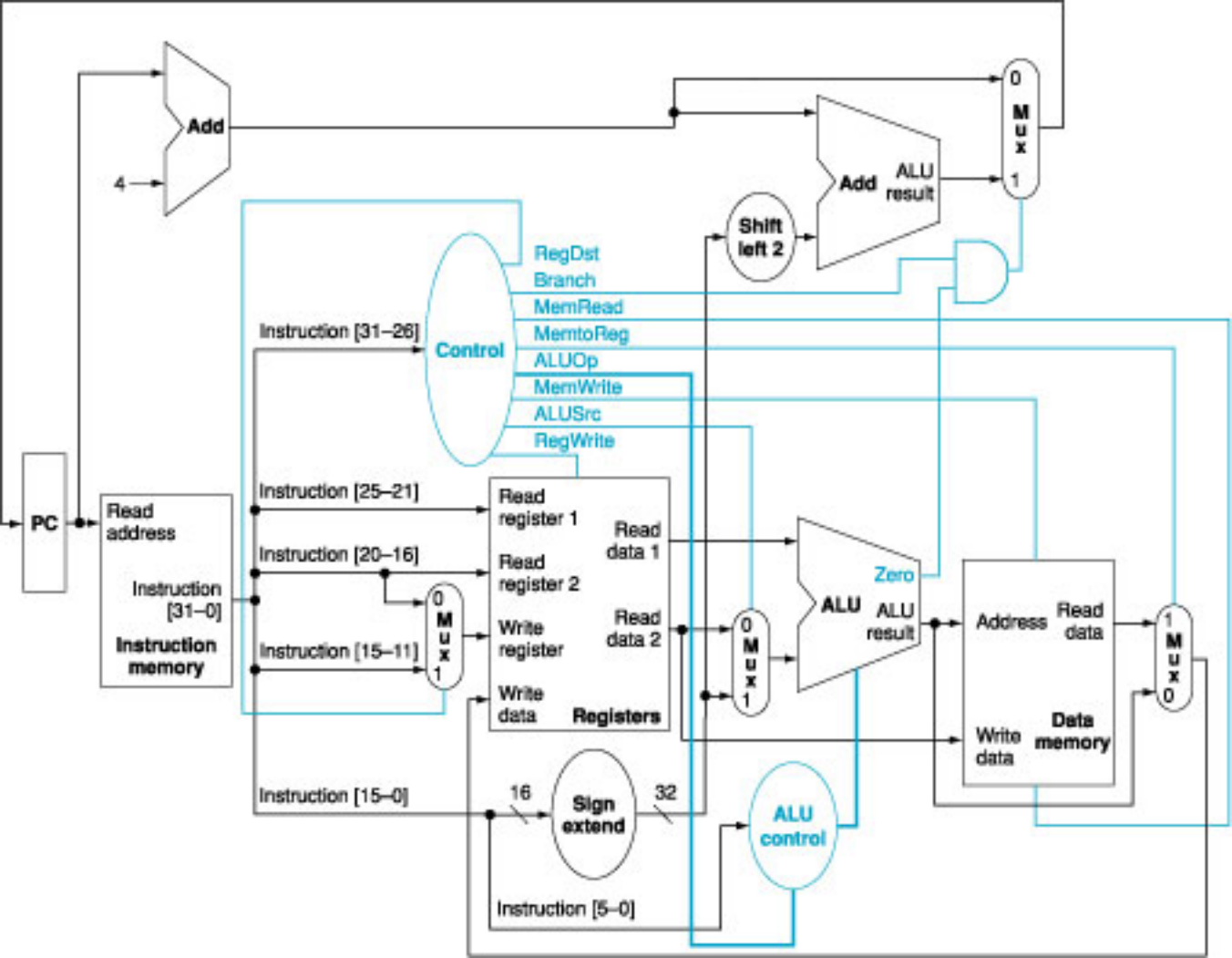


1. We wish to add a variant of the *lw* instruction which increments the index register after loading word from memory. This instruction (*l\_inc*) corresponds to the following two instructions:

