Comp 150 Final Exam Overview.

Resources During the Exam

You may use notes on **four** sides of 8.5x11 inch paper (either both sides of two sheets, or four sheets written on single sides). Write this as you study! I mostly want to test you on concepts, not memorized rote facts. Opcode table supplied showing both Assembler mnemonics and binary opcodes.

Main topics that may be on the final exam: Previous exam topics on Python + a bit on client/server programming, web notes on binary arithmetic, Pip assembler, gates/circuits/truth tables, and SQL.

1. Previous exam topics on Python
2. Formatting with local variables: format parameter **locals()
3. Limited topics from cgi scripting: no HTML, but I could show you the output of my dumpcgi.cgi script from some web page form, as an indication of what is produced by a form, and show you an alternate cgi script to receive the information, and ask you what the script prints.
4. Converting between binary and decimal (both directions!) converting between binary and hexadecimal (the easy way, both directions)
5. Read/write Pip assembler and play computer, using all instructions except AND and CPZ. Understand the use of the accumulator and symbolic variables and labels for jumps. Follow and be able to write short computational sequences and if-else or while-loop logic with Pip assembler code
6. Be able to convert individual instructions between assembler and machine language (both ways). I will post a table of op code to assembler mnemonic conversions, like 0010 MUL, so you do not have to put that in your notes. (and still be able to play computer on Pip assembler code).
7. Understand how millions of circuit elements can be created simultaneously in chip fabrication.
8. Be able to convert any way between Boolean expressions, sequential logic circuits, and truth tables.
9. Understand circuits for adders and multiplexers
10. SQL SELECT, WHERE with =, inequalities, LIKE '...%..., IN, AND (*not* join – only one table at a time)

Exam emphases

1. Individual topics that are new since the last exam will be more emphasized than the topics you have been examined on before, probably meaning 45-55% of the exam will be on new topics.
2. Problems from later in the semester generally include skills needed from early in the semester implicitly, so most questions will not be straight from the early part of the course, though there may be some topics from earlier in the semester that did not get used much in the later part of the course.
3. The best characterization of the course is the course itself, but I have tried to give you tutorial work or homework on all topics, so reviewing your work is a good review. Looking at old exams or sample exams is a quick but *not complete* way to review the older material: You should have covered much more in all your work than there was space for in exams or even sample exams. Obviously if you missed something on an exam, it would be good to make sure you know it now, but exams involve a number of arbitrary choices and omissions, and different choices are likely to be made on the final. Major topics are likely to reappear, but often be treated from a somewhat different angle than last time, or combined in different ways. A mostly different collection of secondary topics is likely to be on the final.
4. I repeat: the best review of what you need to be able to do is to go over what you have worked on. If you need further exercises on any subject, let me know.

Same Instructions for studying and using th esample problems as on the last two exams

Study first and then look at the sample problems.... See earlier review pages for the rest of those instructions.

New sample problems start on the next page.

Review Problems for the Final Exam (Solutions follow the problems.)

1. Write a sequence of PIP Assembler or machine code instructions that will copy the value of memory location 130 into memory location 131. (You do not need to write a whole program -- no HLT required.)
2. Convert the PIP machine code to assembler
   00010100 00000111
   00000101 10000000
   00001111 00000000
3. Convert the PIP Assembler to Machine code
   LOD 129
   MUL #3
   HLT

Assem	Opcode
ADD	0000
AND	1000
CPL	1011
DIV	0011
HLT	1111
JMP	1100
JMZ	1101

Assem	Opcode
LOD	0100
MUL	0010
NOP	1110
NOT	1001
STO	0101
SUB	0001

4. Play computer with the silly program below, completing the log at the right, showing the machine state after each instruction. To save time, you may choose to show only those values that change at each line. To be consistent with the simulator display, I show columns for both the current and next IP addresses, but you only need to fill in the current IP address (left column) of the instruction just executed. The initial values are shown.

```
IP-->  ACCUM   X   Y
-----------------
--   0       0   0   0
 0   2      -5   0   0
```

```
Address    Assembler code
0          LOD #-5
2          STO X
4          MUL #-1
8          STO Y
8          CPL X
10         JMZ L1
12         LOD X
14         ADD Y
16         JMZ L2
18     L1: LOD X
20     L2: ADD X
22         JMP L3
24         SUB #1
26     L3: HLT
```

5. Convert the following code to Pip Assembler.
   if X == Z:
   Y = 3
   else:
   X = Y
   Z = X + Y

6. Draw a circuit diagram that corresponds to the following Boolean expression: A(B + (CA)')

A -

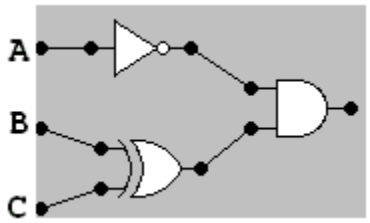
B -

C -

7. Complete the truth table below.

A	B	A'	A'B	A+B	A'B ⊕ (A+B)
0	0				
0	1				
1	0				
1	1				

8. Write a Boolean expression involving A, B, and C that corresponds to the following circuit:



9. Given the truth table below, write a Boolean expression in terms of A, B, and C for X.

A	B	C	X
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

10. Complete the truth table if X is true whenever B is different from both A and C

A	B	C	X
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

11. What is printed? Be careful to follow the order of execution, not the order of the text!

```
def foo(x):                #1
    return x*2             #2

def bar(a, n):             #3
    print(foo(n+1))        #4
    print(foo(a))          #5

print('go')                #6
bar('now', 4)              #7
```

