The Sales Calculator

1. Introduction

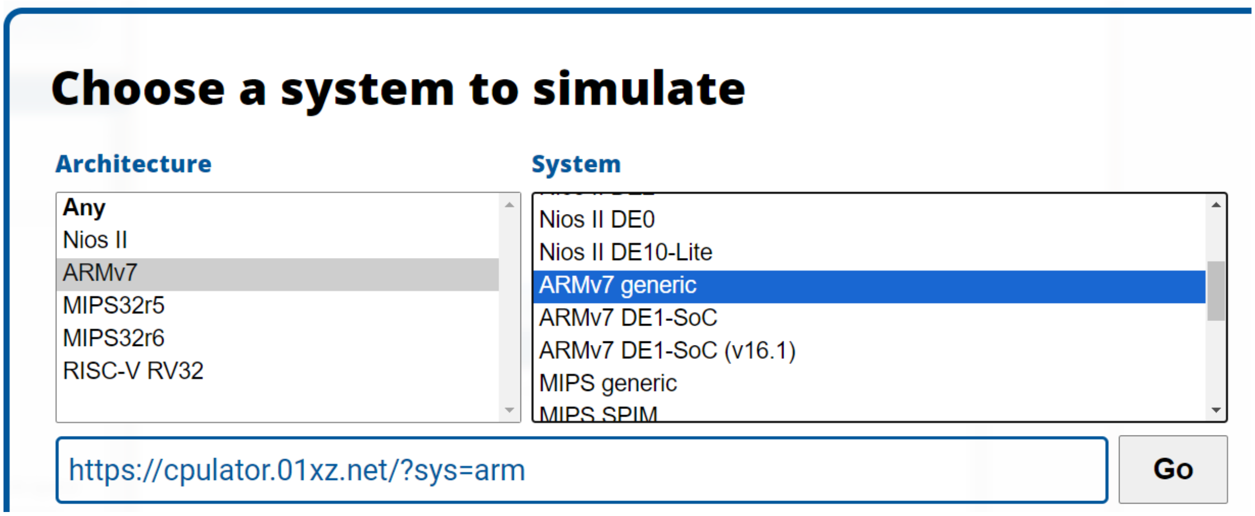
This report examines an ARM assembly program designed to analyze weekly sales data for a company. The program processes 5 days of sales data, where each day's data is stored in a variable-length array format. This report will cover the program's key components, including data initialization, sales processing, calculations and result storage.

The main objectives of the program are:

1. Total sales for the whole 5 days
2. Average sales per day (averaged over the whole 5 days)
3. The maximum amount sold in the whole 5 days
4. The minimum amount sold in the whole 5 days

Environment: CPUlator (ARM7)

For this project, we are using **CPUlator**, an online ARM7 (ARMv7 32-bit) emulator, can be accessed at: https://cpulator.01xz.net/?sys=arm-de1soc



2. Program Overview.

The program performs the following operations:

1. Initialization: Sets up memory for total, max, min, and average sales.
2. Data Processing: Iterates through each day's variable-length sales data.
3. Calculations: Computes total sales, determines average daily sales via repeated subtraction, and finds max/min values.
4. Result Storage: Saves all computed statistics in designated memory locations.

3. Detailed Program Analysis

3.1 Initializing Data

```
_start:
    LDR R0, =total_sales
    MOV R1, #0
    STR R1, [R0]
    LDR R0, =max_value
    STR R1, [R0]
    LDR R0, =min_value
    MVN R1, #0
    STR R1, [R0]
```

Variable Initialization:

1. Initializing total_sales and max_value:

   ```
   LDR R0, =total_sales
   MOV R1, #0
   STR R1, [R0]
   LDR R0, =max_value
   STR R1, [R0]
   ```

   1. Loads the addresses of total_sales and max_value into R0.
   2. Moves the value 0 into R1.
   3. Stores the value in R1 (0) at the addresses of total_sales and max_value (R0).

2. Setting up min_value:

   ```
   LDR R0, =min_value
   MVN R1, #0
   STR R1, [R0]
   ```

   1. Loads the address of min_value into R0.
   2. Uses MVN (Performs a bitwise NOT on 0) to set R1 to 0xFFFFFFFF (largest 32-bit number).
   3. Stores this value at the address of min_value.

