## **Department of Electrical and Computer Engineering**

### **The University of Texas at Austin**

EE 460N, Spring 2017

Problem Set 4 Solutions

Due: April 3, before class

Yale N. Patt, Instructor

Chirag Sakhuja, Sarbartha Banerjee, Jon Dahm, Arjun Teh, TAs

## **Instructions**

You are encouraged to work on the problem set in groups and turn in one problem set for the entire group. The problem sets are to be submitted on Canvas. Only one student should submit the problem set on behalf of the group. The only acceptable file format is PDF. Include the name of all students in the group in the file.

*You will need to refer to the assembly language handouts and the LC-3b ISA on the course website.*

/\* This problem has been moved here from the previous problem set \*/

### **Problem 1**

Size of a page is 512 bytes.

Number of bits of address required to calculate the offset within a page is 9.

Number of frames in physical memory is (8K bytes) ÷ (512 bytes) = 213 ÷ 29 = 24.

Size of PFN is 4 bits.

Size of PTE equals 1 (Valid) + 1 (Modified) + 2 (access control) + 4 (PFN) = 8 bits = 1 byte.

Number of virtual pages in System Space is (3 × 212) ÷ (29) = 24 pages.

Size of System Page Table is 24 × 1 byte = 24 bytes = 24 × 8 bits = 192 bits.

### 

### 

### **Problem 2**

|  |  |  |
| --- | --- | --- |
| **Reference** | **TLB hit** | **Page Fault** |
| 0 | X |  |
| 13 | X |  |
| 5 |  |  |
| 2 |  |  |
| 14 |  | X |
| 14 | X |  |
| 13 |  |  |
| 6 |  | X |
| 6 | X |  |
| 13 | X |  |
| 15 |  | X |
| 14 |  |  |
| 15 | X |  |
| 13 | X |  |
| 4 |  | X |
| 3 |  | X |

TLB hit rate = 7/16.

TLB contains entries for pages 3, 4, and 13.

Solutions for the final contents of the frames of physical memory may differ slightly depending on what order the initially empty frames were allocated; however, no page should appear in more than one frame. Possible answers are shown below.

|  |  |
| --- | --- |
| Frame 0 | Page 14 (or 6 or 15) |
| Frame 1 | Page 13 |
| Frame 2 | Page 3 |
| Frame 3 | Page 2 |
| Frame 4 | Page 6 (or 14 or 15) |
| Frame 5 | Page 4 |
| Frame 6 | Page 15 (or 6 or 14) |
| Frame 7 | Page Table |

**Problem 3**

We can determine from the given information that:

* A Virtual Address (VA) is 16 bits (216 = 64KB)
* A Physical Address (PA) is 12 bits (212 = 4KB)
* The number of bits for the offset is 8 (28 = 256 bytes)

The breakdown for a Virtual Address (VA) must be:

* VA[15:14] (2 bits) : Denotes the region of memory (P0, P1, System)
* VA[13:8] (6 bits) : The Virtual Page Number (VPN)
* VA[7:0] (8 bits) : The offset

The breakdown for a Physical Address (PA) must be:

* PA[11:8] (4 bits) : The Page Frame Number (PFN)
* PA[7:0] (8 bits) : The offset

1. Answer: x825C  
     
   Given the VAX 2-level translation scheme, we know that this VA must be in system space. Therefore, the top 2 bits of the VA (VA[15:14]) must be 10 (in binary). We also know that the offset of the VA is the same as the offset of the PA, so the bottom 8 bits of the VA (VA[7:0]) must be x5C (01011100 in binary). Now, all we have to figure out is the 6 bit Virtual Page Number (VPN). It was given that the 1st access to physical memory (let's call this PA1) was at location x108. Once again, given the VAX 2-level translation scheme, we know that PA1 = SBR + (size of PTE in bytes) × VPN. Solving this equation for VPN we get VPN = (PA1 - SBR) ÷ (size of PTE in bytes).  
     
