# PMAT 402 - Systems Programming Assignment -3

Gandholi Sarat - 23008

April 11, 2025

## Contents

**Q1 Write a sequence of instructions for SIC/XE to set ALPHA equal to 4*BETA - 9. Use immediate addressing for constants.** 2

**Q2 Write SIC instruction to swap the values ALPHA and BETA.** 3

**Q3 Draw parse trees according to the grammar for the following ⟨exp⟩:** 4

**Q4 Draw FSM to recognize a token type named `Real constant`.** 5

**Q5 What are the different classes of interrupts and their corresponding types in SIC/XE architecture? Explain with code examples.** 6

## Q1 Write a sequence of instructions for SIC/XE to set ALPHA equal to 4*BETA - 9. Use immediate addressing for constants.

To compute:

```
ALPHA = 4 * BETA - 9
```

in SIC/XE, we follow these steps:

1. Load the value stored at `BETA`.
2. Multiply it by 4 using immediate addressing.
3. Subtract 9 using immediate addressing.
4. Store the result in `ALPHA`.

**Instruction Sequence:**

```
        LDA     BETA        ; Load value of BETA into accumulator A
        MUL     #4          ; Multiply A by 4 (immediate)
        SUB     #9          ; Subtract 9 from A (immediate)
        STA     ALPHA       ; Store result into ALPHA
```

**Data Declarations:**

```
ALPHA   RESW    1           ; Reserve 1 word for ALPHA
BETA    WORD    5           ; Example value: BETA = 5
```

**Explanation:**

- `LDA BETA` loads the content of memory location `BETA` into register A.
- `MUL #4` multiplies A by 4 using immediate addressing.
- `SUB #9` subtracts the constant 9.
- `STA ALPHA` stores the final result into `ALPHA`.

This sequence assumes the program is running in SIC/XE mode with support for immediate addressing and that appropriate directives (such as `START`, `END`) and base register setup are present elsewhere in the program if needed.

## Q2 Write SIC instruction to swap the values ALPHA and BETA.

To swap the values of two memory locations `ALPHA` and `BETA` in SIC (Standard Instruction Computer), we need a temporary location to hold one of the values during the swap.

**Instruction Sequence:**

```
        LDA     ALPHA       ; Load value of ALPHA into A
        STA     TEMP        ; Store it in TEMP

        LDA     BETA        ; Load value of BETA into A
        STA     ALPHA       ; Store it into ALPHA

        LDA     TEMP        ; Load original ALPHA from TEMP
        STA     BETA        ; Store it into BETA
```

**Data Declarations:**

```
ALPHA   WORD    10          ; Example value
BETA    WORD    20          ; Example value
TEMP    RESW    1           ; Temporary word for swapping
```

**Explanation:**

- Load `ALPHA` into register A and store it in `TEMP`.
- Load `BETA` and store it into `ALPHA`.
- Load the original value of `ALPHA` from `TEMP` and store it into `BETA`.

This correctly swaps the values in memory locations `ALPHA` and `BETA` using a simple temporary variable and only the accumulator (A), which is typical in SIC programming.

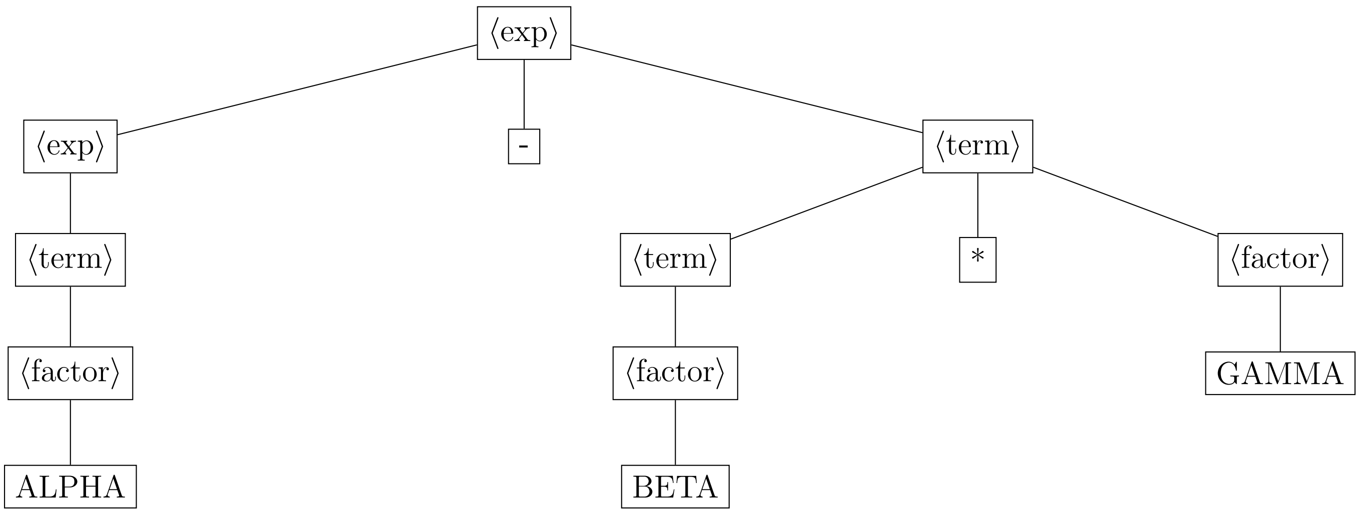
## Q3 Draw parse trees according to the grammar for the following $\langle\text{exp}\rangle$:

