Department of Electrical and Computer Engineering
The University of Texas at Austin

EE 460N Spring 2013
Y. N. Patt, Instructor
Faruk Guvenilir, Sumedha Bhangale, Stephen Pruett, TAs
Exam 2
April 17, 2013

Name:____

Problem 1 (20 points):_______

Problem 2 (20 points):_______

Problem 3 (20 points):_______

Problem 4 (20 points):_______

Problem 5 (20 points):_______

Total (100 points):______________

Note: Please be sure that your answers to all questions (and all supporting work that is required) are contained in the space provided.

Note: Please be sure your name is recorded on each sheet of the exam.

Please sign the following. I have not given nor received any unauthorized help on this exam.

Signature:____

GOOD LUCK!

Name:__________________________________

Problem 1 (20 points)

Part a (5 points): A processor that supports 8 interrupt priority levels (0 is lowest priority, 7 is highest priority) is executing a process at priority level 3 when a page fault occurs. The page fault service routine would normally execute at priority level

Explain your choice of priority level in 20 words or fewer.

Part b (5 points): VLIW instructions have two characteristics that hinder performance. DAE got rid of one of them. Describe the characeristic DAE got rid of in fewer than ten words:

Give an example of a very small snippet of code that illustrates how DAE improves performance if the VLIW characteristic is not present.

Name:___________________________________

Problem 1 continued

Part c (5 points): Two structures are added to the data path to support out of order execution and one structure is added to support in order retirement. Identify them.

To support out-of-order execution: ,.

To support in-order retirement: .

Part d (5 points): Computers that implement virtual memory generally include a structure wherein each entry in the structure has two parts, a page number and a descriptor. What is the structure called?

Answer:

The processor uses the descriptor to perform several functions, some more important than others. Describe in 20 words or fewer two essential functions that the processor uses this descriptor to perform.

Name:___________________________________

Problem 2 (20 points)

Part a (10 points): We wish to represent the number 33/128 exactly in a radix-2 floating point format that has all major characteristics consistent with the IEEE Floating Point standard. The format must use an excess-17 code. What is the fewest number of bits we can use for this format to represent the number 33/128 exactly? Show your work.

Answer:

How is the number 33/128 represented in this format?

Answer:

Part b (10 points): A physically indexed, physically tagged 64 KB write through cache has been designed to support a 4 MB physical memory. The cache has the following characteristics: 16 byte line size, 2-way set associative, LRU replacement. How many bits of storage are required to implement the tag store of this cache? Please show your work.

Answer:

Name:____________________________________

Problem 3 (20 points)
We have talked about vector processing and we have talked about predication. It turns out that what we learned about predication can make our vector processor even more powerful, as shown in this problem.

Recall that predication lets us remove a conditional branch from a program by first evaluating the condition (which we call a predicate – it has a Boolean value, either 0 or 1), and then storing the results of the instructions either along the true path or along the false path of the branch, depending on the value of that predicate. In a vector processor, executing a single instruction stream (a "for loop," for example) SIMD on multiple data sets (all the iterations of the loop body), is accomplished by each iteration operating on the same *i*th component of all the one dimensional arrays. If the loop body contains a conditional branch, we first use a vector instruction to evaluate that condition (predicate) for all iterations, and then conditionally execute the correct path depending on the value of the predicate for each iteration of the loop body.

We accomplish this with two mechanisms:

1. A Vector Mask Register, called VMASK which has the same number of components as the other vector registers, which is specified by VLENGTH. For example, we can execute a vector instruction

```
        VMASK = V0 > V1
```

by writing the Boolean value VMASK[i] = 1 or 0, depending on whether or not V0[i] $>$ V1[i].

2. Then we execute a predicated version of the normal vector instructions, for example VADDp instead of VADD, where

VADD V0,V1,V2 becomes VADDp V0,V1,V2. That is, store the sum of V1[i] + V2[i] into V0[i] if VMASK[i] = 1. If VMASK[i] =0, do not write the result.

