15-740/18-740 Computer Architecture, Fall 2011 Midterm Exam I

Instructor: Onur Mutlu

Teaching Assistants: Justin Meza, Yoongu Kim

Date: October 24, 2011

Name:			
	Problem I (50 points)	:	
	Problem II (12 points)	:	
	Problem III (30 points)	:	
	Problem IV (35 points)	:	
	Problem V (20 points)	:	
	Problem VI (15 points)	:	
	Bonus Problem VII (18 points)	:	
	Legibility and Name (5 points)	:	
	Total $(167 + 18 \text{ points})$:	

Instructions:

- 1. This is a closed book exam. You are allowed to have one letter-sized cheat sheet.
- 2. No electronic devices may be used.
- 3. This exam lasts 1 hour 50 minutes.
- 4. Clearly indicate your final answer.
- 5. Please show your work when needed.
- 6. Please write your initials on every odd page.
- 7. Please make sure that your answers to all questions (and all supporting work that is required) are contained in the space provided.
- Be cognizant of time. Do not spend too much time on one question.
- f. Be concise. You will be penalized for verbosity.
- . Show work when needed. You will receive partial credit.
- . Write legibly. Show your final answer.

I. Potpourri (50 points)

1.	Static	vs.	Dyna	amic	Sched	luling

In class, we discussed that the compiler can reorder instructions dynamically to minimize the effect of
data dependencies on execution time. However, it is limited in its ability to do so. Give three reasons
as to why this is the case (each reason < 15 words):
2. MIPS
The original MIPS architecture design did not have hardware interlocks between pipeline stages. Hence the name MIPS, Microprocessor without Interlocking Pipeline Stages. Assume a four-stage pipelined implementation of MIPS, with the pipeline stages we discussed in class: F, D, E, SR. Assume no data forwarding between stages.
add R1 <- R1, R2
add R1 <- R1, R3 add R1 <- R1, R4
add R1 <- R1, R5
a) Whose task is it to guarantee that the above piece of code executes correctly? (< 10 words.)
b) How so? Explain. Give the resulting code if there need to be changes to the code.
3. Addressing Modes
a) What is the benefit of the autoincrement addressing mode? (< 15 words.)

b) What kind of data structures and access patterns would benefit from this mode? (< 30 words.)

Initials:	3
4. CRAY	
CRAY-1 was a vector processor. Yet, it had the fastest scalar unit of all processing designed. Explain why the designers made this choice. (< 20 words.)	cocessors at the time it was
5. Fine-Grained Multithreading	_
In lecture, we discussed the idea of fine-grained multithreading as a techni	ique to tolerate latency.
a) Which of the following latencies can fine-grained multithreading tolerat Explain very briefly (< 20 words).	e in a pipelined processor?
Control-flow dependencies (Please circle YES or NO.)	
YES NO	
Data dependencies (Please circle YES or NO.)	
YES NO	
Resource contention latency (Please circle YES or NO.)	
YES NO	
	1 4 2
b) Does fine-grained multithreading have any benefit in terms of hardward	e complexity?
YES NO	
Explain why so (< 20 words).	

c) Does fine-grained multithreading have any downside in terms of hardware complexity?
YES NO
Explain why so (< 20 words).
6. Interconnect
a) Company X is designing a multi-core processor with 100 cores. Their initial design connects the cores with a single shared bus. Give two reasons as to why this is not a good idea. (< 20 words each.)
b) An engineer suggests using a crossbar to connect each core to every other, instead. Give two reasons as to why this is a good idea. (< 20 words each.)

Give two reasons as to why this is not a good idea. (< 20 words each.)

