CHALMERS

2022-10-17

Exam in DAT 105 (DIT 051) Computer Architecture

Time: October 24, 14-18

Person in charge of the exam: Per Stenström, Phone: 0730-346 340.

Supporting material/tools: Chalmers approved calculator

Grading intervals:

- **Fail**: Result < 24
- **Grade 3**: 24 <= Result < 36
- **Grade 4:** 36 <= Result < 48
- **Grade 5:** 48 <= Result

NOTE 1: Bonus points from Real-stuff studies and Quizzes will be added to the exam results for approved exams used solely for higher grades.

NOTE 2: Answers must be given in English

GOOD LUCK!
Per Stenström

[General disclaimer: If you feel that sufficient facts are not provided to solve a problem, either 1) ask the teacher when he visits the exam, or 2) make your own additional assumptions. Additional assumptions will be accepted if they are reasonable and required to solve the problem. Always make sure to motivate your answers.]

ASSIGNMENT 1

1A) We want to compare the performance of two computer systems A and B using the execution times of four programs P1, P2, P3 and P4. The table below lists the execution times of the programs on the two systems and a reference system R.

	P1 [s]	**P2 [s]**	**P3 [s]**	**P4 [s]**
System A	**1**	**10**	**5**	**4**
System B	**0.5**	**17**	**2**	**2**
System R	**2**	**20**	**10**	**8**
Weights	**0.4**	**0.2**	**0.2**	**0.2**

i) Derive the weighted arithmetic mean of the execution times to compare the performance of the two systems (A and B). Why is the comparison unfair? **(3 points)**

ii) Derive the geometric mean speedup assuming the weights are the same over the reference machine R for the two systems A and B. Why is this metric better than the other? (**3 points)**

1B) A computer architect at Intel is choosing between two improvement techniques for the next generation of the microprocessor.

- **Improvement 1:** Four processors on the chip but no change to the size of the shared level 2 cache.
- **Improvement 2:** Two processors on the chip and a 50% larger shared level 2 cache.

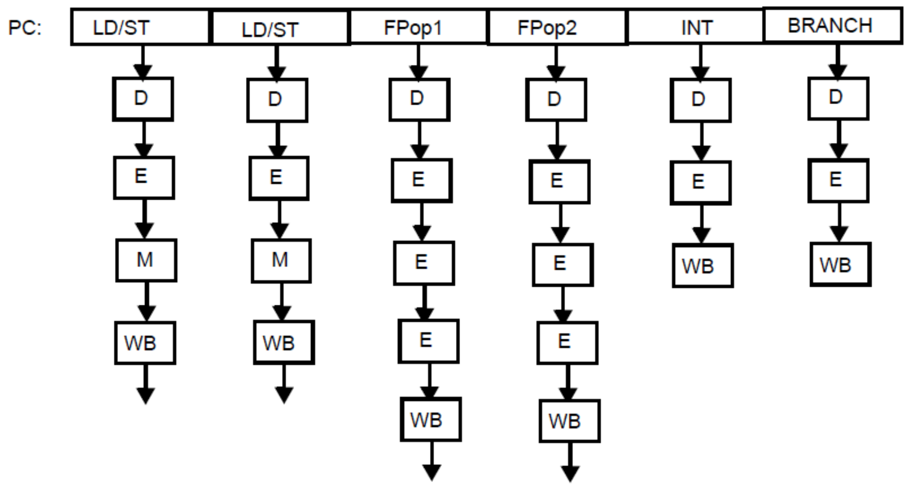
The benchmark used to quantitatively compare the two design alternatives spend 80% of its time, when run on a single processor, in a program section that can be parallelized to enjoy linear speedup on at least four processors. MPKI for the cache is 10 and 7 for the cache in Improvement 1 and 2, resp. CPI is 100 for instructions that miss in the cache and 1 otherwise. Which design alternative should be chosen? **(6 points)**

ASSIGNMENT 2

In this assignment, we consider how much instruction-level parallelism can be extracted statically through compiler techniques.

