UBC

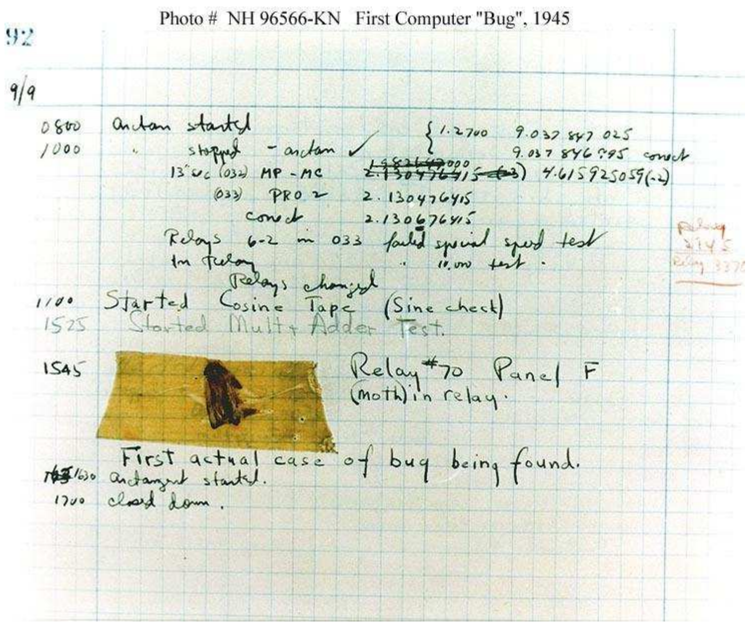
University of British Columbia
Electrical and Computer Engineering
Introduction to Microcomputers EECE259

Lecture 14: Microcomputer Integer Arithmetic I

Dr. Jesús Calviño-Fraga P.Eng.
Department of Electrical and Computer Engineering, UBC
Office: KAIS 3024
E-mail: jesusc@ece.ubc.ca
Phone: (604)-827-5387

June 5, 2014

Grace Murray Hopper's log book.



Electromechanical Relay

Objectives

- Perform single-byte-unsigned arithmetic add and subtract operations.
- Perform multi-byte unsigned arithmetic add and subtract operations.
- Compare single-byte and multi-byte numbers (=, <, >)
- Perform single-byte and multi-byte signed arithmetic add and subtract operations.
- Perform Binary to BCD and BCD to Binary conversion.

Unsigned-byte (8 bit) addition

Adding decimal numbers, by hand:

LSD

	4	4
+	5	5
	9	9

MOV A, #44

ADD A, #55

Adding binary numbers, by hand:

LSB

	0	0	1	0	1	1	0	0	(44)
+	0	0	1	1	0	1	1	1	(55)
	0	1	1	0	0	0	1	1	(99)

Unsigned-byte (8 bit) addition

Add the content of R0 and R1 and store it in R7:

```
MOV A, R0
ADD A, R1
MOV R7, A
```

Add the content of memory locations 50 and 51, and save the result in location 52:

```
MOV A, 50
ADD A, 51
MOV 52, A
```

Unsigned-byte (8 bit) subtraction

Subtract decimal numbers, by hand:

LSD

	5	5
–	4	4
	1	1

CLR C

MOV A, #55

SUBB A, #44

Subtract binary numbers, by hand:

LSB

	0	0	1	0	1	1	0	0	(55)
–	0	0	1	1	0	1	1	1	(44)
	0	0	0	0	1	0	1	1	(11)

Unsigned-byte (8 bit) subtraction

Subtract the content of R0 and R1 and store it in R7:

```
MOV A, R0
CLR C
SUBB A, R1
MOV R7, A
```

Subtract the content of memory locations 60 and 61, and save the result in register DPL:

```
MOV A, 60
CLR C
SUBB A, 61
MOV DPL, A
```

Unsigned multi-byte addition / subtraction and the Carry / Borrow flag

- We can handle integers of more than 8 bits by cascading addition/subtraction operation and use the carry/borrow flag.
- The MCS-51 have instructions to add both with carry and without carry, but it can only subtract with borrow!
  - ADD
  - ADDC
  - SUBB

16-bit addition

LSD

```
   2 0 4 7
+    3 2 2
  --------
   2 3 6 9
```

Carry

	High byte	Low byte
	0 0 0 0 0 1 1 1	1 1 1 1 1 1 1 1
+	0 0 0 0 0 0 0 1	0 1 0 0 0 0 1 0
	0 0 0 0 1 0 0 1	0 1 0 0 0 0 0 1

16-bit addition

In MCS-51 Assembly language:

```
mov a, #low(2047)
add a, #low(322)
mov b, a ; save low result in reg. b
mov a, #high(2047)
addc a, #high(322)
```

'high' and 'low' are assembler directives that take the upper or lower 8 bits of a 16-bit number.

