БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра метрологии и стандартизации

ОТЧЁТ

По практической работе

«ИЗМЕРИТЕЛЬ СКВАЖНОСТИ НА МИКРОКОНТРОЛЛЕРЕ»

Выполнили: Проверил:

студент гр. 862101 Гурский А.Л.

Чайко А.М.

Минск 2011

Содержание

[Введение 3](#_Toc294594624)

[1 Обзор аппаратных возможностей микроконтроллера 4](#_Toc294594625)

[2 Выбор и обоснование функциональной схемы 8](#_Toc294594626)

[3 Принципиальная схема 9](#_Toc294594627)

[4 Алгоритм работы 11](#_Toc294594628)

[5 Оценка метрологических характеристик 13](#_Toc294594629)

[6 Список используемых источников 15](#_Toc294594630)

[Приложение 1. Листинг программы 16](#_Toc294594631)

# Введение

Целью данной работы является разработка устройства для измерения скважности импульсных сигналов на основе микроконтроллера семейства Intel8051.

Сква́жность (в физике, электронике) — один из классификационных признаков импульсных систем, определяющий отношение периода следования (повторения)импульса к его длительности. Величина, обратная скважности и часто используемая в англоязычной литературе, называется коэффициентом заполнения (англ. Duty cycle)

Таким образом, для импульсного сигнала справедливы следующие соотношения:

S = \frac{T}{\tau} = \frac{1}{D}~,

где S — скважность, D — коэффициент заполнения, T — период импульсов, \tau~ — длительность импульса.

Скважность определяет отношение пиковой [мощности](http://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D1%89%D0%BD%D0%BE%D1%81%D1%82%D1%8C) импульсной установки (например, передатчика [радиолокационной станции](http://ru.wikipedia.org/wiki/%D0%A0%D0%9B%D0%A1)) к её средней мощности и таким образом является важным показателем работы импульсных систем. В устройствах и системах дискретной передачи и обработки информации недостаточно высокая скважность может приводить к искажению информации.

Частое применение в практике находит сигнал со скважностью, равной двум — [меандр](http://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B0%D0%BD%D0%B4%D1%80_(%D1%80%D0%B0%D0%B4%D0%B8%D0%BE%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D0%BA%D0%B0)).

# 1 Обзор аппаратных возможностей микроконтроллера

В составе микроконтроллера Intel8051 есть два программируемых 16-битных таймера/счетчика (TCNT0 и TCNT1), которые могут быть использованы как таймеры или счетчики внешних событий. При работе в качестве таймера содержимое TCNT инкрементируется в каждом машинном цикле, т.е. через каждые 12 периодов резонатора. При работе в качестве счетчика содержимое TCNT инкрементируется под воздействием перехода из 1 в 0 внешнего входного сигнала, подаваемого на соответствующий (Т0, Т1) вывод МК. Так как на распознавание перехода требуется два машинных цикла, то максимальная частота подсчета входных сигналов равна 1/24 частоты резонатора. На длительность периода входных сигналов ограничений сверху нет. Для гарантированного прочтения входного считаемого сигнала он должен удерживать значение 1 как минимум в течение одного машинного цикла МК.

Для управления режимами работы TCNT и для организации взаимодействия таймеров с системой прерывания используются два регистра специальных функций (TMOD и TCON), описание которых приводится в табл. 1.1 и 1.2 соответственно. Как следует из описания управляющих бит TMOD, для обоих TCNT режимы работы 0, 1 и 2 одинаковы. Режимы 3 для TCNT0 и TCNT1 различны. Рассмотрим кратко работу TCNT во всех четырех режимах.

Таблица 1.1 — Регистр режима работы таймера/счетчика

|  |  |  |
| --- | --- | --- |
| Символ | Позиция | Имя и назначение |
| GATE | TMOD.7 для TCNT1 и TMOD.3 для TCNT0 | Управление блокировкой. Если бит установлен, то таймер/счетчик «x» разрешен до тех пор, пока на входе «INTx» высокий уровень и бит управления «TRx» установлен. Если бит сброшен, то TCNT разрешается, как только бит управления «TRx» устанавливается |
| С/Т | TMOD.6 для TCNT1 и TMOD.2 для TCNT0 | Бит выбора режима таймера или счетчика событий. Если бит сброшен, то работает таймер от внутреннего источника сигналов синхронизации. Если бит установлен, то работает счетчик от внешних сигналов на входе «Тх» |
| M1 | TMOD.5 для TCNT1 и TMOD.1 для TCNT0 | Режим работы (примечание) |
| М0 | TMOD.4 для TCNT1 и TMOD 0 для TCNT0 |  |

Примечание.

