Course Project

ECEN/CSCI 4593

Spring 2017

Revision 1.2

Objective: The course project builds upon the concepts from the textbook and taught in class to develop a deeper or further understanding of the concepts of computer architecture. This understanding can be applied to developing a CPU or any computer based platform. The project is to develop a 5 stage MIPS processor with an instruction and data cache with a simulated memory controller.

Appendices: The simulation will be a User mode program and no requirements to support Supervisor or Kernel mode.

Appendix A: Required MIPS instructions required to support this course project.

Appendix B: Program1 is an optimized program that takes a 250-byte array and sorts it using a Bubble Sort routine and then the program takes the same original 250-byte array and sorts it using an Insertion Sort routine. Both sorted arrays are then compared to determine if the sorted arrays are identical.

Appendix C: Program2 takes a quote from Julius Caesar, encrypt it using his encryption algorithm, decrypting it, and then comparing it to the original quote. The encryption algorithm works on a single character at a time which means this program will be accessing memory with load and store bytes.

Appendix D: Course project deliverables

Appendix E: Course project rubric

Due date: Friday, May 5th, 2017, at 11:59pm

Bonus scores are available based on the following:

- Determine minimum I and D caches for each program 1% towards final grade
  - That provides best performance
    - This cache size could be greater than the specified maximum 3K bytes of cache memory available
  - Must be supported by simulation results
  - Will be graded from 0-2% depending on the results
- Simulating a 2-way set associative data cache 2% towards final grade
- Simulating a unified instruction / data cache 2% towards final grade
- Submitting the course project by Tuesday May 2nd 2% towards final grade
  - Successful simulator that will run both code segments provided
- Submitting the course project by Friday, April 28th 4% towards final grade
  - Successful simulator that will run both code segments provided

Functionality required by the simulator:

- Simulation of a MIPS CPU
- Goal is to simulate a limited instruction set of integer based instruction set
- Found in the text book pages A-51 through A-70
  - No plan to require multiply or division instructions
- 5 stage pipeline
- IF, ID, EX, MEM, and WB (same as we have been learning in the text book)
- Forwarding from the MEM:EX and MEM:WB pipeline stages are to be implemented for ALU and Branch detection
- Branch detection should be in the ID, Decode, stage of the pipeline

Simulate memory hierarchy

- Single level, L1, cache with main memory
- Cache memory hit access is zero penalty
- Cache hit penalty is 8 clock cycles
- If multiple cache lines are read, subsequent cache lines come every 2 cycle

Simulate the following cache memory organization

- The cache memory will become the Instruction and Data memories of the pipeline
  - Separate Instruction and Data caches (also known as split caches)
- Simulate cache misses to main memory (Single line fill, 4-line, and 16-line block fill)
- Cache memory sizes to simulate provided in their respective appendices.
- Total of 2 or 3 cache combinations to simulate per program/data set (single line, 4-line, and 16-line block fill)

Simulate two different cache write policies

- All cache schemes will support an early start function on instruction cache fills
- All cache schemes will support a single deep write buffer on write through and write back caches to improve performance
  - On write back, the write buffer holds the data to be written to occur until the new cache block has been filled. After the new cache block, has been loaded into the data cache, the write buffer then will write the data in the write buffer to main memory
- Block size of 1 (read miss penalty is 8)
- Block size of 4 and 16 (read total cache access penalty, 1st fill = 8 cycles, subsequent fills = 2 cycle penalty)
- Write to memory will cause main memory to be busy, not able to support a memory access, of 6 clock cycles on the first word of a block and 2 clock cycles for subsequent blocks

Total cache simulations

- Approximately per program = 1 cache architectures (split I and D caches) * 3 cache fill policies (1, 4, and 16 block lines) * 2 write policies (write through and write back) * 3 cache memory sizes (defined in the program appendices) = 18

