CG2028 Computer Organisation

Semester 2 2024/2025

S2AY2425 Assignment
Group 51

Name	Matric No.
Roderick Kong Zhang	A0286550Y
Gandhi Kishen	A0266842W

Table of Contents

Chapter 1 | Discussion of Program Logic

```
@ R0: building[][] (Initial state of the car park)
@ R1: exit[][] (Number of cars exiting each section)
@ R2: entry[] (Number of cars entering the car park. You can assume the size of this array is always 5.)
@ R3: result[][] (Array to store the final cars parked, also containing F, S)
@ R4: F, entry value, section cars
@ R5: S, entry counter, building counter
@ R6: Total cars entered, section cars exiting
@ R8: Pointer to R3
@ R9: Temporary value to store SECTION_MAX
@ R10: Store F*S (length of building[][])
@ R11: Vacancy value

@ Constants
.equ ENTRY_LEN, 5            @ Length of entry array
.equ SECTION_MAX, 12         @ Maximum size of section

asm_func:
    PUSH {LR}                @ Save return address

    @ Get F*S
    LDR R4, [R3]             @ Load F value from [R3] to R4
    LDR R5, [R3, #4]         @ Load S value from [R3] to R5
    MUL R10, R4, R5          @ Store F*S in R10

    @ Get total cars entered
    MOV R5, #0               @ Initialise loop counter to 0
    MOV R6, #0               @ Initialise total cars entered to 0
    SUM_ENTRIES:
        LDR R4, [R1], #4     @ Load value from R1 into R4 and increment R1 by 4 (pointing to entries[])
        ADD R6, R4           @ Add R4 to total cars entered
        ADD R5, #1           @ Increment loop counter
        CMP R5, ENTRY_LEN    @ Repeat until all 5 entries are summed
        BNE SUM_ENTRIES

    @ Continue adding cars entered until no cars left
    MOV R5, #0               @ Initialise loop counter to 0
    MOV R8, R0               @ Set R8 as a pointer to building[][]
    ADD_ENTRIES:
        LDR R4, [R8]         @ Load value from R8 into R4
        SUBS R11, R4, #12    @ Get vacancy for section
        NEG R11, R11

        CMP R6, R11          @ Compare R6 with R11
        BGT THEN_BLOCK       @ If R6 > R11, branch to THEN_BLOCK (Remaining cars left)
        B ELSE_BLOCK         @ If R6 <= R11, branch to ELSE_BLOCK (No remaining cars)

        THEN_BLOCK:
            MOV R9, SECTION_MAX
            STR R9, [R8], #4 @ Store section cars in building[][] and move pointer R8 to next section
            SUB R6, R11      @ Cars left -= vacancy
            ADD R5, #1       @ Increment loop counter
            CMP R5, R10      @ Check if there are any sections left
            BNE ADD_ENTRIES
        ELSE_BLOCK:
            ADD R4, R6       @ Add remaining cars to section
            STR R4, [R8], #4 @ Store section cars in building[][]
            MOV R6, #0       @ Set cars left to 0

    @ Remove exited cars from each section
    MOV R5, #0               @ Initialise loop counter to 0
    MOV R8, R0               @ Set R8 as a pointer to building[][]
    REMOVE_EXITS:
        LDR R4, [R8], #4     @ Get section cars and move pointer R8 to next section
        LDR R6, [R2], #4     @ Get exited cars and move pointer R2 to next section
        SUB R4, R6           @ Cars left -= exited cars
        STR R4, [R3], #4     @ Store cars left in result[][]
        ADD R5, #1           @ Increment loop counter
        CMP R5, R10          @ Continue for all sections
        BNE REMOVE_EXITS

    POP {LR}                 @ Restore return address
    BX LR                    @ Return
```

Chapter 2 | Assignment Questions

1. *Knowing the starting address of array Building[][], how to calculate the memory address of element building[A][B] with floor index A and section index B, with the index starting from 0? Use drawing or equation to explain your answer. (4 marks)*

memory address of element building[A][B]
= address of building + 4 * (A * S + B)

2. *Describe what you observe in (i) and (ii) and explain why there is a difference. (4 marks)*

(i) **Observation:** It does not return correctly to main(), possibly leading to a segmentation fault or unexpected results.

