

Computer Organization
Fall 2002
Final Exam

Name _____ **Answer Key** _____

There are 8 pages - make sure you have all of them.
Answer all questions - pay attention to the # of points for each question.
Don't leave anything blank - partial credit is always possible!

Question 1 (10pts) : Write Y86 Assembly code that provides the same functionality as the following IA32 instruction. The values in `%ebx`, `%esi` and `%eax` are established before the code starts; you can use any other registers to hold intermediate values. **IMPORTANT:** your code must not change `%ebx` or `%esi` (since this instruction does not change them)!.

```
addl 12(%ebx,%esi,2),%eax
```

```
rrmovl %esi,%edx      # edx <- esi
addl   %edx,%edx      # edx <- esi*2
addl   %ebx,%edx      # edx <- esi*2 + ebx
rrmovl 12(%edx),%edx  # edx <- Mem[12+esi*2 + ebx]
addl   %edx,%eax      # eax <- eax + Mem[12+esi*2 + ebx]
```

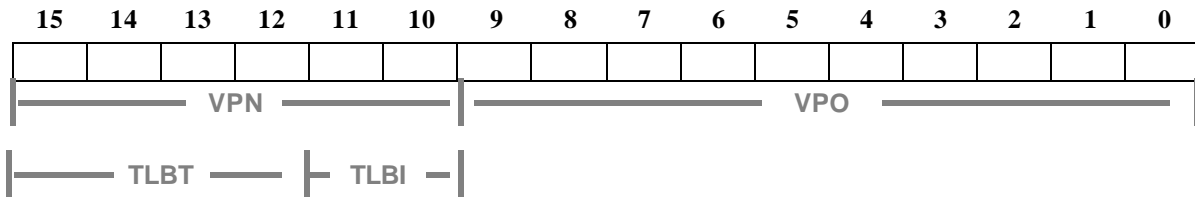
Question 2 (15 pts): The following problem concerns the way virtual addresses are translated into physical addresses. The memory system is described below:

- The memory is byte addressable.
- Virtual addresses are 16 bits wide.
- Physical addresses are 14 bits wide.
- The page size is 1024 bytes.
- The TLB is 4-way set associative with 16 total entries.

2a (8 pts):

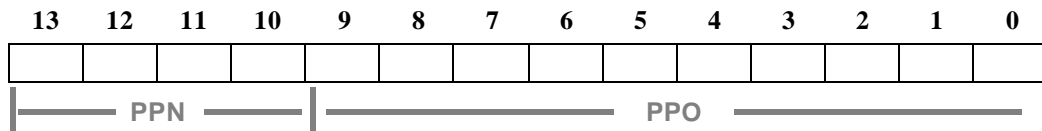
The box below shows the format of a virtual address. Indicate (by labeling the diagram) the fields (if they exist) that would be used to determine the following: (If a field doesn't exist, don't draw it on the diagram.)

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



2b (4 pts): 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



2c (3pts): What is the maximum size of a page table used with this memory system (how many page table entries are necessary to provide mappings for all possible virtual addresses for a single process?). Provide the number of entries necessary (don't worry about the size of each entry in the page table, just the number of page table entries).

The number of page table entries is the same as the number of virtual pages.

Number of virtual pages is $2^6 = 64$

Question 3 (8 pts): Based on the assembly code below, fill in the blanks below in its corresponding C source code. (Note: you may only use the symbolic variables `x`, `y`, and `result` in your expressions below — *do not use register names.*)

```
loop:
    pushl %ebp
    movl %esp,%ebp
    movl 8(%ebp),%ecx
    movl 12(%ebp),%edx
    xorl %eax,%eax
    cmpl %edx,%ecx
    jle .L4
.L6:
    decl %ecx
    incl %edx
    incl %eax
    cmpl %edx,%ecx
    jg .L6
.L4:
    incl %eax
    movl %ebp,%esp
    popl %ebp
    ret
```

```
int loop(int x, int y)
{
    int result;

    for ( result=0 ; x > y; result++ ) {
        x--;
        y++;
    }
    result++;
    return result;
}
```

