The University of Alabama in Huntsville
Electrical and Computer Engineering Department
CPE 221 01
Fall 2012
Final Exam Solution

1. (1 point). A _**compiler**_ maps a high-level language into machine language constructs.
2. (1 point) The _**instruction set architecture**_ forms the interface between the program and the functional units of the computer
3. (1 point) _**Simulators**_ mimic hardware performance in software, they are usually slower in operation by orders of magnitude
4. (1 point) A 1 address instruction has a register, called the _**accumulator**_to hold one operand & the result.
5. (1 point) RTN is a _**meta**_ language – a language used to describe a language.
6. (4 points) `r2` contains a value of -39452 in decimal. What is the binary value of `r1` after this instruction is executed?

```
not r1, r2
```

```
r2 = -39452 =     1111 1111 1111 1111 0110 0101 1110 0100 = FFFF 65E4
r1 =              0000 0000 0000 0000 1001 1010 0001 1011 = 0000 9A1B
```

7. (4 points) `r2` contains a value of -5179 in decimal. What is the binary value of `r1` after this instruction is executed?

```
neg r1, r2
```

```
r2 = -5179
r1 = 5179   0000 0000 0000 0000 0001 0100 0011 1011 = 0000 143B
```

8. (4 points) `r2` contains a value of -5179 in decimal while `r3` contains a value of 3058 in decimal. What is the binary value of `r1` after this instruction is executed?

```
or r1, r2, r3
```

```
r2 = -5179 =      1111 1111 1111 1111 1110 1011 1100 0101 = FFFF EBC5
r3 = 3058  =      0000 0000 0000 0000 0000 1011 1111 0010 = 0000 0BF2

r1 =              1111 1111 1111 1111 1110 1011 1111 0111 = FFFF EBF7
```

9. (4 points) If we want to examine the last bit of a binary number to see whether it was 0 or 1, we use a mask with a value of 1 and and the mask with the number. What mask value would we use if we wanted to examine bit 9?

$2^9 = 512$

10. (10 points) Encode `11001110010` using the Hamming code and odd parity. What is the final Hamming code?

P_1 = XOR(1, D_3, D_5, D_7, D_9, D_{11}, D_{13}, D_{15}) = XOR(1, 1, 1, 0, 1, 1, 0, 0) = 1
P_2 = XOR(1, D_3, D_6, D_7, D_{10}, D_{11}, D_{14}, D_{15}) = XOR(1, 1, 0, 0, 1, 1, 1, 0) = 1
P_4 = XOR(1, D_5, D_6, D_7, D_{12}, D_{13}, D_{14}, D_{15}) = XOR(1, 1,0, 0, 0, 0, 1, 0) = 1
P_8 = XOR(1, D_9, D_{10}, D_{11}, D_{12}, D_{13}, D_{14}, D_{15}) = XOR(1, 1, 1, 1, 0, 0, 1, 0) = 1

Encoded Word `111_1100_1111_0010`

11. (10 points), Extract the correct data value from the odd parity SECDED string `1111101100011100`.

P = XOR(P_0, P_1, P_2, D_3, P_4, D_5, D_6, D_7, P_8, D_9, D_{10}, D_{11}, D_{12}, D_{13}, D_{14}, D_{15})
= XOR(1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0) = 0, for odd parity, it should be 1

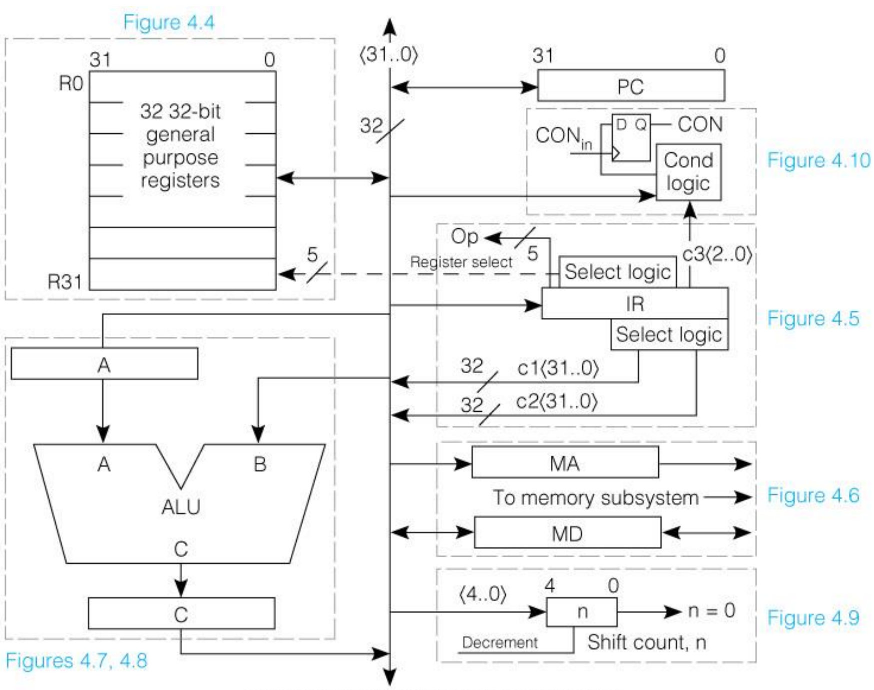
C_1 = XOR(P_1, D_3, D_5, D_7, D_9, D_{11}, D_{13}, D_{15}) = XOR(1, 1, 0, 1, 0, 1, 1, 0) = 1
C_2 = XOR(P_2, D_3, D_6, D_7, D_{10}, D_{11}, D_{14}, D_{15}) = XOR(1, 1, 1, 1, 0, 1, 0, 0) = 1
C_4 = XOR(P_4, D_5, D_6, D_7, D_{12}, D_{13}, D_{14}, D_{15}) = XOR(1, 0,1, 1, 1, 1, 0, 0) = 1
C_8 = XOR(P_8, D_9, D_{10}, D_{11}, D_{12}, D_{13}, D_{14}, D_{15}) = XOR(0, 0, 0, 1, 1, 1, 0, 0) = 1

