Question 1. Multiple Choice (26 points)

For each question, mark any that apply. Only answers that mark all correct choices will be considered correct.

1. What are some benefits of virtual memory?
   - (a) It allows the address space to be larger than the physical address space.
   - (b) It isolates address spaces of different processes.
   - (c) It enables faster memory accesses.
   - (d) It increases the amount of instruction level parallelism.
2. When you print the address of a variable from C, what kind of address is that?
   - (a) Physical Address
   - (b) Virtual Address
   - (c) Depends on the context
   - (d) None of the above
3. What's the purpose of the linker?
   - (a) Optimize the code's performance.
   - (b) Support the abstraction of virtual memory.
   - (c) Enable separate compilation.
   - (d) Simplify the contract between hardware and software.
4. Why could loop unrolling sometimes be harmful to performance?
   - (a) It introduces extra dynamic instructions, potentially adding to the critical path length.
   - (b) It introduces extra static instructions, putting more strain on the instruction cache.
   - (c) It results in unnecessary strength reduction, causing more expensive operations to be executed.
   - (d) Trick question, it is never harmful for performance.
5. What is the standard order of steps in a compilation flow?
   - (a) Compile, Preprocess, Link
   - (b) Compile, Link, Preprocess
   - (c) Preprocess, Link, Compile
   - (d) Preprocess, Compile, Link
6. What of the following are reason(s) why multi-threaded code might go slower than expected?
   - (a) More contention for data in the cache.
   - (b) Overhead of synchronizing shared variables.
   - (c) Overhead of starting/stopping threads.
   - (d) Space overhead from .bss/.data sections.
7. Each thread in a multithreaded program has its own:
   - (a) Heap
   - (b) Stack
   - (c) Global values
   - (d) Text Section
   - (e) Register Values

8. What is the purpose of a multi-level page table, over a single level?
   (a) Faster access to physical memory.
   (b) It occupies less space.
   (c) Enables memory protection.
   (d) Helps to make context-switching faster.
9. Which of the following could be contained in an ELF file?
   (a) Machine Code
   (b) Stack
   (c) Global Variables
   (d) Symbol Table
   (e) Heap
10. Given the simple implicit-free-list approach that we discussed in class for implementing malloc, which of the following scenarios would cause the most external *fragmentation*?
    (a) Repeatedly calling malloc() on many different sizes.
    (b) Repeatedly calling malloc() on many different sizes, while sometimes freeing some memory.
    (c) Repeatedly calling malloc() with the same size.
    (d) Repeatedly calling malloc() with the same size, while somtimes freeing some memory.
11. For a floating point number, what would be an effect of allocating more bits to the exponent part by taking them from the fraction part?
    (a) You could represent fewer numbers, but they could be much larger.
    (b) You could represent the same numbers, but with more decimal places.
    (c) You could represent both larger and smaller numbers, but with less precision.
    (d) Some previously representable numbers would now round to infinity
12. Assuming no errors, which one of the following functions returns exactly once?
    (a) fork()
    (b) execve()
    (c) exit()
    (d) wait()
13. What distinguishes a trap from a fault?
    (a) A trap is an exception, while a fault is not
    (b) A trap is intentional, while a fault is not
    (c) A trap is recoverable, while a fault is not
    (d) A trap is synchronous, while a fault is not

Question 2. And just who is responsible for this? (9pts)

For each blank, write in which entity is primarily responsible for each of the following items. The options are: Compiler, Linker, Loader, Operating System Kernel, or Hardware.

1. Code Relocation ____________
2. Symbol Resolution ____________
3. Code Optimization ____________
4. Initiating a context switch ____________
5. Enabling Virtual Memory abstraction ____________, ____________
6. Simultaneous Multithreading ____________
7. Allocating memory for uninitialized global variables. ____________
8. Allocating and storing data in cache memories (eg. L1, L2, etc). ____________

Question 3. Easy Linking (6 pts)

Suppose we compile main.c and foo.c separately, then link them together.

main.c

```
#include <stdio.h>
int a = 1;
static int b = 2;
int c = 3;
int main() {
  int c = 4;
  foo();
  printf("a=%d,b=%d,c=%d\n", a, b, c);
  return 0;
}
```

foo.c

```
int a, b, c;
void foo() {
  a = 5;
  b = 6;
  c = 7;
}
```

1. What is printed? a=____________,b=____________,c=____________

Question 4. Hard Linking (16 pts)

Suppose we compile main.c and func.c separately, then link them together.

main.c

```
#include <stdio.h>
#include <stdlib.h>

long long int x=0;
int y;
int* pointer;

int func(int* p);

int main(int argc, char**argv) {
    int my_array[5];

    pointer=(int*)malloc(100);
    y=1;

    func(&pointer[0]);
    func(&y);
    printf("%d,%d\n",y,pointer[0]);
}
```

func.c

```
#include <stdio.h>

double x;
int y=10;

int func(int* p)  {
  static int s=0;
  s+=1;
  y+=1;

  *p=y+s;

  x+=0.25;
  printf("x=%f,s=%d\n",x,s);
}
```

1. What linker errors or warnings (if any) are there for this code?

2. If there are linker errors, how could you fix them? If there are no linker errors, what is printed when we run this program?