Results to be recorded per simulation

- Total instructions executed
- Total clock cycles executed

- Effective CPI
- Cache Hit Rate for I-cache
- Cache Hit Rate for D-cache

Grade

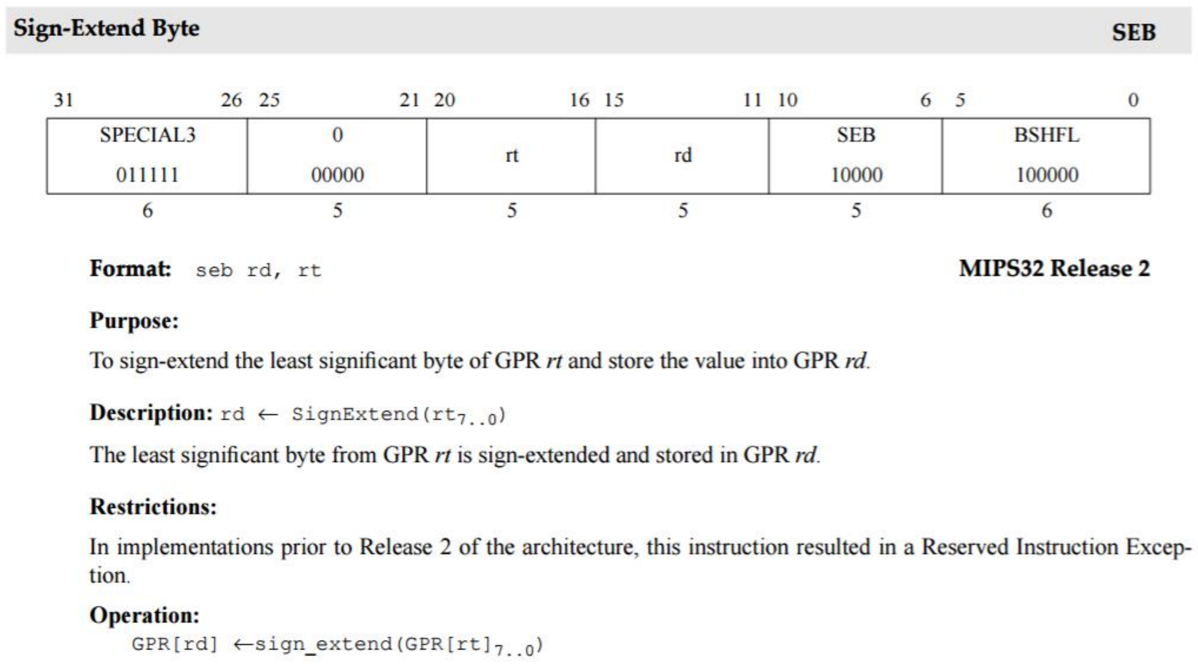
- Refer to Appendix D, Deliverables, and Appendix E, Rubric.

Appendix A: Required MIPS instruction set

nop
add
addu
and
jr
nor
or
movn
movz
slt
sltu
sll
slr
sub
subu
xor
addi
addui
andi
xori
beq
bne
bgtz
bltz
blez
j
jal
lb
lbu
lhu
lui
lw
ori
slti
sltiu
sb
sh
sw
seb *

Note: Details of these instructions can be found in the textbook, “Computer Organization and Design,” pages A-51 through A-80.

seb: sign-extend byte



Sign-Extend Byte **SEB**

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL3 011111	0 00000	rt	rd	SEB 10000	BSHFL 100000
6	5	5	5	5	6

Format: `seb rd, rt` **MIPS32 Release 2**

Purpose:

To sign-extend the least significant byte of GPR *rt* and store the value into GPR *rd*.

Description: `rd ← SignExtend(rt`$_{7..0}$`)`

The least significant byte from GPR *rt* is sign-extended and stored in GPR *rd*.

Restrictions:

In implementations prior to Release 2 of the architecture, this instruction resulted in a Reserved Instruction Exception.

