Name: _____

Computer System Architecture
6.823 Quiz #5
December 14th, 2005
Professor Arvind
Dr. Joel Emer

Name:_____

This is a closed book, closed notes exam.
80 Minutes
15 Pages

Notes:

- Not all questions are of equal difficulty, so look over the entire exam and budget your time carefully.
- Please carefully state any assumptions you make.
- Please write your name on every page in the quiz.
- You must not discuss a quiz's contents with other students who have not yet taken the quiz.

Writing name on each sheet	_____	2 Points
Part A	_____	14 Points
Part B	_____	18 Points
Part C	_____	18 Points
Part D	_____	18 Points
Part E	_____	10 Points
TOTAL	_____	**80 Points**

Name: _____

Preamble

At the end of this quiz, there is a copy of this preamble that you can detach.

In this quiz, unless we indicate otherwise, we will use a single short C program, to ask you questions about performance of VLIW, Vector and Multithreaded machines. The program we will use is listed below:

C code

```
for (i=0; i<256; i++) {
  if (A[i] > 0)
    C[i] = A[i] * B[i];
}
```

Assembly Code

```
; initial values
; r1 = &A[0]
; r2 = &B[0]
; r3 = &C[0]
; r4 = 0

loop:
1.   l.s   f1, 0(r1)   ; f1 = A[i]
2.   blez  f1, skip    ; if (A[i]<=0) goto skip
3.   l.s   f2, 0(r2)   ; f2 = B[i]
4.   mul.s f3, f1, f2  ; f3 = f1 * f2
5.   s.s   f3, 0(r3)   ; C[i] = f3

skip:
6.   addi  r1, r1, 4   ; i++
7.   addi  r2, r2, 4
8.   addi  r3, r3, 4
9.   addi  r4, r4, 1
10.  slti  r5, r4, 256
11.  bnez  r5, loop    ; if (i < 256) loop
```

In all questions, you should assume that arrays **A**, **B** and **C** do not overlap in memory.

In each part or question, we will either provide you with appropriate corresponding assembly code or ask you to write one, but all of those programs will perform the computation described above.

Name: _____

Part A: Compiler Techniques (14 Points)

Question 1. (6 Points)

In examining the assembly code in the preamble, Ben Bitdiddle decides that it would be advantageous to move the load of `f2` (instruction 3) above the `blez` (instruction 2). Please explain the advantages of doing so or discuss why his thinking is wrong.

Name: _____

Question 2. (8 Points)

In order to eliminate branches, several architectures have a conditional move (`CMOV`) instruction of the form:

`CMOV<cond>Z Rd, Rs, Rt` (where `<cond> := EQ|NE|GT|LT|GE|LE`)

which operates, for example, as follows:

`CMOVEQZ Rd, Rs, Rt` # if (`Rs` == 0) then `Rd` <- `Rt`; otherwise, do nothing.

Please rewrite the assembly code using `CMOV` to eliminate the branch at instruction 2. Assume that no memory operations will generate an exception.

```
; initial values
; r1 = &A[0]; r2 = &B[0]; r3 = &C[0]; r4 = 0

loop:

# Start of your code









# End of your code

addi  r1, r1, 4   ; i++
addi  r2, r2, 4
addi  r3, r3, 4
addi  r4, r4, 1
slti  r5, r4, 256
bnez  r5, loop    ; if (i<256) loop
```

Name: _____

Part B: VLIW Machines (18 Points)

In this part, we will be rewriting the code in the preamble for the following single-issue in-order VLIW machine:

- 1 load/store units. A load has a latency of 2 cycles but is fully pipelined.
- 1 integer ALU/Branch unit. single cycle
- 1 floating-point add/multiplier. 3 cycles, fully pipelined
- 128 GPRs and 128 FPRs

A single instruction can issue to all the above units simultaneously. By definition, the operations in a VLIW instruction are independent. Every operation in a VLIW instruction reads the operands and issues simultaneously. Thus, if one operation is waiting for a result of a previous VLIW instruction, the entire VLIW instruction is stalled in the decode stage.