We consider in this assignment a VLIW architecture that can issue two memory, two floating-point, one integer, and one branch instruction each cycle according to the pipeline organization below.

- In each instruction word, two memory operations, two floating-point operations, and one integer/branch instruction can be scheduled.
- Forwarding circuitry is provided so that a value can be forwarded from a load, an integer, a floating-point, and a store operation to a subsequent instruction in 1, 0, 2, and 0 cycles, respectively.



Consider the following simple computation:

```
LOOP: L.D    F0, 0(R1)
      ADD.D  F4,F0 F2
      S.D    F4,0(R1)
      SUBI   R1,R1, #8
      BNE    R1,R2, LOOP
```

2A) Unroll the loop three times and show the static schedule for it **(4 points)**

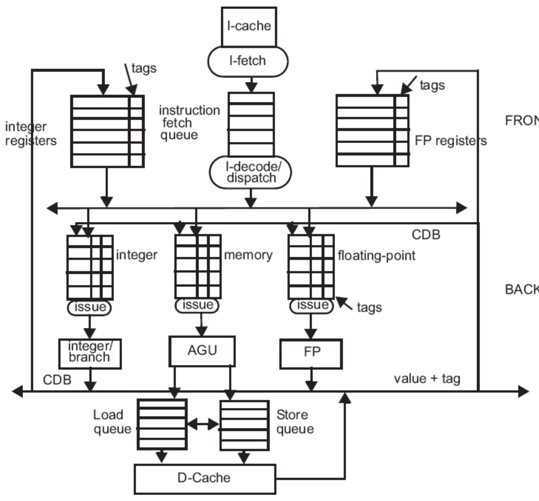
2B) We want to use software pipelining to execute the loop on the VLIW architecture. Derive the kernel and the prologue code for a *software-pipelined loop.* **Hint:** For the kernel, please fill out a schedule table for as many cycles needed to fill the "software pipeline" (i.e., not the VLIW pipeline). (**4 points)**

2C)
The trace through basic blocks A and B is found to be more common that through basic blocks A and D. Schedule and optimize the trace to minimize the number of instructions in the common case and also provide fix-up code in case the less likely case of block A and D is executed. (**4 points**)

```
        LW R4,0(R1)
        ADDI R6,R4,#1
/* block A*/
        BEQ R5,R4,LAB
        LW R6,0(R2)
/*block B*/
/*block D* empty/
  LAB: SW R6,0(R1)
```

ASSIGNMENT 3

The diagram below shows a pipeline using the Tomasulo algorithm for dynamic instruction scheduling. The pipeline consists of 5 stages: Dispatch, Issue, Execution, Cache access and CDB write. Assume that all integer and branch instructions execute in a single cycle whereas floating-point instructions execute in three cycles, thus the duration of the execution stage varies. A cache access (instruction and data) takes a single cycle whereas stores are decomposed into two instructions (one for address generation and one for cache access).



3A) Describe all the necessary changes that are needed to extend the above pipeline to support speculation. Show the new block diagram. If new pipeline stages are added, explain the use of them. (**5 points)**

3B) Determine how long time it takes to execute the first iteration of the code below, in the extended machine of assignment 3A. Fill in the appropriate table to explain your answer. Show also the execution of the first instruction in the second loop iteration. (**5 points)**

```
 LOOP:  L.S F2, 0(R1)
        ADD.S F4, F2, F4
        L.S F6, 0(R2)
        ADD.S F8, F6, F4
        SUBI R1, R1, #8
        SUBI R2, R2, #8
        SUBI R3, R3, #1
        S.S F8, 8(R1)
        BNEZ R3, LOOP
```

3C) What is a 2-level branch predictor? Give an example of a 2-level branch predictor to successfully predict the following branches: (**2 points)**

```
if (X==0) then a++;
if (X==1) then b++;
```

