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

EE 360N, Fall 2005
Yale Patt, Instructor
Aater Suleman, Linda Bigelow, Jose Joao, Veynu Narasiman, TAs
Final Exam, December 16, 2005

Name (**1 point**):___SOLUTION____

Problem 1 (20 points):_______

Problem 2 (10 points):_______

Problem 3 (9 points):_______

Problem 4 (20 points):_______

Problem 5 (25 points):_______

Problem 6 (20 points):_______

Problem 7 (25 points):_______

Total (130 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.

GOOD LUCK!

Name:____________________________________

Problem 1 (20 points)

Part a (5 points): Amdahl's Law states that the best Speed-up you can ever achieve with a system containing many processors is limited by the part of the program that can't be parallelized

Part b (5 points): Recall that the VAX System Page Table was in physical memory. That is, the SBR was a physical address. The designers could have put the VAX System Page Table in System Virtual Space, and an additional data structure in physical memory to keep track of where the pages that make up the System Page Table actually reside. Recall that the VAX had 1 GB of system virtual space, pages were 512 bytes, and each PTE is 32 bits. Assuming System space is completely mapped, how much storage would that additional data structure require?

Answer:

$1GB = 2^{30} bytes$

$\frac{2^{30}}{2^9} = 2^{21}\ pages\ in\ System\ Space$

$2^{21} PTEs * (\frac{4bytes}{PTE}) = 2^{23} bytes\ (size\ of\ Sys\ Pg\ Table)$

$\frac{2^{23}}{2^9} = 2^{14} pages\ (size\ of\ Sys\ Pg\ Table\ in\ pages)$

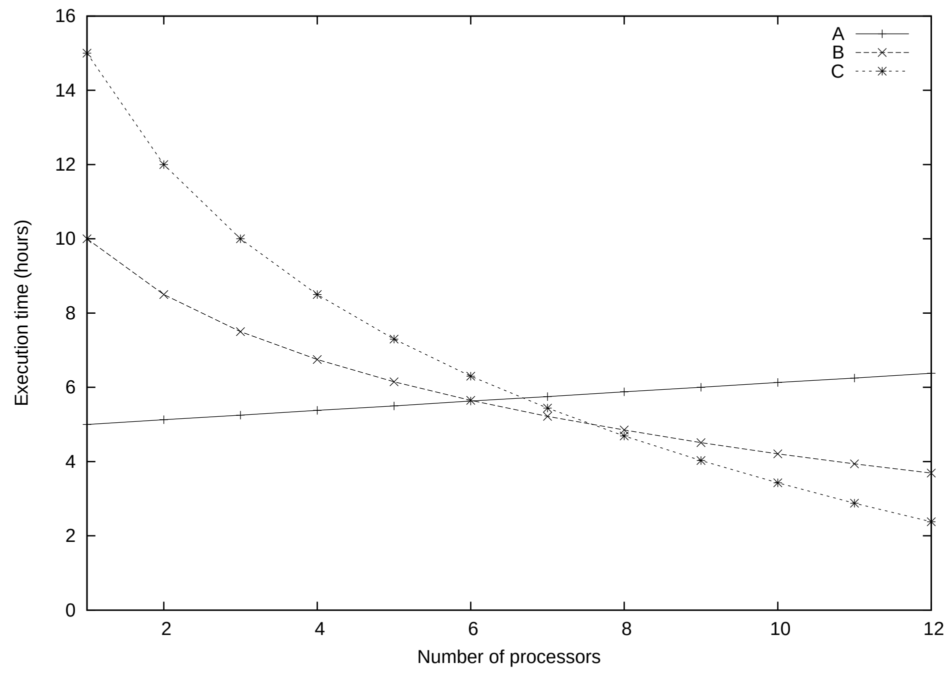
$2^{14} PTEs * (\frac{4bytes}{PTE}) = 2^{16} bytes = 64kB$

ANSWER: 64 kB

Name:________________________

Problem 1 continued

Part c (5 points): Algorithms A, B, and C all solve the same problem. The curves show the execution time of each algorithm as a function of the number of processors that are available to the algorithm. Which algorithm should we use to get maximum speed up if we have 9 processors available to us? The execution times for 9 processors are 6 hours for algorithm A, 4.5 hours for algorithm B and 4 hours for algorithm C. What is the Speed-up for this case?



Which algorithm $\boxed{C}$ Speed-Up $\boxed{5/4}$

Part d (5 points): Two paradigms for fetching and decoding multiple instructions each cycle are

$\boxed{\text{superscalar}}$ and $\boxed{\text{VLIW}}$.

