Indian Institute of Technology Kharagpur

Department of Computer Science and Engineering

Exam-3, Autumn 2021-22

Computer Organization and Architecture (CS31007)

Students: 133 **Date**: 15-November-2021

Full marks: 60 **Time**: 12:00 noon – 2:00 PM

Credit: 40%

INSTRUCTIONS: This is an OPEN-BOOK, OPEN-NOTES test. The questions are such that they require either numerical answers or very short answers (in a few sentences or a few lines of code). Please submit ONLY THE ANSWERS. It might help if you typeset your solution using software like MS Word/LibreOffice Writer, convert it to PDF, and then upload the PDF file on Moodle. DO NOT FORGET TO WRITE YOUR NAME AND ROLL NUMBER AT THE TOP OF YOUR ANSWER SHEET. You may use calculators if required. ANSWER ALL QUESTIONS.

1. (a) Consider a 64-byte cache with 8-byte blocks, an associativity of 2 and LRU block replacement. Virtual addresses are 16 bits. The cache is physically tagged. The processor has 16KB of physical memory. What is the total number of tag bits? [**3**]

Answer: The cache is 64-bytes with 8-byte blocks, so there are 8 blocks. The associativity is 2, so there are 4 sets. Since there are 16KB = 2^{14} bytes of physical memory, a physical address is 14 bits long. Of these, 3 bits are taken for the offset (8-byte blocks), and 2 for the index (4 sets). That leaves 9 tag bits per block. Since there are 8 blocks, that implies 72 tag bits.

(b) Assume that the processor in part-(a) is part of a computer where each page is 64 bytes. How large would a single-level page table be given that each page requires 4 protection bits, and entries must be an integral number of bytes? [**3**]

Answer: There is 16KB of physical memory and 64-byte pages, meaning there are 256 physical pages. Hence, each physical page identifier is 8 bits long. Each page table entry has a physical page identifier and 4 protection bits, that's 12 bits which when round up to be an integral number of bytes is 2 bytes. A single-level page table has one entry per virtual page. With a 16-bit virtual address space, there is 64 KB of virtual memory. Since each page is 64 bytes, that means there are 1K pages. At 2 bytes per page, the page table is 2KB in size.

(c) Consider a data cache which follows the Write Back with Write Allocate policy, to handle read misses and write misses. The cache has a 95% read hit rate, 90% write hit rate, zero clock cycle hit latency, and 50 clock cycles latency to write/read a single block to/from main memory. The CPU is executing a program in which 5% instructions are load instructions, and 0% instructions are store instructions. On the occasions when a cache read miss happens, 10% of times the target cache block is found to be dirty. Calculate the effective CPI when the above-mentioned program is executed, assuming all instructions execute in one clock cycle when interaction with memory is not required, or when cache hit happens. [**4**]

Answer: Effective CPI = Fraction of non-load instructions * $CPI_{non\text{-}load}$ + Fraction of load instructions * CPI_{load}

= (1- 0.05) * 1 + 0.05 * CPI_{load}

= 0.95 + 0.05 * CPI_{load}

CPI_{load} = Hit rate * Hit latency + Miss rate * Miss latency

= 0.95 * 1 + (1 – 0.95) * $(Miss\ latency)_{load}$

= 0.95 + 0.05 * [Probability of dirty block * 2 * 50 + Probability of non-dirty block * 50]

= 0.95 + 0.05 * [0.1 * 100 + 0.9 * 50]

= 3.70

Hence, Effective CPI = 0.95 + 0.05 * 3.70 = 1.135

2. Consider a CPU with only one level of cache memory.

(a) For a data cache with a 92% hit rate and a 2-cycle hit latency (unlike zero cycle hit latency in most processors), calculate the *Average Memory Access Time* (AMAT). Assume that latency to memory and the cache miss penalty together is 124 cycles. Note: the cache must be accessed after memory returns the data. [2.5]

Answer: Recall that AMAT = Time for a hit + Miss rate * Miss penalty.

