

Computer Organization

Fall 2005 Test #2

Answer Key

Name _____

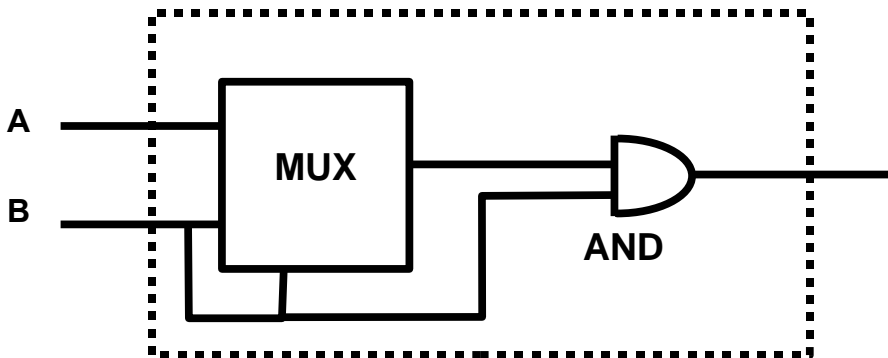
There are 6 questions - make sure you have all
 Answer all questions - pay attention to the # of points for each
 Don't leave anything blank - partial credit is always given

Truth Table

A	B	MuxOut	Out
0	0	0	0
0	1	1	1
1	0	1	0
1	1	1	1

Question 1 (15 pts): Short Answer – 5 points each.

a) What does this circuit compute?



The mux computes
 $A \text{ or } B$
 so the entire circuit computes:
 $(A \text{ or } B) \text{ and } B$
 which is always B
 The answer is "B"

b) Briefly describe why a ripple-carry adder is *slow*, and how a carry-lookahead adder is faster.

Each stage of a ripple carry adder depends on the output of the previous stage, so there is some wasted time waiting for the right input to the last stage. A carry-lookahead adder precomputes the carry bits, so that all stages can do useful computation at the same time.

c) For the IA32 assembly code segment shown below, assume that register `%edx` initially holds the value 3. What value is in `%eax` after the instructions in the code segment are executed?

<pre>movl %edx,%eax addl %eax,%eax leal (%eax,%eax,2),%eax sall \$2,%eax addl %edx,%eax</pre>	<p>call the original value in edx k</p> <p>$eax = k$</p> <p>$eax = 2k$</p> <p>$eax = 2k + 2(2k) = 6k$</p> <p>$eax = 4 * %eax = 24k$</p> <p>$eax = eax + k = 25k$</p>
--	--

Answer is 75 (25k where k is 3).

Question 2 (15 pts): Below is the C code and corresponding assembly code for a function that deals with a 2-dimensional array where NUMTESTS and NUMSTUDENTS are defined elsewhere. Your job is to determine what NUMTESTS and NUMSTUDENTS are by looking at the assembly code. Show your work (partial credit is available). Note that part of the assembly code is already commented for you (the division at the end of average_test_grade – we have not talked about integer division).

```
int testgrades[NUMTESTS][NUMSTUDENTS];

int average_test_grade(int student) {
    int test;
    int total=0;
    for (test=0;test<NUMTESTS;test++) {
        total += testgrades[test][student];
    }
    return(total/test);
}
```

NUMTESTS = **10**

NUMSTUDENTS = **22**

```
average_test_grade:
    pushl   %ebp
    movl   %esp, %ebp
    pushl   %ebx

    xorl   %edx,%edx    # edx is total
    xorl   %ebx,%ebx    # ebx is test

.L2:
    cmpl   $9, %ebx    # test < 10?
    jg     .L3

.L5:
    movl   %ebx, %eax   # eax = test
    sall   $2, %eax    # eax = test*4
    addl   %ebx, %eax   # eax = test*5
    addl   %eax, %eax   # eax = test*10
    addl   %ebx, %eax   # eax = test*11
    addl   %eax, %eax   # eax = test*22
    addl   8(%ebp), %eax
    movl   testgrades(,%eax,4), %ecx
    addl   %ecx,%edx
    inc    %ebx
    jmp    .L2

.L3:
    movl   %edx, %eax
    sarl   $31, %edx
    idivl  %ebx        # eax = eax / ebx
    popl   %ebx
    leave
    ret
```

NUMTESTS 

NUMSTUDENTS 

Question 3 (20 pts): Consider the C code for a function named `add_student()` and a `main()` that calls `add_student`. The C code and the corresponding IA32 assembly (generated by `objdump`) are shown below and on the next page of this test.