Everything is fully bypassed. Each functional unit has a dedicated writeback port, so there is never any contention. Writing to the same register multiple times in the same instruction is disallowed in this ISA. WAW hazards will also cause stalls. This ISA resembles MIPS, except that there can be up to 3 instructions on each line separated by semicolons. Here is an example.

```
addi r1, r1, 4; l.s f1, 0(r2); fadd f2, f2, f3
```

The branch unit has no delay slots and assumes 100% branch prediction accuracy

Question 3. (8 Points)

Frequently, VLIW architectures incorporate the notion of predication by adding predicate registers `p1`, `p2`, …, and allowing operation execution to be conditional on whether the predicate is true or not. We extend our VLIW architecture with a new set of predicated instructions as follows:

1) Augment the ISA with a set of 32 predicate bits `P0-P31`.
2) Every standard non-control instruction now has a predicated counterpart, with the following syntax: `(pbit1) OPERATION1; (pbit2) OPERATION2`
   (Execute the first operation of the VLIW instruction if `pbit1` is set and execute the second operation of the VLIW instruction if `pbit2` is set.)
3) Include a set of compare operations that conditionally set a predicate bit:

```
CMPLTZ pbit,reg     ; set pbit if reg < 0
CMPLEZ pbit,reg     ; set pbit if reg <= 0
CMPGTZ pbit,reg     ; set pbit if reg > 0
CMPGEZ pbit,reg     ; set pbit if reg >= 0
CMPEQZ pbit,reg     ; set pbit if reg == 0
CMPNEZ pbit,reg     ; set pbit if reg != 0
```

Rewrite the loop in the preamble using predication and fill out the table on the next page.

Name: _____

ALU/Branch	LSU	FADD/FMUL
`addi r4, r4, #1`	`l.s f1, 0(r1)`	

Table for Question 3 (to be filled out)

Name: _____

Question 4. (10 Points)

Unroll the loop of Question 3 four times and optimally schedule. (Don't do software pipelining.)

ALU/Branch	LSU	FADD/FMUL
	`l.s f1, 0(r1)`	
	`l.s f2, 4(r1)`	
	`l.s f3, 8(r1)`	
	`l.s f4, 12(r1)`	

Table for Question 4 (to be filled out)

Name: _____

Part C: Vector Machines (18 Points)

In this problem, we analyze the performance of vector machines. We start with a baseline vector processor with the following features:

- 32 elements per vector register
- 8 lanes
- One integer ALU per lane: 1 cycle latency
- One floating-point ADD/Compare per lane: 1 cycle latency
- One floating-point MULT per lane: 2 cycle latency, fully pipelined
- One LOAD/STORE unit per lane: 4 cycle latency, fully pipelined
- Register files have enough ports to serve all function units at the same time.
- No dead time / No support for chaining
- Scalar instructions execute on a separate 5-stage fully-bypassed pipeline
- Assume no branch delay slot and perfect branch prediction.

This architecture includes a vector mask that operates as follows:

1) Augment the ISA with a vector mask register, `VM`.
2) We have the second form of each vector instruction (`opcode.M`). An instruction of this form executes each element operation only if the corresponding bit in the mask register set
3) Masked operations are implemented by disabling register writeback.
4) Include compare operations that conditionally set the mask register:

`S<cond>V V1,V2`
`S<cond>SV V1,F1`

Compare the elements (where <cond> := `EQ|NE|GT|LT|GE|LE`) in `V1` and `V2`. If condition is true, put a 1 in the corresponding bit vector; otherwise put 0. Put resulting bit vector in vector-mask register (`VM`). The instruction `S<cond>SV` performs the same compare but using a scalar value as one operand.