|  |  |  |
| --- | --- | --- |
| М1 | М0 | Режим работы |
| 0 | 0 | Таймер МК 8048. «TLx» работает как 5-битный предделитель |
| 0 | 1 | 16-битный таймер/счетчик. «ТН*х*» и «TLx» включены последовательно |
| 1 | 0 | 8-битный автоперезагружаемый таймер/счетчик. «ТНx» хранит значение, которое должно быть перезагружено в «TLx» каждый раз по переполнению |
| 1 | 1 | Таймер/счетчик 1 останавливается. Таймер/счетчик 0: TL0 работает как 8 ‑ битный таймер/счетчик, и его режим определяется управляющими битами таймера 0. ТН0 работает только как 8-битный таймер, и его режим определяется управляющими битами таймера 1 |

Таблица 1.2 — Регистр управления/статуса таймера

|  |  |  |
| --- | --- | --- |
| Символ | Позиция | Имя и назначение |
| TF1 | TCON.7 | Флаг переполнения таймера 1. Устанавливается аппаратно при переполнении таймера/счетчика. Сбрасывается при обслуживании прерывания аппаратно. |
| TR1 | TCON.6 | Бит управления таймера 1. Устанавливается/сбрасывается программой для пуска/останова |
| TF0 | TCON.5 | Флаг переполнения таймера 0. Устанавливается аппаратно. Сбрасывается при обслуживании прерывания |
| TR0 | TCON.4 | Бит управления таймера 0. Устанавливается/сбрасывается программой для пуска/останова таймера/счетчика |
| IE1 | TCON.3 | Флаг фронта прерывания 1. Устанавливается аппаратно, когда детектируется срез внешнего сигнала INT1. Сбрасывается при обслуживании прерывания |
| IT1 | TCON.2 | Бит управления типом прерывания 1. Устанавливается/ сбрасывается программно для спецификации запроса INT1 (срез/низкий уровень) |
| IE0 | TCON.1 | Флаг фронта прерывания 0. Устанавливается по срезу сигнала INT0. Сбрасывается при обслуживании прерывания |
| IT0 | TCON.0 | Бит управления типом прерывания 0. Устанавливается/ сбрасывается программно для спецификации запроса INT0 (срез/низкий уровень) |

Рассмотрим кратко работу таймеров/счетчиков (TCNT, от англ. timer/counter) во всех четырех режимах.

**Режим 0.** Перевод любого TCNT в режим 0 делает его похожим на таймер МК 8048 (8-битный счетчик), на вход которого подключен 5-битный предделитель частоты на 32. В режиме 0 работы TCNT таймерный регистр имеет разрядность 13 бит. При переходе из состояния «все единицы» в состояние «все нули» устанавливается флаг прерывания от таймера TF1. Входной синхросигнал таймера 1 разрешен (поступает на вход TCNT), когда управляющий бит TR1 установлен в 1 и либо управляющий бит GATE (блокировка) равен 0, либо на внешний вывод запроса прерывания INT1 поступает уровень 1. Установка бита GATE в 1 позволяет использовать таймер для измерения длительности импульсного сигнала, подаваемого на вход запроса прерывания.

**Режим 1.** Работа любого TCNT в режиме 1 такая же, как и в режиме 0, за исключением того, что таймерный регистр имеет разрядность 16 бит.

**Режим 2.** В режиме 2 работа организована таким образом, что переполнение (переход из состояния «все единицы» в состояние «все нули») таблица 1.5, регистр управления/статуса таймера 8-битного счетчика TL1 приводит не только к установке флага TF1, но и автоматически перезагружает в TL1 содержимое старшего байта (ТН1) таймерного регистра, которое предварительно было задано программным путем. Перезагрузка оставляет содержимое ТН1 неизменным. В режиме 2 TCNT0 и TCNT1 работают совершенно одинаково.

**Режим 3.** В режиме 3 TCNT0 и TCNT1 работают по-разному. TCNT1 сохраняет неизменным свое текущее содержимое. Иными словами, эффект такой же, как и при сбросе управляющего бита TR1 в нуль. В режиме 3 TL0 и ТН0 функционируют как два независимых 8-битных счетчика. Работу TL0 определяют управляющие биты TCNT0 (С/Т, GATE, TR0), входной сигнал INT0 и флаг переполнения TF0. Работу ТН0, который может выполнять только функции таймера (подсчет машинных циклов МК), определяет управляющий бит TR1. При этом ТН0 использует флаг переполнения TF1. Режим 3 используется в тех случаях применения МК 8051, когда требуется наличие дополнительного 8-битного таймера или счетчика событий. Можно считать, что в режиме 3 МК 8051 имеет в своем составе три таймера/счетчика. В том случае, если TCNT0 используется в режиме 3, TCNT1 может быть или включен, или выключен, или переведен в свой собственный режим 3, или может быть использован последовательным портом в качестве генератора частоты передачи, или, наконец, может быть использован в любом применении, не требующем прерывания.

# 2 Выбор и обоснование функциональной схемы

Так как в проектируемом устройстве измеряется параметр внешнего сигнала, то для одного из счётчиков будет использовать режим 1. Использование всех 16 бит позволяет расширить диапазон измеряемой длительности импульса. Второй счётчик будет работать также в режиме 1, но от внутреннего кварцевого резонатора.