*lw \$rt,L(\$rs)*

*addi \$rs,\$rs,4*



## ~~CHAPTER 5~~

~~5.11~~ A modification is required for the datapath of Figure 5.17 to perform the autoincrement by adding 4 to the \$rs register through an incrementer. Also we need a second write port to the register file because two register writes are required for this instruction. The new write port will be controlled by a new signal, "Write 2", and a data port, "Write data 2." We assume that the Write register 2 identifier is always the same as Read register 1 (\$rs). This way "Write 2" indicates that there is second write to register file to the register identified by "Read register 1," and the data is fed through Write data 2.

A new line should be added to the truth table in Figure 5.18 for the `l_inc` command as follows:

RegDst = 0: First write to \$rt.

ALUSrc = 1: Address field for address calculation.

MemtoReg = 1: Write loaded data from memory.

RegWrite = 1: Write loaded data into \$rt.

MemRead = 1: Data memory read.

MemWrite = 0: No memory write required.

Branch = 0: Not a branch, output from the PCSrc controlled mux ignored.

ALUOp = 00: Address calculation.

Write2 = 1: Second register write (to \$rs).

Such a modification of the register file architecture may not be required for a multiple-cycle implementation, since multiple writes to the same port can occur on different cycles.

~~5.12~~ This instruction requires two writes to the register file. The only way to implement it is to modify the register file to have two write ports instead of one.

2. If the time for an ALU operation can be shortened by 25%:
  - a. Will it affect the speedup obtained from pipelining? If yes, by how much? Otherwise, why?
  - b. What if the ALU operation now takes 25% more time?
3. Using a drawing, show the forwarding paths needed to execute the following four instructions:

*add \$3, \$4, \$6*

*sub \$5, \$3, \$2*

*lw \$7, 100(\$5)*

*Add \$8, \$7, \$2*

## ~~CHAPTER 6~~

### ~~6.1~~

a.

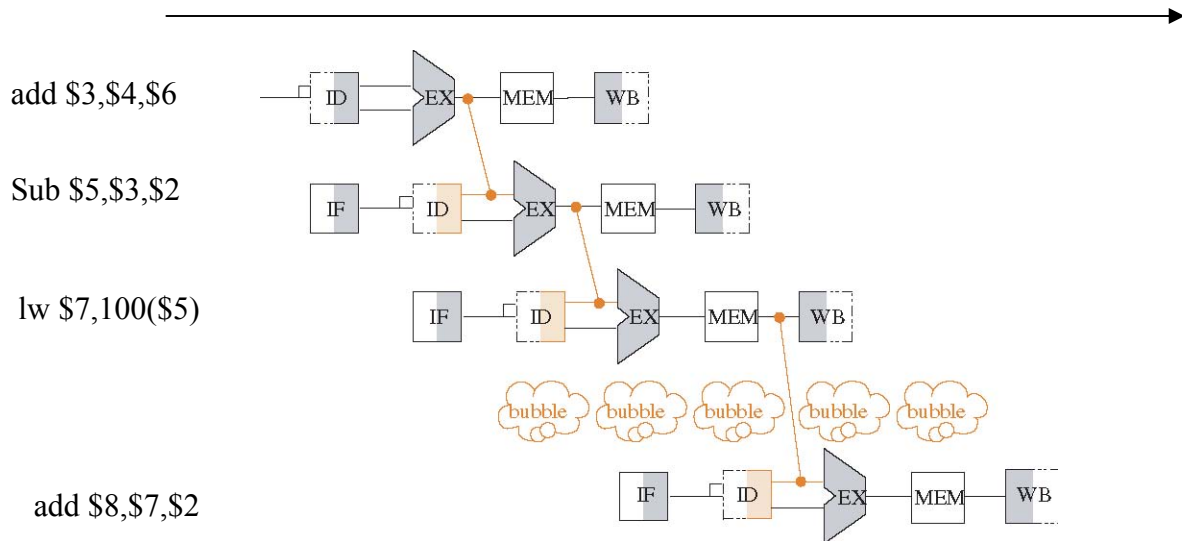
Shortening the ALU operation will not affect the speedup obtained from pipelining. It would not affect the clock cycle.

b.

