The University of Alabama in Huntsville
Electrical and Computer Engineering Department
CPE 221 01
Test 2 Solution
Spring 2016

1. (1 point) **ld** instructions read from memory in stage **_ME(4)_** of the SRC pipeline.
2. (1 point) Branches that do not link complete in stage **_ID(2)_** of the SRC pipeline.
3. (1 point) In certain circumstances, a data dependence can become a data **_hazard_.**
4. (1 point) In the microcoding described in this class, a PLA uses the **_opcode_** of an instruction to determine where the microcode for that instruction is located.
5. (1 point) A **_reset_** capability is one that initializes a processor to a known, defined state.

6. (4 points) r2 contains a value of -720 in decimal. What is the decimal value of r1 after this instruction is executed?

```
shc r1, r2, 10
```

-720 = -1024 + 256 + 32 + 16 = 101_0011_0000
r2 = 1111_1111_1111_1111_1111_1101_0011_0000
r1 = 1111_1111_1111_0100_1100_0011_1111_1111
-r1 = 0000_0000_0000_1011_0011_1100_0000_0001 = $2^{19} + 2^{17} + 2^{16} + 2^{13} + 2^{12} + 2^{11} + 2^{10} + 2^{0}$
736257=
r1 - -736257

7. (4 points) r2 contains a value of -495 in decimal. What is the decimal value of r1 after this instruction is executed?

```
shra r1, r2, 5
```

-495 = -512 + 16 + 1 = 10_0001_0001
r2 = 1111_1111_1111_1111_1111_1110_0001_0001
r1 = 1111_1111_1111_1111_1111_1111_1111_0000
-r1 = 0000_0000_0000_0000_0000_0000_0001_0000 = 2^{4} = 16
r1 = -16

8-12. (20 points) What are the values of the following registers and memory addresses when the program executes "brpl r31, r5" for the second time? Answer in decimal.

8. r31: **_1076_** 9. r5: **_-9_** 10. r10: **_212_** 11. r3: **_0_** 12. r4: **_1_**

```
                .org 200
200   size:     .dc 10
204   neg_count: .dw 1
208   pos_count: .dw 1
212   nums:     .dc 100, 3, -1, 2, 4, 37, -100, 13, -5, 0
      orig:     .org 1000
1000            la   r10, nums      r10=212
1004            ld   r1, size       r1=10
1008            la   r2, 0          r2=0
1012            la   r3, 0          r3=0
1016            la   r4, 0          r4=0
1020            la   r31, done      r31=1076
1024            la   r30, loop      r30=1032
1028            la   r29, negative  r29=1068
1032  loop:     sub  r5, r2, r1     r5=0-10=-10,   r5=1-10=-9
1036            brpl r31, r5        PC=1040
1040            shl  r5, r2, 2      r5=0*4=0
1044            add  r5, r5, r10    r5=212+0=212
1048            ld   r5, 0(r5)      r5=M[212]=100
1052            addi r2, r2, 1      r2=0+1=1
1056            brmi r29, r5        PC=1060
1060            addi r4, r4, 1      r4=0+1=1
1064            br   r30            PC=1032
1068  negative: addi r3, r3, 1
1072            br   r30
1076  done:     st   r3, neg_count
1080            st   r4, pos_count
1084            stop
```

13. (10 points) Identify all of the data dependencies in the following code.

```
(1)  add   r3, r5, r4
(2)  ld    r4, 28(r1)
(3)  add   r5, r4, r3
(4)  st    r5, 100(r4)
(5)  add   r6, r1, r8
(6)  brpl  r29, r9
```

(3) uses the r3 result produced by (1)
(3) uses the r4 result produced by (2)
(4) uses the r4 result produced by (2)
(4) used the r5 result produced by (3)

14. (2 points) If the clock frequency is 750 MHz, what is the clock period?

T = 1/f = 1/750 MHz = 1.33 ns

15. (10 points) Design a microcode sequence to implement the 1-bus SRC *lddr* instruction you created on Test 1 using the concrete RTN shown below.

address	100	101	102	700	701	702	703	704
Mux control	00	00	01	00	00	00	00	11
PC_{out}	1							
C_{out}		1				1		
MD_{out}			1					1
MD_{in}								
R_{out}				1	1			
$c2_{out}$								
BA_{out}								
MA_{in}	1					1		
C_{in}	1				1			
A_{in}				1				
R_{in}								1
PC_{in}		1						
IR_{in}			1					
ADD					1			
OR								
Wait		1					1	
Read		1					1	
AND								
INC4	1							
Gra								1
Grb				1				
Grc					1			
End								1
Address	d	d	d	d	d	d	d	100