Operation:

```
    GPR[rd] ←sign_extend(GPR[rt]7..0)
```

Opcode details from “MIPS Instruction Set Reference Manual”

Appendix B: Program 1

Program1 is an optimized program that takes a 250-byte array and sorts it using a Bubble Sort routine and then the program takes the same original 250-byte array and sorts it using an Insertion Sort routine. Both sorted arrays are then compared to determine if the sorted arrays are identical.

Results of the program can be found at memory locations 6, 7, 8, and 9.

Memory[6] = sorted value at a location in the middle of the array.
    Result = 112
Memory[7] = the number of passes through the bubble sort loops
    Result = 29355
Memory[8] = the number of passes through the insertion sort loops
    Result = 14305
Memory[9] = 0 if the two sorted arrays are identical, and 1 if they are not
    Result = 0

Key input values can be found in memory locations 0, 1, and 5.

$sp	= Memory[0];
$fp	= Memory[1];
starting $pc address	= Memory[5];

Note: The program is completed when the $pc = 0x0000 0000.
Note: Memory array should be 1200 words in size.

The following must be implemented:

- Instruction cache fill Early start function
- Write back buffer of 1 which means that on a write through, the pipeline can proceed as the write of 1 store is being stored into main memory and that for write back, the read of the data block occurs first to enable the pipeline to proceed while the store of the dirty block is occurring to main memory.

- Delayed branch per the MIPS architecture that means the instruction after a branch or jump is always executed.

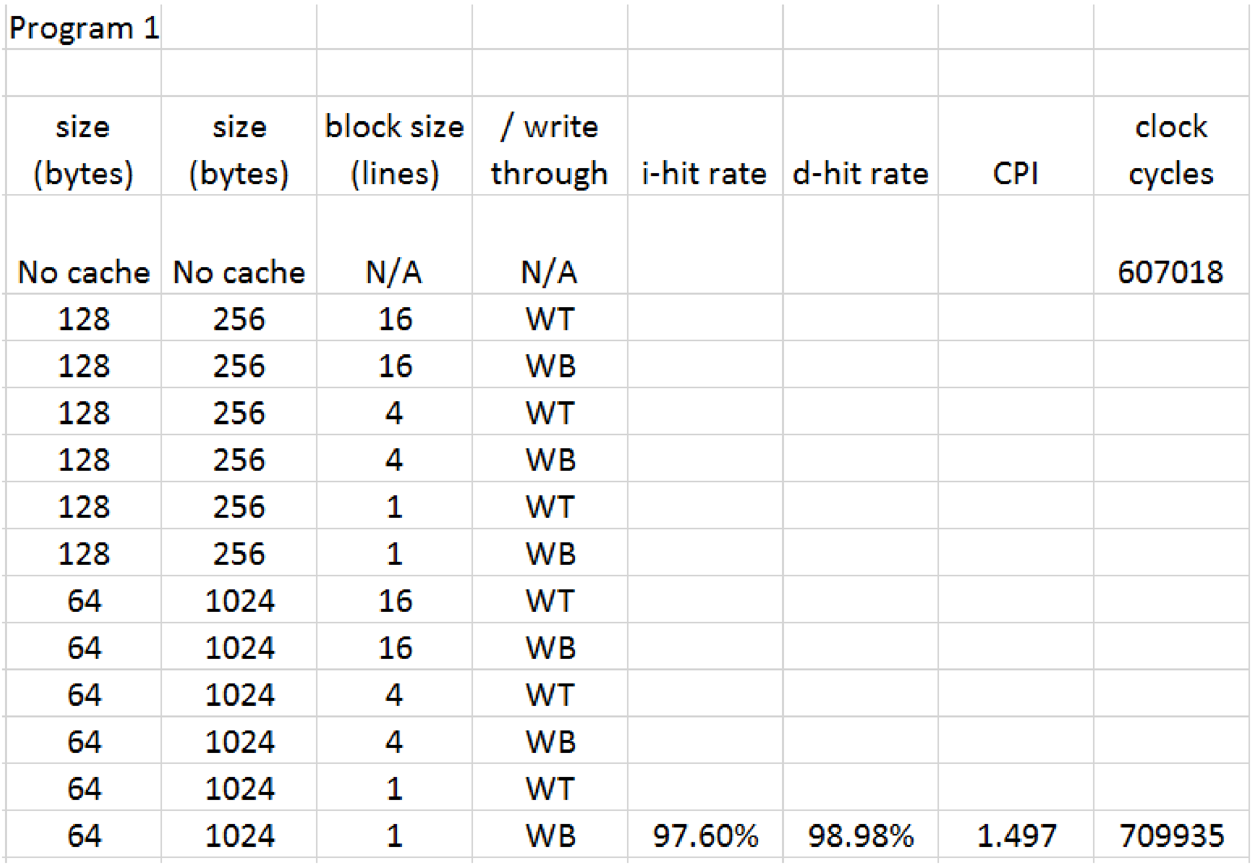
Below is the matrix of test results that are required to be completed for this program.