   It was given that PA1 is x108, SBR is x100, and the size of a PTE is 4 bytes. Therefore, the VPN is x2 (000010 in binary). The complete VA is therefore 10 000010 01011100 (in binary) which is x825C.
2. Answer: x80000009  
     
   The contents of physical address x45C (the 2nd access to physical memory) is a PTE. We know that a PTE is a 32 bit value that consists of 1 valid bit (PTE[31]), and a 4-bit PFN (PTE[3:0]). All other bits of the PTE (PTE[30:4]) are 0. The contents of this PTE are used to form the address of the 3rd access to physical memory (x942). Therefore, the PFN bits of the PTE must be x9, and the valid bit must be 1. This implies that the PTE is x80000009.
3. Answer x0342  
     
   First, we must determine if this VA is in P0 space or P1 space. To determine this, we have to figure out if P0BR or P1BR was used to compute the virtual address x825C (the answer to part a). It was given that P0BR is x8250, and that P1BR is x8350. Since x8350 is greater than x825C, we know that we could not have used P1BR to compute x825C, and therefore we must have used P0BR which means the VA is in P0 space. Therefore, the top 2 bits of the virtual address (VA[15:14]) must be 00 (in binary). We also know that the offset of the VA is the same as the offset of the PA, so the bottom 8 bits of the VA (VA[7:0]) must be x42 (01000010 in binary). Now, all we have to figure out is the 6 bit Virtual Page Number (VPN). Once again, given the VAX 2-level translation scheme, we know that x825C = P0BR + ((size of PTE in bytes) × VPN ).  
     
   Solving this equation for VPN we get VPN = (x825C - P0BR) ÷ (size of PTE in bytes).  
     
   It was given that P0BR is x8250, and the size of a PTE is 4 bytes. Therefore, the VPN is x3 (000011 in binary). The complete VA is therefore 00 000011 01000010 (in binary) which is x0342.

**Problem 4**

1. # virtual pages = virtual address space ÷ size of page = 212 Bytes ÷ 25 Bytes/page = 27 pages
2. # physical frames = physical address space ÷ size of frame = 29Bytes ÷ 25 Bytes/frame = 24 frames. Therefore, 4 bits are needed to specify the PFN.  
     
   Size of PTE = Valid bit + Modified bit + access control bits + PFN bits = 1 + 1 + 2 + 4 = 8 bits (1 Byte)
3. User space = (3/4) × Virtual address space = (3/4) × 27 pages = 3 × 25 pages. Each page of user space will have a PTE in the user space page table.  
     
   Size of user page table is # of entries × size of PTE = (3 × 25 entries) × 1 Byte/entry = 96 Bytes.  
     
   # of pages = 96 Bytes ÷ 25 Bytes/page = 3 pages.
4. We'll use the prefix “b” to indicate a binary number in this solution. Also, for clarity, we will call the virtual address x7AC the virtual address of X (VA\_X).  
   Virtual address VA\_X = x7AC  
   The three parts of this virtual address are:  
   VA\_X[11:10]: b01 (indicates that this is an address in user space)  
   VA\_X[11:5] (7 bits): Virtual Page Number = b0111101  
   VA\_X[4:0] Offset within page: b01100
   * X is on page x03D of user space
   * VA of the PTE of the page containing x is VA\_PTE\_X = PTBR + (x03D × 1) = x380 + x03D = x3BD.
   * Virtual page of System Space of PTE is VA\_PTE\_X[11:5] = x01D.
   * PA of the PTE of this page of System Space is PA\_PTE\_PTE = SBR + VA\_PTE\_X[11:5] × 1 = x1E0 + x01D = x1FD.
   * The PTE of this page of System Space is:   
     PTE\_PTE\_X = Memory[x1FD] = xB8  
     PFN\_PTE\_X = PTE\_PTE\_X[3:0] = x8  
     PA of the PTE of the page containing X:  
     PA\_PTE\_X = PFN\_PTE\_X concatenated with VA\_PTE\_X[4:0] = x11D  
     PTE\_X = Memory[x11D] = x83  
     PFN\_X = PTE\_X[3:0] = x3  
     PA\_X = PFN\_X concantenated with VA\_X[4:0] = x06C

**Problem 5**

We'll use the prefix “b” to indicate a binary number in this solution.

