Cache Byte returned	0x0F



Problem 2

addr	val	210	N	P	C	block size
400b030	00000017	111	1	1	1	00000010
400b02c	00000002	010	0	1	0	00000000
400b028	00000001	001	0	0	1	00000000
400b024	00000017	111	1	1	1	00000010
400b020	0000001d	101	1	0	1	00000018
400b01c	fffffffc	100	1	0	0	fffffff8
400b018	fffffffd	101	1	0	1	fffffff8
400b014	fffffffe	110	1	1	0	fffffff8
400b010	ffffffff	111	1	1	1	fffffff8
400b00c	0000001d	101	1	0	1	00000018
400b008	00000016	110	1	1	0	00000010
400b004	200b601c	100	1	0	0	200b6018
400b000	800b522c	100	1	0	0	800b5228
400affc	00000016	110	1	1	0	00000010

Problem 2

addr	val	210	N	P	C	block size
400b030	00000017	111	1	1	1	00000010
400b02c	00000002	010	0	1	0	00000000
400b028	00000001	001	0	0	1	00000000
400b024	00000017	111	1	1	1	00000010
400b020	0000001d	101	1	0	1	00000018
400b01c	fffffffc	100	1	0	0	fffffff8
400b018	fffffffd	101	1	0	1	fffffff8
400b014	fffffffe	110	1	1	0	fffffff8
400b010	ffffffff	111	1	1	1	fffffff8
400b00c	0000001d	101	1	0	1	00000018
400b008	00000016	110	1	1	0	00000010
400b004	200b601c	100	1	0	0	200b6018
400b000	800b522c	100	1	0	0	800b5228
400affc	00000016	110	1	1	0	00000010

Problem 2

addr	val	210	N	P	C	block size
400b030	00000017	111	1	1	1	00000010
400b02c	00000002	010	0	1	0	00000000
400b028	00000001	001	0	0	1	00000000
400b024	00000017	111	1	1	1	00000010
400b020	0000001d	101	1	0	1	00000018
400b01c	fffffffc	100	1	0	0	fffffff8
400b018	fffffffd	101	1	0	1	fffffff8
400b014	fffffffe	110	1	1	0	fffffff8
400b010	ffffffff	111	1	1	1	fffffff8
400b00c	0000001d	101	1	0	1	00000018
400b008	00000016	110	1	1	0	00000010
400b004	200b601c	100	1	0	0	200b6018
400b000	800b522c	100	1	0	0	800b5228
400affc	00000016	110	1	1	0	00000010

addr	val	210	N	P	C	block size
400b030	00000015	101	1	0	1	00000010
400b02c	00000002	010	0	1	0	00000000
400b028	00000001	001	0	0	1	00000000
400b024	00000015	101	1	0	1	00000010
400b020	0000002e	110	1	1	0	00000028
400b01c	fffffffc	100	1	0	0	fffffff8
400b018	fffffffd	101	1	0	1	fffffff8
400b014	fffffffe	110	1	1	0	fffffff8
400b010	ffffffff	111	1	1	1	fffffff8
400b00c	0000001d	101	1	0	1	00000018
400b008	00000016	110	1	1	0	00000010
400b004	200b601c	100	1	0	0	200b6018
400b000	800b522c	100	1	0	0	800b5228
400affc	0000002e	110	1	1	0	00000028

Problem 3-a

3. Consider an allocator that uses an implicit free list. Each memory block, either allocated or free, has a size that is a multiple of eight bytes. Thus, only the 29 higher order bits in the header and footer are needed to record block size, which includes the header and footer and is represented in units of bytes. The usage of the remaining 3 lower order bits is as follows:

   - `bit 0` indicates the use of the current block: 1 for allocated, 0 for free.
   - `bit 1` indicates the use of the previous block: 1 for allocated, 0 for free.
   - `bit 2` indicates the use of the next block: 1 for allocated, 0 for free.

```
/* Given a pointer to a valid block header or footer, returns the size of the
   block (number of bytes). */
int size(void* hp) {
  return (*hp) & (~0x7);    /* Correct? */
}
```

Circle one: Yes or No

Rewritten line of code: _____

Problem 3-a

3. Consider an allocator that uses an implicit free list. Each memory block, either allocated or free, has a size that is a multiple of eight bytes. Thus, only the 29 higher order bits in the header and footer are needed to record block size, which includes the header and footer and is represented in units of bytes. The usage of the remaining 3 lower order bits is as follows:

   - `bit 0` indicates the use of the current block: 1 for allocated, 0 for free.
   - `bit 1` indicates the use of the previous block: 1 for allocated, 0 for free.
   - `bit 2` indicates the use of the next block: 1 for allocated, 0 for free.

```
/* Given a pointer to a valid block header or footer, returns the size of the
   block (number of bytes). */
int size(void* hp) {
  return (*hp) & (~0x7);    /* Correct? */
}
```

Circle one: Yes or No

Rewritten line of code: _____

Problem 3-a

3. Consider an allocator that uses an implicit free list. Each memory block, either allocated or free, has a size that is a multiple of eight bytes. Thus, only the 29 higher order bits in the header and footer are needed to record block size, which includes the header and footer and is represented in units of bytes. The usage of the remaining 3 lower order bits is as follows:

   - `bit 0` indicates the use of the current block: 1 for allocated, 0 for free.
   - `bit 1` indicates the use of the previous block: 1 for allocated, 0 for free.
   - `bit 2` indicates the use of the next block: 1 for allocated, 0 for free.

```
/* Given a pointer to a valid block header or footer, returns the size of the
   block (number of bytes). */
int size(void* hp) {
  return (*hp) & (~0x7);    /* Correct? */
}
```

Circle one: Yes or No

Rewritten line of code: _____

```
return (*(int*)hp) & (~0x7);
```

Problem 3-b

3. Consider an allocator that uses an implicit free list. Each memory block, either allocated or free, has a size that is a multiple of eight bytes. Thus, only the 29 higher order bits in the header and footer are needed to record block size, which includes the header and footer and is represented in units of bytes. The usage of the remaining 3 lower order bits is as follows:

   - `bit 0` indicates the use of the current block: 1 for allocated, 0 for free.
   - `bit 1` indicates the use of the previous block: 1 for allocated, 0 for free.
   - `bit 2` indicates the use of the next block: 1 for allocated, 0 for free.

```
/* Given a pointer p to an allocated block, i.e,. p is a pointer returned by
   a previous malloc/realloc call; returns a pointer to the block's footer. */
void* footer(void* p) {
  return (char*)p + size((char*)p - 4) - 8;     /* Correct? */
}
```

Circle one: Yes or No

Rewritten line of code: _____

Problem 3-b

3. Consider an allocator that uses an implicit free list. Each memory block, either allocated or free, has a size that is a multiple of eight bytes. Thus, only the 29 higher order bits in the header and footer are needed to record block size, which includes the header and footer and is represented in units of bytes. The usage of the remaining 3 lower order bits is as follows:

   - `bit 0` indicates the use of the current block: 1 for allocated, 0 for free.
   - `bit 1` indicates the use of the previous block: 1 for allocated, 0 for free.
   - `bit 2` indicates the use of the next block: 1 for allocated, 0 for free.

```
/* Given a pointer p to an allocated block, i.e,. p is a pointer returned by
   a previous malloc/realloc call; returns a pointer to the block's footer. */
void* footer(void* p) {
  return (char*)p + size((char*)p - 4) - 8;    /* Correct? */
}
```

Circle one: Yes or No

Rewritten line of code: _____

Problem 3-b

3. Consider an allocator that uses an implicit free list. Each memory block, either allocated or free, has a size that is a multiple of eight bytes. Thus, only the 29 higher order bits in the header and footer are needed to record block size, which includes the header and footer and is represented in units of bytes. The usage of the remaining 3 lower order bits is as follows:

   - `bit 0` indicates the use of the current block: 1 for allocated, 0 for free.
   - `bit 1` indicates the use of the previous block: 1 for allocated, 0 for free.
   - `bit 2` indicates the use of the next block: 1 for allocated, 0 for free.

```
/* Given a pointer p to an allocated block, i.e,. p is a pointer returned by
   a previous malloc/realloc call; returns a pointer to the block's footer. */
void* footer(void* p) {
  return (char*)p + size((char*)p - 4) - 8;     /* Correct? */
}
```

Circle one: Yes or No

Rewritten line of code: _____

addr	desc
0x400b014	(char*)p + size((char*)p - 4)
0x400b010	junk
0x400b00c	pfooter (char*)p + size((char*)p - 4) - 8
0x400b008	junk
0x400b004	junk
0x400b000	p
0x400affc	pheader (char*)p - 4

Problem 3-c

3. Consider an allocator that uses an implicit free list. Each memory block, either allocated or free, has a size that is a multiple of eight bytes. Thus, only the 29 higher order bits in the header and footer are needed to record block size, which includes the header and footer and is represented in units of bytes. The usage of the remaining 3 lower order bits is as follows:

   - `bit 0` indicates the use of the current block: 1 for allocated, 0 for free.
   - `bit 1` indicates the use of the previous block: 1 for allocated, 0 for free.
   - `bit 2` indicates the use of the next block: 1 for allocated, 0 for free.

```
/* Given a pointer to a valid block header or footer, returns the usage of
   the previous block, 1 for allocated, 0 for free. */
int prev_allocated(void* hp) {
  return (*(int*)hp) & 0x4;    /* Correct? */
}
```

Circle one: Yes or No

Rewritten line of code: _____

Problem 3-c

3. Consider an allocator that uses an implicit free list. Each memory block, either allocated or free, has a size that is a multiple of eight bytes. Thus, only the 29 higher order bits in the header and footer are needed to record block size, which includes the header and footer and is represented in units of bytes. The usage of the remaining 3 lower order bits is as follows:

   - `bit 0` indicates the use of the current block: 1 for allocated, 0 for free.
   - `bit 1` indicates the use of the previous block: 1 for allocated, 0 for free.
   - `bit 2` indicates the use of the next block: 1 for allocated, 0 for free.

```
/* Given a pointer to a valid block header or footer, returns the usage of
   the previous block, 1 for allocated, 0 for free. */
int prev_allocated(void* hp) {
  return (*(int*)hp) & 0x4;    /* Correct? */
}
```

Circle one: Yes or No

Rewritten line of code: _____

Problem 3-c

3. Consider an allocator that uses an implicit free list. Each memory block, either allocated or free, has a size that is a multiple of eight bytes. Thus, only the 29 higher order bits in the header and footer are needed to record block size, which includes the header and footer and is represented in units of bytes. The usage of the remaining 3 lower order bits is as follows:

   - `bit 0` indicates the use of the current block: 1 for allocated, 0 for free.
   - `bit 1` indicates the use of the previous block: 1 for allocated, 0 for free.
   - `bit 2` indicates the use of the next block: 1 for allocated, 0 for free.

```
/* Given a pointer to a valid block header or footer, returns the usage of
   the previous block, 1 for allocated, 0 for free. */
int prev_allocated(void* hp) {
  return (*(int*)hp) & 0x4;    /* Correct? */
}
```

Circle one: Yes or No

Rewritten line of code: _____

```
((*(int*)hp) & 0x2) >> 1;
```

Problem 4-a-i

(a) A context switch between two threads in the same process is faster than a context switch between two processes.

i. Explain why.

Problem 4-a-i

(a) A context switch between two threads in the same process is faster than a context switch between two processes.

i. Explain why.

i. Thread contexts are much smaller than process contexts. This is primarily due the to fact that all threads running in a process share the entire virtual memory address space.

Problem 4-a-ii

ii. Assume that you are running a badly programmed version of UNIX in which context switches between threads are actually slower than context switches between processes. Describe a situation in which you, as a programmer, would still prefer to use multiple threads as opposed to multiple processes.

Problem 4-a-ii

ii. Assume that you are running a badly programmed version of UNIX in which context switches between threads are actually slower than context switches between processes. Describe a situation in which you, as a programmer, would still prefer to use multiple threads as opposed to multiple processes.

ii. Sharing data between threads is easier and incurs less overhead than between processes because threads share the same virtual memory address space.

Problem 4-b-i

(b) A global variable is "shared" if it is accessed by more than one thread during the execution of a program. Recall that most of the time, a thread should only access a shared global variable while holding the lock that protects that variable.

i. Describe a situation in which a correct program would not need to hold any lock while accessing a shared global variable.

Problem 4-b-i

(b) A global variable is "shared" if it is accessed by more than one thread during the execution of a program. Recall that most of the time, a thread should only access a shared global variable while holding the lock that protects that variable.

i. Describe a situation in which a correct program would not need to hold any lock while accessing a shared global variable.

i. A lock is not required for correctness if all threads accessing the shared variable do only reads.

Problem 4-b-ii

ii. Explain why it is critical that each of the P and V operations on a semaphore occur indivisibly.

Problem 4-b-ii

ii. Explain why it is critical that each of the P and V operations on a semaphore occur indivisibly.

ii. The semaphore is itself a shared variable. Without atomicity, it could suffer the same interruptions in the load/update/store sequence that we must prevent for other shared variables to ensure correctness.

Problem 4-c

(c) Consider two multi-threaded web servers. Server A creates a new thread for each incoming connection. Server B, on the other hand, pre-creates a fixed pool of threads (eight, for example) at startup time and uses these to handle all connections. First, describe a workload that will cause A to perform better than B. Second, describe a workload that will cause B to perform better than A. In each case, explain why there is a difference in performance. We say that a web server performs better than another if it handles more requests per second.

