CS 61C

Spring 1997
Midterm #2

Professor Brian Harvey

This exam was worth 24 points, and was open book.

Problem #1 (4 points)

Translate the following procedure to MAL. (Assume that the argument and return value for baz() are integers.)

```
int foo(int x, int y) {
   return x+baz(y+1);
}
```

The arguments and return value are in the standard registers. Don't use unnecessary stack space.

Problem #2 (4 points)

Mark each of the following true or false. (assume that we are talking about a MIPS system running C or C++.)

___ It will cause a runtime error if a procedure returns a pointer to an array declared as `static` within the procedure

___ If the compiler chooses to put local variables on the stack, it will need a bigger stack frame that if it chooses to put those local variables in registers `$16-$23.`

___ The expressions `*p` and `p[0]` have the same meaning if `p` is a pointer to an array of integers

___ The expressions `*a` and `a[0]` have the same meaning if `a` is an array of integers.

Problem #3 (2 points)

True of false: A branch instruction poses only control hazards to the pipeline, not data hazards.

Explain your answer.

Problem #4 (2 points)

Suppose we write computer programs to do each of the following computations in floating point arithmetic. Indicate which of the three is *most likely* to compute the mathematically correct answer, and which is *least likely* to compute the mathematically correct answer.

most likely ____ least likely ____ (((1 + 1/2) + 1/4) + 1/8) ...

most likely ____ least likely ____ 1/r^2 where r is the distance from an electron to the nucleus of its atom, in meters

most likely ____ least likely ____ 1/0

Problem #5 (2 points)

The following MAL code takes a nonzero value in `$8` and computes some function of that value in `$9`. Describe that function in one English sentence.

```
sub  $9, $0, $8
and  $9, $9, $8
```

Problem #6 (2 points)

Here is some C/C++ code:

```
int x[10];

void foo(void) {
    int y[10];

    y[3] = x[3];

}
```

(The procedure doesn't do anything very interesting.)

In the MAL translation of the body of the procedure there will be a `LW` instruction to get the value of `x[3]` followed by a `SW` instruction to store that value into `y[3]`. For each of these instructions, we want to know the *earliest* stage in the compilation process at which the final numeric values of the fields in those instructions are known.

For the `LW`: ____ by the C compiler ____ by the assembler ____ by the linker

For the `SW`: ____ by the C compiler ____ by the assembler ____ by the linker

Problem #7 (2 points)

In a multiprocessing system, it is best to use interrupts for both the keyboard (receiver) and the screen (transmitter). But given the following two possible alternatives, which would be *worse*? Explain why, in at most two sentences.

A. The system uses interrupts for keyboard input, but polling for screen output.

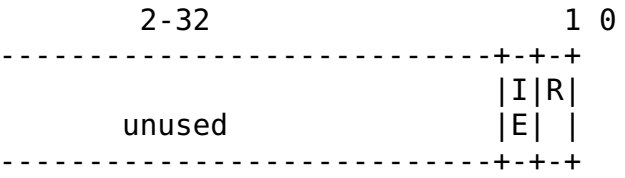
B. The system uses polling for keyboard input, but interrupts for screen output.

Problem #8 (4 points)

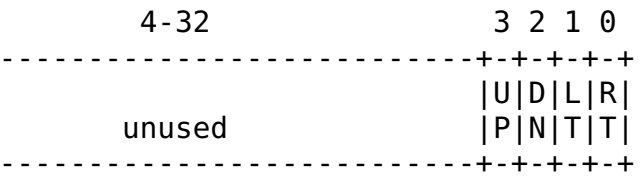
Suppose we have a mouse attached to our MIPS computer. It has a control register and a data register, like the other devices we've seen. You are going to use interrupts to handle

the mouse.

The control register, at `0xffff0010`, looks just like the others:



The data register, at `0xffff0014`, has four input bits:



Whenever the mouse is moved a little bit, it generates a signal in at least one (and at most two) of four directions: up, down, left, right. It also turns on the Ready bit, which is turned off when you read the data register, just as in the case of the receiver. (If the mouse is moved a lot, each little bit of the motion generates a signal.)

We keep two global variables, `mousex` and `mousey`, that hold the mouse's horizontal and vertical position. When the system starts these variables both contain zero. You are to keep track of the mouse's position by adding or subtracting 1 to/from these numbers, according to which bits are set, on each interrupt.

To simplify the coding, assume that the mouse is the only device that will interrupt, so your interrupt handler can assume that the mouse has moved.

To simplify the problem further, ignore the Up and Down bits, and ignore the vertical position. Just keep the horizontal position correctly. (Pretend that your partner is writing the vertical part!)

You must write the `intrp` procedure:

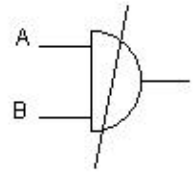
```
        .kdata
mousex: .word   0
mousey: .word   0

        .ktext

intrp:
```

Problem #9 (2 points)

Instead of having AND, OR, and NOT gates to work with, suppose you are given only NAND gates. A NAND gate has two input wires and one output wire; the output has the value `not (a and b)` where `a` and `b` are the inputs. We'll represent a NAND gate with this symbol:



Using only NAND gates, build a circuit that computes the AND function. (Two gates should be enough.)

Solutions

Problem #1

```
foo:   addi   $29, $29, -8
       sw     $16, 0($29)
       sw     $31, 4($29)

       add    $16, $0, $4
       addi   $4, $5, 1
       jal    baz
       addi   $2, $2, $16

       lw     $16, 0($29)
       lw     $31, 4($29)
       addi   $29, $29, 8
       jr     $31
```

Problem #2

1) F
2) T
3) T
4) T

Problem #3

False. No explanation was given.

Problem #4

1) least likely
2) least likely
3) most likely

Problem #5

The function produces the greatest integer of the form 2^N, where n is a nonnegative integer, that evenly divides the number in `$8`.

For example, the function will return 1 for any odd integer, 4 for 12, 2 for 14, etc.

Problem #6

For the LW: by the linker
For the SW: by the C compiler

Problem #7

B. Screen output is on average much faster than keyboard input, therefore if the CPU is required to wait for one or the other, it might as well wait for the faster device.

Problem #8

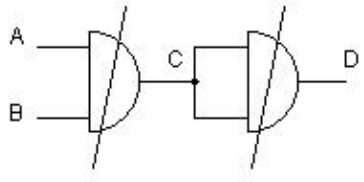
```
          .kdata
mousex:   .word    0
mousey:   .word    0

          .ktext

intrp:
          lui   $26, 0xffff
          lw    $27, 0x14($26)
          andi  $27, $27, 1
          bnez  $27, plus
          lw    $27, 0x14($26)
          andi  $27, $27, 2

          bnez  $27, minus
          b     exit
plus:
          lw    $27, mousex
          addi  $27, $27, 1
          sw    $27, mousex
          b     exit
minus:
          lw    $27, mousex
          addi  $27, $27, -1
          sw    $27, mousex
          b     exit
exit:
          mfc0  $26, $14
          .set noat
          lw    $1, 0($29)
          .set at
          addi  $29, $29, 4
          rfe
          jr    $26
```

Problem #9



Here is the truth table:

a	b	c	d
0	0	1	0
1	0	1	0
0	1	1	0
1	1	0	1