AMAT = 2 + 0.08 * 124 = **11.92 cycles**

(b) Calculate the performance of a processor taking into account stalls due to data cache and instruction cache misses. The data cache (for loads and stores) is the same as described in part-(a) above and 30% of instructions are loads and stores. The instruction cache has a hit rate of 90% with a miss penalty of 50 cycles. Assume the base CPI using a perfect memory system is 1.0. Calculate the CPI of the pipeline, assuming everything else is working perfectly. Assume the load never stalls a dependent instruction and assume the processor must wait for stores to finish when they miss the cache. Finally, assume that instruction cache misses and data cache misses never occur at the same time.

i. Calculate the additional CPI due to the instruction cache stalls. [2.5]

Answer: Additional CPI due to Icache stalls

= Hit rate * Hit latency + Miss rate * Miss latency

= 0.9 * 2 + 0.1 * 50 = 1.8 + 5 = **6.8**

ii. Calculate the additional CPI due to the data cache stalls. [2.5]

Answer: (Using same formula as part-(i)

Additional CPI due to Dcache stalls

= 0.92 * 2 + 0.08 * 124 = **11.76**

iii. Calculate the overall CPI for the machine. [2.5]

Answer: The overall CPI

= 0.3 * 11.76 + 0.7 * 1.0 + 1.0 * 6.8 = **11.03**

3. (a) Explain two differences between a process in wait state and a process in ready state. [2]

Answer:

(i) A process in the **wait state** is not under current consideration of the operating system to get executed, while a process in the **ready state** is ready for execution when its turn comes.

(ii) The wait time for a process in **wait state** is unpredictable, while that for a process in **ready state** (waiting in the ready queue) is more accurately predictable, if the position of the process in the ready queue and the length of the time-slice assigned to each process is known.

(b) A 32-bit CPU receives an interrupt with an interrupt code 0x04. It is known that the interrupt vector table for this computer starts at address 0x0000ab00. The value of the relevant entry in the interrupt vector table for the raised interrupt (say, VT) is related to the address of the entry (say A) by:
$VT = A$ + **0x00010000**. The interrupt service routine to handle this interrupt is of length 8 KB, and the entire interrupt service routine is stored contiguously in memory. Calculate the (starting) address of the last instruction of the interrupt service routine. Note: 1 KB = 2^{10} bytes. [4]

Answer: The address of the relevant entry (i.e., the starting address of the interrupt service routine) can be calculated by considering the value 0x04 as an index into the interrupt vector table, and is given by:

A = 0x0000ab00 + 0x04 * 4 = 0x0000ab10.

Hence, the value of the entry is:

VT = 0x0000ab10 + 0x00010000 = 0x0001ab10, which is also the starting address of the interrupt service routine.

The length of the interrupt service routine = 8 KB = 2^{13} bytes = 0x00002000 bytes.

Hence, the (starting) address of the last instruction of the interrupt service routine

= VT + 0x00002000 – 0x04 = 0x0001ab10 + 0x00002000 – 0x04 = **0x0001cb0c**

Note the adjustment of the address by 1 word-width, i.e. 4 bytes, in the calculation above.

(c) Suppose we have a magnetic disk with the following parameters: average seek time 12 ms; rotation rate 3600 RPM; transfer rate 3.5 MB/second (with 1 MB ≡ 2^{20} bytes); number of sectors per track 64; sector size 512 bytes; controller overhead 5.5 ms.

i. Calculate the average time to read a single sector. [**2**]

Answer: Av. time to read a single sector

= seek time + rotational delay + transfer time + controller overhead

= [12 + (0.5 * 60 * 10^3 / 3600) + (512 / (3.5 * 2^{20})) * 1000 + 5.5] ms = **25.97 ms**

ii. Calculate the average time to read 8 KB in 16 consecutive sectors on the same cylinder. [**2**]

Answer: Only the transfer time quantity would get changed.

