Full Name:____________________________________

15-740/18-740, Spring 2017

Exam 1

March 9, 2017, 3:00pm-4:20pm

Instructions:

- Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer. A few pages of scratch paper are provided at the end of the text booklet, but your final answer should be written by the question.
- Show your work and discuss your answer. You will be graded more on your explanation than on your final answer.
- The exam has a maximum score of 20,000 points.
- The problems are of varying difficulty. The point value of each problem is indicated. Pile up the easy points quickly and then come back to the harder problems.
- This exam is CLOSED BOOK, CLOSED NOTES. You may use a calculator, but no networked devices (e.g., phones, laptops, etc.).

Do not write below this line

TODO: SCORE BREAKDOWN

Prefetching

Problem 1. (??? points total):

One way to reduce the number of cache misses is by *prefetching*, a mechanism by which caches load lines that have not been requested by the processor in anticipation of being needed. For example, the cache controller may detect that the processor is walking an array, and on each miss, bring both the requested line, and the following line, under the expectation that the processor will access it next.

A. How can prefetching help performance?

It moves some fetches off the critical path

B. How can prefetching hurt performance?

If the line is line is never use, but evicts a line that is used.

C. What basic principles do prefetchers use to predict which data will be needed in the future

Spatial locality (e.g., walking an array)
Heuristics to determine if its enough to warrant a prefetch (we didn't cover this)

D. Some have claimed that prefetching is not required for dynamic scheduling (i.e., out-of-order) CPUs with a large instruction window. Why do they make this argument?

OoO brings the loads up on their own.

E. ~~If a cache miss takes 100 cycles and loads constitute 25% of instructions, how large of an instruction window would be required to hide the latency of all loads?~~

F. All modern processors have prefetchers - why?

~130 is too big to be practical

Problem 2. (??? points total):

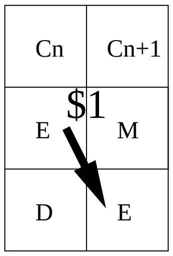
This question will test your understanding of how instructions flow through the simple 5-stage pipeline that we discussed in class. Recall that the five stages in the pipeline are: **IF** (instruction fetch), **ID** (instruction decode and register fetch), **EX** (execute), **MEM** (memory access), and **WB** (write back). Recall that Alpha instructions have the following semantics, where $X means *register* X:

`ldq $1,16($2)`	load the value from Memory[[$2]+16] into $1.
`stq $1,16($2)`	store the value from $1 into Memory[[$2]+16].
`addq $1,$2,$3`	$3 = $1 + $2.

Assume that data hazards are handled by *forwarding* (aka bypassing) whenever possible, using the EX-EX, MEM-EX, and MEM-MEM forwarding paths that we discussed in class. Whenever forwarding is insufficient, assume that *stalling* is used to resolve data hazards. You can also assume that there are no cache misses, and that data dependences through memory (e.g., a store to memory location *A* followed immediately by a load from *A*) do not cause any additional stalls.

In the diagram below, *indicate which instruction (if any) is in each pipeline stage* during a given cycle by filling in the appropriate box with one of the following: **F** (IF stage), **D** (ID stage), **E** (EX stage), **M** (MEM stage), or **W** (WB stage). If an instruction stalls, just fill in the same letter twice.

You should also *indicate all instances of forwarding* by drawing an arrow from the stage from which data is forwarded to the stage in which data is used. Also indicate the register being forwarded. Be careful to only indicate forwarding when an instruction reads data that another instruction is writing. As an example, the following diagram shows EX-EX forwarding of register $1:



Fill in your answers in the following table (note that we changed the program from the practice!):

Instr	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}
ldq $3, 0($2)	F	D	E	M	W								
ldq $5, 0($4)		F	D	E	M	W							
addq $3, $5, $5			F	D	D	$\xrightarrow{\$5}$ E	M	W					
stq $5, 0($3)				F	F	D	E	$\xrightarrow{\$5}$ M	W				
addq $3, $2, $2						F	D	E	M	W			
stq $2, 0($4)							F	D	E	$\xrightarrow{\$2}$ M	W		

VLIW

Problem 3. (??? points total):

VLIW processors such as Intel Itanium and many DSPs (digital signal processors) represent an alternative to superscalar and OoO for exploiting ILP. Their potential, but reputation for commercial failure, have led them to be described as ”always the processor of the future”.

A. Explain the basic design philosophy behind VLIW.

Get same ILP as OoO statically -> lower power, smaller, simpler
Move complexity to compiler

B. Give 2 advantages VLIW processors over superscalar/OoO. Why are these advantages particularly notable considering recent technology trends?

Lower power, smaller, simpler
Power wall, OoO don’t scale, batteries are limiting in may cases

C. Give 2 limitations that VLIW processors have relative to superscalar/OoO. Why have they failed to gain a foothold in the traditional server/PC/phone markets?

Can’t cope with dynamic situations as well
Poor binary compatibility
Needs better compilers

Poor binary compatibility
Compilers weren’t always up to the task

D. Consider a VLIW processor with 2 integer slots, 1 FP slot, and 1 memory slot. There are 32 integer registers (`r0-r31`) and 32 double-precision floating point registers (`f0-f31`). Instruction classes and delays are as follows (with all others having a one cycle latency):

add in multiply A[i] time constant split FP into adder and multiplayer

op	delay	class
ld	2	M
fp add	1	F
fp mult	2	F
integer op	1	I
branch	1	I

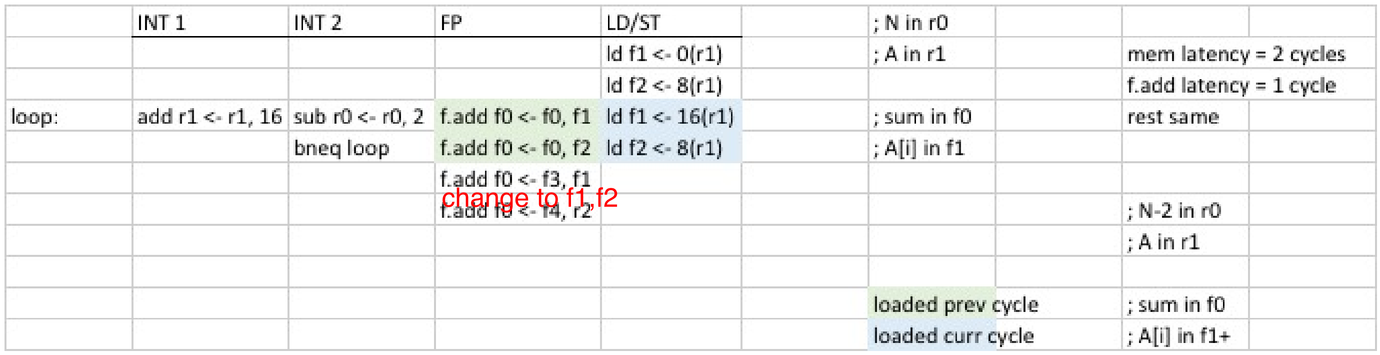
```
double sum_array(int N, double *A){
   int sum = 0;
    for(int i=0; i<N; i++){
        sum += A[i];
    }
}
```

have them write unrolled code first

Compile, optimize, and software pipeline the loop in the above program. Produce any reasonably unambiguous assembly. Make the target register/memory explicit with a syntax such as:

```
op target <- src1, src2
```

You only need to show the code for the loop, the required setup can be indicated by comments. E.g., `; r1 <- starts the loop with 0`. At the start of your compiled code use comments to specify the meaning of each register.



	INT 1	INT 2	FP	LD/ST		; N in r0			
				ld f1 <- 0(r1)		; A in r1		mem latency = 2 cycles	
				ld f2 <- 8(r1)				f.add latency = 1 cycle	
loop:	add r1 <- r1, 16	sub r0 <- r0, 2	f.add f0 <- f0, f1	ld f1 <- 16(r1)		; sum in f0		rest same	
		bneq loop	f.add f0 <- f0, f2	ld f2 <- 8(r1)		; A[i] in f1			
			f.add f0 <- f3, f1						
			change to f1,f2 f.add f0 <- f4, r2					; N-2 in r0	
								; A in r1	
						loaded prev cycle		; sum in f0	
						loaded curr cycle		; A[i] in f1+	

what advantage does software pipelining have over simple unrolling

E. DSPs are small, special purpose processors embedded in traditional computers; your laptop probably has one. They tend to have highly irregular instruction sets designed to maximize performance on the particular mathematical operations required for digital signal processing (naturally). The programs they run are special purpose, and often hand-written in assembly. Why might VLIW be acceptable for these processors, in contrast to more traditional processor domains.

Special purpose - binary compatibility isn't an issue
Very regular programs - compilation isn't as difficult, more naturally suited for VLIW

Scaling

Problem 4. (??? points total):

Big OoO processors provide high speed computation, but are large and power-inefficient. Furthermore, some studies have suggested that the largest contributer to their success is their improved code schedules, and *not* the ability dynamically change the schedule each time a piece of code is run. One mechanism proposed to take advantage of these properties is DynaMOS[1], which provides dynamic schedule migration on heterogeneous cores. Essentially, the first time a basic block (i.e., straight-line code, with only one entrance and exit) is run, it is run on the large, OoO core, and the schedule is recorded. From then on, it is run on a small, efficient, in-order core, with the previously recorded schedule. Their results suggested that this provides most of the benefits of OoO, with a large reduction in overhead.

A. What is an example of a scheduling change that an OoO processor can do, but that might be difficult (or impossible) to do statically?

Moving memory operations is tricky if you don't know if they can alias

Branch prediction benefits a lot from dynamic information

bold ^ safely ^ from the out of order core

B. An unmodified in-order core would not be able to run arbitrary traces. Give an example of a program and a code schedule that could not always be run on an unmodified core safely.

Ideal: Moving potentially aliasing memory accesses
(when one is a write)

Also acceptable: exceptions, mispredicted branches, etc

[1]Padmanabha, Shruti, et al. "Dynamos: dynamic schedule migration for heterogeneous cores." Proceedings of the 48th International Symposium on Microarchitecture. ACM, 2015.

C. Explain what mechanism(s) you would add to an in-order core to address these issues.

Ideal: keep record of original order, bail and run on OoO if it ends up
two memory accesses which end up aliasing's order was switch

Also acceptable: on exception, etc, bail and run on OoO

D. What kind of programs would not benefit from DynaMOS's scheme?

A program with unbiased branches, unpredictable memory behavior, etc

Parallelism

Problem 5. (??? points total):

Multithreading (SMT and multicore) and vector processors (including GPUs) are two approaches to providing explicit parallelism to the user (we are currently disregarding more advanced topics such as autovectorization).

A. How do the programming models differ for multithreading and vector processors?

Multithreading give each unit of parallelism its own thread of control (duh)
Vector processors do the same operation on each element of an array

B. What kind of program would you chose to run on a multithreaded processor? How about a vector processor? Explain your reasoning.

Multithreaded: one where tasks differ
Vector: uniformly performing operation on array

what are vector processors memory systems optimized for

C. Historically, vector machines were less likey to have caches than conventional processors. Why?

They can effectively hide latency and amortize startup cost

Scratch paper

Scratch paper

Scratch paper