Solutions exam 2008-12-18

By Thomas L, 2008-12-29

Assignment 1

A) Let us call the first design: basic, and the one with higher clock frequency: high. Then, the speedup can be expressed as:

$$Speedup = \frac{Performance_{high}}{Performance_{basic}} = \frac{\text{CPU Time}_{basic}}{\text{CPU Time}_{high}}$$

Where CPU Time is the execution time of the program which can be expressed as:

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

Here, the average CPI and T_C will differ between the designs. The number of instructions IC will be the same in both cases. We get:

$$Speedup = \frac{CPI^{basic}_{average} \times T^{basic}_{c}}{CPI^{high}_{average} \times T^{high}_{c}} = \frac{CPI^{basic}_{average}}{CPI^{high}_{average}} \times 1.20$$

Since the clock cycle time of the basic design is 20% longer than in the high frequency design. To calculate the speedup we now need to compute the average CPI of both design options. To do this, we only consider the loop (a slight approximation) and compute the clock cycles needed for the execution of one iteration in the loop divided by the number of instructions executed for one iteration.

The average CPI can be split into:

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

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

The latencies for each instruction in the loop are:

Latency of each instruction	basic	high
LD	1	1
DMUL	2	3
DADD	1	1
SD	1	1
DADDI	1	1
DADDI	1	1
BNE	2	2
SUM	**9**	**10**
CPI execution	**9/7**	**10/7**

The miss penalty for the basic design is given as 10 cycles. For the high frequency design, the miss penalty will increase in proportion to the frequency increase since the absolute miss penalty time stays the same. We therefore get 12 cycles miss penalty in the high frequency design.

It is time to put everything together:

$$CPI_{average}^{basic} = 9/7 + 0.125 \times 2/7 \times 10$$

$$CPI_{average}^{high} = 10/7 + 0.125 \times 2/7 \times 12$$

$$Speedup = \frac{CPI_{average}^{basic}}{CPI_{average}^{high}} \times 1.20 = \frac{11.5}{13.0} \times 1.2 = 1.062$$

Thus, we get a speedup of 6.2%.

NOTE: An additional valid assumption would be to insert a stall cycle between the LD and the DMUL due to the data dependency if you assumea normal 5-stage pipeline with forwarding. So even if the text says "No other stalls occur", adding one more cycle to the loop forboth designs would be ok as well. The speedup would be slightly higher (7.1% speedup) in that case.

B) Amdahl's law states that:

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

So it gives the overall speedup when speeding up a fraction of the original execution time.

If we halve the miss penalty in the high freqency design, we speed up a fraction of the execution time by a factor of 2. But, to compute the fraction we need to look at the original CPU Time formula:

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

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

For the high frequency design we had:

$$CPI_{average}^{high} = 10/7 + 0.125 \times 2/7 \times 12$$

So, the fraction we speed up is the second term(the miss penalty term):

$$Fraction_{enhanced} = \frac{0.125 \times 2/7 \times 12}{10/7 + 0.125 \times 2/7 \times 12} = \frac{1.5}{10 + 1.5} = 0.13$$

Using Amdahl's law we get the additional speedup as:

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}} = \frac{1}{(1 - 0.13) + \frac{0.13}{2}} = 1.07$$

Thus, we get an additional 7% speedup fromhalving the miss penalty in the high frequency design.

C) The yield is the fraction of good dies (chips) on a wafer. It depends on the complexity and maturity of the manufacturing process (defectsper unit area) and the die area. Bigger chip area means lower yield. The cost of a chip goes up when the yield goes down. Thus, a bigger chip area means bigger cost per chip due to a decreased yield.

D) The trade-off here comes from looking at the dynamic power use. For CMOS chips, the dynamic power use is:

$$Power_{dynamic} = 1/2 \times Capacitive\ load \times Voltage^2 \times Frequency\ switched$$

Running a task on two cores with lower clock frequency can be more power efficient than using only one core with double the frequency. The reason is that using a lower clock frequency can permit the use of a lower voltage and thereby the use of less power (see formula above).

Assignment 2

A) The different types of dependencies together with correponding examples are:

Type	Example
Data dependencies	An instruction uses a value produced by an earlier instruction. For example, the DMUL reads register R4 which is written by the preceding LD instruction.
Name dependencies	Two instructions use the same name (register) but there is no data flow between the instructions associated with that name. For example, both the DMUL and the DADD write to register R5(output dependence) but the second R5 could be renamed to something else without changing the data flow between these instructions.
Control dependencies	The execution of an instruction depends on the outcome of previous branch instructions. For example, the execution of the loop body is control dependent on the BNE instruction in all loop iterations except the first.

B) Data hazards can occur in pipelines due to data dependencies in the executing program. Different types of data hazards:

Type	Explanation / Example
RAW	Read After Write. An instruction i tries to read a source before a preceding instruction writes it. Instruction i might read an old value.
WAR	Write After Read. An instruction writes a new value before a preceding instruction i has finished reading it. The instruction i might get the new value.
WAW	Write After Write. An instruction writes a new value before a preceding instruction writes an old value. The old value might be the one remaining in the destination.