Hence, Av. time to read 16 consecutive sectors

= [12 + (0.5 * 60 * 10^3 / 3600) + (16 * 512 / (3.5 * 2^{20})) * 1000 + 5.5] ms = **28.07 ms**

4. (a) Consider a regular single-cycle data path implementation of a non-pipelined MIPS processor that implements only a small subset (lw, sw, add, addi, slt, beq) of instructions. Write down the appropriate values 0, 1, x (don't care) of the following control signals that are needed to execute the MIPS instruction **sw $t1, -64($t2)**:
RegDest, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch, ALUOp1, ALUOp0, Function Fields (6-bit value). [**2**]

Answer: RegDest = *x*, ALUSrc = 1, MemtoReg = *x*, RegWrite = 0, MemRead = 0, MemWrite = 1, Branch = 0, ALUOp1 = 0, ALUOp0 = 0, Function Fields = {*x, x, x, x, x. x*};

(b) Now suppose *one wire* of the 32-bit bus that leaves the Sign-Extension module, has become defective with a stuck-at-0 fault (the wire is always fixed at logic-0 regardless of the actual signal arriving on it). Which of the following statement(s) is(are) **true**? Justify your answer in brief. [**2**]

A. The execution of some *lw* and *sw* instructions might be affected, but other instructions will be fine.

B. The execution of some *lw*, *sw* and *beq* instructions might be affected but other instructions will be fine.

C. The execution of some *sw* instructions, some *addi* instructions, and all *beq* instructions will be affected.

D. The execution of all add instructions and all branch instructions of type **beq $t1, $t2, Target** with ($t1) ≠ ($t2), will remain unaffected.

E. None of the above.

Answer: The *correct answer* is **D**, because all *R*-type and "*beq-not-taken*" instructions will **not** be affected by the fault on the 32-bit bus leaving the sign-extension module. Choice **A** is not correct because some *beq, addi* instructions will be affected; **B** is not correct since *addi* will be affected; **C** is not correct because some *beq* (those not taken) will not be affected.

5. Consider a regular single-cycle data path implementation of a non-pipelined MIPS processor as in Question–4(a). The operation-times for the major functional units are given as follows:

- Instruction and Data Memory Units: 2 ns each;
- Register File Read/Write: 1 ns each;
- ALU: 3 ns; Adder for PC+4: 2 ns; Adder for branch-address computation: 2 ns;
- MUX block: 1 ns each; Sign-Extend and Shift-Left: 1 ns each;
- other units: negligible.

(a) Calculate the maximum clock frequency with which this CPU can run. [**2**]

Answer: We consider *lw* and a *R*-type instruction for demonstration. Other instructions would be similar and would take shorter time to complete execution.
For *lw*, the longest path is through:
IM → Reg.File.Read → MUX → ALU → DM → MUX → Reg.File.Write; other units such as Sign-Extend, PC-update, etc., are working in parallel. This gives a delay of 2 + 1 + 1 + 3 + 2 + 1 + 1 = 11 *ns*;
For a *R*-type the longest path is through:
IM → Reg.File.Read → MUX → ALU → MUX → Reg.File.Write; delay is 9 *ns*.

Since the maximum delay is 11 *ns*, CCT = 11 *ns*, and hence,
the maximum operating clock frequency = $1/(11 \times 10^{-9})$ Hz ≈ 91 MHz.

(b) The same machine is now implemented as 5-stage pipeline where the delay through each of the inter-stage pipeline-latches is 1 ns. Estimate the maximum speed-up that is achievable by the pipelined machine compared to the non-pipelined one (assume no hazard is present). [**2**]

Solution: We consider the standard five-stage pipelined implementation with IF, ID/RR, EX, Mem, and WB. Among these stages, the EX-stage takes the longest time: one MUX plus the ALU-unit, i.e., the delay is 1 + 3 = 4 *ns*.
The worst-case delay among all stages including pipeline overhead is thus = 4 + 1 = 5 *ns*.
Therefore, when the number of instructions is large, the maximum speed-up compared to the non-pipelined machine becomes = 11/5 = 2.2

6. Consider the following sequence of actual outcomes in a program for a branch instruction that is iteratively executed 20 times. "T" means "the branch is taken", whereas "N" means "the branch is not taken". Assume that this is the only branch instruction in the program.

