

CS 110 Midterm Examination

General instructions:

1. The exam is open book, open notes, closed computer. If nothing else, you will definitely be using the M68000 programmer's reference manual.
2. You may, but are not required to, use a simple hex calculator on the exam as mentioned via e-mail.
3. You will have 1 hour, 15 minutes to complete the exam (a standard class period). I tried to make the exam a 60 minute exam (2 points a minute) with 15 minutes extra, but it may be a bit more difficult than that. Don't spend too much time on any one question.

Problem	Points	Score
1	35	
2	30	
3	30	
4	25	
Total	120	

Name: _____

ID Number: _____

Honor Code:

"In accordance with both the letter and the spirit of the Honor Code, I didn't cheat on this exam."

Signature: _____

Problem 1 – Some Assembly (and Disassembly) Required (35 points)

Convert the following instructions from assembly to machine language. You should **give answers in hexadecimal**, though if you want to show us your binary, we can give you partial credit if you make a mistake...

Assembly	Machine Code (in HEX)
1. <code>MOVE.L A6, -(A7)</code>	
2. <code>LSL.W #1, D0</code>	
3. <code>LEA -20(A5, D0.W), A3</code>	

Convert the following instructions from machine language to assembly.

Machine Code	Hint	Assembly
4. <code>0x2C4F</code>	MOVEA	
5. <code>0x514F</code>	SUBQ	
6. <code>0xB7F9 B7F9 B7F8</code>	CMPA	

Bonus (+1 point): Numbers 1, 4, and 5 taken together be replaced by a single instruction. Give it:

Problem 2 – Condition Codes and Branching (30 points)

Given the following initial conditions:

D0 = 0x0123 0005
 D1 = 0xFEED 876A
 A0 = 0X0001 0000

a) Fill in the following table. Show the condition codes and values after the following instructions (each condition code should be either “0” or “1”) and tell whether the given branch would be taken (“y” or “n”). If a register didn’t change, you are free to put a dash as the answer:

IMPORTANT: Each instruction depends on the previous instructions.

Instruction	D0	D1	A0	N	Z	V	C	BLS	BLE
CMP.L D0, D1									
SUB.L D0, D1									
CMP.B D1, D0									
MOVEA.L D1, A0									

b) Reread the instructions for part **a)**. Make sure you realized that each instruction depends on the previous instruction.

Problem 3 – Tables (30 points)

Convert the following C code into assembly language. You should use the method discussed in lecture to efficiently code a switch statement (use a case table). You may assume:

- x starts in D0, though you are free to clobber D0
- y should end up in D1
- x and y are both 16-bit ints.

```
switch (x) {  
    case 13:  
    case 16:  
        y = 2 * x;  
        break;  
    case 12:  
    case 14:  
    case 15:  
        y = -1;  
        break;  
    default:  
        y = x;  
}
```

a) Write the code for the cases:

b) Specify the table (use Metrowerks syntax):

c) Figure out where to jump to and jump there:

Problem 4 – Addressing Modes (25 points)

Given the following initial conditions, show the destination address (or register) and 32-bit value (in hex) **after the execution of these instructions**. Note: unlike problem #2, **every instruction is executed independently** with the following initial conditions:

0000 0000 to 0001 B894	????
0001 B896	0725
0001 B898	5303
0001 B89A	0195
0001 B89C	0000
0001 B89E to FFFF FFFE	DEAD
FFFF 0000 to FFFF FA1C	????
FFFF FA1E	00BF
FFFF FA20	0800
FFFF FA22	0000
FFFF FA24	0000
FFFF FA26 to FFFF FFFE	????

D0	9899 0004
D1	2300 FFFE
A1	10C6 F52C
A2	0001 B89A
A3	0001 C6CD
A4	0001 D2A4
A5	0001 B8B0
A6	0001 C6BA
A7	0001 C5EE
PC	0000 7000

```
#define      kPolMask          0x01
#define      LCD_PolReg        0xFFFFFA21
static      Byte gOldVal;      // located -(0x1A) off A5
...
pattern:    DC.W              0x0000, 0x0000, 0xAAAA, 0xAAAA, 0xFFFF
            DC.W              0x0000, 0x5555, 0x5555, 0xFFFF, 0xFFFF
```

	destination EA (or register) in hex.	32-bit hex value beginning at EA (after instruction).
MOVE.B LCD_PolReg, gOldVal		
ORI.B #kPolMask, LCD_PolReg		
MOVE.L D1, -(A7)		
MOVE.W pattern(D0.W), D1		

...**REMEMBER**: even though these instructions do make sense together, you should execute them all independently (all starting with the initial conditions given above).