Question 4 (8pts): List the advantages and disadvantages of caching virtual addresses vs. caching physical addresses:

	Advantages	Disadvantages
cache virtual addresses	No address translation is necessary before checking the cache (speed).	Two processes might use the same virtual address to access different physical memory locations, so the cache either needs to be cleared out each time we change processes, or additional information (a process identifier) needs to be in the cache.
cache physical addresses	Cache hardware is independent of the virtual memory system.	Overall hit time is slower, since address translation must happen before cache lookup..

Question 5 (5pts): Which of the C functions below compiled into the assembly code shown?

```
int fun1(int *ap, int *bp){
    int a = *ap;
    int b = *bp;
    return a+b;
}

int fun2(int *ap, int *bp){
    int b = *bp;
    *bp += *ap;
    return b;
}

int fun3(int *ap, int *bp){
    int a = *ap;
    *bp += *ap;
    return a;
}
```

```
pushl %ebp
movl %esp,%ebp
movl 8(%ebp),%edx
movl 12(%ebp),%eax
movl %ebp,%esp
movl (%edx),%edx
addl %edx,(%eax)
movl %edx,%eax
popl %ebp
ret
```

fun3() is the right answer.

Question 6 (20 pts)? Fill in the blanks, 2 points each.

- The 4 bit 2's complement representation of the number -7 is 1001
- The name of the dedicated cache that speeds up page table lookups is the Translation Lookaside Buffer (TLB)
- Using the IEEE 32-bit floating point format, show the 23 bit representation of the significand (just the significand!) for the value 2.0_{10}
00000000000000000000000
- The logical AND operator in the C programming language looks like this: & (also accepted &&)
- According to the HCL expression:

```
int x = [  
    y in {0,1,2} : 13;  
    z in {0,13,25} : 1;  
    1 : 25  
];
```

if y has the value 13, the value of x will be x = [z in {0,13,25}:1; 1:25];
- The IA32 instruction used to add two signed, 32 bit integers is addl
- The IA32 instruction used to add two unsigned, 32 bit integers is addl
- How long is an address (how many bits) in a memory system in which each address corresponds to a single byte, and which can address 2Mbytes of memory? 21 bits
- Is the following statement true or false? false
"The protection provided by a virtual memory system makes it impossible for the kernel (the operating system) to change the value of any memory location that is dedicated for use by your process"
- How do you spell Unix (in your choice of French, Russian or Japanese) ? Unix

- **Question 7 (6pts):** Assume that you have a fixed cache size, and a function that accesses a single array of integers. You run the function many times. Fill in the table below to indicate the impact of changing the stride and working set size on temporal locality, spatial locality and on overall memory performance. (For example: will temporal locality be exploited more or less if we increase the stride?)

	Temporal Locality	Spatial Locality	Overall Memory Performance
Increasing the stride	No effect (The exploitation of temporal locality will not change when stride is increased).	Spatial Locality will be exploited less as the stride is increased.	Overall memory performance will get worse as stride is increased.
Increasing the array size	Temporal Locality will be exploited less as the array size is increased (once the cache won't hold the array).	No effect. (The exploitation of spatial locality will not change when the array size is increased).	Overall memory performance will get worse as the working set size increases

Question 8 (5 pts): We want to design a circuit that finds the minimum value among a set of words A, B and C. Describe this circuit using an HCL expression:

```
int min = [
  A<=B && A<=C : A;
  B<=A && B<=C : B;
  1: C;
];
```

Question 9 (10pts) This question involves a couple of strategies used to time a C function.

a (5pts): A system with a timer interval of 10ms is used to time a C function (using interval timing) and the resulting count indicates that the function takes 40ms (combining both user and system time). What are the minimum and maximum actual times used by the function?