Assume that the program has executed from the beginning of `main`, up through the address `8048456`. The next instruction to be executed is the `ret` instruction at the end of `add_student`. Keep in mind the following:

- The leave instruction does the following:


```
movl %ebp,%esp
popl %ebp
```
- IA32 machines are little-endian.
- C strings are null terminated.
- a C int variable requires 4 bytes.
- There is an ASCII reference at the end of this test.

10 pts: What is the output of the program so far (what does the `printf` in `add_student` print out)?

Student grade is 706f6e6d q

5 pts: What is the value of register `%ebp`? (show this in hex) **0x00777675**

5 pts: If we continued execution, to what address will `add_student` return (what is the return address that will be used when `add_student` returns)? Show this in hex. **0x08048479**

The return address is not changed, the string is not long enough to reach the return address.

```
typedef struct {
    char name[10];
    int ave;
    char letter_grade;
} student_rec;

void add_student(char *s_name, char s_grade, int s_average) {
    student_rec r;
    r.letter_grade = s_grade;
    r.ave = s_average;
    strcpy(r.name,s_name);
    printf("Student grade is %x %c\n",r.ave,r.letter_grade);
}

int main() {
    add_student("abcdefghijklmnopqrstuvw", 'B', 84);
    printf("done\n");
    return(0);
}
```

The Stack

return address			
old %ebp			
letter_grade	unused	unused	unused
ave			
name[8]	name[9]	unused	unused
name[4]	name[5]	name[6]	name[7]
name[0]	name[1]	name[2]	name[3]

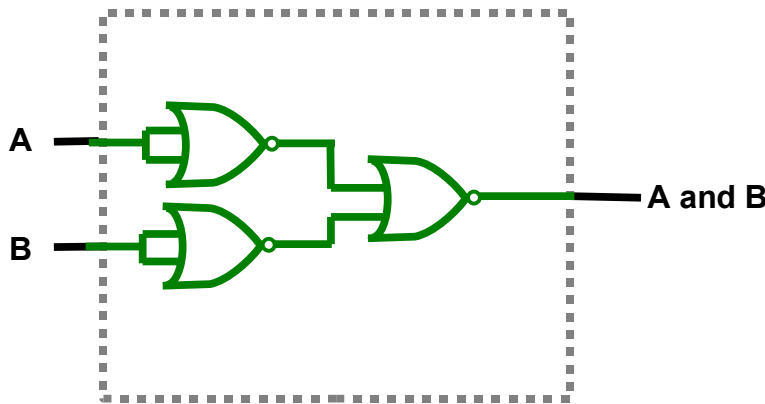
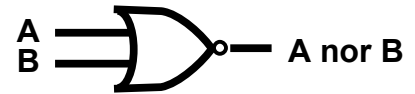
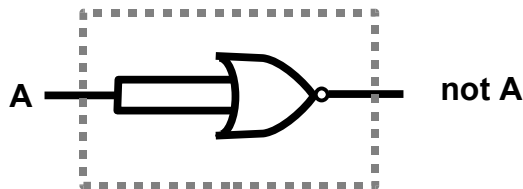
Question continued: objdump of add_student and main

```
08048420 <add_student>:
8048420:    55                push   %ebp
8048421:    89 e5             mov    %esp,%ebp
8048423:    83 ec 14         sub   $0x14,%esp
8048426:    8b 45 0c         mov   0xc(%ebp),%eax
8048429:    8b 55 08         mov   0x8(%ebp),%edx
804842c:    88 45 fc         mov   %al,0xffffffff(%ebp)
804842f:    8b 45 10         mov   0x10(%ebp),%eax
8048432:    89 45 f8         mov   %eax,0xffffffff8(%ebp)
8048435:    8d 45 ec         lea  0xffffffffec(%ebp),%eax
8048438:    52              push  %edx
8048439:    50              push  %eax
804843a:    e8 11 ff ff ff   call  8048350 <strcpy>
804843f:    83 c4 0c         add   $0xc,%esp
8048442:    0f be 45 fc     movsbl 0xffffffff(%ebp),%eax
8048446:    50              push  %eax
8048447:    8b 45 f8         mov   0xffffffff8(%ebp),%eax
804844a:    50              push  %eax
804844b:    68 a8 85 04 08   push  $0x80485a8
8048450:    e8 eb fe ff ff   call  8048340 <printf>
8048455:    c9              leave
8048456:    c3              ret

08048460 <main>:
8048460:    55                push   %ebp
8048461:    89 e5             mov    %esp,%ebp
8048463:    51              push  %ecx
8048464:    51              push  %ecx
8048465:    83 e4 f0         and   $0xfffffffff0,%esp
8048468:    83 ec 14         sub   $0x14,%esp
804846b:    6a 54             push  $0x54
804846d:    6a 42             push  $0x42
804846f:    68 c8 85 04 08   push  $0x80485c8
8048474:    e8 a7 ff ff ff   call  8048420 <add_student>
8048479:    c7 04 24 c0 85 04 08 movl  $0x80485c0,(%esp)
8048480:    e8 9b fe ff ff   call  8048320 <printf>
8048485:    31 c0             xor   %eax,%eax
8048487:    c9              leave
8048488:    c3              ret
```