3. For each of the following, describe where that variable is, both in terms of where it is in the ELF file after linking (.text,.data,.bss,.debug,.data, etc), and where it is dynamically in the virtual memory space as it is executing(stack,heap,text,etc). If the associated memory does not exist in one of those places (eg. if the ELF does not explicitly contain space allocated for the data associated with that variable) mark with a dash "–". Also, assume the compiler does not optimize-out unused variables during compilation.

	ELF:	In VM:
x		
func		
my_array		
my_array[0]		
pointer		
pointer[0]		

Question 5. This problem makes my neurons hurt... (15 pts)

Your task is to analyze the performance of a neural network computation, which is a slightly modified matrix-vector multiplication. Basically, this code multiplies a vectcor (called `neuron_in`) by a matrix (called `synapse`), and the output will be a vector (called `neuron_out`). Also, the `sigmoid` operation is applied to each output element (the `sigmoid` function is not shown below).

Nevermind what the program is doing, we can optimize it without understanding it at all! Below is the original code, and 4 different "optimized" versions of `main`.

ORIGINAL CODE:

```
uint16_t synapse [Nout][Nin], neuron_in[Nin], neuron_out[Nout];

void apply_sigmoid(uint16_t* elem) {
  *elem = sigmoid(*elem);
}
void multiply_elem(uint16_t* output, uint16_t synapse, uint16_t neuron_in) {
   *output += synapse * neuron_in;
}

void main() {
  for(int i = 0; i < Nin; i++) {
    for(int j = 0; j < Nout; j++) {
      multiply_elem(&neuron_out[j],synapse[j][i],neuron_in[i]);
    }
  }
  for(j = 0; j < Nout; j++) {
    apply_sigmoid(&neuron_out[j])
  }
}
```

Version A:

```
void main() {
  for(int i = 0; i < Nin; i++) {
    for(int j = 0; j < Nout; j++) {
      neuron_out[j] += synapse[j][i] *
                       neuron_in[i];
    }
  }
  for(int j = 0; j < Nout; j++) {
    apply_sigmoid(&neuron_out[j])
  }
}
```

Version B:

```
void main() {
  for(int i = 0; i < Nin; i++) {
    for(int j = 0; j < Nout; j++) {
      multiply_elem(&neuron_out[j],
         synapse[j][i],neuron_in[i]);
    }
  }
  for(int j = 0; j < Nout; j++) {
    neuron_out[j] =
      sigmoid(neuron_out[j]);
  }
}
```

Version C:

```
void main() {
  for(int j = 0; j < Nout; j++) {
    for(int i = 0; i < Nin; i++) {
      neuron_out[j] += synapse[j][i] *
                       neuron_in[i];
    }
  }
  for(int j = 0; j < Nout; j++) {
    neuron_out[j] =
       sigmoid(neuron_out[j]);
  }
}
```

Version D:

```
void main() {
  for(i = 0; i < Nin; i++) {
    for(j = 0; j < Nout; j++) {
      neuron_out[j] += synapse[j][i] *
                       neuron_in[i];
    }
  }
  for(j = 0; j < Nout; j++) {
    neuron_out[j] =
         sigmoid(neuron_out[j]);
  }
}
```

1. What optimization was applied to Versions A & B? (2pts)

2. Which version, A or B, is faster and why? (use no more than 10 words, 2pts)

3. Which version, C or D, is faster and why? (use no more than 15 words, 2pts)

4. What is one additional optimization that can be performed, besides multi-threading and the optimizations applied above? (use no more than 10 words, 3pts)

5. We disassembled Version C's innermost loop below (the one that updates i).

```
0x00000000004003ef <+15>:    mov    0x601140(%rdi,%rdx,1),%si
0x00000000004003f7 <+23>:    add    $0x2,%rdx
0x00000000004003fb <+27>:    imul   0x60105e(%rdx),%si
0x0000000000400403 <+35>:    add    %esi,%ecx
0x0000000000400405 <+37>:    cmp    $0xc8,%rdx
0x000000000040040c <+44>:    jne    0x4003ef <main+15>
```

   Assume that all instructions take 1 cycle, that all branches are perfectly predicted, and that we have an out-of-order core with infinite functional units. If we run this loop 100 times, about how many cycles will it take to execute? (4pts)

6. If we changed the core so that the latency of a multiply is 3 cycles, now how long would it take? (2pts)

Question 6. Caching Out (18 pts)

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 4 byte line size and 16 total lines.

