Зображення, що містить будівля, ескіз, малюнок, замок

Автоматично згенерований опис  
Національний технічний університет України

"Київський політехнічний інститут імені Ігоря Сікорського"

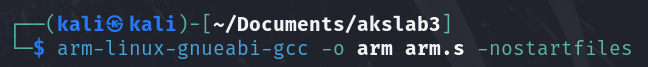
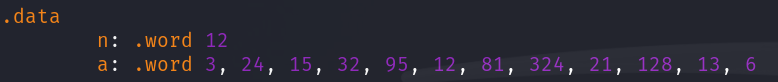
Архітектура комп’ютерних систем

Студент: Бичок Вадим Вячеславович

Лабораторна робота 3: Основи побудови програми на асемблері в архітектурі ARM

Варіант 1

Завдання:

1. Завантажити VisUAL2 (https://github.com/tomcl/V2releases).  
   Спочатку я завантажив VisUAL2 але мені не сподобалося що він не підтримує операції множення і ділення. Тому я вирішив на свою віртуальну машину на kali Linux скачати емулятор ARM асемблера gnueabi.   
   Команди:   
   Виявилося що там також не працюють ці операції, але вже були витрачені зусилля тому мною було прийняте рішення виконувати лабораторну роботу на Linux.  
   Компілювати будемо командою 
2. Ознайомитись з теоретичними положеннями
3. Визначити змінні, занести відповідні значення у регістри та організувати цикл роботи для архітектури ARM згідно свого варіанту
   1. Дано натуральні числа n, , ..., . Визначити кількість членів послідовності , ..., , що мають непарні порядкові номери, при цьому кратними 3 і не кратними 5
   2. Визначаємо список і його довжину:  
      
   3. Вручну задаємо лінкеру точку входу, а при компіляції використовуємо параметр . Цей параметр повідомляє GCC не використовувати стандартний файл запуску (який включає точку входу \_start).   
      
   4. Починаємо виконувати завдання. Задаємо початкові значення для циклів(ініціалізуємо їх). Регістр R0 буде ходити по списку, і приймати адреси змінних. R9 відповідатиме за закінчення циклу при досяганні кінця. R11 буде рахувати кількість чисел які кратні 3 і номера яких не парні. А R4 буде рахувати не кратні 5 з тих що кратні 3.  
      Зображення, що містить текст, Шрифт, знімок екрана, Графіка

      Автоматично згенерований опис
   5. В циклі mult перевіряємо чи регістр R9 рівний числу 12, тобто чи дійшов цикл кінця списку, і якщо R9 і 12 рівні CMP дає флаг Z = 1, тобто їхня різниця 0 і тому вони рівні. Так умовний перехід BEQ відповідає за перехід якщо флаг Z = 1, тоді нас перекине в завершення програми. А третій рядок переносить в регістр R1 значення яке лежить за адресою в R0.  
      Зображення, що містить текст, Шрифт, знімок екрана, годинник

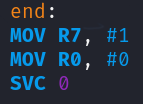
      Автоматично згенерований опис
   6. У циклі ifmult3 ми перевіряємо чи R1 і 3 рівні, якщо вони рівні це означає що число в регістрі R1 кратне 3, то CMP дає флаг Z = 1. Так умовний перехід BEQ відповідає за перехід якщо флаг Z = 1, тоді нас перекине в вихід з циклу і додавання 1 до лічильника. Якщо CMP видасть N = 1 тобто різниця від’ємна, це означатиме що число не кратне 3, і умовний перехід BMI виконується якщо CMP видає флаг N = 1, і перекидає в кінець циклу. І якщо число не є 3 ми відніматимемо від нього 3 поки не виконається умовний перехід, це принцип ділення. І безумовний перехід B завжди перекидатиме в початок циклу, допоки не виконається якийсь умовний перехід.  
      Зображення, що містить текст, Шрифт, знімок екрана, число

      Автоматично згенерований опис
   7. Додавання до лічильника R11(лічильник чисел кратних 3) одиниці. І додаткова ініціалізація внутрішнього циклу, який буде брати значення яке пройшло через перший фільтр кратності заносити в регістр R3 і буде його перевіряти на другий.  
      Зображення, що містить текст, Шрифт, знімок екрана, Графіка

      Автоматично згенерований опис
   8. Аналогічно до перевірки на кратність з 3 перевіряємо на кратність з 5, тільки умовні переходи поміняні місцями, так як нам потрібно тепер вибрати не кратні числа.  
      Зображення, що містить текст, Шрифт, знімок екрана, число

      Автоматично згенерований опис
   9. І, якщо число не кратне 5, збільшуємо лічильник на 1  
      Зображення, що містить Шрифт, текст, Графіка, знімок екрана