Problem 4-c

(c) Consider two multi-threaded web servers. Server A creates a new thread for each incoming connection. Server B, on the other hand, pre-creates a fixed pool of threads (eight, for example) at startup time and uses these to handle all connections. First, describe a workload that will cause A to perform better than B. Second, describe a workload that will cause B to perform better than A. In each case, explain why there is a difference in performance. We say that a web server performs better than another if it handles more requests per second.

(c) Server B will perform better than server A is there is a steady stream of eight or fewer concurrent requests, because B does not incur the overhead of thread creation and removal for each request (as A does).

Server A will perform better than server B if there are regularly more than eight concurrent requests, because A can dynamically create a new thread to handle each new request.

Problem 5-a

Program 1 Initially, a is 1, b is 1, and c is 1.

Thread 1	*Thread 2*
P(a)	P(c)
P(b)	P(a)
P(c)	V(c)
V(a)	P(b)
V(b)	V(a)
V(c)	V(b)

Circle *one* of the following outcomes. When **Program 1** executes, deadlock:

always occurs might occur never occurs

Thread 1	Thread 2	a	b	c
P(a)		0	1	1
P(b)		0	0	1
P(c)		0	0	0
V(a)		1	0	0
V(b)		1	1	0
V(c)		1	1	1
	P(c)	1	1	0
	P(a)	0	1	0
	V(c)	0	1	1
	P(b)	0	0	1
	V(a)	1	0	1
	V(b)	1	1	1

Thread 1	Thread2	a	b	c
P(a)		0	1	1
	P(c)	0	1	0
P(b)		0	0	0
P(c)	P(a)	-	-	-

Problem 5-b

Program 2 Initially, `a` is 1, `b` is 1, and `c` is 1.

Thread 1	*Thread 2*
P(a)	P(a)
P(b)	V(a)
P(c)	P(c)
V(a)	P(b)
V(b)	V(c)
V(c)	V(b)

Circle *one* of the following outcomes. When **Program 2** executes, deadlock:

always occurs might occur never occurs

Thread 1	Thread 2	a	b	c
P(a)		0	1	1
P(b)		0	0	1
P(c)		0	0	0
V(a)		1	0	0
V(b)		1	1	0
V(c)		1	1	1
	P(a)	0	1	1
	V(a)	1	1	1
	P(c)	1	1	0
	P(b)	1	0	0
	V(c)	1	0	1
	V(b)	1	1	1

Thread 1	Thread2	a	b	c
	P(a)	0	1	1
	V(a)	1	1	1
	P(c)	1	1	0
P(a)		0	1	0
P(b)		0	0	0
P(c)	P(b)	-	-	-

Problem 5-c

Program 3 Initially, `a` is 1, `b` is 1, and `c` is 1.

Thread 1	*Thread 2*
P(a)	P(b)
P(c)	P(a)
V(c)	P(c)
V(a)	V(a)
P(b)	V(c)
V(b)	V(b)

Circle *one* of the following outcomes. When **Program 3** executes, deadlock:

always occurs might occur never occurs

Thread 1	Thread 2	a	b	c
P(a)		0	1	1
P(c)		0	1	0
V(c)		0	1	1
V(a)		1	1	1
P(b)		1	0	0
V(b)		1	1	1
	P(b)	1	0	1
	P(a)	0	0	1
	P(c)	0	0	0
	V(a)	1	0	0
	V(c)	1	0	1
	V(b)	1	1	1

Problem 6-a

```
foo1:
  pushl %ebp
  movl %esp,%ebp
  movl 8(%ebp),%eax
  sall $4,%eax
  subl 8(%ebp),%eax
  movl %ebp,%esp
  popl %ebp
  ret
```

```
int choice1(int x) {
  return (x < 0);
}

int choice2(int x) {
  return (x << 31) & 1;
}

int choice3(int x) {
  return 15 * x;
}

int choice4(int x) {
  return (x + 15) / 4;
}

int choice5(int x) {
  return x / 16;
}

int choice6(int x) {
  return (x >> 31);
}
```

Assembler routine foo1 corresponds to C function _____

Problem 6-a

```
foo1:
  pushl %ebp
  movl %esp,%ebp
  movl 8(%ebp),%eax
  sall $4,%eax
  subl 8(%ebp),%eax
  movl %ebp,%esp
  popl %ebp
  ret
```

```
int choice1(int x) {
  return (x < 0);
}

int choice2(int x) {
  return (x << 31) & 1;
}

int choice3(int x) {
  return 15 * x;
}

int choice4(int x) {
  return (x + 15) / 4;
}

int choice5(int x) {
  return x / 16;
}

int choice6(int x) {
  return (x >> 31);
}
```