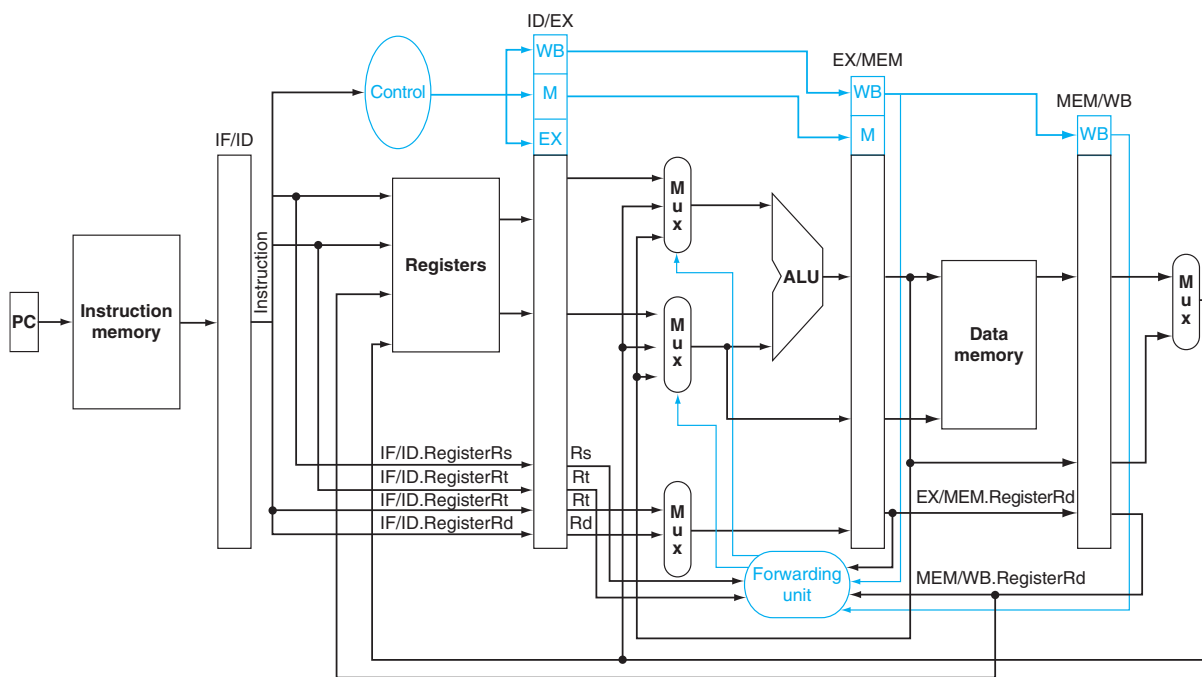
If the ALU operation takes 25% more time, it becomes the bottleneck in the pipeline. The clock cycle needs to be 250 ps. The speedup would be 20% less.

### ~~6.3~~

Program  
Execution  
Order  
(in inst.)



4. We have a program of  $10^3$  instructions in the format of '*lw, add, lw, add,...*' The *add* instruction depends (and only depends) on the *lw* instructions right before it. The *lw* instruction also depends (and only depends) on the *add* instruction before it. If the program is executed on the pipelined datapath of the following figure
- What would be the actual CPI?
  - Without forwarding, what would be the actual CPI?



**FIGURE 4.56 The datapath modified to resolve hazards via forwarding.** Compared with the datapath in Figure 4.51, the additions are the multiplexers to the inputs to the ALU. This figure is a more stylized drawing, however, leaving out details from the full datapath, such as the branch hardware and the sign extension hardware. Copyright © 2009 Elsevier, Inc. All rights reserved.