Virtual address VA\_X = x3456789A

The three parts of this virtual address are:

VA\_X[31:30]: b00 (indicates that this is an address in P0 space)

VA\_X[29:9] (21 bits): Virtual Page Number = b 11 0100 0101 0110 0111 100 (x1A2B3C)

VA\_X[8:0] Offset within page: b010011010

* x is on page x1A2B3C of P0 space
* VA of the PTE of the page containing X:  
  VA\_PTE\_X = P0BR + (x1A2B3C × 4) = x8AC40000 + x68ACF0 = x8B2CACF0
* Virtual page of System Space of PTE is VA\_PTE\_X[29:9] = x59656
* PA of the PTE of this page of System Space is PA\_PTE\_PTE = SBR + VA\_PTE\_X[29:9] × 4 = x22D958
* The PTE of this page of System Space is PTE\_PTE\_X = Memory[x22D958] = x800F5D37  
  PFN\_PTE\_X = PTE\_PTE\_X[20:0] = xF5D37  
  PA of the PTE of the page containing X:  
  PA\_PTE\_X = PFN\_PTE\_X concatenated with VA\_PTE\_X[8:0] = x1EBA6EF0  
  PTE\_X = Memory[x1EBA6EF0] = x80000A72  
  PFN\_X = PTE\_X[20:0] = xA72  
  PA\_X = PFN\_X concantenated with VA\_X[8:0] = x14E49A

**Problem 6**

Including instruction fetch, every instruction can generate a page fault. Ignoring instruction fetch, LDB, LDW, STB, STW, TRAP, RTI can generate a page fault (If the trap vector table or system stack is always in physical memory, then the TRAP or RTI won't generate a page fault).

Including the instruction fetch, RTI can generate the maximum number of page faults (3) and LDB, LDW, STB, STW, TRAP can generate the next most number of page faults (2). (Points will not be deducted for RTI)

**Problem 7**

An 8KB cache size with a 8B line size, in a 4-way set associative cache means there are 8KB ÷ (4 × 8B) = 256 sets in the cache.

Since there are 256 or 28 sets, 8 bits are required to index into the correct set. Since there are 8B or 23 bytes in a cache line, 3 bits are required to find the correct byte within a block. Given a 24-bit address space, this leaves 24 - 8 - 3 = 13 bits left over for the tag store. Additionally, the tag store must hold 2 bits for the V/NV replacement policy and 1 valid bit. This means each cache line must have a 16-bit tag store associated with it. 2B of tag store times (256 × 4) cache lines in the cache means that the tag store, in total takes up 2048 bytes, which is 16384 bits

**Problem 8**

The size of the tag store is 212 + 28. We know that the size of the tag store can be given as the product of number of sets and number of bits per set.

We also know that address space is 16-bits. Hence, tag + index + bib = 16

In order to find bib, we need to find index and tag. The following bits are necessary for each set of the tag store:

* 1 bit for LRU (remember: you only need one bit per set for a 2-way associative cache)
* 2 valid bits
* 2 dirty bits
* 2 × tag tag bits

Total number of bits per set is 5 + 2 × tag. An important conclusion that can be drawn is that number of bits per set will always be an odd number.

We will also use the fact that number of sets is always a power of 2 (since it is indexed using an integer number of bits). Given the size of the tag store, index has to be a number less than 8 (the size of the tag store is indivisible by 29 or greater).

The only value of index that fits both criterion is 8. Therefore:

Number of sets = 28 = 256

Bits per row = 4352 ÷ 256 = 17

17 = 5 + 2 × tag ⇒ tag = 6

6 + 8 + bib = 16 ⇒ bib = 2

Hence, the cache block size is 4 bytes

**Problem 9**

average\_access\_time\_per\_instruction = 1 × instruction\_access\_time + 0.3 × data\_access\_time

We can use the following equations in both part a and part b to compute the instruction and data access time.

instruction\_access\_time = 1 + 0.05 × (6 + 0.15 × mem\_latency)

data\_access\_time = 1 + 0.10 × (6 + 0.25 × mem\_latency)

1. We need to calculate the memory latency. Since each cache block is 8 words, it will take 8 accesses to get a cache block from memory.   
   (-)(...)(-)  
    (-)(...)(-)  
    (-)(...)(-)  
    (-)(...)(-)  
    (-)(...)(-)  
    (-)(...)(-)  
    (-)(...)(-)  
    (-)(...)(-)  
     
   |<---------------------- 169 cycles ----------------------------->|  
     
   Using the equations, we get 2.5675 cycles for instruction\_access\_time and 5.825 for data\_access\_time.  
     
   The average latency is 2.5675 + 0.3 × 5.825 = 4.315 cycles
2. We again need to compute the memory latency:  
   (-)(.............)(-)  
    (-)(.............)(-)  
    (-)(.............)(-)  
    (-)(.............)(-)  
    (-)(...........)(-)  
    (-)(...........)(-)  
    (-)(...........)(-)  
    (-)(...........)(-)  
     
   |<--------21-----><1><-----20----><---- 4 --->|  
     
   So the mem\_latency is 46 cycles. Again using the equations, the instruction\_access\_time is 1.645 and data\_acces\_time is 2.75.  
     
   The average latency is 1.645 + 2.75 × 0.3 = 2.47 cycles
3. We need to compute memory latency for 8-way interleaving  
   (-)(.............)(-)  
    (-)(.............)(-)  
    (-)(.............)(-)  
    (-)(.............)(-)  
    (-)(...........)(-)  
    (-)(...........)(-)  
    (-)(...........)(-)  
    (-)(...........)(-)  
   |--------21-------|--------8-----------|  
     
   So memory latency is 29 cycles. Using the equations, we get the instruction\_access\_time as 1.517 and data\_acces\_time as 2.325.  
     
   The average latency is 1.517 + .3 \* 2.325 = 2.215
4. For the 4-way interleaved memory, the improvement is (4.315 - 2.47) ÷ 4.315 = 0.4276 Therefore, the average latency improves by 42.76%  
     
   For the 8-way interleaved memory, the improvement is (4.315 - 2.215) ÷ 4.315 = 0.4867 Therefore, the average latency improves by 48.67%

**Problem 10**

1. The address is divided into 3 portions:

* address[1:0] (2 bits) for the byte in the block
* address [4:2] (3 bits) for the cache index
* address[11:5] (7 bits) for the tag  
    
  The contents of the cache at the end of each pass are the same. They are shown below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Valid** | **Tag** | **Data (addresses are written inside each byte)** | | | |
| 1 | 0010 000 | 203 | 202 | 201 | 200 |
| 1 | 0010 000 | 207 | 206 | 205 | 204 |
| 1 | 0010 000 | 20B | 20A | 209 | 208 |
| 1 | 0010 010 | 24F | 24E | 24D | 24C |
| 1 | 0010 111 | 2F3 | 2F2 | 2F1 | 2F0 |
| 1 | 0010 111 | 2F7 | 2F6 | 2F5 | 2F4 |
| 1 | 0010 000 | 21B | 21A | 219 | 218 |
| 1 | 0010 000 | 21F | 21E | 21D | 21C |

The hit/miss information for each pass is shown below:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Reference** | 200 | 204 | 208 | 20C | 2F4 | 2F0 | 200 | 204 | 218 | 21C | 24C | 2F4 |
| **Pass 1:** | M | M | M | M | M | M | H | H | M | M | M | H |
| **Pass 2:** | H | H | H | M | H | H | H | H | H | H | M | H |
| **Pass 3:** | H | H | H | M | H | H | H | H | H | H | M | H |
| **Pass 4:** | H | H | H | M | H | H | H | H | H | H | M | H |

Hit rate is 33/48

2. The address is divided into two: 2 bits for identifying the byte in the block, 10 bits for the tag. No bits are needed for cache index. The following table shows the contents of the cache at the end of each pass (Valid bits and Tags are ignored, they should be obvious. Starting addresses of blocks in the cache are provided):

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Pass** | **Way 0 Data** | **Way 1 Data** | **Way 2 Data** | **Way 3 Data** | **Way 4 Data** | **Way 5 Data** | **Way 6 Data** | **Way 7 Data** |
| 1 | 200 | 204 | 24C | 20C | 2F4 | 2F0 | 218 | 21C |
| 2 | 200 | 204 | 21C | 24C | 2F4 | 20C | 2F0 | 218 |
| 3 | 200 | 204 | 218 | 21C | 2F4 | 24C | 20C | 2F0 |
| 4 | 200 | 204 | 2F0 | 218 | 2F4 | 21C | 24C | 20C |