Then the code in the preamble can be vectorized as follows:

Vectorized assembly code

```
; initial values: R1 = &A[0], R2 = &B[0], R3 = &C[0], R4 = 256, F0 = 0.0

       MTC1    VLR, 32          ; set VLR to 32
loop:
       LV      V1, R1           ; load A (unmasked)
       SLTSV   V1, F0           ; set up mask register
       LV.M    V2, R2           ; load B (masked)
       MUL.M   V3, V1, V2       ; C[i] = A[i] * B[i] (masked)
       SV.M    V3, R3           ; store C (masked)

       ADDI    R1, R1, 128      ; increment A ptr (32 words * 4 bytes/word)
       ADDI    R2, R2, 128      ; increment B ptr (32 words * 4 bytes/word)
       ADDI    R3, R3, 128      ; increment C ptr (32 words * 4 bytes/word)
       SUBI    R4, R4, 32       ; update loop counter
       BNEZ    R4, loop         ; if (R4!=0) loop
```

Name: ______________________

Question 5. (8 Points)

Complete the pipeline diagram below, with the vectorized loop code in the previous page for one loop iteration. Do not fill in scalar instructions. Assume the scalar registers are available immediately, whenever needed. You may not require the entire length of the table.

We assume a **magic memory system** with no bank conflicts and no cache misses. Also, scalar operands of vector instructions are read in the Decode stage.

The following **supplementary information** explains the diagram:

Scalar instructions execute in 5 cycles: fetch (**F**), decode (**D**), execute (**X**), memory (**M**), and writeback (**W**).
A vector instruction is also fetched (**F**) and decoded (**D**). Then, it stalls (**—**) until its required vector functional unit is available. With no chaining, a dependent vector instruction stalls until the previous instruction finishes writing back ALL of its elements. A vector instruction is pipelined across all the lanes in parallel. For each element, the operands are read (**R**) from the vector register file, the operation executes on the load/store unit (**M**) or the Adder/Comparator (**X**) or the MUL (**Y**), and the result is written back (**W**) to the vector register file. Assume that there is no structural conflict on the writeback port. A stalled vector instruction does not block a scalar instruction from executing.
LV_1 and LV_2 refer to the first and second LV instructions in the loop.
*Assume that both S<cond>V and S<cond>SV use the floating-point Adder/Comparator (**X**).*

Instr.	Cycle																																							
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
LV_1	F	D	R	M1	M2	M3	M4	W																																
LV_1				R	M1	M2	M3	M4	W																															
LV_1					R	M1	M2	M3	M4	W																														
LV_1						R	M1	M2	M3	M4	W																													

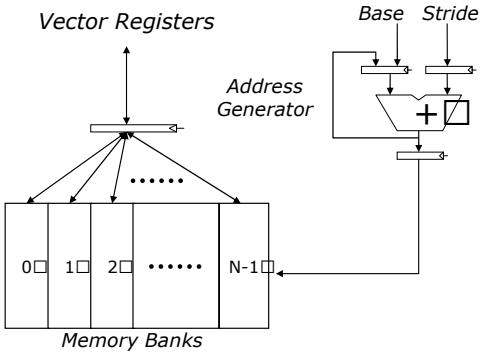
Name: _____

Question 6. (4 Points)

Note: This question does not pertain to the program in the preamble. Instead, assume a program that makes unit-strided accesses and whose performance is bottlenecked by the memory subsystem.

We now replace our magic memory system of the vector machine with a semi-magic banked memory system. Each bank i ($0 \leq i \leq$ N-1) contains the words with (word) addresses x where x mod N = i (N is the number of banks).

The banks are not pipelined and a bank is busy for 4 cycles to service a request. If we want the memory subsystem to be able to deliver 4 words per cycle, what is the minimum number of banks we need to sustain this peak performance?



Question 7. (6 Points)

If our program of Question 6 made memory accesses with the stride of two, how will the performance of your memory subsystem change?

Comment on the performance of your memory subsystem to memory accesses with other strides.

Name: _____

Part D: Multithreading (18 Points)

In this problem, we will analyze the performance of the program in the preamble on a multi-threaded architecture. Here is a summary of our multithreaded architecture:

- Single-issue in-order pipeline
- All instructions take three cycles (Fetch, Decode, Register file read) before execution.
- The fully pipelined execution of instructions take:
  A. 1 cycle for integer operations
  B. 1 cycle for branches to resolve
  C. 4 cycles for memory operations
  D. 3 cycles for floating point operations
- After execution all instructions take 1 cycle to write back their results (no bypassing).

