

Jonathan W. Valvano First Name: _____ Last Name: _____
August 14, 1998, 7-10pm

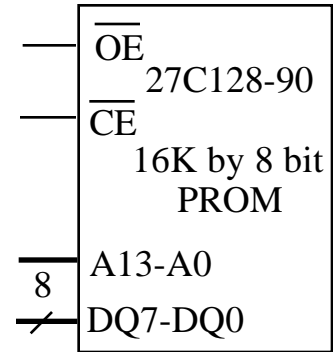
This is an open book, open notes exam. You may put answers on the backs of the pages, but please don't turn in any extra sheets. You have 3 hours, so please allocate your time accordingly.

(25) Question 1. Design a variable gain analog amplifier. The analog input is V_{in} , the 3 bit digital inputs (connected to a microcomputer output digital output) are B_2, B_1, B_0 and the analog output is V_{out} . Exactly one of the digital inputs will be one, and the gain should be

B_2, B_1, B_0	Gain (V_{out}/V_{in})
001	1
010	10
100	100

(25) **Question 2.** Interface a 27C128-90 16K by 8 bit PROM to the MC68HC812A4 address/data bus. The address will be \$C000 to \$FFFF. No other external devices exist. Assume the 6812 is running at 8 MHz in expanded narrow mode.

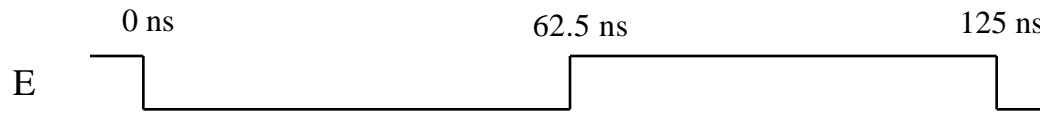
(5) **Part a)** Show digital circuit for the interface between the 6812 and the PROM. Please label TTL chip numbers but not pin numbers.



(10) **Part b)** Develop equations for read data available and read data required, and use them to determine the amount of cycle stretching required to interface this 90ns memory.

(5) **Part c)** Show the software ritual required to initialize this memory interface.

(5) **Part d)** Show the combined **read cycle** timing diagram. Assume a 10 ns gate delay through any digital gate. All signals are outputs except Data Required. Use arrows to signify causal relations.



R/W

A15-A0

$\overline{\text{CE}}$

$\overline{\text{OE}}$

Read Data Available

Read Data Required

(25) Question 3. When a debugger wishes to single step a program which exists in RAM, it can replace op codes one at a time with SWI (it will have to replace two op codes when single stepping a conditional branch.) This approach is not feasible when testing software stored in ROM or PROM. In this question you will use output compare interrupts to implement single step tracing. Your approach will work for programs in RAM or ROM.

Part a) Write a debugging function, called `trace`, which initializes an output compare interrupt then calls the `UserRoutine`. The first output compare interrupt should occur after exactly one instruction of the `UserRoutine` has been executed. You may assume the `UserRoutine` has no input/output parameters. You may also assume `UserRoutine` has no interrupts of its own and it does not disable interrupts. The goal is to execute exactly one microcomputer assembly instruction at a time, and print out all the register values including the stack pointer. I.e., execute one instruction, print, execute one instruction, print, execute one instruction, print, execute one instruction, print,... When the `UserRoutine` returns back to your function, you should shut off the output compare. You can start with the following syntax and add the output compare code. You may write your answer in assembly or C.

```
// trace UserRoutine
void trace(void (*UserRoutine)(void)){
// add stuff here to initialize output compare
(*UserRoutine)();
// add stuff here to stop output compare
}
```

You are given gadfly SCI functions which you will call, but do not need to write:

```
void InitSCI(void); // Initialize 38400 bits/sec
void OutChar(char); // Output a character, gadfly
void OutString(char *); // Output a string, null terminated
void OutUHex8(unsigned char); // Outputs an Unsigned 8 bit Hexadecimal number
void OutUHex(unsigned int); // Outputs an Unsigned 16 bit Hexadecimal number
```

Part b) Write the output compare interrupt handler. First determine the values of the registers at the time of the interrupt. In a real debugger we would process the keyboard input and interact with the user, but in this simple solution the keyboard input is ignored, and you simply print out the register values and continue. Before returning from interrupt you should set up the output compare so that the microcomputer will execute exactly one more instruction of `UserRoutine` before another output compare interrupt is generated. If you setup the next output compare to be requested too early, then the `UserRoutine` will never be executed. Similarly if you setup the next output compare to be too late, then more than one instruction of `UserRoutine` will be executed.

(25) **Question 4.** The objective of this problem is to develop the fixed point equations which implement the a PID controller. You are to implement the following control system

$X(t)$ is the state variable (volts)

X^* is the desired state (volts)

$e(t) = X^* - X(t)$ is the error (volts)

$V(t)$ is the actuator command (volts)

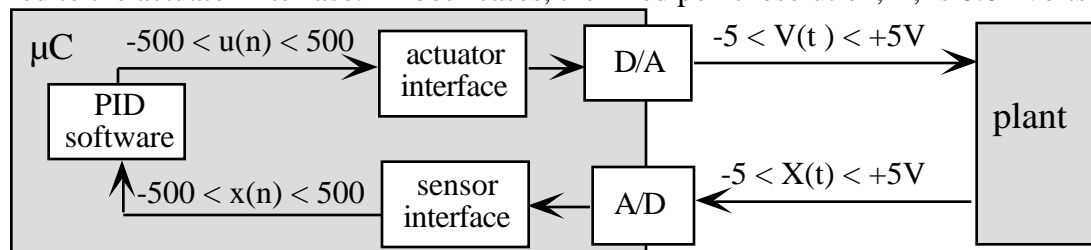
$$V(t) = K_p e(t) + K_I \int_0^t e(\tau) d\tau + K_D \frac{de(t)}{dt}$$

where K_p is 0.1 (dimensionless)

K_I is 10 (1/sec)

K_D is 0.0001 (sec)

The state estimator and actuator output hardware/software interfaces are given. Your PID software is given a signed 16 bit decimal fixed point input, $x(n)$, which represents the current state variable. Similarly, your PID software will calculate a signed 16 bit decimal fixed point output, $u(n)$, which will be fed to the actuator interface. In both cases, the fixed point resolution, Δ , is 0.01 volts.



Assuming the digital controller is executed every 1ms (1000Hz), show the control equation to be executed in the periodic interrupt handler. In this process you will convert from floating to fixed point numbers, and convert from continuous to discrete time. No software is required just show the equations. Explain how you would deal with overflow/underflow. You should assume some noise exists on the sensor input.

