CS 271 Computer Architecture and Assembly Language

Self-Check for Lecture#19

Solutions

1. Given a CISC machine with a 2 GHz clock (i.e., the clock ticks 2 billion times per second). This particular computer uses MASM-like instructions with the following timings:

add	*reg, mem*	9 clock cycles (i.e., the ADD microprogram has 9 instructions)
add	*reg,immed*	3 clock cycles
loop	*label*	7 clock cycles

Here's a short code fragment to sum the elements of a numeric array:

```
     mov  eax,0                ;initialize sum
     mov  ecx,MAX_SIZE         ;initialize loop counter
     mov  esi,OFFSET list      ;initialize array pointer
more:
     add  eax,[esi]            ;add current list element
     add  esi,4                ;move array pointer to next element
     loop more                 ;auto-decrement ecx, jump to more,
                               ;    if ecx ≠ 0
```

Assume unlimited array size. After initialization, how many array elements could be processed in 1 ms. (1 ms. = 1/1000 sec).

105263 elements

A 2 GHz CISC machine can execute 2 **million** (with an M) micro-instructions per **milli-second**. The instructions that comprise the loop require 9+3+7 = 19 micro-instructions to process 1 element of the array. So the loop can process 2,000,000 / 19 = 105,263elements in 1 ms.

2. Given a RISC machine with a 2 GHz clock (i.e., the clock ticks 2 billion times per second). This particular computer uses an instruction cache, a data cache, an operand fetch unit, and an operand store unit. The instruction set includes simple instructions with the following timings:

set	*reg, immed*	1 clock cycle
load	*reg,mem*	2 clock cycles
add	*reg,reg*	2 clock cycles
add	*reg,immed*	1 clock cycle
loop	*label*	3 clock cycles

Here's a short code fragment to sum the elements of a numeric array:

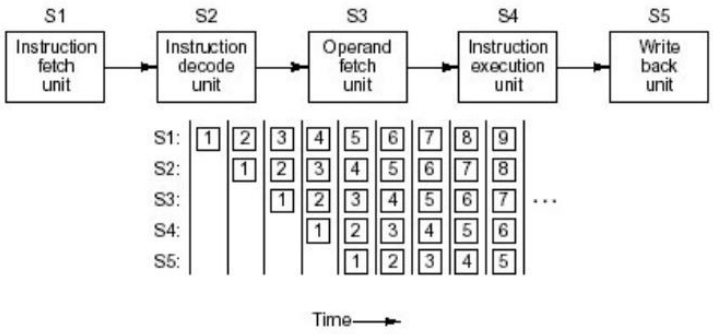
```
      set   r1,0                ;initialize sum
      set   r2,MAX_SIZE         ;initialize loop counter
      set   r3,@list            ;initialize array pointer
more:
      load r4,[r3]              ;fetch current list element
      add   r1,r4               ;add current list element
      add   r3,4                ;move array pointer to next element
      loop more                 ;auto-decrement r2, jump to more,
                                ;     if r2 ≠ 0
```

Assume unlimited array size. After initialization, how many array elements could be processed in 1 ms. (1 ms. = 1/1000 sec).

250000 elements

A 2 GHz CISC machine can execute 2 **million** (with an M) micro-instructions per **milli-second**. The instructions that comprise the loop require 2+2+1+3 = 8micro-instructions to process 1 element of the array. So the loop can process 2,000,000 / 8 = 250,000elements in 1 ms.

3. Given a five-stage pipeline as illustrated at the right: Suppose that each stage requires 3 nanoseconds (ns) to complete its task.



   a. How long will it take to complete 100 instructions with pipelining?

      ___312___ ns

      It takes 100 instructions 300 ns. to complete stage S1, i.e., instruction #100 completes stage S1 at 300 ns. Therefore, instruction #100 will complete stage S2 at 303 ns., stage S3 at 306 ns., stage S4 at 309 ns, and stage S5 at 312 ns.

   b. How long will it take to complete 100 instructions without pipelining?

      ___1500___ ns

      Each instruction takes 15 ns to complete stages S1 through S5. Without pipelining, each instruction must complete all five stages before the next instruction can start into stage S1. Therefore, 100 instructions will take 1500 ns.

4. An algorithm takes 4seconds to execute on a single 2.4G processor. 30% of the algorithm is sequential. Assuming zero latency and perfect parallelism in the remaining code, how long should the algorithm take on a parallel machine with 8 2.4G processors?

__approximately 1.55__ sec

$n = 8,\ f = 0.3,\ T = 4$

$$speedup = \frac{n}{1+(n-1)f} = \frac{8}{1+.3x7} = \frac{8}{3.1} \approx 2.58$$

$$expectedTime = \frac{T}{speedup} \approx \frac{4}{2.58} \approx 1.55$$

So the expected time is 1.55 seconds

5. Cite and explain two major reasons that software parallelism has not kept pace with developments in hardware parallelism.

It's relatively easy to add more processors to a hardware system, but it is very difficult to create generalized software systems that can

a. detect if an algorithm is parallelizable
b. divide a process into parts that can execute simultaneously
c. synchronize the results of parallel execution
d. maintain the necessary communication among the parallel parts of an algorithm

Also, more research and development is needed in

a. high-level languages and debugging systems for parallel programming
b. libraries of functions and procedures for parallel programming
c. operating systems that can utilize the features of parallel hardware