

# Computer Organization

Fall 2002

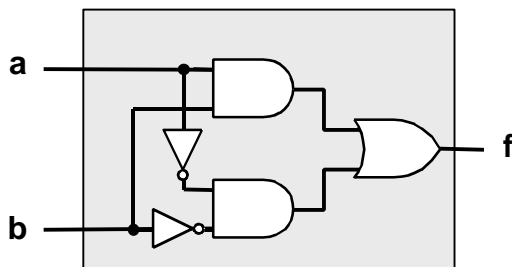
Test #2

Name Answer Key

There are 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 ( 25 pts):** Short answer questions – 5 points each!

**Question 1a:** Write an HCL expression for **f** in terms of **a** and **b** based on the following logic diagram:



```
bool f = (a &&b) || (!a && !b)
```

**Question 1b:** Write Y86 Assembly code that provides the same functionality as the following IA32 instruction. The values in %edx, %esi and %eax are established before the code starts; you can use any other registers to hold intermediate values.

```
addl 100(%edx,%esi,2),%eax
```

```
rrmovl %esi,%ebx      # ebx=esi
addl   %ebx,%ebx      # ebx=2*esi
addl   %edx,%ebx      # ebx=edx+2*esi
mrmovl 100(%ebx),%ebx # ebx = mem[100+edx+2*esi]
addl   %ebx,%eax      # eax = eax + mem[100+edx+2*esi]
```

There are lots of other ways to do this!  
 Make sure they only use Y86 instructions!

## Answer Key

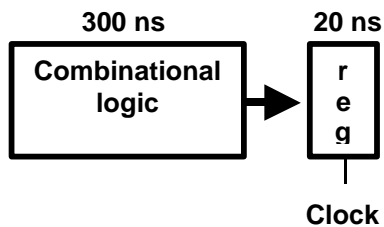
**Question 1c:** The time to access the first word of a specific sector of a hard disk depends on the rotational speed of the disk, the average seek time and the number of sectors per track. If we want not just the first word, but an entire sector (assume a sector holds about 100 words), how much longer will it take: about 100 times longer, about 10 times longer, or about the same amount of time? Explain your answer!

About the same amount of time.

The access time is dominated by the rotational speed and the seek time, once the head is positioned to read part of a sector the remaining bytes in that sector are effectively free.

**Question 1d:** Using a sequential implementation, it takes a total of 320ns for each instruction, 300ns for the combinational logic to complete, and 20 ns to store the results (in a register). This means that the throughput will be about 3.12 Million instructions/second. Assuming you switch to a 3 stage pipeline by splitting the combinational logic in to three equal parts, and all registers take 20ns to store results:

- How long will it take for a single instruction to execute in the pipelined implementation?
- Assuming the pipeline never stalls, what will the improvement in throughput be (just show the calculation, you don't need to come up with a number).



2 points: pipelined version will take  $3 \times 100 + 3 \times 20 = 360$ ns per instruction.

3 points: Throughput will improve to:

3 Instructions/360ns (about 8.33 Million instructions/second).

**Question 1e:** Describe the principles of *spatial* and *temporal locality* and describe how a memory hierarchy exploits them.

Spatial Locality means that whenever a memory element is accessed, it is likely that another nearby element will be accessed in the near future.

Temporal Locality means that whenever a memory element is accessed, it is likely that the same element will be accessed again in the near future.

A memory hierarchy exploits temporal locality by moving a memory element to a much faster memory whenever it is accessed (so the next access will be fast).

Spatial locality is exploited by moving not just the element that was accessed, but also some nearby memory elements (all elements in a block).

## Answer Key

**Question 2 (15 pts):** You need to evaluate the cache performance for the code shown below given a few different cache designs. For all cases you should assume that the address of the array `foo` is 0. You should assume that `i`, `j`, `tot` and `zcnt` are in registers (the only memory accesses are to `foo`). **SHOW YOUR WORK!**

```
int foo[256][2];

int i, j;
int tot=0;
int zcnt=0;

for (i=0; i<256; i++) {
    for (j=0; j<2; j++) {
        tot += foo[i][j];
    }
}

for (i=0; i<256; i++) {
    for (j=0; j<2; j++) {
        if (foo[i][j]==0)
            zcnt++;
    }
}
```

**Question 2a (5 pts):** What is the miss rate for the code assuming the cache holds 1024 bytes of data, is direct mapped and has a block size of 8 bytes?

First loop is 50% miss rate (every miss brings along the next word, which will be a hit).  
Second loop is also 50% miss rate, the array doesn't fit in the cache.

Total miss rate is 50%

**Question 2b (5 pts):** What is the miss rate if we increase the block size to 16 bytes (still direct mapped, 1024 bytes total).

First loop is 25% miss rate (every miss brings along the next 3 words, which will be all be hits).

Second loop is also 25% miss rate, the array doesn't fit in the cache.

Total miss rate is 25%

**Question 2c (5 pts):** What is the miss rate if the cache holds 2048 bytes, is direct mapped and has block size 8 bytes?