In the first case, the number of instructions one can fetch and decode each cycle is determined at (when)

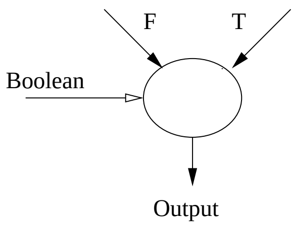
$\boxed{\text{runtime}}$ by $\boxed{\text{the instructions in the I-stream}}$.

In the second case, the number is specified by the ISA and therefore it is determined at the factory when the computer is defined.

Name:__________________________________

Problem 2 (10 points)

This problem involves evaluating a data flow graph that contains a data flow node that we have not discussed before: the mux node. It is illustrated below.



The mux node has three inputs and one output. Two of the inputs are labeled F and T, respectively. The third input is a Boolean (True/False). When all three inputs are ready, the mux node fires. The result is to pass either the T input token or the F input token to the output, depending on the value of the Boolean input.

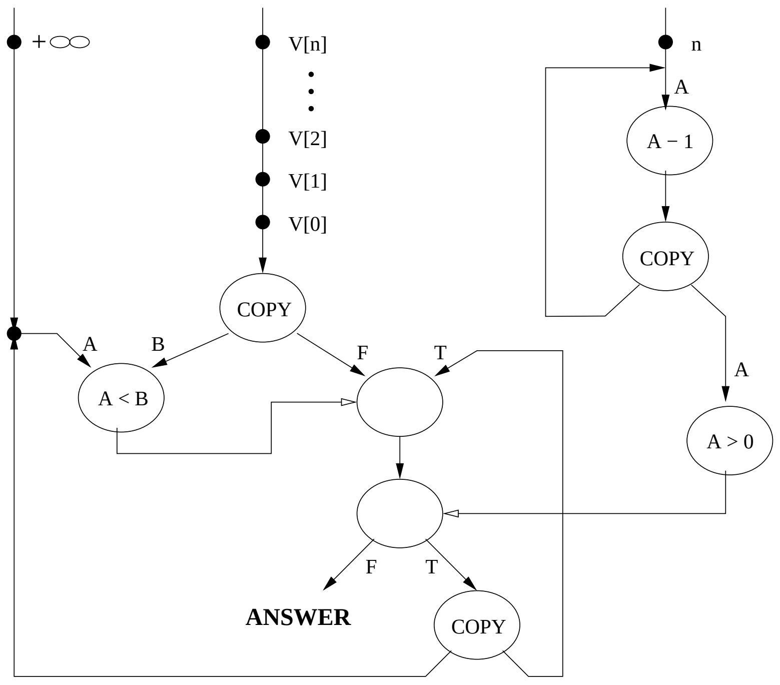
Part a (1 point): If the Boolean is False, and tokens 17 and 5 are on the F and T input lines, respectively, the node fires. What is the output?

$\boxed{17}$

Part b (9 points): The data flow graph shown below has three external inputs and one external output. Note the left-most input contains one value: $+\infty$. The middle input contains, in a queue, the values of an n-element vector. That is, V[0], V[1], V[2], ..., V[n]. The right-most input contains the value n, the length of the vector. When processing completes, the result is output via the output line labeled "Answer."

What does the data flow graph compute (in fewer than ten words)?

$\boxed{\text{The minimum value in the vector}}$



Name:____________________________

Problem 3 (9 points):

Your job in this problem is to analyze the performance impact that **Data Forwarding**, **Multiple Functional Units**, and **Out-of-Order Execution** would have on a particular instruction sequence. In the box next to each instruction sequence, state which combination of these three features (you can choose more than one) would lead to the best performance for that particular instruction sequence. Only include a feature if you can justify that including it will improve performance for that sequence. Explain your answers.

NOTE: Assume that in the baseline system, the Adder and the Multiplier are **NOT PIPELINED**.

Part a (3 points):

```
ADD R3, R2, R1
MUL R5, R4, R3
ADD R7, R6, R5
MUL R9, R8, R7
ADD R11, R10, R9
MUL R13, R12, R11
```

Use **data forwarding**. This sequence is filled with flow dependencies, so forwarding the data would definitely improve performance.

Part b (3 points):

```
ADD R3, R2, R1
ADD R6, R5, R4
ADD R9, R8, R7
MUL R12, R11, R10
MUL R15, R14, R13
MUL R18, R17, R16
```

Use **multiple functional units**. The ADDs and MULs could be executed in parallel, which would improve performance.

