# Ссылки:

<https://habr.com/ru/post/133808/>

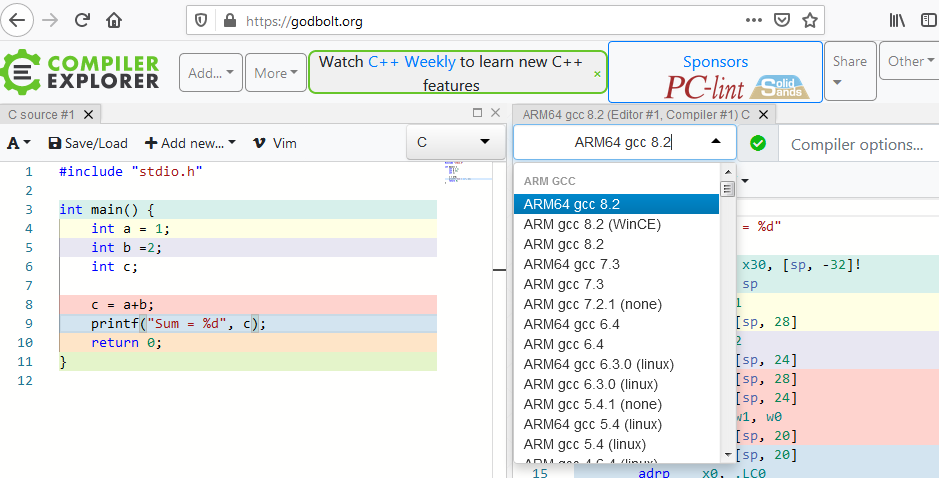
<https://habr.com/ru/post/188712/>

<https://marsohod.org/index.php/prodmarsohod2/amber-arm-soc/226-arm-instr>

[http://www.keil.com](http://www.keil.com/)(много материала по ARM, в том числе поиск по командам – на главной странице справа)

# Получение листинга

1. На сайте godbolt.org следует выбрать ARM gcc или ARM64 gcc в списке компиляторов.



1. В домашней директории пользователя evmpu на сервере лежит текстовый файл README-ARM.txt, в котором записаны команды для компиляции программы на С/C++ до ассемблерного кода ARM (arm-linux-gnueabi-g++ -mfloat-abi=hard -S myprog.cpp –o myprog.s)

# Условное выполнение

Одна из важнейших функций ARM — **условное выполнение**. Каждая инструкция может исполняться условно и для этого используются суффиксы. Если суффикс добавляется к названию инструкции, то прежде чем выполнить ее, происходит проверка параметров. Если параметры не соответствуют условию, то инструкция не выполняется.

Всего в процессоре 4 флага условия:  
•    Negative – результат операции получился отрицательным,   
•    Zero – результат равен нулю,   
•    Carry – при выполнении операции с беззнаковыми чисоами произошел перенос,   
•    oVerflow – при выполнении операции со знаковыми числами произошло переполнение, результат не помещается в регистр

Эти 4 флага формируют множество  возможных комбинаций условия:

|  |  |  |
| --- | --- | --- |
| Суффикс | Значение | Флаги |
| eq | Equal | Z=1 |
| ne | Not equal | Z=0 |
| cs / hs | Carry set / unsigned higher or same | C=1 |
| cc / lo | Carry clear / unsigned lower | C=0 |
| mi | Minus / negative | N=1 |
| pl | Plus / positive or zero | N=0 |
| vs | Overflow | V=1 |
| vc | No overflow | V=0 |
| hi | Unsigned higher | C=1and Z=0 |
| ls | Unsigned lower or same | C=0 or Z=1 |
| ge | Signed greater than or equal | N =V |
| lt | Signed less than | N != V |
| gt | Signed greater than | Z =0,N =V |
| le | Signed less than or equal | Z =1 or N != V |
| al | Always (unconditional) |  |
| - | Invalid condition |  |

Разберем для примера почему суффикс «eq» соответствует установленному флагу Zero. Дело в том, что при сравнении двух чисел *a* и *b* в процессоре, на самом деле, происходит сравнение с 0 разности *a-b.* Если *a-b*=0, то устанавливается флаг Zero.

Пример условной операции: сложение при условии, что флаг Z установлен – это команда addeq как add + суффикс eq. Переход на подпрограмму в случае, если флаг N=0 – это blpl как bl + суффикс pl.  
  
Флаги { Negative, Zero, Carry, oVerflow }  устанавливаются то же не всегда при арифметических или логических операциях, как это бывает скажем в x86 процессоре, а только, когда захочет программист. Для этого есть еще один суффикс к мнемонике команд: «s». Таким образом, команда сложения add не меняет флагов, а команда adds меняет флаги. А может еще быть условная команда сложения, но которая меняет флаги. Например: addgts.

# Регистры

16 32-битных регистров общего назначения (r0 – r15) и 32-битный регистр состояния (CPSR). В скобках указаны альтернативные имена:

* 1. r0..r3 (a1..a4) – используются, в частности, для хранения аргументов функций
  2. r0..r1 – могут использоваться для возвращаемых значений
  3. r4..r11 (v1..v8) – для переменных
  4. r13 (sp) - указатель вершины стека (stack pointer - sp)
  5. r14 (lr) – link pointer - используется для сохранения адреса возврата в результате исполнения специальной команды условного перехода
  6. r15 (pc) – program counter – адрес текущей исполняемой инструкции
  7. CPSR – current program status register) хранит однобитовые флаги

# Инструкции

Одна из основных особенностей ARM-ассемблера заключается в том, что **все операции производятся над регистрами и константами** (в отличие от ассемблера x86, где операндом мог быть и участок в памяти). Над памятью возможны только 2 операции: запись в память (str) и считывание из памяти (ldr).

Еще одна особенность – многие инструкции, имевшие в x86 2 операнда, здесь имеют 3. Например, add, sub и др. Операнд-назначение идет здесь первым, что тоже не характерно для ассемблера AT&T.

Константы обозначаются символом # (вспомните как в x86). Обращение в память – квадратными скобками [] (вспомните как в x86).

|  |  |  |
| --- | --- | --- |
| Название | Синтаксис | Применение |
| ADD (сложение) | ADD r0, r1, r2 | r0 = r1 + r2 |
| SUB (вычитание) | SUB r0, r1, r2 | r0 = r1 — r2 |
| RSB (обратное вычитание) | RSB r0, r1, #10 | r0 = 10 — r1 |
| MUL (умножение) | MUL r0, r1, r2 | r0 = r1 \* r2 |
| MOV | MOV r0, r1 | r0 = r1 |
| ORR( логическая операция) | ORR r0, r1, r2 | r0 = r1 | r2 |
| TEQ | TEQ r0, r1 | r0 == r1 |
| LDR (загрузка, чтение - load) | LDR r4, [r5] | r4 = \*r5 |
| STR (запись в память - store) | STR r4, [r5] | \*r5 = r4 |
| ADR | ADR r3, a | a — переменная. r3 = &a |

# Переходы

Принцип формирования инструкций перехода аналогичен архитектуре x86. Только на x86 они начинаются на букву «j» (jmp – безусловный, jne – “**j**ump if **n**ot **e**qual” и др.), а в ARM-е на «b» от слова «branch» (b – безусловный, bne, beq и др.)

# **ЗАДАНИЕ К ЛАБОРАТОРНОЙ №4**

Задание аналогично лабораторной работе №3 – получить ассемблерные листинги своих программ на уровнях оптимизации O0 и O2 для архитектуры ARM. Программы должны быть взяты из лабораторной №1 без каких-либо изменений.