Explanation: R14 (also known as LR or Link Register) stores the return address when calling functions. The instruction BL SUBROUTINE (Branch with Link) updates R14 with the address to return to after SUBROUTINE finishes. Since PUSH {R14} is commented out, R14 is not saved before the function call. POP {R14} is also commented out, so the correct return address is not restored. When BX LR executes at the end, it is not able to return to the main() function in the C program.

(ii) **Observation:** The program executes correctly, returning to main() as expected. The printf() statement displays the output properly.

Explanation: PUSH {R14} saves the original return address before calling SUBROUTINE. After SUBROUTINE completes, POP {R14} restores the correct return address. This ensures that BX LR correctly jumps back to main().

3. *What can you do if you have used up all the general purpose registers and you need to store some more values during processing? (2 marks)*

a. **Use the Stack (PUSH & POP)**

The ARM Cortex-M4 has **R0-R12** as general-purpose registers, but you can temporarily store data on the stack using the PUSH and POP instructions.

b. **Use Memory (RAM) Instead of Registers**

If you need to store additional values, use global/static variables or dynamically allocated memory instead of registers.

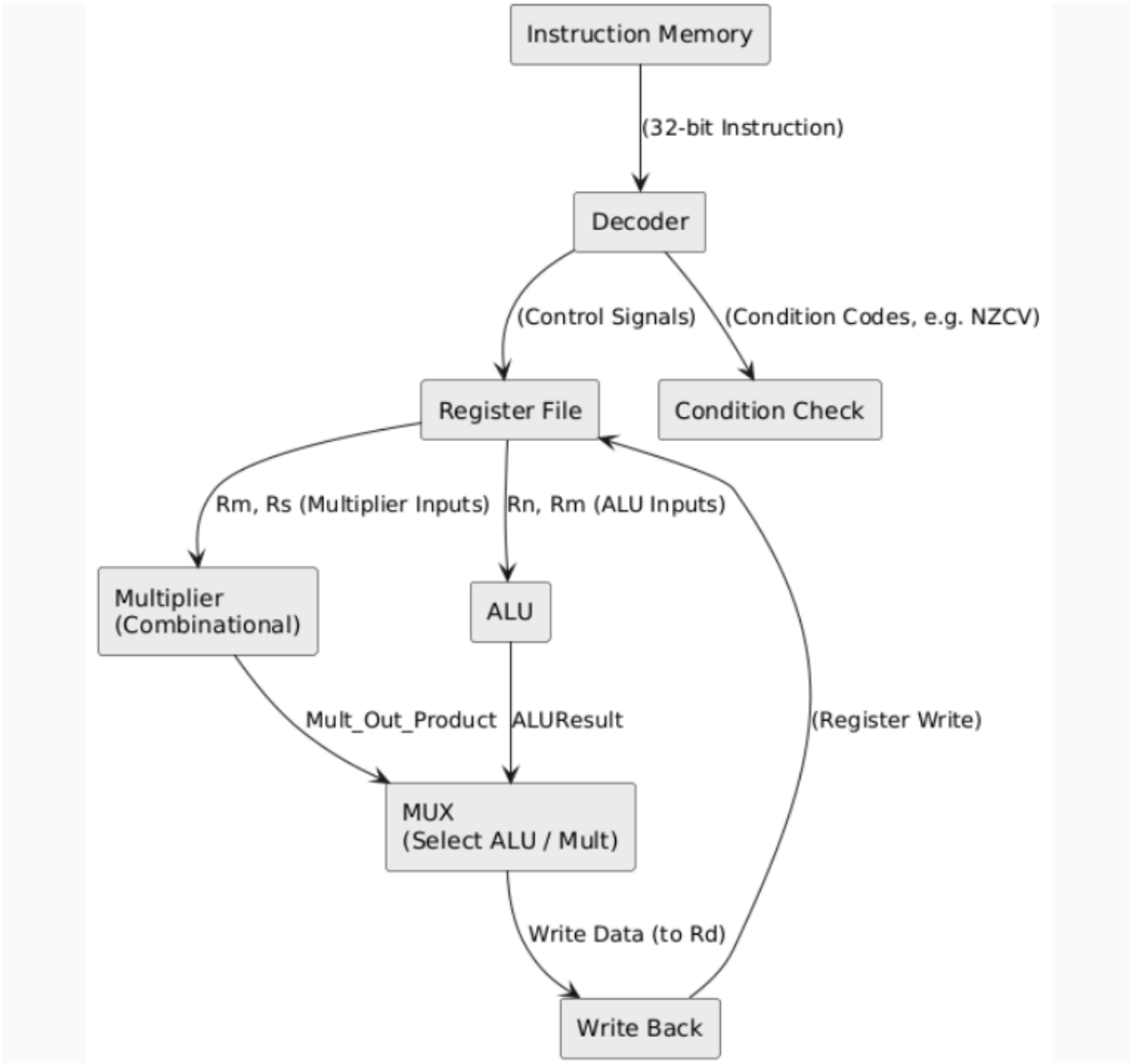
c. **Reuse Registers**

Overwrite unused registers by identifying registers that are no longer needed and overwriting them. Use registers for temporary values, and use shift and rotate instead of storing extra values. Instead of storing multiple constants, reuse the same register with shifting operations.

Chapter 3 | Machine Code

No.	Instruction Type	Instruction	Machine Code
1	Data Processing	`ADD R6, R4`	`0000 00 0100 0 0110 0000 0000 0100`
2	Memory Access	`LDR R4, [R1], #4`	`0000 01 0101 0 0100 0001 0000 0000 0100`
3	Branching	`BNE SUM_ENTRIES`	`0001 1010 0000 0000 0000 0000 0010 0000`
4	Data Processing	`MOV R5, #0`	`0011 1101 0000 0101 0000 0000 0000 0000`
5	Memory Access	`STR R9, [R8], #4`	`0000 01 0100 0 1001 1000 0000 0000 0100`

Chapter 4 | Microarchitecture Design



Chapter 5 | Discussion of Improvements Made to Enhance Efficiency

Reusing Registers

We reused registers by identifying registers that are no longer needed and overwriting them. This can be seen in all sections of the code, where R4, R5, and R6 are constantly reused for each section.

Storing Intermediate Values

Intermediate or temporary values are stored in registers to prevent recalculation and to improve efficiency. The F*S value is stored in R10 since it is constantly used to check against the loop counter.

Updating Registers Instead of Using More

Instead of creating another pointer, there are many instances where we simply used the given pointer to traverse the array. This can be observed in SUM_ENTRIES, where R1 which holds a pointer to the exit[][] array is used directly to traverse the array, instead of using another register to point to the same array.

Appendix | Members' Joint and Individual Contributions

Member	Matric No.	Type of Contribution	Contribution
Roderick Kong Zhang	A0286550Y	Report	- Assignment Question 1 - Assignment Question 3 - Microarchitecture Design
		Code	- Implemented function to return value of result[][] from asm to C program - Implemented ADD_ENTRIES to add cars to each section, ensuring that the number of cars does not exceed SECTION_MAX (12) - Optimised code (see Chapter 5) - Accounted for edge cases (cars exceeding capacity, etc.)
Gandhi Kishen	A0266842W	Report	- Assignment Question 2 - Code Documentation and Discussion - Machine Code
		Code	- Implemented SUM_ENTRIES to sum all cars entering the carpark - Implemented REMOVE_EXITS to remove cars from each section that are exiting the carpark