Our machine has the following vector registers and scalar registers:

```
        Vector Registers (VR): V0 to V7, VMASK
        Scalar Registers (SR): R0 to R7, VLENGTH, VSTRIDE
```

Our machine also has the following instructions, with maximum vector length of 64:

Instructions:	Cycles	Description:
LDIMM SRx, #C	1	SRx = #C
NOT SRx, SRy	2	SRx = $\sim$SRy
ADD SRx, SRy, SRz	4	SRx = SRy + SRz
VLD VRx, A	11	while (i < VLENGTH) { VRx[i] = M[A+(i$*$VSTRIDE)]; i=i+1;}
VST VRx, A	11	while (i < VLENGTH) { M[A+(i$*$VSTRIDE)] = VRx[i]; i=i+1;}
VMOV VRx, VRy	1	while (i < VLENGTH) { VRx[i] = VRy[i]; i=i+1;}
VNOT VRx, VRy	2	while (i < VLENGTH) { VRx[i] = $\sim$VRy[i]; i=i+1;}
VADD VRx, VRy, VRz	4	while (i < VLENGTH) { VRx[i] = VRy[i] + VRz[i]; i=i+1;}
VCMPEQ VRx, VRy	3	while (i < VLENGTH) { VMASK[i] = VRx[i] == VRy[i]; i=i+1;}
VCMPGT VRx, VRy	3	while (i < VLENGTH) { VMASK[i] = VRx[i] > VRy[i]; i=i+1;}
VCMPLT VRx, VRy	3	while (i < VLENGTH) { VMASK[i] = VRx[i] < VRy[i]; i=i+1;}

In addition, every NON-MEMORY vector instruction has a predicated counterpart, designated by appending p to the non-predicated opcode. For example, as explained above, VADD can be predicated on the basis of the VMASK register by using the opcode VADDp.

Name:____________________________________

Problem 3 continued

Now (finally!) we are ready to use our new knowledge.

Part a (12 points): Compile the following C code fragment into its corresponding vector assemly language code, using the instructions from the list above. Optimize your code so that it completes in as few cycles as possible. You may not need all the space provided for your answer.

```
for (int i = 0; i < 60; i=i+3) {
    if (A[i] > B[i]) {
        C[i] = A[i];
    } else {
        C[i] = B[i];
    }
}
```

```
LDIMM VLENGTH,
LDIMM VSTRIDE,
```

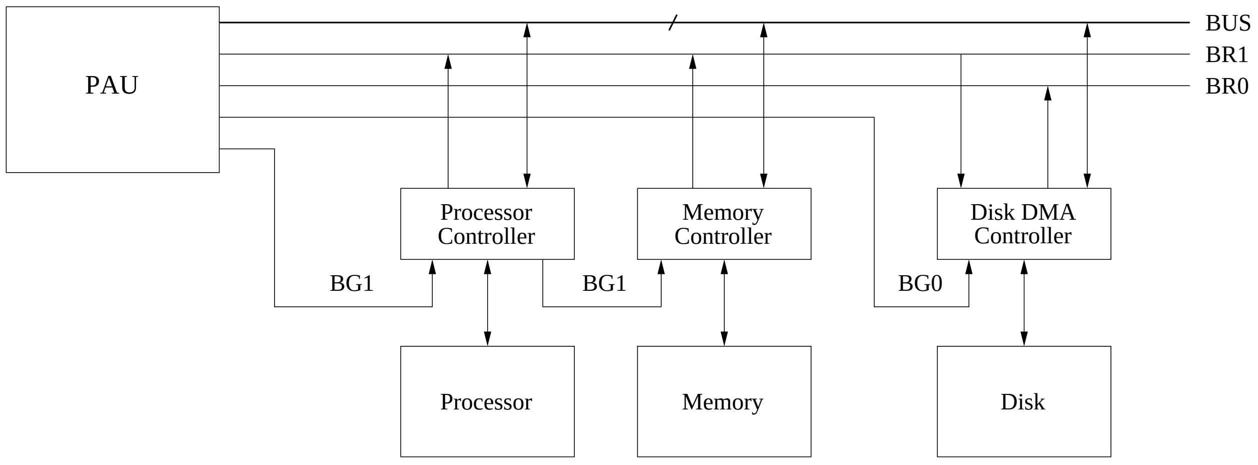
Part b (8 points): How many cycles does it take to execute this code. Assume 1 port to memory, memory is 16 way interleaved, vector chaining is supported, there is one pipelined functional unit for each type of operate instruction. Note that the time it takes to execute a predicated vector instruction is independent of the number of 0s in VMASK. That is the predicated vector instruction has to still execute each component, even though some results do not get stored.