An additional test line of no cache has been added to better evaluate and grade the pipeline and cache implementations.

Program 1							
size (bytes)	size (bytes)	block size (lines)	/ write through	i-hit rate	d-hit rate	CPI	clock cycles
No cache	No cache	N/A	N/A				
128	256	16	WT				
128	256	16	WB				
128	256	4	WT				
128	256	4	WB				
128	256	1	WT				
128	256	1	WB				
64	1024	16	WT				
64	1024	16	WB				
64	1024	4	WT				
64	1024	4	WB				
64	1024	1	WT				
64	1024	1	WB				

To help you determine if your simulation run is matching my results, I have provided my last line of my test results for program 1.

I have calculated a total # of instructions to be 474,173.



Program 1							
size (bytes)	size (bytes)	block size (lines)	/ write through	i-hit rate	d-hit rate	CPI	clock cycles
No cache	No cache	N/A	N/A				607018
128	256	16	WT				
128	256	16	WB				
128	256	4	WT				
128	256	4	WB				
128	256	1	WT				
128	256	1	WB				
64	1024	16	WT				
64	1024	16	WB				
64	1024	4	WT				
64	1024	4	WB				
64	1024	1	WT				
64	1024	1	WB	97.60%	98.98%	1.497	709935

Appendix C: Program 2

Program2 takes a quote from Julius Caesar, encrypt it using his encryption algorithm, decrypting it, and then comparing it to the original quote. The encryption algorithm works on a single character at a time which means this program will be accessing memory with load and store bytes.

Results of the program can be found at memory locations 6, 7, 8, and 9.

Memory[6] = Compares the decrypted string with the original quote
Result = 1 (1 = match, 0 = they are different)
Memory[7] = 4 ASCI-II characters in the middle of the quote
Result = 0x20696e71
Memory[8] = the 4 bytes in memory[7] encrypted with key = 5
Result = 0x206e7376
Memory[9] = 4 ASCI-II characters in the middle of the decrypted string
Result = 0x20696e71

Key input values can be found in memory locations 0, 1, and 5.

$sp	= Memory[0];
$fp	= Memory[1];
pointer to quote	= Memory[2];
starting $pc address	= Memory[5];

Note: The program is completed when the $pc = 0x0000 0000.
Note: Memory array should be 1200 words in size.

The following must be implemented:

- Instruction cache fill Early start function
- Write back buffer of 1 which means that on a write through, the pipeline can proceed as the write of 1 store is being stored into main memory and that for write back, the read of the data block occurs first to enable the pipeline to proceed while the store of the dirty block is occurring to main memory.

- Delayed branch per the MIPS architecture that means the instruction after a branch or jump is always executed.

Below is the matrix of test results that are required to be completed for this program.