~~6.21~~

a. There will be a bubble of 1 cycle between a lw and the dependent add since the load value is available after the MEM stage.

There is no bubble between an add and the dependent lw since the add result is available after the EX stage and it can be forwarded to the EX stage for the dependent lw. Therefore,  $\text{CPI} = \text{cycle/instruction} \sim 1.5$ .

b. Without forwarding, the value being written into a register can only be read in the same cycle. As a result, there will be a bubble of 2 cycles between an lw and the dependent add since the load value is written to the register after the MEM stage. Similarly, there will be a bubble of 2 cycles between an add and the dependent lw. Therefore,  $\text{CPI} \cong 3$ .

5. Consider executing the following code on the pipelined datapath

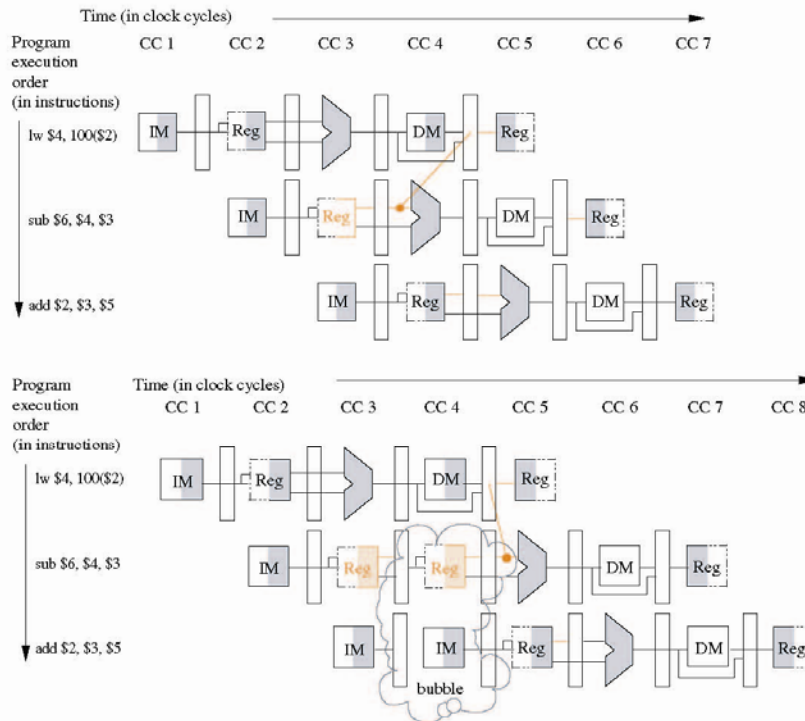
*lw \$4, 100(\$2)*

*sub \$6,\$4,\$3*

*add \$2,\$3,\$5*

How many cycles will it take to execute it? Draw a diagram that illustrates the dependences that need to be resolved, and provide another diagram that illustrates how the code will actually be executed.

**6.22** It will take 8 cycles to execute this code, including a bubble of 1 cycle due to the dependency between the `lw` and `sub` instructions.



6. A new processor can use either a write-through or write-back cache selectable through software
  - a. Assume the processor will run data-intensive applications with a large number of load and store operations. Explain which cache write policy should be used.
  - b. Consider the same question, but this time for a safety-critical system in which data integrity is more important than memory performance.
7. Here is a series of address references given as word addresses 2, 3, 11, 16, 21, 13, 64, 48, 19, 11, 3, 22, 4, 27, 6, and 11. Assuming a direct-mapped cache with 16 one-word blocks that is initially empty, label each reference in the list as a hit or a miss and show the final contents of the cache.

