ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
СРЕДНЕГО (ПОЛНОГО) ОБЩЕГО ОБРАЗОВАНИЯ
ЛИЦЕЙ ПРИ СПБГУТ

Вендор-ориентированный учебный курс в системе
«Старшая профильно-профессиональная школа-ВУЗ-Работодатель»:
«Программирование микроконтроллеров Microchip»

Богураев М.В., Кисляков С.В.

«ПРОГРАММНАЯ ОБРАБОТКА ПРЕРЫВАНИЯ»

Методические указания к выполнению лабораторной работы

Санкт - Петербург
2009

Богураев М.В., Кисляков С.В. «ПРОГРАММНАЯ ОБРАБОТКА ПРЕРЫВАНИЯ». Методические указания к выполнению лабораторной работы №5(7). СПб: ГОУ «Лицей при СПбГУТ», 2009.

ЛАБОРАТОРНАЯ РАБОТА №5
«ПРОГРАММНАЯ ОБРАБОТКА ПРЕРЫВАНИЯ»

Цель работы

Ознакомиться с программированием прерываний микроконтроллеров PIC среднего подсемейства. Освоить симуляцию в MPLAB IDE программ с прерываниями для микроконтроллеров PIC16F877A. Овладеть навыками программирования PIC микроконтроллеров.

Теоретические основы

Микроконтроллер снабжён внутренними периферийными модулями и может взаимодействовать с внешними устройствами. Взаимодействие – это управление модулем или устройством, а также обмен информацией. И периферийные модули, и внешние устройства могут работать в асинхронном режиме или иметь свой тактовый генератор с частотой, отличной от частоты, на которой работает микроконтроллер. В этих устройствах могут происходить события. С точки зрения аппаратного обеспечения события – это изменения электрических уровней. Например, нажатие кнопки вызывает изменение напряжения на выводе RB0/INT; окончание работы счётчика или таймера вызывает установку флага, допустим это T0IF. На происходящие события можно реагировать по-разному. Рассмотрим некоторые возможные варианты реакции на события:

1. Можно осуществлять постоянный контроль процесса - реакция на событие будет мгновенной. Но ничего другого микроконтроллер с такой программой сделать не сможет.

2. Можно выполнять текущие задачи, а наступление события проверять периодически. По сравнению с первым вариантом появляется возможность обрабатывать несколько задач. Однако, помимо того, что потребуется программная поддержка циклов опроса во всех частях программы, есть вероятность пропустить момент наступления события и не отреагировать вовремя.

3. Если назначить бит (флаг), который изменит значение, когда произойдёт событие. Тогда можно будет выполнять текущие задачи, а событие обрабатывать только после изменения флага, а когда событие обработано, вернуться к продолжению выполнения текущих задач. Реакция будет несколько замедленной, поскольку нужно сделать контекстное сохранение регистров и перейти к подпрограмме обработки прерывания (ISR – Interrupt Service Routine).

Каждый из вариантов, или их разные сочетания, могут быть применены в решении задач. Метод реакции на события выбирается так, чтобы способствовать решению задачи с имеющимся аппаратным обеспечением.

Реакция на событие в соответствии с третьим пунктом называется прерыванием. Прерывание – это действие микропроцессорной системы в тот момент, когда совершается событие.

Механизм прерываний можно описать так: сразу после совершения события возникает прерывание, - выполнение текущей программы приостанавливается; в стеке сохраняется адрес команды, которая выполнится после окончания прерывания; микроконтроллер переходит к выполнению подпрограммы, расположенной по адресу 0x04 в памяти команд. Эта подпрограмма называется обработчиком прерывания (ISR – Interrupt Service Routine). После выполнения обработчика прерывания продолжается выполнение прерванной программы по адресу, который из стека переписывается в счётчик команд (PC – Program Counter). Обработчик прерывания должен заканчиваться одной из команд возврата: RETFIE, RETURN, RETLW.

Для настройки прерываний используются регистры INTCON, PIE1, PIE2, PIR1, PIR2. Все эти регистры находятся в памяти данных в области SFR. В регистрах PIE1 и PIE2 содержатся биты разрешения прерываний, а в регистрах PIR1 и PIR2 флаги прерываний.

Регистр INTCON (Interrupt Configuration) является очень важным регистром и поэтому доступен во всех четырёх банках памяти данных. Функция регистра INTCON заключается в настройке и разрешении (или запрете) прерываний. Для осуществления этой функции в регистре имеются биты настройки прерываний. Все биты в регистрах, связанных с прерываниями, делятся на две группы: биты разрешения/запрета прерываний и биты – флаги прерываний.

Биты разрешения/запрета прерываний запрещают или разрешают прерывание. Эти биты имеют название вида ???E (Enable); например, бит T0IE разрешает прерывание от нулевого таймера (Timer 0 Interrupt Enable). Функция бита GIE (Global Interrupt Enable) в регистре INTCON заключается в разрешении или запрете абсолютно всех прерываний. В момент входа в прерывание микроконтроллер сбрасывает этот бит. Чтобы после обработки текущего прерывания разрешить следующее прерывание программист должен установить этот флаг. Возможно использование команды RETFIE, которая завершает текущее прерывание и разрешает следующее, так как устанавливает бит GIE.

Флаг прерывания – это специальный бит, который устанавливается (или сбрасывается) сразу же после наступления события, которое может вызвать прерывание. Флаги сигнализируют о наступлении события и устанавливаются независимо от того, разрешены прерывания или нет. Эти биты имеют название вида ???F (Flag); например, бит T0IF (Timer 0 Interrupt Flag) устанавливается (принимает значение «1») если произошло событие – переполнение нулевого таймера. Флаги прерываний должны сбрасываться программистом в программе обработки прерывания. Это необходимо для исключения повторного вхождения в прерывание по ранее установленному флагу – так как прерывание по этому флагу уже обработано. Если T0IE и T0IF установлены, то в результате переполнения нулевого таймера возникнет прерывание.

Комбинацию установленных и сброшенных битов разрешения/запрета прерываний называют маской прерываний. Запрещённые прерывания называют замаскированными. Разрешённые прерывания называют незамаскированными.

Адрес начала программы обработчика прерываний (ISR – Interrupt Service Routine) называют вектором прерывания. В микроконтроллерах PIC среднего семейства (PIC16F877A – тоже относится к этому семейству) вектор прерывания один. Адрес вектора прерываний 0x04, это адрес ячейки памяти команд.

В механизме прерываний важную роль играет стек. Стек нужен для того, чтобы сохранять адрес возврата. Это адрес, по которому продолжится выполнение прерванной программы. Стек в микроконтроллерах PIC среднего семейства аппаратный и недоступен из программы напрямую. Стек имеет восемь уровней и может содержать восемь адресов возврата. Однако стек не имеет никаких механизмов для сигнализации о переполнении. Переполнение – это ситуация, когда все восемь уровней стека уже заняты, но нужно сохранить ещё один адрес. Такая ситуация в результате приведёт к потере первого адреса возврата и вызовет сбой программы. Контроль стека осуществляется программистом.

Перед вхождением в прерывание нужно сохранить значения регистров W и STATUS. Этот процесс называется контекстным сохранением регистров. Это сохранение должно происходить в первых строках обработчика прерываний и необходимо для того, чтобы в прерванной программе не произошло сбоя после прерывания. Сбой возможен по той причине, что прерванная программа использует W для передачи данных. Обработчик прерывания наверняка изменит содержимое W, тогда после завершения прерывания программа получает изменённые данные в W. То же самое может произойти с регистром STATUS. Поэтому указанные регистры сначала надо сохранить, а перед выходом из прерывания восстановить прежние (до входа в прерывание) значения.

