CpE 313 Fall 2004
Microprocessor Systems Design
HW 05
Due in class on Tuesday, October 12.

Problem 1 (*submitted by Matt Hendren's group*)

Examine the code sequence:

```
1       lw r2,2(r1)
2       lw r3,4(r1)
3       add r4,r2,r3
4       lw r5,2(r1)
5       sub r7,r5,r8
```

Assume a 5-stage MIPS pipeline enabled with forwarding. By reordering the code sequence above,

(a) all hazards can be eliminated.
(b) some of the hazards can be eliminated.
(c) none of the hazards can be eliminated, even though code can be re-ordered.
(d) none of the hazards can be eliminated, and code cannot be re-ordered.

Problem 2 (*submitted by Ibrahim*)

For the code sequence given in the question above, which one of the following statements is correct? Assume a 5-stage MIPS pipeline enabled with forwarding.

(a) The execution of instructions 3 and 4 is stalled by 2 cycles each.
(b) The execution of instructions 3 and 4 is stalled by 1 cycle each.
(c) The execution of instructions 3 and 5 is stalled by 2 cycles each.
(d) The execution of instructions 3 and 5 is stalled by 1 cycles each.

Problem 3 (*submitted by Matt Hendren*)

Examine the code sequence below.

```
add r1,r2,r3
sw 20(r1),r5
lw r15,20(r6)
add r7,r15,r8
```

What kind of data hazard is present in the above code assuming a 5-stage MIPS pipeline enabled with forwarding?

(a) RAW (Read After Write)
(b) WAR (Write After Read)
(c) WAW (Write After Write)
(d) RAR (Read After Read)

Problem 4 (*submitted by Patrick Keeven*)

Assume a 5-stage pipeline enabled with forwarding, and a 20-cycle stall for memory access. Consider the code:

```
lw r1,0(r2)
add r2,r3,r4
sub r5,r5,r4
```

How many cycles does this code take?

(a) 27 without reordering and 26 with reordering
(b) 27 without reordering and 27 with reordering
(c) 26 without reordering and 25 with reordering
(d) 27 without reordering and 25 with reordering

Problem 5 (*submitted by Ryan Underwood*)

How is an ALU-to-ALU RAW hazard resolved in hardware?

(a) By putting another adder in the ID stage.
(b) By having the hardware implement a pipeline bubble.
(c) By forwarding the value from the output of ALU to the input of ALU.
(d) By forwarding the value from the output of MEM to the input of ALU.

Problem 6 (*submitted by Tim Lorenz*)

Given that the memory location at address 50 + r3 contains the byte 0xFF, what will be the contents of r1 be after LB r1, 50(r3)?

(a) 0x0000 00FF
(b) 0x00FF
(c) 0xFFFF FFFF
(d) 0xFFFF

Problem 7 (*submitted by Evan Mansker*)

Which is the reason for using fixed width instructions?

(a) Memory is used more efficiently due to smaller code size.
(b) More versatility than with variable sized instructions.
(c) Makes the process of decoding instructions easier for the IF unit.
(d) Makes it necessary to decode instruction partially before knowing the number of total operands needed.

Problem 8 (*submitted by David Middendorf*)

For a variable width instruction, which of the following is the action of IF?

(a) Decode and have the operands ready to fetch.
(b) Decode the partially fetched instruction and determine how many more accesses to instruction cache are necessary.
(c) Start fetching operands.
(d) Decode the instruction in the IR while simultaneously reading the source register values.

Problem 9 (*submitted by Jeff Muir*)

What is the primary benefit of having fixed locations for source registers and the immediate values?

(a) It simplifies things for the assembly programmer
(b) It allows the ID unit to decode the instruction in parallel with reading the source registers and the immediate value, which increases efficiency.
(c) It decreases the code size.
(d) It allows for use of variable width instructions without decreasing efficiency.

Problem 10 (*submitted by Chip Bilbrey*)

Which of the following is a reason for enforcing object alignment?

(a) Improves memory access performance.
(b) Makes it easier to catch RAW hazards.
(c) Compilers can perform their task of data allocation more quickly.
(d) Makes it easier for the computer to check for data errors.

Problem 11 (*submitted by Bill Stallard*)

Assume that, in a given computer, object alignment is enforced and each memory access fetches four words at a time. At what memory location could be a new word NOT be stored?

(a) 4
(b) 16
(c) 30
(d) 20

Problem 12 (*submitted by Alex Vernon*)

What is the best solution to the problem of C programs using constants between 16 and 32 bits of binary representation? Assume that we need to compile them on the MIPS ISA.

(a) Compiler inserts enough NOP instructions after register load to ensure that the RAW hazard will be eliminated.
(b) Such constants are loaded using LUI.
(c) Code is re-ordered to avoid such constants.
(d) The lower 16 bits of such constants are loaded first, and then the upper 16 bits are forwarded through pipeline registers.

Problem 13 (*submitted by Alex Vernon*)

Is object alignment typically used? Why or why not?

(a) No, aligning objects causes memory holes resulting in decreased object retrieval time.
(b) Yes, objects can be stored at arbitrary addresses, allowing more code flexibility.
(c) Yes, memory speed is a bottleneck and object alignment reduces memory accesses needed to retrieve an object.
(d) No, non-aligned objects require less code space to be addressed.

Problem 14 (*submitted by Alexis's group*)

The two sequences of assembly code given below cause a RAW hazard. Which of them can be avoided by forwarding?