      Автоматично згенерований опис
   10. Кінець циклу mult в якому до регістра R9 додаємо 2, так як перевіряємо числа через одне. Додаємо до R0(адреса першого числа в списку) 8, щоб перейти до наступного непарного за номером елемента в списку, так як в списку який задається різниця адрес сусідніх елементів списку дорівнює 4. І безумовний перехід буде завжди перекидати в початок циклу, допоки не виконається умовний перехід в закінчення програми.  
       Зображення, що містить текст, Шрифт, знімок екрана, Графіка

       Автоматично згенерований опис
   11. Закінчення програми. В регістр R7 заносимо 1(Syscall number 1 (exit)), що є номером системної функції, яка завершує програму. В регістр R0 заносимо 0 що сповіщає про статус завершення 0. І через SVC виконуємо системний виклик який лежить в R7, тобто завершення програми.  
       
4. Перевірка програми в cutter
   1. Регістр R0 взяв адресу списку.  
      Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

      Автоматично згенерований опис
   2. Перевірили трійку регістр R11(лічильник кратності 3) має значення 1, так само як і R4(лічильник не кратності 5), тобто код праює успішно для 3. Так само правильно збільшився регістр R9 збільшився на 2,   
      Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

      Автоматично згенерований опис
   3. Адреса списку збільшилася на 8, тобто буде братися третій елемент списку.  
      Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

      Автоматично згенерований опис
   4. Перевіряємо 15 на наші умови.  
      Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

      Автоматично згенерований опис
   5. Лічильник R11 для кратності 3 збільшився.  
      Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

      Автоматично згенерований опис
   6. Лічильник для не кратності 5 регістр R4 не збільшився, отже програма працює правильно. Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

      Автоматично згенерований опис
5. Отже, все правильно і наша відповідь лежить в регістрі R4.
6. Контрольні запитання
   1. В чому особливість RISC?  
      1) RISC (Reduced Instruction Set Computer) — передбачає використання простого, оптимізованого набору інструкцій для процесорів. Основна задумка це максимальна ефективність виконання інструкцій, забезпечення швидшої роботи на апаратному рівні за рахунок спрощення процесорних операцій.  
        
      2) ***Фіксована довжина інструкцій****:*  
      Інструкції в RISC мають однакову довжину (як правило, 32 біти). Це спрощує декодування інструкцій процесором і дозволяє ефективно використовувати пайплайнове(У процесорах RISC широко використовується **конвеєр** (pipeline), що дозволяє одночасно виконувати кілька інструкцій на різних етапах (фазах) їх обробки. Це збільшує пропускну здатність процесора та підвищує його продуктивність.) виконання інструкцій (конвеєр).  
        
      3) **Навантаження-збереження (*load/store) архітектура*.**   
      Операції з пам'яттю (завантаження і збереження даних) виконуються тільки через спеціальні інструкції **LOAD** і **STORE**. Тобто, арифметичні операції виконуються лише над регістрами, а взаємодія з пам'яттю обмежується тільки інструкціями завантаження і збереження.  
        
      4) ***Регістрова архітектура****:*Процесори RISC зазвичай мають велику кількість регістрів (порівняно з CISC), що дозволяє зберігати більше даних і проміжних результатів без необхідності часто звертатися до пам'яті.
   2. Різниця між RISC та CISC  
      ***1)Простий набір інструкцій:***  
      Інструкції RISC як правило прості та виконуються за один такт процесора. На відміну від CISC, де інструкції можуть бути складними і виконуватись за кілька тактів, інструкції RISC часто виконують одну просту операцію, наприклад, завантаження або арифметичну дію.  
        
      2)  
      CISC (Complex Instruction Set Computer) процесори мають великий і складний набір інструкцій, де одна інструкція може виконувати кілька дій (наприклад, читання з пам'яті і виконання арифметичної операції одночасно). Це призводить до складнішого апаратного виконання інструкцій і меншої ефективності на такт. RISC робить ставку на простоту і швидкість виконання, тоді як CISC орієнтований на зменшення кількості інструкцій за рахунок їх складності.
   3. В чому особливість структури процесора ARM?  
      1) ***Cпрощений набір інструкцій RISK***  
      Перша особливість тому, що вона використовує спрощений набір інструкцій RISK, де кожна інструкція виконує просту операцію, зазвичай за один такт. Це робить їх більш ефективними і дозволяє використовувати менше енергії при виконанні тих же завдань у порівнянні з CISC (Complex Instruction Set Computer) архітектурою, як у процесорів x86.  
        
