The University of Alabama in Huntsville
Electrical & Computer Engineering Department
CPE 431 01
Final Exam Solution
Fall 2017

You must show your work to receive full credit!!

1. (1 point) _Spatial_ locality states that an item is likely to be referenced if its neighbor is referenced.
2. (1 point) _Benchmarks_ are programs specifically chosen to measure performance.
3. (1 point) _Clusters_ are networks of off-the-shelf, whole computers.
4. (1 point) A _cache coherency_ protocol maintains consistency in the value of data between several processors.
5. (1 point) _True_ (True or False) Overhead for communication is one of the reasons that it is difficult to write fast parallel processing programs.
6. (8 points) What number does 0xE09B AC00 represent, assuming the IEE 754 single precision format?

0xE09B AC00 = 1110 0000 1001 1011 1010 1100 0000 0000
1 1100 0001 0011 0111 0101 1000 0000 000
S = 1, the number is negative
Stored exponent is 12* 16 + 1 = 193, subtracting 127 gives 66. The fraction is 0011 0111 0101 1000 0000 000 = $(2^{20} + 2^{19} + 2^{17} + 2^{16} + 2^{15} + 2^{13} + 2^{11} + 2^{10})/2^{23} = 0.2162$
Number is $-1.2162 \times 2^{66} = -8.974$ E19

7. (15 points) Here is a series of address references given as hexadecimal word addresses: 21, 4, 8, 5, 20, 37, 19, 5E, 209, 11, 4, 43, 5, 3E, 8, 16, 59, 187, 2E8, 30. Assuming a direct mapped cache with four word blocks, a total size of 16 words that is initially empty, (a) label each reference in the list as a hit or a miss and (b) show the entire history of the cache, including tag and data.

Word Address (Hexadecimal)	Word Address (Decimal)	Binary	Miss/Hit
0x21	33	0000 0000 0010 **00** 01	**Miss**
0x4	4	0000 0000 0000 **01** 00	**Miss**
0x8	8	0000 0000 0000 **10** 00	**Miss**
0x5	5	0000 0000 0000 **01** 01	**Hit**
0x20	32	0000 0000 0010 **00** 00	**Hit**
0x37	55	0000 0000 0011 **01** 11	**Miss**
0x19	25	0000 0000 0001 **10** 01	**Miss**
0x5E	94	0000 0000 0101 **11** 10	**Miss**
0x209	521	0000 0010 0000 **10** 01	**Miss**
0x11	17	0000 0000 0001 **00** 01	**Miss**
0x4	4	0000 0000 0000 **01** 00	**Miss**
0x43	67	0000 0000 0100 **00** 11	**Miss**
0x5	5	0000 0000 0000 **01** 01	**Hit**
0x3E	62	0000 0000 0011 **11** 10	**Miss**
0x8	8	0000 0000 0000 **10** 00	**Miss**
0x16	22	0000 0000 0001 **01** 10	**Miss**
0x59	89	0000 0000 0101 **10** 01	**Miss**
0x187	391	0000 0001 1000 **01** 11	**Miss**
0x2E8	744	0000 0010 1110 **10** 00	**Miss**
0x30	48	0000 0000 0011 **00** 00	**Miss**

16 words x 1 block/4 words x 1 set/1 block = 4 sets
Byte offset = 0 bits, since there are word addresses only
Block offset = 2 bits, since they are four word blocks
Index = 2 bits, since there are four sets

Index	Tag	Data
0	~~2~~, ~~1~~, 4, 3	~~M[32..35]~~, ~~M[16..19]~~, ~~M[64..67]~~, M[48..51]
1	~~0~~, ~~3~~, ~~0~~, ~~1~~, 24	~~M[4..7]~~, ~~M[52..55~~], ~~M[4..7]~~, ~~M[20..23]~~, M[388..391]
2	~~0~~, ~~1~~, ~~32~~, ~~0~~, ~~5~~, 46	~~M[8..11]~~, ~~M[24..27]~~, ~~M[520..523]~~, ~~M[8..11]~~, ~~M[88..91]~~, M[744..747]
3	~~5~~, 3	~~M[92..95]~~, M[60..63]

8. (10 points) Consider the following portions of three programs running at the same time on three processors in a symmetric multicore processor (SMP). Assume that before this code is run, w is 4 x is -2 and y is 2 and z is 3. w, x, y, and z are type `int`. What are all the possible outcomes of executing these instructions?

