Computer Science 230
Computer Architecture and Assembly Language
Spring 2022

Assignment 4

Due: Monday, April 11th, 11:55 pm by Brightspace submission
(Late submissions **not** accepted)

Programming environment

For this assignment you must ensure your work executes correctly on the MIPS Assembler and Runtime Simulator (MARS) as has been used for all previous assignments this semester. Assignment submissions prepared with the use of other MIPS assemblers or simulators will not be accepted.

Individual work

This assignment is to be completed by each individual student (i.e., no group work). Naturally you will want to discuss aspects of the problem with fellow students, and such discussion is encouraged. **However, sharing of code fragments is strictly forbidden without the express written permission of the course instructor (Zastre).** If you are still unsure regarding what is permitted or have other questions about what constitutes appropriate collaboration, please contact me as soon as possible. (Code-similarity analysis tools will be used to examine submitted work.) The URLs of significant code fragments you have found and used in your solution must be cited in comments just before where such code has been used.

Objectives of this assignment

- Obtain more practice with problem solving when using assembly-language programming.
- Use the "Instruction Statistics", "BHT Simulator", and "Data Cache Simulator" to capture some statistics about the behavior of your code, and to reason about your codes behavior.

Supplied files

You are provided with three MARS assembly files:

- `a4-parts-ABCD.asm` is to be used when completing the first four parts of this assignment.
- `a4-part-E.asm` is to be used when completing the second-to-last part of this assignment.
- `a4-support.asm` are procedures provided by the instructor that should help with you coding and debugging.

Also provided is a ZIP file contain several binary files containing sequences of integers. **Please note that these are stored as binary; the contents of these files <u>cannot</u> be viewed with a text editor.**

The program `a4-parts-ABCD.asm` demonstrates the way procedures in `a4-support.asm` can be used. The code as given for the procedures `read_file_of_ints` and `dump_ints_to_console` will read into MARS and print out (respectively) the contents of `integers-10-314.bin` (i.e. one of the binary files in the ZIP archive). You may, of course, modify this code as you proceed with completion of the assignment.

Part (a): Write procedure `accumulate_sum`

`accumulate_sum:`

parameters:
`$a0` holds the address of an array of integers for input
`$a1` holds the address of an array in which the result is to be held
`$a2` holds number of integers in each array (i.e., same number in both)

return value: *none*

This procedure sums the values in an array, but in an interesting way. Each element indexed by `i` in the result array is the sum of all elements `1 .. i` in the input array (i.e., assuming we index starting at `1`). Therefore if the input array contains:

```
65 893 435 -25 -100 198 863 664 554 -56
```

then the result array must contain:

```
65 958 1393 1368 1268 1466 2329 2993 3547 3491
```

Part (b): Write procedure `accumulate_max`

```
accumulate_max:
```

parameters:
- `$a0` holds the address of an array of integers for input
- `$a1` holds the address of an array in which the result is to be held
- `$a2` holds number of integers in each array (i.e., same number in both)

return value: *none*

This procedure determines the maximum value in the array, but in an interesting way. Each element indexed by `i` in the result array is the maximum of all elements `1 .. i` in the input array (i.e., assuming we index starting at `1`). Therefore if the input array contains:

```
65 893 435 -25 -100 198 863 664 554 -56
```

then the result array must contain:

```
65 893 893 893 893 893 893 893 893 893
```

Part (c): Write procedure `reverse_array`

```
reverse_array:
```

parameters:
- `$a0` holds the address of an array of integers for input
- `$a1` holds the address of an array in which the result is to be held
- `$a2` holds number of integers in each array (i.e., same number in both)

return value: *none*

This procedure creates a copy of the input array, but with the values reversed. If the input array contains:

```
65 893 435 -25 -100 198 863 664 554 -56
```

then the result array must contain:

```
-56 554 664 863 198 -100 -25 435 893 65
```