C) Rescheduling is done in order to move dependent instructions further apart (to reduce the risk of hazards forcing the pipeline to stall). Loop unrolling duplicates the loop body in order to make the rescheduling easier and to make the code more efficient by eliminating some instructions. Register renaming is often needed to make the unrolling work. The program with the loop body unrolled one time (one copy):

```
   ANDI  R3, R3, 0      # R3 = 0
LOOP:
   LD    R4, 0(R1)
   DMUL  R5, R4, R4
   DADD  R5, R3, R5
   SD    R5, 0(R1)      # new[i] = old[i-1] + old[i]*old[i]
   DADDI R3, R4, 0
   DADDI R1, R1, 8
   LD    R4, 0(R1)
   DMUL  R5, R4, R4
   DADD  R5, R3, R5
   SD    R5, 0(R1)      # new[i] = old[i-1] + old[i]*old[i]
   DADDI R3, R4, 0
   DADDI R1, R1, 8
   BNE   R1, R2, LOOP
```

Next steps is to merge the DADDI instructions, rename registers, and reschedule instructions. This is **one example** of how the program can be modified:

```
   ANDI  R3, R3, 0      # R3 = 0
LOOP:
   LD    R10, 0(R1)
   LD    R14, 8(R1)
   DADDI R1, R1, 16
   DADDI R13, R10, 0
   DMUL  R11, R10, R10
   DMUL  R15, R14, R14
   DADD  R12, R3, R11
   DADD  R16, R13, R15
   DADDI R3, R14, 0
   SD    R12, -16(R1)    # new[i] = old[i-1] + old[i]*old[i]
   SD    R16, -8(R1)     # new[i] = old[i-1] + old[i]*old[i]
   BNE   R1, R2, LOOP
```

Here, each destination register is renamed to a new register number except for the loop carried dependencies: R1 and R3. This permits mixing instructions from both loop bodies while making sure not to break any true data dependency. The general rescheduling principle is to try to put independent instructions between the writes and the later reads to a register to avoid stalls due to RAW hazards. There are many ways to do this so the important thing in this answer is to point out the principles.

Assignment 3

A) See book pages 82-83 and Figure 2.4. For our example program we would have one entry corresponding to the BNE instruction in the end of the program. The prediction would evolve according to this table if we assume we start in state 00:

Execution of the BNE instuction	Prediction state before execution	Prediction	State after execution
First	00	Not taken (wrong)	01 (it was taken)
Second	01	Not taken (wrong)	11 (it was taken)
Third	11	Taken (correct)	11 (it was taken)
4th, 5th, ...	11	Taken (correct)	11 (it was taken)
1000th	11	Taken (wrong)	10 (was not taken)

For a program with two branches, we would get a 2-bit state entry for each branch as long as they do not share the same index in the branch-prediction buffer. Branches with different index (entries) are not correlated with each other.

B) The key point for supporting speculation is to commit architectural visible changes in program order. It must be possible to cancel (flush) speculative results without any remaining visible changes in the registers or in the memory. Registers and memory should only be written in the commit stage. Before that, the reorder buffer holds the speculative (temporary) results.

C) The corresponding steps for a store instruction is:

Store Processing

1. Issue when reservation station and ROB entry is available
   - Read already available operands from registers and instruction
   - Tag unavailable operands with ROB entry
   - Mark ROB entry as busy
2. Execute after issue
   - Wait for operand values on CDB (if not already available)
   - Compute address and store it in ROB entry
3. Write result when CDB and ROB available
   - Update ROB entry with source register value, and mark as ready
   - Free reservation station
4. Commit when at head of ROB and ready
   - Write result (source register value) to memory at computed address
   - Free ROB entry

The important points are: (1) The actual write is done at the commit stage, (2) To carry out the write, the address needs to be computed (execute stage), and, (3) the source operand (the value to write) is needed before commit is possible.

D) RAW hazards are avoided by delaying instruction execution until the source operands are available. Instructions wait in a reservation station until source operands are available. Earlier instructions that write dependent values send their results directly to the waiting reservation stations.

Assignment 4

A) The advantage is a reduced hit time since the indexing in the cache does not need to wait for the virtual to physical address translation to finish. Disadvantages: (1) Page-level protection needs to be enforced even when no translation and lookup in the TLB is done, (2) The translation is process dependent. When processes are switched, the cache must maybe be flushed. (3) Alias problems might cause same data to appear in different cache locations leading to inconsistencies.

B) To reduce miss-rate, some options are: (1) Larger block size (compulsory),(2) Bigger cache (capacity), (3) Higher associativity (conflict), (4) Compiler techniques to reduce memory accesses and cache misses (all miss types), (5) Hardware prefetching (all miss types),and (6) Software prefetching (all miss types).

C) The virtual memory system can cause problems since a guest OS should not be allowed to modify page tables directly. Thus, the VMM must trap any attempt by the guest OS to change its page table. This is commonly accomplished by write protecting the page tables and letting the VMM tell the hardware to use a shadow page table instead of the guest OS page tables.

D) Redundant Array of Inexpensive (Independent) Disks. See book, pages 362-366.

Assignment 5D

A) Some factors that limit the exploitable ILP (book Section 3.2):

Factor	Comment
Window size (the size of buffers to keep track of instructions)	At small window sizes, the processor simply cannot see the future instructions that could have been issued in parallel.
Branch, jump prediction	It is hard and expensive to reduce the miss prediction rate beyond a few percent. Misspeculations reduce the exploitable ILP.
Finite number of registers	Registers are needed to store temporary results. This can limit the amount of parallel work possible.
Imperfect alias analysis	If false dependencies through memory and name dependencies can be detected, more instructions can be issued in parallel. However, this is sometimes impractically complex.

B) To realize SMT, we need to have a per-thread renaming table, having separate PC registers, and provide the capability for instructions from multiple threads to commit.

C) Informally, cache coherence means that a value read from the memory systems should reflect the latest write to that same memory location. For an example of what happens when cache coherence is missing, refer to the book, Figure 4.3 (page 206).

D) The correct connections:

```
A 4
B 6
C 13
D 7
E 1
F 2
G 8
H 12
I 5
J 14
K 11
L 9 or L 3
M 3 or M 9
N 10
```

Note: 3 and 9 is equivalent