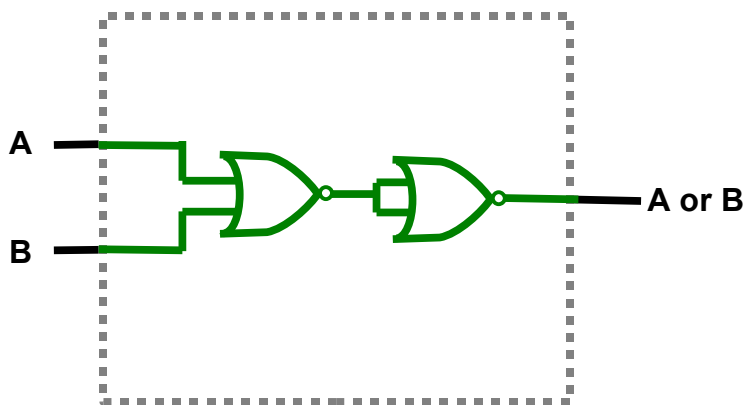


Question 4 (15 pts): Boolean algebra is defined over the binary operations AND and OR, and the unary operation NOT. This means that if we can build AND, OR and NOT gates, we can build anything (that can be defined by boolean algebra). Your roommate figures out how to build a NOR gate out of Pizza boxes (of which you have an endless supply). Prove that you can build anything defined by boolean algebra with these pizza boxes, by showing how you can compute the functions AND, OR and NOT using nothing more than NOR gates. I'm starting you off by showing how you can compute NOT using a NOR gate, you need to do the same for AND and OR. The Truth Table for NOR is shown below.



NOR Truth Table

A	B	A nor B
0	0	1
0	1	0
1	0	0
1	1	0



Question 5 (25 pts): The following C function returns the maximum integer found in an array of integers. n is the number of elements in the array.

```
int max(int *x, int n) {
    int m;
    if (n==1){
        return(x[0]);
    } else {
        m = max(x+1,n-1);
        if (m>x[0])
            return(m);
        else
            return(x[0]);
    }
}
```

Translate this C code to an IA32 assembly language subroutine that follows the GCC calling convention and register usage conventions. Recall that %ebx, %esi and %edi are *callee-save* registers.

IMPORTANT! Your IA32 subroutine must be code that could be generated from the above C code, so it must be recursive!

```
max:
    pushl   %ebp
    movl   %esp, %ebp
    pushl   %ebx                # save %ebx
    movl   8(%ebp), %ebx        # ebx is x
    movl   12(%ebp), %eax       # eax is n
    cmpl   $1, %eax            # n == 1 ?
    je     .L6                  # yes - jump
    subl   $8, %esp             #
    decl   %eax                 # eax = n-1
    pushl   %eax                # push n-1
    leal   4(%ebx), %eax        # eax is x+1
    pushl   %eax                # push x+1
    call   max                   # eax is max(x+1,n-1)
    addl   $16, %esp            #
    movl   %eax, %edx           # edx is max(x+1,n-1)
    cmpl   (%ebx), %eax         # max(x+1,n-1) > x[0]?
    jg     .L1                  #yes, done (eax is max)
.L6:
    movl   (%ebx), %edx         # x[0] is max
.L1:
    movl   %edx, %eax           # put max in eax
    movl   -4(%ebp), %ebx       # restore %ebx
    leave
    ret
```

Question (10 pts): The IA32 assembly code for a function named `blah` is shown below. Write a C function named `blah` that could be compiled into this assembly code. Don't just write a C function that computes the same thing, make sure your function could actually be compiled into this assembly code.

```
blah:
    pushl   %ebp
    movl   %esp, %ebp
    xorl   %eax, %eax      # tot = 0
.L2:
    cmpl   $0, 8(%ebp)    # is x == 0 ?
    jne   .L4             # no jmp
    jmp   .L3             # yes, done. jmp to end
.L4:
    movl   8(%ebp), %edx
    addl   %edx, %eax     # edx = x
    decl  8(%ebp)        # tot+=x
    jmp   .L2            # x--
.L3:
    leave
    ret
```

```
int blah(unsigned int x) {
    int tot=0;

    while(x) {
        tot+=x;
        x--;
    }
    return(tot);
}
```