ASSIGNMENT 4

4A) Explain the 3C classification of cache misses and propose a methodology by which misses can be classified quantitatively. (**6 points)**

4B) A computer architect wants to establish the relative performance between a system with a blocking and a non-blocking cache using software prefetching. In software prefetching, prefetch instructions are appropriately inserted in the code shown below. Assume that five instructions are executed per iteration where the CPI for each instruction is one assuming that it doesn't cause a cache miss. On the other hand, if the instruction causes a cache miss, CPI is 100. In addition, prefetch instructions result in CPI=2. Caches have a block size of four words and each vector element is a word. In the non-blocking cache, there are 8 miss-status-holding registers.

```
for (i=0; i<1000; i++)
      C+=A[i];
```

First show how the code is annotated with prefetch instructions. How much faster does the program run on the system with a non-blocking cache using software prefetching? A convincing explanation that is easy to follow is needed for full points. (**6 points)**

ASSIGNMENT 5

5A) Consider a multicore system on a chip comprising a number of processors (cores) that are connected to a single-level private cache. The private caches use the *write-back* write policy. X_i=R_i and X_i=W_i, mean a read and a write request to the *same* address from processor *i,* respectively, where W_i=*C* means that value C is written by processor *i*. Now consider the following access sequence to a location that initially is assigned the value 0:

R_1
R_2
W_1=1
R_1
R_2
W_1=2
R_2

What should be returned by the second read operation from processor 2, what is returned and what is the reason that the correct value is not returned given the cache write policy assumed? (**4 points)**

5B)
Assume a write-back cache and the MSI-protocol. Explain the protocol by means of a finite-state machine with proper use of bus-side requests. **(6 points)**

5C) How is a simple five-stage pipeline extended to support fine-grain multithreading
(4 points)

****** GOOD LUCK! ******

Solutions to the exam in DAT105/DIT 051 2022-10-24

ASSIGNMENT 1

1A)

i)

$T_{A,\text{ weighted arith}} = (0.4\text{x}10+0.2(10+5+4))/4 = 7.8$
$T_{B,\text{weighted arith}} = (0.4\text{x}0.5 + 0.2(17+2+2))/4 = 4.5$

Arithmetic means can be misleading if there are outliers. In this case, P2 is an outlier on both A and B so the conclusion that B is faster appears fair.

Here, A is faster than B. This is unfair as B outperforms A on P1, P3 and P4 and it is the outlier P2 that makes A better in light of arithmetic mean.

ii)
We first establish the speedup numbers with respect to the reference machine
A: SP(P1) = 2/1 = 2; SP(P2) = 20/10=2; SP(P3) = 2; SP(P4) = 2. Hence
$T_{A,\text{ geom}}=(2\text{x}2\text{x}2\text{x}2)^{1/4} = 2$

B: SP(P1) = 2/0.5 = 4; SP(P2) = 20/17=1.2; SP(P3) = 5; SP(P4) = 4. Hence
$T_{B,\text{ geom}}=(4\text{x}1.2\text{x}5\text{x}4)^{1/4} = 3.1$

Since higher is better (speedup), B is better than A. What happens is that outliers tend to be smoothened out when using geometric mean. B does better than A on all programs except for P2 where it does only slightly worse on B than on A.

1B)

Let's assume that the execution time on a single processor on the system with Improvement 1 is T. Then the execution time assuming Improvement 1 is 0.2T + 0.8T/4.

Let's now see how T is reduced assuming a 50% larger cache. The fraction of T spent on cache misses on the cache for Improvement 1 is 10x100/(1000x1 + 10x100) = 0.5. So, 50% of the execution time is spent servicing cache misses. For the cache in Improvement 2, we reduce the time servicing cache misses by 10-7/10 = 30%. The impact of the overall execution time is 15% as 50% of the execution time is spent servicing cache misses. Hence T′=0.85T if run on a single processor on the system of Improvement 2. With two processors, the execution time becomes 0.2x0.85T + 0.8x0.85T/2

Improvement 1/Improvement 2 = 0.2T + 0.8T/4/(0.2x0.85T + 0.8x0.85T/2) = 0.4/0.51 meaning that Improvement 1 offers slightly shorter execution time than Improvement 2.