Part (d): Write procedure `pairwise_max`

`pairwise_max:`

parameters:
- `$a0` holds the address of the first array of integers for input
- `$a1` holds the address of the second array of integers for input
- `$a2` holds the address of an array in which the result is to be held
- `$a3` holds number of integers in each array (i.e., same number in both)

return value: *none*

This procedure compares the value at index `i` within the first input array with the value at index `i` of the second array. If the value in the first array is larger than the value from the second array, the result array will have the first array's value at index `i`; otherwise it will be the second array's value at index `i`. If the first array is:

```
65 893 435 -25 -100 198 863 664 554 -56
```

and the second array is:

```
237 756 756 476 881 342 872 949 275 934
```

then the result array must contain:

```
237 893 756 476 881 342 872 949 554 934
```

Part (e): Complete `a4-part-E.asm`

Assuming you have completed the four procedures listed so far, and assuming you copy your solutions for these into `a4-part-E.asm`, you are to implement the following sequence of operations (which have been written in a kind of pseudo code):

```
A3 = accumulate_max(A1)

A4 = accumulate_max(A2)

A5 = reverse(A3)

A6 = pairwise_max(A4, A5)

A7 = accumulate_sum(A6)

output to console: A7[-1]
```

Notice that the pseudocode uses procedure-call notations, result-assignment notations, and the -1 index which is familiar to Python coders (i.e., last index in the array). So if `A1` is:

```
65 893 435 -25 -100 198 863 664 554 -56
```

and `A2` is:

```
237 756 756 476 881 342 872 949 275 934
```

then the result (i.e., the final output to the console) will be:

```
9098
```

Part (f): Obtain some MARS/MIPS32 statistics from Part (e)

This semester we have considered such concepts as cache memory, characterization of programs by instruction counts for different instruction types, and (very briefly!) branch prediction.

In this last part of the assignment you will gather and report statistics of your implementation of the previous part by using:

- The Branch-History Simulator ("BHT") tool;
- The Instruction Statistics tool; and
- The Data Cache Simulator tool

For this you will also use the provided binary files containing integers, and do so in pairs. An example of the expected output appears as the last page of this assignment description, and you are to submit this as a single-page PDF.

Some of these statistics change from one data set to another. Others seem to stay more or less the same, regardless of the size of the data set. Did you see this behaviour? What is your explanation (in your own words) for this? Provide your explanation in a single paragraph (also on a single-paged PDF) and include your reasoning that justifies your explanation.

What you must submit

- Your completed work in the two assembly files:
  - `a4-part-ABCD.asm`
  - `a4-part-E.asm`.
- Two PDFs that make up your work for Part (f):
  - `statistics.pdf`
  - `explanation.pdf`

Evaluation

- 2 marks: Solution for part A.
- 2 marks: Solution for part B.
- 2 marks: Solution for part C.
- 2 marks: Solution for part D.
- 6 marks: Solution for part E.
- 6 marks: Work for part F.

Therefore the total mark for this assignment is 20.

Some of the evaluation above will also take into account whether or not submitted code is properly formatted (i.e., indenting and commenting are suitably used), and the file correctly named.

Note: The following is the result of the instructor's use of the various MARS tools their sample solution for the assignment. **However, different integer files appear here – you must work with the files I have provided.**

	test case A	**test case B**	**test case C**
File 1	integers-10-6	integers-40-91	integers-200-150
File 2	integers-10-21	integers-40-101	integers-200-12345
Instructions statistics			
ALU	296 (44%)	811 (41%)	3531 (39%)
Jump	72% (11%)	223 (11%)	1031 (11%)
Branch	89 (13%)	329 (16%)	1609 (18%)
Memory	175 (26%)	505 (25%)	2265 (25%)
Other	40 (6%)	130 (7%)	610 (7%)
***BHT prediction**:*			
Best	100% (index 2)	100% (index 14)	100% (index 14)
Median	86.36% (index 10)	97.56 (index 3)	99.50% (index 6)
Worst	50% (index 12)	50% (index 12)	50% (index 12)
Cache (8 lines, 8 words per line)			
Direct Mapping	64%	52%	48%
2-way set associative	77%	73%	71%
4-way set associative	86%	83%	83%
Fully Associative	86%	83%	83%