Part c (3 points):

```
MUL R3, R2, R1
ADD R5, R4, R3
MUL R8, R7, R6
ADD R11, R10, R9
MUL R14, R13, R12
ADD R17, R16, R15
```

Use **out-of-order execution** and **multiple functional units** so that the third through sixth instructions could execute while the second instruction is put aside due to a flow dependency. **Data forwarding** would also help a little since there is one flow dependency.

Name:__________________________

Problem 4 (20 points):

A computer system has a VA space of 32 bits, page size of 4KB, and PA address space of 16 MB. There is no restriction on the O/S with respect to mapping virtual pages to physical frames, except that shared data is allocated only within the common system space.

We want to design a 4-way set-associative, write-back cache with Victim/Next_Victim pseudo LRU replacement. Cache line size is to be 32 bytes. Since we wish to access the TLB and Tag Store concurrently, the index bits of the Tag store access must come from the VA.

Part a (7 points): If the cache is to be 8 KB,

$8kB = 2^{13} bytes;\ 1\ set: \frac{32 bytes}{block} * 4 blocks = 128 \frac{bytes}{set} = 2^7 \frac{bytes}{set};\ \frac{2^{13} bytes}{2^7 \frac{bytes}{set}} = 2^6 sets\ --> \ 6\ index\ bits$

```
    <--- 12 ---><--- 12 --->
    ------------------------
PA: |    PFN   |  offset   |
    ------------------------
    <--- 13 ----><-6 -><-5->
        tag      index  BiB
```

$Tag\ store\ entry:\ 13\ (tag)\ +\ 1\ (valid)\ +\ 1 (dirty)\ +\ 2\ (V/NV) = 17\ bits$
$Tag\ store:\ 17 \frac{bits}{block} * 4 \frac{blocks}{set} * 2^6 sets = 17 * 2^8 bits$

Compute the number of index bits: $\boxed{6}$

Identify the bits that comprise the tag: $\boxed{PA[23:11]}$

Compute the size of the tag store in bits: $\boxed{17 * 2^8}$

Part b (8 points): If the cache size is to be 128 KB,

$128kB = 2^{17} bytes;\ \frac{2^{17} bytes}{2^7 \frac{bytes}{set}} = 2^{10} sets\ --> \ 10\ index\ bits$

```
    <--- 12 ---><--- 12 --->
    ------------------------
PA: |    PFN   |  offset   |
    ------------------------
             <-- 10 --><-5->
               index    BiB
    <--- 12 --->
        tag
```

$Tag\ store\ entry:\ 12\ (tag)\ +\ 1\ (valid)\ +\ 1 (dirty)\ +\ 2\ (V/NV) = 16\ bits$
$Tag\ store:\ 16 \frac{bits}{block} * 4 \frac{blocks}{set} * 2^{10} sets = 2^{16} bits$

Compute the number of index bits: $\boxed{10}$

Identify the bits that comprise the tag: $\boxed{PA[23:12]}$

Compute the size of the tag store in bits: $\boxed{2^{16}}$

Name:__________________________

Problem 4 continued

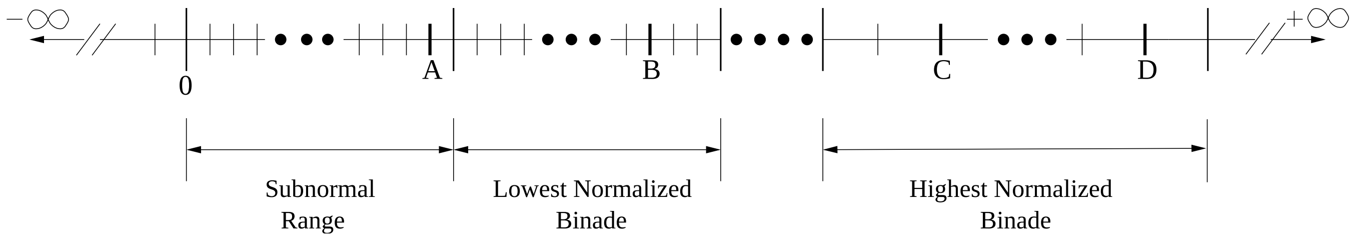
Part c (5 points): In either part a or part b, is it possible to have the physical addresses in two different places in the cache? Explain.

No. Since shared data is allocated within the common system space, this would not be possible.

Name:______________________________________

Problem 5 (25 points)

The real number line for an 11-bit floating point representation is shown below. Four values A, B, C, and D have been identified. Assume all the floating point representations are based on the IEEE floating point standard.



Part a (6 points): If the value A is 63/1024, how many bits are used for sign, exponent, and fraction in this format? **Show your work.**

Answer:
$\frac{63}{1024} = \frac{32+16+8+4+2+1}{1024} = 111111 * 2^{-10}\ (in\ binary)$

Since it is the largest subnormal number, this is: $0.111111 * 2^{-4}$.

