CSCE 513 Computer Architecture **Test 1** **October 2, 2013**

Name:_____ Email:_____

Instructions

- **No Calculators!!**
- Your notes on one side of an 8.5 x 11 sheet of paper should be signed and submitted with your test.
- Make sure your exam is complete.
- No Calculators, cell phones, or other electronic devices.
- All questions are equally weighted.
- Answer in the space provided if at all possible.
- If a question is unclear please ask early in the test.
- There is a Take Home question. It will be emailed today.
- Good Luck!

1.

Answer: 2 points each

(a) What is Mem/WB.LMD ?

Answer: The is the field in the Mem/WB pipeline register that stores the data loaded from memory (Load memory Data).

(b) Why does it make sense to give reads priority over Writes? What is necessary to allow this? That is what is added to caches to allow this?

Answer: The CPU needs to wait on the data from a load (read), but on a store(write) the CPU can proceed without waiting on the write to complete.
To allow writes to be stored until later caches typically will have write-buffers.

(c) What is Moore's law?

Answer: In 1965 from previous growth data Moore extrapolated the trend to 1975 and projected that the number of components per chip would reach 65,000; a doubling every 12 months.
At the 1975 IEEE International Electron Devices Meeting Moore he slowed the future rate of increase in complexity to "a doubling every two years, rather than every year."
http://www.computerhistory.org/semiconductor/timeline/1965-Moore.html

(d) Explain critical word first and early restart.

Answer: In transferring a block from the slower cacher it if frequently the case that the slower cache is divided into banks and the block is spread across all the banks. Then when a block is read, all segments of the block can be read in parallel, but then are transfered sequentially P0, P1,...Pk, where Pi is the ith segment of the block we are bringing in.
In early restart as soon as the portion of the block that has the data that the CPU needs is received the CPU can restart without waiting on the rest of the block.
In critical word first the block segments are not sent in linear order P0, P1 ... but the block containing the data that the CPU needs is sent first then the remaining blocks are sent linearly. In critical word first it also does early restart.

(e) What is meant by way-prediction?

Answer: In way predition the select inputs to the multiplexer are preset to select ''the way'' and guess which tag will match and thus which block in the set will be transferred. This presetting saves a small bit of time in the case of a hit, thus reducing the Hit-Time. If the guess is wrong it will actually slow things down.

(f) What is meant by statically scheduling code?

Answer: Scheduling is the rearrangement of assembly code (actually machine code) without changing the semantics of the program to avoid stalls in the pipeline.
Static scheduling is when the compiler does this Dynamic Scheduling is when the hardware does this as in Tomasulo's or ReOrder Buffer.

2. Classical 5 stage pipeline: Assuming the classical 5-stage pipeline with no forwarding except through the registers. Assume all of these instructions are integer and execute in 1 cycle. Given the code below:

```
loop1: LD      F6, 0(R1)
loop2: LD       F8, 0(R2)
       ADD     F6, F6, F8
      DADDIU  R2, R2, +8
      BNE     R2, R6, loop2
      DADDIU  R1, R1, +8
      BNE     R1, R4, loop1
```

   (a) (8 pts)Show how the first two iterations of the inner loop would proceed through the pipeline. Stop with the fetch of the first instruction of the inner loop on the third iteration or when you fill the table. Assume forwarding only through the registers and that you predict branch not taken. Also, state any assumptions you make about when the branch target address is computed.

```
Answer:    I will assume that the branch target address is
calculated in the Decode stage with an extra adder and the
branch target address is routed around to the Instruction
Memory so that in the cycle after the BNE finally finishes
the decode stage in the next cycle we correctly fetch the
target [LD F8, 0(R2)].
```

Instruction		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
LD	F6, 0(R1)	F	D	E	M	W															
LD	F8, 0(R2)		F	D	E	M	W														
ADD	F5, F6, F8			F	D	stall	stall	E	M	W											
DADDUI	R2,R2,+8				F	stall	stall	D	E	M	W										
BNE	R1,R4,loop							F	D	stall	stall	E	M	W							
DADDIU	r1,R1,+8								F	stall	stall	-	-	-	-						
LD	F8, 0(R2)											F	D	E	M	W					
ADD	F5, F6, F8												F	D	stall	stall	E	M	W		
DADDUI	R2,R2,+8													F	stall	stall	D	E	M	W	
BNE	R1,R4,loop																F	D	stall	stall	E