An additional test line of no cache has been added to better evaluate and grade the pipeline and cache implementations.

i-cache size (bytes)	d-cache size (bytes)	block size (lines)	writeback / write through	i-hit rate	d-hit rate	CPI	total clock cycles
No cache	No cache	N/A	N/A				
64	512	16	WT				
64	512	16	WB				
64	512	4	WT				
64	512	4	WB				
64	512	1	WT				
64	512	1	WB				
128	256	16	WT				
128	256	16	WB				
128	256	4	WT				
128	256	4	WB				
128	256	1	WT				
128	256	1	WB				
256	128	16	WT				
256	128	16	WB				
256	128	4	WT				
256	128	4	WB				
256	128	1	WT				
256	128	1	WB				

To help you determine if your simulation run is matching my results, I have provided the last line of my test results for program 2.

I have calculated a total # of instructions to be 12,179.

Program 2							
i-cache size (bytes)	d-cache size (bytes)	block size (lines)	writeback / write through	i-hit rate	d-hit rate	CPI	total clock cycles
No cache	No cache	N/A	N/A				14472
64	512	16	WT				
64	512	16	WB				
64	512	4	WT				
64	512	4	WB				
64	512	1	WT				
64	512	1	WB				
128	256	16	WT				
128	256	16	WB				
128	256	4	WT				
128	256	4	WB				
128	256	1	WT				
128	256	1	WB				
256	128	16	WT				
256	128	16	WB				
256	128	4	WT				
256	128	4	WB				
256	128	1	WT				
256	128	1	WB	98.65%	73.74%	1.561	19009

Appendix D: Course Project Deliverables

Due date: Friday, May 5th, 2017, at 11:59pm

Submitted: Via D2L dropbox and both team members should submit the course project!

Deliverables:

- All files required to run your simulation
    - The program should be set to run Program2
- Final Project Report

Final Project Report:

- Detail instructions on how to install and run your simulation
    - Specify what functionality is working in the simulator to set expectations when the simulator is ran
- Detail instructions on where and/or how to modify program2
- Self-assessment on functionality of simulator
    - Additional point will be taken off on functionality stated via self-assessment, but does not work
    - Categories of self-assessment
        - Pipeline works without any caches
        - Pipeline works with Instruction cache w/o early start
        - Pipeline works with Instruction cache and early start
        - Pipeline works with data cache with write through
        - Pipeline works with data cache and write back
        - Pipeline works with data cache with write through and write back
        - Pipeline works with instruction and data caches with write through
        - Pipeline works with instruction and data caches with write back
        - Pipeline works with instruction and data caches with write through and write back

        - Other? Please specify
- Test results per Appendix B, Program 1
    - Written conclusion of the test results that also include for each simulated cache size what would be the preferred cache block size for instruction and data caches for that cache memory size if different cache block size is implemented for the instruction and data caches
- Test results per Appendix C, Program 2
    - Written conclusion of the test results that also include for each simulated cache size what would be the preferred cache block size for instruction and data caches for that cache memory size if different cache block size is implemented for the instruction and data caches
- Test results per bonus item for Unified cache running Program 1 (not required, bonus item only)
    - Written conclusion of the test results that also include for each simulated cache size what would be the preferred cache block size for instruction and data caches for that cache memory size if different cache block size is implemented for the instruction and data caches
- Test results per bonus item for 2-way set associative for program 1 (not required, bonus item only)
    - Written conclusion of the test results that also include for each simulated cache size what would be the preferred cache block size for instruction and data caches for that cache memory size if different cache block size is implemented for the instruction and data caches
- Conclusion of the combined results of Program 1, Program 2, and any bonus simulations
    - From these conclusions, can you make any general statements regarding pipelines and caches
- Detailed lessons learnt through doing this course project. These lessons learnt should be thing you learned through the process of doing this project, and not what was learned in lecture or reading the book.

Appendix E: Course Project Rubric

Course project grade is out of 25 points

Points for the course project will be broken down as follows:

	Potential points
• Simulator running modified program 2	5 pts
○ Simulator does not function to the level specified	-3
• Self-assessment of simulator	5
○ In accurate self-assessment	-3
• Program 1 test results & conclusion	5
• Program 2 test results & conclusion	5
• Simulation combined result summary	3
• Lessons learnt	2
• Total	25 pts

Note: Each category has a sliding scale per grading item, so partial credit is possible for each grade item.