Further, since it is the largest subnormal number, all the 1s must be represented, and there can't be any 0s after it. Thus, 6 bits for fraction. Since there is always 1 bit for sign, that leaves 4 bits for exponent.

Sign: $\boxed{1}$ Exponent: $\boxed{4}$ Fraction: $\boxed{6}$

Part b (3 points): What is the excess (BIAS) for the excess code:

Answer:
Since A is subnormal and has an exponent of -4, -4 must be the smallest possible exponent. This means that when the exponent field contains 0001, the actual exponent is -4.

$Exponent\ field\ -\ bias\ =\ actual\ exponent$
$1\ -\ bias\ =\ -4$
$bias\ =\ 5$

$\boxed{5}$

Name:____________________________________

Problem 5 continued

Part c (6 points): What are the **values** B, C, and D?

Note: Values B, C, and D may NOT be specified in the following format: $\boxed{\text{sign} \mid \text{exp} \mid \text{fraction}}$.

Answer:
D is the largest normalized number, which means the fraction bits will all be 1, and the exponent field contains 1110 (in binary). This corresponds to an actual exponent of 9. So, D is $1.111111 * 2^9$.

Since C is in the same binade as D, C will have the same exponent as D (9). The lowest number in that binade has the fraction bits all set to 0 (i.e. $1.000000 * 2^9$*). Since C is two ULPs larger, C is* $1.000010 * 2^9$.

The lowest normalized binade has an exponent of -4. Since B is in the same binade, it has the same exponent (-4). The largest number in that binade has all 1s for the fraction bits (i.e. $1.111111 * 2^{-4}$*). Since B is two ULPs smaller than the largest number in that binade, B is* $1.111101 * 2^{-4}$.

B: $\boxed{1.111101 * 2^{-4}}$ C: $\boxed{1.000010 * 2^9}$ D: $\boxed{1.111111 * 2^9}$

Part d (10 points): IEEE Floating Point arithmetic specifies five different types of exceptions. That is, an instruction OPCODE X,Y,Z has the potential to cause any one of the five exceptions.

In the table below, identify the five distinct exception types.

For each exception type, specify an OPCODE and the values of the two source operands Y,Z that would cause that exception. Use the format derived in parts a and b.

Note: Source operands Y and Z may NOT be specified in the following format: $\boxed{\text{sign} \mid \text{exp} \mid \text{fraction}}$.

Exception Type	OPCODE	Y	Z
underflow	DIVIDE	$0.000001 * 2^{-4}$	$1.111111 * 2^9$
overflow	MULTIPLY	$1.111111 * 2^9$	$1.111111 * 2^9$
inexact	ADD	$1.111111 * 2^9$	$0.000001 * 2^{-4}$
divide by 0	DIVIDE	$1.111111 * 2^9$	0
invalid	DIVIDE	0	0

Name:____________________________________

Problem 6 (20 points)

A processor has an 8-bit physical address space and a physically addressed cache. Memory is byte addressable. The cache uses perfect LRU replacement.

The processor supplies the following sequence of addresses to the cache. The cache is initially empty. The hit/miss outcome of each access is shown.

Address	Outcome
0	Miss
2	Hit
4	Miss
128	Miss
0	Hit
128	Hit
64	Miss
4	Hit
0	Miss
32	Miss
64	Hit

Your job: Determine the block size, associativity, and size of the cache. Note: It is not necessary to give an explanation for every step, but you should show sufficient work for us to know that you know what you are doing.

Answer:
From the first 3 references, the **block size** *must be* **4***. To get the associativity, notice that the 5th reference (to address 0) is a hit, but the 9th reference (again to address 0) is a miss. This means the data for address 0 was kicked out of the cache somewhere in between. If the associativity were 4 or more, the data for address 0 would not have been kicked out; therefore, we know the associativity is either 2 or 1. If the associativity were 1, the cache size would have to be greater than 128 bytes since the 5th and 6th references (to 0 and 128, respectively) are both hits. However, if the cache size were greater than 128 bytes, the data for address 0 would not have been kicked out, which means the* **associativity** *must be* **2***. To get the cache size, notice that addresses 0, 64, and 128 must map to the same set (index bits must be the same), but address 32 should map to a different set. This is only possible if the number of index bits is 4. This implies a* **cache size** *of* **128 bytes***:*

$$\frac{4bytes}{block} * \frac{2blocks}{row} * 2^4 rows = 128bytes$$

Block size: $\boxed{4}$ bytes

Associativity: $\boxed{2}$ -way

Size: $\boxed{128}$ bytes

Name:____________________________________

Problem 7 (25 points)

Recall the LmmVC-3 (Little mickey mouse Vector Computer 3) that you implemented in Problem Set 5, and modfied on Exam 2 to be able to interrupt VADD instructions. For your convenience, the data path that you constructed in Problem set 5 is shown on page 15. Today's task is to extend the capability of the LmmVC-3 to be able to interrupt vector loads (VLD).