Таким образом, счётчик TR1 служит для измерения периода сигнала, TR0 — для измерения длительности сигнала

Функциональная схема измерения длительности и периода сигнала приведена на рисунке 2.1

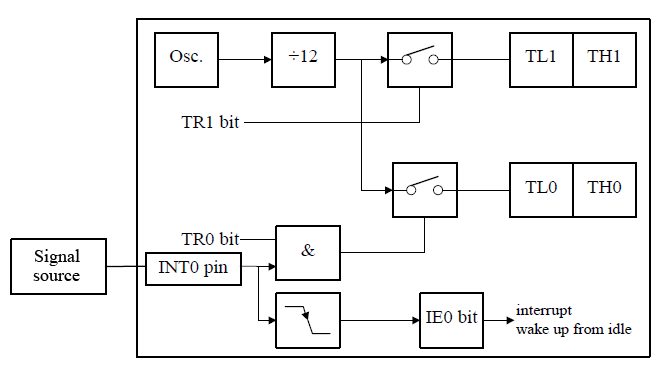


Рисунок 2.1 — Функциональная схема

Сигнал с делителя частоты поступает на входы счётчиков TR0 и TR1. При этом счётчик TR1 ведёт подсчёт всё время своей работы, а счётчик TR0 ведёт подсчёт только во время подачи на вход INT0(P3.2) положительного импульса.

Зная период сигнала T и его длительность t, можем найти скважность сигнала по формуле:

Для определения начала периода будем использовать прерывания, генерируемые по фронту импульса.

# 3 Принципиальная схема

На рисунке 3.1 приведена принципиальная схема разрабатываемого устройства.

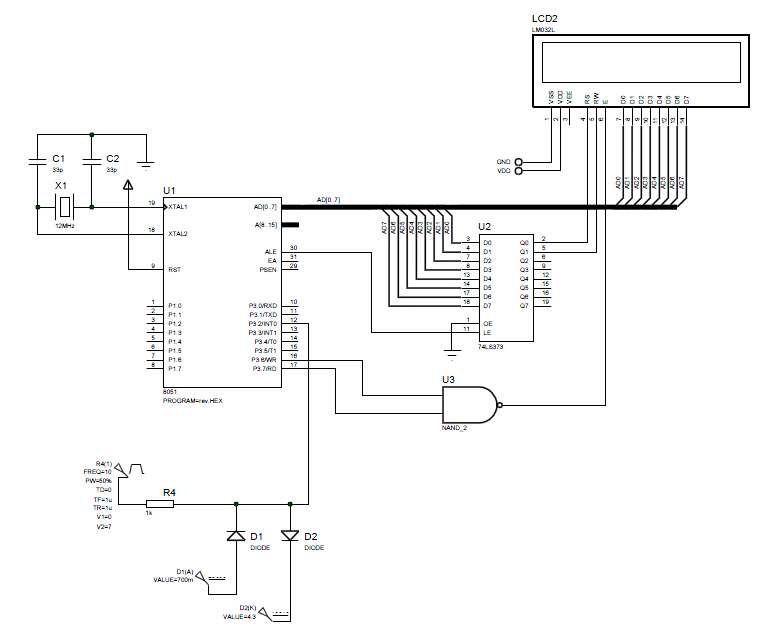


Рисунок 3.1 — Принципиальная схема измерителя скважности

Для тактирования микроконтроллера используется кварцевый резонатор со стандартной частотой в 12MHz. В качестве устройства отображения результата измерения LCD панель, подключённая через шину к порту P0.

Для предотвращения подачи на вход сигнала превышающего 5В используем двухтактный диодный ограничитель.

Тогда, источник питания в цепи диода D2 должен обеспечивать 4,3 В питания(4,3В+0,7В = 5В) для ограничения сверху до 5В. А в цепи диода D1 — 0,7 В (0,7-0,7 = 0В) для ограничения снизу.

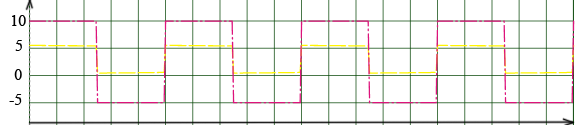


Рисунок 3.2 — Иллюстрация работы диодного ограничителя(красная линия — источник питания, жёлтая — на выходе ограничителя).

На графике 3.2 изображён процесс ограничения по напряжения с помощью диодного ограничителя.

Используемые элементы схемы:

LCD2 — LCD панель HD47780,

D1,D2 — диоды,

R4 — резистор номинала 1 кОм,

U3 — штрих Шеффера,

U2 — D-триггер 74LS373 для организации работы с шиной

U1 — микроконтроллер Intel8051

X1 — кварцевый резонатор на 12MHz

# 4 Алгоритм работы

На блок схеме 4.1 изображён алгоритм работы программы микроконтроллера.

