

Computer Organization

Spring 2004

Final Exam

Name _____ Answer Key _____

There are 9 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 (5 pts) : The C function `login` is shown below, the assembly code produced by `gcc` for this function is also shown. It is possible to force `login` to return true without knowing what `validpassword` is, your job is to tell me what to type in to make this happen (show me a password that will result in a valid login no matter what value of `validpassword` is passed in). It is essential that the function returns normally, we can't have a SEGV or any corrupt register values!

Notes:

`gets` reads a string from `stdin` (reads until it sees a newline) and stores this string at the address passed to `gets`.
`strcmp` returns 0 if the two strings are equal, otherwise it returns either -1 or 1.

```
/* login returns true (non zero) if the password
   typed by a user matches validpassword,
   otherwise it returns false (0).
*/
int login(char *validpassword) {
    int validlogin=0;
    char newpass[8];

    gets(newpass);
    if (strcmp(newpass,validpassword)==0)
        validlogin=1;
    return(validlogin);
}
```

```
login:
    pushl   %ebp
    movl   %esp, %ebp
    subl   $24, %esp
    movl   $0, -4(%ebp)
    leal   -16(%ebp), %eax
    pushl   %eax
    call   gets
    addl   $4, %esp
    pushl   8(%ebp)
    leal   -16(%ebp), %eax
    pushl   %eax
    call   strcmp
    testl  %eax, %eax
    jne    .L2
    movl   $1, -4(%ebp)
.L2:
    movl   -4(%ebp), %eax
    leave
    ret
```

The Secret Password is: any string with length 13 (or 14, 15)

Question 2 (11 pts) The following problem concerns cache lookups.

- Memory is byte addressable.
- Memory accesses are to bytes (not 4-byte words).
- Physical addresses are 10 bits wide.
- The cache is direct mapped and is 32 bytes.
- The block size is 4 bytes.

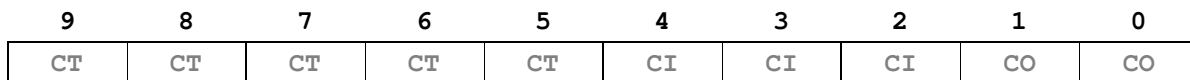
In the following table, all numbers are given in hexadecimal. The contents of the cache are as follows:

Direct Mapped Cache

Index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	09	1	86	30	3F	10
1	0F	1	60	4F	E0	23
2	13	0	2F	81	FD	9
3	06	0	3D	94	9B	F7
4	C7	1	06	78	07	C5
5	0B	1	0B	DE	18	4B
6	5	1	A0	B7	26	2D
7	16	0	B1	0A	32	0F

2a (6 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:

- CO The block offset within the cache line
- CI The cache index
- CT The cache tag



2b (5 pts) For the address 0×175 , indicate the cache entry accessed and the cache byte value returned in hex. Indicate whether a cache miss occurs. If there is a cache miss, enter "-" for "Cache Byte returned".

Parameter	Value
Byte offset	0x 01
Cache Index	0x 05
Cache Tag	0x 0B
Cache Hit? (Y/N)	Yes
Cache Byte returned	0x DE

Address is 01011 101 01

Question 3 (7 pts): Consider the following C code for a procedure `divide_array()`:

```
/* function used by many web browsers when determining the
   likelihood that some javascript code is nasty and should be
   ignored by the browser.

   divides each element of the array by d,
   and does this x times, where x is the length of the
   string url

   NOTE: n can be quite large (into the millions, depending on
         the IP address of the web server).
*/

void divide_array(char *url, float *f, int n, float d) {

    int i, j;

    for (i=0; i<strlen(url); i++) {
        for (j=0; j<n; j++) {
            f[j] = f[j]/d;
        }
    }
}
```

There are a number of places where this function can be optimized. Describe two of them (two different optimizations), including rewriting the code and predicting whether the change will make the function a little faster or a lot faster.

The call to `strlen` should be moved out of the loop, since it will return the same value each time. This should make a big difference if the length of `url` is large.

```
int len = strlen(url);

for (i=0; i<len; i++)
```

The division can be replaced with multiplication (division is about 10 times slower!). This will make a big difference if `n` is large.

```
float tmp = 1/d;

    f[j] = f[j] * tmp;
```

Can also consider loop-unrolling for either (or both) loops, although after elimination of the `strlen` call each iteration of the outer loop, the loop overhead is not large (so this probably won't make a big difference no matter how the `url` or `n` is).