5

7. Exceptions

Initials:

Consider the following state of the reorder buffer in an out-of-order execution engine:

ROB-ID	OP	COMPLETE?	EXCEPTION?
1	ld	NO	NO
2	add	NO	NO
3	br	NO	NO
4	add	YES	NO
5	ld	YES	YES
6	mul	NO	NO
7	ld	NO	NO

Assume the load in ROB-ID 5 generated an access protection exception because it references a NUI pointer. Give two reasons why this could be the case (< 20 words each):

8. Hybrid Main Memory

Justin gave a lecture on some recent work in hybrid main memory data placement policies. A hybrid main memory consists of a small amount of DRAM which acts as a cache to a large amount of non-volatile memory (NVM), and Justin discussed the trade offs of storing certain types of data on either device. Lets say you run a program on a hybrid main memory system and observe that (1) the total number of memory accesses is large and (2) the DRAM cache miss rate is large (meaning not many accesses are serviced from DRAM), but the program is hardly slowed down. What characteristics could the program have for this to occur (be specific)? (< 20 words.)

could the program have for t	his to occur (be specific)? (< 20 words.)	

9. Accelerated Critical Sections

Justin and Yoongu run their favorite parallel application, ResearchIdeaGenerator, on the same ma
chine with 16 symmetric cores with one difference: it either executes each critical sections on the
core that encounters the critical section or it executes every critical section on one core that is ded
icated to the execution of critical sections and serial bottlenecks. They measure the performance
of ResearchIdeaGenerator and find that the first machine completes the application in 240 minute
whereas the second in 300 minutes. They dig deeper into the problem and find that the cache miss rate
is significantly higher in the second machine than in the first and that is the cause of the performance
degradation. Provide two reasons as to why this could be the case (< 20 words each):

II. Branch Misprediction Penalty (12 points)

Assume a baseline in-order fetch, in-order dispatch processor design has four pipeline stages corresponding to fetch, decode, execute, and writeback; only executes arithmetic and branch instructions; and all instructions only require one execute pipeline stage. Assume you are benchmarking the performance of a program which is composed of 80% arithmetic instructions and 20% branch instructions.

a) What is the misprediction penalty (in number of cycles) for this processor design?
b) Calculate the throughput of the baseline processor design in terms of retired (written back) instructions per cycle (IPC) assuming no branches are mispredicted by the branch predictor.
c) Calculate the IPC assuming 50% of the branches are mispredicted by the branch predictor.
One alternative design might be to change the fetch stage into two stages as well as change the decode stage into two stages (for a total of six stages in the new pipeline) in order to increase frequency.
d) What misprediction rate would your branch predictor on this new six-stage pipeline design need to have in order to achieve the same IPC as the baseline four-stage pipeline design (still assuming the baseline design utilizes a 50% accurate branch predictor)?

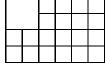
III. Amdahl's Law (30 points)

Let us assume that you are designing a multi-core processor to be fabricated on a fixed silicon die area budget of A. As the architect, you are to partition this total area of A into one "strong" core and many "weak" cores. The strong core will have an area of S, while the weak cores will each have an area of 1 (but there will be A-S of them). Assume that the single-thread performance of a core scales with the square root of its area.

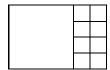
1. "Strong" vs. "Weak"

On this multiprocessor, we will execute a workload where P fraction of its work is *infinitely* parallelizable, and where 1 - P of its work is serial.

•	Config	gur	atio	on	X:	A =	24,	S =	= 4 ((One	stroi	ng	core,	20	weak	cores	;.)



• Configuration Y: A = 24, S = 16 (One strong core, eight weak cores.)



Assume that the serial portion of the workload executes only on the strong core and the parallel portion of the workload executes on both the strong and weak cores.

a) \	What is t	he speedup	of the	workload	on co	nfiguration	X?	(Compared	to the	e execution	time	on a
$\sin g$	le-core pi	rocessor of a	area 1.)									

\ XX 71	an and un of the	11 1	C 1: 37	2 (0 1 4	.1	, •

b) What is the speedup of the workload on configuration Y? (Compared to the execution time on a single-core processor of area 1.)

