```
$NOLIST
CSEG

;---------------------------------------------------
; Converts the 32-bit hex number in 'x' to a
; 10-digit packed BCD in 'bcd' using the
; double-dabble algorithm.
;---------------------------------------------------
hex2bcd:
    push acc
    push psw
    push AR0
    push AR1
    push AR2

    clr a
    mov bcd+0, a ; Initialize BCD to 00-00-00-00-00
    mov bcd+1, a
    mov bcd+2, a
    mov bcd+3, a
    mov bcd+4, a
    mov r2, #32  ; Loop counter.

hex2bcd_L0:
    ; Shift binary left
    mov a, x+3
    mov c, acc.7 ; This way x remains unchanged!
    mov r1, #4
    mov r0, #(x+0)
hex2bcd_L1:
    mov a, @r0
    rlc a
    mov @r0, a
    inc r0
    djnz r1, hex2bcd_L1

    ; Perform bcd + bcd + carry using BCD arithmetic
    mov r1, #5
    mov r0, #(bcd+0)
hex2bcd_L2:
    mov a, @r0
    addc a, @r0
    da a
    mov @r0, a
    inc r0
    djnz r1, hex2bcd_L2

    djnz r2, hex2bcd_L0

    pop AR2
    pop AR1
    pop AR0
    pop psw
    pop acc
    ret


;---------------------------------------------------
; hex2bcd2:
; Converts the 32-bit hex number in 'x' to a
; 10-digit packed BCD in 'bcd' using the
; double-dabble algorithm.  This is what you would
; have to do in a proccessor without a bcd addition
; instruction.  The 8051 can add bcd number, so
; this function is here for your reference only.  Compare
; to the function above which uses the DA A instruction
; resulting in faster and smaller code.
;---------------------------------------------------
hex2bcd2:
    push acc
    push psw
    push AR0
    push AR1
    push AR2

    clr a
    mov bcd+0, a ; Initialize BCD to 00-00-00-00-00
    mov bcd+1, a
    mov bcd+2, a
    mov bcd+3, a
    mov bcd+4, a
    mov r2, #32  ; We need process 32 bits

hex2bcd2_L0:
    ; Shift binary left
    mov a, x+3
    mov c, acc.7 ; This way x remains unchanged!
    mov r1, #4
    mov r0, #(x+0)
hex2bcd2_L1:
    mov a, @r0
    rlc a
    mov @r0, a
    inc r0
    djnz r1, hex2bcd2_L1

    ; Shif bcd left
```

```
        mov r1, #5              ; BCD byte count = 5
        mov r0, #(bcd+0)        ; r0 points to least
significant bcd digits
hex2bcd2_L2:
        push psw                ; Save carry
        mov a, @r0
        add a, #33h             ; Pre-correction before
shifting left
        jb acc.7, hex2bcd2_L3 ; If the bcd digit was > 4
keep the correction
        add a, #(100h-30h)    ; Remove the correction to
the MSD by subtracting 30h
hex2bcd2_L3:
        jb acc.3, hex2bcd2_L4 ; If the bcd digit was > 4
keep the correction
        add a, #(100h-03h)    ; Remove the correction to
the LSD by subtracting 03h
hex2bcd2_L4:
        pop psw                 ; Restore carry
        rlc a
        mov @r0, a
        inc r0
        djnz r1, hex2bcd2_L2

        djnz r2, hex2bcd2_L0

        pop AR2
        pop AR1
        pop AR0
        pop psw
        pop acc

        ret

;------------------------------------------------
; bcd2hex:
; Converts the 10-digit packed BCD in 'bcd' to a
; 32-bit hex number in 'x'
;------------------------------------------------
bcd2hex:
        push acc
        push psw
        push AR0
        push AR1
        push AR2

        mov r2, #32  ; We need 32 bits

bcd2hex_L0:
        mov r1, #5              ; BCD byte count = 5
        clr c                   ; clear carry flag
        mov r0, #(bcd+4)        ; r0 points to most
significant bcd digits
bcd2hex_L1:
        mov a, @r0              ; transfer bcd to
accumulator
        rrc a                   ; rotate right
        push psw                ; save carry flag
        ; BCD divide by two correction
        jnb acc.7, bcd2hex_L2  ; test bit 7
        add a, #(100h-30h)     ; bit 7 is set. Perform
correction by subtracting 30h.
bcd2hex_L2:
        jnb acc.3, bcd2hex_L3  ; test bit 3
        add a, #(100h-03h)     ; bit 3 is set. Perform
correction by subtracting 03h.