You can actually rewrite the code to execute the `i=0; i<len` loop once, calculating the total divisor, then execute the `j` loop once dividing each element by the total divisor (very significant improvement).

Question 4 (10 pts): The following questions are about virtual memory, each should be answered with a sentence or two (no essays!), each is worth 2 points.

- List two reasons (motivations) for having a virtual memory system.

Simplifies linking and loading – each process can pretend it has the entire address space and can use any address it wants.

Protection – makes it impossible for one process to access the memory of another process.

Allows each process (and all processes together) to pretend there is a lot more memory than is physically installed.

- Describe (briefly) how a virtual address is converted to a physical address.

Page number is extracted from virtual address (leftmost bits other than page offset) and used as index into page table. Page table entry holds the physical page number of the page holding the memory location (or a flag indicating that it is currently on disk).

- What is a *Translation Lookaside Buffer* ?

Dedicated cache for page table entries (caches only data that makes up a page table).

- Briefly compare the performance of a virtual memory system with and without a TLB.

Without TLB each access by CPU to memory requires 2 accesses to physical memory – one to get the appropriate entry from the page table, the second to actually get the data found in the resulting page.

With TLB we can expect something much closer to 1 access to physical memory per memory reference by the CPU (we assume that most page table entries used are in the TLB which is fast).

- What is a *page fault* and why is a significant effort made to minimize the number of page faults?

Page fault occurs when a program references a virtual memory location that is not in memory (has been moved to disk). This requires that the operating system free up a page in physical memory (possibly requires writing to disk), and that the page holding the referenced data is moved from disk to physical memory. Since disk is very slow, this results in a delay (to the process) of millions of cycles.

Question 5 (20 pts) Short answer, 2 points each.

- Using a 11-bit two's complement representation, what is the maximum integer that can be represented (show your answer in decimal). $01111111111 = 1023$
- Using a 11-bit two's complement representation, what is the minimum integer that can be represented (show your answer in decimal). $10000000000 = -1024$
- How many page-table entries are necessary (for one process) with a virtual memory system in which virtual addresses (byte addresses) are 16 bits long and each page is 1024 bytes? $2^{16}/2^{10} = 2^6 = 64$
- A hard disk spins at 120 revolutions per minute. What is the average *rotational latency* – the time waited for the right sector of a track to be under the head (once the head is over the right track)? $.5$ seconds/revolution.
 $\frac{1}{2}$ revolution = $\frac{1}{4}$ second.
- 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 256K bytes of memory? $256K = 2^{18}$, so 18 bits needed
- Name a single Y86 instruction that writes to a memory location and to a register. `pushl` (stores on stack, writes to `%esp`)
`call` (stores on stack, changes `%eip`)
- Consider the addition of 8-bit **unsigned** integers: `c = a + b;`
Show a C expression that would evaluate to true only if the value computed is wrong (if overflow occurred and the correct answer won't fit in an 8 bit unsigned integer) `c < a || c < b`
unsigned! it's not possible for any of these to be `< 0!`
- Name a single Y86 instruction that reads the Condition Code register (not an instruction that changes the CC, just one that uses the CC). `JNE` (any conditional jump)
- Is IA32 CISC or RISC ? `CISC`
- Create a C program variable name inspired by your favorite donut. `double fudge;`
`float inginfat;`

Question 6 (15 pts) Longer short answer, 3 points each

- Consider the following fragment of IA32 code:

```
0x400446e3 <foo+7>: call 0x400446e8 <foo+12>
0x400446e8 <foo+12>: popl %eax
```

After the `popl` instruction completes, what hex value does register `%eax` contain?