ASSIGNMENT 2

2A)

Here is the unrolled loop (three times) after renaming and code motion and elimination of replicated subtract instruction and branches.

```
LOOP: L.D   F0, 0(R1)
      L.D   F6, -8(R1)
      L.D   F12, -16(R1)
      ADD.D F4,F0 F2
      ADD.D F8,F6 F2
      ADD.D F14,F12 F2
      S.D   F4,0(R1)
      S.D   F8,-8(R1)
      S.D   F14,-16(R1)
      SUBI  R1,R1, #24
      BNE   R1,R2, LOOP
```

The first iteration uses F0 and F4, whereas the second and third iteration use F6/F8 and F12/F14, resp. By hoisting loads to the top and moving all stores to the end, we maximize the distance between a load and an add and an add and a store to avoid pipeline bubbles.

2B)

Let's denote the kernel instructions O1, O2 and O3. Given the forwarding mechanism assumed, the floating-point ADD must wait a cycle and the store (S.D) must wait two cycles.

```
LOOP: L.D   F0, 0(R1)     --O1
      ADD.D F4,F0 F2      --O2
      S.D   F4,0(R1)      --O3
      SUBI  R1,R1, #8
      BNE   R1,R2, LOOP
```

We show below the prologue and kernel code below:

	ITE 1	ITE 2	ITE3	ITE4	ITE5	ITE6	ITE7	ITE8	ITE9	
INST1	O1									
INST2		O1								
INST3	O2		O1							Pro-logue
INST4		O2		O1						
INST5			O2		O1					
INST6	O3			O2		O1				Kernel
INST7		O3			O2		O1			
INST8			O3			O2		O1		
INST9				O3			O2			
INST10					O3			O2		
INST11						O3				
INST12							O3			

2C)

```
          LW R4,0(R1)
…………LW R6, (R2)
/* block A*/
          BEQ R5,R4,LAB1
LAB:      SW R6,0(R2)
          ….

LAB1: ADDI R6,R4,#1
          J LAB
/*block B*/
```

ASSIGNMENT 3

3A)

In order to extend the pipeline, a new stage is introduced called commit. For speculative execution to be supported, *branch prediction* is needed so that instructions following a branch can be speculatively executed before the branch instruction is validated. A branch predictor is added to the instruction fetch stage and does a branch prediction in a single cycle. Each speculatively executed instruction will be tracked, in program order, in a structure called *reorder buffer*. The reorder buffer may also host all speculatively generated results as they cannot be written back to the register file until an instruction is non-speculative, i.e., committed. This happens when a preceding branch instruction is validated to be correctly predicted. Then the branch instruction will be *committed* (it becomes non-speculative). In subsequent cycles the subsequent speculative instructions are also committed. When an instruction is committed, the value it has generated (if any) will be written back to the registerfile and the instruction along with its result is removed from the reorder buffer.

3B)

We first enumerate the instructions as follows and use IK,L to refer to instruction IK in iteration L:

```
I1:L:  L.S F2, 0(R1)
I2     ADD.S F4, F2, F4
I3     L.S F6, 0(R2)
I4     ADD.S F8, F6, F4
I5     SUBI R1, R1, #8
I6     SUBI R2, R2, #8
I7     SUBI R3, R3, #1
I8     S.S F8, 8(R1)
I9     BNEZ R3, LOOP
```