Assembler routine foo1 corresponds to C function choice3

Problem 6-b

```
foo2:
  pushl %ebp
  movl %esp,%ebp
  movl 8(%ebp),%eax
  testl %eax,%eax
  jge .L4
  addl $15,%eax
.L4:
  sarl $4,%eax
  movl %ebp,%esp
  popl %ebp
  ret
```

```
int choice1(int x) {
  return (x < 0);
}

int choice2(int x) {
  return (x << 31) & 1;
}

int choice3(int x) {
  return 15 * x;
}

int choice4(int x) {
  return (x + 15) / 4;
}

int choice5(int x) {
  return x / 16;
}

int choice6(int x) {
  return (x >> 31);
}
```

Assembler routine foo2 corresponds to C function _____

Problem 6-b

```
foo2:
  pushl %ebp
  movl %esp,%ebp
  movl 8(%ebp),%eax
  testl %eax,%eax
  jge .L4
  addl $15,%eax
.L4:
  sarl $4,%eax
  movl %ebp,%esp
  popl %ebp
  ret
```

```
int choice1(int x) {
  return (x < 0);
}

int choice2(int x) {
  return (x << 31) & 1;
}

int choice3(int x) {
  return 15 * x;
}

int choice4(int x) {
  return (x + 15) / 4;
}

int choice5(int x) {
  return x / 16;
}

int choice6(int x) {
  return (x >> 31);
}
```

Assembler routine foo2 corresponds to C function choice5

Problem 6-c

```
foo3:
  pushl %ebp
  movl %esp,%ebp
  movl 8(%ebp),%ecx
  xorl %eax,%eax
  testl %ecx,%ecx
  jge .L6
  incl %eax
.L6
  movl %ebp,%esp
  popl %ebp
  ret
```

```
int choice1(int x) {
  return (x < 0);
}

int choice2(int x) {
  return (x << 31) & 1;
}

int choice3(int x) {
  return 15 * x;
}

int choice4(int x) {
  return (x + 15) / 4;
}

int choice5(int x) {
  return x / 16;
}

int choice6(int x) {
  return (x >> 31);
}
```

Assembler routine foo3 corresponds to C function _____

Problem 6-c

```
foo3:
  pushl %ebp
  movl %esp,%ebp
  movl 8(%ebp),%ecx
  xorl %eax,%eax
  testl %ecx,%ecx
  jge .L6
  incl %eax
.L6
  movl %ebp,%esp
  popl %ebp
  ret
```

```
int choice1(int x) {
  return (x < 0);
}

int choice2(int x) {
  return (x << 31) & 1;
}

int choice3(int x) {
  return 15 * x;
}

int choice4(int x) {
  return (x + 15) / 4;
}

int choice5(int x) {
  return x / 16;
}

int choice6(int x) {
  return (x >> 31);
}
```

Assembler routine foo3 corresponds to C function choice1

Problem 7

7. Consider the following code fragment containing the incomplete definition of a data type `struct matrix_entry` with four fields.

`struct matrix_entry` with four fields.

```
struct matrix_entry{

  --------- a;

  --------- b;

  short  c;

  --------- d;

};
```

```
struct matrix_entry matrix[2][5];

short return_entry(int i, int j) {
  return matrix[i][j].c;
}
```

```
return_entry:
  pushl %ebp
  movl %esp,%ebp
  movl 8(%ebp),%eax
  leal (%eax,%eax,4),%eax
  addl 12(%ebp),%eax
  sall $4,%eax
  movl matrix+6(%eax),%eax
  movl %ebp,%esp
  popl %ebp
  ret
```

Type	Size (bytes)	Alignment (bytes)
char	1	1
short	2	2
int	4	4
double	8	4

Problem 7

7. Consider the following code fragment containing the incomplete definition of a data type `struct matrix_entry` with four fields.

`struct matrix_entry` with four fields.

```
struct matrix_entry{

  --------- a;

short or char
  --------- b;

  short  c;

  --------- d;

};
```

```
struct matrix_entry matrix[2][5];

short return_entry(int i, int j) {
  return matrix[i][j].c;
}
```

```
return_entry:
  pushl %ebp
  movl %esp,%ebp
  movl 8(%ebp),%eax
  leal (%eax,%eax,4),%eax
  addl 12(%ebp),%eax
  sall $4,%eax
  movl matrix+6(%eax),%eax
  movl %ebp,%esp
  popl %ebp
  ret
```

Type	Size (bytes)	Alignment (bytes)
char	1	1
short	2	2
int	4	4
double	8	4

Problem 7

7. Consider the following code fragment containing the incomplete definition of a data type `struct matrix_entry` with four fields.