      2) ***Thumb і Thumb-2 інструкції:***   
      **2.1) Thumb** — це розширення архітектури ARM, що використовує 16-бітові інструкції замість стандартних 32-бітових. Це дозволяє зменшити розмір коду, що важливо для пристроїв з обмеженою пам'яттю. Процесор може переключатися між 32-бітовими та 16-бітовими інструкціями без значних втрат продуктивності.  
      **2.2) Thumb-2** — це подальше розширення, яке дозволяє змішувати 16-бітові та 32-бітові інструкції, зберігаючи ефективність та компактність коду.  
        
        
      3) ***Універсальна регістрова структура:***  
      ARM-процесори мають велику кількість універсальних регістрів (зазвичай 16 або 32), що дозволяє ефективно виконувати операції без необхідності часто звертатися до пам'яті. Це підвищує продуктивність і знижує витрати енергії, оскільки доступ до регістрів значно швидший, ніж до пам'яті.  
        
      4) ***Спеціальні регістри:***  
      Серед цих регістрів виділяються спеціальні регістри: програмний лічильник (PC, **R15**), регістр стану процесора (CPSR), регістри для стеку і повертання після виклику функцій (link register, **LR**).  
      5) ***Архітектура з підтримкою безпеки (TrustZone)****:*  
      ARM також включає технології для забезпечення безпеки, такі як **TrustZone**, яка дозволяє розділити систему на "безпечне" і "незахищене" середовище, що дає можливість реалізувати безпечне виконання чутливих операцій (наприклад, в платіжних системах або мобільних додатках).

***6)Умовне виконання практично усіх інструкцій:***  
ARM-процесори мають унікальну особливість: майже кожна інструкція може бути умовною. Тобто, будь-яка інструкція може виконуватись або пропускатись залежно від значень флагів. Це дозволяє зменшити кількість умовних переходів (**B**, **BL**) і зробити код більш компактним і ефективним. Я їх не використовував, тому що не хотів ускладнювати собі життя.

* 1. Поясніть особливості регістрів у ARM ?

1) ***Універсальна регістрова структура:***  
ARM-процесори мають велику кількість універсальних регістрів (зазвичай 16 або 32), що дозволяє ефективно виконувати операції без необхідності часто звертатися до пам'яті. Це підвищує продуктивність і знижує витрати енергії, оскільки доступ до регістрів значно швидший, ніж до пам'яті.  
  
2) ***Спеціальні регістри:***  
Серед цих регістрів виділяються спеціальні регістри: програмний лічильник (PC, **R15**), регістр стану процесора (CPSR), регістри для стеку і повертання після виклику функцій (link register, **LR**).

* 1. В чому особливості умового переходу у ARM порівняно з стандартною архітектурою х86?  
     ***1)Умовне виконання практично усіх інструкцій:***  
     ARM-процесори мають унікальну особливість: майже кожна інструкція може бути умовною. Тобто, будь-яка інструкція може виконуватись або пропускатись залежно від значень флагів. Це дозволяє зменшити кількість умовних переходів (**B**, **BL**) і зробити код більш компактним і ефективним. Я їх не використовував, тому що не хотів ускладнювати собі життя.
  2. Особливості побудови циклу у мові асемблеру для архітектури ARM

0) Цикли в асемблері ARM реалізуються за допомогою умовних або безумовних переходів, лічильників і порівнянь. Основною особливістю побудови циклу в архітектурі ARM є використання **регістрів** для зберігання лічильників циклів і **інструкцій умовного переходу** для контролю виконання циклу. Цикли в ARM можуть бути організовані з використанням простих інструкцій, таких як **B** (безумовний перехід) і **CMP** (порівняння).

1) ***Регістр як лічильник циклу****:*  
Для зберігання кількості ітерацій циклу зазвичай використовуються регістри.  
  
2) ***Інструкції умовного переходу****:*  
ARM підтримує широкий набір умовних інструкцій, таких як **BEQ** (перехід при рівності), **BNE** (перехід при нерівності), **BGT** (перехід, якщо більше), **BLT** (перехід, якщо менше), і багато інших. Вони використовуються для контролю і завершення циклу, коли умова стає хибною.

3) ***Постфіксна адресація і зсуви****:*  
ARM дозволяє легко працювати з адресами в масивах або списках, використовуючи **постінкремент** або **постдекремент**. Наприклад, можна автоматично збільшувати адресу на кожній ітерації, щоб працювати з наступним елементом масиву.