9

c) For what range of P does the workload run faster on Y than X?
d) For workloads that have limited parallelism, which configuration would you recommend and why? (< 20 words.)
2. Dual-Core Execution
In class, we discussed some ways in which multiple cores can be utilized to emulate the single-thread performance of a larger core. For this question, we assume that two strong cores can operate in a "dual-core execution" manner to achieve the single-thread performance of an even "stronger" core that is $1.5 \times$ faster than any single strong core.
• Configuration DCE: $A=24, S_1=4, S_2=4$ (Two strong cores, sixteen weak cores.)
Assume that dual-core execution occurs only during the serial portion of the workload and that during the parallel portion, the two strong cores separate from each other (on-the-fly) and operate as two independent cores. Furthermore, the serial portion executes only on the "dual-core", and the parallel portion executes on all the cores.
a) What is the speedup of the workload on configuration DCE? (Compared to the execution time on a single-core processor of area 1.)

b) For what range of P does the workload run faster on DCE than X?
c) Suppose you are executing workloads with a wide range of parallelism. Which configuration would you choose? DCE or Y? Why?

IV. Reorder Buffer & Register Alias Table (35 points)

In class and the review session, we covered the operation of an out-of-order execution engine that employs a reorder buffer (ROB), a consolidated physical register file, frontend and backend (retirement) register alias tables, and checkpointing. The figure below shows the state of these structures after the last multiply instruction at ROB-ID 10 is fetched and renamed. Only the relevant fields of each entry of each structure are shown.

Reorder Buffer							
ROB-ID	Valid	OP	Dest. PhyReg	Completed			
1	1	ld	r1	p55	0		
2	1	add		p57			
3	1	mul					
4	1	br	-	-	0		
5	1	add	r1				
6	1	mul	r5				
7	1	br	-	-	0		
8	1	add					
9	1	br	-	-	0		
10	1	mul	r7				

	$\begin{array}{c} {\bf Backend} \\ {\bf RAT} \end{array}$
	PhyReg
r1	p1
r2	p2
r3	р3
r4	p4
r5	p5
r6	p6
r7	p7
r8	p8

	\mathbf{R}	AT1	\mathbf{R}	AT2	\mathbf{R}	AT3		\mathbf{R}	AT4
	Rdy	PhyReg	Rdy	PhyReg	Rdy	PhyReg		Rdy	PhyReg
r1	1	p80			0	p57		1	p80
r2	1	p2	1	p2	1	p2		1	p2
r3	0	p100			1	p61		1	p61
r4	1	p4	1	p4	1	p4		1	p4
r5	0	p90			1	p5		0	p90
r6	1	p6	1	p6	1	p6		1	p6
r7	1	p7	0	p88	1	p7		1	p7
r8	1	p8	1	p8	1	p8		1	p8
							-		

The figure was drawn by a computer architect who was tracing the operation of a program. Unfortunately, his playful kid erased parts of the state using whiteout (marked with gray boxes in your handout). Your task: complete the entire state by filling in the gray boxes. First, start with answering the following questions.

a) How many architectural registers does the depicted machine have?

Four register alias tables (RATs) are depicted in the frontend. These four correspond to the current "Frontend RAT" which is used for renaming incoming instructions as well as three checkpointed RATs. A checkpoint of the frontend RAT was taken when each of the branches was renamed (hence, three checkpoints since there are three branches in the ROB). Remember that each entry in each RAT contains a "Ready (R)" bit (0 indicates "not ready"; as you recall, this bit is also called the "Valid" bit) and a tag corresponding to the physical register ID.



There is also a backend (retirement) RAT depicted, but it does not contain any "Ready" bits.

c) Why does the backend RAT not contain a ready bit for each register? (< 20 words.)

We have not told you which of the RATs correspond to the "Frontend RAT" and the three checkpoints. Given all you know about the state of the machine, your task is to identify which RAT corresponds to these.

- d) In the box under each RAT, please identify the RAT as one of the following:
 - "Frontend RAT" (current)
 - "CP1" (Checkpointed RAT corresponding to the first branch Branch at ROB-ID 4)
 - "CP2" (Checkpointed RAT corresponding to the second branch Branch at ROB-ID 7)
 - "CP3" (Checkpointed RAT corresponding to the third branch Branch at ROB-ID 9)

Note that the RATs' order of apperance on paper does not correspond to any timeline.