`struct matrix_entry` with four fields.

```
struct matrix_entry{

  int     a;

short or char b;

  short   c;

  ---------- d;

};
```

```
struct matrix_entry matrix[2][5];

short return_entry(int i, int j) {
  return matrix[i][j].c;
}
```

```
return_entry:
  pushl %ebp
  movl %esp,%ebp
  movl 8(%ebp),%eax
  leal (%eax,%eax,4),%eax
  addl 12(%ebp),%eax
  sall $4,%eax
  movl matrix+6(%eax),%eax
  movl %ebp,%esp
  popl %ebp
  ret
```

Type	Size (bytes)	Alignment (bytes)
char	1	1
short	2	2
int	4	4
double	8	4

Problem 7

7. Consider the following code fragment containing the incomplete definition of a data type `struct matrix_entry` with four fields.

`struct matrix_entry` with four fields.

```
struct matrix_entry{

   int    a;

   short or char  b;

   short  c;

   double  d;

};
```

```
struct matrix_entry matrix[2][5];

short return_entry(int i, int j) {
  return matrix[i][j].c;
}
```

```
return_entry:
  pushl %ebp
  movl %esp,%ebp
  movl 8(%ebp),%eax
  leal (%eax,%eax,4),%eax
  addl 12(%ebp),%eax
  sall $4,%eax
  movl matrix+6(%eax),%eax
  movl %ebp,%esp
  popl %ebp
  ret
```

Type	Size (bytes)	Alignment (bytes)
char	1	1
short	2	2
int	4	4
double	8	4

Problem 8

```
/* copy string x to buf */
void foo(char *x) {
  int buf[1];
  strcpy((char *)buf, x);
}

void callfoo() {
  foo("abcdefghi");
}
```

Functions `foo` and `callfoo` have the following disassembled form on an IA32 machine.

```
080484f4 <foo>:
080484f4: 55              pushl  %ebp
080484f5: 89 e5           movl   %esp,%ebp
080484f7: 83 ec 18        subl   $0x18,%esp
080484fa: 8b 45 08        movl   0x8(%ebp),%eax
080484fd: 83 c4 f8        addl   $0xfffffff8,%esp
08048500: 50              pushl  %eax    # push x
08048501: 8d 45 fc        leal   0xfffffffc(%ebp),%eax
08048504: 50              pushl  %eax    # push buf
08048505: e8 ba fe ff ff  call   80483c4 <strcpy>
0804850a: 89 ec           movl   %ebp,%esp
0804850c: 5d              popl   %ebp
0804850d: c3              ret

08048510 <callfoo>:
08048510: 55              pushl  %ebp
08048511: 89 e5           movl   %esp,%ebp
08048513: 83 ec 08        subl   $0x8,%esp
08048516: 83 c4 f4        addl   $0xfffffff4,%esp
08048519: 68 9c 85 04 08  pushl  $0x804859c    # push string address
0804851e: e8 d1 ff ff ff  call   80484f4 <foo>
08048523: 89 ec           movl   %ebp,%esp
08048525: 5d              popl   %ebp
08048526: c3              ret
```

Problem 8

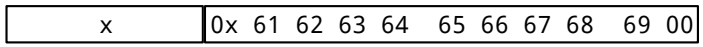
```
/* copy string x to buf */
void foo(char *x) {
  int buf[1];
  strcpy((char *)buf, x);
}

void callfoo() {
  foo("abcdefghi");
}
```

Functions `foo` and `callfoo` have the following disassembled form on an IA32 machine.

```
080484f4 <foo>:
080484f4: 55              pushl  %ebp
080484f5: 89 e5           movl   %esp,%ebp
080484f7: 83 ec 18        subl   $0x18,%esp
080484fa: 8b 45 08        movl   0x8(%ebp),%eax
080484fd: 83 c4 f8        addl   $0xfffffff8,%esp
08048500: 50              pushl  %eax    # push x
08048501: 8d 45 fc        leal   0xfffffffc(%ebp),%eax
08048504: 50              pushl  %eax    # push buf
08048505: e8 ba fe ff ff  call   80483c4 <strcpy>
0804850a: 89 ec           movl   %ebp,%esp
0804850c: 5d              popl   %ebp
0804850d: c3              ret

08048510 <callfoo>:
08048510: 55              pushl  %ebp
08048511: 89 e5           movl   %esp,%ebp
08048513: 83 ec 08        subl   $0x8,%esp
08048516: 83 c4 f4        addl   $0xfffffff4,%esp
08048519: 68 9c 85 04 08  pushl  $0x804859c    # push string address
0804851e: e8 d1 ff ff ff  call   80484f4 <foo>
08048523: 89 ec           movl   %ebp,%esp
08048525: 5d              popl   %ebp
08048526: c3              ret
```