First loop is 50% miss rate (every miss brings along the next word, which will be a hit).  
Second loop is 0% miss rate, the array does now fit in the cache.

Total miss rate is 25%

# Answer Key

**Question 4 (20 pts):** A C function named foo is compiled by gcc to IA32 assembly language, the result is shown below. Based on the assembly code, fill in the blanks in the C source code. You can only use C variable names (no register names) and all accesses to elements of a must be shown using array notation.

```
foo:
    pushl %ebp
    movl %esp,%ebp
    pushl %esi
    pushl %ebx
    movl 8(%ebp),%ebx
    movl 12(%ebp),%esi
    xorl %edx,%edx
    xorl %ecx,%ecx
    cmpl %ebx,%edx
    jge .L25
.L27:
    movl (%esi,%ecx,4),%eax
    cmpl %edx,%eax
    jle .L28
    movl %eax,%edx
.L28:
    incl %edx
    incl %ecx
    cmpl %ebx,%ecx
    jl .L27
.L25:
    movl %edx,%eax
    popl %ebx
    popl %esi
    movl %ebp,%esp
    popl %ebp
    ret
```

```
int foo(int n, int *a) {
    int i;
    int x = _____;

    for(i = _____; _____; i++) {
        if (_____ )
            x = _____;
        _____;
    }
    return x;
}
```

```
int foo(int n, int *a) {
    int i;
    int x = 0;
    for(i = 0; i < n; i++) {
        if (a[i] > x)
            x = a[i];
        x++;
    }
    return x;
}
```

2 points

4 points

or !a[i] <= x;  
6 points

or x=x+1;  
2 points

6 points

NOTE: some people will get the parameters wrong (think a is at 8(%ebp) and n is at 12\*%ebp). They should lose points, but should get partial credit if other things are wrong because of this mistake.

# Answer Key

**Question 5 (40 pts):** We want to add a new instruction to the Y86 sequential implementation. The new instruction will be called at the beginning of subroutines instead of the 2 instruction sequence shown below:

```
push %ebp
movl %esp,%ebp
```

**Question 5a (20 pts):** The new Y86 instruction will be named **setup**, this will be a single byte instruction that is encoded as **E0**. Fill out the following form to describe what needs to happen during each stage of this new instruction, a copy of the form for the push instruction is included for reference.

	<b>pushl rA</b>	<b>setup</b>
<b>Fetch</b>	icode:ifun <- M <sub>1</sub> [PC] rA:rB <- M <sub>1</sub> [PC+1] valP <- PC+2	icode:ifun <- M <sub>1</sub> [PC] valP <- PC+1
<b>Decode</b>	valA <- R[rA] valB <- R[%esp]	valA <- R[%ebp] valB <- R[%esp]
<b>Execute</b>	valE <- valB + (-4)	valE <- valB + (-4)
<b>Memory</b>	M <sub>4</sub> [valE] <- valA	M <sub>4</sub> [valE] <- valA
<b>Write Back</b>	R[%esp] <- valE	R[%esp] <- valE R[%ebp] <- valE
<b>PC update</b>	PC <- valP	PC <- valP



2 points for each (only 9 lines, 2 points are free)

**Question 5b (10 pts):** Modify the HCL expressions show below to support this new instruction. You can assume that the symbol ISETUP is defined.

```
int srcA = [
  icode in { IRRMOVL, IRMMOVL, IOPL, IPUSHL } : rA;
  icode in { IPOPL, IRET } : RESP;
  icode in { ISETUP } : REBP;
  1 : RNONE; # Don't need register
];
```

3 points

```
int aluA = [
  icode in { IRRMOVL, IOPL } : valA;
  icode in { IIRMOVL, IRMMOVL, IMRMOVL } : valC;
  icode in { ICALL, IPUSHL, ISETUP } : -4;
  icode in { IRET, IPOPL } : 4;
];
```

3 points

```
int dstE = [
  icode in { IRRMOVL, IIRMOVL, IOPL } : rB;
  icode in { IPUSHL, IPOPL, ICALL, IRET, ISETUP } : RESP;
  1 : RNONE; # Don't need register
];
```

4 points.  
 Could also say that dstE should be REBP for ISETUP!  
 Could also say there is a problem here!  
 These are all correct answers

**Question 5c (10 pts):** Assuming we have the hardware and interconnections shown in Figure 4.21 (attached to the back of this test), it is not possible to implement this new instruction! Describe what the problem is, and propose a solution (what needs to be added to 4.21 to make it possible to implement this new instruction?).

**The problem is that both %ebp and %esp must be set to the original value of \$esp minus 4. We currently can't save valE (which is %esp-4) in 2 places, one register can be set from valE and another from valM.**

**To fix this we could put a Multiplexor in front of the M input to the register file and have it pick between valE and valM. (There are other reasonable ways to fix this as well).**

**6 points for identifying the problem, 4 points for suggesting a reasonable solution.**