12. Do the following base conversions. Show work.
    a. Convert the decimal number 54 into binary.
    b. Convert the binary number 1111001101100100 10 into hexadecimal, without converting the entire base 2 representation to base 10 first.

13. Do the following base conversions. Show work.
    a. Convert the hexadecimal 2AF to decimal.
    b. Convert the decimal 844 to hexadecimal.

14. Add the following binary numbers. Hint: Work place by place; add and each 2 makes a carry. Show work

```
    101011
  + 100110
```

15. What is printed? Hint: The list `nums` is modified while it is being referred to as `newVals` in `foobar`.

```
def foobar(oldVals, newVals):  #1
    for i in oldVals:          #2
        newVals.append(i+1)    #3

nums = [6]                     #4
foobar([1, 3, 8], nums)        #5
print(nums)                    #6
```

16. What is printed? Hint:

```
def f(x):
    return 2*x + 1

print(f(1), f(f(1)))
```

17. What is printed?

```
x = 16                  #1
while x > 2:            #2
  x = x/2               #3
  if x > 3 and x < 7:   #4
    print(3*x)          #5
  else
    print(x)            #6
```

18. What is printed? Be careful of the order of completion of the nested loops!

```
for s in ['abc', 'de', 'f']: #1
  for ch in s:               #2
    print(ch*2, end=' ')     #3
  print()                    #4
```

19. Write a function `upper2` that takes a single string `s` as parameter and *prints* the string twice on a line in upper case.

```
def  upper2(s):
```

20. Write a function that takes a single string `s` as parameter and *returns* the string repeated twice in upper case.

```
def  upper2ret(s):
```

21. Redefine the function `upper2` so it *uses* the function upper2ret.

```
def  upper2(s):
```

22. Write a function `printListUpper` that has a parameter `words`, which is a list of strings, and prints each in upper case on the same line. If names were ['hi', 'there'] then the following would be printed:
    HI THERE

```
def printListUpper(words):
```

23. Write a function `printListShortUpper` that has a parameter `words`, which is a list of strings, and prints each string *that is shorter than the numeric parameter n* in upper case on the same line. If `words` were ['hi', 'there'] and `n` were 4, then the following would be printed:
    HI

```
def printListShortUpper(words, n):
```

24. Write a function `newListUpper` that has a parameter `words`, which is a list of strings, and creates and returns a new list containing each string in upper case. If `words` were ['hi', 'there'] then ['HI', 'THERE'] would be returned.

```
def newListUpper(words):
```

25. The start of a database table is shown. Write SQL queries:

Classes				
name	section	prof	time	credits
Comp 150	1	Harrington	TTh 10AM	3
Comp 170	1	Akhtar	TTh 1PM	3
Comp 363	1	Harrington	Th 4:15PM	3

a. Find the name and time for all classes with prof "Harrington"
b. Find the name, prof, and time of all classes for more than 3 credits.

Answers on the next page

Final Exam Review Problem Answers

1. LOD 130
   STO 131

2. LOD #7 ; pound sign from 0001; LOD code 0100; 7 from binary 00000111
   STO 128 ; STO code 0101; 128 in binary 10000000
   HLT ; HLT code is 1111

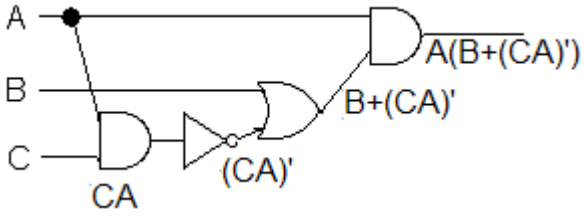
3. 00000100 10000001
   00010010 00000011
   00001111 00000000

4.

```
IP--> ACCUM  X Y
---------------
--  0     0  0 0
 0  2    -5  0 0 LOD #-5;acc=-5
 2  4    -5 -5 0 STO X ;X=acc=-5
 4  6     5 -5 0 MUL #-1 ;acc=-5*-1=5
 6  8     5 -5 5 STO Y ;Y=acc=5
 8 10     1 -5 5 CPL X ;-5<0 true acc=1
10 12     1 -5 5 JMZ L1; acc!=0;no jump
12 14    -5 -5 5 LOD X ; acc=X=-5
14 16     0 -5 5 ADD Y ;acc=acc+Y=-5+5=0
16 20     0 -5 5 JMZ L2 ;acc is 0; jump
20 22    -5 -5 5 ADD X ; acc=acc+X=0+-5
22 26    -5 -5 5 JMP L3 must jump
26 --    -5 -5 5 HLT
```

5.
```
      LOD X
      SUB Z  ; same as if x-z == 0
      NOT    ; same as if not x-z == 0
      JMZ ELSE
      LOD #3
      STO Y
      JMP PAST
ELSE: LOD Y
      STO X
PAST: LOD X
      ADD Y
      STO Z
```

6. (Could use NAND instead of AND and NOT)



7.