Minimum is 30ms (plus one cycle).
Maximum is 50ms (minus one cycle).

b (5pts): The same function is timed using cycle counting, and the following times are measured (the function is run many times, the numbers below indicate the time measured using cycle counting):

38.0, 43.0, 74.5, 38.1, 117.5, 121.0, 37.9, 112.2, 38.0, 38.0

Using the K-Best measurement scheme with $K = 3$ and the tolerance set to 1%, what would the result be (what is the estimate of the run time of the function)?

38, 38.1, 37.9 satisfy the tolerance, estimate is the minimum: 37.9

Question 10 (5pts) Consider the following C code segment. Suggest some changes to the code that *will* improve the performance and state whether you think the change in performance will be significant *and why*.

```
int i;
int a[MAXSIZE];

for (i=0;i<length(a);i++) {
    a[length(a)-i]++;
}

int length(int x[MAXSIZE]) {
    return(x[0]);
}
```

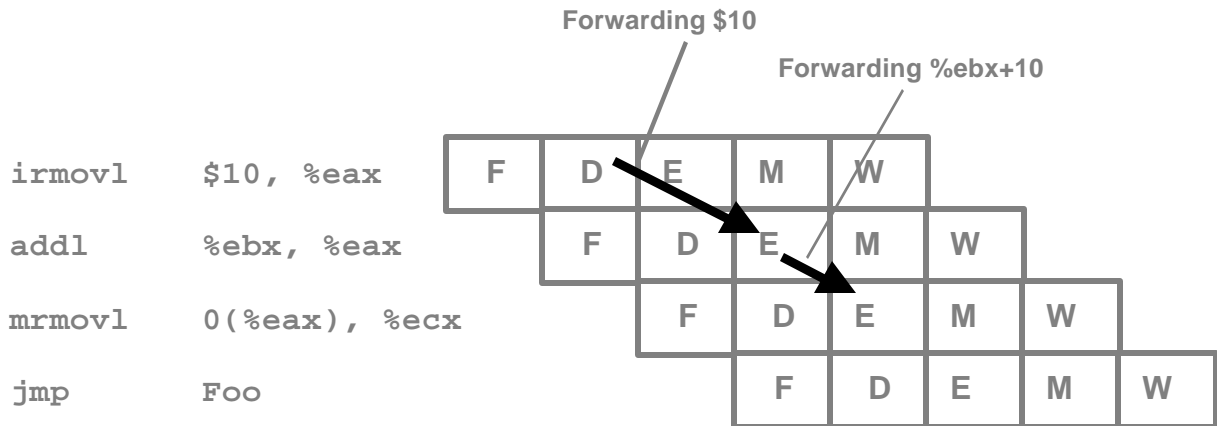
- moving the function call `length(a)` out of the loop will have a significant impact.
- removing the function call entirely (replacing with inline code) will have an impact only if left in the loops.
- Using pointers instead of arrays will improve performance, how much depends on how well the compiler did when generating assembly code (could be significant if the compiler generates full array access inside the loop).

Question 11(8 pts): Identify all the hazards (data hazards and control hazards) in the following Y86 code and state whether the each hazard can be avoided by forwarding or whether some stall cycles are necessary (indicate the number of stall cycles necessary). Assume the 5 stage pipeline we looked at, with stages Fetch, Decode, Execute, Memory and Writeback.

```

irmovl    $10, %eax
addl     %ebx, %eax
mrmovl   0(%eax), %ecx
jmp      Foo

```



There is a data hazard in the second instruction, the value \$10 has not been put in register eax when the addl looks for it there. No stalls are necessary if forwarding is used.

There is a data hazard in the third instruction, the result of the previous instruction has not been saved in eax when the mrmovl instruction looks for it there. No stalls are necessary, the value can be forwarded from the result of the execute stage of the addl instruction.

There are no control hazards. The jmp instruction is unconditional, so there is no problem (the address of the next instruction is always Foo, this doesn't depend on anything).