16-bit addition

In MCS-51 Assembly language:

```
mov a, #0FFH
add a, #42H
mov b, a ; save low result in reg. b
mov a, #03H
addc a, #01H
```

Same as the previous slide, but you need to convert the numbers 2047 and 322 to hex (or binary!) and write them down in the program. Windows 'calc' is very handy for this!

32-bit addition

- Add the two 32 bit numbers located at RAM memory addresses 30H and 34H and store the result at XRAM address 100H.

NOTE: For regular internal memory we use the **MOV** instruction. For expanded memory we use the **MOVX** instruction.

First try: brute force!

```
mov a, 30H
add a, 34H
mov dptr, #100H
movx @dptr, a
mov a, 31H
addc a, 35H
inc dptr
movx @dptr, a
```

```
mov a, 32H
addc a, 36H
inc dptr
movx @dptr, a
mov a, 33H
addc a, 37H
inc dptr
movx @dptr, a
```

32-bit addition

- Add the two 32 bit numbers located at RAM memory addresses 30H and 34H and store the result at XRAM address 100H.
- Now, let us use assembler directives and indirect addressing (@R0, @R1, and @DPTR) in a compilable program:

32-bit addition

```
; Add32.asm: shows how to add two 32-bit numbers at RAM
; addresses 30H and 34H and place result at XRAM address
; 100H

$MODDE2

org 0000H
       ljmp myprogram

DSEG at 30H

Num1:  DS     4
Num2:  DS     4

XSEG at 100H
Result:       DS     4
```

32-bit addition

```
CSEG
myprogram:
        mov SP, #07FH
; Make Num1=55555555H for testing
        mov R0, #Num1
L1:     mov @R0, #55H
        inc R0
        cjne R0, #Num1+4, L1
; Make Num1=66666666H for testing
        mov R1, #Num2
L2:     mov @R1, #66H
        inc R1
        cjne R1, #Num2+4, L2
; Initialize pointers
        mov R0, #Num1
        mov R1, #Num2
        mov DPTR, #Result
```


32-bit addition

```
        clr c
        mov R2, #4
; Add the bytes, one by one
L3:     mov A, @R0
        addc A, @R1
        movx @dptr, A
        inc R0
        inc R1
        inc dptr
        djnz R2, L3

; Done!  Loop forever
forever:
        jmp forever
END
```

16-bit subtraction

LSD

$$
\begin{array}{rccccc}
 & 1 & 4 & 0 & 9 & 6 \\
- & & 4 & 1 & 2 & 8 \\
\hline
 & & 9 & 9 & 6 & 8
\end{array}
$$

Borrow

High byte | Low byte

$$
\begin{array}{rcccccccc|cccccccc}
 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
- & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\hline
 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0
\end{array}
$$

16-bit subtraction

In MCS-51 Assembly language:

```
clr c ; the 'carry' is also the 'borrow'!
mov a, #low(14096)
subb a, #low(4128)
mov b, a ; save low result in register b
mov a, #high(3710H)
subb a, #high(1020H)
```

14096=3710H=0011011100010000B

4128=1020H=0001000000100000B

32-subtraction…

```
clr c
mov a, 30H
subb a, 34H
mov dptr, #100H
movx @dptr, a
mov a, 31H
subb a, 35H
inc dptr
movx @dptr, a
```

```
mov a, 32H
subb a, 36H
inc dptr
movx @dptr, a
mov a, 33H
subb a, 37H
inc dptr
movx @dptr, a
```


32-bit subtraction…

Simply modify the example given above for 32-bit addition:

```
      clr c
      mov R2, #4
; Subtract the bytes, one by one
L3:   mov A, @R0
      subb A, @R1
      movx @dptr, A
      inc R0
      inc R1
      inc dptr
      djnz R2, L3

; Done!  Loop forever
forever:
      jmp forever
END
```

Comparing numbers

- We use single or multi-byte subtraction to compare two numbers:
  - If the result of the subtraction is zero: minuend = subtrahend.
  - If the result of the subtraction is not zero: minuend ≠ subtrahend.
  - If 'borrow' set: minuend < subtrahend.
  - If no 'borrow' set: minuend > subtrahend.
- To check, we use the 'zero' and 'carry' flags and the instructions: JZ, JNZ, JC, and JNC.