Problem 8

```
/* copy string x to buf */
void foo(char *x) {
  int buf[1];
  strcpy((char *)buf, x);
}

void callfoo() {
  foo("abcdefghi");
}
```

Functions `foo` and `callfoo` have the following disassembled form on an IA32 machine.

```
080484f4 <foo>:
080484f4: 55              pushl  %ebp
080484f5: 89 e5           movl   %esp,%ebp
080484f7: 83 ec 18        subl   $0x18,%esp
080484fa: 8b 45 08        movl   0x8(%ebp),%eax
080484fd: 83 c4 f8        addl   $0xfffffff8,%esp
08048500: 50              pushl  %eax    # push x
08048501: 8d 45 fc        leal   0xfffffffc(%ebp),%eax
08048504: 50              pushl  %eax    # push buf
08048505: e8 ba fe ff ff  call   80483c4 <strcpy>
0804850a: 89 ec           movl   %ebp,%esp
0804850c: 5d              popl   %ebp
0804850d: c3              ret

08048510 <callfoo>:
08048510: 55              pushl  %ebp
08048511: 89 e5           movl   %esp,%ebp
08048513: 83 ec 08        subl   $0x8,%esp
08048516: 83 c4 f4        addl   $0xfffffff4,%esp
08048519: 68 9c 85 04 08  pushl  $0x804859c    # push string address
0804851e: e8 d1 ff ff ff  call   80484f4 <foo>
08048523: 89 ec           movl   %ebp,%esp
08048525: 5d              popl   %ebp
08048526: c3              ret
```

x	0x 61 62 63 64 65 66 67 68 69 00

buf[2]	0x 08 04 00 69
buf[1]	0x 68 67 66 65
buf[0]	0x 64 63 62 61

Problem 8

```
/* copy string x to buf */
void foo(char *x) {
  int buf[1];
  strcpy((char *)buf, x);
}

void callfoo() {
  foo("abcdefghi");
}
```

Functions `foo` and `callfoo` have the following disassembled form on an IA32 machine.

```
080484f4 <foo>:
080484f4: 55              pushl  %ebp
080484f5: 89 e5           movl   %esp,%ebp
080484f7: 83 ec 18        subl   $0x18,%esp
080484fa: 8b 45 08        movl   0x8(%ebp),%eax
080484fd: 83 c4 f8        addl   $0xfffffff8,%esp
08048500: 50              pushl  %eax    # push x
08048501: 8d 45 fc        leal   0xfffffffc(%ebp),%eax
08048504: 50              pushl  %eax    # push buf
08048505: e8 ba fe ff ff  call   80483c4 <strcpy>
0804850a: 89 ec           movl   %ebp,%esp
0804850c: 5d              popl   %ebp
0804850d: c3              ret

08048510 <callfoo>:
08048510: 55              pushl  %ebp
08048511: 89 e5           movl   %esp,%ebp
08048513: 83 ec 08        subl   $0x8,%esp
08048516: 83 c4 f4        addl   $0xfffffff4,%esp
08048519: 68 9c 85 04 08  pushl  $0x804859c    # push string address
0804851e: e8 d1 ff ff ff  call   80484f4 <foo>
08048523: 89 ec           movl   %ebp,%esp
08048525: 5d              popl   %ebp
08048526: c3              ret
```

x	0x 61 62 63 64 65 66 67 68 69 00

buf[2]	0x 08 04 00 69
buf[1]	0x 68 67 66 65
buf[0]	0x 64 63 62 61

callfoo return address	buf[2]	0x 08 04 00 69
save ebp	buf[1]	0x 68 67 66 65
leal 0xfffffffc(%ebp)	buf[0]	0x 64 63 62 61

Problem 8

```
/* copy string x to buf */
void foo(char *x) {
  int buf[1];
  strcpy((char *)buf, x);
}

void callfoo() {
  foo("abcdefghi");
}
```