* 1. Особливості регістрів для зберігання значень з плавоючою точкую у архітектурі ARM  
     В архітектурі ARM підтримка обчислень з плаваючою точкою забезпечується за допомогою **регістрів плаваючої точки**, які використовуються для зберігання і маніпуляції з даними типу **float** або **double**. Ці регістри зазвичай пов'язані з окремим блоком для обчислень з плаваючою точкою, що може бути частиною **NEON** або **FPU (Floating Point Unit)** підсистеми в процесорах ARM.
  2. Різниця між операціями CMP, CMN, TST і TEQ.  
     1) CMP  
     Інструкція **CMP** виконує віднімання другого операнда від першого (перший операнд мінус другий) і встановлює флаги, не змінюючи значення регістрів. (N, Z, C, V)  
       
     2) CMN (Compare Negative)  
     Інструкція **CMN** виконує додавання другого операнда до першого і встановлює флаги. Вона фактично виконує операцію **Rn + Operand2**.(N, Z, C, V)  
       
     3) TST (Test Bits)  
     Інструкція **TST** виконує побітову операцію **AND** між двома операндами і встановлює флаги на основі результату, не змінюючи значення регістрів.(N, Z)  
       
     4) TEQ (Test Equivalence)  
     Інструкція **TEQ** виконує побітову операцію **XOR** між двома операндами і встановлює флаги. Це дозволяє перевірити еквівалентність двох значень.(N, Z, Інші флаги можуть бути змінені, але зазвичай не використовуються.)  
       
     **N (Negative flag)** — встановлюється, якщо результат від'ємний.  
     **Z (Zero flag)** — встановлюється, якщо результат дорівнює 0.  
     **C (Carry flag)** — встановлюється, якщо результат не від'ємний при виконанні віднімання.  
     **V (Overflow flag)** — встановлюється, якщо сталося переповнення зі знаком.
  3. Опишіть особливість операції LDR  
     Операція **LDR** (Load Register) в архітектурі ARM використовується для завантаження значення з пам'яті або констант пам'яті в регістр. Це одна з основних інструкцій для роботи з пам'яттю в ARM.  
     Команда **MOV** переносить дані з одного місця в інше. Вона використовується для переміщення значень між регістрами або для завантаження невеликих констант безпосередньо в регістр.
  4. Опишіть різницю між прапорцями у архітектурі ІА-32 та ARM  
       
     **У архітектурі ІА-32 більше флагів, у ARM є декілька аналогічних і декілька індивідуальних, наприклад Q (Saturation Flag) T (Thumb State Flag)  
     Основні прапорці (флаги) в архітектурі IA-32 (x86):**  
       
     **x86** має більш складну і різноманітну систему прапорців через історичну еволюцію архітектури і підтримку старих режимів.

**ARM** використовує більш компактний і простий набір прапорців, орієнтований на ефективність і сучасні вимоги до продуктивності.  
  
• ZF (Zero Flag) — встановлюється, якщо результат операції дорівнює нулю.  
Приклад: після команди CMP встановлюється, якщо обидва операнди рівні.

• SF (Sign Flag) — встановлюється, якщо результат операції є від'ємним (старший біт результату дорівнює 1).

• OF (Overflow Flag) — встановлюється, якщо результат операції з переповненням перевищує діапазон представлення числа (переповнення зі знаком).

• CF (Carry Flag) — встановлюється, якщо при додаванні або відніманні відбулося перенесення або запозичення. Це важливо для роботи з числами без знака.

• PF (Parity Flag) — встановлюється, якщо кількість одиниць у молодшому байті результату парна.

• AF (Adjust Flag) — використовується для коригування при операціях з BCD (Binary-Coded Decimal), встановлюється при перенесенні між ніблами (4-бітові частини байта).

• DF (Direction Flag) — використовується для інструкцій з операціями на масивах (стрічки). Визначає напрямок обробки — вперед або назад.

• IF (Interrupt Flag) — дозволяє або забороняє апаратні переривання.

Основні прапорці в архітектурі ARM:

У ARM прапорці зберігаються в **регістрі CPSR** (Current Program Status Register), де зберігаються основні флаги, що визначають результат операцій.

1. **Z (Zero Flag)** — встановлюється, якщо результат операції дорівнює нулю.  
   Аналог **ZF** в IA-32.
2. **N (Negative Flag)** — встановлюється, якщо результат операції від'ємний (старший біт результату дорівнює 1).  
   Аналог **SF** в IA-32.
3. **C (Carry Flag)** — встановлюється, якщо при додаванні було перенесення або при відніманні — запозичення (важливо для чисел без знака).  
   Аналог **CF** в IA-32.
4. **V (Overflow Flag)** — встановлюється, якщо відбулося переповнення результату при операціях зі знаком.  
   Аналог **OF** в IA-32.
5. **Q (Saturation Flag)** — використовується в операціях з обмеженням на максимальне/мінімальне значення (у розширених наборах інструкцій, таких як SIMD).
6. **T (Thumb State Flag)** — визначає, чи працює процесор у режимі Thumb (16-бітові інструкції) або ARM (32-бітові інструкції).