C = 0, P_0 = 0 mean that P_0 is in error, so the correct data is `101_1001_1100`

12. (3 points) The fields `ra`, `rb`, and `rc` in the SRC instruction format are 5 bits long. If the register file were enlarged to contain 128 registers, how many bits are required for each of these fields?

$\log_2 128 = 7$

13. (15 points) Write concrete RTN steps for the SRC instruction `brzrl` using the 1-bus SRC microarchitecture shown.



Copyright © 2004 Pearson Prentice Hall, Inc.

T0	C ← PC + 4; MA ← PC
T1	PC ← C; MD ← M[MA]
T2	IR ← MD
T3	CON ← R[rc] = 0 ∧ Cond(c3 = 3)
T4	R[ra] ← PC
T5	CON → PC ← R[rb]

14. (10 points) Compute the total memory traffic in bytes for both instruction fetch and instruction execution for the code below.

```
        ld    r8, 0(r13)
        shl   r13, r6, 2
        add   r13, r13, r12
        sub   r9, r7, r8
        brpl  r29, r9
lless:  st    r7, 0(r13)
        addi  r4, r4, 1
```

Each instruction is 4 bytes, so the total traffic for fetch is 7 instructions * 4 bytes/instruction =-28 bytes. Additionally, the ld and st instructions generate 4 bytes of data traffic each for 8 bytes, the total is 28 + 8 = 36 bytes

15. (6 points) Encode the `brnz` statement from the SRC program shown below in hexadecimal.

```
cnt:    .equ 8
        .org 0
seq:    .dc 1
next:   .dc 1
ans:    .dw cnt
        .org 1000
         lar   r31, loop
         la    r0, cnt
         la    r1, seq
loop:    ld    r2, 0(r1)
         ld    r3, 4(r1)
         add   r2, r2, r3
         st    r2, 8(r1)
         addi  r1, r1, 4
         addi  r0, r0, -1
         brnz  r31, r0
```

Instruction	op	ra	rb	rc	c1	c2	c3
`brnz r31, r0`	**8**	**0**	**31**	**0**			**3**

`01000 00000 11111 00000 000000000 011 = 0x403E_0003`

16. (25 points) Complete the SRC assembly language program below so that it implements the following C++ statements.

```
int neg_count = 0;
int pos_count = 0;
int size = 10;
int nums[10] = {5, 3, -1, 2, 4, 37, -100, 13, -5, 0};
for (i = 0; i < size; i++)
   if (nums[i] < 0)
      neg_count++;
   else
      pos_count++;
```

```
;
;      This program computes two counts: pos_count – the number of positive
;      numbers and neg_count – the number of negative numbers.
;
            .org   200
size:       .dc    10
neg_count:  .dw    1
pos_count:  .dw    1
nums:       .dc    100, 3, -1, 2, 4, 37, -100, 13, -5, 0
orig:       .org   1000

            lar    r10, nums          ; pointer to first element of array
            ld     r1, size           ; holds size of array
            la     r2, 0              ; i = 0, index1 into array

            la     r3, 0              ; neg_count = 0
            la     r4, 0              ; pos_count = 0
            lar    r31, done
            lar    r30, loop
            lar    r29, negative

loop:       sub    r5, r2, r1         ; Check to see whether i < size.
            brpl   r31, r5            ; If not, done.
            shl    r5, r2, 2          ; Multiply i by 4 to access entry
                                      ; in array by byte address.
            add    r5, r5, r10        ; Add index to base array pointer.
            ld     r5, 0(r5)          ; Load nums[i] into r5.
            addi   r2, r2, 1          ; i++
            brmi   r29, r5            ; If nums[i] < 0, go to negative.
            addi   r4, r4, 1          ; pos_count++
            br     r30                ; Go back to loop.
negative:   addi   r3, r3, 1          ; neg_count++
            br     r30                ; Go back to loop.
done:       st     r3, neg_count
            st     r4, pos_count

            stop
```