- e) Now, go ahead and fill all the gray boxes in the figure. (If you make a mess, clearly indicate your final answer. Use blank sheets if needed.)
- f) Show the state of the Backend RAT after the third branch (at ROB-ID 9) is retired (assuming all branches were predicted correctly and there are no exceptions).

	Backend RAT
	PhyReg
r1	
r2	
r3	
r4	
r5	
r6	
r7	
r8	

Now, assume instead that the second branch (at ROB-ID 7) was mispredicted. Also assume its misprediction is resolved long before the load is completed and the multiply at ROB-ID 6 is dependent on the load. Immediately after the branch is resolved, the state is recovered and the processor starts fetching from the correct path.

g) Show the state of the Frontend RAT immediately after misprediction recovery and before the first instruction on the correct path is renamed.

	Frontend RAT					
	Rdy	PhyReg				
r1						
r2						
r3						
r4						
r5						

 $^{\rm r6}_{\rm r7}$

h) Given the above information, can you construct the dataflow graph of the code that is currently in the ROB? If so, draw the dataflow graph below. If not, why not? Tell us what other structures' state would you need to see in the machine to be able to construct the data flow graph. (<20 words.)

V. Tomasulo's Algorithm (20 points)

For this question, we'll consider an in-order fetch, out-of-order dispatch processor with fetch, decode, execute, reorder, and writeback stages, and a future register file. Our system has two functional units: an adder that has three pipeline stages, and a multiplier that has five pipeline stages. Assume that tags and data are broadcast on the common data bus during the last cycle of the execute stage. During cycle 0, we fetch the first of six instructions on our processor. We show the state of the future file at cycle 0 below as well as the state of the machine at cycle 9. ("—" is used to indicate information that is unknown.)

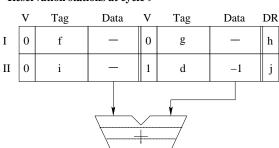
Future register file at cycle 0

	V	Tag	Data
r0	1	с	3
r1	1	b	5
r2	1	Z	16
r3	1	x	0
r4	1	y	100
r5	1	d	-1
r6	1	a	2

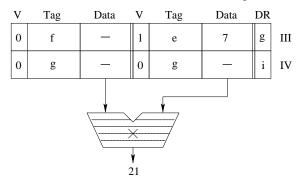
Future register file at cycle 9

	V	Tag	Data
r0	1	c	3
r1	1	b	5
r2	0	f	_
r3	1	e	7
r4	0	σρ	_
r5	1	d	-1
r6	0	j	_

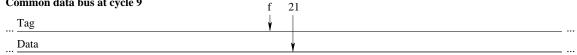
Reservation stations at cycle 9



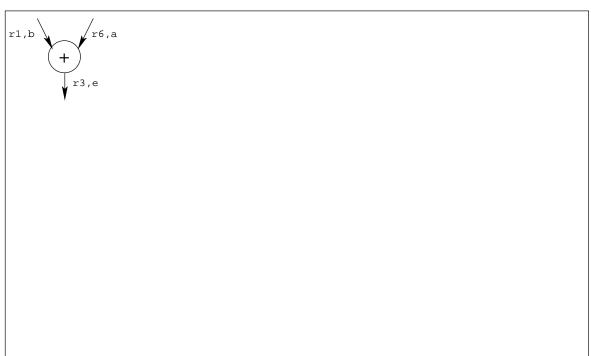
V = ValidDR = Destination Register







a) Your first task is to use only the supplied information to draw the data flow graph for the six instructions which have been fetched. Label nodes with the operation being performed and edges with the architectural register and corresponding tag being used. Note that the first instruction (which was fetched in cycle 0) was ADD r3 <- r1, r6, and this instruction has already committed its data to architectural state.



b) Then, use the data flow graph to fill in the table below with the six instructions being executed on the processor. When referring to registers, please use their architectural names (r0 through r6) with source registers written in ascending order (as shown in the filled-in entry below). Note that the operation for instruction 5 has already been filled in.

