Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

CEG2136: Computer Architecture I
CEG2536: Architecture des Ordinateurs I

MIDTERM EXAMINATION

Profs. Voicu Z. Groza and Naim M. Rahmani

Duration: 1 hour and 30 minutes   November 5, 2011, 10:00

Question 1

1.1 (12 x 1.5 point)

Fill out each row of the following table with the corresponding representation in other bases of the number given in that row; show the details of your calculations**.**

	decimal	binary	hexadecimal	octal
(a)	25.5	**1 1001.1**	**19.8**	**31.4**
(b)	**10.25**	1010.01	**A.4**	**12.2**
(c)	**26**	**1 1010**	1A	**32**
(d)	*26*	*1 1010*	*1A*	32

(a) $\mathbf{25.5_{10}}$ = 16 + 8 + 1 + ½ = $\mathbf{1\ 1001.1_2}$ = 1 1001.1000=$\mathbf{19.8_{16}}$ = 11 001.100=$\mathbf{31.4_8}$
(b) $\mathbf{1010.01_2}$ = 8+2+1/4 = $\mathbf{10.25_{10}}$ =1010.0100 = $\mathbf{A.4_{16}}$ = 1 010.010 = $\mathbf{12.2_8}$
(c) $\mathbf{1A_{16}}$ = $\mathbf{1\ 1010_2}$ = 16+10 = $\mathbf{26_{10}}$ **=** 11 010 = **32**
(d) *Since the last 2 numbers in the last column is the same* (32)*, it means that the last 2 rows represent the same value, such that their elements are identical.*

1.2 (4 points)

In the 8-bit signed 2's complement representation, the number of distinct numbers is:
(a) 256 from [-128,127]
(b) 128
(c) 255
(d) None of these

1.3 (6+6+8 = 20 points)

7-bit registers are used in this question to store numbers expressed in **2's complement representation**.

(a) Convert the following two signed numbers to binary observing the above assumption:

$A = (-31)_{10}$

	sgn	2^5	2^4	2^3	2^2	2^1	2^0
A's magnitude represented on 7 bits: 31 = 16 + 8 + 4 + 2 + 1 =	**0**	0	1	1	1	1	1

=> A = -31 = 2's complement (001 1111) = **110 0001**

B = $(+63)_{10}$ = **011 1111**

(b) Find the 2's complement of the signed binary numbers A and B, and give your results in decimal, too:

Finding the 2's complement of a signed binary number will negate the number:

2's complement of A = 2's complement (110 0001) = **0**01 1111

2's complement of A in decimal: **0**01 1111_2 = $+31_{10}$

2's complement of B = 2's complement (011 1111) = **1**00 0001, which is a negative number, say ***–x***.

To find the magnitude (***x***) *of this number* (***-x***) we *complement it*, *since* $x = -(-x)$ = 2's complement (***-x***): x = 2's complement ($-x$) = 2's complement (100 0001) = 011 1111 = +63

=> 2's complement of B in decimal = ***- x*** = - 63, which makes sense, since complementing B = 63, we negate its value to -63.

(c) Perform the following arithmetic operations in signed-2's complement representation, using a 7-bit ALU; show operations and results (including intermediary steps), both in **binary** and in **decimal**. Are there any overflows? How can a computer detect overflow?

$(+63)_{10} + (-31)_{10}$

	Carry:	1	1	1	1	1	1	1	
+63			**0**	1	1	1	1	1	1
- 31			**1**	1	0	0	0	0	1
+32			**0**	1	0	0	0	0	0

Converted to decimal: +32

$(-63)_{10} - (-31)_{10}$

	Carry:	0	0	1	1	1	1	1	
- 63			**1**	0	0	0	0	0	1
+31			**0**	0	1	1	1	1	1
-32			**1**	1	0	0	0	0	0

The result 110 0000 is a negative number, say – *x.*

To convert 110 0000 (i.e., $-x$) to decimal, first we find x, by complementing 110 0000, i.e.,

$x = -(-x)$ = 2's complement (110 0000) = 001 1111 +1 = 010 0000 = +32,

=> the result in decimal $-x = -32$… which was expected.