   (b) (2 points) How many cycles does the two iterations take.

```
Answer:    The BNE is decided in its D stage, but after the
Fetch there are two stalls waiting on R2.  In cycle 19 this
value is written back by the DADDIU, and read by the BNE and
thus in cycle 20 we fetch the LD for the 3rd iteration of
the loop.  Since the inner loop starts in Cycle 2 and goes
to 19, it needs 18 cycles.
```

   (c) (3 points) Identify a place where forwarding could help and explain in detail how the forwarding condition would be recognized and which pipeline register/field the data would be forwarded from.

```
Answer:    In the ''LD F8, 0(R2)'' followed by ''ADD
F5,F6,F8'' there is a load-use hazard but we can eliminate
one stall (compared to forward through the register) by
passing the result from Mem/WB.LMD to ALUinputB.
Detailed conditions are:
 i. Mem/WB.IR[rt] == ID/EX.IR[rt], note rt in the ADD is an
    operand and is the target in the LD
 ii. Mem/WB.IR[opcode] == LOAD
iii. ID/EX.IR[opcode] == ALU opcode or ALU-Immediate opcode
```

3. (a) In the original pipeline approach to branches how was the branch target address calculated.

Answer:
$Ex/Mem.ALUoutput \leftarrow \ NPC \ + \ SignExtend(Imm << 2)$
where the addition is done by the ALU in the Execute Stage.
Actually it is
$Ex/Mem.ALUoutput \leftarrow \ ID/EX.NPC \ + \ (ID/EX.SignExt) << 2$
where the values on the right-hand side were latched into ID/EX fields during the decode stage.

(b) Later we improved on this by adding what and to what stage?

Answer: An adder plus additional circuitry to determine whether the branch should be taken were added to the decode stage.

(c) Draw the diagram of a 3-bit saturating counter branch predictor.

Answer: picture to appear later; 8 states from MostStronglyTaken TTT to Most StronglyNotTaken NNN; arcs labelled either Taken (move towards MostStronglyTaken) or NotTaken(move towards MostStronglyNotTaken).

(d) If a loop executes 1000 times, has only the branch at the bottom of the loop and starts out in the most strongly not taken state how many branches are mispredicted?

Answer: The behavior followed by the branch at the bottom of this loop is: to branch the first 999 followed by not taking the branch in the last iteration.
So starting in StronglyNotTaken(NNN):
In iteration 1 (the branch at the bottom) in State (NNN) misspredicts BranchNotTaken moving one state towards ``MostStronglyTaken'' to (NNT),
In iteration 2 in state (NNT) misspredicts BranchNotTaken moving (NTN),
In iteration 3 in state (NTN) misspredicts BranchNotTaken moving (NTT),
In iteration 4 in state (NTT) misspredicts BranchNotTaken moving (TNN),
Finally in iteration 5 in state (TNN) the branch predictor correctly predicts BranchTaken moves to state (TNT),
And in iterations 6-999 the predictor correctly predicts BranchTaken.
In the last iteration the branch is incorrectly predicted as BranchTaken.
So 5 miss-predictions, 4 at the start then one at the end.

4. Given an application that runs in 1 minute and executes 1G instructions.

   (a) If the applicaton is 80% parallelizable (both time and instructions) and you have 10 processors how fast will the application run?

   Answer:

$$\begin{aligned} ExecTime_{new} &= (1 - Frac_{parallel}) * ExecTime_{old} + [Frac_{parallel}) * ExecTime_{old}]/processors \\ &= (1 - .8) * 1minutes + * .8 * 1minutes/10 \\ &= .2 + .08 = .28\ minutes \end{aligned}$$

   If you just plugged into Amdahl's law and said the answer was 1/.28 or even divided and said 3.5 then you lost 2.5 points along with about half the class.

   (b) If the CPI of the sequential portion of the program equals the CPI of the parallelizable part and we scaled the problem without changing the serial fraction. This means we worked a larger problem and took the same amount of time (1 minute) with the 100 processors. What number of instructions that could be executed with 100 processors in that 1 minute?