Cycles.

Name:__________________________________

Problem 4 (20 points)

Consider the asynchronous bus discussed in class, with central arbitration, as shown below. The bus has a PAU with a processor, a memory, and a disk that can all request the bus with their respective device controllers. Data and Address lines are multiplexed. The processor and memory are at request level 1 (highest priority), and the disk at request level 0. The disk controller supports data transfers directly to memory (i.e., Direct Memory Access, or DMA), which may require transmitting several bus widths worth of data to memory in a single bus transaction.



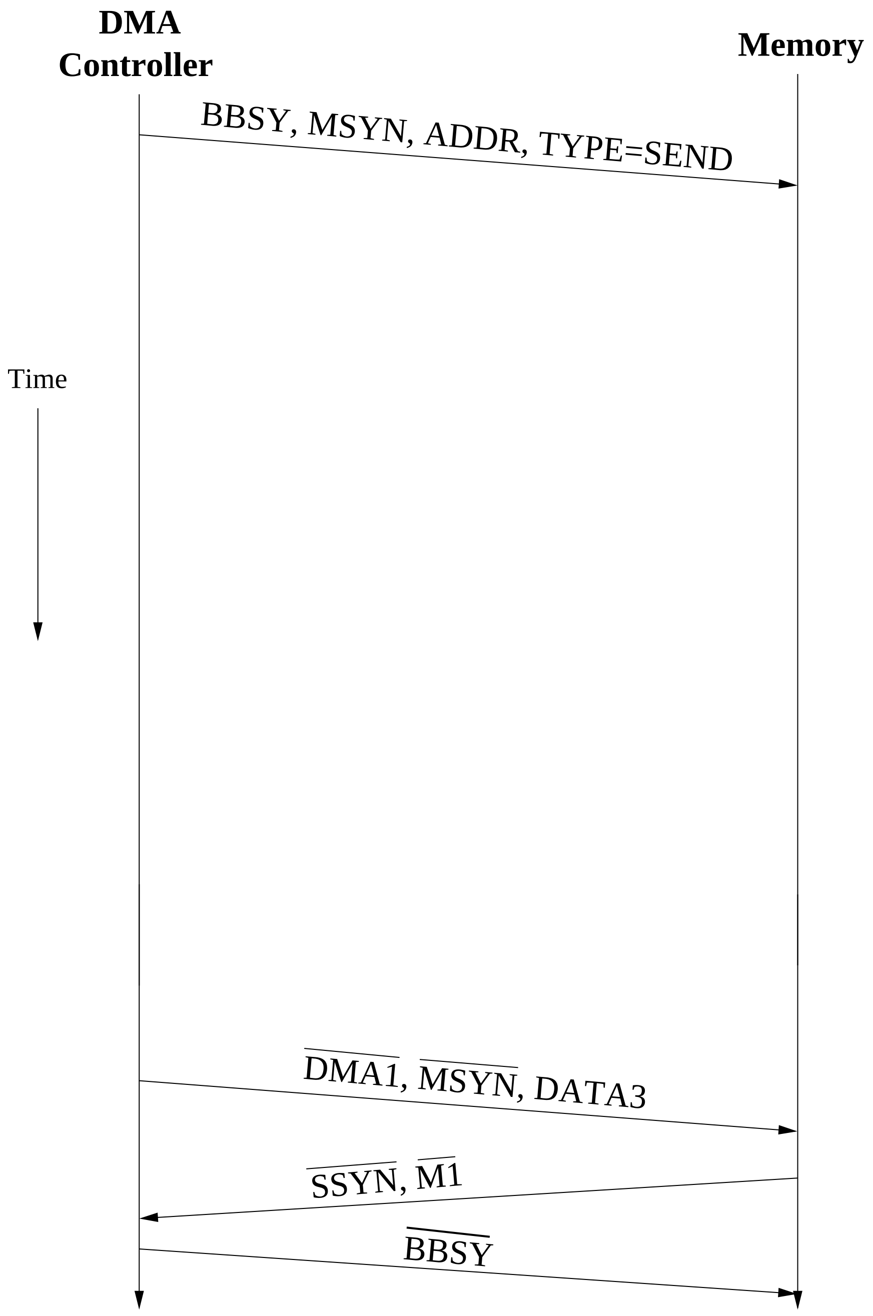
Note: not all bus signals are shown.