Инициализация

Главный цикл

Обработка прерывания

0

1

IP.7

Вывод результата на LCD

Рисунок 4.1 — Блок схема работы алгоритма программы

Условно процесс работы программы можно разделить на две логических части. Ожидания результата измерения(«главный цикл») и обработку прерывания. Само измерение происходит в фоновом режиме и не нуждается в контроле. Задача программы определить моменты готовности результата измерения и сигнализировать об этом остальным подпрограммам.

Прерывание установлено на фронт импульса. Бит IP.7 является флагом, который сигнализирует о готовности результатов измерения.

Нет

Да

TR0 запущен?

Остановка счётчика

Очистка счётчиков

Установка бита IP.7

Запуск счётчиков

Рисунок 4.2 — Блок схема работы подпрограммы обработки прерывания

# 5 Оценка метрологических характеристик

Диапазон измерения определяется максимально возможным периодом и минимально возможной длительностью сигнала.

Так как счётчики 16 битные и тактирование осуществляется с делителя частоты кварцевого резонатора на 12, то

Так как для гарантированного прочтения входного считаемого сигнала он должен удерживать значение 1 как минимум в течение одного машинного цикла МК, то минимальная длительность сигнала составляет:

Таким образом, минимальная частота входного сигнала:

Максимальная частота определяется минимально возможной длительностью:

Основными источниками погрешности измерения в данном устройстве могут служить: нестабильность кварцевого резонатора, не идеальность среза фронта импульса.

Относительная погрешность измерения периода импульсного сигнала нормируется по следующей формуле:

,

где — нестабильность кварцевого генератора, — период счётных импульсов, — значение измеренного периода.

Относительная погрешность измерения периода импульсного сигнала нормируется по следующей формуле

,

где — дополнительная составляющая погрешность обусловленная крутизной фронта и спада импульса.

Тогда относительная погрешность измерения скважности сигнала определяется следующей формулой

# 6 Список используемых источников

1 <http://ru.wikipedia.org/wiki/Скважность>

2 Paul C. de Jong and Ferry N. Toth “Measuring Duty Cycles with an Intel MCS-51 Microcontroller”

3 Kenneth J. Ayala “The 8051 Microcontroller architecture, programming and applications”.

4 А.Л. Гурский «Лабораторный практикум по курсу «Цифровые и микропроцессорные устройства средств измерений» для специальности 1-54 01 04 «Метрологическое обеспечение информационных систем и сетей»»

5 Елизаров А.С. «Электрорадиоизмерения» [Учебник для вузов по спец. «Радиотехника»]. — Мн.: Выш. шк., 1986. — с. 132 - 137

# Приложение 1. Листинг программы

;LCD Registers addresses

LCD\_CMD\_WR equ 0

LCD\_DATA\_WR equ 1

LCD\_BUSY\_RD equ 2

LCD\_DATA\_RD equ 3

;LCD Commands

LCD\_CLS equ 1

LCD\_HOME equ 2

LCD\_SETMODE equ 4

LCD\_SETVISIBLE equ 8

LCD\_SHIFT equ 16

LCD\_SETFUNCTION equ 32

LCD\_SETCGADDR equ 64

LCD\_SETDDADDR equ 128

org 0000h

sjmp over

;====================================================================================

; Interuption handler

;====================================================================================

org 0003H

jbc tcon.4, stop ;if TR0 is running stop it

setb tcon.4 ;else run it

mov TL0, #00h ; clear TR0

mov TH0, #00h ; clear TR1

mov TL1, #00h

mov TH1, #00h

setb TR1 ;start TR1

fin:

reti

;====================================================================================

; Stop TR0 subroutine and clear counters

;====================================================================================

stop:

setb IP.7 ;notify the main programm. the result of the measurement is ready

mov R3, TH0 ; mov TR0 and TR1 to R3-R6 registers

mov R4, TL0

mov R5, TH1

mov R6, TL1

clr TR0

clr TR1

mov TL0, #00h ;clear TR0 and TR1

mov TH0, #00h

mov TL1, #00h

mov TH1, #00h

reti

;====================================================================================

; Setup subroutine. Run once.

;====================================================================================

over:

; set timer mode

mov DPTR, #0x0455

mov TMOD, #19h

mov TCON, #01h

; set interuption

setb EX0

setb EA

; clear timers

mov TL0, #00h

mov TH0, #00h

mov th1, #00h

mov tl1, #00h

mov ie, #81h

setb ppflag

; set LCD

mov A,#038h

MOV p1, #00011011b

call wrcmd

;====================================================================================

; Main subroutine. Loop ever. Check result status and if result is ready - display it

;====================================================================================

simulate:

jbc IP.7, getw

sjmp simulate

;====================================================================================