Before we get into today's problem, we reproduce below (for those who want it) the LmmVC-3 specification. Note: The remainder of this page contains the **EXACT** same specification that was given in Problem Set 5 and on Exam 2.

Little Computer Inc. is now planning to build a new computer that is more suited for scientific applications. LC-3b can be modified for such applications by replacing the data type Byte with Vector. The new computer will be called LmmVC-3 (Little "mickey mouse" Vector Computer 3). LmmVC-3 ISA will support all the scalar operations that LC-3b currently supports except the LDB and STB will be replaced with VLD and VST respectively. Our data path will need to support the following new instructions:

	15 14 13 12	11 10 9	8 7 6	5	4	3	2 1 0
MOVI Vstride, amount6	1011	000	000	amount6			
MOVI Vlength, amount6	1011	001	000	amount6			
VLD VDR, BaseR, offset6	0010	VDR	BaseR	offset6			
VADD VDR, VSR1, VSR2	1010	VDR	VSR1	0	1	0	VSR2
VADD VDR, VSR1, SR2	1010	VDR	VSR1	0	0	0	SR2
VST VSR, BaseR, offset6	0011	VSR	BaseR	offset6			

Note: VDR = Vector Destination Register, VSR = Vector Source Register

MOVI
If IR[11:9] = 000, MOVI moves the unsigned quantity amount6 to Vector Stride Register (Vstride).
If IR[11:9] = 001, MOVI moves the unsigned quantity amount6 to Vector Length Register (Vlength).
This instruction has already been implemented for you.

VLD
VLD loads a vector of length Vlength from memory into VDR. VLD uses the opcode previously used by LDB. The starting address of the vector is computed by adding the LSHF1(SEXT(offset6)) to BaseR. Subsequent addresses are obtained by adding LSHF1(ZEXT(Vstride)) to the address of the preceding vector element.

VST
VST writes the contents of VSR into memory. VST uses the opcode previously used by STB. Address calculation is done in the same way as for VLD.

VADD
If IR[4] is a 1, VADD adds two vector registers (VSR1 and VSR2) and stores the result in VDR.
If IR[4] is a 0, VADD adds a scalar register (SR2) to every element of VSR and stores the result in VDR.

VLD, VST, and VADD do not modify the content of Vstride and Vlength registers.

Name:__________________________

Problem 7 continued

In Exam 2 we noted that interrupt capability is useful while processing vector instructions since vector instructions take many cycles to complete. That is, we wish to be able to interrupt the vector instruction at any time, service the interrupt, and then return to the point in the vector instruction where we left off. We do NOT want to repeat the execution of any part of the vector instruction that we had already carried out before the interrupt occurred.

A clean way to accomplish this (better than the solution we implemented in Exam 2) is to introduce a First Part Done (FPD) flag. When the vector instruction has performed some operations that should not be re-executed if the vector instruction is interrupted, the processor sets FPD to 1. When a vector instruction is fetched, FPD is checked. If FPD=0, this is the initial fetch of the vector instruction, and processing proceeds normally. If FPD=1, this indicates that the vector instruction has been re-fetched after an interrupt service routine has finished. In this case, the vector instruction picks up where it left off when the interrupt occurred.

For your reference, the solution for the state machine of the VLD instruction in Problem Set 5 is shown on page 16.

Part a (5 points): Consider all the operations involving the FPD flag. We do not want to have a separate register for FPD. Where should FPD information be kept to get an efficient implementation? **Why?**

The FPD flag can be stored as a bit in the PSR. Since the PSR is already saved during interrupt initiation, this would be very efficient.

Part b (6 points): Assume that the interrupt initiation sequence is already implemented as in Lab 4, i.e. just by pushing PSR and PC, and then jumping to the appropriate interrupt vector. Describe all the changes to the basic interrupt initiation sequence required to implement the interruptible VLD. **Make sure that your implementation is able to correctly execute other vector instructions -including VLD- in the interrupt service routine**.