IA32 Reference

IA32 Instructions

<code>movl Src, Dest</code>	<code>Dest = Src</code>
<code>addl Src, Dest</code>	<code>Dest = Dest + Src</code>
<code>subl Src, Dest</code>	<code>Dest = Dest - Src</code>
<code>imull Src, Dest</code>	<code>Dest = Dest * Src</code>
<code>sall Src, Dest</code>	<code>Dest = Dest << Src</code>
<code>sarl Src, Dest</code>	<code>Dest = Dest >> Src</code>
<code>shrl Src, Dest</code>	<code>Dest = Dest >> Src</code>
<code>xorl Src, Dest</code>	<code>Dest = Dest ^ Src</code>
<code>andl Src, Dest</code>	<code>Dest = Dest & Src</code>
<code>orl Src, Dest</code>	<code>Dest = Dest Src</code>
<code>incl Dest</code>	<code>Dest = Dest + 1</code>
<code>decl Dest</code>	<code>Dest = Dest - 1</code>
<code>negl Dest</code>	<code>Dest = - Dest</code>
<code>notl Dest</code>	<code>Dest = ~ Dest</code>
<code>leal Src, Dest</code>	<code>Dest = address of Src</code>
<code>cmpl Src2, Src1</code>	Sets CCs Src1 – Src2
<code>testl Src2, Src1</code>	Sets CCs Src1 & Src2
<code>jmp label</code>	jump
<code>je label</code>	jump equal
<code>jne label</code>	jump not equal
<code>js label</code>	jump negative
<code>jns label</code>	jump non-negative
<code>jl label</code>	jump greater (signed)
<code>jge label</code>	jump greater or equal (signed)
<code>jl label</code>	jump less (signed)
<code>jle label</code>	jump less or equal (signed)
<code>ja label</code>	jump above (unsigned)
<code>jb label</code>	jump below (unsigned)
<code>push Src</code>	<code>Mem[%esp-4] = Src,</code> <code>%esp = %esp-4</code>
<code>pop Dest</code>	<code>Dest = Mem[%esp],</code> <code>%esp = %esp+4</code>
<code>call label</code>	<i>push address of next instruction,</i> <code>jmp label</code>
<code>ret</code>	<code>%eip = Mem[%esp],</code> <code>%esp = %esp+4</code>

Addressing Modes

Immediate	<code>\$val</code>	Val
	<code>movl \$17, %eax</code>	
Normal	(R)	Mem[Reg[R]]
	Register R specifies memory address	
	<code>movl (%ecx), %eax</code>	
Displacement	D(R)	Mem[Reg[R]+D]
	Register R specifies start of memory region	
	Constant displacement D specifies offset	
	<code>movl 8(%ebp), %edx</code>	
Indexed	D(Rb, Ri, S)	Mem[Reg[Rb]+S*Reg[Ri]+ D]
	D: Constant "displacement" 1, 2, or 4 bytes	
	Rb: Base register: Any of 8 integer registers	
	Ri: Index register:	
	S: Scale: 1, 2, 4, or 8	
	<code>movl 0x100(%ecx, %eax, 4), %edx</code>	

Condition Codes

CF Carry Flag
ZF Zero Flag
SF Sign Flag
OF Overflow Flag

%eax

%ebx

%ecx

%edx

%edi

%esi

%ebp

%esp

ASCII Reference Chart (in Hex)

	30	0	40	@	50	P	60	`	70	p	
21	!	31	1	41	A	51	Q	61	a	71	q
22	"	32	2	42	B	52	R	62	b	72	r
23	#	33	3	43	C	53	S	63	c	73	s
24	\$	34	4	44	D	54	T	64	d	74	t
25	%	35	5	45	E	55	U	65	e	75	u
26	&	36	6	46	F	56	V	66	f	76	v
27	'	37	7	47	G	57	W	67	g	77	w
28	(38	8	48	H	58	X	68	h	78	x
29)	39	9	49	I	59	Y	69	i	79	y
2a	*	3a	:	4a	J	5a	Z	6a	j	7a	z
2b	+	3b	;	4b	K	5b	[6b	k	7b	{
2c	,	3c	<	4c	L	5c	\	6c	l	7c	
2d	-	3d	=	4d	M	5d]	6d	m	7d	}
2e	.	3e	>	4e	N	5e	^	6e	n	7e	~
2f	/	3f	?	4f	O	5f	_	6f	o		

Question	Possible	Score
1	15	
2	15	
3	20	
4	15	
5	25	
6	10	
Total	100	