8. Given the following pseudo-code

```
Int array[10000, 100000]  
For each element array[i][j]{  
    array[i][j]= array[i][j]*2;  
}
```

9. Compute the total number of bits required to implement the cache in the figure (see below). The number is different from the size of the cache, which usually refers to the number of bytes of data stored in the cache. The number of bits needed to implement the cache represents the total amount of memory needed to store all the data, tags, and valid bits

## ~~CHAPTER 7~~

~~7.5~~ For the data-intensive application, the cache should be write-back. A write buffer is unlikely to keep up with this many stores, so write-through would be too slow.

For the safety-critical system, the processor should use the write-through cache.

This way the data will be written in parallel to the cache and main memory, and we could reload bad data in the cache from memory in the case of an error.

~~7.9~~ 2-miss, 3-miss, 11-miss, 16-miss, 21-miss, 13-miss, 64-miss, 48-miss, 19-miss, 11-hit, 3-miss, 22-miss, 4-miss, 27-miss, 6-miss, 11-set.

Cache set	Address
0000	48
0001	
0010	2
0011	3
0100	4
0101	21
0110	6
0111	
1000	
1001	27
1010	
1011	11
1100	
1101	13
1110	
1111	

7.11 C stores multidimensional arrays in row-major form. Therefore accessing the array in row-major form will be faster since there will be a greater degree of

temporal and spatial locality. Column-major form, on the other hand, will result

in capacity misses if the block size is more than one word.

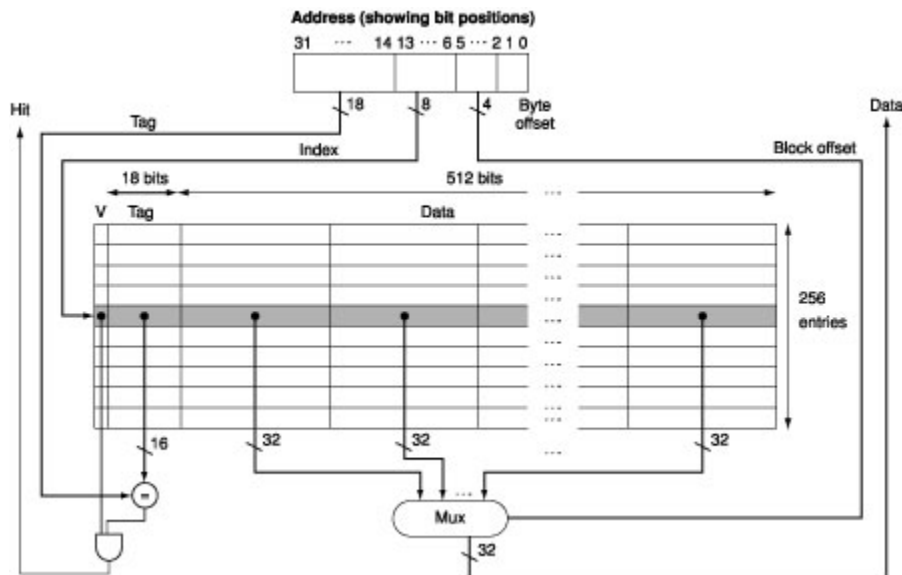
~~7.12 The Intrinsicity caches are 16 KB caches with 256 blocks and 16 words per~~

~~block. Data is 64 bytes = 512 bytes. The tag field is 18 bits (32 – (8 + 6)).~~

~~Total bits = 256 × (Data + Tag + Valid)~~

~~= 256 × (512 bits + 18 bits + 1 bit)~~

~~= 135,936 bits~~



~~7.14 The miss penalty is the time to transfer one block from main memory to the~~

~~cache. Assume that it takes 1 clock cycle to send the address to the main memory.~~

~~a.~~

~~Configuration (a) requires 16 main memory accesses to retrieve a cache block, and words of the block are transferred 1 at a time.~~

~~Miss penalty = 1 + 16 × 10 + 16 × 1 = 177 clock cycles.~~

~~b.~~

~~Configuration (b) requires 4 main memory accesses to retrieve a cache block and words of the block are transferred 4 at a time.~~

~~Miss penalty = 1 + 4 × 10 + 4 × 1 = 45 clock cycles.~~

~~c.~~

~~Configuration (c) requires 4 main memory accesses to retrieve a cache~~

11. Consider a virtual memory system with the properties: 40-bit virtual byte address, 16KB pages, 36-bit physical byte address.

What is the total size of the page table for each process on this processor, assuming that the valid, protection, dirty, and use bits take a total of 4 bits and all the virtual pages are in use?

~~block, and words of the block are transferred 1 at a time.  
Miss penalty =  $1 + 4 \times 10 + 16 \times 1 = 57$  clock cycles~~

~~7.39~~ The total size is equal to the number of entries times the size of each entry.  
Each page is 16 KB, and thus, 14 bits of the virtual and physical address will be used as a page offset. The remaining  $40 - 14 = 26$  bits of the virtual address constitute the virtual page number, and there are thus  $2^{26}$  entries in the page table, one for each virtual page number. Each entry requires  $36 - 14 = 22$  bits to store the physical page number and an additional 4 bits for the valid, protection, dirty, and use bits. We round the 26 bits up to a full word per entry, so this gives us a total size of  $2^{26} \times 32$  bits or 256 MB.

**6.2.2**

<b>a.</b>	0.996168582375479
<b>b.</b>	0.998628257887517

**6.2.3** Availability approaches 1.0. With the emergence of inexpensive drives, having a nearly 0 replacement time *for hardware* is quite feasible. However, replacing file systems and other data can take significant time. Although a drive manufacturer will not include this time in their statistics, it is certainly a part of replacing a disk.

**6.2.4** MTTR becomes the dominant factor in determining availability. However, availability would be quite high if MTTF also grew measurably. If MTTF is 1000 times MTTR, the specific value of MTTR is not significant.

**Solution 6.3****6.3.1**

<b>a.</b>	15.196 ms
<b>b.</b>	13.2 ms

**6.3.2**

<b>a.</b>	15.225
<b>b.</b>	13.233

**6.3.3** The dominant factor for all disks seems to be the average seek time, although RPM would make a significant contribution as well. Interestingly, by doubling the block size, the RW time changes very little. Thus, block size does not seem to be critical.

**Solution 6.4****6.4.1**

<b>a.</b>	Yes	A satellite database will process infrequent requests for bulk information. Thus, increasing the sector size will allow more data per read request.
<b>b.</b>	No	This depends substantially on which aspect of the video game is being discussed. However, response time is critical to gaming. Increasing sector size may reduce response time.

**6.4.2**

<b>a.</b>	Yes	Increasing rotational speed will allow more data to be retrieved faster. For bulk data, this should improve performance.
<b>b.</b>	No	Increasing rotational speed will allow improved performance when retrieving graphical elements from disk.

**6.4.3**

<b>a.</b>	No	A database system that is collecting data must have exceptionally high availability, or data loss is possible.
<b>b.</b>	Yes	Increasing disk performance in a non-critical application such as this may have benefits.

**Solution 6.5**

**6.5.1** There is no penalty for either seek time or for the disk rotating into position to access memory. In effect, if data transfer time remains constant, performance should increase. What is interesting is that disk data transfer rates have always outpaced improvements with disk alternatives. Flash is the first technology with potential to catch hard disk.

**6.5.2**

<b>a.</b>	No	Databases are huge and Flash is expensive. The performance gain is not worth the expense.
<b>b.</b>	Yes	Anything that improves performance is of benefit to gaming.

**6.5.3**

<b>a.</b>	Maybe	Decreasing download time is highly beneficial to database downloads. However, the data rate for some satellites may be so low that no gain would result.
<b>b.</b>	Yes	

**Solution 6.6**

**6.6.1** Note that some of the specified Flash memories are controller limited. This is to convince you to think about the system rather than simply the Flash memory.

<b>a.</b>	28.7 ms
<b>b.</b>	32.5 ms