Computing system: performance, speedup, performance/cost

Origins and benefits of scalar instruction pipelines and caches

Pipeline Hazards
- Structural – need more HW to wait less
- Data Dependence – RAW (WAR & WAW resolvable)
  - ByPass HW lessens RAW delays even if in-order Xeq
  - Dynamic HW support out-of-order Xeq & speculation
- Control – where fetch after jumps or branches

How Bit-Fields of Addresses Access L1 Caches + VirtMem Translation Look-aside Buffer

Instruction Level Parallelism (ILP) => pipelines
- Hazards limit fast pipelining
- Ways to lessen impact of hazards
  - More memory and register ports
  - Branch/Jump prediction
  - Out-of-order execution
  - Speculative execution
Code rescheduling (moving)
  Static – by compiler SW
    Loop Unrolling
    (Software pipelining)

Dynamic – by Tomasulo-style HW
  Advantages of Tomasulo
  Disadvantages of Tomasulo
  Register Renaming via Reservation Stations
  Results Broadcast via Common Data Bus

Superscalar (2-8 instructions per cycle) vs VLIW
  Advantages and disadvantages of each
  Which better?  Why?
  Fast CPUs use both – how?  Why?

Thread-level support
  Cray/Tera MTA-1 128 threads * 256 CPUs, NO caches
  Multi-CPU (2-8 cores) on a chip instead of speculation
  Less power per core from lower voltage, less logic
Vector Processing
  Loop unrolling
  Loop execution time
    Without chaining
    With chaining

Multiprocessor and Multicore Chip Designs

Snoopy Cache Coherence Protocols
  Sequence of Signals on Shared Memory Bus
You have 75 minutes for 3 questions with 8 parts on 3 pages = 100 points. Read all questions before you start. Some easy questions may be later. Show your math work for partial credit. Note any special assumptions that you make for any question. Raise your hand for help. Write your answers on fronts and backs of this exam, and hand it in directly to me.

**OPEN BOOK**            **OPEN NOTES**           **CLOSED FRIENDS**

1) (40 pts) Your NETalker company is designing a new wireless ZeBox, that will run your patented netsearch code 95% of its working time. You must make a critical hardware choice affecting both the cost and speed of ZeBox. The old Box costs $36 and takes an average of 40 seconds per search. A new ZeBox that takes 40 seconds per search costs only $10.

Four optional hardware improvements to ZeBox are feasible: W, X, Y, Z. Each speeds up only parts of netsearch execution, as shown in the figure.

For an average run of netsearch divided into 20 equal time periods, only W can speed up 3 periods; W or X (but not both together) can speed the next 3 periods; only X for 2 periods; Y or X for 5; only Y for 3, and Y or Z for 4. Each improvement has a different speedup factor for its sections of the run and an additional cost. W is 15 times faster for $15 more. X is 5 times faster for $5. Y is 10x for $10 more. Z is 20 times for $20 more. If two improvements are possible during a time period, only the faster one has any effect. The question is what WXYZ combination gives the best performance-to-total_cost ratio. For example, if ZeBox has all four optional improvements W+X+Y+Z, total_cost for each new ZeBox will be $60. If only X, $15. If Y+Z, $40.

1A) (15 pts) **What** is the maximum speedup possible for netsearch? (If none of WXYZ used, speedup is 1.)

1B) (10 pts) **What** is the performance versus total_cost ratio for the fastest version of the new ZeBox?

1C) (15 pts) **Which** of the 16 possible W,X,Y,Z combinations is best? **Give** its performance-cost ratio.

{ Do NOT attempt to evaluate all 16 combinations explicitly unless you have extra time at the end! }

{Instead, evaluate some carefully chosen combinations and state your reasons for skipping the rest.}
2) (30 pts) A dual-issue (two pipelines) multiscalar version of the integer MIPS 5-stage pipeline has all feasible data forwarding and enough hardware to avoid all structural hazards. However, ALU results and memory loads are available only at stage (EX, MM) end. ALU inputs and memory stores are needed at stage (EX, MM) start.

2A) (5 pts) **What** is the **instruction penalty** for a **load** memory instruction **followed by** an **ALU** instruction using its result? **What** is the **average cycle penalty**? You may use a sketch to justify your answer, if you wish.
2B) (10 pts) **Identify and give the average cycle penalty** for two other (non-zero) MIPS data hazards.

Integer MIPS pipelines have no WAR nor WAW data hazards, so we seek only pairs with a RAW hazard. Taking a hint from 2D below, the other likely pairs are ALU-ALU, ALU-store, and ALU-load. There is also a tiny ALU-Branch problem, ignored in 2D. The other combinations of linked Load-Store (Load-Load) (redundant unless a bad non-atomic Test&Set Lock or a simulated and costly indirect address for a Store or Load) and Load-Branch (who would keep a loop index or condition in so-slow memory?) are reasonably ignored, as unlikely. See blue line in figures below for cycle penalties after leading ALU of an ALU-xxx pair.