Functions `foo` and `callfoo` have the following disassembled form on an IA32 machine.

```
080484f4 <foo>:
080484f4: 55              pushl  %ebp
080484f5: 89 e5           movl   %esp,%ebp
080484f7: 83 ec 18        subl   $0x18,%esp
080484fa: 8b 45 08        movl   0x8(%ebp),%eax
080484fd: 83 c4 f8        addl   $0xfffffff8,%esp
08048500: 50              pushl  %eax    # push x
08048501: 8d 45 fc        leal   0xfffffffc(%ebp),%eax
08048504: 50              pushl  %eax    # push buf
08048505: e8 ba fe ff ff  call   80483c4 <strcpy>
0804850a: 89 ec           movl   %ebp,%esp
0804850c: 5d              popl   %ebp
0804850d: c3              ret

08048510 <callfoo>:
08048510: 55              pushl  %ebp
08048511: 89 e5           movl   %esp,%ebp
08048513: 83 ec 08        subl   $0x8,%esp
08048516: 83 c4 f4        addl   $0xfffffff4,%esp
08048519: 68 9c 85 04 08  pushl  $0x804859c    # push string address
0804851e: e8 d1 ff ff ff  call   80484f4 <foo>
08048523: 89 ec           movl   %ebp,%esp
08048525: 5d              popl   %ebp
08048526: c3              ret
```

x	0x 61 62 63 64 65 66 67 68 69 00

buf[2]	0x 08 04 00 69
buf[1]	0x 68 67 66 65
buf[0]	0x 64 63 62 61

callfoo return address	buf[2]	0x 08 04 00 69
save ebp	buf[1]	0x 68 67 66 65
leal 0xfffffffc(%ebp)	buf[0]	0x 64 63 62 61

reg	old	answer
%ebp	????????	0x 68 67 66 65
%eip	0x 80 04 85 23	0x 08 04 00 69

Problem 9-a

9. Consider the following C program. Assume that all functions return normally and that the proper header files have been included.

```
int main() {
   int status;
   printf("start\n");
   printf("%d\n", !fork());
   if(wait(&status) != -1)
      printf("%d\n", WEXITSTATUS(status));
   printf("end\n");
   exit(2);
}
```

Recall the following:

- Function `wait` returns `-1` when there is an error, e.g., when there is no child.
- Macro `WEXITSTATUS` extracts the exit status of the terminating process.

(a) Draw a diagram that illustrates the processes at run-time.

Problem 9-a

9. Consider the following C program. Assume that all functions return normally and that the proper header files have been included.

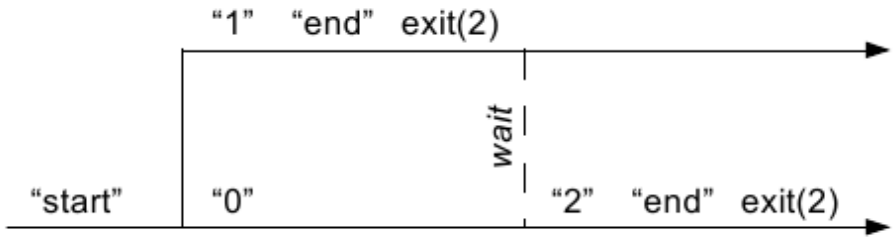
```
int main() {
   int status;
   printf("start\n");
   printf("%d\n", !fork());
   if(wait(&status) != -1)
      printf("%d\n", WEXITSTATUS(status));
   printf("end\n");
   exit(2);
}
```

Recall the following:

- Function `wait` returns `-1` when there is an error, e.g., when there is no child.
- Macro `WEXITSTATUS` extracts the exit status of the terminating process.

(a) Draw a diagram that illustrates the processes at run-time.

(a)



Problem 9-b

```
int main() {
   int status;
   printf("start\n");
   printf("%d\n", !fork());
   if(wait(&status) != -1)
      printf("%d\n", WEXITSTATUS(status));
   printf("end\n");
   exit(2);
}
```

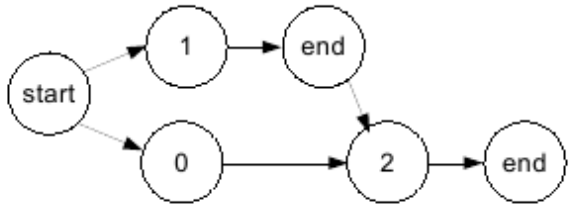
(b) Give *three* possible outputs of this program.

Problem 9-b

```
int main() {
   int status;
   printf("start\n");
   printf("%d\n", !fork());
   if(wait(&status) != -1)
      printf("%d\n", WEXITSTATUS(status));
   printf("end\n");
   exit(2);
}
```

(b) Give *three* possible outputs of this program.

(b) All three valid topological sorts of the following graph.



start	start	start
1	0	1
end	1	0
0	end	end
2	2	2
end	end	end