   Answer:

$$\begin{aligned} NumberOfInstructions_{SerialPortion} &= .2 * (1G) \\ NumberOfInstructions_{ParallelSectionByOneProcessor} &= .8 * (1G) \\ NumInstr_{new} &= .2G + .8G * 100 \\ &= 80.2Ginstructions \end{aligned}$$

5. AMAT: In the system we are analyzing the memory has:

   - Separate L1 instruction and data caches, HitTime = Processor Cycle Time
   - 32KB L1 instruction cache with 1% miss rate, 64B blocks
   - 256KB L1 data cache with 10% miss rate, 16B blocks
   - 256K L2 unified cache with 64B blocks, local miss rate 20%, Hit Time = 6 cycles,
   - Main Memory Access time is 50 cycles for the first 64 bits and subsequent 64 bit chunks are available every 4 cycles.
   - Both L1 caches are direct mapped, L2 four-way associative.
   - Assume there are no misses to main memory.

   (a) Just for this subproblem assume the L1-data miss penalty is 14 cycles and ignore the information about Instruction cache, L2 and main memory. In this case what is the AMAT for data references?

   Answer:

   $$\begin{aligned} AMAT &= HitTime_{L1-data} + MissRate_{L1-data} * MissPenalty_{L1-data} \\ AMAT &= 1 + .1 * (14)\ cycles \\ AMAT &= 2.4\ cycles \end{aligned}$$

   (b) What is the Miss Penalty for accesses to L2?

   Answer: 64bits=8B transferred at a time, so there is a total of 64B/8B = 8 chunks transferred

   $$\begin{aligned} MissPenalty_{L2} &= HitTime_{main-memory} \\ HitTime_{main-memory} &= initial\ chunk\ +\ 7\ remaining\ chunks \\ HitTime_{main-memory} &= 50 + 7 * 4\ cycles \\ HitTime_{main-memory} &= 78\ cycles \end{aligned}$$

   $$MissPenalty_{L2} = 78\ cycles$$

   (c) What is the average memory access time for instruction references?

   Answer:

   $$\begin{aligned} AMAT &= HitTime_{L1-Instr} + MissRate_{L1-Instr} * MissPenalty_{L1-Instr} \\ AMAT &= HitTime + MR_{L1-Instr} * (HitTime_{L2} + MR_{L2} * MP_{L2}) \\ AMAT_{Instruction} &= 1 + MR_{L1-Instr} * (HitTime_{L2} + MR_{L2} * MP_{L2}) \\ AMAT_{Instruction} &= 1 + .01 * (6 + .20 * 78) \\ AMAT_{Instruction} &= 1 + .01 * (6 + 15.6) \\ AMAT_{Instruction} &= 1.216 \end{aligned}$$

   (d) Assume the only memory reference instructions are loads(30%) and stores(5%). What percentage of total memory references are data references?

   Answer: Let #instructions = I then #datareferences = .35*I

   So the %total memory references that are data references is

   $\frac{DataReferences}{TotalReferences} = \frac{.35*I}{I+.35*I} = \frac{.35}{1.35} = .26 = 26\%$

(e) What is the Average memory access time? Use the AMAT-data from part a.

`Answer:`

$$\begin{aligned} AMAT_{overall} &= \%InstrRef * AMAT_{Instr} + \%DataRef * AMAT_{Data} \\ AMAT_{overall} &= (1 - .26) * AMAT_{Instr} + .26 * AMAT_{Data} \end{aligned}$$

6. Virtual Memory/TLB - Given

   - 40 bit virtual addresses
   - the page size is 4KB,
   - 32 bit physical addresses
   - 128KB cache, only one level unified, 8 way associativity, 256B lines
   - 128 entry TLB, 4 way associative
   - If the virtual address is 0x0F F123 4567 and if the physical page number is 0x322 with leading zeroes not shown

   (a) What is the page offset field?

   Answer:
   page size = 4K = $2^2 * 2^{10} = 2^{12}$
   pageOffsetFieldSize = $log_2 2^{12}$ = 12 bits = 3 hex digits
   PageOffset = 0x567

   (b) What is the VPN?

   Answer: VPN = the rest of the Virtual Address = 0x0FF1234

   (c) How big is a PPN?

   Answer: 32 bits - 12 bits = 20 bits

   (d) What is the physical address?