This initialization ensures that total_sales and max_value start at 0, while min_value starts at the highest possible value. As a result, any sales data processed will correctly update these variables.

3.2 Processing Daily Sales

```
    LDR R0, =day1
    BL process_day
    LDR R0, =day2
    BL process_day
    LDR R0, =day3
    BL process_day
    LDR R0, =day4
    BL process_day
    LDR R0, =day5
    BL process_day
```

This section processes each day's data:

1. LDR R0, =day1   - Loads the address of day1's data into R0.
2. BL process_day   - Branch with Link to the process_day subroutine. The **Link** part saves the return address in the Link Register (LR), allowing the subroutine to return here after completion.

The process_day subroutine:

```
process_day:
    PUSH {R4-R7, LR}
process_day_loop:
    LDR R1, [R0], #4
    CMP R1, #0
    BEQ process_day_end
    LDR R2, =total_sales
    LDR R3, [R2]
    ADD R3, R3, R1
    STR R3, [R2]
    LDR R4, =max_value
    LDR R5, [R4]
    CMP R1, R5
    BHI update_max
    B skip_max_update
update_max:
    STR R1, [R4]
skip_max_update:
    LDR R6, =min_value
    LDR R5, [R6]
    CMP R1, R5
    BLO update_min
    B skip_min_update
update_min:
    STR R1, [R6]
skip_min_update:
    B process_day_loop
process_day_end:
    POP {R4-R7, PC}
```

1. PUSH {R4-R7, LR} - Saves these registers on the stack, crucial for preserving data across subroutine calls.
2. LDR R1, [R0], #4 - Loads a value from the address in R0, then increments R0 by 4 (moving to the next word).
3. CMP R1, #0 and BEQ process_day_end - Compares R1 to 0. If equal, the end of the day's data has been reached.
4. BHI update_max - Branch if Higher - used when a new maximum value is found.
5. BLO update_min - Branch if Lower - used when a new minimum value is found.
6. POP {R4-R7, PC} - Restores the saved registers and returns from the subroutine by popping directly into PC (Program Counter).

3.3 Calculating Average Sales

```
    LDR R1, =total_sales
    LDR R2, [R1]
    MOV R3, #5
    MOV R4, #0
    MOV R5, R2
div_loop:
    CMP R5, R3
    BLO div_end
    SUB R5, R5, R3
    ADD R4, R4, #1
    B div_loop
div_end:
    LDR R6, =average_sales
    STR R4, [R6]
```

This section calculates the average daily sales by dividing the total sales by 5 using a method of repeated subtraction.

1. LDR R1, =total_sales and LDR R2, [R1] - Loads the total sales value into R2.
2. MOV R3, #5 - Sets R3 to 5, which is our divisor (number of days).
3. MOV R4, #0 - Initializes R4 to 0. This will be our average (quotient).
4. MOV R5, R2 - Copies the total sales to R5 for manipulation.

The division loop:

1. CMP R5, R3 - Compares the remaining total (R5) with 5 (R3).
2. BLO div_end - If R5 is less than 5, branch to div_end (division complete).
3. SUB R5, R5, R3 - Subtracts 5 from the remaining total.
4. ADD R4, R4, #1 - Increments the quotient by 1.
5. B div_loop: Repeats the loop.

After the loop:

LDR R6, =average_sales and STR R4, [R6] - Stores average in memory.

This method effectively divides the total sales by 5 through repeated subtraction. The number of successful subtractions (stored in R4) represents the average sales per day. Any remainder is discarded, resulting in integer division.

3.4 Storing Results

```
    LDR R7, =min_value
    LDR R7, [R7]
    LDR R8, =max_value
    LDR R8, [R8]

    B end_program

end_program:
    B end_program
```

Loads the final min and max values into R7 and R8 accordingly.