Are two 16-bit numbers Equal?

A 16-bit number is stored in R0 and R1, where R0 is the LSB. Write the assembly code to check if the stored number is 1000.

```
clr c
mov a, #low(1000)
subb a, R0
jnz NotEqual
mov a, #high(1000)
subb a, R1
jnz NotEqual
sjmp Equal
```

'NotEqual' and 'Equal' are labels somewhere else in the program.

Number1 > Number2?

Number1 is stored in R0 and R1, where R0 is the LSB. Number2 is stored at addresses 48H and 49H, where 48H is the LSB.

```
clr c
mov a, 48H
subb a, R0
mov a, 49H
subb a, R1
jc MyLabel ; jump if number1 > number2
```


Number1 < Number2?

Number1 is stored in R0 and R1, where R0 is the LSB. Number2 is stored at addresses 48H and 49H, where 48H is the LSB.

```
clr c
mov a, 48H
subb a, R0
mov a, 49H
subb a, R1
jnc MyLabel ; jump if number1 < number2
```

WRONG!

Is the carry set if number1 = number2?

On the other hand, this is a perfectly good (Number1 <= Number2)

Number1 < Number2?

Number1 is stored in R0 and R1, where R0 is the LSB. Number2 is stored at addresses 48H and 49H, where 48H is the LSB.

```
clr c
mov a, R0
subb a, 48H
mov a, R1
subb a, 49H
jc MyLabel ; jump if number1 < number2
```


Signed integer representation

True magnitude number representation:

$\boxed{0}$ 0 1 1 0 1 0 0 = +52

Sign bit

$\boxed{1}$ 0 1 1 0 1 0 0 = -52

Sign bit

Easy to understand, HARD to work with!

Signed integer representation

2's-complement number representation:

$\boxed{0}$ 0 1 1 0 1 0 0 = +52

Sign bit

$\boxed{1}$ 1 0 0 1 1 0 0 = -52

Sign bit

Very convenient to perform arithmetic!

2's Complement

To obtain the 2's complement of a number:

0 0 1 1 0 1 0 0	Original number: +52
1 1 0 0 1 0 1 1	1's complement
1	Add 1
1 1 0 0 1 1 0 0	2's complement

In 8051's assembly language:

```
MOV A, #52
CPL A
INC A
```

Or

```
CLR C
CLR A
SUBB A, #52
```

Signed Integer Addition

Case 1: two positive numbers:

	0	0	1	1	0	1	1	1	55
+	0	0	0	1	0	1	0	0	20
	0	1	0	0	1	0	1	1	75

Case 2: positive number larger than negative number:

	0	0	1	1	0	1	1	1	55
+	1	1	0	0	1	1	0	0	-52
	0	0	0	0	0	0	1	1	3

Signed Integer Addition

Case 3: negative number larger than positive number :

	0	0	0	1	0	1	0	0	20
+	1	1	0	0	1	1	0	0	-52
	1	1	1	0	0	0	0	0	-32

Case 4: Two negative numbers:

	1	1	1	1	0	0	0	1	-15
+	1	1	0	0	1	1	0	0	-52
	1	0	1	1	1	1	0	1	-67

Signed Integer Addition

Case 5: equal and opposite numbers :

	0	0	1	1	0	1	0	0	52
+	1	1	0	0	1	1	0	0	-52
	0	0	0	0	0	0	0	0	0

This technique also works for decimal number calculations. It is particularly handy when performing subtraction by hand. For more information, check the "method of complements" in Wikipedia:

http://en.wikipedia.org/wiki/Method_of_complements

Method of Complements for Hand Calculation of Decimal Subtraction

	9	8	5	3	1	8	0	3	1
–		4	5	1	7	0	2	4	6

'Negate' in base 10 and add 1 to get the ten's complement

	9	5	4	8	2	9	7	5	3
+									1
	9	5	4	8	2	9	7	5	4

Add the Ten's complement

		9	8	5	3	1	8	0	3	1
+		9	5	4	8	2	9	7	5	4
	1	9	4	0	1	4	7	7	8	5

Correct answer!

Disregard!

