Занятие 3: Занятие 3: Ассемблер для ARM

Теоретическая часть

Знакомство с языком программирования ассемблер для ARM.

Длина одной команды ассемблера для ARM- 4 байта или 32 бита. Микроконтроллер ARM относится к RISC процессорам, что означает малый набор инструкций. Однако количество возможностей модификации инструкций впечатляет. Начнём с перечисления часто использующихся математических и других основных возможностей ARM.

Команда	Значение
ADD	Сложить
SUB	Вычесть
MUL	Умножить
AND	Побитовое "И"
EOR	Побитовое "Исключающее ИЛИ"
MVN	Побитовое "НЕ"
ORR	Побитовое "ИЛИ"
B	Переход выполнения программы в другое место в памяти (branching)
MOV	Запись значения одного регистра в другой
LDR	Чтение из памяти в регистр
STR	Запись из регистра в память
TST	Установка флагов согласно логическому "И"
CMP	Установка флагов согласно вычитанию
SWP	Перестановка начальной и конечной части регистра

Эти команды допускает важные модификации, такие как:

- Выполнение в зависимости от значение флагов
- Установка или неустановка флагов в результате операции
- Знаковая или беззнаковая форма операции
- Дополнительная модификация операнда в процессе его использования

В целом язык ассемблера для ARM очень гибкий. Писать неоптимизированные программы на нём достаточно просто, а при достаточном умении он хорошо оптимизируется.

Разберём затронутые выше понятия, такие как регистры, флаги и branching.

У ARM есть 16 регистров, называющиеся R0-R15. Следует их отличать от специальных регистров, управляющих периферией. Ядро процессора выполняет операции только с этими регистрами. Для сложения двух чисел в памяти придётся загрузить каждое из них в свой регистр, сложить их и поместить результат обратно в память. Часть регистров отведены под специальные нужды. Это R15, называемый также PC. Он всегда равен адресу выполняемой в данный момент команды. R14, называемый также LR. Это регистр, хранящий адрес того места, откуда был совершён вызов выполняемой функции. Иными словами это адрес возврата. Без него невозможно вернуться после выполнения функции. R13, называемый также SP. Это адрес текущего значения стека. Без него стек не будет работать.

Флаги это 4 бита C (Carry), N (Negative), Z (Zero), V (Overflow), которые устанавливаются процессором после выполнения логических или математических операций. C означает перенос, происходящий при битовых сдвигах, а также при возникновении переноса самого старшего бита. V означает, что результат операции не уместился в регистре результата, Z означает, что результат операции нулевой, N означает отрицательный результат операции. Флаги позволяют проверить какое либо условие с помощью устанавливающей флаги команды, а затем перейти к условному выполнению.

Branching или переход выполнения позволяет вызывать функции. Для того чтобы вызвать функцию надо загрузить адрес возврата в LR регистр, а затем перенести выполнение на адрес функции. В конце функции должен быть переход выполнения по адресу возврата. Именно с помощью перехода выполнения можно делать циклы, где в конце цикла стоит переход выполнения на его начало.

Необходимые адреса для перехода выполнения или для считывания данных считываются из меток. Чтобы поставить метку нужно с самого начала строки написать её имя. Ничто кроме меток не стоит писать с самого начала строки. Команды и директивы компилятору пишутся после табуляции.

Данные вводятся после команды ассемблера DCD или DCW. Первая означает, что данные должны занимать 4 байта, а вторая - 2 байта.

Считывание и запись в специальные регистры, как и просто в память реализуются через команды STR и LDR. Сначала нужно загрузить адрес специального регистра в регистр процессора: *LDR R0, DataLabel*, где DataLabel это метка, содержащая *DCD 0x????????*. Теперь можно загрузить информацию из специального регистра с помощью *LDR R1,[R0]*. Информация по адресу, хранящемуся в R0 попадёт в R1. Можно изменить значение R1, например умножив на 2: *ADD R1,R1* и записать его назад *STR R1,[R0]*. Возможно указывать дополнительное смещение к адресу в регистре. Например если данные хранятся порциями по 4 байта, то можно считать следующее значение в регистр R2 вот так: *LDR R2,[R0,#04]*. Напомню, что в R0 находится адрес DataLabel.

Среда программирования Keil обладает поддержкой ассемблерных вставок, когда в любом месте кода можно выполнить набор ассемблерных команд. Однако эта поддержка неполная. Поэтому проще записать необходимые процессорные команды в отдельной функции, написанной в ассемблерном файле. Для этого нужно:

- Создать файл с расширением *.s
- Добавить этот файл к проекту
- В этом файле написать объявление зоны кода начиная с табуляции: AREA ZoneNameHere, CODE, READONLY
- Придумать имя функции и поставить метку с этим именем с самого начала следующей строки
- Написать в столбец всю функцию добавляя в неё метки при необходимости
- Написать в конце директиву: EXPORT ИмяФункции
- Последней строчкой написать: END
- Добавить в C файл функцию с помощью объявления: extern void ИмяФункции(void);

Теперь при вызове функции выполнение начнётся непосредственно с первой после метки команды ассемблера. Если функция подразумевает возврат, то для возврата нужно совершить переход выполнения по адресу из регистра LR.

Условное выполнение и циклы.

Практически каждая команда процессора может быть выполнена в зависимости от состояния флагов. Также в команде может содержаться информация устанавливать ли флаги по результатам своей операции. Общий синтаксис команды это <команда><CC><S>. CC (Conditional Execution) означает выполнять если флаги соответствуют требуемым. Возможные значения этого поля:

CC	Значение	CC	Значение
CS	Перенос произошёл	CC	Переноса не произошло
EQ	Равенство	NE	Не равно
VS	Произошло переполнение	VC	Не произошло переполнения
GT	Больше	LT	Меньше
GE	Больше или равно	LE	Меньше или равно

PL	Положительное	MI	Отрицательное
HI	Выше	LO	Ниже
HS	Выше или так же	LS	Ниже или так же

Букву S подисываю если флаги нужно поменять и не дописывают если не нужно. Пример команд: *MOVNE, SUBCSS, BLO, ADDEQS*.

Циклы реализуются с помощью условного выполнения операции перехода выполнения. Например как 256 раз выполнить набор команд:

```
Lim
        DCD   0x00000100
        DCD   0x00000001
Begin
        LDR   R0,Lim              ; Область данных
        LDR   R1,[R0]                     ; начальное значение
        LDR   R2,[R0,#04]                 ; шаг вычитания
Label1
        ;------- Тут команды в цикле -------
        SUBS  R1,R2                       ; уменьшить значение на шаг и установить
                                          ; флаги
        BEQ   Label1
```

Практическая часть

Повтор программы ввода-вывода с использованием ассемблерных вставок.

Требуется написать ассемблерную функцию, реализующую вечный цикл и доступ к специальным регистрам, с помощью которой выполняется предыдущее задание::

- Пока кнопка нажата горит светодиод
- Светодиод загорается только после нажатия кнопки
- Светодиод изначально горит, при изменении состояния кнопки светодиод гаснет
- Светодиод мигает всё медленнее, кнопка ускоряет мигание
- После нажатия кнопки светодиод загорается и гаснет
- Светодиод мигает столько раз, сколько была нажата кнопка