In the following tables, **all numbers are given in hexadecimal**. The contents of the TLB, the page table for the first 32 pages, and the cache are as follows:

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

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	1F	1	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	1F	1	11	C2	11	33	1E	1	00	C0	0F	00

First, determine the bit-level mappings. (8pts)

The box below shows the format of a virtual address. Indicate (by labeling the diagram) the fields (if they exist) that would be used to determine the following: (If a field doesn't exist, don't draw it on the diagram.)

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

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

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

(extra scratch space)

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

12	11	10	9	8	7	6	5	4	3	2	1	0

Fill in the following: (10 pts)

Virtual Addr.	Physical Address	TLB Miss?	Page Fault?	Cache Miss?	Byte Read
0x1DFD	0x				0x
0x1DFE	0x				0x
0x1DFF	0x				0x
0x1E00	0x				0x

1. Simulate what happens when a program executes 4 load statements (one after the other), with the corresponding virtual addresses appearing below. For each one, indicate the physical address, whether there was a TLB miss, page fault and cache miss, as well as the byte read. If certain information cannot be determined from the above tables, *leave it blank!*.

Question 7. You've got to be forking kidding me! (15 pts)

Consider the following multi-threaded and multi-process versions of a code. (of course, only one "main" function would be included at a time)

```
pthread_t threads[3];
sem_t     s1,s2,s3;

void* func1(void* x) { sem_wait(&s1); printf("Do\n");  sem_post(&s2); }
void* func2(void* x) { sem_wait(&s2); printf("Se\n");  sem_post(&s3); }
void* func3(void* x) { sem_wait(&s3); printf("Yek\n"); sem_post(&s1); }


int main(int argc, char** argv) {  //Thread Version!
  sem_init(&s1,0,0);
  sem_init(&s2,0,0);
  sem_init(&s3,0,1);

  pthread_create(&threads[0], NULL, func1, NULL);
  pthread_create(&threads[1], NULL, func2, NULL);
  pthread_create(&threads[2], NULL, func3, NULL);

  for(int i = 0; i < 3; ++i) {
    void* status;
    pthread_join(threads[i], &status);
  }
}


int main(int argc, char** argv) { //Fork Version!
  sem_init(&s1,0,0);
  sem_init(&s2,0,0);
  sem_init(&s3,0,1);

  int id1 = fork();
  int id2 = fork();

  if(id1 == 0 && id2 == 0) { func1( (void*) 0); }
  if(id1 != 0 && id2 != 0) { func2( (void*) 0); }
  if(id1 != 0 && id2 == 0) { func3( (void*) 0); }
}
```

Answer the following questions BOTH for the multi-thread and multi-process versions of main: (2 pts each)

1. What could possibly be printed?

   Multithread: Multiprocess:

2. Is there a possibility for race?

   Multithread: Multiprocess:

3. Is there a possibility for deadlock?

   Multithread: Multiprocess:

4. Write the letter T next to any characters relevant to the multi-threaded version (if any), write the letter M next to any characters relevant to the multi-process version (if any). (3pts)

NINJA
PIRATE
ZOMBIE
ROBOT

Question 8. Time for Revenge! (16pts)

It's time to get revenge on the big banks. Your friend Sombra works at InsecureBankCo, and manages to steal some code. She also gets the dissassembly from gdb, which is printed below.

Each user's data is in a bank_account_t struct, and there is a table for all the users called "users". You know your account data is inside.