Signed Integer Addition

Write the assembly code to add two 16-bit signed numbers at RAM addresses 20H-21H and 30H-31H and save the result in R0-R1.

```
mov A, 21H
add A, 31H
mov R1, A
mov A, 20H
addc A, 30H
mov R0, A
```


Signed Integer Subtraction

Exactly the same as signed addition. Just get the 2's complement of the subtrahend before the addition!

Write the assembly code to subtract the signed 16-bit number at addresses 30H-31H from the 16-bit signed number at RAM addresses 20H-21H. Save the result in R0-R1.

```
mov A, 31H
cpl A
add A, #1
mov R1, A
mov A, 30H
cpl A
addc A, #0
mov R0, A
```

```
mov A, 21H
add A, R1
mov R1, A
mov A, 20H
addc A, R0
mov R0, A
```

Binary to BCD Conversion

- Registers, counters, addresses, numbers, etc. in the microcontroller are represented in binary.
- People like to work with decimal numbers! Also, it is very easy to convert BCD to ASCII:

Code used to represent English characters and symbols in displays, terminals, and printers

```
mov R1,#91H ; BCD number
mov a, R1
swap a
anl a,#0FH
orl a,#30H ; ASCII “9” is #39H
mov 50H, a
mov a, R1
anl a,#0FH
orl a,#30H ; ASCII “1” is #31H
mov 51H, a
```


ASCII Table

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Binary to BCD conversion of 8-bit numbers in the 8051

```
; Eight bit number to convert passed in acc.
; Result in r1,r0
Bin2BCD_8bit:
        mov b, #100
        div ab
        mov r1, a    ; Save hundreds
        mov a, b     ; Remainder is in register b
        mov b, #10
        div ab
        swap a
        orl a, b
        mov r0, a    ; Save tens and ones
        ret
```


Binary to BCD Conversion: the double dabble algorithm

1. BIN: 01011011 BCD=000
   Multiply BDC by two and add the underlined bit:
   BCD=(000+000+0)=000
2. BIN: 01011011 BCD=000
   Multiply BDC by two and add the underlined bit:
   BCD=(000+000+1)=001
3. BIN: 01011011 BCD=001
   Multiply BDC by two and add the underlined bit:
   BCD=(001+001+0)=002
4. BIN: 01011011 BCD=002
   Multiply BDC by two and add the underlined bit:
   BCD=(002+002+1)=005

Binary to BCD Conversion : the double dabble algorithm

5. BIN: 01011011   BCD=005
   Multiply BDC by two and add the underlined bit:
   BCD=(005+005+1)=011
6. BIN: 01011011   BCD=011
   Multiply BDC by two and add the underlined bit:
   BCD=(011+011+0)=022
7. BIN: 01011011   BCD=022
   Multiply BDC by two and add the underlined bit:
   BCD=(022+022+1)=045
8. BIN: 01011011   BCD=045
   Multiply BDC by two and add the underlined bit:
   BCD=(045+045+1)=091

Final result is 91

Multiply a BCD number by 2.

- If the processor supports addition of BCD numbers then add the BCD number to itself.
- If the processor DOES NOT support addition of BCD numbers:
  - Add 3 (0011B) to each BCD digit > 4 (0100B).
  - Shift Left one bit.

BCD addition in the 8051: the "DA A" instruction.

- **Function:** Decimal Adjust Accumulator
- **Description:** DA adjusts the contents of the Accumulator to correspond to a BCD (Binary Coded Decimal) number after two BCD numbers have been added by the ADD or ADDC instruction. If the carry bit is set or if the value of bits 0-3 exceed 9, 06H is added to the accumulator. If the carry bit was set when the instruction began, or if 06H was added to the accumulator in the first step, 60H is added to the accumulator.

Page 61 of course textbook has more details on the works of this instruction!

How to multiply a BCD by two if BCD addition is not available.