1.
```
lw R1, 2(R2)
addi R3, R1, #1
```

2.
```
addi R1, R2, #1
addi R3, R1, #2
```

(a) Only 1
(b) Only 2
(c) Both 1 and 2
(d) Neither 1 nor 2

Problem 15 (*submitted by Alexis's group*)

How many NOP instructions should a compiler insert in the following code to avoid all RAW hazards, if any? Assume a MIPS pipeline without forwarding.

```
lw R1, 100(R13)
add R3, R1, R3
sub R2, R1, R5
```

(a) 0
(b) 1
(c) 2
(d) 3

Problem 16 (*submitted by Alexis's group*)

Which of the items is true for a non-forwarded MIPS architecture running the following section of code?

```
lw R2, 4(R4)
sub R1, R4, R5
add R4, R2, R5
and R3, R6, R5
```

(a) There is no RAW hazard.
(b) The compiler can re-order the instructions to avoid the RAW hazard without adding any NOPs.
(c) The compiler must insert NOPs to avoid the RAW hazard.
(d) The compiler cannot avoid the RAW hazard.

Problem 17 (*submitted by Alexis's group*)

When will this code cause a RAW hazard?

sw 100(R1), R5
lw R3, 54(R2)

(a) Always
(b) Never
(c) Only if R5 equals R3
(d) Only if (R1+100) equals (R2+54)

Problem 18 (*submitted by Keith's group*)

Examine the code sequence.

LD R1, 0(R2)
DADDI R1,R1,#1

How does the hardware pipeline hazard detection handle the RAW hazard in the above code? Assume a forwarding-enabled MIPS pipeline.

(a) By inserting exactly one NOP before the DADDI instruction
(b) By stalling after IF stage of DADDI
(c) By stalling before IF stage of DADDI
(d) No stalling is needed, because the forwarding eliminates this RAW hazard

Problem 19 (*submitted by Keith's group*)

What is the difference between a stall and a NOP?

(a) Stalls are used by hardware, NOPs are used by software
(b) NOPs are used by hardware, stalls are used by software
(c) No difference. NOPs and stalls are used interchangeably
(d) Stalls are used before an instruction only, whereas NOPs can be used anywhere in an instruction's execution.

Problem 20 (*submitted by Keith's group*)

How can a two-cycle stall resulting from a branch hazard best be reduced to zero? Assume that the branches are resolved at the end of EXE stage.

(a) Include additional hardware to resolve the branch after the ID stage.
(b) Insert two NOPs after the branch instruction.
(c) Reorder the code such that two "safe" instructions execute after the branch instruction.
(d) You cannot; branch hazards will always result in at least a single-cycle stall.

Problem 21 (*submitted by Keith's group*)

Examine the code sequence below.

DSUB R4, R3, R2
BNEZ R4, <target-label>

In this code segment, assuming forwarding-enabled MIPS pipline, does the branch require a stall before its execution can be completed? Assume that the branches are resolved at the end of ID stage.

(a) No, because forwarding provides the value needed to BNEZ.
(b) Yes, because otherwise the result of DSUB isn't produced until the WB stage.
(c) Yes, because otherwise the result of DSUB is forwarded to the ALU instead of the conditional logic unit where it is needed.
(d) No, because there is no pipeline hazard present.

Problem 22 (*submitted by David Bubenik*)

What type of architecture automatically executes the next sequential instruction after a branch statement?

(a) delayed load architecture
(b) delayed branch architecture
(c) flushed branch architecture
(d) load-store architecture

Problem 23 (*submitted by Zheng Chen*)

How many RAW hazards are there in the code below if (r27+4) equals (r8+10)?

```
lw r1, 3(r8)
add r2, r3, r1
sub r5, r15, r2
lw r4, 4(r5)
add r5, r4, r1
sw 4(r27), r5
lw r1, 10(r8)
add r2, r1, r5
```

(a) 6
(b) 7
(c) 8
(d) 9

Problem 24 (*submitted by Zheng Chen*)

Assume that cache is not perfect, and a 3-cycle stall is needed for all load-to-ALU RAW hazards. Determine the total number of clock cycles needed to execute the following program. Assume a MIPS pipeline without forwarding.

```
lw r1, 3(r8)
add r2, r3, r1
sub r5, r1, r2
lw r4, 4(r3)
```

(a) 10
(b) 12
(c) 13
(d) 14

Problem 25 (*submitted by Hai Bui*)

For the code sequence given below, calculate the speed-up achieved by using the pipeline. Assume a MIPS pipeline without forwarding.

lw r1, 3(r8)
add r2, r3, r1
sub r5, r1, r2
lw r4, 4(r3)

(a) 5
(b) 12
(c) 3.45
(d) 1.67

Problem 26 (*submitted by Josh Lorenz*)

When an extra adder is used for branch resolution, when does the branch calculation finish?

(a) after stage ID
(b) after stage EXE
(c) after stage MEM
(d) after stage WB

Problem 27 (*submitted by Todd Acinelli*)

If we have the following code sequence for a MIPS machine, what will happen when we move instruction 3 before instruction 2?

1. lw r3, 20(r4)
2. add r4, r3, r5
3. lw r5, 20(r6)

(a) Register r5 will be read by the add instruction before a new value is written into it by the load instruction.
(b) Register r5 will be read by the add instruction after a new value is written into it by the load instruction.
(c) This reordering will remove all stalls, without causing incorrect execution.
(d) Both a and c are true.