4. Data Section

```
.data
    .align 4
day1:
    .word 5, 10, 15, 0
day2:
    .word 8, 12, 0
day3:
    .word 6, 9, 11, 0
day4:
    .word 7, 13, 0
day5:
    .word 14, 16, 0
total_sales:
    .word 0
max_value:
    .word 0
min_value:
    .word 0xFFFFFFFF
average_sales:
    .word 0
```

This section defines the data:

1. Sales data for each day, ending with 0 to mark the end of the day.
2. Memory allocated for results: total_sales, max_value, min_value and average_sales.

5. Evidence of Correctness

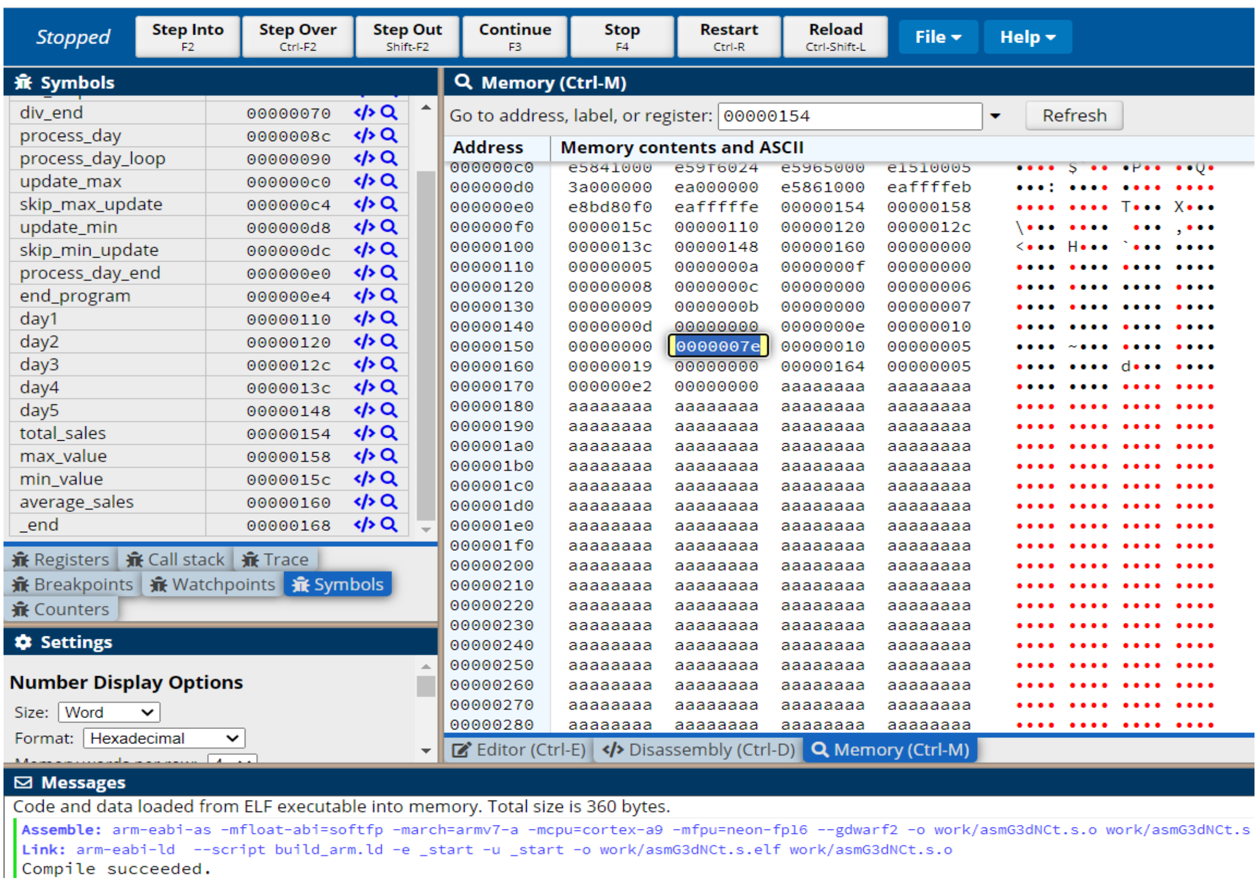
Data:

- ✓ Day 1: 5, 10, 15 (sum: 30)
- ✓ Day 2: 8, 12 (sum: 20)
- ✓ Day 3: 6, 9, 11 (sum: 26)
- ✓ Day 4: 7, 13 (sum: 20)
- ✓ Day 5: 14, 16 (sum: 30)

Expected results:

1. **Total Sales**: 30 + 20 + 26 + 20 + 30 = 126
2. **Average Sales**: 126 / 5 = 25 (integer division)
3. **Maximum Sale**: 16
4. **Minimum Sale**: 5

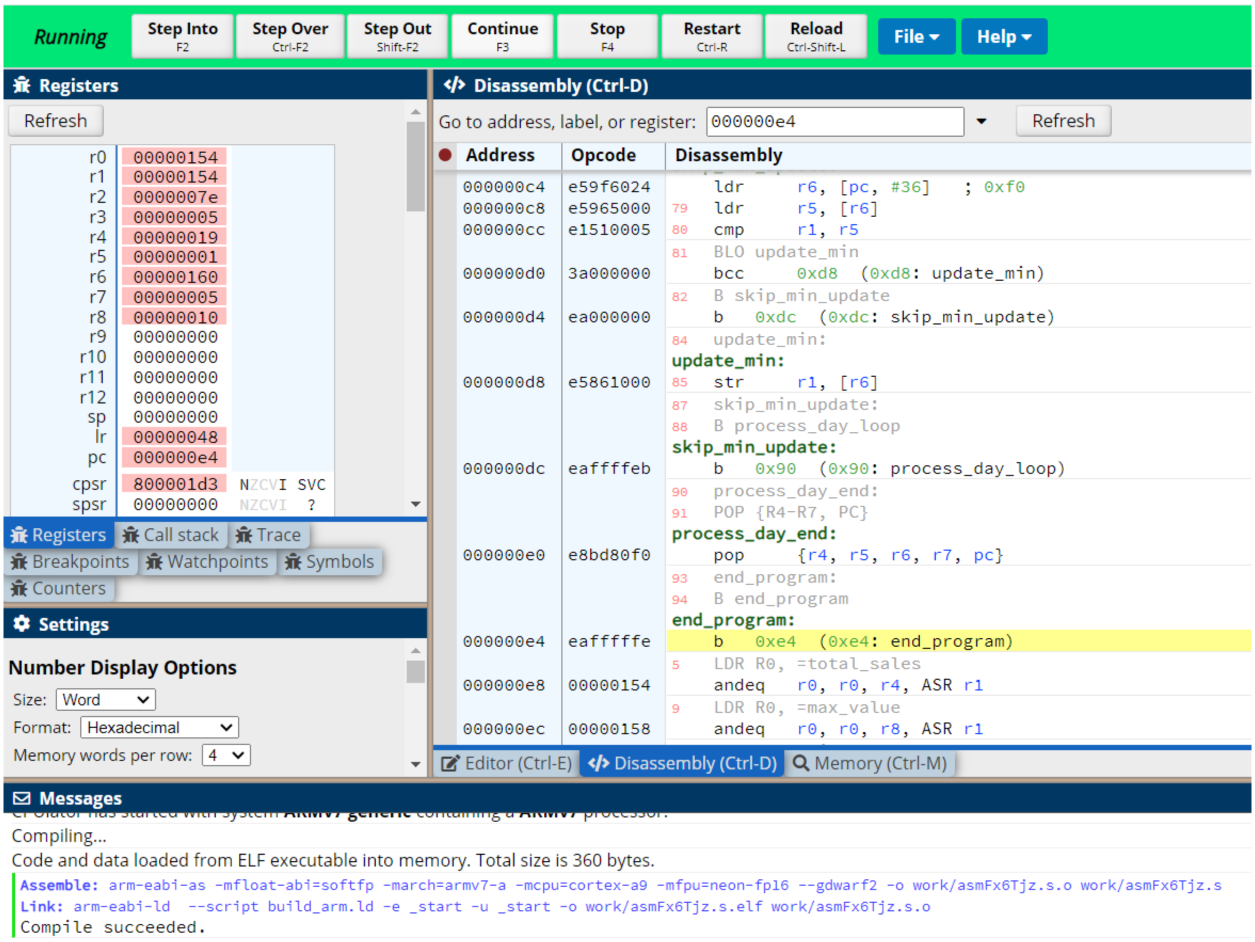
The program stores these values in their respective memory locations. Using CPUlator's memory view and register inspection features, these results can be verified after program execution.



On the left side of the interface, we can observe the memory addresses for key variables: total_sales, max_value, min_value, and average_sales.

The right side of the interface, in memory section, displays the actual values stored at these addresses upon program completion.
Example:

- In the Address Pane, we identify that total_sales is allocated at address 00000154.
- Shifting our attention to the Memory Pane on the right, we observe that this address holds the value 0000007e.
- Translating this hexadecimal value, we arrive at 126 in decimal, which accurately represents the total sales figure for our given dataset.



Upon program execution in CPUlator, the results of our calculations are also visible in the registers. Specifically, the left side of the CPUlator interface, within the registers section, displays the final values of key variables.
This feature allows for immediate verification of our program's output directly from the processor's registers.

Memory Address	Value	Description	Register
total_sales	126	Sum of sales from all days	R2
max_value	16	Maximum sales in a day	R8
min_value	5	Minimum sales in a day	R7
average_sales	25	Average sales over 5 days	R4

6. ARM Assembly Commands Used

Data Movement Commands

1. LDR (Load Register) - Loads a value from memory into a register.

   Example: LDR R0, =total_sales - Loads the address of 'total_sales' into R0.

2. STR (Store Register) - Stores a value from a register into memory.

   Example: STR R1, [R0] - Stores the value in R1 at the memory address in R0.

3. MOV (Move) - Moves a value into a register.

   Example: MOV R1, #0 - Puts the value 0 into R1.

4. MVN (Move Not) - Moves the bitwise NOT of a value into a register.

   Example: MVN R1, #0 - Puts the bitwise NOT of 0 (all 1s) into R1

Arithmetic Commands

5. ADD (Add) - Adds two values and stores the result in a register.

   Example: ADD R3, R3, R1 - Adds R1 to R3 and stores the result in R3.

6. SUB (Subtract) - Subtracts one value from another and stores the result.

   Example: SUB R5, R5, R3 - Subtracts R3 from R5 and stores the result in R5.

7. CMP (Compare) - Compares two values by subtraction, updating condition flags.

   Example: CMP R1, #0 - Compares R1 with 0, setting flags for conditional branches.

Branch Commands

8. B (Branch) - Unconditionally jumps to a specified label.

   Example: B process_day_loop - Jumps to the 'process_day_loop' label.

9. BL (Branch with Link) - Jumps to a subroutine, storing the return address.

   Example: BL process_day - Calls the 'process_day' subroutine, saving return address.

10. BEQ (Branch if Equal) - Branches if the previous comparison resulted in equality.

Example: BEQ process_day_end - Jumps to 'process_day_end' if the last comparison was equal.

11. BHI (Branch if Higher) - Branches if the previous comparison resulted in a higher unsigned value.

    Example: BHI update_max - Jumps to 'update_max' if the last comparison was higher (unsigned).
12. BLO (Branch if Lower) - Branches if the previous comparison resulted in a lower unsigned value.

    Example: BLO update_min - Jumps to 'update_min' if the last comparison was lower (unsigned).

Stack Operations

13. PUSH (Push onto Stack) - Stores multiple registers on the stack.

    Example: PUSH {R4-R7, LR} - Saves registers R4 to R7 and the Link Register on the stack.

14. POP (Pop from Stack) - Loads multiple registers from the stack.

    Example: POP {R4-R7, PC} - Restores R4 to R7 from the stack and loads PC for return.

9. References

1. ARM Limited. (2021). ARM Architecture Reference Manual. Retrieved from https://developer.arm.com/documentation/ddi0406/latest/
2. Seal, D. (2000). ARM Architecture Reference Manual. Addison-Wesley Longman Publishing Co., Inc.
3. Sloss, A. N., Symes, D., & Wright, C. (2004). ARM System Developer's Guide: Designing and Optimizing System Software. Morgan Kaufmann Publishers Inc.
4. ARM tutorial