One of the features of the bank is that they let users set a notification time for any account alerts – you notice that this occurs in "set_notification_time". This function takes a user ID, an id for the day-of-the-week (0 represents Monday, 4 represents Friday), and a integer representing the notification time (9 = 9:00am). This function looks *very* interesting for an exploit.

```
typedef struct acc {
  char username[16];                  //username string
  long int dollars;                   //each user's money
  long int daily_notification_time[5]; //time of day for notificaiton
} bank_account_t;

bank_account_t users[num_accounts];

int set_notification_time(int user_id, int day_of_week, long int time) {
  bank_account_t b = users[user_id];
  b.daily_notification_time[day_of_week] = time;
  users[user_id] = b;
}

void delete_all_accounts() {
  memset(users,0,sizeof(users)); //sets the users array to 0
}

(gdb) p/x &users
$2 = 0x601060

(gdb) disassemble set_notification_time
Dump of assembler code for function set_notification_time:
   0x0000000000400546 <+0>:     movslq %edi,%rdi
   0x0000000000400549 <+3>:     shl    $0x6,%rdi
   0x000000000040054d <+7>:     mov    0x601060(%rdi),%rcx
   0x0000000000400554 <+14>:    mov    %rcx,-0x40(%rsp)
   0x0000000000400559 <+19>:    mov    0x601068(%rdi),%rcx
   0x0000000000400560 <+26>:    mov    %rcx,-0x38(%rsp)
   0x0000000000400565 <+31>:    mov    0x601070(%rdi),%rcx
   0x000000000040056c <+38>:    mov    %rcx,-0x30(%rsp)
   0x0000000000400571 <+43>:    mov    0x601078(%rdi),%rcx
   0x0000000000400578 <+50>:    mov    %rcx,-0x28(%rsp)
   0x000000000040057d <+55>:    mov    0x601080(%rdi),%rcx
   0x0000000000400584 <+62>:    mov    %rcx,-0x20(%rsp)
   0x0000000000400589 <+67>:    mov    0x601088(%rdi),%rcx
   0x0000000000400590 <+74>:    mov    %rcx,-0x18(%rsp)
   0x0000000000400595 <+79>:    mov    0x601090(%rdi),%rcx
   0x000000000040059c <+86>:    mov    %rcx,-0x10(%rsp)
   0x00000000004005a1 <+91>:    mov    0x601098(%rdi),%rcx
   0x00000000004005a8 <+98>:    mov    %rcx,-0x8(%rsp)
   0x00000000004005ad <+103>:   movslq %esi,%rsi
   0x00000000004005b0 <+106>:   mov    %rdx,-0x28(%rsp,%rsi,8)
   0x00000000004005b5 <+111>:   mov    -0x40(%rsp),%rdx
   0x00000000004005ba <+116>:   mov    %rdx,0x601060(%rdi)
   0x00000000004005c1 <+123>:   mov    -0x38(%rsp),%rdx
   0x00000000004005c6 <+128>:   mov    %rdx,0x601068(%rdi)
   0x00000000004005cd <+135>:   mov    -0x30(%rsp),%rdx
   0x00000000004005d2 <+140>:   mov    %rdx,0x601070(%rdi)
   0x00000000004005d9 <+147>:   mov    -0x28(%rsp),%rdx
   0x00000000004005de <+152>:   mov    %rdx,0x601078(%rdi)
   0x00000000004005e5 <+159>:   mov    -0x20(%rsp),%rdx
   0x00000000004005ea <+164>:   mov    %rdx,0x601080(%rdi)
   0x00000000004005f1 <+171>:   mov    -0x18(%rsp),%rdx
   0x00000000004005f6 <+176>:   mov    %rdx,0x601088(%rdi)
   0x00000000004005fd <+183>:   mov    -0x10(%rsp),%rdx
   0x0000000000400602 <+188>:   mov    %rdx,0x601090(%rdi)
   0x0000000000400609 <+195>:   mov    -0x8(%rsp),%rdx
   0x000000000040060e <+200>:   mov    %rdx,0x601098(%rdi)
   0x0000000000400615 <+207>:   retq

(gdb) disassemble delete_all_accounts
Dump of assembler code for function delete_all_accounts:
   0x0000000000400616 <+0>:     mov    $0x601060,%edx
   0x000000000040061b <+5>:     mov    $0x320,%ecx
   0x0000000000400620 <+10>:    mov    $0x0,%eax
   0x0000000000400625 <+15>:    mov    %rdx,%rdi
   0x0000000000400628 <+18>:    rep stos %rax,%es:(%rdi)
   0x000000000040062b <+21>:    retq
```

%rsp - 0x00	
%rsp - 0x08	
%rsp - 0x10	
%rsp - 0x18	
%rsp - 0x20	
%rsp - 0x28	
%rsp - 0x30	
%rsp - 0x38	
%rsp - 0x40	
%rsp - 0x48	
%rsp - 0x50	

1. In the box above, draw the stack as it would appear just before instruction 0x4005b5 after calling the function "set_notification_time(0,1,9)". (Meaning user=0, day=Tuesday, time=9:00am). Draw everything you know about the stack. Hint: you won't know all the data values to put into the stack, instead, label the stack with appropriate variable names (like "username, bytes 0-7").

2. With what arguments could you call "set_notification_time" to give yourself ONE MILLION dollars? (assume your user_id is 10)

3. With what arguments could you call "set_notification_time" to delete all the accounts (the users array)?