bcd2hex_L3:
        mov @r0, a              ; store the result
        dec r0                  ; point to next pair of bcd digits
        pop psw                 ; restore carry flag
        djnz r1, bcd2hex_L1    ; repeat for all bcd pairs

        ; rotate binary result right
        mov r1, #4
        mov r0, #(x+3)
bcd2hex_L4:
        mov a, @r0
        rrc a
        mov @r0, a
        dec r0
        djnz r1, bcd2hex_L4

        djnz r2, bcd2hex_L0

        pop AR2
        pop AR1
        pop AR0
        pop psw
        pop acc

   ret

;------------------------------------------------
; x = x + y
;------------------------------------------------
add32:
        push acc
        push psw
```

```
	mov a, x+0
	add a, y+0
	mov x+0, a
	mov a, x+1
	addc a, y+1
	mov x+1, a
	mov a, x+2
	addc a, y+2
	mov x+2, a
	mov a, x+3
	addc a, y+3
	mov x+3, a
	pop psw
	pop acc
	ret

;----------------------------------------------
; x = x - y
;----------------------------------------------
sub32:
	push acc
	push psw
	clr c
	mov a, x+0
	subb a, y+0
	mov x+0, a
	mov a, x+1
	subb a, y+1
	mov x+1, a
	mov a, x+2
	subb a, y+2
	mov x+2, a
	mov a, x+3
	subb a, y+3
	mov x+3, a
	pop psw
	pop acc
	ret

;----------------------------------------------
; mf=1 if x < y
;----------------------------------------------
x_lt_y:
	push acc
	push psw
	clr c
	mov a, x+0
	subb a, y+0
	mov a, x+1
	subb a, y+1
	mov a, x+2
	subb a, y+2
	mov a, x+3
	subb a, y+3
	mov mf, c
	pop psw
	pop acc
	ret

;----------------------------------------------
; mf=1 if x > y
;----------------------------------------------
x_gt_y:
	push acc
	push psw
	clr c
	mov a, y+0
	subb a, x+0
	mov a, y+1
	subb a, x+1
	mov a, y+2
	subb a, x+2
	mov a, y+3
	subb a, x+3
	mov mf, c
	pop psw
	pop acc
	ret

;----------------------------------------------
; mf=1 if x = y
;----------------------------------------------
x_eq_y:
	push acc
	push psw
	clr mf
	clr c
	mov a, y+0
	subb a, x+0
	jnz x_eq_y_done
	mov a, y+1
	subb a, x+1
	jnz x_eq_y_done
	mov a, y+2
	subb a, x+2
	jnz x_eq_y_done
	mov a, y+3
	subb a, x+3
```

```
        jnz x_eq_y_done
        setb mf
x_eq_y_done:
        pop psw
        pop acc
        ret

;-----------------------------------------------
; mf=1 if x >= y
;-----------------------------------------------
x_gteq_y:
        lcall x_eq_y
        jb mf, x_gteq_y_done
        ljmp x_gt_y
x_gteq_y_done:
        ret

;-----------------------------------------------
; mf=1 if x <= y
;-----------------------------------------------
x_lteq_y:
        lcall x_eq_y
        jb mf, x_lteq_y_done
        ljmp x_lt_y
x_lteq_y_done:
        ret

;-----------------------------------------------
; x = x * y
;-----------------------------------------------
mul32:

        push acc
        push b
        push psw
        push AR0
        push AR1
        push AR2
        push AR3

        ; R0 = x+0 * y+0
        ; R1 = x+1 * y+0 + x+0 * y+1
        ; R2 = x+2 * y+0 + x+1 * y+1 + x+0 * y+2
        ; R3 = x+3 * y+0 + x+2 * y+1 + x+1 * y+2 + x+0
* y+3

        ; Byte 0
        mov     a,x+0
        mov     b,y+0
```

```
mul     ab              ; x+0 * y+0
mov     R0,a
mov     R1,b

; Byte 1
mov     a,x+1
mov     b,y+0
mul     ab              ; x+1 * y+0
add     a,R1
mov     R1,a
clr     a
addc a,b
mov     R2,a

mov     a,x+0
mov     b,y+1
mul     ab              ; x+0 * y+1
add     a,R1
mov     R1,a
mov     a,b
addc a,R2
mov     R2,a
clr     a