Multiply BCD number 3925 by 2:

```
0011 1001 0010 0101   BCD number to multiply by 2
0000 0011 0000 0011   Correction before left shift
0011 1100 0010 1000   Sum
0111 1000 0101 0000   Shift left
  7    8    5    0
```

3925 * 2 = 7850

16-bit binary to BCD

```
;Converts the hex number in R0-R1 to BCD
;in R2-R3-R4
hex2bcd:
    mov R2, #0  ;Set BCD result to 00000
    mov R3, #0
    mov R4, #0
    mov R5, #16 ;Loop counter.
L0:
    mov a, R1 ;Shift R0-R1 left through carry
    rlc a
    mov R1, a
    mov a, R0
    rlc a
    mov R0, a
```

Continues…

16-bit binary to BCD

```
; Perform bcd + bcd + carry
; using BCD numbers
mov a, R4
addc a, R4
da a
mov R4, a
mov a, R3
addc a, R3
da a
mov R3, a
mov a, R2
addc a, R2
mov R2, a
djnz R5, L0
ret
```

BCD to Binary Conversion

- We may need to convert to binary any number provided by humans!
- Many chips work only with BCD numbers, for example Real Time Clocks (RTCs) often count in BCD only.
- One way to convert to BCD to binary is by multiplying by ten (10) and adding a four bit number:

BCD to Binary Conversion for 8-bit binary numbers in the 8051.

1. BCD=<u>1</u>37 BIN: 00000000
   Multiply BIN by ten and add the underlined digit:
   BIN=((00000000*1010)+0001)=00000001
2. BCD=1<u>3</u>7 BIN: 00000001
   Multiply BIN by ten and add the underlined digit:
   BIN=((00000001*1010)+0011)=00001101
3. BCD=13<u>7</u> BIN: 00001101
   Multiply BIN by ten and add the underlined digit:
   BIN =((00001101*1010)+0111)=**10001001**

BCD to Binary Conversion for 8-bit binary numbers in the 8051.

- Convert a three digit BCD number (0 to 255) stored in R6-R7 to binary and save the result in R5.

```
mov a, R6
mov b, #10
mul ab
mov R5, a
mov a, R7
swap a
anl a,#0fH
add a, R5
mov b, #10
mul ab
mov R5, a
```

```
mov a, R7
anl a,#0fH
add a, R5
mov R5, a
```


BCD to Binary Conversion for n-bit binary numbers.

Convert 205 to binary:

```
205 ÷ 2 = 102   R = 1
102 ÷ 2 =  51   R = 0
 51 ÷ 2 =  25   R = 1
 25 ÷ 2 =  12   R = 1
 12 ÷ 2 =   6   R = 0
  6 ÷ 2 =   3   R = 0
  3 ÷ 2 =   1   R = 1
  1 ÷ 2 =   0   R = 1
```

To convert BCD to binary we need to divide a BCD number by two! This algorithm works in reverse from the double dabble algorithm!!!

205 = 11001101B = 0CDH

Divide BCD by two

- Shift the BCD number right by one
- If the most significant bit of a BCD digit is one, subtract 3 to that digit.
- The least significant bit of the least significant BCD digit is the remainder.

Divide BCD by two

Divide BCD number 3925 by 2:

```
0011 1001 0010 0101   BCD number to divide by 2
0001 1100 1001 0010   Right shift
0000 0011 0011 0000   Correction
0001 1001 0110 0010   Subtract
  1    9    6    2
```

3925 / 2 = 1962, remainder=1

16-bit BCD to binary

```
; bcd2hex: Converts the BCD number in R2-R3-R4
; to binary in R0-R1

rrc_and_correct:
    rrc a
    mov r6, psw ; Save carry (it is changed by the add)
    jnb acc.7, nocor1
    add a, #(100H-30H) ; subtract 3 from packed BCD MSD
nocor1:
    jnb acc.3, nocor2
    add a, #(100H-03H) ; subtract 3 from packed BCD LSD
nocor2:
    mov psw, r6 ; Restore carry
    ret
```


16-bit BCD to binary

```
bcd2hex:
    mov R5, #16 ;Loop counter.
L0:
    ; Divide BCD by two
    clr c
    mov a, R2
    lcall rrc_and_correct
    mov R2, a
    mov a, R3
    lcall rrc_and_correct
    mov R3, a
    mov a, R4
    lcall rrc_and_correct
    mov R4, a
```

16-bit BCD to binary

```
;Shift R0-R1 right through carry
mov a, R0
rrc a
mov R0, a
mov a, R1
rrc a
mov R1, a

djnz R5, L0
ret
```


Exercises

- Write an assembly subroutine for the 8051 that checks if a 32-bit number stored in registers R0 to R3 is zero.
- Write the assembly equivalent of this piece of C code (the size of int is 2 bytes):

```
unsigned int x, y;
unsigned char z;
.
[other code comes here]
.
if (x>y) z=0;
else z=1;
```

Assembly for this!