Core 1: `y = 5/z + w;`
Core 2: `x = (x + y)/w + 1;`
Core 3: `z = w*(x - y) + z;`

	w	x	y	z
123	**4**	**1**	**5**	**-13**
132	**4**	**1**	**5**	**-25**
213	**4**	**1**	**5**	**-13**
231	**4**	**1**	**-1**	**-1**
312	**4**	**1**	**4**	**-13**
321	**4**	**1**	**4**	**-13**

9. (10 points) One of the biggest impediments to widespread use of virtual machines is the performance overhead incurred by running a virtual machine. Listed below are various performance parameters and application behavior.

Base CPI	Privileged O/S Accesses per 10,000 Instructions	Performance Impact to Trap to the Guest O/S	Performance Impact to Trap to VMM	I/O Accesses per 10,000 Instructions	I/O Access Time (Includes Time to Trap to Guest O/S)
1.5	120	15 cycles	175 cycles	30	1100 cycles

(a) (3 points) Calculate the CPI for the system listed above assuming that there are no accesses to I/O.
(b) (3 points) What is the CPI if the VMM performance impact doubles?
(c) (4 points) If a virtual machine software company wishes to obtain a 10% performance degradation, what is the longest possible penalty to trap to the VMM?

(a) CPI = 1.5 + 120/10000 × (15 + 175) = 3.78
(b) If VMM performance impact doubles: CPI = 1.5 + 120/10000 × (15 + 350) = 5.88

(c) For 10% degradation, $CPI_{degraded}$ = 1.65
For a performance impact to trap to VMM of 0 cycles, $CPI_{degraded}$ = 1.5 + 120/10000x 15 = 1.68.
Since 1.68 > 1.65, it is not possible.

10. (10 points) Consider program P, which runs on a 2 GHz machine M in 10 seconds. An optimization is made to P, replacing all instances of multiplying a value by 8 (mult X, X, 8) with three instructions that set X to X + X thrice (add X, X; add X, X; add X, X) Call this new optimized program P'. The CPI of a multiply instruction is 6, and the CPI of an add is 1. After recompiling, the program now runs in 8.5 seconds on machine M. How many multiplies were replaced by the new compiler?

ET_{before} = 10s, ET_{after} = 8.5 s, let x be the number of multiplies replaced
$ET_{affected}$ = 1.5s = (x * 6 cycles – 3(x * 1 cycle)/2 $\times$ 10^9 cycles/s
$3 \times 10^9 = 3x$
$x = 1 \times 10^9$

11. (7 points) The following list provides parameters of a virtual memory system.

Virtual Address (bits)	Physical DRAM Installed	Page Size	PTE Size (byte)
47	16 GiB	16 KiB	4

Using a multilevel page table can reduce the physical memory consumption of page tables, by only keeping active PTEs in physical memory.
(a) (5 points) How many levels of page tables will be needed in this case?
(b) (2 points) how many memory references are needed for address translation if missing in TLB?

Page offset = $\log_2$ 16 KiB = 14
Number of Virtual pages is $2^{47-14} = 2^{33}$, this is the number of entries in the page table

You want each level of the page table to fit in one page, so 16 KiB/4B = 4 KiB entries, using 12 bits each. So, there are 33 bits of virtual page number, requiring $\lceil 33/12 \rceil$ = 3 levels of page tables. Each address translation will take 3 memory accesses if not found in the TLB.

12. (20 points) Consider the following loop in C and this MIPS assembly version. $s6 is the pointer to the base of array A and $s3 holds the value of y.

```
for (i = 0; i < 120; i++)
   A[i] = A[i] + y
```

```
(a)          addi   $t0, $s6, 480          # $t0 ← &A[119]
(b)          addi   $t1, $s6, $zero        # $t1 ← &A[0]
(c) Loop:    lw     $t2, 0($t1)            # $t2 ← A[i]
(d)          add    $t2, $t2, $s3          # $t2 ← A[i] + y
(e)          sw     $t2, 0($t1)            # A[i] ← A[i] + y
(f)          addi   $t1, $t1, 4            # $t1 ← &A[i+1]
(g)          bne    $t0, $t1, Loop
```

Dependences: (b) – (c) (first time only), (c) – (d), (d) – (e), (f) – (g), (f) – (c) (all iterations other than the first)

Calculate the number of cycles it takes to execute this complete code snippet if

(a) (3 points) There is full forwarding and the bne completes in the EX stage

Cycle	IF	ID	EX	MEM	WB
1	a				
2	b	a			
3	c	b	a		
4	d	c	b	a	
5	e	d	c	b	a
6	e	d	bubble	c	b
7	f	e	d	bubble	c
8	g	f	e	d	bubble
9	after1	g	f	e	d
10	after2	after1	g	f	e
11	c	bubble	bubble	g	f