The hit/miss information for each pass is shown below:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Reference:** | 200 | 204 | 208 | 20C | 2F4 | 2F0 | 200 | 204 | 218 | 21C | 24C | 2F4 |
| **Pass 1:** | M | M | M | M | M | M | H | H | M | M | M | H |
| **Pass 2:** | H | H | M | M | H | M | H | H | M | M | M | H |
| **Pass 3:** | H | H | M | M | H | M | H | H | M | M | M | H |
| **Pass 4:** | H | H | M | M | H | M | H | H | M | M | M | H |

Hit rate is 21/48

3. The address is divided into 3 portions: 2 bits for identifying the byte in the block, 1 bit for the cache index, 9 bits for the tag. The contents of the cache at the end of Pass 1:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Way 0** | | | **Way 1** | | | **Way 2** | | | **Way 3** | | |
| **V** | **Tag** | **Data** | **V** | **Tag** | **Data** | **V** | **Tag** | **Data** | **V** | **Tag** | **Data** |
| 1 | 0010 0000 0 | 203-200 | 1 | 0010 0000 1 | 20B-208 | 1 | 0010 1111 0 | 2F3-2F0 | 1 | 0010 0001 1 | 21B-218 |
| 1 | 0010 0000 0 | 207-204 | 1 | 0010 0100 1 | 24F-24C | 1 | 0010 1111 0 | 2F7-2F4 | 1 | 0010 0001 1 | 21F-21C |

The hit/miss information for each pass is shown below:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Reference:** | 200 | 204 | 208 | 20C | 2F4 | 2F0 | 200 | 204 | 218 | 21C | 24C | 2F4 |
| **Pass 1:** | M | M | M | M | M | M | H | H | M | M | M | H |
| **Pass 2:** | H | H | H | M | H | H | H | H | H | M | M | H |
| **Pass 3:** | H | H | H | M | H | H | H | H | H | M | M | H |
| **Pass 4:** | H | H | H | M | H | H | H | H | H | M | M | H |

Contents of the second set of the cache (index equals 1) after pass 2, 3, and 4 (the first set remains the same):

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Pass** | **Way 0** | | | **Way 1** | | | **Way 2** | | | **Way 3** | | |
| **V** | **Tag** | **Data** | **V** | **Tag** | **Data** | **V** | **Tag** | **Data** | **V** | **Tag** | **Data** |
| 2 | 1 | 0010 0000 0 | 207-204 | 1 | 0010 0001 1 | 21F-21C | 1 | 0010 1111 0 | 2F7-2F4 | 1 | 0010 0100 1 | 24F-24C |
| 3 | 1 | 0010 0000 0 | 207-204 | 1 | 0010 0100 1 | 24F-24C | 1 | 0010 1111 0 | 2F7-2F4 | 1 | 0010 0001 1 | 21F-21C |
| 4 | 1 | 0010 0000 0 | 207-204 | 1 | 0010 0001 1 | 21F-21C | 1 | 0010 1111 0 | 2F7-2F4 | 1 | 0010 0100 1 | 24F-24C |

Hit Rate is 30/48

**Problem 11**

### **Block Size**

The following table lists hit ratios for different cache block sizes given the access sequence 1 (0, 2, 4, 8, 16, 32):

|  |  |
| --- | --- |
| **Block Size** | **Hit Ratio** |
| 1B | 0/6 |
| 2B | 0/6 |
| 4B | 1/6 |
| 8B | 2/6 |
| 16B | 3/6 |
| 32B | 4/6 |

Since the hit ratio is reported as 0.33 for this sequence, the block size must be 8 bytes. Therefore, the accesses look like this:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Address** | 0 | 2 | 4 | 8 | 16 | 32 |
| **Hit/Miss** | M | H | H | M | M | M |

### **Associativity**

Notice that all of the addresses of sequence 2 (0, 512, 1024, 1536, 2048, 1536, 1024, 512, 0) are multiples of 512. This means that they all map to the same set, since the size of the cache can only be 256 or 512 bytes. Therefore, the hit ratio would be 0/9 in case of a direct-mapped cache. In case of a 2-way cache, the accesses would behave as follows:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Address** | 0 | 512 | 1024 | 1536 | 2048 | 1536 | 1024 | 512 | 0 |
| **Hit/Miss** | M | M | M | M | M | H | M | M | M |

The resulting hit rate would be 1/9. Similarly, for 4-way cache:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Address** | 0 | 512 | 1024 | 1536 | 2048 | 1536 | 1024 | 512 | 0 |
| **Hit/Miss** | M | M | M | M | M | H | H | H | M |

The resulting hit rate would be 3/9, and thus the cache is 4-way set associative.