   Answer: PA= PhysicalPageNumber +[concatenated] PageOffset
   = 0x322 + 0x567 = 0x322567

   (e) What is the the cache "block offset" field?

   Answer: B = cache block size = 256B,
   so b = cacheOffsetFieldSize = $log_2 256$ =8 bits,
   and cacheOffset = lowest 8 bits = 0x67

   (f) What is "set-index" field?

   Answer:
   C = cachesize = 128K
   A = associativity = 8
   L = num_lines =C/B
   L = C/B = 128K/256 = 2^9
   S = num_sets = $2^9/8 = 2^9/2^3 = 2^6 = 64$
   s = setIndexFieldSize = $log_2 S = log_2 2^6 = 6$ bits num_sets =
   num_lines/associativity = $2^{10}/2^4 = 2^6 = 64$
   sizeOfSetIndex = $log_2(2^6) = 6$ bits
   setIndex = bits 8, 9, 10, 11, 12, 13 of PPN
   = low order 6 bits of 0x3225 = $\cdots$ 0011 0010 0010 0101= 10
   0101 = 0x025

   (g) What is the cache "tag" field?

   Answer:

$$
\begin{aligned}
cache\_tag &= \text{the high order } 32-8-6 = 18 \text{ bits} \\
PhysicalAddress &= 0x322567 \\
&= 0000\ 0000\ 0011\ 0010\ 0010\ 0101\ 0110\ 0111 \\
cache\_tag &= 0000\ 0000\ 0011\ 0010\ 00 \\
cache\_tag &= 000000000011001000 \\
cache\_tag &= 00\ 0000\ 0000\ 1100\ 1000 \\
cache\_tag &= 0x000C8
\end{aligned}
$$

(h) What happens on a TLB miss?

```
Answer:    The virtual address is looked up in the page
table and the page-table entry (= TBL-block) is transferred
to the TLB. If it is not found in the page table then a page
fault occurs.
```

7. Given the code below and a direct mapped cache with 256 lines of 32 bytes.

```
float a[4096];
double sum = 0.0;
for(i=0; i < 4096; ++i)
      sum = sum + a[i];
```

Assume that the non-array variables are stored in registers and ignore instruction references.

(a) How many array elements fit in a block?

Answer: Floats are 4B, blocksize = 32B therefore 8 elements per block.

(b) What is the hit ratio?

Answer: In processing the array you move straight through the array in row-major order. Each block (group of 8 elements) will have a cold miss to the first element in the block and then seven hits. HitRatio = 7/8 = 87.5%

(c) What is the hit ratio is we assume 128B blocks?

Answer: If the block 128B/4B = 32, so when you have the first cold miss that loads the block into the cache then you have 31 straight hits. So the hit ratio = 31/32 = 96.75%

8. Forwarding

   (a) In full forwarding data is forwarded from a number of locations. What are they?

   Answer: Figure C.26

   - EX/MEM.IR[ALUoutput]
   - MEM/WB.IR[ALUoutput]
   - MEM/WB.IR[LMD]

   (b) What kind of circuit implements EX/MEM.IR[rd] == ID/EX.IR[rs]?

   Answer: A 5-bit comparator

   (c) What other conditions in addition to the EX/MEM.IR[rd] == ID/EX.IR[rs] indicate that we need to do some forwarding?

   Answer: Figure C.26

   - EX/MEM.IR[opcode] == Reg-Reg ALU or ALU immediate, an instruction writing rd, and
   - ID/EX.IR[opcode] == Reg-Reg ALU, ALU immediate, load, store, branch, an instruction needing rs= EX/MEM.IR[rd] as an operand

   (d) Give an example of a code that would make this type of forwarding occur

   Answer:

```
DADD R1, R2, R3
DADD R4, R1, R5
```

9. Extra credit: What IEEE 754 float does 0 1000 0000 000 0000 0000 ... 0000 represent?

   Answer: SignBit = 0, $\Rightarrow$ float is positive,
   Exponent-Field = 1000 0000 = 128,
   ActualExponent = 128 - BIAS = 128-127 = 1
   fraction = .000 $\cdots 0000_2$ and so
   Mantissa = 1.000 $\cdots 0000_2$ and
   Value = +1.00 $\star 2^1$ = $+2.0_2$ = 2.0_{10}