A	B	A'	A'B	A+B	A'B ⊕ (A+B)
0	0	1	0	0	0
0	1	1	1	1	0
1	0	0	0	1	1
1	1	0	0	1	1

8. A’(B ⊕ C)

9. A'B'C' + A'BC + ABC

10.

A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

11. go
    10
    nownow

```
line comment
6    print go  (earlier lines only definitions)
7    Call bar
3    a is 'now' and n is 4
4    n+1 is 4+1 is 5; call foo(5)
1    x is 5
2    return 2*5 is 10
4    print returned 10
5    call foo
1    x is 'now'
2    return 'now'*2 is 'nownow'
5    print returned nownow
```

12a. 110110: 54/2 = 27 R 0, 27/2 = 13 R 1, 13/2 = 6 R 1, 6/2 = 3 R 0, 3/2 = 1 R 1, 1/2 = 0 R 1
remainders backwards: 110110
b. 3CD92 11 1100 1101 1001 0010 group from the right!
  3 C D 9 2

13a $2*16^2 + 10*16 + 15 = 512 + 160 + 15 = 687$
  b. 844/16 = 52 R 12; 52/16 = 3 R 4; 3/16 = 0 R 3 Read remainders from right: 3 4 12; convert to hexadecimal digits: 34C. (If you do not like arithmetic with 16's, you could do binary conversions in the middle: part a: convert to binary, then decimal. Part b: convert to binary; then hexadecimal.)

14.
```
   1 111    carries
    101011
  + 100110
    1010001
```

15. [6, 2, 4, 9]
step by step

Line	nums	i	comment
4	[6]		execution starts at line 4 -- after the definitions
5			call foobar
1			oldVals is [1, 3, 8] and newVals is an alias for nums
2		1	i is first element of oldVals
3	[6, 2]		i+1 is 1+1 is 2, append to newVals (nums)
2		3	i is next element of oldVals
3	[6, 2, 4]		i+1 is 3+1 is 4, append to newVals (nums)
2		8	i is next amd last element of oldVals
3	[6, 2, 4, 9]		i+1 is 8+1 is 9, append to newVals (nums)
2		-	done with sequence and done with loop
6			print [6, 2, 4, 9] (with square braces and commas)

16. 3 7 # f(1) is 2*1+1 = 3; f(f(1)) is f(3) = 2*3+1=7

17.
```
8
12
2
```

line	x	comment
1	16	
2		16 >2 is True
3	8	16/2 is 8
4		8>3 and 8 < 7 is true and false is false
6		print 8
2		8 >2 is True
3	4	8/2 is 4
4		4>3 and 4 < 7 is true and true is true
5		4*3 is 12 -- printed
2		4 >2 is True
3	2	4/2 is 2
4		2>3 and 2 < 7 is false and true is false
6		print 2
2		2>2 false: skip loop

18.
```
aa bb cc
dd ee
ff
```

line	s	ch	comment
1	abc		first in list
2		a	first in character sequence 'abc'
3			print aa (but stay on same line)
2		b	next in character sequence 'abc'
3			print bb (but stay on same line)
2		c	last in character sequence 'abc'
3			print cc (but stay on same line)
2		-	done with character sequence 'abc'
4			on to new line, done with inner loop
1	de		next in list for outer loop
2		d	first in character sequence 'de'
3			print dd (but stay on same line)
2		e	next and last in character sequence 'abc'
3			print ee (but stay on same line)
2		-	done with character sequence 'de'
4			on to new line, done with inner loop
1	f		next in list for outer loop
2		f	first in character sequence 'f'
3			print ff (but stay on same line)
2		-	done with character sequence 'f'
4			on to new line, done with inner loop
1			done with list and outer loop

19.
```
def upper2(s):
    print(s.upper()*2)
```

20.
```
def upper2ret(s):
    return s.upper()*2
```

21.
```
def upper2(s):
    print(upper2ret(s))
```

22.
```
def printListUpper(words):
    for s in words:
        print(s.upper(), end=' ')
```

23.
```
def printListShortUpper(words, n):
    for s in words:
        if len(s) < n:
            print(s.upper(), end=' ')
```

24.
```
def newListUpper(words):
        up = []
    for s in words:
        up.append(s.upper())
    return up
```

25a
```
SELECT name, time FROM classes
  WHERE prof = "Harrington"
```

b.
```
SELECT name, prof, time FROM classes
  WHERE credits > 3
```