### **Cache Size**

If the cache size were 256B, there would be 3 index bits and all of the adresses in sequence 3 (0, 64, 128, 256, 512, 256, 128, 64, 0) would map to the same set:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Address** | 0 | 64 | 128 | 256 | 512 | 256 | 128 | 64 | 0 |
| **Hit/Miss** | M | M | M | M | M | H | H | H | M |

The resulting hit ratio would be 3/9, and thus the cache size is 256B. For completeness, here's how the accesses would look like in case of a 512B cache (4 index bits):

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Address** | 0 | 64 | 128 | 256 | 512 | 256 | 128 | 64 | 0 |
| **Hit/Miss** | M | M | M | M | M | H | H | H | H |

### **Replacement Policy**

In case of the FIFO replacement policy, the accesses of sequence 3 (0, 512, 1024, 0, 1536, 0, 2048, 512) would look as follows:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Address** | 0 | 512 | 1024 | 0 | 1536 | 0 | 2048 | 512 |
| **Hit/Miss** | M | M | M | H | M | H | M | H |

The resulting hit ration would be 3/8. However, in case of the LRU replacement policy the hit rate would be 0.25:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 512 | 1024 | 0 | 1536 | 0 | 2048 | 512 |
| M | M | M | H | M | H | M | M |

Thus the replacement policy is LRU.

**Problem 12**

The state diagram for the FSM device controller can be found in the handouts section of the website under the notes for I/O. Note that in the notes, the transition from BGout to IDLE is based on the SACK signal. This is to illustrate the second race condition which is corrected by basing the transition on (NOT BGin).

The input and output signals of the controller are:

|  |  |  |
| --- | --- | --- |
| Signal | Type | Function |
| D | Input | Asserted when the device needs to initiate a bus transaction |
| BGin | Input | Incoming bus grant signal, asserted by the priority arbitration unit |
| BBSYin | Input | Asserted by the current bus master. Negative edge indicates the end of a bus cycle. |
| MSYN | Input | Master-side handshaking signal that controls the bus transaction between the bus master and the slave |
| SSYN | Input | Slave-side handshaking signal that controls the bus transaction between the bus master and the slave |
| BRout | Output | Asserted to request the bus. Goes to the priority arbitration unit. |
| SACK | Output | Asserted by the device that has won the arbitration |
| BBSYout | Output | Same as BBSYin |
| BGout | Output | Asserted when the controller needs to pass the bus grant signal down the daisy chain. |

Two race conditions are:

1. This race condition is subtle. From the PAU side the PAU asserts BGj, which works its way down the daisy chain to all devices at BRj. Before SACK is asserted, a controller asserts BRk, where k is higher priority than j. PAU now asserts BGk, and you have two BG signals propagating, which will result in two controllers thinking they are the next bus master.  
     
   Solution: PAU latches BR signals when it sees NOT-BBSY, indicating it is okay to grant the bus again. NOT-SACK is also gated (after sufficient delay) with the BG signals, guaranteeing that a BG signal cannot be asserted until after PAU logic has taken effect. Any subsequent BR signal does not get latched and so can not affect the PAU logic. We also discussed that using NOT-BBSY as load-enable can be problematic if the bus is not used for a long preiod. By using NOR(BG0, BG1,...) as load-enable, we solved this issue.
2. Let's say device controller D1 is in BGout state. This means that some device D2 that is down the same daisy chain as D1 had requested and is granted the bus. Let's say the device of D1 asserts the D signal while D1 is in BGout state. D2 will eventually receive the BGin signal and transition to the SACK state. It will take some time for the SACK signal to travel to the priority arbitration unit. The SACK signal probably reaches D1 before it reaches the priority arbitration unit. Hence, when the SACK signal reaches D1 the BGin input of D1 is still being asserted. Therefore, upon receiving the SACK signal D1 will immediately transition to IDLE to BRout to SACK states. Hence, both D1 and D2 will be asserting the SACK signal which is not desirable. A simple solution that fixes this race condition is not transitioning to IDLE state if the BGin signal is still high.