0x400446e8

- Using IEEE single precision (32 bit) representation, what is the encoding of the significand in the representation of the number 11.75_{10} (hint: the significand is represented using 23 bits). 01111000000000000000000

- Using a K-Best measurement scheme ($K=3$, tolerance = 1.0), with 5 initial measurements ($n=5$), what is the reported time given the following sequence of measurements? The reported time should be the average of the K-best measurements. average is about 1.6

2.5, 1.4, 5.3, 2.2, 2.7,
 1.3, 5.3, 2.1, 6.4, 1.4,
 7.5, 1.3, 1.3, 8.8, 3.2

- On a system with a timer interval of 5ms, some (continuous) segment of process A is recorded as using 40 ms. What is the actual maximum time that could have been used by this segment? By continuous, we mean that A runs continually without interruption during this segment (except for the timer itself). < 45ms

- In the Y86 Sequential implementation based on the datapath shown on the crib sheet, explain why we can't create a instruction that adds 1 to a memory location, without changing the hardware (the datapath). can't read and write from memory in the same instruction – only one interface to the memory (only 1 memory operation/instruction).

Question 7 (6pts) Describe two cache write policies, include a brief list of pros and cons of each.

Write-through: all writes are written to memory (and possibly also to the cache).

Write-back: writes to cache only (might need to allocate). Memory is updated for the entire block when the block is removed from the cache.

Question 8 (6pts): Show a program that has the property that when a relatively simple change is made to the program (that does not effect what the program computes, but only changes the order of memory accesses) the run time can change significantly. Make sure you state which version is faster!

Basic 2-D array access pattern, good stride vs poor

Here is good memory access pattern:

```
for (i=0; i<n; i++)
  for (j=0; j<n; j++)
    something with a[i][j]
```

Here is poor memory access pattern:

```
for (j=0; j<n; j++)
  for (i=0; i<n; i++)
    something with a[i][j]
```

Question 9 (12 pts): Consider a direct mapped cache of size 4K bytes. You will calculate the miss rate for the following code using this cache. Remember that `sizeof(int) == 4`. **For each question below, assume that the cache starts empty** and that local variables and computations take place completely within the registers (only the matrix is in memory).

Now consider the following code to sum all the elements in the matrix. Assume that array `m` starts at address 0.

```
int sum_matrix(int m[SIZE][SIZE]) {
    int i, j;
    int tot=0;

    for (i=0; i<SIZE; i++) {
        for (j=0; j<SIZE; j++) {
            tot += src[i][j];
        }
    }
    return (tot);
}
```

9a (3pts) What is the cache miss rate if `SIZE = 32`, the cache block size is 8 bytes and we run the function exactly once?

each block loaded into cache once, both elements accessed.
50% miss rate

9b (3pts) What is the cache miss rate if `SIZE = 32`, the cache block size is 8 bytes and we run the function 10 times in a row (with no other code using the cache)?

`m` is $32*32*4=4K$, so entire array fits in the cache. last 9 loops are all hits.
5% miss rate

9c (3pts) What is the cache miss rate if `SIZE = 32`, the cache block size is 16 bytes and we run the function exactly once?

each block holds 4 elements, so miss rate is 25%

9d (3pts) What is the cache miss rate if `SIZE = 128`, the cache block size is 8 bytes and we run the function exactly once?

Nothing changes here since we only run once – still 50% miss rate

Question 10 (8 pts): Consider the source code below, where M and N are constants declared with #define.

```
int array1[M][N];
int array2[N][M];

int copy(int i, int j)
{
    array1[i][j] = array2[j][i];
}
```

Suppose the above code generates the following assembly code:

```
copy:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
    movl 8(%ebp),%ecx          # ecx is i
    movl 12(%ebp),%ebx        # ebx is j
    leal (%ecx,%ecx,8),%edx    # edx is i*9
    sall $2,%edx              # edx is i*36
    movl %ebx,%eax            # eax is j
    sall $4,%eax              # eax is j*16
    subl %ebx,%eax            # eax is j*16-j (15*j)
    sall $2,%eax              # eax is 60*j
    movl array2(%eax,%ecx,4),%eax # eax is M[array2+i*4+60j]
    movl %eax,array1(%edx,%ebx,4) # put eax in M[array1+i*36+j*4]
    popl %ebx
    movl %ebp,%esp
    popl %ebp
    ret
```

What are the values of M and N? Partial credit is possible (but only if you show your work!).

$$M = 60/4 = 15$$

$$N = 36/4 = 9$$