Total number of cycles: 2 initial instructions + 119 iterations x (5 instructions + 1 data hazard stall + 2 data control stalls)/iteration + 1 iteration (5 instructions + 1 data hazard stall) = 960

(b) (3 points) There is EX/MEM forwarding only and the bne completes in the EX stage

Cycle	IF	ID	EX	MEM	WB
1	a				
2	b	a			
3	c	b	a		
4	d	c	b	a	
5	e	d	c	b	a
6	e	d	bubble	c	b
7	e	d	bubble	bubble	c
8	f	e	d	bubble	bubble
9	g	f	e	d	bubble
10	after1	g	f	e	d
11	after2	after1	g	f	e
12	c	bubble	bubble	g	f

Total number of cycles: 2 initial instructions + 119 iterations x (5 instructions + 2 data hazard stall + 2 data control stalls)/iteration + 1 iteration (5 instructions + 2 data hazard stall) = 1080

(c) (3 points) There is MEM/WB forwarding only and the bne completes in the EX stage

Cycle	IF	ID	EX	MEM	WB
1	a				
2	b	a			
3	c	b	a		
4	d	c	b	a	
5	d	c	bubble	b	a
6	e	d	c	bubble	b
7	e	d	bubble	c	bubble
8	f	e	d	bubble	c
9	f	e	bubble	d	bubble
10	g	f	e	bubble	d
11	after1	g	f	e	bubble
12	after1	g	bubble	f	e
13	after2	after1	g	bubble	f
14	c	bubble	bubble	g	bubble

Total number of cycles: 2 initial instructions + 1 data hazard stall + 119 iterations x (5 instructions + 3 data hazard stalls + 2 data control stalls)/iteration + 1 iteration (5 instructions + 3 data hazard stalls) = 1201

Then,

(d) (9 points) unroll this loop so that two iterations of the loop are done at a time and schedule the unrolled code for execution with full forwarding and branch completion in the EX stage.

```
(a)          addi   $t0, $s6, 480          # $t0 ← &A[119]
(b)          addi   $t1, $s6, $zero        # $t1 ← &A[0]
(c)Loop:     lw     $t2, 0($t1)            # $t2 ← A[i]
(d)          lw     $t3, 4($t1)            # $t2 ← A[i+1]
(e)          add    $t2, $t2, $s3          # $t2 ← A[i] + y
(f)          add    $t3, $t2, $s3          # $t2 ← A[i+1] + y
(g)          sw     $t2, 0($t1)            # A[i] ← A[i] + y
(h)          sw     $t3, 4($t1)            # A[i] ← A[i+1] + y
(i)          addi   $t1, $t1, 8            # $t1 ← &A[i+2]
(j)          bne    $t0, $t1, Loop
```

Cycle	IF	ID	EX	MEM	WB
1	a				
2	b	a			
3	c	b	a		
4	d	c	b	a	
5	e	d	c	b	a
6	f	e	d	c	b
7	g	f	e	d	c
8	h	g	f	e	d
9	i	h	g	f	e
10	j	i	h	g	f
11	after1	j	i	h	g
12	after2	after1	j	i	h
13	c	bubble	bubble	j	i

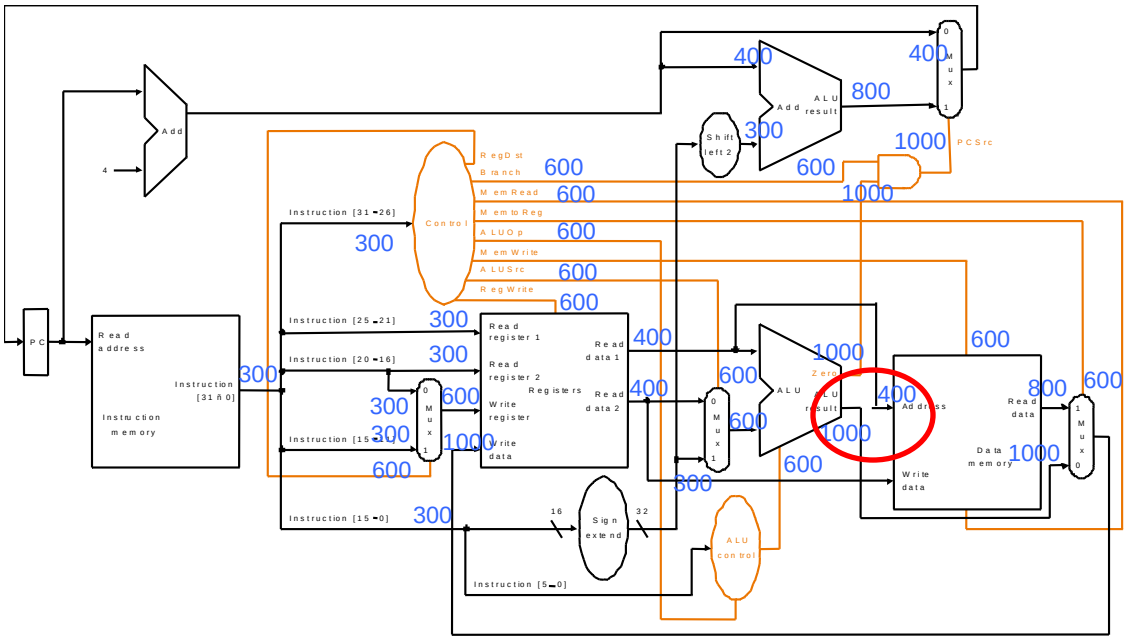
Total number of cycles: 2 initial instructions + 59 iterations x (8 instructions + 0 data hazard stalls + 2 data control stalls)/iteration + 1 iteration (8 instructions + 0 data hazard stalls) = 600

