COMP SCI / SFWR ENG 2GA3

Emil Sekerinski, McMaster University, 2015-16

Assignment 1, due Friday, November 13, 11 pm

For this assignment, use QtSpim, http://spimsimulator.sourceforge.net/. An introduction will be given in the tutorials and in class.

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 **even**, 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 **odd**, you must **not collaborate**. Collaborating with someone is considered to be academic dishonesty.
- All submissions are automatically checked for similarity.
- In the second assignment, the rules for even and odd student numbers are 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: Double-Add [5 points]

Suppose registers \$a1 and \$a0 contain a 64-bit unsigned integer $A = \$a1 \times 2^{32} + \$a0$ and registers \$a3 and \$a2 contain a 64-bit unsigned integer $B = \$a3 \times 2^{32} + \$a2$. Write a program to compute the sum of A and B and store it in \$v1, \$v0 such that $A + B = \$v1 \times 2^{32} + \$v0$. An overflow in case the result does not fit in 64 bits is to be ignored. Your program will be automatically tested. It must be of the following form, otherwise the test will fail. First, a procedure called `doubleadd` is be defined, with `jr $ra` as the last instruction, then there must be an empty line, then there can be some arbitrary code that is ignored for marking:

```
doubleadd:
     ... here comes your code, without empty lines,
     ... that takes $a0, $a1, $a2, $a3 and computes $v0, $v1
     jr   $ra  # this is the last instruction

# here is some sample code that can help with testing
# this part is ignored for marking
main:li   $a0, 0xFFFFFFFF
     li   $a1, 0x22
     li   $a2, 0x1
     li   $a3, 0x44
     b    doubleadd
```

Adding the lower words may lead to a carry that needs to be taken into account. Use only `addu`, not `add`, for adding and use `sltu` to determine if the addition of two unsigned integers produced a carry, without using conditional branch instructions. Your solution can use registers \$t0, … and \$s0, … .

Place your solution in a file named `a1q1.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: Multi-Add [5 points]

Languages like Python allow integers of arbitrary length. Such integers can be stored as an array of words. Let the first element of such an array be the least significant word and the last element the most significant word. Suppose now that \$a0 and \$a1 point to two such arrays that are to be added, \$a2 points to the resulting array, all arrays are of equal length, \$a3 contains the length of the arrays, and the integers represented by the arrays are unsigned. Write a procedure `multiadd` of the form like above to compute the sum. The procedure must only change the array pointed to by \$a2; it may use registers \$t0, … and \$s0, … :

```
multiadd:
     ... here comes your code, without empty lines
     jr   $ra  # this is the last instruction

# anything after an empty line is ignored
```

Place your solution in a file named `a1q2.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: Parity [5 points]

The *even parity* for a sequence of bits is 0 if there is an even number of 1's in the sequence, or 1 if there is an odd number of 1's in the stream. For example, the even parity bit for 101 is 0, while the even parity bit for 100 is 1. The *odd parity* bit is the opposite. A way to remember it is that in even parity, there is always an even number of 1's (including the parity bit). In odd parity, there is always an odd number. A parity bit adds redundancy that allows detecting a one-bit error. Parity bits are used to detect transmission errors in serial data transmissions, when for example to 7 bits representing an ASCII character an 8th parity bit is added, and on a PCI bus. Parity bits are also used to check data in main memory, data in caches, and data spread over disks of a RAID (Redundant Array of Inexpensive Disks) drive.

In this question, you will use even parity. The task is to write a procedure `encodeparity` that takes a 16-bit word in register \$a0 and returns a 17-bit word in register \$v0, with the parity added as the highest bit. To start with, complete following table; the solution is discussed in the tutorials before the due date:

Bit 16 parity	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	1	0	0	0	1	1	0	1	0	1	0	0	1	1	0	1
	0	1	1	0	0	0	0	1	0	0	0	1	1	0	1	1
1	1	0	1	1	1	0	0	1		0	1	0	0	1	1	1
0	0	1	1	0	0	0	0	1	0	1		1	0	1	1	0

The parity p of r can be calculated efficiently by following formula, where $x >>> y$ stands for (logical) shifting of x to the right by y positions:

$$p = (r >>> 0) + (r >>> 1) + \ldots + (r >>> 15)$$

Then the lowest bit of p is the parity. For example with $r = 101$ we have $p = 101 + 10 + 1 = 1000$, so the parity (lowest bit) is 0. With $r = 100$ we have $p = 100 + 10 + 1 = 111$, so the parity is 1. Write a procedure `encodeparity` of the form like above. The procedure may make use of registers \$t0, … and \$s0, … :

```
encodeparity:
     ... here comes your code, without empty lines
     ... returning $a0 with parity in $v0
     jr   $ra  # this is the last instruction

# anything after an empty line is ignored
```

Place your solution in a file named `a1q3.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): String Length [5 points]

In C, strings are arrays of characters (bytes) terminated by the null character (byte with value 0). Assuming that $a0 points to a character string, write a procedure `stringlength` that returns the length of the string in register $v0. Write a procedure `stringlength` of the form like above. The procedure may make use of registers $t0, … and $s0, … :

```
stringlength:
     ... here comes your code, without empty lines
     ... returning the length of string at $a0 in $v0
     jr   $ra  # this is the last instruction

# anything after an empty line is ignored
```

Place your solution in a file named `a1q4.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.