INDIAN STATISTICAL INSTITUTE

Periodical Examination

M. Tech (CS) - I Year (Semester - II)

Computer Architecture

Date: 26.02.2009 Maximum Marks: 40 Duration: 2.5 Hours

Note: Be precise in your answers. This is a three page question paper.

Q 1(i): A computer system contains a special purpose processor for doing floating-point operations. You as a designer have determined that 70% of your computations can use the floating-point processor. When a program uses the floating-point processor, the speedup of the floating-point processor is 45% faster than when it does not use it. Find the overall speedup by using the floating-point processor.

Ans 1(i): Here, fraction enhanced $f_{en} = 0.7$ and speedup enhanced $s_{en} = 1.45$. So, overall speedup $= \frac{1}{(1-0.7)+\frac{0.7}{1.45}} = 1.2775$.

Q 1(ii): In order to improve the speedup, you are considering two options:

Option 1: The compiler design is modified so that 80% of the computations can use the floating-point processor. Cost of this option is Rs. 25 lakhs.

Option 2: The floating-point processor is to be modified. The speedup of the floating-point processor is 100% faster than when it does not use it. Assume in this case that 60% of the computations can use the floating point processor. Cost of this option is Rs. 30 lakhs.

Which option would you recommend? Justify your answer quantitatively. [2+4=6]

Ans 1(ii): The relative figures for the two options are:

Option 1: Here $f_{en} = 0.8$ and $s_{en} = 1.45$. So, overall speedup is equal to $\frac{1}{(1-0.8)+\frac{0.8}{1.45}} = 1.33$.

The cost to speedup ratio is $\frac{25 \text{ lakhs}}{1.33} = 18.7969$ lakhs .

Option 2: Here $f_{en} = 0.6$ and $s_{en} = 2.0$. So, overall speedup is equal to $\frac{1}{(1-0.6)+\frac{0.6}{2.0}} = 1.4286$.

The cost to speedup ratio is $\frac{30 \text{ lakhs}}{1.4286} = 20.9995$ lakhs .

As the cost to speedup ratio is smaller for Option 1, we would choose Option 1.

Q 2: Suppose we make an enhancement to a computer that improves a mode of execution by a factor of 15. The enhanced mode is used 55% of the time measured as a percentage of the execution time when the enhanced mode is in use. Find out (a) what percentage of the original execution time has been converted to fast mode? (b) what is the speedup we have obtained from fast mode? [2+4=6]

Ans 2: Let the old execution time be t and the new execution time be t'. Let t_1 be the time span in t for which the enhanceement was used. Thus, $f_{en} = \frac{t_1}{t}$. Because of the use of enhancement, let t_1 of the old execution now finish in time t_1' in the new execution. Thus, $s_{en} = \frac{t_1}{t_1'}$.

In this problem, what we have been given instead is a new fraction $f' = \frac{t_1'}{t'}$ for the enhanced mode. We have to deduce speedup in terms of f' and s_{en}. The new execution time t' is composed of the enhanced time t_1' and the unenhanced time $(t - t_1)$.

$$\begin{aligned} t' &= t_1' + (t - t_1) \\ &= f't' + (t - t_1) \\ t'(1 - f') &= t - s_{en}t_1' \\ &= t - s_{en}t'f' \\ t'(1 - f' + s_{en}f') &= t \\ \frac{t}{t'} &= 1 + f'(s_{en} - 1) \end{aligned}$$

So, speedup is $\frac{t}{t'} = 1 + f'(s_{en} - 1)$. Here, $s_{en} = 15$ and $f' = 0.55$. So, speedup is equal to $1 + 0.55(15 - 1) = 8.7$. We know, speedup is equal to $\frac{1}{(1-f_{en})+\frac{f_{en}}{s_{en}}}$. Plugging in the values of speedup (8.7) and $s_{en} = 15$, we have $f_{en} = 0.948$.

Q 3 (i): Describe in short the concept of memory hierarchy explaining the role of each level of memory.

Ans 3(i): This was discussed in class.

Q 3 (ii): The Clock cycles per instruction (CPI) of a computer system is 3.0 when all memory accesses hit in the cache. The only data accesses are *loads* and *stores* and these total 53% of the instructions. If the miss penalty is 31 clock cycles and the miss rate is 3%, how much faster would the machine be if all instructions were cache hits? [4+2=6]

Ans 3(ii): CPU execution time for the machine that always hits =

(CPU clock cycles + Memory stall cycles) $\times$ clock cycle(CC) $= (IC \times CPI + 0) \times CC$

$= IC \times 3.0 \times CC.$

Memory stall cycles for the machine with the real cache =

$IC \times$ memory references per instruction $\times$ miss rate $\times$ miss penalty

$= IC \times \underbrace{(1 + 0.53)}_{\text{1 instruction access and 0.53 data accesses per instruction}} \times 0.03 \times 31$

$= IC \times 1.4229.$

CPU execution time in cache

$= (IC \times 3.0 + IC \times 1.4229) \times CC$

$= 4.4229 \times IC \times CC.$

Performance ratio (inversion of execution times)

$= \frac{\text{CPU execution time with cache}}{\text{CPU execution time without cache}}$

$= \frac{4.4229 \times IC \times CC}{3.0 \times IC \times CC}$

$= 1.4743.$

Q 4 Consider the following piece of 'C' code.

```
for (i=0; i<= 100; i++)
  {X[i] = Y[i] + Z;}
```

Assume that X and Y are arrays of 32-bit integers and Z and i are 32-bit integers. Assume that all data values and their addresses are kept in memory at addresses 0, 5000, 1500 and 2000 for X, Y, Z and i respectively except when they are operated on. Assume that values in registers are lost between iterations of the loop.

