COMP SCI / SFWR ENG 2GA3

Emil Sekerinski, McMaster University, 2015-16

Assignment 2, due Monday, November 30, 11 pm

For this assignment, use QtSpim, http://spimsimulator.sourceforge.net/.

Special Instructions:

- Discussions with fellow students and the TAs to get a better understanding are encouraged. As part of ongoing research in the effectiveness of teaching methods, following rules on collaboration apply.
- If your student number is **odd**, you can collaborate with **one** other student in class. You need to put the name of your collaborator as a comment in your submission, such that similarities between your solution and the solution of your collaborator will not be construed as academic dishonesty. You and your collaborator can, but do not need to submit identical solutions. Everyone has to submit their own solution, i.e. you cannot submit a solution for your collaborator. Collaborating with more than one student is considered to be academic dishonesty.
- If your student number is **even**, you must **not collaborate**. Collaborating with someone is considered to be academic dishonesty.
- All submissions are automatically checked for similarity.
- In the first assignment, the rules for even and odd student numbers were reversed. In case one assignment has a lower average than the other assignment, the assignment with the lower average will be curved such that averages are identical. This way, you are not at a disadvantage for not having collaborated at the "more difficult" assignment.

Question 1: Integer Square Root [5 points]

Programming languages have a function for computing the square root of an integer, either built-in or as part of a library, but most processors don't have an instruction for that. An efficient way to compute the square root of non-negative integer x is by binary search (bisection), starting with an upper bound a and lower bound b that contain the (approximate) square root, and halving the interval such that always $a^2 \le x < b^2$. When $b - a = 1$, then a is the result. The midpoint $(a + b) / 2$ can be efficiently calculated by shifting the sum $a + b$ to the right. If x is a 32 bit unsigned integer, then the square root must be less than 0x10000, as that number squared would already not fit in 32 bits, so 0x10000 is a safe initial value for b. Here is the algorithm expressed in C:

```
int sqrt(unsigned x) {
    unsigned a, b, m; // Limits and midpoint
    a = 0; b = 0x10000;
    do {
        m = (a + b) >> 1;
        if (m * m <= x) a = m; else b = m;
    } while (a + 1 != b);
    return a;
}
```

Write a MIPS procedure that takes the argument in register `$a0` and returns the square root in register `$v0`, using registers `$t0`, … as needed. The procedure must be of the following form, otherwise the test will fail: first, a procedure called `sqrt` is defined, with `jr $ra` as the last instruction and without any empty lines, then there must be an empty line, then there can be some arbitrary code that is ignored for marking.

Place your solution in a file named `a2q1.s` and submit it on Avenue. Naming your file differently or with different capitalization or suffix will cause the test of your code to fail.

Question 2: Integer Exponentiation [5 points]

If x, n are unsigned integers, x^n can be computed by n repeated multiplications of x, which can be inefficient. A more efficient method uses the binary representation of the exponent. For example, as $13 = 1101_2 = 8 + 4 + 1$, we have

$$x^{13} = x^{8+4+1} = x^8 \times x^4 \times x^1$$

In general, x^n can be computed as the product of those factors x raised to 2^i, where the i-th bit is 1 in the binary representation of x. Computing those factors requires only squaring: to compute x^8, we multiply x^4 by itself, to compute x^4, we multiply x^2 by itself, and to compute x^2, we multiply x by itself. The algorithm, expressed in C below, sets p initially to x and, at each iteration, multiplies p by itself. The first bit (bit 0) of n is extracted by AND-ing (& in C) n with 1, and then, at each iteration, shifting n to the right (>> in C).

```
unsigned exp(unsigned x, unsigned n) {
    unsigned r, p;
    r = 1; p = x;
    while (1) {
        if (n & 1 == 1) r = r * p;
        n = n >> 1;
        if (n == 0) return r;
        p = p * p;
    }
}
```

Write a MIPS procedure that takes arguments `x` and `n` in register `$a0` and `$a1`, and returns the result in register `$v0`. The procedure must be of the following form, otherwise the test will fail: first, a procedure called `exp` is defined, with `jr $ra` as the last instruction and without any empty lines, then there must be an empty line, then there can be some arbitrary code that is ignored for marking.

Place your solution in a file named `a2q2.s` and submit it on Avenue. Naming your file differently or with different capitalization or suffix will cause the test of your code to fail.

Question 3: X-Intersect [5 points]

Write a MIPS procedure that takes a function and an interval as argument, such that the function must have an x-intersect in that interval, and computes the approximate x-intersect in the interval by binary search. The algorithm, expressed in C, is below.

```
int xintersect(int (*f) (int), int a, int b) {
    /* f(a) ≤ 0 ≤ f(b), a ≤ b */
    while (b-a > 1) {
        int m = (a+b)/2;
```

```
            if (f(m) <= 0) a = m; else b = m;
        }
        return a;
    }
```

The parameter `int (*f) (int)` is a pointer to a C function `f` that takes an `int` as a parameter and returns an `int`. In MIPS assembly, this corresponds to the address of a MIPS procedure. The interval is given by lower bound `a` and upper bound `b`. For example, with

```
int f1(int x) {
    return x*x-16;
}
```

calling `xintersect(f1, 0, 100)` returns 4. The function address is passed in `$a0`, and the lower and upper bound in `$a1` and `$a2`. Avoid using division. The procedure must be of the following form, otherwise the test will fail: first, a procedure called `xintersect` is defined, with `jr $ra` as the last instruction and without any empty lines, then there must be an empty line, then there can be some arbitrary code that is ignored for marking.

Place your solution in a file named `a2q3.s` and submit it on Avenue. Naming your file differently or with different capitalization or suffix will cause the test of your code to fail.

Question 4 (Bonus): Floating-Point Exponentiation [5 points]

Calculate x^n, where x is now a floating-point number, according to the same algorithm as above:

```
float exp(float x, unsigned n) {
    float r, p;
    r = 1.0; p = x;
    while (1) {
        if (n & 1 == 1) r = r * p;
        n = n >> 1;
        if (n == 0) return r;
        p = p * p;
    }
}
```

Write a MIPS procedure that takes a single-precision floating-point parameter in register `$f12`, an unsigned integer parameter in register `$a0`, and returns the result in register `$f0`. The procedure must be of the following form, otherwise the test will fail: first, a procedure called `exp` is defined, with `jr $ra` as the last instruction and without any empty lines, then there must be an empty line, then there can be some arbitrary code that is ignored for marking.

Place your solution in a file named `a2q4.s` and submit it on Avenue. Naming your file differently or with different capitalization or suffix will cause the test of your code to fail.