COMPUTER ARCHITECTURE

Comprehensive Exam: Architecture (60 points) **Autumn 1991**

1. (15 points total) *Instruction Set Design.*

Your company currently produces a load/store processor. The table below lists the instruction distributions (along with the instruction latencies) for your most important application–SPICE.

	latency (cycles)	frequency
intALU	1	45%
branch	2	5%
load	2	27%
store	1	8%
FPadd	4	7%
FPmult	6	8%

(a) (3 points) What is the CPI of execution (CPIe) for this machine?

ANSWER: CPIe = (0.45+0.08) + (0.05+0.27)2 + (0.07)4 + (0.08)6
= 1.93 cycles/instr

(b) (2 points) Now the wonderful group in the technology lab have provided the next generation of the chip with more transistors, and the VLSI people say that they can use these transistors to reduce the latency of the FPadd to 2 cycles and the FPmult to 4 cycles. What is the new CPIe?

ANSWER: CPIe = (0.45+0.08) + (0.05+0.27)2 + (0.07)2 + (0.08)4
= 1.63 cycles/instr

(c) (2 points) What is the resulting speedup, assuming no change in the cycle time?

ANSWER: speedup = 1.93 / 1.63 = 1.18x faster

(d) (5 points) Some other daring VLSI designers in the company claim that they can use the same extra transistors to implement some new, complex instructions. They claim that the FP, intALU, and memory port are essentially independent functional units, and thus, can operate in parallel. They suggest two new types of instructions: (1) an instruction that performs a FPop and a R-R intALU operation; and (2) an instruction that performs a register-indirect load/store and a R-R intALU operation. The bottom line is that 20% of the intALU operations can be collapsed with a load/store operation and that 12% of the intALU operations can be collapsed with a FP operation. What is the resulting CPIe of this technique? State any assumptions that you make.

ANSWER: CPIe = (0.45-0.20-0.12+0.08)/0.68 + ((0.05+0.27)/0.68)2
+ (0.07/0.68)4 + (0.08/0.68)6 = 2.37 cycles/instr

8

(e) (3 points) Again, assuming no change in the cycle time, which use of the extra transistors gives better performance (please show a speedup comparison)?

ANSWER: speedup = 1.93(1) / 2.37(0.68) = 1.20x faster
therefore (d) is better

2. (20 points total) *Memory System Design*

You've been given a promotion to memory system designer, and your company has the following questions for you.

(a) (4 points) The current OS uses a 4KB page size. Your boss is thinking about doing the virtual address translation in parallel with the instruction cache (Icache) access so that she can build a physical cache (i.e. no flushes or PIDs necessary). The VLSI designers tell you that you can have an Icache with an associativity of at most 5 ways. What is the biggest-sized, physical Icache that you can build? How many bits of virtual address (VA) are used to index into this Icache if the block size is 8 words (1 word = 32-bits)?

```
ANSWER: MaxSize = 5 * 4KB = 20KB
        index = 12 - 2 - 3 = 7 bits
```

(b) (2 points) Your boss's boss is really interested in building a 64-bit processor, i.e. a processor with a flat 64-bit virtual address space. If you designed the Icache as a virtually-tagged cache, how many bits of the 64-bit virtual address are kept in the tags if the data portion of the Icache is 32KB in size and only 2-way set associative?

```
ANSWER: tag = 64 - 15 + 1 = 50 bits
```

(c) (8 points) If we kept a 9-bit PID and a valid bit in the tag along with the rest of the virtual address, what percentage of the total Icache space is consumed by the tag array for a 32-bit VA and for a 64-bit VA? The data portion of the Icache is still 32KB in size and 2-way set associative. Please count only bits, not transistors, and ignore control logic.

```
ANSWER: set size = 32KB / (2 * 32B) = 512 entries => 1024 tags

32-bit VA: tag bits = 1024 * (18+9+1) = 28672 bits
           data bits = 32K * 8 = 262144 bits
               28672 / (28672+262144) = 9.9% overhead

64-bit VA: tag bits = 1024 * (50+9+1) = 61440 bits
           data bits = 32K * 8 = 262144 bits
                 61440 / (61440+262144) = 19.0% overhead
```

(d) (4 points) To fully support this 64-bit addressing, someone suggests adding 64-bit instructions to the instruction set architecture so that large immediates can be generated quickly. A problem with 64-bit instructions is that they can fall across Icache line boundaries because instructions only have to be word-aligned. If the Icache miss rate is 2%, the miss penalty is 12 cycles, and probability that a 64-bit instruction falls across an Icache boundary is 6%, what is the average memory access time (AMAT) for fetching an instruction? Assume that a 32-bit or a 64-bit

instruction can be fetched in 1 cycle if you hit in the Icache and if you do not cross a line boundary.

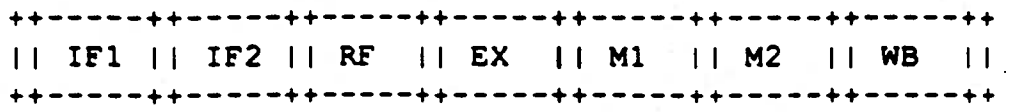
ANSWER: AMAT = 1 + (0.02)(12) + (0.06)(1) = 1.30 cycles

(e) (2 points) This splitting increases the AMAT of the Icache, but it also affects the virtual memory system. Assume your processor uses pages and a TLB, what is the ugly effect of these split instructions?

ANSWER: 2 TLB accesses are necessary for split fetches that miss meaning that a page fault can occur in the middle of an instruction miss. What do you do with the first half of the instruction fetch in the meantime?

3. (25 points total) *Pipelining*.

The pipeline shown below has been designed to work at a very high clock rate. The pipeline ticks twice as fast as the memory latency, although the memory is pipelined so that it returns an item on every clock tick. The consequence is that both the instruction fetch and the memory load/store operations have had to be split into two stages.




```
IF1   - first cycle of instruction fetch
IF2   - second cycle of instruction fetch
RF    - instruction decode and register fetch
EX    - ALUop OR memory address calcuation
M1    - first cycle of data memory load or store
M2    - second cycle of data memory load or store
WB    - write back result into register file
```

(a) (3 point) For such a high performance machine, is it a good idea to have the PC as one of the general purpose registers? If not, why not?

ANSWER: It is not a good idea because it interferes with the ability to efficiently pipeline the machine.

(b) (10 points) We are considering the tradeoffs between the use of delayed branches and squashing branches. For the squashing branch case, assume that the machine executes along the not-taken path. If the branch is actually taken, then the write-back and memory operations corresponding to the incorrectly fetched instructions are suppressed. For the non-squashing case, the following table gives the probabilities of filling the branch delay slots from code above the branch. If the probability that a branch is taken is 60%, which method do you recommend? Justify your decision quantitatively.

slot	probability of fill
1	75%
2	25%
3	5%
4	2%
5	1%

ANSWER: (NOTE this answer is for 3 branch delay slots, but 2 slots is reasonable)

```
CPI-branch-sq     = fraction_taken * 4 + fraction_not_taken * 1
                  = .6 * 4 + .4 = 2.8

CPI-branch-non-sq = 4 - (prob_slot1_full + prob_slot2_full
                    + prob_slot3_full)
                  = 4 - ( .75  + .25 + .05) = 2.95
```

Therefore, it is better to have squashing branches.

(c) (6 points) Assuming that the processor has no hardware interlocks, rewrite the code sequence below inserting the necessary NOPs to avoid hazards. Assume that branches are not squashing.

```
OR   R5,R6 -> R7
LD   1(R1) -> R4
ADD  R1,R5 -> R5
ADD  R3,R4 -> R2
ST   R5 -> 2(R1)
ADD  R7,R5 -> R4
BEQ  R2,R5,target
AND  R1,R6 -> R1
...

ANSWER:
OR   R5,R6 -> R7
LD   1(R1) -> R4
ADD  R1,R5 -> R5
NOP
ADD  R3,R4 -> R2
ST   R5 -> 2(R1)
ADD  R7,R5 -> R4
BEQ  R2,R5,target
NOP
NOP
NOP
AND  R1,R6 -> R1
...

or, without any by-passing:
OR   R5,R6 -> R7
LD   1(R1) -> R4
ADD  R1,R5 -> R5
NOP
NOP
NOP
ADD  R3,R4 -> R2
ST   R5 -> 2(R1)
ADD  R7,R5 -> R4
NOP
NOP
BEQ  R2,R5,target
NOP
```

```
NOP
NOP
AND R1,R6 -> R1
...
```

(d) (6 points) Reorganize the above code to use the fewest number of NOPs.

```
ANSWER:
LD   1(R1) -> R4
OR   R5,R6 -> R7
ADD  R1,R5 -> R5
ADD  R3,R4 -> R2
BEQ  R2,R5,target
ST   R5 -> 2(R1)
ADD  R7,R5 -> R4
NOP
AND  R1,R6 -> R1
...
```