(a) Write the code for DLX. (b) How many memory-data references will be executed? (c) What is the code size in bytes? [8+2+1=11]

Ans 4: Note the following two sentences in the question. *Assume that all data values and their addresses are kept in memory at addresses* 0, 5000, 1500 *and* 2000 *for X, Y, Z and i respectively except when they are operated on. Assume that values in registers are lost between iterations of the loop.*

Because of this assumption, registers are not used to hold updated or intermediate values. Values are stored to memory and reloaded when needed. Also, as all addresses fit inside 16 bits, we use *immediate* instructions.

```
        ADD     R1, R0, R0      ;R1 will store 'i'; initialize it to zero
        SW      2000(R0), R1    ;store 'i'
loop:   LW      R1, 2000(R0)    ;get value of 'i'
        SLL     R2, R1, #2      ;making the offset for Y
        ADDI    R3, R2, #5000   ;add base address and the offset for Y
        LW      R4, 0(R3)       ;load Y[i]
        LW      R5, 1500(R0)    ;load Z
        ADD     R6, R4, R5      ;Y[i]+Z
        LW      R1, 2000(R0)    ;again get value of 'i' as
                                ;it cannot be stored in register
        SLL     R2, R1, #2      ;making the offset for X
        ADDI    R7, R2, #0      ;add base address and the offset for X
        SW      0(R7), R6       ;X[i]=Y[i]+Z
        LW      R1, 2000(R0)    ;get value of 'i'
        ADDI    R1, R1, #1      ;increment 'i'
        SW      2000(R0), R1    ;store 'i'
        LW      R1, 2000(R0)    ;get value of 'i'
        ADDI    R8, R1, #-101   ;is counter at 101?
        BNEZ    R8, loop        ;loop instruction
```

Instructions executed (though not asked for) is the number of instructions for initialization instructions, plus the number of instructions in the loop times the number of iterations and is equal to $2 + (16 \times 101) = 1618$.

The number of memory-data references executed is $= 1 + (8 \times 101) = 809$.

The DLX instruction is 4 bytes wide, so the code size is $18 \times 4 = 72$.

Q 5: Show how the code sequence $A \times B - (A + C \times B)$ will appear on the following architectures: (a) stack, (b) accumulator, (c) register-memory, and (d) load-store. [1.5+1.5+1.5+1.5=6]

Ans 5: The code sequence for $A \times B - (A + C \times B)$ in the following architectures are shown below:

Q 6(i): Explain the effect of instruction pipelining on the bandwidth of the memory systems.

Ans 6(i): This was discussed in the class.

Stack	**Accumulator**	**Register-Memory**	**Load-Store**
push A;	load B;	load R1, A;	load R1, A;
push B;	mul C;	mul R1, B;	load R2, B;
mul;	add A;	store R1, D;	load R3, C;
push A;	store D;	load R2, C;	mul R4, R3, R2;
push C;	load A;	mul R2, B;	add R5, R1, R4;
push B;	mul B;	add R2, A;	mul R6, R1, R2;
mul;	sub D;	sub R2, D;	sub R7, R6, R5;
add;			
sub;			

Q 6(ii): Consider an unpipelined machine with five stages (Instruction Fetch, Instruction Decode/Register Fetch, Execute/Address Calculation, Memory Access and Write Back). Assume that it has 11-ns clock cycles. The machine uses four cycles for ALU operations and branches and five cycles for memory operations. Assume that the relative frequencies of these operations are 45%, 15% and 40% respectively. Pipelining the machine adds 1-ns of overhead to the clock. Find out how much speedup we will gain in the instruction execution rate. You can ignore any latency impact. [3+2=5]

Ans 6(ii): The average instruction execution time on the unpipelined machine is

$$\begin{aligned} \text{Average instruction execution time} &= \text{Clock Cycle}(CC) \times \text{Average CPI} \\ &= 11\text{ns} \times ((0.45 + 0.15) \times 4 + 0.4 \times 5) \\ &= 11\text{ns} \times 4.4 \\ &= 48.4\text{ns} \end{aligned}$$

In the pipelined version, the clock will run at $11 + 1 = 12$ ns. So, the average instruction execution time is 12 ns. So, the speedup from pipelining is

$$= \frac{\text{ave. instruction time in unpipelined version}}{\text{ave. instruction time in pipelined version}}$$

$$= \frac{48.4 \text{ ns}}{12 \text{ ns}} = 4.033$$