**The University of Alabama in Huntsville**
**Electrical & Computer Engineering Department**
**CPE 431 01**
**Final Exam Solution**
**Fall 2014**

1. (1 point) A _**cluster**_ is a set of computers connected over a local area network that function as a single large multiprocessor.
2. (1 point) _**Strong scaling**_ is the speedup achieved on a multiprocessor without increasing the size of the problem.
3. (1 point) _**Data level**_ parallelism achieved by performing the same operation on independent data.
4. (1 point) A _**process**_ includes one or more threads, the address space, and the operating system state.
5. (1 point) A _**reduction**_ is a function that processes a data structure and returns a single value.
6. (7 points) Write down the binary representation of the decimal number 159.375 assuming the IEE 754 double precision format. Express your answer in hexadecimal.

```
159
79          1
39          1
19          1
9           1
4           1
2           0
1           0
0           1

0.375
0.750
1.5
1.0

159.375 = 10011111.011
Normalizing 1.0011111011 x 2^7
Exponent add offset of 1023 = 1030 10000000110
Sign = 0 for positive number
Fraction = 0011111011

0100 0000 0110 0011 1110 1100 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000

In hex 4063 EC00 0000 0000
```

7. (15 points) Here is a series of address references given as hexadecimal word addresses: 1, 4, 8, 5, 20, 17, 19, 56, 209, 11, 4, 43, 5, 36, 8, 16, 59, 187. 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.

$$16words \times \frac{1block}{4words} \times \frac{1set}{1blocks} = 4sets$$

Index block offset

| | | |
|---|---|---|
| 0x1 | **0000 0000 00 01** | **miss** |
| 0x4 | **0000 0000 01 00** | **miss** |
| 0x8 | **0000 0000 10 00** | **miss** |
| 0x5 | **0000 0000 01 01** | **hit** |
| 0x20 | **0000 0010 00 00** | **miss** |
| 0x17 | **0000 0001 01 11** | **miss** |
| 0x19 | **0000 0001 10 01** | **miss** |
| 0x56 | **0000 0101 01 10** | **miss** |
| 0x209 | **0010 0000 10 01** | **miss** |
| 0x11 | **0000 0001 00 01** | **miss** |
| 0x4 | **0000 0000 01 00** | **miss** |
| 0x43 | **0000 0100 00 11** | **miss** |
| 0x5 | **0000 0000 01 01** | **hit** |
| 0x36 | **0000 0011 01 10** | **miss** |
| 0x8 | **0000 0000 10 00** | **miss** |
| 0x16 | **0000 0001 01 10** | **miss** |
| 0x59 | **0000 0101 10 01** | **miss** |
| 0x187 | **0001 1000 0111** | **miss** |

All numbers shown are hex

| | Tag | Data |
|---|---|---|
| 0 | ~~0~~, ~~2~~, ~~1~~, 4 | ~~M[0..3]~~, ~~M[20..23]~~, ~~M[10..13]~~, M[40..43] |
| 1 | ~~0~~, ~~1~~, ~~5~~, ~~0~~, ~~3~~, ~~1~~, 18 | ~~M[4..7]~~, ~~M[14..17]~~, ~~M[54..57]~~, ~~M[4..7]~~, ~~M[34..37]~~, ~~M[14..17]~~, M[184..187] |
| 2 | ~~0~~, ~~1~~, ~~20~~, ~~0~~, 5 | ~~M[8..B]~~, ~~M[18..1B]~~, ~~M[208..20B]~~, ~~M[8..B]~~, M[58..5B] |
| 3 | | |

8. (8 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 2, x is 4 and y is 3 and z is 1. w, x, y, and z are type `int`.

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

| Order | w | x | y | z |
|---|---|---|---|---|
| 1, 2, 3 | 2 | 5 | 1 | 8 |
| 1, 3, 2 | 2 | 5 | 1 | 6 |
| 2, 1, 3 | 2 | 6 | 1 | 10 |
| 2, 3, 1 | 2 | 6 | 0 | 6 |
| 3, 1, 2 | 2 | 5 | 1 | 2 |
| 3, 2, 1 | 2 | 6 | 1 | 2 |

9. (7 points) . Consider a computer running a program that requires 750 s, with 70s spent executing FP instructions, 85 s executing L/S instructions, and 40 s spent executing branch instructions and the rest executing R Type instructions. By how much must we improve the CPI of R Type instructions if we want the program to run two times faster?