**Expression:** `ALPHA - BETA * GAMMA`

**Grammar Rules:**

- $\langle\text{exp}\rangle \rightarrow \langle\text{exp}\rangle\ -\ \langle\text{term}\rangle \mid \langle\text{term}\rangle$
- $\langle\text{term}\rangle \rightarrow \langle\text{term}\rangle\ *\ \langle\text{factor}\rangle \mid \langle\text{factor}\rangle$
- $\langle\text{factor}\rangle \rightarrow$ `id` | `int` | $(\langle\text{exp}\rangle)$

**Parse Tree:**



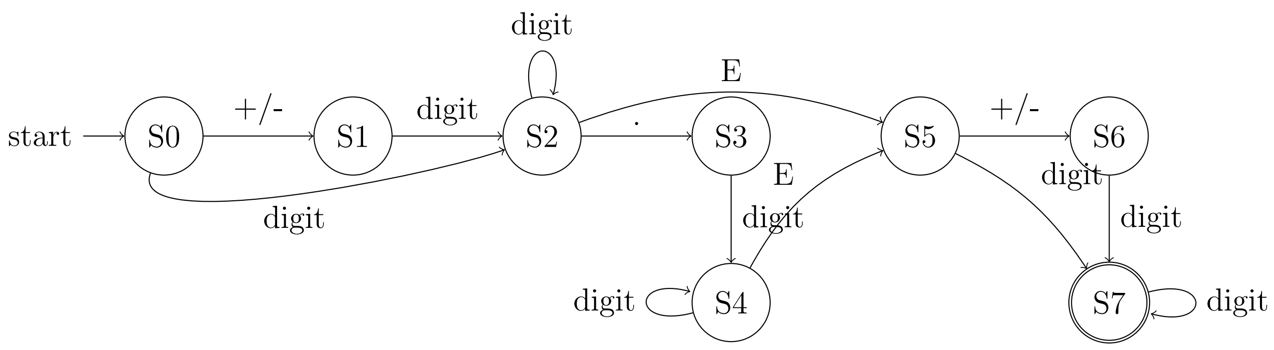
**Explanation:** According to the grammar and operator precedence, multiplication is performed before subtraction. Hence, the expression is grouped as:

`ALPHA - (BETA * GAMMA)`

The parse tree reflects this structure by breaking down the expression first by '-' at the top level and then by '*' within the right subtree.

## Q4 Draw FSM to recognize a token type named `Real constant`.

A real constant may contain digits and must either include a decimal point or a scale factor (or both). The scale factor consists of the letter `E` followed by a positive or negative integer. There must be at least one digit before the decimal point (if any).



## Q5 What are the different classes of interrupts and their corresponding types in SIC/XE architecture? Explain with code examples.

The SIC/XE architecture supports different classes of interrupts that are triggered by software or hardware events. Each class is designed for specific types of operations such as system calls, error handling, timing, or I/O operations.

| Class | Interrupt Type |
|---|---|
| I | Supervisor Call (SVC) |
| II | Program Interrupt |
| III | Timer Interrupt |
| IV | I/O Interrupt |

### Class I: Supervisor Call (SVC)

Class I interrupts are generated by executing the `SVC` instruction. These are software interrupts used to request privileged operations or services from the operating system, such as file I/O, memory management, or printing data.

**Use Case:** When a program requires access to protected resources or services, it uses an `SVC` call. The OS provides the appropriate handler for each service ID.

**Example: Request to print a character**

```
    LDA     CHAR          ; Load character to be printed
    SVC     1             ; Request OS to print character
    ...
    CHAR    BYTE    C'A'

```

*Explanation:* The `SVC 1` call causes a trap to the operating system, transferring control to the interrupt handler associated with service 1. The OS then accesses the character in register A and sends it to the output device.

### Class II: Program Interrupt

Class II interrupts occur due to illegal or exceptional operations during program execution, such as division by zero, invalid instructions, or memory access violations. These interrupts are used to detect and handle errors.

**Use Case:** Error detection and graceful failure handling during abnormal program behavior.

**Example: Divide by zero error (simulated)**

```
    LDA     NUM1          ; Load numerator
    DIV     NUM0          ; Attempt division by zero
    STA     RESULT
    ...
```

```
NUM1    WORD    10
NUM0    WORD    0    ; Division by zero causes Class II interrupt

```

*Explanation:* The division operation checks if the divisor is zero. If true, a program interrupt is generated, which halts execution and passes control to an error handler routine to prevent a crash or undefined behavior.

## Class III: Timer Interrupt

Class III interrupts are generated by a hardware timer, commonly used to implement multitasking, preemptive scheduling, or system resource monitoring. The timer interrupt helps enforce time-sharing between processes.

**Use Case:** Allow the OS to regain control after a time quantum expires, facilitating CPU scheduling.

**Example: Simulated scheduling interrupt after time slice**

```
START   LDA     #0
LOOP    ADD     #1
        COMP    #1000
        JLT     LOOP     ; Loop until 1000 iterations
        ...              ; Timer overflows => Class III interrupt

```

*Explanation:* Although not explicitly triggered in code, a hardware timer running in the background can interrupt the loop when the allocated time expires. The OS then uses this interrupt to perform scheduling or context switching.

## Class IV: I/O Interrupt

Class IV interrupts occur when an I/O operation completes or when a device requires CPU attention. These interrupts allow asynchronous communication between the processor and peripheral devices.

**Use Case:** Efficient I/O handling without polling; devices signal when ready, freeing CPU cycles.

**Example: Read data from device**

```
TD      DEVICE      ; Test device
JEQ     WAIT        ; If device not ready, wait
RD      DEVICE      ; Read from device
STCH    BUFFER      ; Store character in buffer
...
DEVICE  BYTE    X'F1'
BUFFER  RESB    1

```

*Explanation:* The device signals readiness using the `TD` instruction. Once ready, the `RD` instruction reads data, and an I/O interrupt is generated upon completion. The OS then transfers control to an I/O handler that processes the incoming data and resumes normal execution.