Step	**RTN for the lddr Instruction**	**Control Sequence**
T0	MA ← PC: C ← PC + 4;	PC_{out}, MA_{in}, INC4, C_{in}
T1	MD ← M[MA]: PC ← C;	C_{out}, PC_{in}, Read, Wait
T2	IR ← MD;	MD_{out}, IR_{in}
T3	A ← R[rb];	Grb, R_{out}, A_{in}
T4	C ← A + R[rc];	C_{in}. R_{out}, Grc, ADD
T5	MA ← C;	MA_{in}, C_{out}
T6	MD ← M[MA];	Read, Wait
T7	R[ra] ← MD	Gra, R_{in}, End, MD_{out}

Microcode Branching Examples

Address	Mux Ctl	BrUn	BrNotZ	BrZ	BrNotN	BrN	Branch Address	Branching Action
200	00	0	0	0	0	0	ddd	None -201 next
201	01	1	0	0	0	0	ddd	To output of PLA
202	10	0	0	1	0	0	ddd	To external address if Z
203	11	0	0	0	0	1	300	To 300 if N (else 204)

16. (30 points) Complete the SRC assembly language program below so that it implements the following C++ statements.

```
const int size = 10;
int array1[size] = {100, 3, -1, 2, 4, 4, 2, -1, 3, 100};
int array2[size] = {-53, 247, 95, -7, 481, 91, -33, 1500, 29, -83};
int array3[size];
int i;
for (i = 0; i < size; i++)
  array3[i] = array1[i] + array2[i];
```

```
;
;      This program sums two arrays and puts the resulting values
;       in a third array.
;
            .org    200
size:       .dc     10
array1:     .dc     100, 3, -1, 2, 4, 4, 2, -1, 3, 100
array2:     .dc     -53, 247, 95, -7, 481, 91, -33, 1500, 29, -83
array3:     .dw     10
i:          .dc     0
orig:       .org    1000
            la      r31, done
            la      r30, loop
            la      r10, array1         ; pointer to first element of array1
            la      r11, array2         ; pointer to first element of array2
            la      r12, array3         ; pointer to first element of array3
            ld      r1, size            ; holds size of arrays
            ld      r2, i               ; holds loop counter
loop:       sub     r5, r2, r1          ; Calculate i - size
            brpl    r31, r5             ; if i - size is positive, we are done
            ld      r3, 0(r10)          ; r3 = array1[i]
            ld      r4, 0(r11)          ; r4 = array2[i]
            add     r5, r3, r4          ; r5 = array1[i] + array2[i]
            st      r5, 0(r12)          ; array3[i] = array1[i] +array2[i]
            addi    r10, r10, 4         ; point to the next element of array1
            addi    r11, r11, 4         ; point to the next element of array2
            addi    r12, r12, 4         ; point to the next element of array3
            br      r30
done:       stop
```

17. (15 points) Trace the code fragment given through a 5-stage SRC pipeline that has forwarding from the output of stage 3 to the input of stage 3 and from the output of stage 4 to the input of stage 4. No other forwarding is available. Insert nop bubbles into the pipelines as needed to resolve any dependences. Describe how any forwarding that is happening is being accomplished. Use the table if you find it helpful, it is not required. If you don't use it, clearly show how many cycles it takes to execute this code fragment. You may not use all of the rows. If you need more rows, add them.

```
(1) shr  r3, r3, 2
(2) sub  r2, r3, r1
(3) ld   r4, 0(r3)
(4) add  r6, r4, r1
```

The dependences are

from r3 in (1) to r3 in (2)	**Handled by 3 to 3 forwarding, no stalls**
from r3 in (1) to r3 in (3)	**No 4 to 3 forwarding, so cannot forward, 2 stalls**
	With 4 to 3 forwarding, no stalls
from r4 in (3) to r4 in (4)	**No 4 to 3 forwarding, 3 stalls**
	With 4 to 3 forwarding, 1 stall

Cycle	IF	ID	EX	ME	WB
1	**shr**				
2	**sub**	**shr**			
3	**nop**	**sub**	**shr**		
4	**nop**	**nop**	**sub**	**shr**	
5	**ld**	**nop**	**nop**	**sub**	**shr**
6	**nop**	**ld**	**nop**	**nop**	**sub**
7	**nop**	**nop**	**ld**	**nop**	**nop**
8	**nop**	**nop**	**nop**	**ld**	**nop**
9	**add**	**nop**	**nop**	**nop**	**ld**
10		**add**	**nop**	**nop**	**nop**
11			**add**	**nop**	**nop**
12				**add**	**nop**
13					**add**