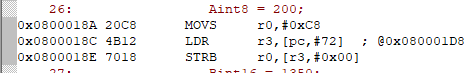
**Отчет №2**

**Часть первая:**

*4.1. Как выглядит типовая последовательность ассемблерных команд, соответствующей строке языка С, в которой происходит присвоение значения переменной? Опишите смысл каждой инструкции в такой последовательности.*



Команда 20C8 MOVS: по адресу 0x800018A хранится: положить в регистр R0 число С8

Команда 4B12 LDR: по адресу 0x800018С хранится: загрузить в регистр r3 значение из ячейки памяти [pc+72].

@0x080001D8 – адрес, по которому находится адрес переменной Aint8

#72 - смещение

pc – program counter

Команда 7018 STRB: по адресу 0x0800018E хранится: сохранить значение регистра r0 в память по адресу, который хранится в r3 со смещением 0. То есть поместить значение переменной Aint8 в память.

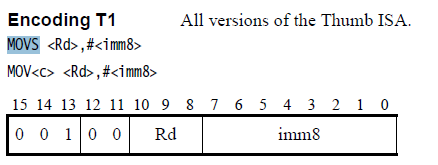
Постфикс B – запись 1-го байта.

Также, используются: H – для полуслова размером 2 байта, D для double, W для слова размером 4 байта, S для знаковых.

*4.2. Найдите в дизассемблере короткую команду с непосредственной адресацией. Разберите двоичное представление этой команды, найдите в ее теле непосредственно задаваемый операнд.*

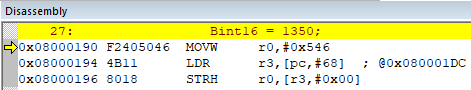


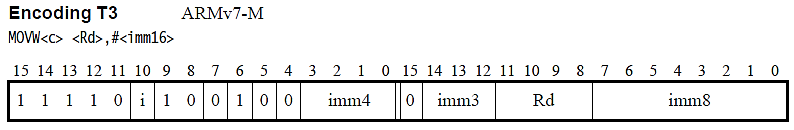
2205: во второй регистр помещается число 5.



|  |  |
| --- | --- |
| **2205 =**  **movs** |  |

*4.3. Найдите длинную команду с непосредственной адресацией, с помощью которой в регистр помещается число Б. Разберите двоичное представление этой команды, найдите в ее теле непосредственно задаваемый операнд.*





**1111 0 0 100100 0000 0 101 0000 0100 0110**

**1111 0 i 100100 imm4 0 imm3 Rd imm8**

**Opcode –** обозначение длинной команды

**Rd –** регистр-приемник результата

**Imm12** – число, собираемое из imm4:i:imm3:imm8 – 16 бит

*4.4. Найдите длинную команду с непосредственной адресацией, с помощью которой в регистр помещается число В. Разберите двоичное представление этой команды, найдите в ее теле непосредственно задаваемый операнд.*



0x35 000 = 110101 0000 0000 0000

F44F3054 =

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1111** | **0100** | **0100** | **1111** | **0011** | **0000** | **0101** | **0100** |
| **1111** | **0i00** | **010S** | **1111** | **0imm3** | **0000** | **a101** | **0100** |

i = 1; imm3 = 011, **a** = 0,

тогда позиция младшего бита = i:imm3:a = 32 -101102 = 32 – 22 = 10

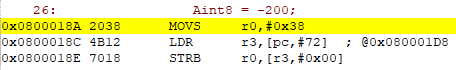
Найдем непосредственный операнд:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i | imm3 | a | Результат (32 бита) | Позиция младшего бита 32 - i:imm3:a |
| 1 | 011 | 0 | 0000 0000 0000 0011 0101 0000 00000000 | 32-22=10 |

110101 0000 000000002 = 3500016 = 21708810

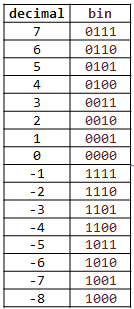
*4.5. Разберите последовательность команд, с помощью которой в переменную знакового типа помещается отрицательное число.*

4.5.1. Здесь используется команда MOVS для помещения в регистр (постфикс S – с учетом знака):



С помощью дополнительного кода получаем:

-20010 = 1110002 = 3816



Таким образом, сначала отрицательное число преобразуется методом дополнительного кода и помещается в регистр r0 с помощью команды MOVS.

Затем командой LDR загружается в регистр r3 адрес переменной.

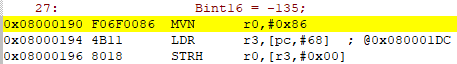
Команда STRB сохраняет значение из регистра r0 в память (B – 1 байт).

4.5.2. Здесь используется команда MVN – побитовое НЕ. Сначала число преобразуется с помощью дополнительного кода, происходит инвертирование битов, и результат кладется в регистр r0.

13410 = 8616 = 1000 01102

НЕ(1000 01102) = 0111 10012 = -13510

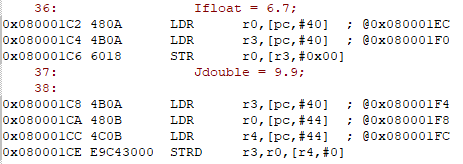
Затем в регистре r3 сохраняется адрес переменной в памяти командой LDR, знаачение переменной Bint16 помещается в память из регистра r0 командой STRH (H, так как 16 бит).



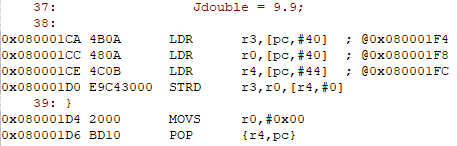
*4.6. Разберите последовательность команд, с помощью которой помещаются значения с плавающей точкой в переменные типа float и double.*

Операции с переменными с плавающей точкой выполняются программно, поэтому используются только команды LDR (загрузка из памяти в регистр) и STR(сохранение из регистра в память), STRD для double.

Float: в регистрах r0 и r3 сохраняются адреса ячеек памяти командой LDR со смещением #40, затем команда STR помещает значение переменной в соответствующую ячейку памяти.



Double: в регистрах r3, r0, r4 LDR сохраняет адреса ячеек памяти, после чего значения r3 и r4 записываются в ячейку, адрес которой хранится в r4 командой STRD (как 2 слова).



*4.7. Перечислите, какие еще способы задания операндов использовались компилятором в вашей программе. Приведите примеры для каждого способа.*

Другие используемые способы задания адреса хранения операндов:

* Косвенно-регистровый – считывание в регистр с заданным смещением, которое задается в самой команде.





* Регистровый – в команде указывается регистр, в котором находится операнд.



*5.1. Какие вы заметили отличия при присвоении переменным отрицательных значений и значений с плавающей точкой?*

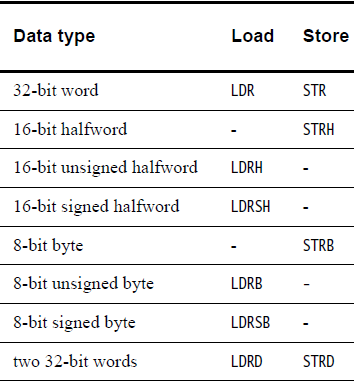
При присвоении отрицательных значений используется дополнительный код и побитовая инверсия. Для значений с плавающей точкой не используются команды типа MOV, только LDR и STR, при этом требуется более одного регистра.

*5.2. Почему компилятор иногда использует команды с постфиксами (например, LDRSB или LDRSH) для занесения в регистры чисел, а иногда не использует? Чем такие команды отличаются от своих вариантов без постфикса S?*

В случае если нужно изменить флаги, ставится S. Для 16-битных данных используется постфикс H, для 8-битных данных B.

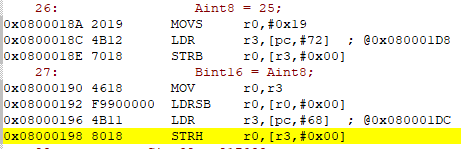
LDRSB – передача байта со знаком, LDRSH – передача полуслова со знаком.

Без постфикса B или H - для передачи слова.

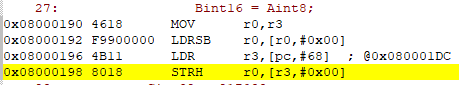


*5.3 Присвойте одной переменной другую. Как организовано копирование значения одной переменной в другую? Какая последовательность команд для этого используется, какие способы адресации использовал компилятор?*

При копировании значения из одной переменной в другую, адрес, в котором хранится это значение, сначала сохраняется в регистре, затем перемещается в другой регистр, из которого сохраняется в память как новая переменная.

При присвоении значения Aint8, в регистр r3 был помещен адрес ячейки памяти, в которой оно хранится командой LDR. 

При присвоении Bint16 Aint8, в регистр r0 помещается адрес Aint8 командой MOV, затем команда LDRSB в регистр r0 кладет значение Aint8. LDR загружает в регистр r3 адрес ячейки памяти, а STRH сохраняет значение переменной из r0 в память по адресу, хранящемуся в регистре r3.



Здесь используется регистровый способ адресации



Здесь – косвенно-регистровый



*5.4. Доступ к переменным через указатель. Создайте указатель и присвойте ему адрес одной из ваших переменных. Присвойте значение этой переменной через указатель.*

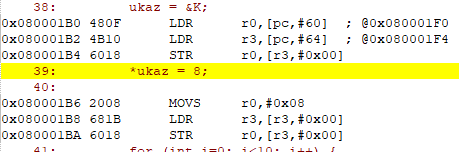
Присвоение переменной К значения 8.



* *Откуда берутся значения адресов переменных для инициализации указателей?*

После объявления указателя с помощью оператора & в него помещается адрес переменной.

* *Как в ассемблере реализовано обращение к переменной через указатель?*



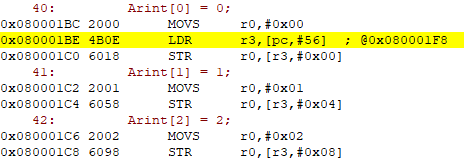
LDR заносит в r0 адрес переменной К, в r3 адрес указателя. STR помещает в указатель адрес переменной K.

MOVS помещает в r0 значение переменной, LDR сохраняет в r3 адрес переменной, STR кладет по этому адресу значение K из r0.

* *Насколько длиннее фрагмент ассемблерного кода при использовании доступа через указатель, и насколько дольше выполняется доступ?*

В 2 раза длиннее, используются 6 команд вместо трех. Требуется писать на 1 строку больше кода.

*5.5. С помощью какой последовательности команд организован доступ к элементам массива?*



MOVS сохраняет значение каждого элемента в регистр. LDR заносит в регистр адрес ячейки памяти только для первого элемента массива. Так как элементы располагаются подряд, адрес каждого следующего получается прибавлением 4 байт (для int). STR помещает значение элемента в ячейку памяти со сдвигом (4\*номер элемента).

**Часть вторая:**

*Каково значение переменной result после выполнения каждого из фрагментов?*

1. result = 1
2. result = 0

*Как вы думаете, почему очень похожий код приводит к разному результату? Подсказка: ответ "потому что в ассемблере разные команды" лишь порождает встречный вопрос "но почему в ассемблере разные команды?"*

К разным ответам приводят разные типы переменной x. В первом случае происходит неявное преобразование типов к int8\_t , и переменная x сохраняет свое значение -1. Во втором случае типы преобразуются к беззнаковому uint32\_t, поэтому значение х становится большим (0хFFFFFFFF), и вычисления происходят неверно.