Как и в предыдущей лабораторной работе в этой используется кнопка. Кнопка подсоединена к порту ввода-вывода. Порт должен работать на вход. Для устранения влияния дребезга контактов применяют задержку с повторным опросом кнопки. На рис. 1 а) приведена временная диаграмма замыкания кнопки как идеального ключа. На рис. 1 б) приведён пример замыкания реальной кнопки с дребезгом контактов.

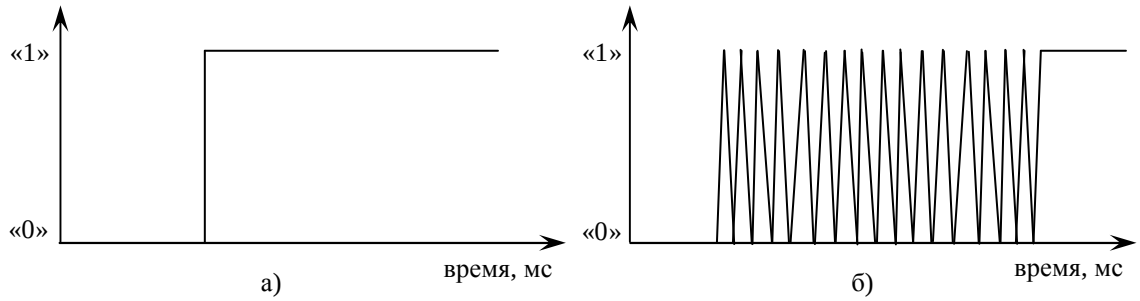

Рис. 1. Включение идеального ключа и реальной кнопки.

Устройством ввода является кнопка, а устройством вывода – светодиод, который подключается к PORTD,0. Светодиод зажигается по нажатию кнопки. Повторное нажатие кнопки отключает светодиод. Следующее нажатие включит светодиод и так далее.

Задание

Создайте проект, откомпилируйте программу Project7. Создайте файл стимулов. Запустите программу в симуляторе MPLAB IDE и промоделируйте замыкание кнопки в симуляторе. Расположите точки останова в начале и в конце обработчика прерывания, проанализируйте работу стека и подпрограммы обработки прерывания. Запустите программу на лабораторном макете в режиме RUN и продемонстрируйте результат работы программы.

Порядок выполнения

На диске C:\ в папке Projects создайте папку Project7. В эту папку скопируйте файл Project7 запустите MPLAB IDE и создайте проект с названием Project7. Откомпилируйте программу.

Для проведения симуляции работы программы нужно выбрать и настроить симулятор, файл стимулов и окно Watch. В качестве отладчика выберите MPLAB SIM. Задайте параметры моделирования: выберите в меню Debugger пункт Settings (рис. 2).

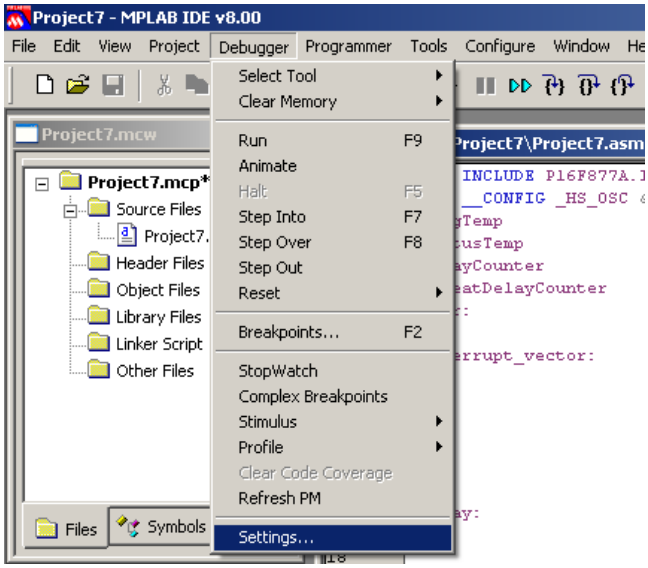

Рис. 2. Вызов окна с настройками симулятора.

В открывшемся окне Simulator Settings (рис. 3) выберите вкладку Animation/Realtime Updates и поставьте галочку напротив Enable Realtime watch updates. В графе Animate step time можно установить время шага для пошагового выполнения программы. По умолчанию оно составляет 300 миллисекунд. В графе Realtime watch можно установить время обновления содержимого окна – здесь, вместо 10 по умолчанию, стоит написать 1. Тогда окно Watch будет обновляться через каждые 100 миллисекунд. Теперь надо нажать кнопку применить и OK.

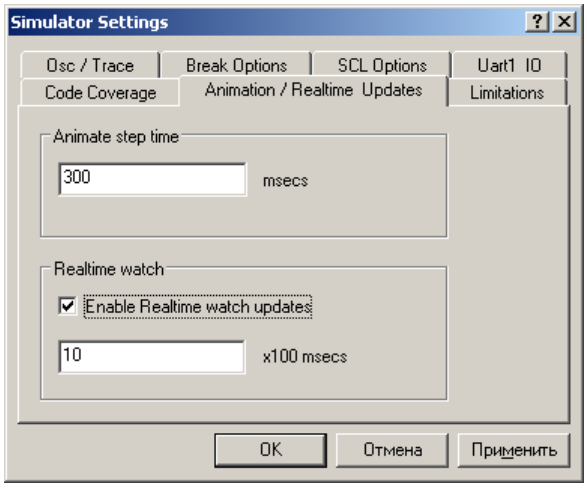

Рис. 3. Настройка симулятора.

После того, как симулятор готов к работе следует создать файл стимулов. В этом файле определяется выводы, на которых будут моделироваться высокие или низкие уровни. Для создания файла стимулов в пункте меню Debugger/Stimulus выберите New Workbook (рис. 4)

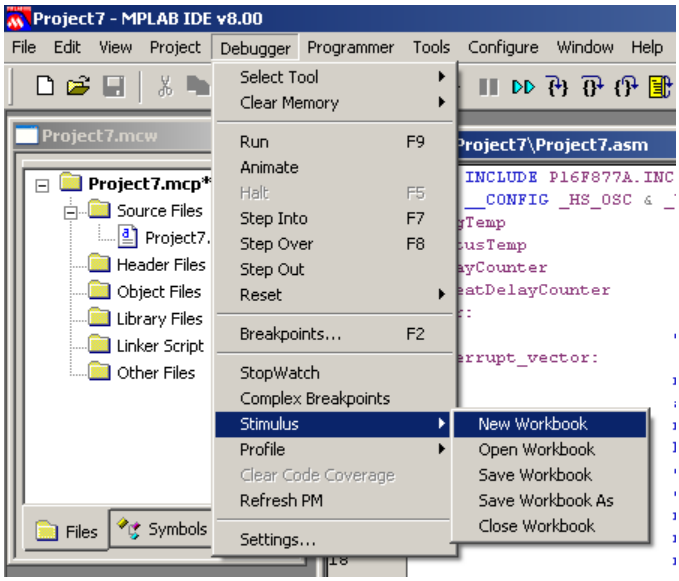

Рис. 4. Создание файла стимулов.

