CS 250 Spring 2017 Homework 08 SOLUTION
Due 11:58pm Wednesday, March 29, 2017
Submit your typewritten file in PDF format to Blackboard.

1. Let store indirect be defined as

```
STOREI (regX, offset, regY)      # store the contents of regX into the memory
                                 # location pointed to by the address found in
                                 # memory at location offset+regY
```

Devise a way to implement STOREI as a macro written in the instruction set architecture (ISA) of Figure 6.1. For full credit, your implementation of STOREI may use no more storage locations and no different storage locations that does a single store instruction and must be expressed in the form of a macro.

**STOREI(regX, offset, regY) in macro code:**

```
load    regY, offset(regY);
store   regX, regY;
```

2. Assume that we make an enhancement to a computer that improves some mode of execution by a factor of 10. Enhanced mode is used 50% of the time, measured as a percentage of the execution time *when the enhanced mode is in use*. Recall that Amdahl's Law depends on the fraction of the original, *unenhanced* execution time that could make use of the enhanced mode. Thus, we cannot directly use this 50% measurement to compute speedup with Amdahl's Law.
    1. What is the speedup we have obtained from fast mode?

    Speedup = Original Time / Enhanced Time = 550 / 110 = 5.5

    **The speedup we have obtained from the fast code is 5.5 times faster.**

    2. What percentage of the original execution time has been converted to fast mode?
    Original time converted into fast mode: 500 seconds
    Total original time: 550 seconds

    (500 / 550) = 0.909090909 = **90.91%**

3. Assume that a computer has a physical memory organized into 64-bit words. Give the word address and offset within the word for each of the following byte addresses: 0, 9, 27, 31, 120, and 256.

| Byte Address | Word Address | Offset |
|---|---|---|
| **0** | 0 | 0 |
| **9** | 1 | 1 |
| **27** | 3 | 3 |
| **31** | 3 | 7 |
| **120** | 15 | 0 |
| **256** | 32 | 0 |

4. Consider the following code snippet and Figure 11.6 from our text.

```
for (i = 0; i <= 1023; i = i + j) {
        array[i] =  array[i] + 7;
}
```

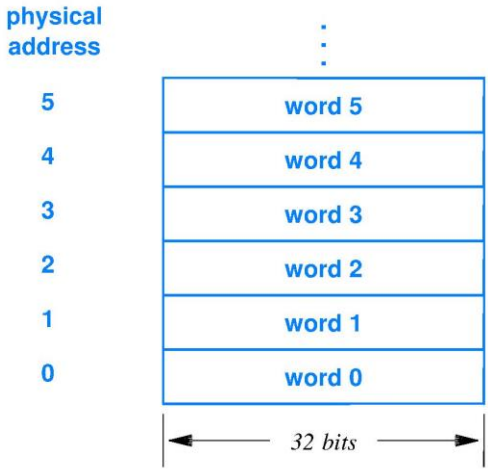


**Figure 11.6** Physical memory addressing on a computer where a word is thirty-two bits. We think of the memory as an array of words.

Assume that the Figure 11.6 memory is built from a single memory chip that can be read during one CPU clock cycle but then requires 3 more CPU clock cycles before the chip is again ready to be accessed by the CPU. Assume that the constant 7 in the code snippet and the current values of variables i and j are available in CPU registers at all times. Finally, assume that the CPU stalls until array[i] can be read from data memory.

1. Describe the stalls that the CPU will experience due only to data memory read access of elements of array[i] as a function of the value of j for 1 <= j <= 1024. Ignore data memory write accesses; ignore all instruction fetch stalls (such as one due to a control hazard). With respect to execution time, what are the best values of j and the worst values?
   The stalls that the CPU will experience will be during each iteration of the loop when the array[i] is accessed. Given the above given system implications I believe it will stall for 3 whole clock cyclers per iteration. I believe that there is no true optimal value for j because the whole system, regardless of j, will cause the stall on access. Therefore there is no ‘best’ value for j as the ‘worst’ values are essentially all the numbers.

2. Now assume all is the same as in part (a) of this question except that the memory system configuration is now as shown in Figure 11.13.

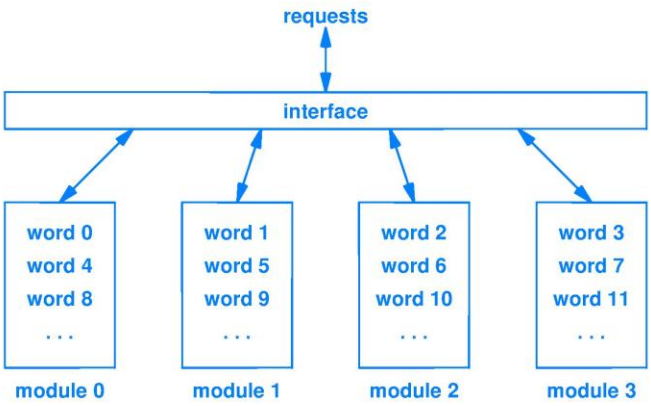


**Figure 11.13** Illustration of 4-way interleaving that illustrates how successive words of memory are placed into memory modules to optimize performance.

Describe the stall cycles that the CPU will experience due only to data memory read access of elements of array[i] as a function of the value of j for 1 <= j <= 1024. Ignore data memory write accesses; ignore all instruction fetch stalls, such as one due to a control hazard. With respect to execution time what are the best values of j and the worst values?

In this system I believe the given code loop would have no stalls per iteration if j=1 (thus making optimal j value being 1) due to the 4 modules of memory that can handle the access of array[i] each time and the 3 stalls will be during the other executions of the other iterations beginnings thus pipelining the operation and removing stalls. I believe all other values for j would be worse values, as it wouldn't take advantage of the 4 module architecture. A special case if j=2 it will skip spaces in memory semi-efficiently and not completely reduce stalls (though still faster than other values of j, it is still NOT optimal). Therefore the best value for j is simply 1 and the worst values would be 2 and 4 and then any other value after that.

5. The CPU time equation is as follows, CPI means Clock cycles per instruction: CPU Time = (Instructions/Program) * (CPI) * (Seconds/Clock cycle). For each of the three factors in the CPU Time equation, answer the following questions: (1) Can loop unrolling ALONE improve this factor, worsen this factor, or cannot affect this factor? (2) If loop unrolling either improves or worsens the factor, how does this occur?

1. **Instructions:** Loop unrolling can either improve or worsen the instructions.
   **CPI:** Loop unrolling improves CPI (Clock Cyclers per Instruction).
   **Clock Cycle:** Loop unrolling worsens the clock cycle.

2. **Instructions:** Loop unrolling acts to increase a program's speed by reducing and/or eliminating instructions that control the loop. Unrolling very large loops can lead to rather large sizes of code, therefore loop unrolling depends on the code here.
   **CPI:** Loop unrolling improves the instruction-level parallelism thus it can improve CPI.
   **Clock Cycle:** Loop unrolling increases the area of the area of the program and thus can have a negative impact on clock cycle.

6. Once a loop has been unrolled, which factor(s) of the CPU Time equation can be improved, worsened, or cannot be affected through the application of instruction scheduling? Explain.
   Instruction scheduling improves instruction-level parallelism so therefore it improves the Clock Cycles per Instruction (CPI). By breaking down operations and scheduling the instructions you can organize and reorder instructions in order to optimize the CPI.