Name:__________________________________

Problem 4 continued

Part a (10 points): The diagram below shows part of the transaction required for the disk to send 4 chunks of data (DATA0, DATA1, DATA2, DATA3) uninterrupted to memory. We have added two additional bus control signals to make life easier: DMA1 for the disk controller and M1 for the memory controller.

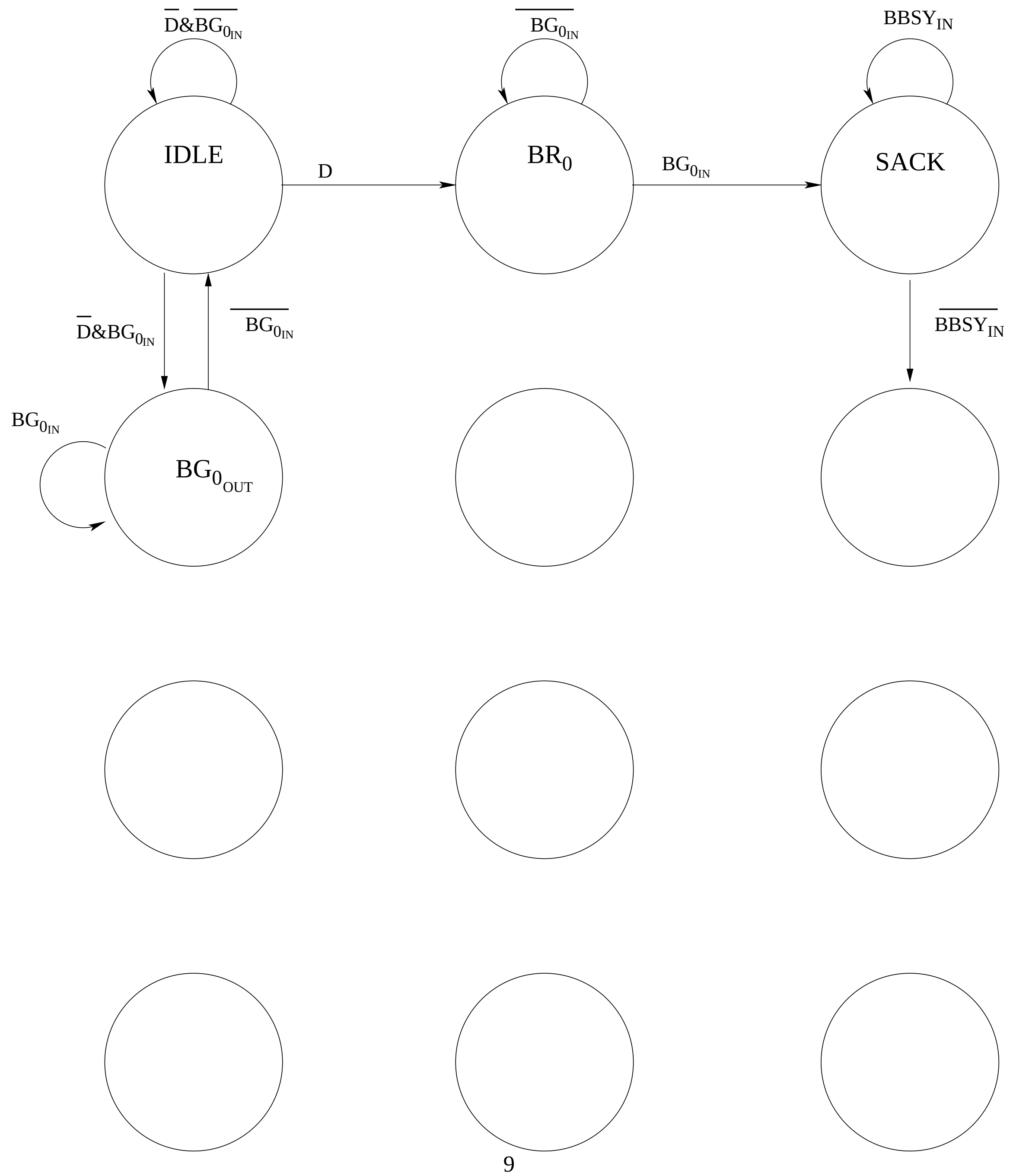
Your job is to complete the diagram, completing the send in as short a time as possible. Only the starting address (ADDR) of the first data chunk needs to be sent to the memory controller.