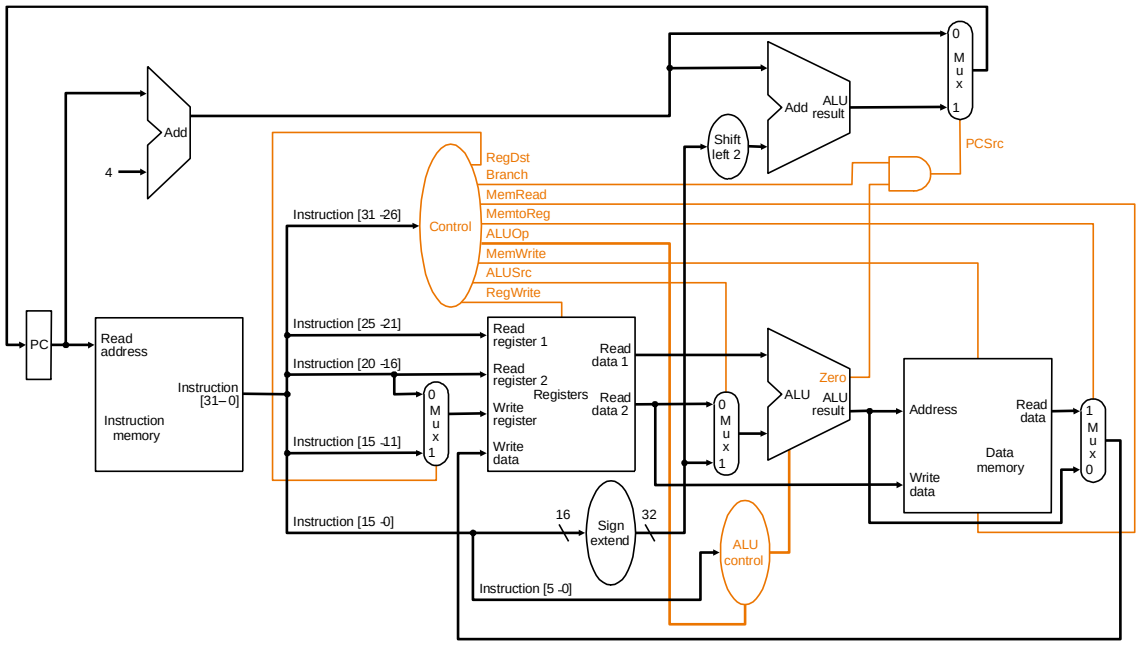
**$ET_{FP}$ = 70 s**
**$ET_{L/S}$ = 85 s**
**$ET_{branch}$ = 40 s**
**$ET_{total}$ = 750 s**
**$ET_{R\text{-}type}$ = 750 – 70 – 85 – 40 = 555 s**
**$ET_{new}$ = 375 s**
**$ET_{R\text{-}type\ new}$ = 375 – 70 – 85 – 40 = 180**

**CPI of r-type must improve by 555/180 = 3.08**

10. (15 points) Add a variant of the `lw` instruction which sums the contents of two registers to obtain the address of the data and which uses the R format to the single-cycle datapath shown in the figure below. Add any necessary datapaths and control signals and show the necessary additions to the table of control signals given.

```
lwv   $9, $6, $22
```
$9 ← MEM[$6+$22]

| 36 | 6 | 22 | 9 | 0 | 32 |
|---|---|---|---|---|---|




| Instruction | RegDst | ALUSrc | Memto-Reg | Reg Write | Mem Read | Mem Write | Branch | ALU Op1 | ALU Op0 | |
|---|---|---|---|---|---|---|---|---|---|---|
| R-format | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | |
| lw | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | |
| sw | d | 1 | d | 0 | 0 | 1 | 0 | 0 | 0 | |
| beq | d | 0 | d | 0 | 0 | 0 | 1 | 0 | 1 | |
| lwv | **1** | **0** | **1** | **1** | **1** | **0** | **0** | **0** | **0** | |

d – don't care

**No additional hardware is required.**

11. (12 points) The following data constitutes a stream of virtual addresses as seen on a system. Assume 8 KiB pages, a 4-entry fully associative TLB, and true LRU replacement. If pages must be brought in from disk, increment the next largest page number.

4669, 2227, 13916, 34587, 18885, 12608, 49225, 9226, 46390

**TLB**

| Valid | Tag | Physical Page Number |
|---|---|---|
| 1 | 0 | 5 |
| 1 | 7 | 4 |
| 1 | 3 | 6 |
| 0 | 4 | 9 |

**Page table**

| Valid | Physical page or in disk |
|---|---|
| 1 | 5 |
| 0 | Disk |
| 0 | Disk |
| 1 | 6 |
| 1 | 9 |
| 1 | 11 |
| 0 | Disk |
| 1 | 4 |
| 0 | Disk |
| 0 | Disk |
| 1 | 3 |
| 1 | 12 |

Given the address stream, and the shown initial state of the TLB and page table, show the final state of the system. Also list for each reference if it is a hit in the TLB, a hit in the page table, or a page fault.