(e) (2 points) Compare the execution times of the original unrolled code from part (a) and the unrolled, scheduled code from part (d).
Speedup = 960 cycles/600 cycles = 1.6

13. (15 points) Consider the following idea: Let's modify the instruction set architecture and remove the ability to specify an offset for memory access instructions. Specifically, all load-store instructions with nonzero offsets would become pseudoinstructions and would be implemented using two instructions. For example:

```
addi $at, $t1, 104        # add the offset to a temporary
lw   $t0, $at             # new way of doing lw $t0, 104 ($t1)
```

(a) (5 points) Show the changes to the single-cycle datapath and control if this simplified architecture is used. **See change in red circle ALU output bypasses memory and address comes from Read data1**



Instruction	RegDst	ALUSrc	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0	
R-format	1	0	0	1	0	0	0	1	0	
lw(new and old)	0	1	1	1	1	0	0	0	0	
sw(new and old)	d	1	d	0	0	1	0	0	0	
beq	d	0	d	0	0	0	1	0	1	

d-don't care

(b) (6 points) If the delay for the instruction memory is 300 ps, the data memory is 200 ps, the register file delay (read or write) is 100 ps, the control (not ALU control) is 300 ps, and the ALU delay is 400 ps, and all other delays are neglible, what are the clock cycle times required for the original datapath and for the modified datapath?

(c) (4 points) What is the highest percentage of load-store instructions with offsets that could be tolerated without total performance being degraded?

lw is the longest for the unmodified datapath

Consider lw, sw, R-type, beq, jump for modified datapath

CC_{lw} = 300(fetch)+max(300, 100)(control, register file read)+200(data memory read)+100(register file write) = 300 + 300 + 200 + 100 = 900 ps

CC_{sw} = 300(fetch) + max(300, 100)(control, register file read) + 200(data memory write) = 300 + 300 + 200 = 800 ps

CC_{Rtype} = 300(fetch) + max(300, 100)(control, register file read) + 400(ALU) + 100(register file write) = 300 + 300 + 400 + 100 = 1100 ps

Cc_{beq} = max((300(fetch) + max(300, 100)(control, register file read) + 400(ALU)), max(400, 300)(ADD, control) + 400 (ALU)) = max(300+300+400, 400+400) = max(1000, 800) = 1000 ps

Cc_{jump} = 300(fetch) = 300 ps

CC_{orig} = CC_{lw}= 300(fetch) + max(300, 100)(control, register file read) + 400(ALU) + 200(data memory read) + 100(register file write) = 300 + 300 + 400 + 200 + 100 = 1300 ps

CC_{mod} = max(CC_{lw}, CC_{sw}, CC_{Rtype}, Cc_{beq}, Cc_{jump}) = max(900, 800, 1100, 1000, 300) = 1100 ps

For every load-store with a nonzero offset, an additional instruction must be executed. Thus, if the fraction of instructions which is a lw with a nonzero offset is x, the number of modified instructions is IC_{orig} (1 + x). Remembering that CPI_{mod} = CPI_{orig} = 1

$$\frac{P_{\text{mod}}}{P_{orig}} = \frac{ET_{orig}}{ET_{\text{mod}}} = \frac{IC_{orig} * CPI_{orig} * CC_{orig}}{IC_{\text{mod}} * CPI_{\text{mod}} * CC_{\text{mod}}} = \frac{IC_{orig} * 1 * 1300\,ps}{IC_{orig}(1+x) * 1 * 1100\,ps} = 1$$

IC_{orig}*1*1.3 ns = IC_{orig}(1 + x)*1*1.1 ns

1.1818 = 1 + x and x = 0.1818, or 18.2 %