DMA Controller
Memory
BBSY, MSYN, ADDR, TYPE=SEND
Time
$\overline{DMA1}$, $\overline{MSYN}$, DATA3
$\overline{SSYN}$, $\overline{M1}$
$\overline{BBSY}$

Name:____________________________________

Problem 4 continued
Part b (10 points): Sometimes, the DMA controller may want to send much more than 4 data chunks to memory. Because this may take a significant amount of time, the disk DMA controller will relinquish control of the bus if a device at a higher level requests the bus, and complete the transaction in a subsequent bus cycle. Your job: Complete the transaction part of the disk DMA controller state machine, with support for sending any number of data chunks to memory and support for interrupting the transaction. Once the controller detects BR1, the DMA controller can finish at most its current send plus 1 more data chunk before relinquishing control of the bus.

To help facilitate DMA transactions, we can assume the DMA controller has auxiliary hardware to take care of certain internal bookkeeping information for you. Your state machine does not have to control this hardware. The DMA controller will maintain an internal register, ADDR, to keep track of the address of the next data chunk that needs to be sent to memory. An internal register, DATA, will contain the current data chunk that needs to be sent to memory. An internal counter register will keep track of how many data chunks remain to be sent. An internal signal, FINAL, will be asserted as soon as exactly one data chunk remains to be sent to memory. You may not need all the states provided.



Name:____________________________________

Problem 5 (20 points)

A cache for a microprocessor has been partially specified, as follows: write-back, LRU replacement, 4-way set associative, no virtual memory, byte addressable. Your job is to specify the line size, number of sets, and cache size, such that the execution of the following silly piece of code will result in a cache hit ratio of exactly 31/32, and the size of the cache will be as small as possible.

```
for ( j = 0; j < 4; j = j + 1 )
    for ( i = 0; i < 1024; i = i + 1 )
             C[i] = C[i] + A[i] * B[i];
```

A, B, and C are non-overlapping arrays of bytes. Assume the inner loop requires four accesses to memory in the following sequence: a load from the C array, a load from the A array, a load from the B array, and a store to the C array. Assume A[0], B[0], and C[0] are all stored in the first byte of their respective cache lines, and that the three lines all map to the same set of the n-way set associative cache. All other data values produced by this code are stored in registers.

You can assume that prior to execution of the above code, **the cache contains all of the C array** and **none** of the elements of A or B.

Part a (15 points): Specify the parameters named below for the smallest cache that will give a cache hit ratio of exactly 31/32 on the above code.

Cache size (Data only, not tag store):

Number of sets:

Cache block size:

Part b (5 points): Why is the above code silly? What simple change can be made to the code above to improve its performance, while guaranteeing the same final values are stored in C, regardless of the initial values stored in C? Assume the cache's initial state is still as described in part a.

What will the hit ratio be with this change?