University of California, Berkeley
College of Engineering
Computer Science Division — EECS

Fall 1998 J. D. Kubiatowicz

Prerequisite Quiz
September 2, 1998
CS252 Graduate Computer Architecture

You are allowed to use a calculator and one 8.5” x 11” double-sided page of notes. Show your work on all problems. Good luck!

Your name:	
SID number:	
Email address:	

1	
2	
3	

Pass	Take next time

Question 1

Below is a series of memory read references sent to a cache. The cache holds 32 bytes total with 2-byte blocks, is 2-way set associative, and uses a least-recently-used replacement policy. Assume the cache is initially empty.

Classify each memory reference as a hit or a miss. Identify each cache miss as a **compulsory** miss or a **conflict** miss. One example is shown below. Blank working space is available on the next page.

Byte address	Hit/Miss?	Miss Type
1	miss	compulsory
19		
10		
30		
42		
51		
11		
18		
27		
34		
43		
10		
50		

Question 1 (continued)

This page is intentionally left blank for scratch space.

Question 2

The code sequence printed below is written in DLX assembly. Assume the code is going to be run on a 5-stage DLX pipeline with a single branch delay slot.

```
0           add    r6, r0, r0     ; reset running sum to zero
4           lw     r1, 16(r19)    ; load base address from stack
8           lw     r2, 32(r19)    ; load number of iterations
12   loop:  shli   r4, r2, #3     ; scale offset by 8
16          addu   r5, r4, r1     ; calculate address of data
20          lw     r4, 0(r5)      ; load data
24          add    r6, r6, r4     ; update sum with data value
28          bnez   r2, loop       ; branch to next iteration
32          addi   r2, r2, -1     ; (delay slot) update counter
36   exit:  hcf                   ; halt and catch fire (exit)
40          add    r10, r0, r0    ; extra example instruction
44          addi   r11, r10, #12  ; extra example instruction
```

Your task is to identify all of the **data dependencies** in the above program, and to classify each one as one of the following: *dependence* (RAW dependence), *antidependence* (WAR dependence), or *output dependence* (WAW dependence). **Note that we are asking you to find all *dependencies* in the program, not just pipeline hazards!** For each dependence, also determine the register involved in the dependence, whether or not forwarding is necessary to resolve the dependence (for the DLX 5-stage pipeline), and whether or not a stall is required even if forwarding is used. Assume that a register can be written and read on the same cycle without forwarding, and this will not cause a stall.

In the table below, "Source Instr" refers to the instruction that is *writing* to the register. "Dest Instr" refers to the instruction that is *reading* from the register. For both the "Source Instr" and "Dest Instr," columns, fill in the number printed to the left of the appropriate instruction above. "Register" refers to the register involved in the hazard. Two examples are given below.

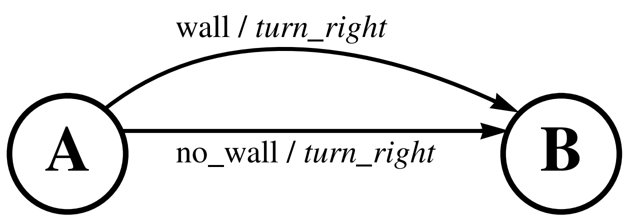
Source Instr	Dest Instr	Type of dependency	Register	Forwarding required?	Stall even if forward?
32	32	RAW (loop-carried)	`r2`	no	n/a
40	44	RAW	`r10`	yes	no

Question 3

"You are in a maze of twisty little passages, all alike" (BSD UNIX `adventure` *program)*

You have just been hired as lead robot designer at Robotz, Inc. On your first day at the job, you find your team of designers in a frenzied state. Apparently, the company is under contract to produce a robot that can autonomously navigate a maze, but, due to the stock market crash and recent budget cuts, there's only enough money to put two states into the state machine that makes up the robot's brain. Your poor team of designers hasn't been able to figure out how to complete the state machine transitions so that the robot successfully navigates the maze, and they want you to help.

Here's what they tell you. The robot's brain has one input, a sensor that detects if there is a wall directly in front of the robot. The brain can command the robot to do one of four things: *go_forward, turn_left, turn_right,* or *do_nothing*. The turns are all 90°. The *incomplete* state machine for the brain is given below. The robot starts out in state **A**.



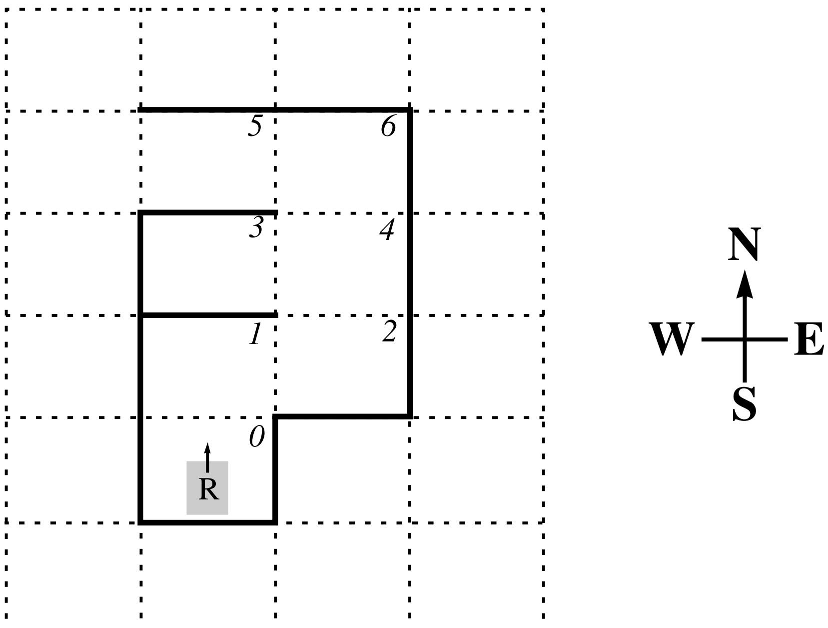
a) Your first job is to complete the state machine by adding any necessary transitions to allow the robot to navigate simple mazes. Make your modifications directly to the diagram above. You may not add states, nor may you alter existing transitions. You should assume that there are no loops in the maze.

b) You now want to test your robot. You are given a maze, printed on the next page. The maze is built on a grid (indicated by the dotted lines); the solid lines represent walls. At any point, the state of the robot is given by the square that it is on (indicated by the numbers in the upper-right of each square), and the compass direction that it is facing (**N**, **E**, **S**, **W**). The robot starts at 0N (grid 0, facing north). Fill in the table on the next page with the steps that the robot will take in solving the given maze. You should stop when the robot enters grid cell 5.

*(part **b** continues on the next page . . .)*

Question 3 (continued)

Here is the maze for part **(b)**:



And here is the table to fill in (the first step has been filled in as an example):

Position	State	Next Operation
0N	A	*turn_right*
0E	B	

Question 3 (continued)

This page is intentionally left blank for scratch space.