OP = operation

 $Dest. = destination \ architectural \ register$

Src.1 = source architectural register 1

Src.2 = source architectural register 2

	OP	Dest.	Src.1	Src.2
Instruction 1	ADD	r3	r1	r6
Instruction 2				
Instruction 3				
Instruction 4				
Instruction 5	MUL			
Instruction 6				

VI. Runahead Potpourri I (15 points)

1. Runahead and Hardware Bugs

A runahead processor was designed with an unintended hardware bug: every other instruction in runahead mode is dropped by the processor after fetch. All other behavior of the runahead mode is exactly as we described in class. When a program is executed, which of the following possible behaviors could happen compared to a runahead processor without the hardware bug? Circle YES if there is a possibility to observe the described behavior. Assume that the program has no bug in it and executes correctly on the processor without the hardware bug.

a) The black head pro	ouggy runahead processor finishes the program correctly and faster than the non-buggy runapocessor.
YES	NO
Explain	why ($< 16 \text{ words}$).
b) The b	buggy processor finishes the program correctly and slower than the non-buggy processor.
YES	NO
Explain	why $(< 16 \text{ words})$.
c) The b	buggy processor executes the program incorrectly.
YES	NO
Explain	why ($< 16 \text{ words}$).

2. Runahead vs. Multithreading

Others have suggested an alternative to runahead execution is to use multithreading of the following form: have multiple hardware contexts in hardware (just like in fine-grained multithreading) and when one incurs an L2 cache miss, start executing from the other context. This method is commonly called switch-on-L2-miss multithreading, or more generally switch-on-event (SoE) multithreading.

<u>a)</u>	What are two advantages of runahead execution over SoE multithreading? ($<$ 20 words each.)
b)	What are two advantages of SoE over runahead execution? (< 20 words each.)
b)	What are two advantages of SoE over runahead execution? (< 20 words each.)
b)	What are two advantages of SoE over runahead execution? (< 20 words each.)
b)	What are two advantages of SoE over runahead execution? ($<$ 20 words each.)
b)	What are two advantages of SoE over runahead execution? (< 20 words each.)
b)	What are two advantages of SoE over runahead execution? (< 20 words each.)
b)	What are two advantages of SoE over runahead execution? (< 20 words each.)

VII. BONUS: Runahead Potpourri II (18 points)

1. Runahead vs. a Larger Instruction Window

In our performance simulation experiments, we found that a processor with a 128-entry instruction window and runahead execution achieves significantly higher performance than a processor with a 1024-entry window but no runahead execution, for a particular program A.

a) Explain why this could be the case. Provide as many reasons as possible, but no more than the correct ones. Enumerate each reason as 1., 2., 3.,
correct ones. Enumerance each reason as 1., 2., 6.,
For program B, we observed the opposite to be true: the 1024-entry window machine without runahead achieves significantly higher performance than a 128-entry window runahead machine.
b) Explain why this could be the case. Provide as many reasons as possible, but no more than the correct ones. Enumerate each reason as 1., 2., 3.,
2. Runahead and Value Prediction
Recall that in conventional runahead execution, when a load instruction misses in the L2 cache, its destination register is marked as invalid (INV). INV bits are propagated through instruction execution in such a way that all registers that are dependent on L2 misses are marked INV.
a) What is the rationale for marking a register as INV? (< 20 words.)
A clever engineer thinks of an alternative mechanism. Her suggestion is to assume that the value in the destination register of an L2 miss load is zero, instead of marking the register INV: in other words, predict the data value of an L2 miss load to be zero.
b) What are two advantages of this proposal over conventional runahead execution? (< 20 words each.)

c) When and why can this proposal be beneficial for performance (or some other metric) compared to conventional runahead? Provide a code example if it helps clarity.
d) What are two disadvantages of the engineer's proposal over conventional runahead execution? (< 20 words each.)
e) When and why can this proposal degrade performance (or some other metric) compared to conventional runahead? Provide a code example if it helps clarity.
Tananeda 1 10 1 de de code champie il 10 nospe ciario,

SCRATCH PAD

SCRATCH PAD

SCRATCH PAD