We refer in the pipeline diagram below to pipeline stages F-Fetch, D-Dispatch, I-Issue, E/C - Execute/Cache access, CD-Common data bus transfer, CM-Commit. We assume that the previous branch is correctly predicted and has committed so that all subsequent instructions will commit one by one.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16
I1,1	F	D	I	E/C	CD	CM										
I2,1		F	D	I	E/C	E/C	E/C	CD	CM							
I3,1			F	D	I	E/C	CD	CM								
I4,1				F	D	I	Y	E/C	E/C	E/C	CD	CM				
I5,1					F	D	I	E/C	CD	CM						
I6,1						F	D	I	E/C	CD	CM					
I7,1							F	D	I	E/C	X	CD	CM			
I8,1								F	D	I	Y	E/C	CD	CM		
I9,1									F	D	I	E/C	X	CD	CM	
I1,2										F	D	I	E/C	E/C	CD	CM

Most of the time, instructions flow through the pipeline but in some cases, they will be stalled due to either a structural hazard (denoted X) or a RAW hazard (denoted Y). There is a RAW hazard between I3,1 and I4,1 so I4,1 cannot proceed to the E/C stage until cycle C8. Next, instruction I7,1 cannot proceed to the CD stage in cycle C11 because it is being used by instruction I4,1 which causes a structural hazard. Similarly, instruction I9,1 cannot proceed to the CD stage in cycle 13 as it is being used by I8,1. There is also a RAW hazard between I4,1 and I8,1 so I8,1 cannot move to the E/C stage until in cycle 12.

3C)

The conditions in the two if-clauses are not true at the same time; they are mutually exclusive. Hence, if the first branch is taken, the second branch is not taken and vice versa. This can be tracked by a two-level branch predictor. See textbook pages 121 and 122.

ASSIGNMENT 4

4A)
The 3C model refers to the root cause of misses in a cache.

First C: *Compulsory or Cold misses*. These are the misses that happen regardless of the size of the cache and corresponds to the number of unique blocks that are accessed during the execution of the program. The number of C-misses are the number of unique block addresses generated by the execution of a program.

Second C: *Capacity misses*. These are the additional misses that would not have happened in an infinitely sized cache. We can establish the number of capacity cache misses by first counting the number of misses on a fully associative cache with n optimal cache replacement algorithm (referred to as OPT) and subtracting it by the number of compulsory or cold misses.

Third C: *Conflict misses*. These are the additional misses that would not happen in a fully associative cache with the OPT replacement algorithm. We can establish the number of conflict misses by first counting the number of misses on a cache and subtracting it by the sum of the number of capacity and compulsory/cold misses.

4B) Let's first establish how long time the program takes on the architecture with a blocking cache.

We note that four iterations can be executed between cache misses as a block contains four vector elements. The first iteration will experience a cache miss and will take 100 + 4 cycles = 104 cycles. The next three iterations take 3x5 = 15 cycles. Hence, four iterations take 119 cycles and all 1000 iterations take 250 x 119 = 29,750 cycles.

Let's now annotate the code with a prefetch instruction (prefetch(A[i]):

```
for (i=0; i<1000; i++){
    prefetch(A[i+4);
    C+=A[i];
}
```

The prefetch instruction will prefetch the next block if it is not present in cache. The first miss is unavoidable but subsequent misses can ideally be avoided. It means that the first four iterations will take 119 cycles but the next 996 iterations will only take 5 cycles each. Hence the execution time using prefetching will be 119 + 996*5 = 5,099 cycles. Hence, the speedup of a non-blocking cache with prefetching is 29,750/5,099 = 5.8.

ASSIGNMENT 5

5A)

The value 2 should be returned but, unfortunately, original value (0) is returned. The reason is that when processor 1 modifies the block, processor 2's cache is not notified.

5B)

See textbook pages 253-257.

5C)

A new pipeline stage is added between Fetch and Decode called thread switch or TS for short. This stage will switch to a new thread each cycle. Assuming N threads, we need N program counters where the one used is selected by TS. Moreover, N registerfiles, one for each thread, are also needed.