T T T N T N T T T N T N T N T N N N T N

The pipeline controller uses a dynamic branch predictor based on **2-bit branch history**. Assume that the predictor state is initialized to "N". Calculate the number of mispredictions. [**3**]

Answer:

Predicted state:	N N T T T T T T T T T T T T T N N N
Actual branch status:	T T T N T N T T T N T N T N T N N N T N
Outcome: Correct (C), Mispredicted (M)	M M C M C M C C C M C M C M C MM C M C

The number of mispredictions is therefore, = 10.

7. Consider a standard 5-stage pipelined implementation of MIPS processor, where each of the stages takes the same amount of time (i.e., 1 clock cycle). The PC-Update is being implemented in the Mem-Stage. Given a program *P*, assume that all machine instructions in *P* can be executed without any pipeline stall, except those which follow a *beq* instruction. In *P*, the frequency of branch instructions is 30% and out of those, 60% are not taken. The pipeline controller works on statically predicting that a branch is always "*not taken*". Assume that the pipeline overhead due to inter-stage latches is negligible.

   (a) Calculate the maximum speed-up achievable by this pipelined machine over non-pipelined single-cycle implementation. [**2**]

   Answer: Max_speed-up = CPI for non-pipelined /{1 + (Branch_Freq. * Branch_Penalty)}

   = Pipeline_depth/{1 + (Branch_Freq. * Branch_Penalty)}

   Note that Pipeline_depth = 5 for MIPS. If PC-Update is conducted in the Mem-stage, three stalls are needed to determine the branch outcome. However, the pipeline runs with the prediction that a branch is always *not taken*; hence, penalty will be needed for taken-branches only. Since, 40% of branches are taken,

   Max_speed-up = 5/(1 + 0.3*0.4*3) ≈ 3.67

   (b) If you are allowed to do certain hardware modification concerning zero-checking and PC-Update, what will be the speed-up in the modified machine? Describe the scheme for modification and justify your answer. [**2**]

   Answer: By deploying hardware modifications, we can bring branch comparator (zero-checking) and PC-Update to the ID/RR-stage, and the penalty for an incorrect prediction can be reduced to one stall only. Thus, Max_speed-up = 5/(1 + 0.3*0.4*1) ≈ 4.46

8. (a) Consider the execution of the following code *P* in a standard five-stage MIPS pipelined machine (IF, ID/RR, EX, Mem, WB), without any additional forwarding hardware. However, Register-Read (Write) is always accomplished in the second (first) half-cycle of the system clock pulse. Assume that the first stage (IF: Instruction Fetch) of the first instruction (I0) is being executed during clock cycle #1. Also assume that there will be no arithmetic overflow while running *P*.

   P:

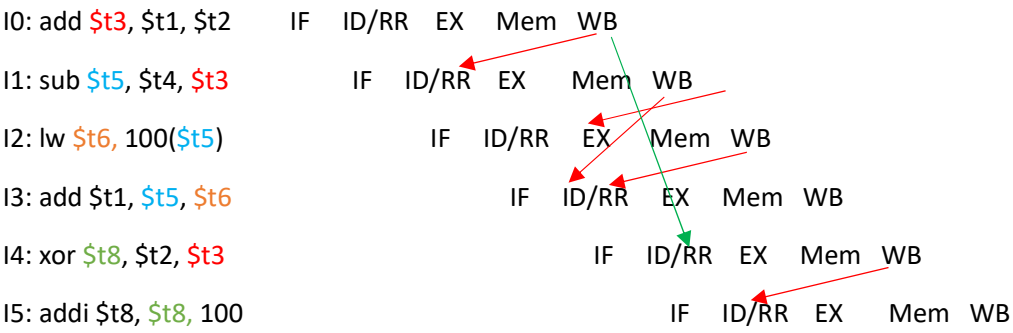
```
I0: add $t3, $t1, $t2
I1: sub $t5, $t4, $t3
I2: lw $t6, 100($t5)
I3: add $t1, $t5, $t6
I4: xor $t8, $t2, $t3
I5: addi $t8, $t8, 100
```

   i. Identify all data-hazards in *P*; [**5**]

   ii. In the worst case, how many clock cycles are needed to complete *P*? [**3**]

