Мяснянкин Никита 33328/2

Часть 1:

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

сначала идет команда записи числа в регистр:

0x080003A6 2007 MOVS r0,#0x07

это означает, что по адресу 0x080003A6 лежит команда 2007, означающая записать(2) в регистр 0 (0) число 7 (07)

далее идет команда считывания в регистр r1 значения, лежащего по адресу «то, что pc+312»:

0x080003A8 494E LDR r1,[pc,#312] ; @0x080004E4

потом идет команда записи

0x080003AA 7008 STRB r0,[r1,#0x00]

вычисляется адрес: значение регистра r1+0, и по этому адресу записывается младший байт из r0

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

команда 2007 MOVS r0,#0x07 имеет следующее представление в двоичном виде:

0010000000000111 первые 5 бит – указывают на саму команду mov, следующие 3 бита указывают на регистр, последние 8 бит отведены на непосредственный операнд: 00000111=7

1.4.3 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.

длинная команда: 0x080003C4 F64070FF MOVW r0,#0xFFF

ее двоичное представление: 11110110010000000111000011111111

111101001000 – длинная команда

111111111111 – непосредственный операнд

0000 – регистр слагаемое

0000 – регистр приемник результата

1.4.4 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.

0x080003EC F44F20F0 MOV r0,#0x78000

11110100010011110010000011110000

111100001000 – команда

i=1

imm3=010

a=1

10101=21, 32-21=11 – позиция значащего бита.

непосредственный операнд: 1111000000000000000=0x78000

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

0x080003B6 20FF MOVS r0,#0xFF

0x080003B8 7008 STRB r0,[r1,#0x00]

последовательность практически не отличается, т. к. используется доп. код

FF=-1.

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

86: f=3.3;

В регистр r0 считывается адрес «то что в pc+60»

0x080004D0 480F LDR r0,[pc,#60] ; @0x08000510

В регистр r1 считывается адрес «то что в pc+64»

0x080004D2 4910 LDR r1,[pc,#64] ; @0x08000514

в регистр r0 записывается значение по адресу «то что в r1+0»

0x080004D4 6008 STR r0,[r1,#0x00]

таким образом мы имеем адрес начала переменной, переменная записана в 2 слова

87: d=3.3;

0x080004D6 F04F3066 MOV r0,#0x66666666

0x080004DA 490F LDR r1,[pc,#60] ; @0x08000518

0x080004DC 4A0F LDR r2,[pc,#60] ; @0x0800051C

0x080004DE C203 STM r2!,{r0-r1}

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

косвенная базовая адресация со смещением :

0x080003B2 7008 STRB r0,[r1,#0x00]

косвенная базовая индексная адресация:

0x080004C4 5411 STRB r1,[r2,r0]

1.5.1 Какие вы заметили отличия при присвоении переменным отрицательных значений и значений с плавающей точкой?

при присвоении целочисленным переменным дробные значения, дробная часть отбрасывается.

при присвоении отрицательных значений используется дополнительный код.

1.5.2 Присвойте одной переменной другую. Как организовано копирование значения одной переменной в другую? Какая последовательность команд для этого используется, какие способы адресации использовал компилятор? а16=а8

в регистр r0 считывается значение по адресу… (косвенная базовая адресация со смещением)

0x080004E2 4803 LDR r0,[pc,#12] ; @0x080004F0

0x080004E4 F9900000 LDRSB r0,[r0,#0x00]

в регистр r1 считывается значение по адресу…

0x080004E8 4902 LDR r1,[pc,#8] ; @0x080004F4

сохраняет регистр r0 по адресу […](сохраняет значение а8 по адресу а16)

0x080004EA 8008 STRH r0,[r1,#0x00]

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

а) Откуда берутся значения адресов переменных для инициализации указателей?

значение адреса лежит недалеко от команды, его использующей, оно хранится по адресу “то, что в pc+8”

0x080004E0 4902 LDR r1,[pc,#8] ; @0x080004EC88

б) Как реализовано обращение к переменной через указатель?

ассемблеру все равно на имена, ему нужны адреса, а так как указатель это и есть адрес, то никакой разницы нет: обращение имеет такой же вид

0x080004E0 4902 LDR r1,[pc,#8] ; @0x080004EC

считывание в регистр r1 адреса переменной а8

помещаем число 6 в регистр r0

0x080004E2 2006 MOVS r0,#0x06

записываем содержимое регистра r0 адресу «то что в r1+0»

0x080004E4 7008 STRB r0,[r1,#0x00]

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

часть 2

#include <stdint.h>

#include <stdbool.h>

int8\_t x = -1;

uint8\_t y = 1;

bool result = **false**;

int main(void)

{

if( x < y )

{

result = **true**;

}

else

{

result = **false**;

}

while(1);

return 0;

}

в этом случае используется команда условного перехода BGE, которая смотрит на флаги N и V. результатом работы этой программы будет ответ true.

#include <stdint.h>

#include <stdbool.h>

int32\_t x = -1;

uint32\_t y = 1;

bool result = **false**;

int main(void)

{

if( x < y )

{

result = **true**;

}

else

{

result = **false**;

}

while(1);

return 0;

}

этот код отличается только типом сравниваемых чисел(и ответом). В таком коде используется ассемблерная команда BCS, которая смотрит на флаг С. результатом будет ответ false.

Полагаю это разница обусловлена неявным преобразованием типов переменных. в первом случае переменные преобразуются в знаковый тип, во втором – в беззнаковый.