; Obtaining the measurement result and display it

;====================================================================================

getw:

clr EA

call init\_lcd

mov R1, #0

;=============================================

; backup timer0 value

;=============================================

mov 53h, R3 ;low

mov 54h, R4 ;high

;=============================================

; prepare for a division

;=============================================

mov A, R6

mov R0, A ;low

mov A, R5

mov R1, A ;high

mov A, R4

mov R2, A ;low

;=============================================

; call division subroutine

;=============================================

call DIV16

;=============================================

; call backup remainder of division subroutine

;=============================================

call backup\_remainder

;=============================================

; prepare for a convertation to ASCII code

;=============================================

mov A, R1

mov R3, A

mov A, R0

mov R2, A

;=========================================================================

; call convertation subroutine and display the result

; Input: R2 (low) and R3 (high) (binary)

; Output: 30h,31h,32h,33h,34h (internal RAM address) (ASCII codes)

; high--^ ^------low

;=========================================================================

call BINTOASC

call display\_ascii

;=============================================

; display dot symbol from DB

;=============================================

mov DPTR, #string4

call wrstr

;=============================================

; extract remainder from internal RAM

;=============================================

call extract\_remainder

;============================================================================

; Multiply the remainder by 10000 and divide by Timer0 value

; to calculate a fraction part

; For example:

; A = 4999 and B = 2500

; DIV A,B is result Q=1 as integral part and R=2499 as remainder part

; But we need 1.9996

; 9996 we can get if we devide R\*10000/B

;============================================================================

mov R1, #27h

mov R0, #10h

call UMUL16

mov R4, 54h ;low

mov R5, 53h ;high

call DIV32

mov A, R0

mov R2, ACC

mov A, R1

mov R3, A

;=========================================================================

; call convertation subroutine and display the result

;=========================================================================

call BINTOASC

call display\_ascii

; return to the main loop

setb EA

sjmp simulate

;====================================================================================

; LCD initialisation

;====================================================================================

init\_lcd:

mov A,#LCD\_SETVISIBLE+6 ;Make the display & blink visible:

call wrcmd

mov A,#LCD\_CLS ;Clear screen

call wrcmd

mov A,#LCD\_SETDDADDR+0

call wrcmd

mov DPTR, #string1a

call wrstr

ret

;====================================================================================

; Backup the remainder to the data storage

;====================================================================================

backup\_remainder:

mov A, R2 ; R2 - low

mov 36h, A

mov A, R3 ; R3 - high

mov 37h, A

ret

;====================================================================================

; Extract the remainder from the data storage

;====================================================================================

extract\_remainder:

mov A, 36h

mov R2, A

mov A, 37h

mov R3, A

ret

;====================================================================================

; Calculate ASCII code for each number and display it. Skip zero value

;====================================================================================

display\_ascii:

mov R1, #30h

setb F0

bigloop:

jb F0, miniloop

ret

miniloop:

inc R1

mov A, @R1

CJNE R1, #35h, countinue

clr F0

jmp bigloop

showDigit:

mov 40h, R1

call wrdata

mov A, 40h

mov R1, A

jmp bigloop

countinue:

CJNE A, #30h, showDigit

jmp bigloop

;====================================================================================

; Subroutine to write null terminated string at DPTR in program ram.

;====================================================================================

wrstr: mov R0,#LCD\_DATA\_WR

wrstr1: clr A

movc A,@A+DPTR

jz wrstr2

movx @R0,A

call wtbusy

inc DPTR

push DPL

push DPH

pop DPH

pop DPL

jmp wrstr1

wrstr2: ret

wrdata:

mov R0, #LCD\_DATA\_WR

movx @R0, A

call wtbusy

ret

;====================================================================================

;Sub routine to write null terminated string at DPTR in program ram. Slowly

;====================================================================================

wrslow: mov R0,#LCD\_DATA\_WR

wrslw1: clr A

movc A,@A+DPTR

jz wrslw2

movx @R0,A

call wtbusy

inc DPTR

push DPL

push DPH

mov DPTR,#100

call wtms

pop DPH

pop DPL

jmp wrslw1

wrslw2: ret

;====================================================================================

;Sub routine to write command:

;====================================================================================

wrcmd: mov R0,#LCD\_CMD\_WR

movx @R0,A

jmp wtbusy

;====================================================================================

;Sub routine to write character:

;====================================================================================

wrchar: mov R0,#LCD\_DATA\_WR

movx @R0,A

;====================================================================================

;Subroutine to wait for busy clear

;====================================================================================

wtbusy: mov R1,#LCD\_BUSY\_RD

movx A,@r1

jb ACC.7,wtbusy

ret

;====================================================================================

;Wait for number of seconds in A

;====================================================================================

wtsec: push ACC

call wtms

pop ACC

dec A

jnz wtsec

ret

string1a:db 'Duty cycle = '

db 0

string4:db '.'

db 0

;==========================================================

; Routine to convert a 16bit binary number in ASCII |

; Input : R2 (Lsb) and R3 (Msb) ( binary)

; Output : 30h,31h,32h,33h,34h (internal RAM address) (ASCII)

; msb--^ ^------lsb

BINTOASC:

MOV R0,#30h ; R0 = POUT

MOV DPTR,#TAB ; R=TAB(P)

COM1:

CLR A ; P <- 0

MOVC A,@A+DPTR ; R <- TAB(P)

MOV R7,A

INC DPTR

CLR A

MOVC A,@A+DPTR

MOV R6,A

MOV R4,#'0' ; C <- '0'

SOMA: ; N <- N-R

CLR C ;

MOV A,R2 ;

SUBB A,R6 ;

MOV R2,A ;

MOV A,R3 ;

SUBB A,R7 ;

MOV R3,A ;

JC SAIDA ; If < 0 goto SAIDA

INC R4 ; If >0 then C <- C +1

SJMP SOMA ; goto SOMA

SAIDA:

MOV A,R4

MOV @R0,A ;TABOUT (POUT) <- C

MOV A,R2

ADD A,R6 ; N=N+R

MOV R2,A

MOV A,R3

ADDC A,R7

MOV R3,A

INC R0 ; PSAIDA=PSAIDA +1

CLR A

MOVC A,@A+DPTR

CJNE A,#1,INCREMENTA ; TAB(P) = 1 ?

RET ; If yes, END

INCREMENTA: ; If No, P <- P+1

INC DPTR

LJMP COM1 ; goto COM1

TAB:

DW 10000

DW 1000

DW 100

DW 10

DW 1

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* \*

;\* Maths Subroutines for the 8051 microcontroller \*

;\* W.G.Marshall 2002 \*

;\* \*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; All parameters in Register bank 0, (r0 to r7)

; Bits 21H and 22H reserved for sign bits

;===================================================================

; subroutine Cr0r1

; 16-Bit 2's Complement -> magnitude / Sign Bit Conversion

;

; input: r1, r0 = signed word

;

; output: r1, r0 = magnitude

; Bit 21H = sign (21H is set if negative number)

;

; alters: acc, C

;===================================================================

Cr0r1: mov a, r1 ; high byte into accumulator

jb acc.7, c0a ; negative if bit 7 is 1

clr 21H ; clear sign bit if 'positive'

ret ; done

c0a: setb 21H ; set sign flag

mov a, r0 ; number is negative

cpl a ; complement

add a, #1 ; and add +1

mov r0, a

mov a, r1 ; get next byte

cpl a ; complement

addc a, #0

mov r1, a

ret

;====================================================================

; subroutine Cr2r3

; 16-Bit 2's Complement -> magnitude / Sign Bit Conversion

;

; input: r3, r2 = signed word

;

; output: r3, r2 = magnitude

; Bit 22H = sign (22H is set if negative number)

;

; alters: acc, C

;====================================================================

Cr2r3: mov a, r3 ; read high into accumulator

jb acc.7, c1a ; negative if bit 7 is 1

clr 22H ; clear sign bit if 'positive'

ret ; done

c1a: setb 22H ; set sign flag

mov a, r2 ; number is negative

cpl a ; complement

add a, #1 ; and add +1

mov r2, a

mov a, r3 ; get next byte

cpl a ; complement

addc a, #0

mov r3, a

ret

;====================================================================

; subroutine Cr4r5

; 16-Bit 2's Complement -> magnitude / Sign Bit Conversion

;

; input: r5, r4 = signed word

;

; output: r5, r4 = magnitude

; Bit 22H = sign (22H is set if negative number)

;

; alters: acc, C

;====================================================================

Cr4r5: mov a, r5 ; read high into accumulator

jb acc.7, c3a ; negative if bit 7 is 1

clr 22H ; clear sign bit if 'positive'

ret ; done

c3a: setb 22H ; set sign flag

mov a, r4 ; number is negative

cpl a ; complement

add a, #1 ; and add +1

mov r4, a

mov a, r5 ; get next byte

cpl a ; complement

addc a, #0

mov r5, a

ret

;====================================================================

; subroutine Cr0r3

; 32-Bit 2's Complement -> magnitude / Sign Bit Conversion

;

; input: r3, r2, r1, r0 = signed word

;

; output: r3, r2, r1, r0 = magnitude

; Bit 21H = sign (21H is set if negative number)

;

; alters: acc

;====================================================================

Cr0r3: mov a, r3 ; read high into accumulator

jb acc.7, c2a ; negative if bit 7 is 1

clr 21H ; clear sign flag if 'positive'

ret ; done

c2a: setb 21H ; set sign flag

mov a, r0 ; number is negative

cpl a ; complement

add a, #1 ; and add +1

mov r0, a

mov a, r1 ; get next byte

cpl a ; complement

addc a, #0

mov r1,a

mov a, r2 ; get next byte

cpl a ; complement

addc a, #0

mov r2,a

mov a, r3 ; get next byte

cpl a ; complement

addc a, #0

mov r3, a

ret ; done

;====================================================================

; subroutine Mr0r1

; 16-Bit magnitude / Sign Bit -> 2's Complement Conversion

;

; input: r1, r0 = magnitude

; Bits 21H & 22H = sign bits of operands X and Y

; (set if negative)

;

; output: r1, r0 = signed word

;

; alters: acc, C

;====================================================================

Mr0r1: jb 21H, Mr0r1b ; test X sign

jb 22H, Mr0r1a ; test Y sign

ret

Mr0r1b: jnb 22H, Mr0r1a

ret

Mr0r1a: mov a, r0 ; negate number

cpl a ; complement

add a, #1 ; and add +1

mov r0, a

mov a, r1 ; get next byte

cpl a ; complement

addc a, #0

mov r1, a

ret

;====================================================================

; subroutine Mr0r3

; 32-Bit magnitude / Sign Bit -> 2's Complement Conversion

;

; input: r3, r2, r1, r0 = magnitude

; Bits 21H & 22H = sign bits of operands X and Y

; (set if negative)

;

; output: r3, r2, r1, r0 = signed word

;

; alters: acc, C

;====================================================================

Mr0r3: jb 21H, Mr0r3b ; test X sign

jb 22H, Mr0r3a ; test Y sign

ret

Mr0r3b: jnb 22H, Mr0r3a

ret

Mr0r3a: mov a, r0 ; negate number

cpl a ; complement

add a, #1 ; and add +1

mov r0, a

mov a, r1 ; get next byte

cpl a ; complement

addc a, #0

mov r1, a

mov a, r2 ; get next byte

cpl a ; complement

addc a, #0

mov r2, a

mov a, r3 ; get next byte

cpl a ; complement

addc a, #0

mov r3, a

ret ; done

;====================================================================

; subroutine MUL16

; 16-Bit x 16-Bit to 32-Bit Product Signed Multiply

; 2's Complement format

;

; input: r1, r0 = multiplicand X

; r3, r2 = multiplier Y

;

; output: r3, r2, r1, r0 = product P = X x Y

;

; calls: UMUL16, Cr0r1, Cr2r3, Mr0r3

;

; alters: acc, C, Bits 21H & 22H

;====================================================================

MUL16: anl PSW, #0E7H ; Register Bank 0

acall Cr0r1 ; 2's comp -> Mag/Sign

acall Cr2r3 ; 2's comp -> Mag/Sign

acall UMUL16

acall Mr0r3 ; Mag/Sign -> 2's Comp

ret

;====================================================================

; subroutine UMUL16

; 16-Bit x 16-Bit to 32-Bit Product Unsigned Multiply

;

; input: r1, r0 = multiplicand X

; r3, r2 = multiplier Y

;

; output: r3, r2, r1, r0 = product P = X x Y

;

; alters: acc, C

;====================================================================

UMUL16: push B

push dpl

mov a, r0

mov b, r2

mul ab ; multiply XL x YL

push acc ; stack result low byte

push b ; stack result high byte

mov a, r0

mov b, r3

mul ab ; multiply XL x YH

pop 00H

add a, r0

mov r0, a

clr a

addc a, b

mov dpl, a

mov a, r2

mov b, r1

mul ab ; multiply XH x YL

add a, r0

mov r0, a

mov a, dpl

addc a, b

mov dpl, a

clr a

addc a, #0

push acc ; save intermediate carry

mov a, r3

mov b, r1

mul ab ; multiply XH x YH

add a, dpl

mov r2, a

pop acc ; retrieve carry

addc a, b

mov r3, a

mov r1, 00H

pop 00H ; retrieve result low byte

pop dpl

pop B

ret

;====================================================================

; subroutine DIV16

; 16-Bit / 16-Bit to 16-Bit Quotient & remainder signed Divide

; 2's Complement Format

;

; input: r1, r0 = Dividend X

; r3, r2 = Divisor Y

;

; output: r1, r0 = quotient Q of division Q = X / Y

; r3, r2 = remainder

; Carry C is set if Y = 0, i.e. divide by 0 attempted

;

; calls: UDIV16, Cr0r1, Cr2r3, Mr0r1

;

; alters: acc, r4, r5, r6, r7, flags, Bits 21H & 22H

;====================================================================

DIV16: anl PSW, #0E7H ; Register Bank 0

mov a, r3 ; get divisor high byte

orl a, r2 ; OR with low byte

jnz div\_OK ; divisor OK if not 0

setb C ; else, overflow

ret

div\_OK: push dpl

push dph

push b

acall Cr0r1 ; 2's comp -> Mag/Sign

acall Cr2r3 ; 2's comp -> Mag/Sign

acall UDIV16

acall Mr0r1 ; Mag/Sign -> 2's Comp

clr C

pop b

pop dph

pop dpl

ret ; done

;====================================================================

; subroutine UDIV16

; 16-Bit / 16-Bit to 16-Bit Quotient & Remainder Unsigned Divide

;

; input: r1, r0 = Dividend X

; r3, r2 = Divisor Y

;

; output: r1, r0 = quotient Q of division Q = X / Y

; r3, r2 = remainder

;

; alters: acc, B, dpl, dph, r4, r5, r6, r7, flags

;====================================================================

UDIV16: mov r7, #0 ; clear partial remainder

mov r6, #0

mov B, #16 ; set loop count

div\_loop: clr C ; clear carry flag

mov a, r0 ; shift the highest bit of

rlc a ; the dividend into...

mov r0, a

mov a, r1

rlc a

mov r1, a

mov a, r6 ; ... the lowest bit of the

rlc a ; partial remainder

mov r6, a

mov a, r7

rlc a

mov r7, a

mov a, r6 ; trial subtract divisor

clr C ; from partial remainder

subb a, r2

mov dpl, a

mov a, r7

subb a, r3

mov dph, a

cpl C ; complement external borrow

jnc div\_1 ; update partial remainder if

; borrow

mov r7, dph ; update partial remainder

mov r6, dpl

div\_1: mov a, r4 ; shift result bit into partial

rlc a ; quotient

mov r4, a

mov a, r5

rlc a

mov r5, a

djnz B, div\_loop

mov a, r5 ; put quotient in r0, and r1

mov r1, a

mov a, r4

mov r0, a

mov a, r7 ; get remainder, saved before the

mov r3, a ; last subtraction

mov a, r6

mov r2, a

ret

;====================================================================

; subroutine DIV32

; 32-Bit / 16-Bit to 32-Bit Quotient & remainder signed Divide

; 2's Complement Format

;

; input: r3, r2, r1, r0 = Dividend X

; r5, r4 = Divisor Y

;

; output: r3, r2, r1, r0 = quotient Q of division Q = X / Y

; r7, r6, r5, r4 = remainder

; Carry C is set if Y = 0, i.e. divide by 0 attempted

;

; calls: UDIV32, Cr0r3, Cr4r5, Mr0r3

;

; alters: acc, flags, Bits 21H & 22H

;====================================================================

DIV32: anl PSW, #0E7H ; Register Bank 0

mov a, r4 ; get divisor high byte

orl a, r5 ; OR with low byte

jnz div32\_OK ; divisor OK if not 0

setb C ; else, overflow

ret

div32\_OK: acall Cr0r3 ; 2's comp -> Mag/Sign

acall Cr4r5 ; 2's comp -> Mag/Sign

acall UDIV32

acall Mr0r3 ; Mag/Sign -> 2's Comp

clr C ; divisor is not 0

ret ; done

;====================================================================

; subroutine UDIV32

; 32-Bit / 16-Bit to 32-Bit Quotient & Remainder Unsigned Divide

;

; input: r3, r2, r1, r0 = Dividend X

; r5, r4 = Divisor Y

;

; output: r3, r2, r1, r0 = quotient Q of division Q = X / Y

; r7, r6, r5, r4 = remainder

;;

; alters: acc, flags

;====================================================================

UDIV32: push 08 ; Save Register Bank 1

push 09

push 0AH

push 0BH

push 0CH

push 0DH

push 0EH

push 0FH

push dpl

push dph

push B

setb RS0 ; Select Register Bank 1

mov r7, #0 ; clear partial remainder

mov r6, #0

mov r5, #0

mov r4, #0

mov B, #32 ; set loop count

div\_lp32: clr RS0 ; Select Register Bank 0

clr C ; clear carry flag

mov a, r0 ; shift the highest bit of the

rlc a ; dividend into...

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

setb RS0 ; Select Register Bank 1

mov a, r4 ; ... the lowest bit of the

rlc a ; partial remainder

mov r4, a

mov a, r5

rlc a

mov r5, a

mov a, r6

rlc a

mov r6, a

mov a, r7

rlc a

mov r7, a

mov a, r4 ; trial subtract divisor from

clr C ; partial remainder

subb a, 04

mov dpl, a

mov a, r5

subb a, 05

mov dph, a

mov a, r6

subb a, #0

mov 06, a

mov a, r7

subb a, #0

mov 07, a

cpl C ; complement external borrow

jnc div\_321 ; update partial remainder if

; borrow

mov r7, 07 ; update partial remainder

mov r6, 06

mov r5, dph

mov r4, dpl

div\_321: mov a, r0 ; shift result bit into partial

rlc a ; quotient

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

djnz B, div\_lp32

mov 07, r7 ; put remainder, saved before the

mov 06, r6 ; last subtraction, in bank 0

mov 05, r5

mov 04, r4

mov 03, r3 ; put quotient in bank 0

mov 02, r2

mov 01, r1

mov 00, r0

clr RS0

pop B

pop dph

pop dpl

pop 0FH ; Retrieve Register Bank 1

pop 0EH

pop 0DH

pop 0CH

pop 0BH

pop 0AH

pop 09

pop 08

ret

end