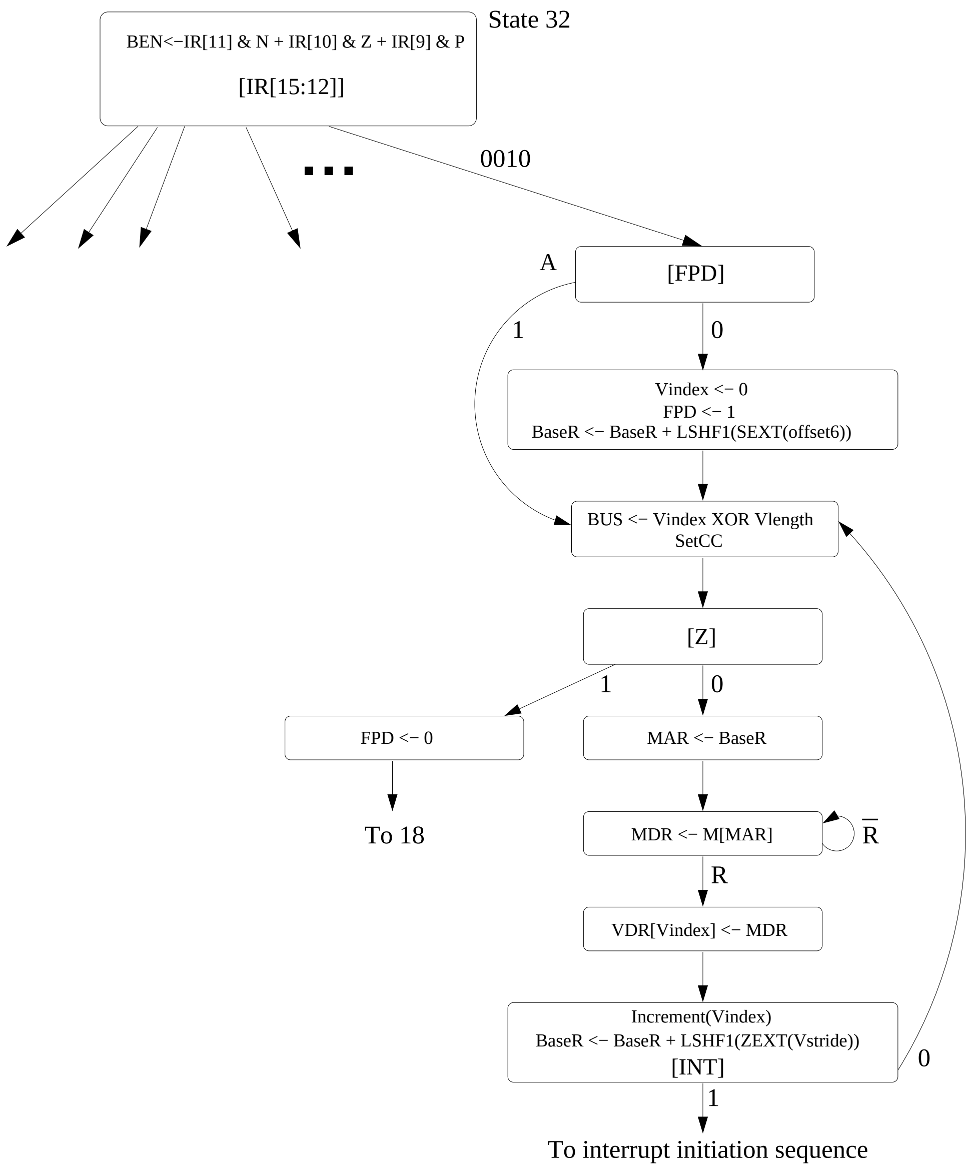
Push Vindex, Vlength, and Vstride onto the stack. Clear the FPD bit after pushing the PSR onto the stack.

Name:__________________________

Problem 7 continued

Part c (9 points): We show the beginning of the state diagram necessary to implement VLD. Using the notation of the LC-3b State Diagram, add the states you need to implement an interruptible VLD that uses the FPD flag. Inside each state describe what happens in that state. You can assume that you are allowed to make any changes to the data path and microsequencer that you find necessary. You do not have to make/show these changes.

NOTES:
1. Your implementation must support servicing interrupts in the middle of VLD.
2. Clearly indicate in which state you check for interrupts.
3. Assume that FPD is initialized to 0 when the machine is reset.
4. As in the problem set, you are allowed to clobber the condition codes and the BaseR.
5. Make sure that your implementation works for Vlength = 0.