Cycle 0 1 2 3 4 5 6
ALU 0 IF ID EX MM WB <<<
ALU 1 IF ID EX MM WB <<<

ALU ok Load ok

ALU-ALU
If first ALU fetched at Cycle 0 and issued into pipe1, the following dependent ALU will not be issued until cycle 2 and its EX will start in cycle 3, for a "CYCLE PENALTY" of 1.0 cycle. If the first ALU fetched at Cycle 0 is issued into pipe2, the following dependent ALU will be issued in cycle 1 into pipe2, and there is no (0 cycle) penalty. The average "CYCLE PENALTY" is 0.5 cycle.

ALU-Store
If ALU operation determines the value to be stored in memory, it is not needed until start of MM, so there is no stall and no penalty.

ALU-Load
If the ALU operation determines a register value used to calculate the effective address of the Load, this is same as ALU-ALU; the "CYCLE PENALTY" is 0.5.
2C) (5 pts) Choice Q has a Branch Target Buffer (BTB) that is searched in each stage 1 (IF) with the address of the instruction being fetched. Each BTB entry gives the address (the search key) of a recently taken branch or a jump plus its target address for the next instruction to fetch after it. If an instruction address is not in BTB, it is assumed not to change the Program Counter. If it is a control instruction, the actual target address is known at the end of stage 2 (ID) and checked against the prediction. If the prediction was wrong, the BTB is updated, incorrectly fetched instructions are flushed from the pipeline (annulled) and the correct fetch address is used at the middle of stage 3 (EX). If each control instruction that is taken has its address in the BTB 90% of the time and each control instruction with its address in the BTB is taken 80% of the time, what is the average cycle penalty for each MIPS control instruction if choice Q is implemented?

2D) (10 pts) Assume that BTB choice Q has been implemented and dual-issue MIPS instructions run with these frequencies: 53% ALU, 21% Load, 12% Store, and 14% Branch/Jump. Assume these data hazard frequencies:

- Closest possible hazard after each ALU instruction - 10% adjacent dependent ALU, 5% 2-away dependent ALU, and 2% 3-away dependent ALU; 10% adjacent dependent store, 5% 2-away dependent store, and 2% 3-away dependent store; and
- Closest possible hazard after each memory load instruction - 20% adjacent dependent ALU, 10% 2-away dependent ALU, and 5% 3-away dependent ALU (the compiler eliminates redundant stores).

If there are no structural hazards and no memory system stalls, what is the average number of instructions completed per cycle (IPC) for this dual-issue integer MIPS processor?

NOTE: There originally was a question part 2E) giving a choice R for Branches, a 3-instruction delay slot with a 68% chance of compiler moving one useful instruction into the slot, 11% for 2 in the slot and 1% for 3 useful. The resulting branch penalty is \( \frac{2.20}{2} \) cycles = 1.10 cycles. For this choice R, CPI = 0.60 + 0.154 = 0.754 and \( \text{IPC} = \frac{1}{0.754} = 1.326 \) instructions completed per cycle. BTB speeds runs by \( \frac{1.528}{1.326} = 1.152, 15\% \) faster.
3) (20 pts) This is one possible MIPS code to calculate a Sum A*X Plus Y inner loop for Gaussian elimination.

```
start: ; assume R1, R2, F2 set correctly before here
  saxpy: LD F2, 0(R1) ; load X[j]
  MULTD F4, F0, F2 ; Multiply a * X[j]
  LD F6, 0(R2) ; load Y[j]
  ADDDD F6, F6, F4 ; Add a*X[j] + Y[j]
  SD 0(R2), F6 ; STORE A * X[I] + Y[I]
  ADDI R1, R1, #8 ; increment X index
  ADDI R2, R2, #8 ; increment Y index
  SGTI R3, R1, final_ndx ; test if done
  BEQZ R3, saxpy ; loop if not done

exit:
```

Assume this code runs on a single-issue MIPS INT+FP pipeline with a one cycle delay slot after each branch. **Unroll** this loop **twice** and schedule its instructions to **minimize** the number of stalls between instructions, using the latencies in the table that follows. Eliminate any unneeded overhead instructions. **Write** the resulting code. If the loop is executed for vectors X[j] and Y[j], each of size 50, **what** is the total number of stall cycles wasted between start and exit using your unrolled code? **What** is the value of R1 at exit?

<table>
<thead>
<tr>
<th>Instruction producing result</th>
<th>Instruction using result</th>
<th>Latency in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) FP ALU op</td>
<td>Another FP ALU op</td>
<td>3</td>
</tr>
<tr>
<td>b) FP ALU op</td>
<td>Store double</td>
<td>2</td>
</tr>
<tr>
<td>c) Load double</td>
<td>FP ALU op</td>
<td>1</td>
</tr>
<tr>
<td>d) Load double</td>
<td>Store double</td>
<td>0</td>
</tr>
<tr>
<td>e) Integer op</td>
<td>Integer op</td>
<td>0</td>
</tr>
</tbody>
</table>