

CS 110 Practice Midterm Examination

General instructions:

- The exam is open book, open notes, closed computer. If nothing else, you will definitely be using the M68000 programmer's reference manual.
- You may, but are not required to, use a simple hex calculator on the exam as mentioned in class.
- You will have 1 hour, 15 minutes to complete the exam (a standard class period). I tried to make the exam a 50 minute exam (2 points a minute) with 25 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	16	
2	16	
3	20	
4	28	
5	20	
Total	100	

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 – Hand Assembly (16 points)

Convert the following instructions from assembly to machine language. You should **give answers in hexadecimal**.

Assembly	Machine Code (in HEX)
MOVE.L (A0)+, D3	
EORI.L #-1, D3	
ADDQ.L #1, D3	
SUB.L D3, -4(A0)	

Bonus (+1 point): What single-word instruction will produce almost the same effect as executing the preceding series of instructions (the same thing except that A0 isn't modified)?

Problem 2 – Hand Disassembly (16 points)

Convert the following instructions from machine language to assembly.

Machine Code	Hint	Assembly
\$0000 0000	ORI	
\$3585 D803	MOVE	
\$BEEF AAAA	CMPPA	
\$4498	NEG	

Problem 3 – Condition Codes and Branching (20 points)

Given the following initial instructions:

```
CLR.L D0
CLR.L D1
MOVE.W #$65, D0
MOVE.W #$FC, D1
```

a) Give the values of D0 and D1 after the instructions execute (give a 32-bit value in hex):

D0: _____

D1: _____

b) Show the condition codes resulting after the following instructions (each should be either a “0” or a “1”) and tell whether the given branches would be taken (“y” or “n”):

Instruction	N	Z	V	C	BHI	BGT
CMP.L D0, D1						
CMP.W D0, D1						
CMP.B D0, D1						
CMP.B D1, D0						
ADD.B D0, D0						

Problem 4 – Performance and Tables (28 points)

The following piece of code sets up a table of the first 16 powers of three in order to try to speed up calculations in another part of the program (thus, the “0th” element in the table is $3^0 = 1$, and the “15th” element is $3^{15} = 14,348,907$).

a) Fill in the following:

- Fetch reads – the number of reads to fetch the instruction.
- Execute reads – the number of reads to load the operands of the instruction.
- Execute writes – the number of writes to store the result of the instruction.

			Fetch reads	Exec reads	Exec writes
ThreesTable	DS.L	15			
	MOVE.L	#0, D0	_____	_____	_____
	MOVE.L	#1, D1	_____	_____	_____
	LEA	ThreesTable(A5), A0	_____	_____	_____
Loop	MOVE.L	D1, 0(A0, D0)	_____	_____	_____
	ADD.L	#4, D0	_____	_____	_____
	MULU.L	#3, D1	_____	_____	_____
	CMP.L	#60, D0	_____	_____	_____
	BNE	Loop	_____	_____	_____
	RTS				

Note: memory reads and writes were covered last year before the midterm. They will not be covered on this year’s midterm, though you should be able to figure this stuff out...

b) Given that some value n ($0 \leq n \leq 15$) is stored in D0, write the code to access 3^n . In other words, if D0 contains 1, you want to retrieve the “1st” element in the table (which should be 3). If D0 contains 0, you want to retrieve the “0th” element. Place the result in D1. You need not worry about performance in this problem (though you shouldn’t need more than a few instructions).

c) The code given to initialize the table is quite inefficient both in terms of speed and in terms of space. There are multiple ways to optimize it (looking at it quickly, I see about five). Name two optimizations that could be applied to this code. The optimizations should be different in nature (don't just say you could apply one optimization to multiple instructions). You may not use more than the space given to describe your optimizations.

1.

2.

d) This code is quite silly because it goes through the work of computing values at runtime that we actually know at "assembly time." How could we create a similar table that needs no runtime initialization (you may not use this as one of your optimizations for part c)? You may not use more than a few lines for your answer.

Problem 5 – Addressing Modes (20 points)

Given the following initial conditions, show the destination address and 32-bit value (in hex) after the execution of these instructions.

Notes:

- There is an ACSII chart on page C-5 of the M68000 reference manual if you need it.
- “doh” (defined below) is $-\$20$ off of A5.
- **Every instruction is executed independently** with the following initial conditions:

0000 0000 to	????
0000 E100	
0000 E102	E11A
0000 E104	0000
0000 E106	E114
0000 E108	BABA
0000 E10A	600D
0000 E10C	F00D
0000 E10E	E10C
0000 E110	B000
0000 E112	AAAA
0000 E114	5000
0000 E116	BEEE
0000 E118	BAD1
0000 E11A to	DEAD
FFFF FFFF	

D0	0000 0004
D1	00FF 7000
A1	0000 7100
A2	0000 E104
A3	0000 E0F8
A4	0000 E11A
A5	0000 1000
PC	0000 2000

```
doh      DS.L  4
mnmnm   DC.B  "DONUTS"
woohoo   EQU   $12
cerealPort EQU $0000 E10C
```

	Dst EA in hex	32-bit value in hex beginning at dst EA.
MOVE.L mnmnm(PC), doh(A5)		
MOVE.W mnmnm(PC, D0.L), cerealPort		
MOVE.B -(A4), woohoo(A3)		
MOVE.W #woohoo, (A2)		
MOVE.L woohoo(A1, D1.W), woohoo		