**These are byte addresses. The page offset is $\log_2$ 8K = 13. The virtual page number can be obtained by dividing the byte address by 8KB (8192) and taking the integer part of the result.**

| | VPN | TLB? | PT? | PF? |
|---|---|---|---|---|
| 4669 | 0 | H | H | N |
| 2227 | 0 | H | H | N |
| 13916 | 1 | M | M | Y |
| 34587 | 4 | M | H | N |
| 18885 | 2 | M | M | Y |
| 12608 | 1 | H | H | N |
| 49225 | 6 | M | M | Y |
| 9226 | 1 | H | H | N |
| 46390 | 5 | M | H | N |

TLB

| Valid | Tag | Physical Page Number |
|---|---|---|
| 1 | ~~0~~, 6 | ~~5~~, 15 |
| 1 | ~~7~~, 4, 5 | 4, ~~9~~, 11 |
| 1 | ~~3~~, 2 | ~~6~~, 14 |
| ~~0~~, 1 | 4, 1 | ~~9~~, 13 |

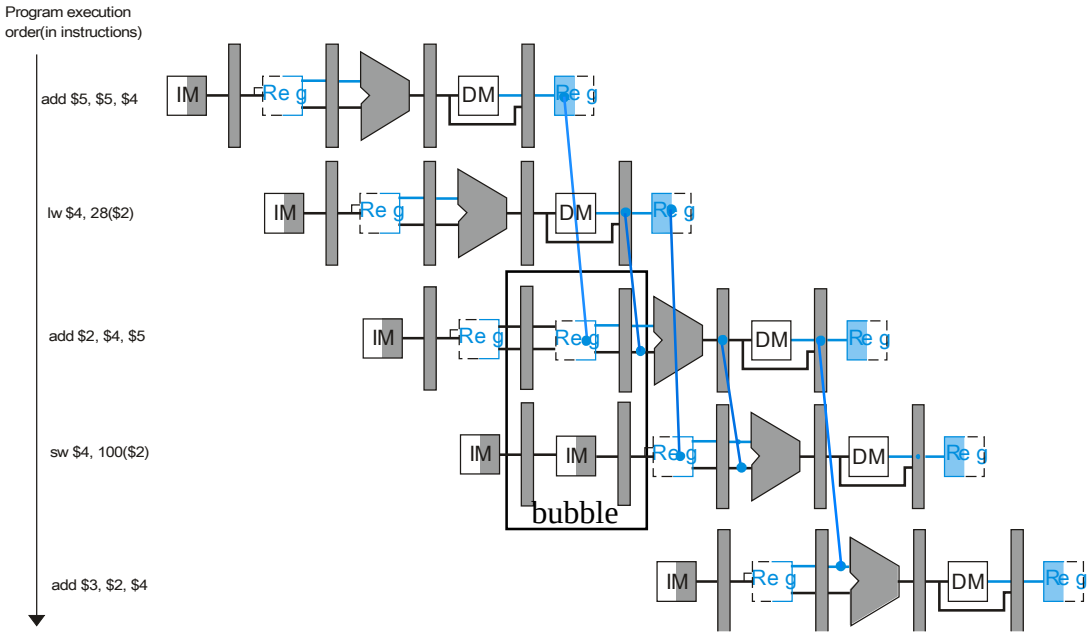
Page table

| | Valid | Physical page or in disk |
|---|---|---|
| 0 | 1 | 5 |
| 1 | ~~0~~, 1 | ~~Disk~~, 13 |
| 2 | 0 | Disk, 14 |
| 3 | 1 | 6 |
| 4 | 1 | 9 |
| 5 | 1 | 11 |
| 6 | ~~0~~, 1 | ~~Disk~~, 15 |
| 7 | 1 | 4 |
| 8 | 0 | Disk |
| 9 | 0 | Disk |
| 10 | 1 | 3 |
| 11 | 1 | 12 |

12. (15 points) (a) Identify all of the data dependencies in the following code. (b) How is each data dependency either handled or not handled by forwarding? Draw a multiple clock cycle style diagram to support your answer.

```
a       add   $5, $5, $4
b       lw    $4, 28($2)
c       add   $2, $4, $5
d       sw    $4, 100($2)
e       add   $3, $2, $4
```

| Dependencies | Hazard? | |
|---|---|---|
| a-c | yes | forward from MEM/WB pipeline register |
| b-c | yes | stall, forward from MEM/WB pipeline register |
| b-d | no | data is available in register file after stall |
| b-e | no | data is available in register file |
| c-d | yes | forward from EX/MEM pipeline register |
| c-e | yes | forward from MEM/WB pipeline register |



13. (8 points) Pseudoinstructions are not part of the MIPS instruction set but often appear in MIPS programs. For the pseudoinstruction listed, produce a minimal sequence of actual MIPS instructions to accomplish the same thing. You may need to use $at for some of the sequences. In the table, big refers to a specific number that requires 32 bits to represent and small to a number that can fit in 16 bits.

| Pseudoinstruction | What it accomplishes |
|---|---|
| lw $t5, big($t2) | $t5 = Memory[$t2 + big] |

```
lui     $at, big[31..16]
ori     $at, $at, big[15..0]
add     $at, $at, $t2
lw      $t5, 0($at)
```

14. (8 points) Consider a SEC code that protects 4 bit words with 3 parity bits. If we read the value 0x57, is there an error? If so, correct the error.

```
123 4567
101 0111
```

**C4 = XOR (4, 5, 6, 7) = XOR(0, 1, 1, 1) = 1**
**C2 = XOR (2, 3, 6, 7) = XOR(0, 1, 1, 1) = 1**
**C1 = XOR (1, 3, 5, 7) – XOR(1, 1, 1, ,1) = 0**

**Bit 6 is incorrect, correct is 0x55**