State 32
BEN<−IR[11] & N + IR[10] & Z + IR[9] & P
[IR[15:12]]
...
0010
A
[FPD]
1
0
Vindex <− 0
FPD <− 1
BaseR <− BaseR + LSHF1(SEXT(offset6))
BUS <− Vindex XOR Vlength
SetCC
[Z]
1
0
FPD <− 0
To 18
MAR <− BaseR
MDR <− M[MAR]
$\overline{R}$
R
VDR[Vindex] <− MDR
Increment(Vindex)
BaseR <− BaseR + LSHF1(ZEXT(Vstride))
[INT]
0
1
To interrupt initiation sequence

Name:__________________________

Problem 7 continued

Part d (5 points): Does the interrupt initiation sequence push BaseR onto the stack? Why or why not?

No. According to the LmmVC-3 ISA, the hardware is not responsible for saving registers R0 - R7. The ISR/ESR programmer will save and restore a register if need be.

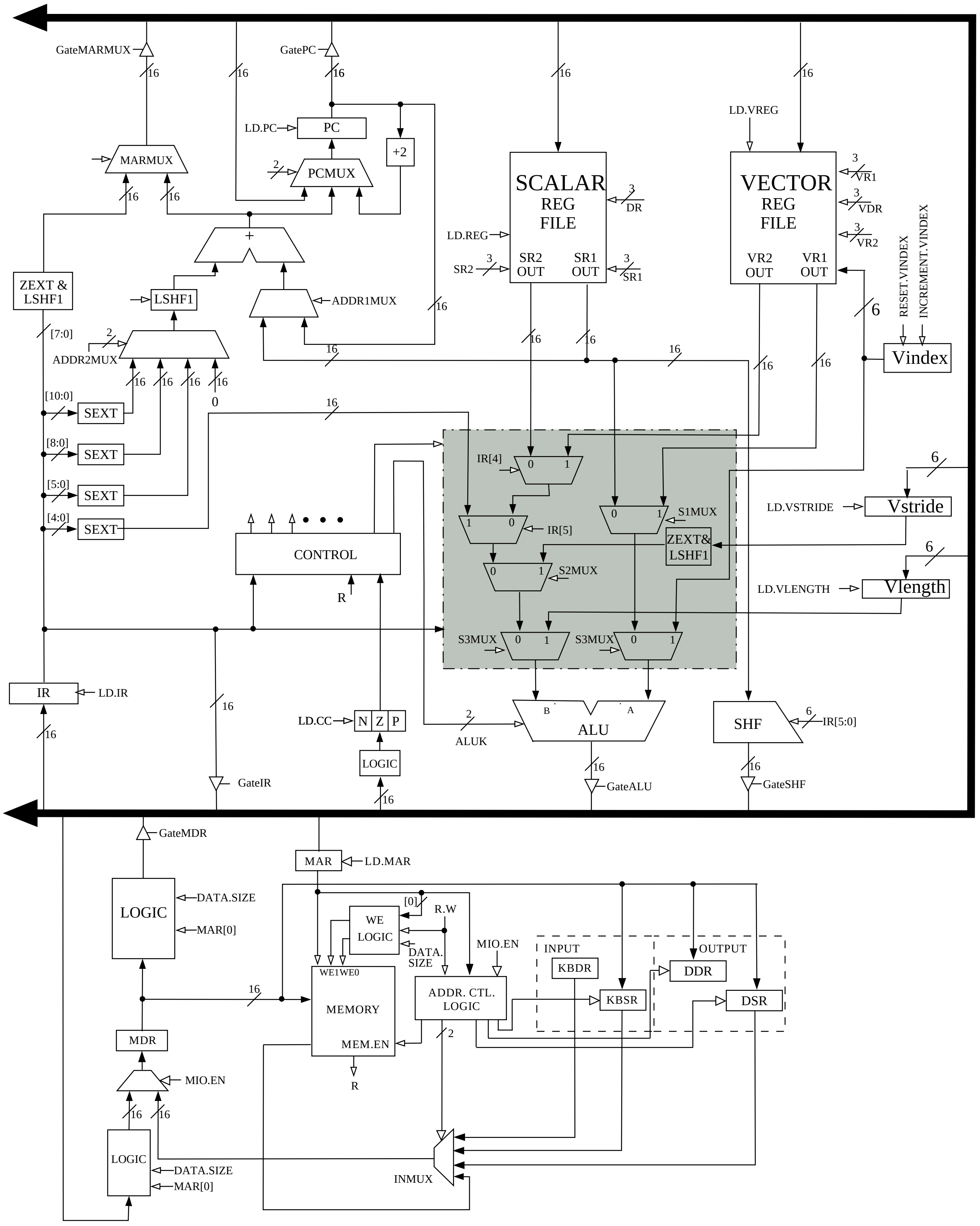


Figure 1: Modified data path to implement vector instructions

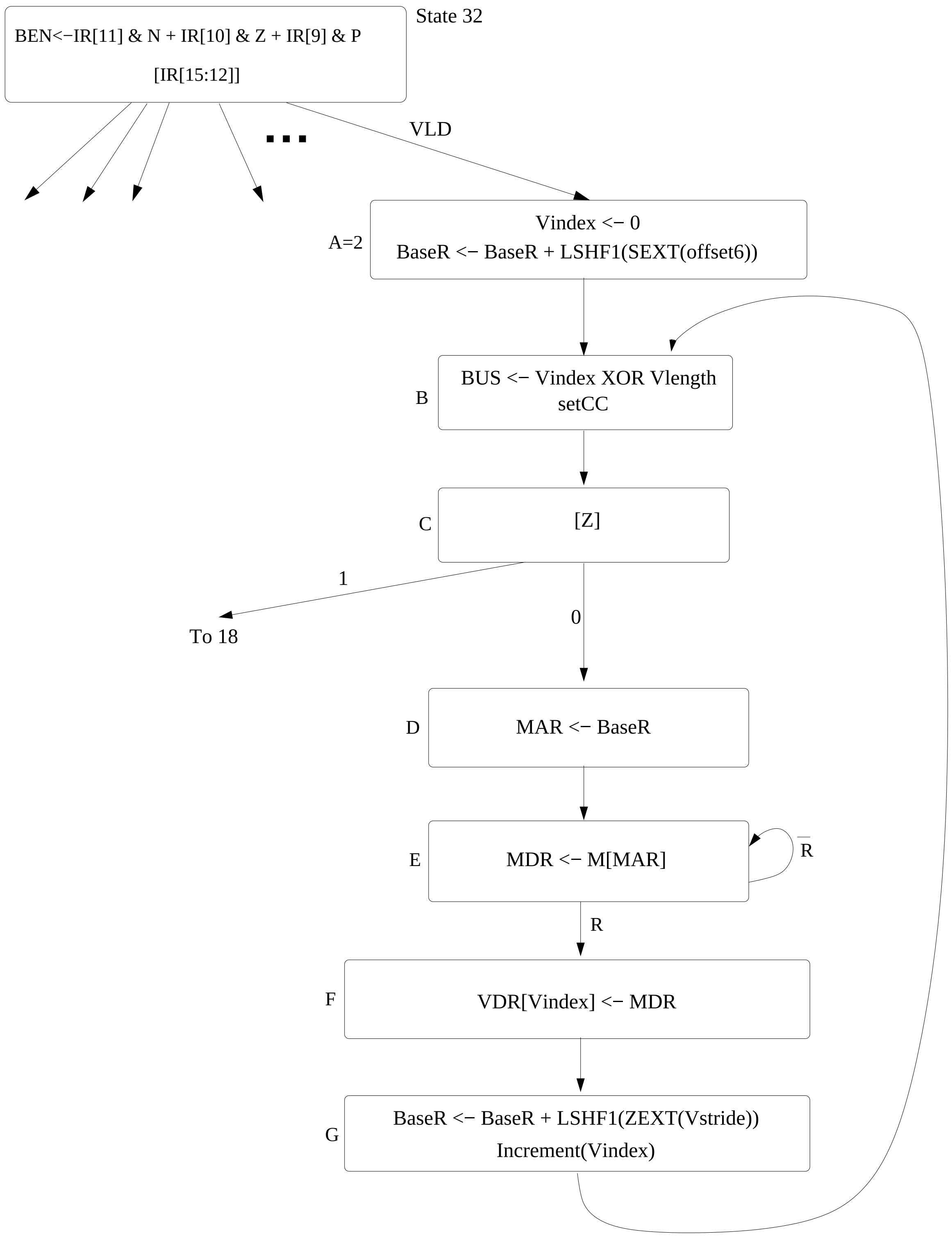

State 32
BEN<−IR[11] & N + IR[10] & Z + IR[9] & P
[IR[15:12]]
...
VLD
A=2
Vindex <− 0
BaseR <− BaseR + LSHF1(SEXT(offset6))
B
BUS <− Vindex XOR Vlength
setCC
C
[Z]
1
To 18
0
D
MAR <− BaseR
E
MDR <− M[MAR]
$\overline{R}$
R
F
VDR[Vindex] <− MDR
G
BaseR <− BaseR + LSHF1(ZEXT(Vstride))
Increment(Vindex)

Figure 2: State diagram of the non-interruptible VLD instruction from Problem Set 5