rlc     a
mov     R3,a

; Byte 2
mov     a,x+2
mov     b,y+0
mul     ab              ; x+2 * y+0
add     a,R2
mov     R2,a
mov     a,b
addc a,R3
mov     R3,a

mov     a,x+1
mov     b,y+1
mul     ab              ; x+1 * y+1
add     a,R2
mov     R2,a
mov     a,b
addc a,R3
mov     R3,a

mov     a,x+0
mov     b,y+2
mul     ab              ; x+0 * y+2
add     a,R2
```

```
        mov     R2,a
        mov     a,b
        addc a,R3
        mov     R3,a

        ; Byte 3
        mov     a,x+3
        mov     b,y+0
        mul     ab              ; x+3 * y+0
        add     a,R3
        mov     R3,a

        mov     a,x+2
        mov     b,y+1
        mul     ab              ; x+2 * y+1
        add     a,R3
        mov     R3,a

        mov     a,x+1
        mov     b,y+2
        mul     ab              ; x+1 * y+2
        add     a,R3
        mov     R3,a

        mov     a,x+0
        mov     b,y+3
        mul     ab              ; x+0 * y+3
        add     a,R3
        mov     R3,a

        mov     x+3,R3
        mov     x+2,R2
        mov     x+1,R1
        mov     x+0,R0

        pop AR3
        pop AR2
        pop AR1
        pop AR0
        pop psw
        pop b
        pop acc

        ret

;-----------------------------------------------
; x = x / y
; This subroutine uses the 'paper-and-pencil'
; method described in page 139 of 'Using the
; MCS-51 microcontroller' by Han-Way Huang.
;-----------------------------------------------
div32:
        push acc
        push psw
        push AR0
        push AR1
        push AR2
        push AR3
        push AR4

        mov     R4,#32
        clr     a
        mov     R0,a
        mov     R1,a
        mov     R2,a
        mov     R3,a

div32_loop:
        ; Shift the 64-bit of [[R3..R0], x] left:
        clr c
        ; First shift x:
        mov     a,x+0
        rlc a
        mov     x+0,a
        mov     a,x+1
        rlc     a
        mov     x+1,a
        mov     a,x+2
        rlc     a
        mov     x+2,a
        mov     a,x+3
        rlc     a
        mov     x+3,a
        ; Then shift [R3..R0]:
        mov     a,R0
        rlc     a
        mov     R0,a
        mov     a,R1
        rlc     a
        mov     R1,a
        mov     a,R2
        rlc     a
        mov     R2,a
        mov     a,R3
        rlc     a
        mov     R3,a

        ; [R3..R0] - y
```

```
	clr c
	mov	a,R0
	subb a,y+0
	mov	a,R1
	subb a,y+1
	mov	a,R2
	subb a,y+2
	mov	a,R3
	subb a,y+3

	jc	div32_minus		; temp >= y?

	; -> yes;  [R3..R0] -= y;
	; clr c ; carry is always zero here because of the jc above!
	mov	a,R0
	subb a,y+0
	mov	R0,a
	mov	a,R1
	subb a,y+1
	mov	R1,a
	mov	a,R2
	subb a,y+2
	mov	R2,a
	mov	a,R3
	subb a,y+3
	mov	R3,a

	; Set the least significant bit of x to 1
	orl	x+0,#1

div32_minus:
	djnz R4, div32_loop	; -> no

div32_exit:

	pop AR4
	pop AR3
	pop AR2
	pop AR1
	pop AR0
	pop psw
	pop acc

	ret

; Copy x to y
copy_xy:
	mov y+0, x+0
	mov y+1, x+1
	mov y+2, x+2
	mov y+3, x+3
	ret

; Exchange x and y
xchg_xy:
	mov a, x+0
	xch a, y+0
	mov x+0, a
	mov a, x+1
	xch a, y+1
	mov x+1, a
	mov a, x+2
	xch a, y+2
	mov x+2, a
	mov a, x+3
	xch a, y+3
	mov x+3, a
	ret

Load_X MAC
	mov x+0, #low (%0 % 0x10000)
	mov x+1, #high(%0 % 0x10000)
	mov x+2, #low (%0 / 0x10000)
	mov x+3, #high(%0 / 0x10000)
ENDMAC

Load_y MAC
	mov y+0, #low (%0 % 0x10000)
	mov y+1, #high(%0 % 0x10000)
	mov y+2, #low (%0 / 0x10000)
	mov y+3, #high(%0 / 0x10000)
ENDMAC

$LIST
```