We run the program in the preamble on this machine. It switches to a different thread every cycle using fixed round robin scheduling. Each of the N threads executes one instruction every N cycles. We allocate the code to the threads such that every thread executes every Nth iteration of the original C code (each thread increments `i` by N).

Question 8. (4 Points)

How many threads are needed to guarantee no bubbles will be inserted in the pipeline?

Name: _____

Question 9. (6 Points)

(1) For strict round-robin scheduling of 16 threads (this may or may not be the answer to Question 8) and assuming that 25% of the values in the array A are less than or equal to zero, how many instructions will each thread execute on average?

(2) For the best case distribution of values less than or equal to zero in the array A (still assuming that 25% of the values are ≤ 0), how long (in cycles) will it take for all the threads to finish? (You can measure the completion time from the insertion of the first instruction into the pipeline to the insertion of the last instruction into the pipeline.)

(3) For the worst case distribution of values less than or equal to zero in the array A (still assuming that 25% of the values are ≤ 0), how long (in cycles) will it take for all the threads to finish?

Name: _____

Question 10. (4 Points)

Ben Bittledittle has some multithreaded code that he needs to run. This code has a serial portion of 100,000 instructions and a parallel portion (parallelized 4 ways) of 100,000 instructions per thread that he is going to run on his 4-way multithreaded SMT processor. When running the serial portion he finds that the processor runs at 2 instructions per cycle (IPC). And when running the parallel portion it runs at a total of 3 IPC. What is the runtime of the program (in cycles)?

Question 11. (4 Points)

Compare the performance you found in Question 10 to the performance the Ben would observe on a 2-processor CMP (chip multiprocessor) in which each processor runs at 1.5 IPC.

Name: _____

Part E: Reliability (10 Points)

Recalling the definition of Architectural Vulnerability Factor (AVF) in Lecture 24 (2005) is the probability that a fault (bit flip) results in an architecturally visible error, we consider a store buffer. This store buffer consists of 64 32-bit data entries and a series of measurements have determined the average throughput and latency of various types of instructions below. Assume that all stores write all 32 bits of an entry.

Instruction Type	Throughput (Entries/cycle)	Latency (cycles)
Committed ACE*	2.0	14.0
Wrong-path	1.0	5.0
Dynamically dead	0.5	10.0

* ACE = required for architecturally correct execution

Question 12. (6 Points)

What is the Architectural Vulnerability Factor (AVF) of this store buffer?

Question 13. (4 Points)

If this machine were enhanced to support SMT, qualitatively how would you expect the AVF to change and why?

Name: _____

Copy of Preamble

You can detach this page for use during the quiz.

In this quiz, unless we indicate otherwise, we will use a single short C program, to ask you questions about performance of VLIW, Vector and Multithreaded machines. The program we will use is listed below:

C code

```
for (i=0; i<256; i++) {
  if (A[i] > 0)
    C[i] = A[i] * B[i];
}
```

Assembly Code

```
; initial values
; r1 = &A[0]
; r2 = &B[0]
; r3 = &C[0]
; r4 = 0

loop:
1.   ld    f1, 0(r1)    ; f1 = A[i]
2.   blez  f1, skip     ; if (A[i]<=0) goto skip
3.   ld    f2, 0(r2)    ; f2 = B[i]
4.   mul.d f3, f1, f2   ; f3 = f1 * f2
5.   sd    f3, 0(r3)    ; C[i] = f3

skip:
6.   addi  r1, r1, 8    ; i++
7.   addi  r2, r2, 8
8.   addi  r3, r3, 8
9.   addi  r4, r4, 1
10.  slti  r5, r4, 256
11.  bnez  r5, loop     ; if (i < 256) loop
```

In all questions, you should assume that arrays **A**, **B** and **C** do not overlap in memory.

In each part or question, we will either provide you with appropriate corresponding assembly code or ask you to write one, but all of those programs will perform the computation described above.