Solutions exam 2007-12-20

By Thomas L, 2008-12-10

Assignment 1

A) The commonly used way of comparing performance is to first normalize the execution times relative to a reference machine. Then, the geometric mean of the relative performance figures is calculated. In our case, let us choose Computer A as our reference. We then get the relative performance for computer X as:

$$Relative\ performance_X = \frac{1}{Relative\ execution time_X} = \frac{Time_A}{Time_X}$$

	Relative Performance		
Benchmark	**Computer A**	**Computer B**	**Computer C**
B1	1	1.11	0.91
B2	1	1.25	1.11
B3	1	0.91	0.91
Geometric mean	1	1.08	0.97

The winning machine is therefore Computer B with the highest average relative performance of 1.08.

B) This is an example of a simple queueing system with a single server servicing request from a queue. Requests arrive randomly according to a poisson process (exponential distribution of the intervals between requests). We can use the following formula from the book, Section 6.5:

$$Time_{response} = Time_{queue} + Time_{server} = (1 + \frac{Server\ utilization}{1 - Server\ utilization}) \times Time_{server}$$

where:

$$Server\ utilization = Arrival\ rate \times Time_{server}$$

In our case we have from the problem description:

$$Arrival\ rate = 10\ req/sec$$
$$Time_{server} = 50\ ms$$

which give us:

$$Server\ utilization = 10 \times 0.05 = 0.5$$

$$Time_{response} = (1 + \frac{0.5}{1 - 0.5}) \times 50 = 100\ ms$$

Thus, a request spends on average 50 ms in the queue and 50 ms being processed.

C) Here we need to look at different contributions to the average CPI. First, we should break down the execution time into enough detail (refer to the book, App C):

$$CPU\,time = IC \times CPI_{average} \times T_c$$

$$CPI_{average} = CPI_{execution} + CPI_{memory\,stalls}$$

$$CPI_{memory\,stalls} = Miss\,rate \times Memory\,accesses\,per\,instruction \times Miss\,penalty$$

For our problem, the instruction count, IC, and clock cycle time, T_c, remains fixed so we can use the average CPI as equivalent to the total execution time. Also, $CPI_{execution} = 1.5$ since for perfect caches we get no contribution to the CPI from any memory stalls. We also know that the number of memory accesses per instruction is 0.45. We get:

$$CPI_{average} = 1.5 + Miss\,rate \times 0.45 \times 20$$

We will now compare two different cases. First, when our miss rate is 10% and then when the miss rate is 0% since that represents the best speedup possible.

$$Speedup = \frac{CPI^{old}_{average}}{CPI^{new}_{average}} = \frac{1.5 + 0.1 \times 0.45 \times 20}{1.5 + 0 \times 0.45 \times 20} = \frac{1.5 + 0.9}{1.5} = 1.6$$

So the best possible speedup is 1.6 or 60%.

Assignment 2

A) The code has data dependencies due to the use of earlier written registers. These might cause data hazards in the pipeline which we resolve by forwarding or stalling. We might also have control hazards due to the branch in the program. A table showing where instructions are in the pipeline in each clock cycle shows how many cycles one iteration of the loop takes.

```
LOOP: LD    R4, 0(R1)
      DMUL  R5, R4, R4      # RAW dependency R4
      SD    R5, 0(R1)       # RAW dependency R5
      DADDI R1, R1, 8
      BNE   R1, R2, LOOP    # RAW dependency R1
      NOP
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
LD	IF	ID	EXE	MEM	WB												
DMUL		IF	ID	stall	EXE	MEM	WB										
SD			IF	stall	ID	stall	EXE	MEM	WB								
DADDI					IF	stall	ID	EXE	MEM	WB							
BNE							IF	ID	stall	EXE	MEM	WB					
NOP								IF	stall	ID	EXE	MEM	WB				
LD										IF	ID	EXE	MEM	WB			

We find that one iteration will take 9 cycles. The data hazards cause one stall cycle each.

B) To avoid the stall cycles in A we can rearrange the code to try to make dependent instruction become more separated. We also put something more useful than the NOP in the delay slot:

```
LOOP: LD    R4, 0(R1)
      DADDI R1, R1, 8
      DMUL  R5, R4, R4
      BNE   R1, R2, LOOP
      SD    R5, -8(R1)     # compensate for the already incremented R1
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
LD	IF	ID	EXE	MEM	WB												
DADDI		IF	ID	EXE	MEM	WB											
DMUL			IF	ID	EXE	MEM	WB										
BNE				IF	ID	EXE	MEM	WB									
SD					IF	ID	EXE	MEM	WB								
LD						IF	ID	EXE	MEM	WB							

This completely avoids the stall cycles and one iteration now takes 5 cycles.

C) For even n, we can safely unroll the loop once and make each iteration do the work of two previous iterations. We then get two DADDI which we merge. We also rearrange further to avoid stall cycles:

```
LOOP: LD    R4, 0(R1)
      DADDI R1, R1, 16    # bump index 8 x 2 = 16
      DMUL  R5, R4, R4
      LD    R4, -8(R1)    # start second loop body before first is done
      SD    R5, -16(R1)
      DMUL  R5, R4, R4
      BNE   R1, R2, LOOP
      SD    R5, -8(R1)
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
LD	IF	ID	EXE	MEM	WB												
DADDI		IF	ID	EXE	MEM	WB											
DMUL			IF	ID	EXE	MEM	WB										
LD				IF	ID	EXE	MEM	WB									
SD					IF	ID	EXE	MEM	WB								
DMUL						IF	ID	EXE	MEM	WB							
BNE							IF	ID	EXE	MEM	WB						
SD								IF	ID	EXE	MEM	WB					
LD									IF	ID	EXE	MEM	WB				

Again, it is possible to completely avoid stall cycles and the instructions corresponding to the old loop body now takes 4 cycles (onenew iteration takes 8 cycles).

Assignment 3

A) See lecture slides

B) Exceptions are handled by not recognizing the exception until it is ready to commit.

Assignment 4

Refer to Appendix C and Chapter 5 in the book.

Assignment 5

See Chapter 3 and 4 in the book.