   (b) We now modify the machine by using additional resources such as hazard detector and forwarding hardware (EX → EX, and Mem → EX). Also, a smart compiler is available to reshuffle the code if needed. Write the modified sequence of instructions which would minimize the number of pipeline stalls under this arrangement. Also, write the estimated number of clock cycles needed to complete *P* now. [**5**]

Answer: 8(a) i



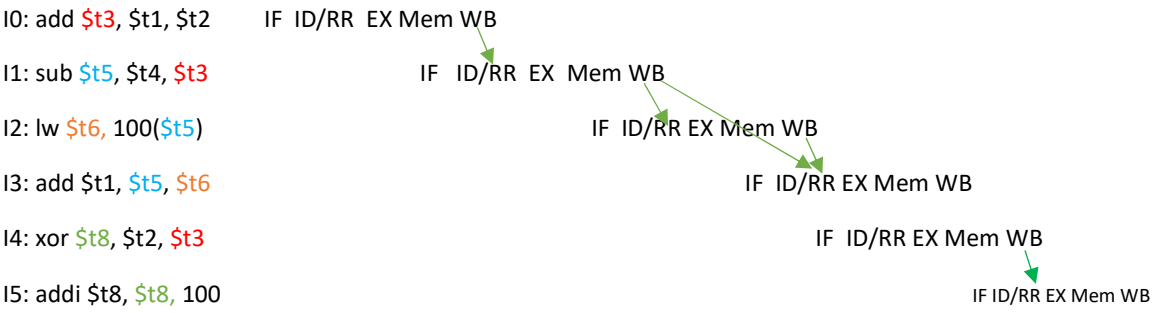
In the absence of any forwarding hardware, in the static case, i.e., when the pipeline is allowed to advance without any stall, *five* RAW hazards can be identified as shown with red arrows:

$t3 (I0, I1); $t5 (I1, I2); $t6 (I2, I3); $t5 (I1, I3); $t8 (I4, I5);

Note that there is no hazard $t3 (I0, I4); this does not arise because of half-cycle register read-write policy. Also, clearing of one hazard by inserting a stall may automatically resolve subsequent hazards.

Answer: 8(a) ii

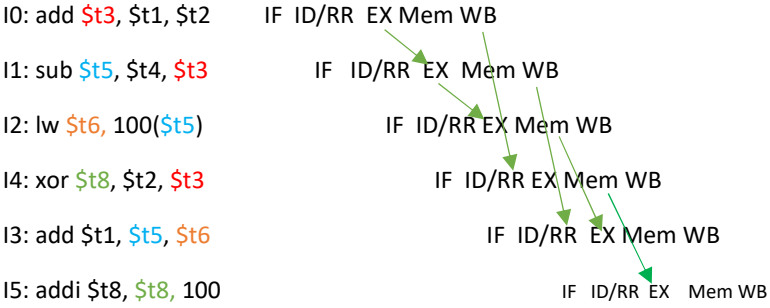
In the absence of forwarding, we need to insert stall as follows:



Thus, we need at least 18 clock cycles.

Answer: 8(b)

In the presence of forwarding and reshuffling we may have the following timing diagram:



We use EX → EX and Mem → EX forwarding, and swap I3 and I4 to resolve load-use data hazard. As seen from the pipeline diagram above, *P* can be completed in just 10 clock cycles without any stall. The forwarded data are shown with green arrows.

(c) If there is a data hazard encountered by an instruction that immediately follows a load instruction, how is that detected by the hardware? [**2 marks**]

Answer: 8(c)

Data hazard due to load-use in the next instruction can be detected when the Boolean condition is TRUE:

ID/EX.MemRead is TRUE (1)

and

((ID/EX.RegisterRt = IF/ID.RegisterRs) *or* (ID/EX.RegisterRt = IF/ID.RegisterRt))