В появившемся окне файла стимулов (рис. 5) нужно выбрать вкладку Asynch. В этой вкладке задаются асинхронные воздействия. Затем в графе PIN/SFR следует выбрать вывод RB0 (рис. 6). На этом выводе будут моделироваться разные уровни сигналов.

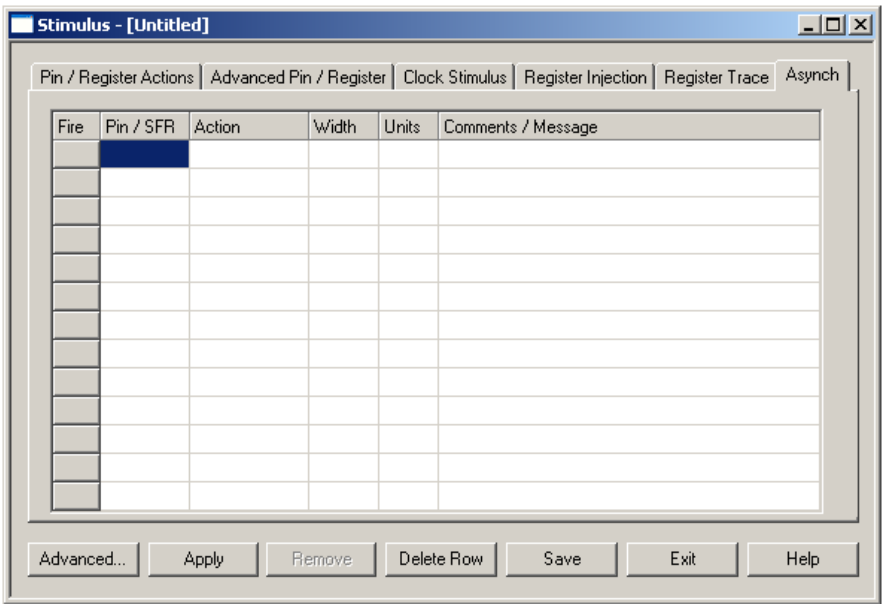

Рис. 5. Настройка стимулов.

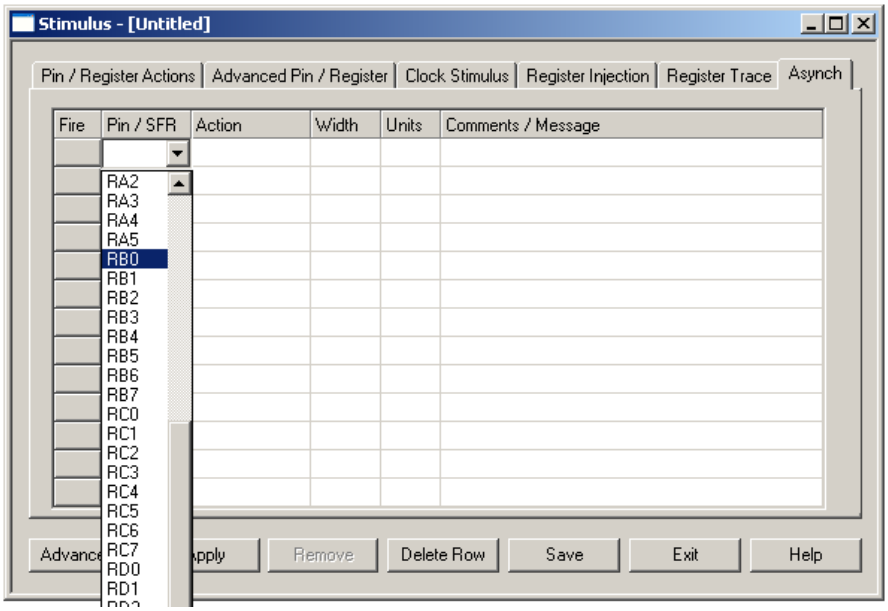

Рис. 6. Выбор вывода, на котором будут моделироваться уровни сигнала.

В графе Action выбираем тип воздействия Toggle – переключатель (рис. 7). При такой модели воздействия кнопка Fire Button (рис. 8) переключает уровень сигнала на выбранном выводе модели микроконтроллера.

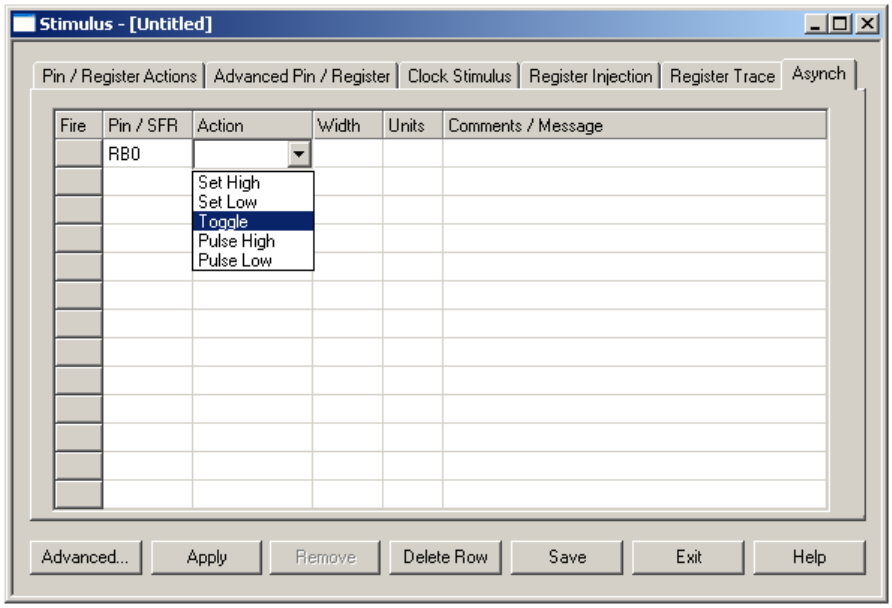

Рис. 7. Выбор модели воздействия.

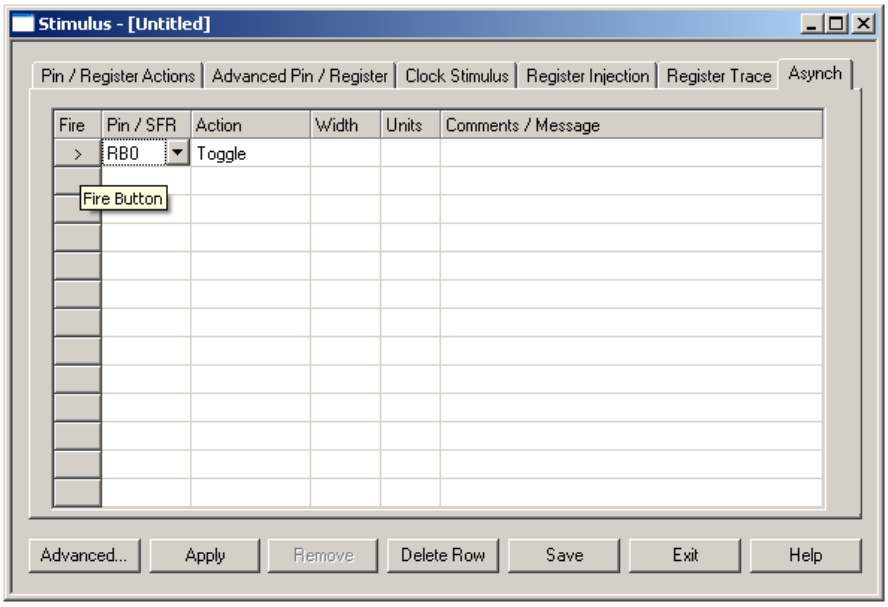

Рис. 8. Кнопка включения воздействия.

Щелчок мышью на прямоугольник Fire Button (рис. 8) «нажмёт кнопку» (переключит уровень). Для того чтобы «отпустить кнопку» надо ещё раз нажать Fire Button. После того, как вы нажали Fire Button, в окне Output появится соответствующее сообщение (рис. 9).

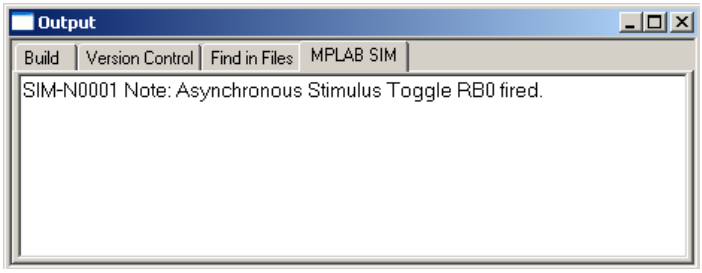

Рис. 9. Сообщение симулятора MPLAB SIM о применении стимула.

За ходом выполнения программы будем наблюдать в окне Watch. Для открытия окна выберем в пункте меню View/Watch (рис. 10). На экране появится окно Watch. Настроим окно Watch для отображения содержимого регистров PORTB и PORTD. Для этого в выпадающем списке выбираем PORTB (рис.11) и нажимаем кнопку Add SFR. Выбранный регистр появляется в окне (рис.12). Аналогичным образом добавляем регистр PORTD (рис. 13).

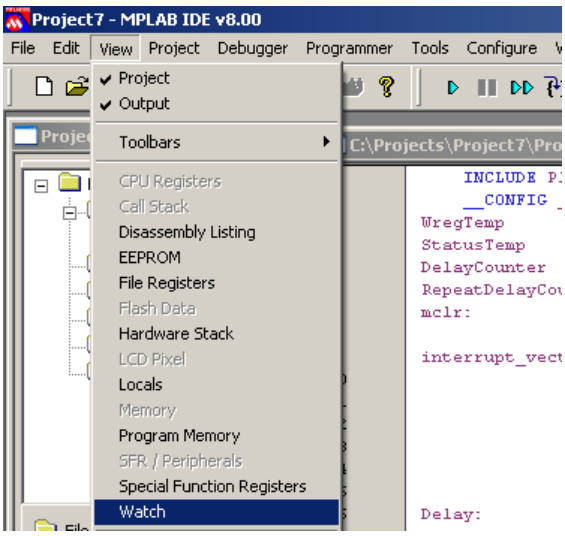

Рис. 10. Открытие окна Watch.

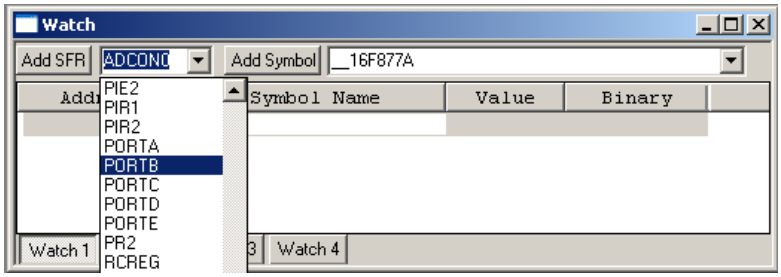

Рис. 11. Выбор регистров.

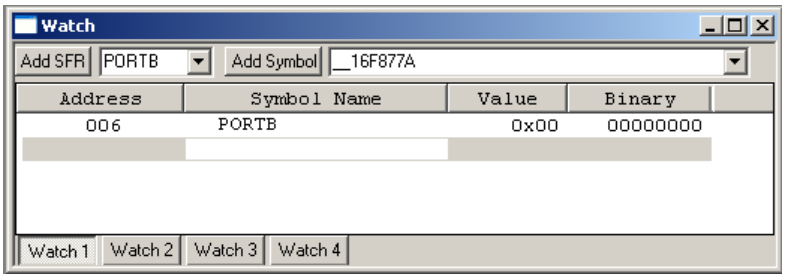

Рис. 12. Окно Watch с изображением содержимого регистра PORTB.

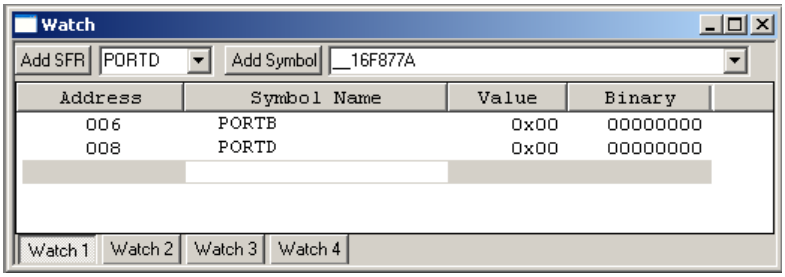

Рис. 13. Вид окна Watch с содержимым регистров.

Теперь регистр PORTB содержит информацию о том, нажата «виртуальная кнопка» или отпущена. Регистр PORTD показывает состояние «виртуального диода». Если PORTD,0 установлен в единицу (RD0=1), то «виртуальный диод» горит. Если PORTD,0 сброшен в ноль (RD0=0), то «виртуальный диод» не горит. После нажатия Fire Button (рис. 8) реакцию нужно наблюдать в окне Watch (рис. 13).

Запустим программу в симуляторе. Для этого нужно нажать кнопку Run (рис. 14). Нажмём несколько раз на Fire Button (рис. 8) и посмотрим изменения в окне Watch.

Рис. 14. Запуск программы.

Остановка симуляции программы происходит при нажатии кнопки Halt (рис. 15).

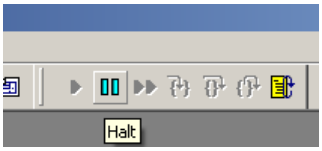

Рис. 15. Кнопка Halt.

Для исследования работы механизма прерываний нужно просматривать содержимое стека, памяти программ. А также счётчика команд PCL и портов PORTD, PORTB в окне Watch. Откроем окно стека, для чего в меню View выбираем Hardware Stack (рис. 16). Содержимое аппаратного стека будем наблюдать в открывшемся окне (рис. 17).

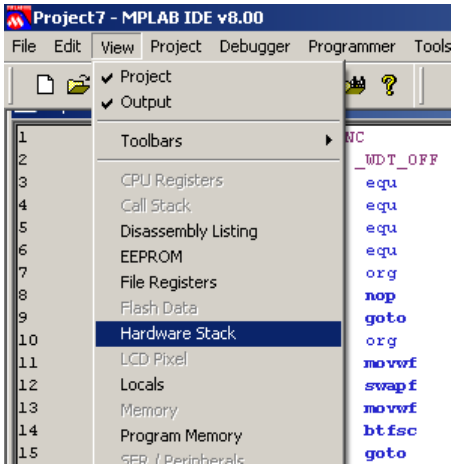

Рис. 16. Открытие окна Hardware Stack

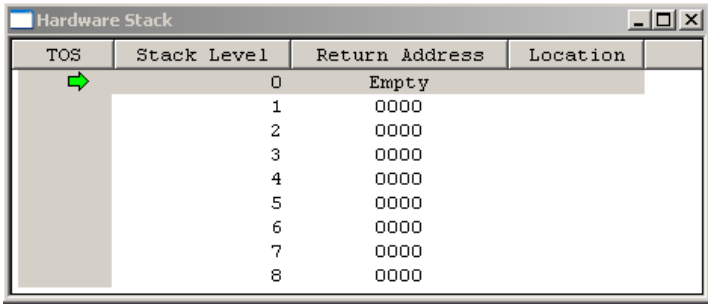

Рис. 17. Вид окна Hardware Stack.

Поставим точку останова на первой команде после директивы org 0x04 (рис. 18). Чтобы создать точку останова нужно навести курсор мыши на строку с командой, в которой предполагается приостановка, и два раза щёлкнуть левой кнопкой. После этого появится красный кружок с буквой «B» внутри. Вторую точку останова поставим напротив команды RETFIE. Обработчик прерывания окажется внутри точек останова. Теперь нужно добавить регистр PCL в окно Watch и можно будет увидеть адрес входа в прерывание, и адрес выхода из прерывания.

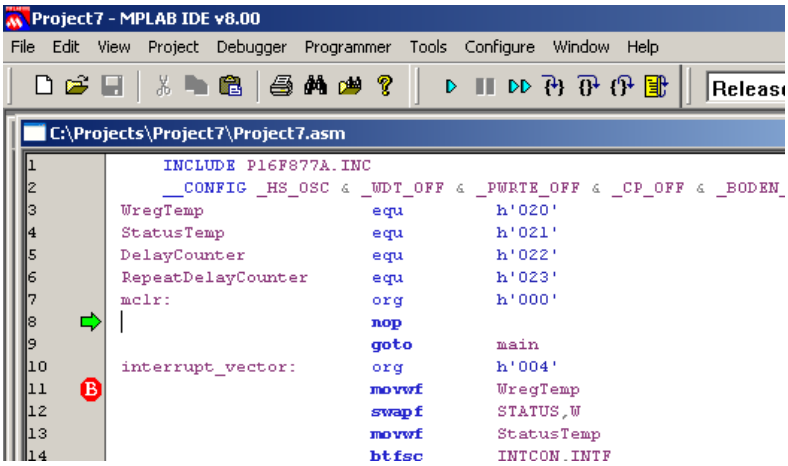

Рис. 18. Точка останова.

Запустим программу (рис. 14), в окне стимулов нажмём Fire Button – это будет соответствовать нажатию «виртуальной кнопки». Если ещё раз нажать Fire Button, то это будет соответствовать отпусканию кнопки. После этого симуляция будет приостановлена и зелёная стрелка окажется на точке останова (рис. 19).

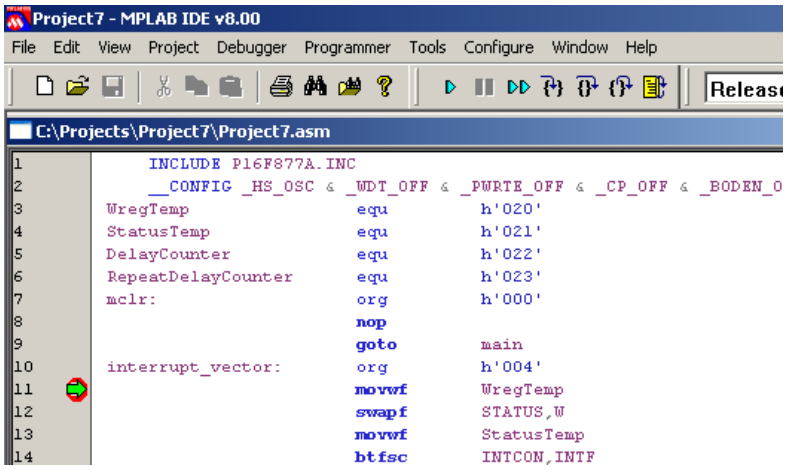

Рис. 19. Приостановка симуляции на точке останова.

После того, как симуляция остановилась, проанализируйте содержимое окна Hardware Stack. Откройте окно Program Memory, для чего в меню View выберите Program Memory (рис. 20). В открывшемся (рис. 21) окне откройте адрес, который указан первым в окне Hardware Stack. Это адрес возврата из прерывания, который хранится в стеке. Запустите программу (рис. 14), выполнение приостановится на второй точке, снова проанализируйте Hardware Stack и Program Memory. Опять запустите программу и убедитесь, что происходит выполнение бесконечного цикла Loop.

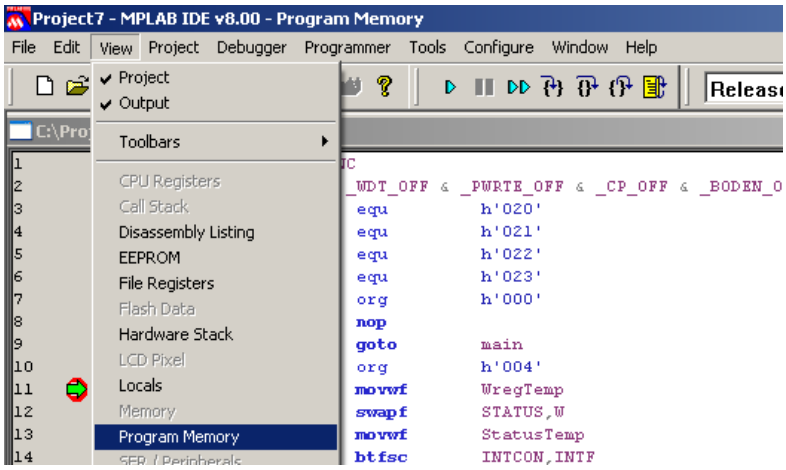

Рис. 20. Открытие окна Program Memory.

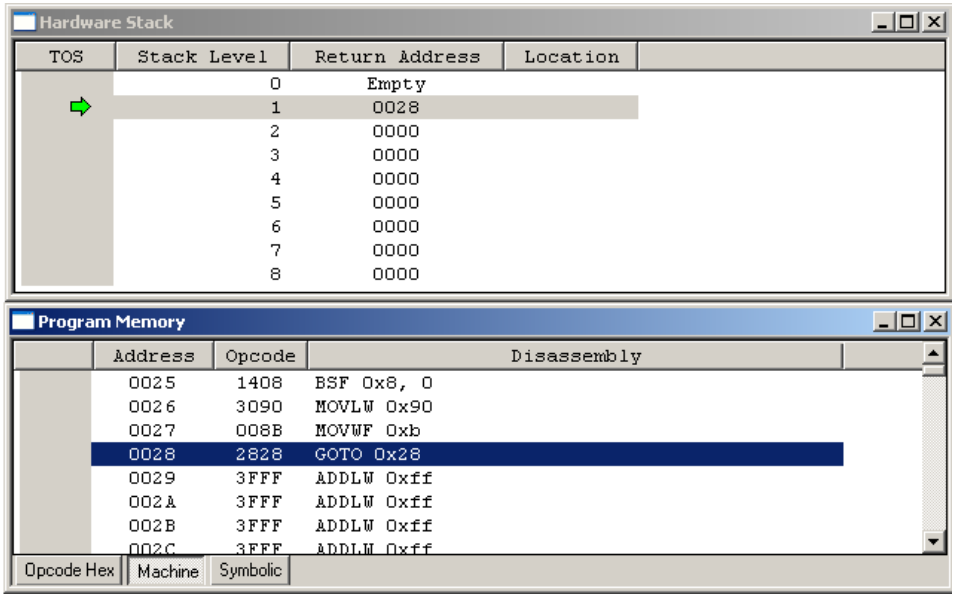

Рис. 21. Окно Program Memory с адресом из Hardware Stack.

Для удаления всех точек останова нужно остановить выполнение программы, щёлкнуть правой кнопкой мыши на тексте программы и выбрать Breakpoints\Remove All Breakpoints (Рис. 22)

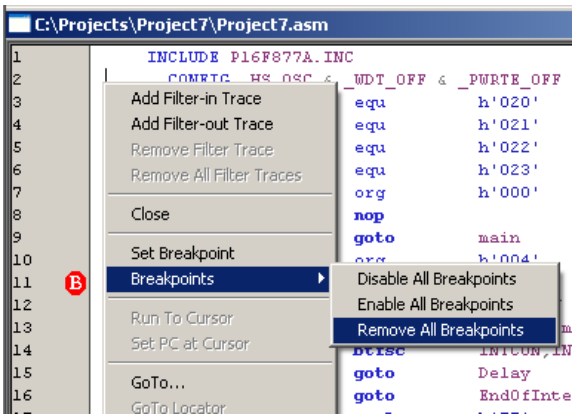

Рис. 22. Удаление точек останова.

Если симуляция прошла успешно соберите схему рис. 26 на лабораторном макете как показано на рис. 27. После этого в среде разработки выберите имеющийся у вас отладчик, например PICkit 2 (рис. 23).

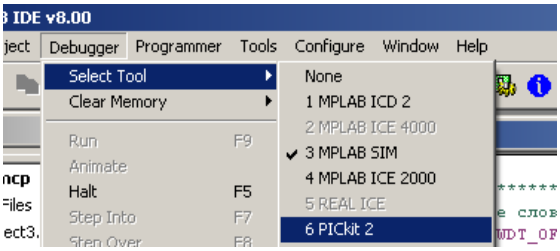

Рис. 23. Выбор отладчика в среде разработки.

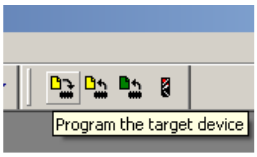

Рис. 24. Кнопка для программирования лабораторного макета.

Запрограммируйте лабораторный макет (рис. 24). Запустите программу в режиме RUN (рис. 14). Продемонстрируйте работу программы.

Аппаратное обеспечение

Эта работа выполняется как на компьютере, в среде разработки MPLAB IDE, так и на лабораторном макете. Принципиальная электрическая схема блока кнопок лабораторного макета изображена на рис. 25. Все кнопки присоединены к подтягивающим резисторам. Если кнопка отпущена, то на соответствующем выходе разъёма XS 1.1 высокий уровень сигнала (+5 Вольт). Если кнопка нажата, то соответствующий вывод разъёма соединяется с нулевым потенциалом.

Принципиальная электрическая схема для выполнения лабораторной работы изображена на рис. 26. Для работы на макете нужно соединить светодиод с выводом PORTD,0, а кнопку с выводом PORTB,0 как показано на рис. 27.

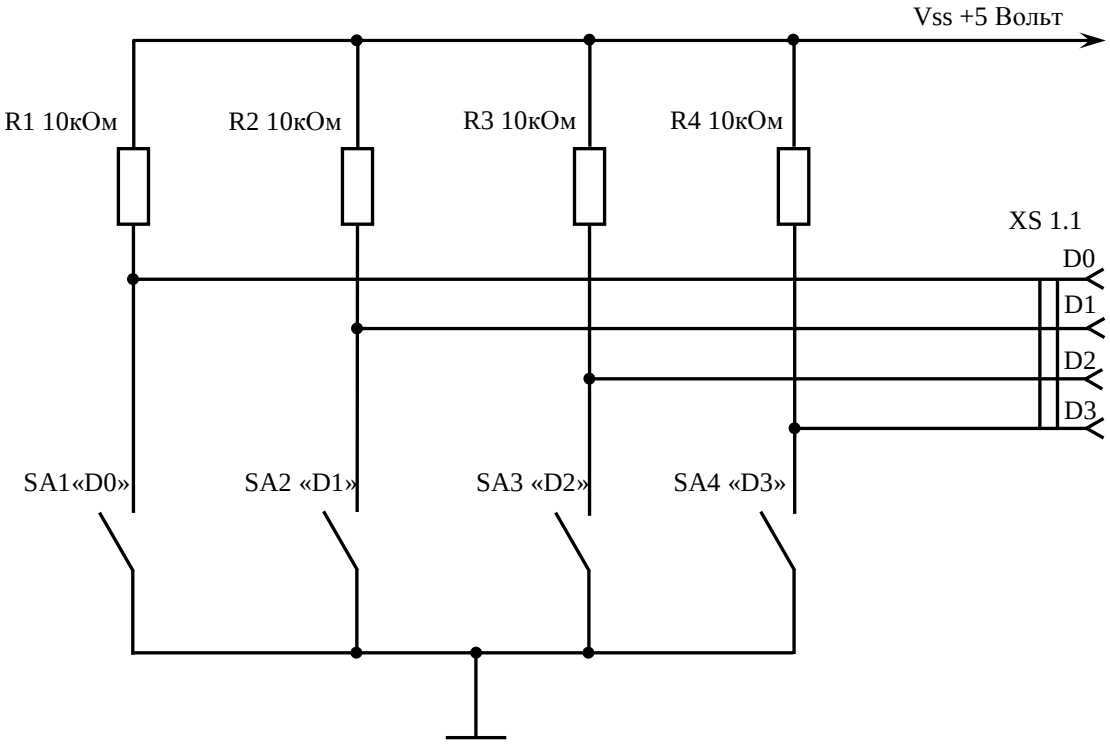

Рис. 25. Схема электрическая принципиальная блока кнопок макета.

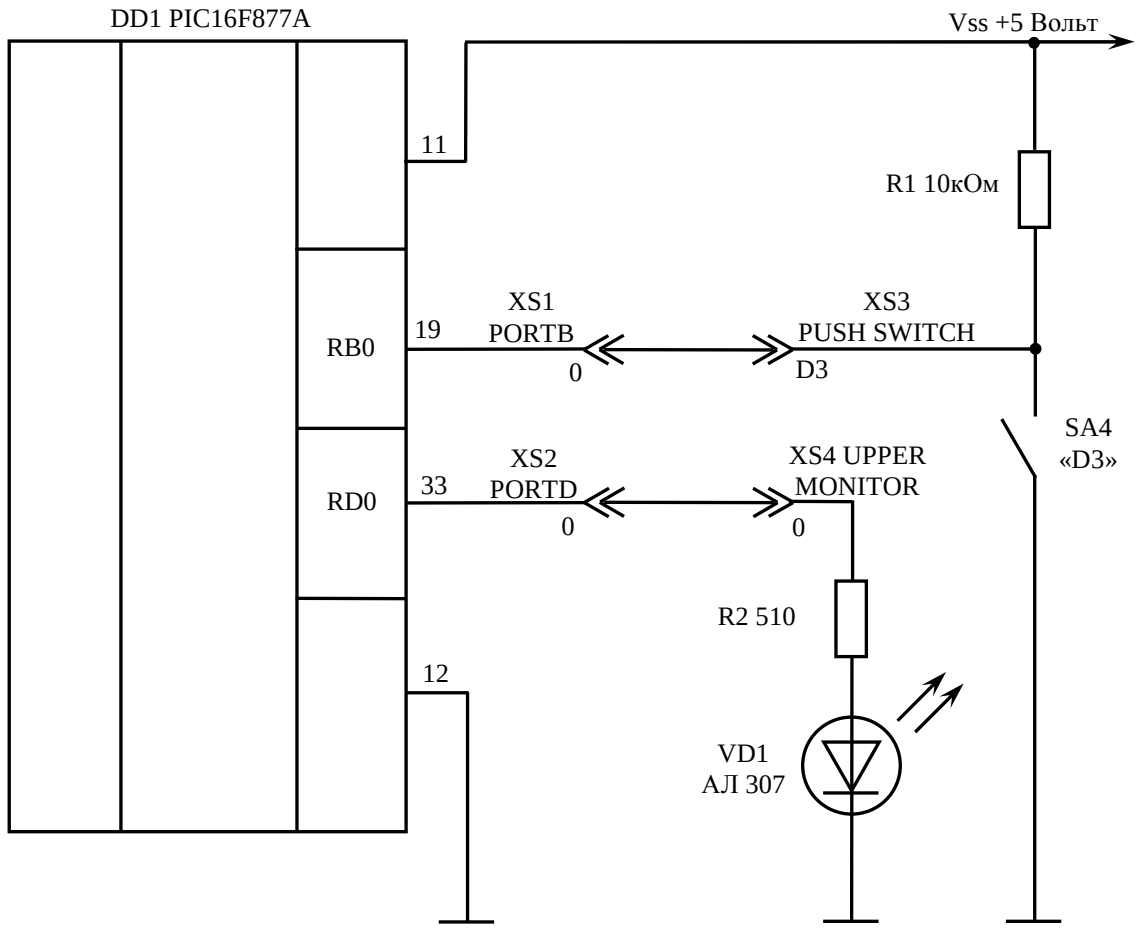

Рис. 26. Принципиальная схема для выполнения лабораторной работы.

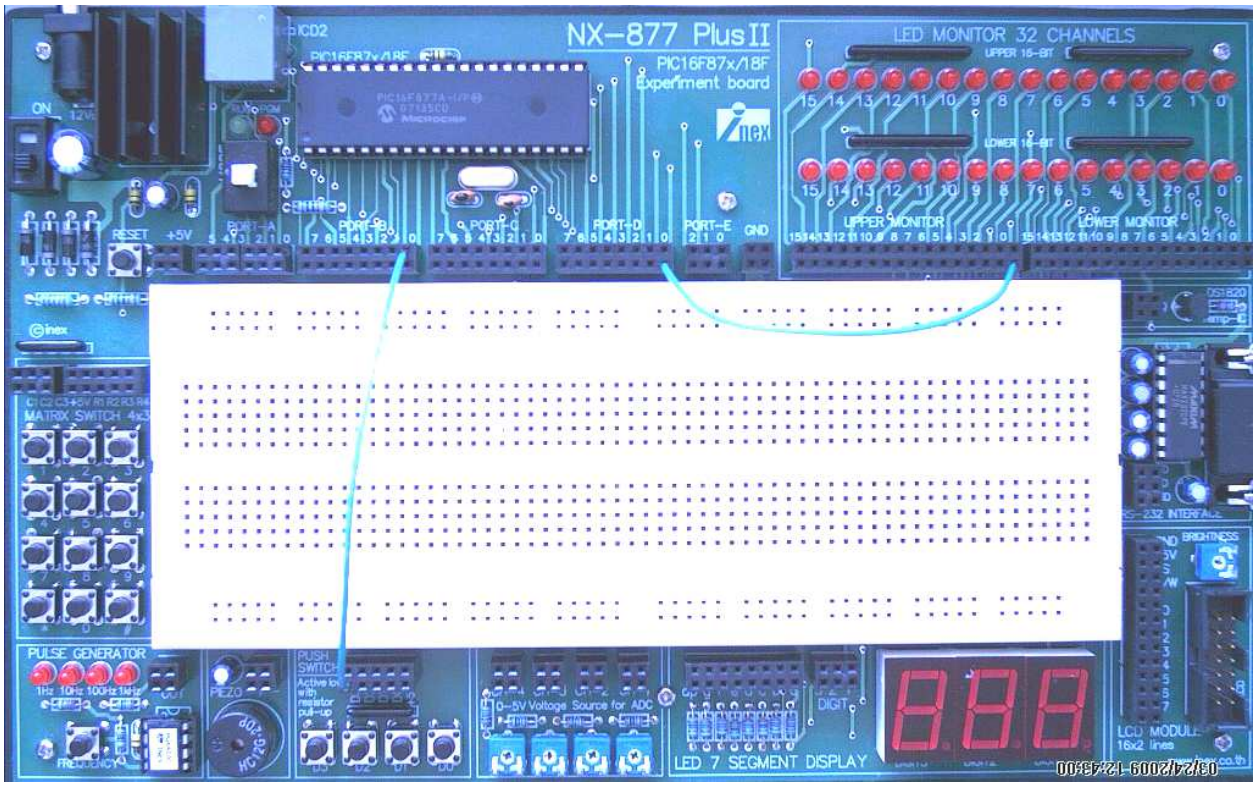

Рис. 27. Схема, собранная на лабораторном макете.

Алгоритм программы изображён на рисунках 28 а) и 28 б).

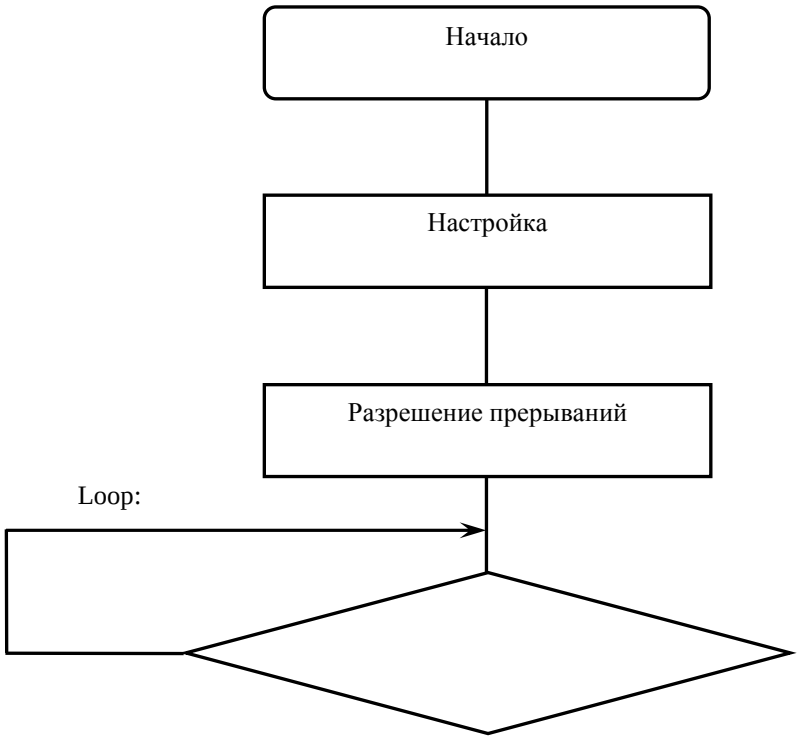

Рис. 28 а) алгоритм основной программы.

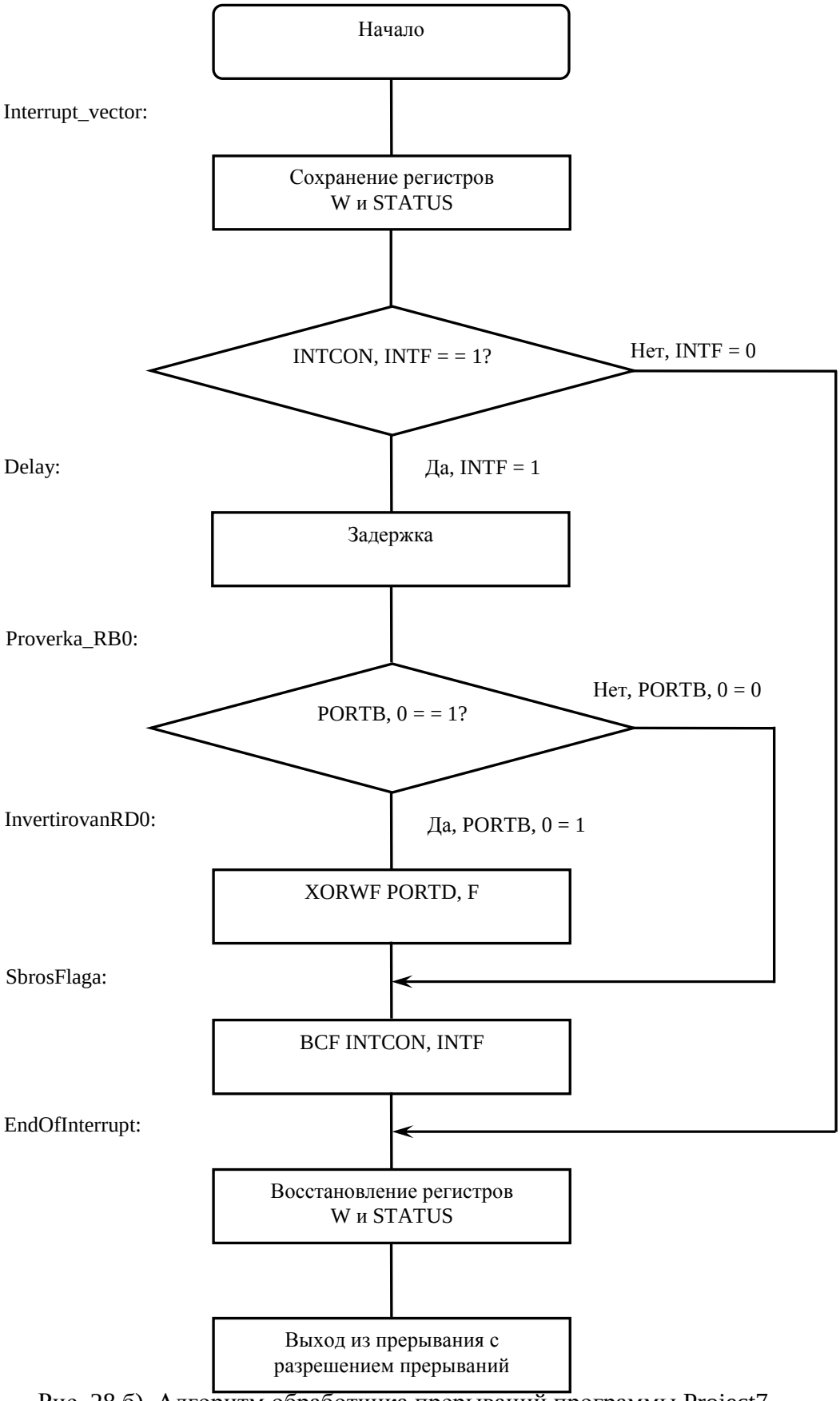

Рис. 28 б). Алгоритм обработчика прерываний программы Project7.

Текст файла Project7.ASM

```
                INCLUDE P16F877A.INC
                __CONFIG _HS_OSC & _WDT_OFF & _PWRTE_OFF & _CP_OFF &
_BODEN_OFF & _LVP_OFF & _CPD_OFF & _DEBUG_ON
WregTemp        equ     h'020'
StatusTemp      equ     h'021'
DelayCounter    equ     h'022'
RepeatDelayCounter equ  h'023'
                nop
                goto    main
Interrupt_vector: org   h'004'
                movwf       WregTemp
                swapf       STATUS,W
                movwf       StatusTemp
                btfsc   INTCON,INTF
                goto    Delay
                goto    EndOfInterrupt
Delay:          movlw   h'FF'
                movwf   DelayCounter
                movlw   h'FF'
                movwf   RepeatDelayCounter
RestartDelay:   decfsz  DelayCounter,F
                goto    RestartDelay
RestartDelay1:  decfsz  RepeatDelayCounter,F
                goto    RestartDelay1
ProverkaRB0:    btfsc   PORTB,0
                goto    SbrosFlaga; Сброс флага
InvertirovanRD0: movlw  b'00000001'
                xorwf   PORTD,F
SbrosFlaga:     bcf     INTCON,INTF
EndOfInterrupt: swapf   StatusTemp,W
                movwf   STATUS
                swapf   WregTemp,F
                swapf   WregTemp,W
                retfie
main:           clrf    PORTB
                clrf    PORTD
                bsf     STATUS,RP0
                movlw   b'00000000'
                movwf   TRISD
                movwf   OPTION_REG
                movlw   b'00000001'
                movwf   TRISB
                bcf     STATUS,RP0
                bsf     PORTD,0
                movlw   b'10010000'
                movwf   INTCON
Loop:           goto    Loop
                end
```

Индивидуальные задания

Присоедините к PORTD комбинацию из двух светодиодов. Один светодиод должен переключаться по нажатию кнопки, а второй должен гореть всё время.

Присоедините к PORTD комбинацию из четырёх светодиодов. Отобразите на светодиодном индикаторе количество нажатий кнопки (от 0000 до 1111).

Присоедините к PORTD семисегментный индикатор. Отобразите на семисегментном индикаторе количество нажатий кнопки (от 0 до F).

При помощи команды CALL организуйте вызов подпрограммы из обработчика прерывания. Подпрограмма должна закончиться командой RETURN. Проанализируйте содержимое стека в момент, когда вызывается подпрограмма; в момент после завершения подпрограммы.

Напишите программу, которая вызовет переполнение стека. Запустите симулятор и проанализируйте возникший сбой в программе.

Контрольные вопросы

1. Расскажите о механизме прерываний.
2. Какова функция регистра INTCON?
3. Какова функция бита GIE?
4. Расскажите, как работает команда RETFIE?
5. Что такое флаг прерывания?
6. Должен ли программист сбросить флаг перед выходом из прерывания?
7. Что такое маска прерываний?
8. Что такое вектор прерывания?
9. Какой адрес у вектора прерывания в микроконтроллерах PIC среднего семейства?
10. Для чего нужен стек?
11. Зачем сохранять регистры W и STATUS?

Оглавление:

Список рисунков: