МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное агентство по образованию

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Физический факультет
Кафедра физико-технической информатики

Яковлев Даниил Дмитриевич

Доклад

Программирование и организация многозадачности

4 курс, группа 16312

Новосибирск, 2019 г.

Содержание

1. Введение

Программирование — процесс создания компьютерных программ. Программирование основывается на использовании языков программирования, на которых записываются исходные тексты программ. Для программирования микроконтроллеров используются языки: Язык ассемблера, C, C++, Wiring, Python, и т.д. Выбор языка обуславливается требованиями к программе. Если нужна скорость, то язык ассемблера, если быстрая разработка, то Wiring с готовыми библиотеками, если требуется большой и производительный продукт, то это обычно C/C++.

2. Структура программы

2.1. Точка начала исполнения программы

Точка входа (англ. Entry Point (EP) — точка входа) — адрес в оперативной памяти, с которого начинается выполнение программы. Другими словами — адрес, по которому хранится первая команда программы. Однако не надо путать её с «первыми командами» программы на языке высокого уровня. Например, программа на C++ начинает выполнение с функции main(), на самом деле, программа в памяти начинается далеко не с первой команды этой функции.

Оригинальной точкой входа называют адрес, с которого начинает выполняться упакованная программа после завершения работы распаковщика. Микроконтроллер начинает исполнение с нулевого адреса (обратим внимание, что в AVR используется Гарвардская архитектура). Как правило там находится инструкция перехода на другой адрес, в котором лежат инструкции нашей программы. Этот адрес и есть точкой входа.

2.2. Процедура инициализации

При включении в оперативной памяти не гарантируется, что все байты равны 0xff. Равно как и регистры не всегда равны нулю при запуске. Это може привести к некорректной работе программы, поэтому необходимо проводить процедуру инициализации.

В разделе инициализации, до инициализации стека, нужно делать зануление памяти и очистку всех регистров. Вариант кода [1]:

```
1  RAM_Flush:
2      LDI  ZL,Low(SRAM_START)
3      LDI  ZH,High(SRAM_START)
4      CLR  R16
5  Flush:
6      ST   Z+,R16
7      CPI  ZH,High(RAMEND+1)
8      BRNE      Flush
9
10     CPI  ZL,Low(RAMEND+1)
11     BRNE      Flush
12
13     CLR  ZL
14     CLR  ZH
```

Листинг 1: Очистка RAM

Поскольку адрес RAM в нашем случае двухбайтный, то вначале смотрим, чтобы старший байт совпал с концом, а потом дочищаем оставшиеся 255 байт в младшем байте адреса. Далее очищаем все регистры от первого до последнего.

```
1          LDI ZL, 30
2          CLR ZH
3          DEC ZL
4          ST  Z,ZH
5          BRNE    PC-2
```

Листинг 2: Очистка регистров

Возможнн и другой вариант, когда значения сразу же инициализируются нужными величинами.

2.3. Основной цикл

Цикл — самая простая организация программы. Отличается минимальным количеством управляющего кода (код который не выполняет полезную работу, а служит лишь для организации правильного порядка действий). Хорошо подхоит для примитивных задач, например «помигать диодом».

```
1  void main(void) {
2  while(1) {
3      Led_ON();
4      Delay(1000);
5      Led_OFF();
6
7      u = KeyScan();
8      switch(u) {
9          case 1: Action1();
10         case 2: Action2();
11         case 3: Action3();
12         }
13     }
14 }
```

Листинг 3: Опрос в цикле

Такой подход допускает только последовательное выполнение кода. И задержка и опрос клавиатуры и внутри цикла. Но есть существенный недостаток — чем больше мы задействуем процессов в нашем коде, тем он будет более неповоротливым и медленным.

3. Прерывания

3.1. Основной цикл и прерывания

Просто основной цикл используется крайне редко, потому как в любом микроконтроллере множество периферии. Постоянный опрос крайне неэффективен при большом числе устройств. Решением является так называемый принцип Голливуд (принцип проектирования) — сигнал должен приходит от периферии, так появляются прерывания.

Прерывание (англ. interrupt) — сигнал от программного или аппаратного обеспечения, сообщающий процессору о наступлении какого-либо события, требующего немедленного внимания. Прерывание извещает процессор о наступлении высокоприоритетного события, требующего прерывания текущего кода, выполняемого процессором. Процессор отвечает приостановкой своей текущей активности, сохраняя свое состояние и выполняя функцию, называемую обработчиком прерывания (или программой обработки прерывания), которая реагирует на событие и обслуживает его, после чего возвращает управление в прерванный код.

- асинхронные, или внешние (аппаратные) — события, которые исходят от внешних аппаратных устройств (например, периферийных устройств) и могут произойти в любой произвольный момент: сигнал от таймера, сетевой карты или дискового накопителя, нажатие клавиш клавиатуры, движение мыши. Факт возникновения в системе такого прерывания трактуется как запрос на прерывание (англ. Interrupt request, IRQ) - устройства сообщают, что они требуют внимания со стороны ОС;

- синхронные, или внутренние — события в самом процессоре как результат нарушения каких-то условий при исполнении машинного кода: деление на ноль или переполнение стека, обращение к недопустимым адресам памяти или недопустимый код операции;

- программные (частный случай внутреннего прерывания) — инициируются исполнением специальной инструкции в коде программы. Программные прерывания, как правило, используются для обращения к функциям встроенного программного обеспечения (firmware), драйверов и операционной системы.

Здесь ситуация становится несколько иной. За счет прерываний у нас появляются параллельные процессы. Например, мы можем повесить опрос клавиатуры и мигание лампочкой на прерывание по таймеру (псевдокод):

```
ISR(Timer1) {
    u=KeyScan();
}

ISR(Timer2) {
    if(LED_ON) {
        Led_OFF();
    } else {
        Led_ON();
    }
}

void main(void) {
    Timer1_Delay=100;
    Timer1=1<<ON;
    Timer2_Delay=1000;
    Timer2=1<<ON;
    sei();
    while(1); // your code here
}
```

Листинг 4: Основной цикл и прерывания

Теперь программа исполняет три независимых процесса. Такая схема работает до тех пор пока хватает аппаратных ресурсов, т.к. на каждый процесс по прерыванию. Но число таймеров ограничено, прерывания перегружать нельзя — они должны быть быстрыми. Эту проблему решают более высокоуровневые сущности — операционные системы и в частности планировщики, принципы которых описываются в следующей главе.

3.2. Вектора прерываний

Векторами прерывания называют адреса перехода на обработчик прерывания. Список таких адресов называется таблицей векторов прерываний, и он находится в памяти программ по заранее известному адресу. У микроконтроллеров AVR таблица векторов прерываний находится в самом начале памяти программ FLASH по адресу 0 [2]. Содержимое таблицы векторов прерываний определяет программист, когда ему нужно реализовать обработку прерываний.

Каждый вектор прерывания AVR занимает в памяти 2 байта (1 слово кода инструкций AVR), и представляет из себя команду rjmp относительный_адрес. Вот так, например, выглядит на языке ассемблера вектор прерывания микроконтроллера ATmega2560:

```
1 .org 0x00
2   jmp Reset
3
4 .org 0x001e
5   jmp TIM2_OVF
```

Листинг 5: Таблица векторов прерывания

3.3. Обработка прерываний

Обработчик прерываний (или процедура обслуживания прерываний) — специальная процедура, вызываемая по прерыванию для выполнения его обработки. Обработчики прерываний могут выполнять множество функций, которые зависят от причины, которая вызвала прерывание.

Как уже было сказано, вектор прерывания делает переход на подпрограмму, которая по сути и является обработчиком прерывания. Пример:

```
1 .org 0x002E
2     rjmp TIM1_OVF
3
4 TIM1_OVF:
5     cli
6     // Your interrupt code here
7     sei
8     reti
```

Листинг 6: Вектор прерывания и обработчик прерывания

4. Многозадачность

Многозадачность (англ. multitasking) — свойство операционной системы или среды программирования обеспечивать возможность параллельной (или псевдопараллельной) обработки нескольких процессов.

Таким образом, нам необходимо ввести еще одно понятие, назовем его поток исполнения — это набор инструкций с определенным порядком следования, которые выполняет процессор во время работы программы. Поскольку речь идет о многозадачности, то естественно в системе может быть несколько этих самых вычислительных потоков. Поток, инструкции которого процессор выполняет в данный момент времени, называется активным. Поскольку на одном процессорном ядре может в один момент времени выполняться только одна инструкция, то активным может быть только один вычислительный поток. Процесс выбора активного вычислительного потока называется планированием (scheduling). В свою очередь, модуль, который отвечает за данный выбор принято называть планировщиком (scheduler).

4.1. Организация многозадачности

Существует много различных методов планирования [3]. Большинство из них можно отнести к двум основным типам:

- невытесняющие (кооперативные) — планировщик не может забрать время у вычислительного потока, пока тот сам его не отдаст.
- вытесняющие — планировщик по истечении кванта времени выбирает следующий активный вычислительный поток, сам вычислительный поток также может отдать предназначенный для него остаток кванта времени.

4.1.1. Невытесняющий планировщик

Представим, что у нас есть несколько задач, достаточно коротких по времени, и мы можем их вызывать поочередно. Задачу оформим как обычную функцию с некоторым набором параметров. Планировщик будет оперировать массивом структур на эти функции. Он будет проходиться по этому массиву и вызывать функции-задачи с заданными параметрами. Функция, выполнив необходимые действия для задачи, вернет управление в основной цикл планировщика.

Рис. 1: Карта процессорного времени вытесняющего планировщика

4.1.2. Вытесняющий планировщик

Теперь давайте представим следующую картину. У нас есть два вычислительных потока, выполняющих одну и ту же программу, и есть планировщик, который в произвольный момент времени перед выполнением любой инструкции может прервать активный поток и активировать другой. Для управления подобными задачами уже недостаточно информации только о функции вызова потока и ее параметрах, как в случае с невытесняющими планировщиками. Как минимум, еще нужно знать адрес текущей выполняемой инструкции и набор локальных переменных для каждой задачи. То есть

для каждой задачи нужно хранить копии соответствующих переменных, а так как локальные переменные для потоков располагаются на его стеке, то должно быть выделено пространство под стек каждого потока, и где-то должен храниться указатель на текущее положение стека.

Эти данные — instruction pointer и stack pointer — хранятся в регистрах процессора. Кроме них для корректной работы необходима и другая информация, содержащаяся в регистрах: флаги состояния, различные регистры общего назначения, в которых содержатся временные переменные, и так далее. Все это называется контекстом процессора.

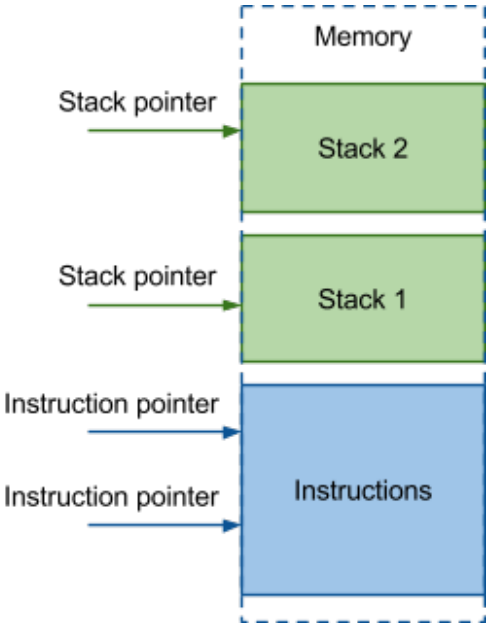

Рис. 2: Организация памяти при двух процессах

Контекст процессора (CPU context) — это структура данных, которая хранит внутреннее состояние регистров процессора. Контекст должен позволять привести процессор в корректное состояние для выполнения вычислительного потока. Процесс замены одного вычислительного потока другим принято называть переключением контекста (context switch).

Понятия контекста процессора и переключения контекста — основополагающие в понимании принципа вытесняющего планирования. Переключение контекста — замена контекста одного потока другим. Планировщик сохраняет текущий контекст и загружает в регистры процессора другой. Выше было сказано, что планировщик может прервать активный поток в любой момент времени, что несколько упрощает модель. На самом же деле не планировщик прерывает поток, а текущая программа прерывается процессором в результате реакции на внешнее событие — аппаратное прерывание — и передает управление планировщику. Например, внешним событием является системный таймер, который отсчитывает квант времени, выделенный для работы активного потока. Если считать, что в системе существует ровно один источник прерывания, системный таймер, то карта процессорного времени будет выглядеть следующим образом:

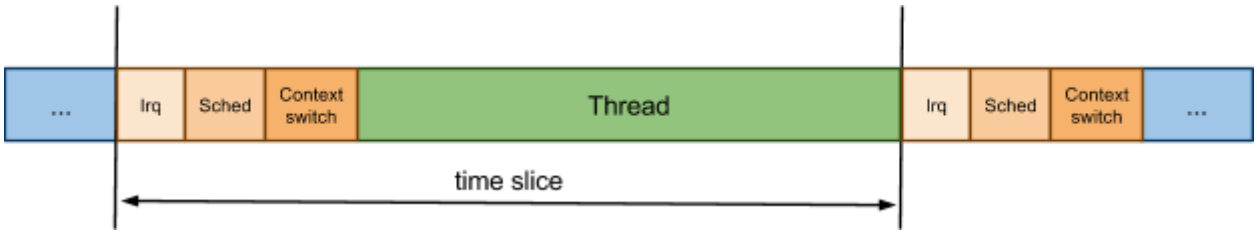

Рис. 3: Карта процессорного времени вытесняющего планировщика

4.2. Watchdog

Сторожевой таймер (англ. watchdog timer букв. «сторожевой пёс») — аппаратно реализованная схема контроля над зависанием системы. Представляет собой таймер, который периодически сбрасывается контролируемой системой. Если сброса не произошло в течение некоторого интервала времени, происходит принудительная перезагрузка системы. В некоторых случаях сторожевой таймер может посылать системе сигнал на

перезагрузку («мягкая» перезагрузка), в других же — перезагрузка происходит аппаратно (замыканием сигнального провода RST или подобного ему) [4].

Для работы со сторожевым таймером для начала необходимо подключить библиотеку "wdt.h"в которой описаны все основные команды для работы со сторожевым таймером. Ниже пример программы, которая будет включать сторожевой таймер, который в свою очередь будет сбрасывать микроконтроллер каждые 4 секунды.

Команды для работы со сторожевым таймером: include <avr/wdt.h> - подключение библиотеки для работы со сторожевым таймером

- wdt_reset() - сброс сторожевого таймера,
- wdt_enable() - включение сторожевого таймера,
- wdt_disable() - выключение сторожевого таймера.

```
1 #include <avr/wdt.h>
2
3 int main(void) {
4     wdt_enable(WDTO_4S);
5     while(1);
6 }
```

Листинг 7: Программа с watchdog

Список литературы

[1] “Avr. Учебный курс. Стартовая инициализация.” http://easyelectronics.ru/avr-uchebnyj-kurs-vazhnye-melochi-1.html, 2019.

[2] “Atmega640/1280/1281/2560/2561 - summary datasheet.” https://www.microchip.com/wwwproducts/en/ATmega2560, 2014.

[3] “Организация многозадачности в ядре ОС.” https://habr.com/ru/company/embox/blog/219431/, 2015.

[4] “Watchdog timer.” https://en.wikipedia.org/wiki/Watchdog_timer, 2019.