Are there any overflows?

There are no overflows since different-sign numbers are added.

How can a computer detect overflow?

1. Overflow = $Carry_7$ xor $Carry_6$

or

2. Overflow: Sign A & Sign B & /Sign result + /Sign A & /Sign B & Sign result

Question 2

2.1 (3x2 = 6 points)

You have a SRAM memory chip with a capacity of 8k x 4

a) How many input-output data lines does it have?

Answer a) ..4..........................

b) How many address lines does it have?

Answer b) ....13........................

c) What is its capacity expressed in "*bits*"?

Answer c) ...32 kbits $= 2^3\ 2^{10}\ 2^2 = 2^{15} = 2^5\ 2^{10}$

2.2 (2x6 = 12 points)

What is wrong with the following transfer statements (RTL)?

a) xT: AR ← AR', AR ← 0

b) yT: PC ← AR, PC ← PC + 1

One cannot load two different data in the same register!

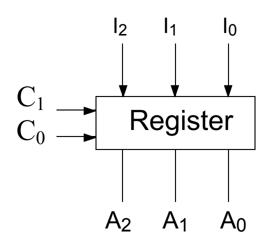
Question 3 (20 points)

Design a 3-bit multi-function register A ($A_2\ A_1\ A_0$) whose operation is described in the following table, where C_1 and C_0 are two control bits.

Use in your design JK flip flops, logic gates and any digital components (encoders, decoders, multiplexers, etc.); draw a detailed diagram of the logic circuit of the multi-function register.

Function table of 3-bit register A.

Clock	C_1	C_0	Operation	$A_2^+\ A_1^+\ A_0^+$
↑	0	0	No change	$A_2\ A_1\ A_0$
↑	0	1	Increment by 3	$A_2\ A_1\ A_0 + 3$
↑	1	0	Decrement by 3	$A_2\ A_1\ A_0 + 3$
↑	1	1	Loading external inputs, say $I_2\ I_1\ I_0$	$I_2\ I_1\ I_0$



Characteristic Table

J	K	$Q(n)$	$Q(n+1)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Excitation Table

$Q(n)$	$Q(n+1)$	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Remember the generation of the FF excitation equations:

- Copy to J the next state $Q(n+1)$ if $Q(n)=0$
- Copy to K the complement of the next state $Q(n+1)$ if $Q(n)=1$
- J and K should be "x" otherwise

J	K	$Q(n+1)$
0	0	$Q(n)$
0	1	0
1	0	1
1	1	$Q'(n)$

Answer:

$C_1\ C_0$ = 00 => no change

<=> $A_i\ (n+1) = A_i\ (n)$; $i = \{0,1,2\}$

=> $J_2 = 0$; $K_2 = 0$; $J_1 = 0$; $K_1 = 0$; $J_0 = 0$; $K_0 = 0$.

$C_1 C_0$ = 01 =>Increment by 3

Present State			Next State			FF Inputs					
$A_2(n)$	$A_1(n)$	$A_0(n)$	$A_2(n+1)$	$A_1(n+1)$	$A_0(n+1)$	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	1	1	0	x	1	x	1	x
0	0	1	1	0	0	1	x	0	x	x	1
0	1	0	1	0	1	1	x	x	1	1	x
0	1	1	1	1	0	1	x	x	0	x	1
1	0	0	1	1	1	x	0	1	x	1	x
1	0	1	0	0	0	x	1	0	x	x	1
1	1	0	0	0	1	x	1	x	1	1	x
1	1	1	0	1	0	x	1	x	0	x	1

J_2

A_2 \ $A_1 A_0$	00	01	11	10
0	0	1	1	1
1	x	x	x	x

K_2

A_2 \ $A_1 A_0$	00	01	11	10
0	x	x	x	x
1	0	1	1	1

$J_2 = K_2 = A_0 + A_1$

$J_1 = K_1 = A_0'$

$J_0 = K_0 = 1$

$C_1 C_0$ = 10 =>Decrement by 3

Present State			Next State			FF Inputs					
$A_2(n)$	$A_1(n)$	$A_0(n)$	$A_2(n+1)$	$A_1(n+1)$	$A_0(n+1)$	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	1	0	1	1	x	0	x	1	x
0	0	1	1	1	0	1	x	1	x	x	1
0	1	0	1	1	1	1	x	x	0	1	x
0	1	1	0	0	0	0	x	x	1	x	1
1	0	0	0	0	1	x	1	0	x	1	x
1	0	1	0	1	0	x	1	1	x	x	1
1	1	0	0	1	1	x	1	x	0	1	x
1	1	1	1	0	0	x	0	x	1	x	1

J_2

A_2 \ $A_1 A_0$	00	01	11	10
0	1	1	0	1
1	x	x	x	x

K_2

A_2 \ $A_1 A_0$	00	01	11	10
0	x	x	x	x
1	1	1	0	1

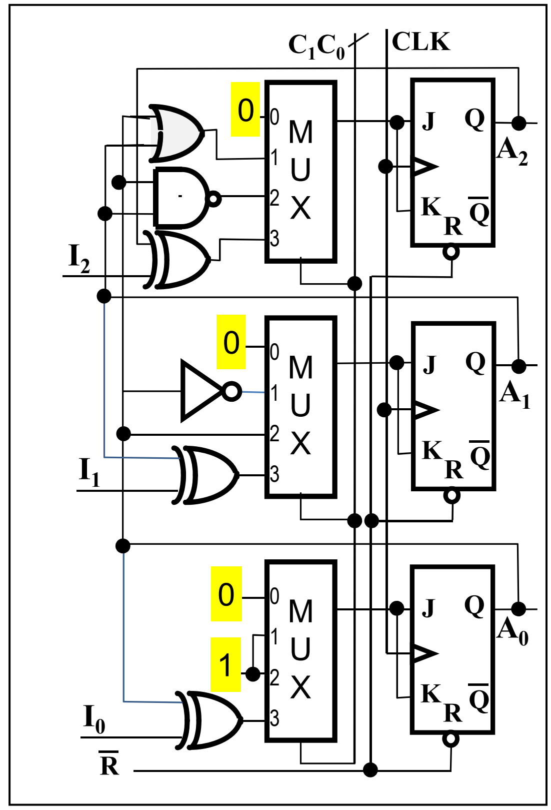
$J_2 = K_2 = A_0' + A_1' = (A_0 A_1)'$

$J_1 = K_1 = A_0$

$J_0 = K_0 = 1$

$C_1 C_0$ = 11 =>Load $I_2I_1I_0$ to $A_2A_1A_0$:

<=> $A_i (n+1) = I_i (n)$; $i= \{0,1,2\}$



$A_i(n)$	I_i	$A_i(n+1)$	J_i	K_i
0	0	0	0	x
0	1	1	1	x
1	0	0	x	1
1	1	1	x	0

$J_i = K_i = I_i$ xor A_i

or

$J_i = I_i$; $K_i = I_1$'

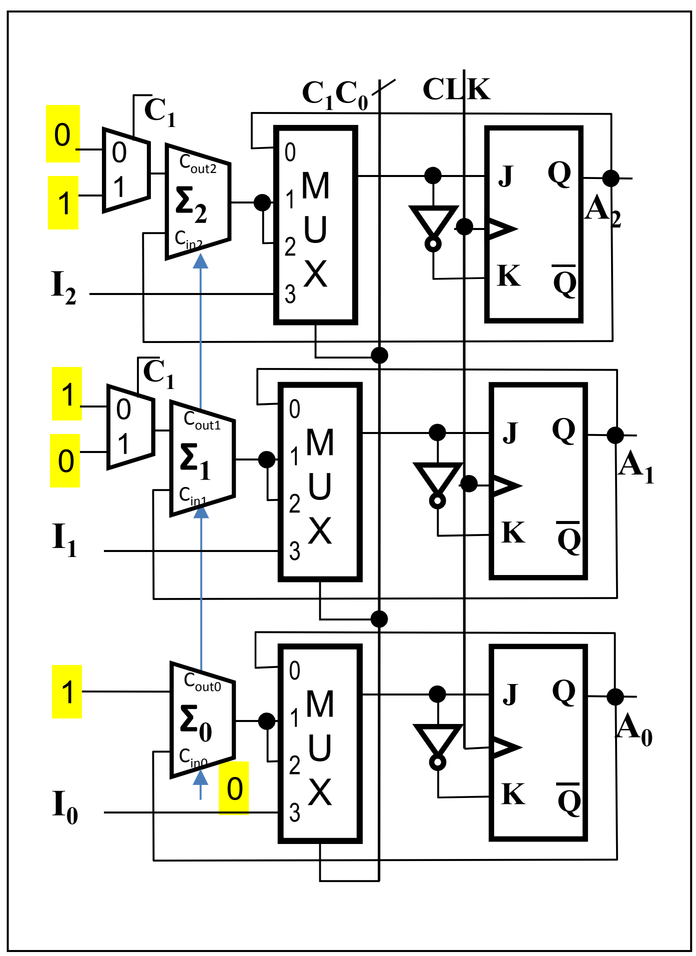
or …

Conclusion

(T FF approach, i.e., J = K = T):

C_1	C_0	$J_2 = K_2$	$J_1 = K_1$	$J_0 = K_0$
0	0	0	0	0
0	1	$A_0 + A_1$	A_0'	1
1	0	(A_0A_1)'	A_0	1
1	1	I_2 xor A_2	I_1 xor A_1	I_0 xor A_0

D FF + adders approach:



Comprehensive description, but longer approach:

Control	Present State			Next State			FF Inputs						*FF input equations*
C_1C_0	$A_2(n)$	$A_1(n)$	$A_0(n)$	$A_2(n+1)$	$A_1(n+1)$	$A_0(n+1)$	J_2	K_2	J_1	K_1	J_0	K_0	
00	0	0	0	0	0	0	0	x	0	x	0	x	
	0	0	1	0	0	1	0	x	0	x	x	0	$J_2 = K_2 = 0$
	0	1	0	0	1	0	0	x	x	0	0	x	
	0	1	1	0	1	1	0	x	x	0	x	0	$J_1 = K_1 = 0$
	1	0	0	1	0	0	x	0	0	x	0	x	
	1	0	1	1	0	1	x	0	0	x	x	0	$J_0 = K_0 = 0$
	1	1	0	1	1	0	x	0	x	0	0	x	
	1	1	1	1	1	1	x	0	x	0	x	0	
01	0	0	0	0	1	1	0	x	1	x	1	x	
	0	0	1	1	0	0	1	x	0	x	x	1	$J_2 = K_2 = A_0 + A_1$
	0	1	0	1	0	1	1	x	x	1	1	x	
	0	1	1	1	1	0	1	x	x	0	x	1	$J_1 = K_1 = A_0'$
	1	0	0	1	1	1	x	0	1	x	1	x	
	1	0	1	0	0	0	x	1	0	x	x	1	$J_0 = K_0 =1$
	1	1	0	0	0	1	x	1	x	1	1	x	
	1	1	1	0	1	0	x	1	x	0	x	1	
10	0	0	0	1	0	1	1	x	0	x	1	x	
	0	0	1	1	1	0	1	x	1	x	x	1	$J_2 = K_2 = A_0' + A_1' =$ $(A_0A_1)'$
	0	1	0	1	1	1	1	x	x	0	1	x	
	0	1	1	0	0	0	0	x	x	1	x	1	$J_1 = K_1 = A_0$
	1	0	0	0	0	1	x	1	0	x	1	x	
	1	0	1	0	1	0	x	1	1	x	x	1	$J_0 = K_0 =1$
	1	1	0	0	1	1	x	1	x	0	1	x	
	1	1	1	1	0	0	x	0	x	1	x	1	
11	0	0	0	I_2	I_1	I_0	I_2	x	I_1	x	I_0	x	
	0	0	1	I_2	I_1	I_0	I_2	x	I_1	x	x	I_0'	$J_2 = I_2$
	0	1	0	I_2	I_1	I_0	I_2	x	x	I_1'	I_0	x	$K_2 = I_2'$
	0	1	1	I_2	I_1	I_0	I_2	x	x	I_1'	x	I_0'	$J_1 = I_1$
	1	0	0	I_2	I_1	I_0	x	I_2'	I_1	x	I_0	x	$K_1 = I_1'$
	1	0	1	I_2	I_1	I_0	x	I_2'	I_1	x	x	I_0'	$J_0 = I_0$
	1	1	0	I_2	I_1	I_0	x	I_2'	x	I_1'	I_0	x	$K_0 = I_0'$
	1	1	1	I_2	I_1	I_0	x	I_2'	x	I_1'	x	I_0'	

Question 4 (20 points)

Design a 4-bit arithmetic circuit, with two selection variables *x* and *y*; the 1-bit variable *z* is an input of full adder. The circuit generates the following eight arithmetic operations under control of *x*, *y* and *z*: (Note: *X'* is the 1's complement of *X*)

x y	*z* = 0	*z* = 1
0 0	F = A+B (add)	F = A+B+1
0 1	F = A (transfer)	F = A+1 (increment)
1 0	F = B' (complement)	F = B'+1 (negate)
1 1	F = A+B'	F = A-B (subtract)

Draw the logic diagram of the two least significant bits of your arithmetic circuit, *only*.
Use full adders and other logic circuits (multiplexer, gates) as required.

Answer:

x y			$Cy_0 = z =$	0		$Cy_0 = z =$	1
0 0	$F_3\ F_2\ F_1\ F_0 =$	$A_3\ A_2\ A_1\ A_0\ +$	$B_3\ B_2\ B_1\ B_0\ +$	0	$A_3\ A_2\ A_1\ A_0\ +$	$B_3\ B_2\ B_1\ B_0\ +$	1
0 1	$F_3\ F_2\ F_1\ F_0 =$	$A_3\ A_2\ A_1\ A_0\ +$	0 0 0 0 +	0	$A_3\ A_2\ A_1\ A_0\ +$	0 0 0 0 +	1
1 0	$F_3\ F_2\ F_1\ F_0 =$	0 0 0 0 +	$B_3'B_2'B_1'B_0'+$	0	0 0 0 0 +	$B_3'B_2'B_1'B_0'+$	1
1 1	$F_3\ F_2\ F_1\ F_0 =$	$A_3\ A_2\ A_1\ A_0\ +$	$B_3'B_2'B_1'B_0'+$	0	$A_3\ A_2\ A_1\ A_0\ +$	$B_3'B_2'B_1'B_0'+$	1

$F_1 = x'y'(A_1 + B_1 + Cy_1) + x'y\,(A_1 + 0 + Cy_1) + xy'(0 + B_1' + Cy_1) + xy\,(A_1 + B_1' + Cy_1)$ or

$F_1 = (x'y'A_1 + x'y\,A_1 + xy\,A_1) + (x'y'B_1 + x'y\,0 + xy'B_1' + xy\,B_1') + Cy_1 = (xy')'A_1 + (x'y'B_1 + x'y\,0 + xy'B_1' + xy\,B_1') + Cy_1$

$F_0 = x'y'(A_0 + B_0 + Cy_0) + x'y\,(A_0 + 0 + Cy_0) + xy'(0 + B_0' + Cy_0) + xy\,(A_0 + B_0' + Cy_0)$; $\mathbf{Cy_0} = z$ **or:**

$F_0 = (x'y'A_0 + x'y\,A_0 + xy\,A_0) + (x'y'B_0 + x'y\,0 + xy'B_0' + xy\,B_0') + Cy_0 = (xy')'A_0 + (x'y'B_0 + x'y\,0 + xy'B_0' + xy\,B_0') + Cy_0$

The two possible implementations are presented below: