EXAMPLE: (a+b) + c + (d+e)

Assumptions:

- Operations CANNOT be performed using the memory locations directly
- ONLY registers can be used to perform any operation
- Data has to be FETCHED into the registers for an operation

```
LD R1, a          // Register R1 = Data in Memory Location a
LD R2, b          // Register R2 = Data in Memory Location b
ADD R3, R1, R2    // Register R3 = Register R1 + Register R2      --- (a+b)

LD R4, c          // Register R4 = Data in Memory Location b
ADD R5, R3, R4    // Register R5 = Register R3 + Register R4      --- (a+b)+c

LD R6, d
LD R7, e
ADD R8, R6, R7    // Register R7 = Register R5 + Register R6      --- (d+e)
ADD R9, R5, R8    // Register R9 = Register R5 + Register R8      --- (a+b)+c+(d+e)
```

Assumption: 1 instruction requires 1 time unit. This means this program requires 9 time units.

Scenario #1: MF (Fetch from Memory) and EX (Execution) can be done in parallel

Time Unit	MF	EX	Comment
1	LD R1, a		
2	LD R2, b		
3	LD R4, c	ADD R3, R1, R2	R3 = R1 + R2 = (a+b)
4	LD R6, d	ADD R5, R3, R4	R5 = R3 + R4 = (a+b)+c
5	LD R7, e		
6		ADD R8, R6, R7	R8 = R6 + R7 = (d+e)
7		ADD R9, R5, R8	R9 = R5 + R8 = (a+b)+c+(d+e)

Execution Time: 7 Time Units !!!

Number of Registers: 9 !!!

Task: Reduce the number of registers.

Time Unit	MF	EX	Comments
1	LD R1, a		R1 = a
2	LD R2, b		R2 = b
3	LD R3, c	ADD R1, R1, R2	R1 = R1 + R2 = (a+b)
4	LD R2, d	ADD R4, R3, R2	R4 = R3 + R2 = (a+b)+c
5	LD R3, e		R3 = e
6		ADD R2, R2, R3	R2 = R2 + R3 = (d+e)
7		ADD R1, R4, R2	R1 = R4 + R2 = (a+b)+c+(d+e)

Execution Time: 7 Time Units !!!
Number of Registers: 4 !!!

Scenario #2: We have 5 parallel units

Time Unit	U1	U2	U3	U4	U5	Comment
1	LD R1, a	LD R2, b	LD R3, c	LD R4, d	LD R5, e	ALL Loads
2	ADD R1, R1, R2	ADD R4, R4, R5				a+b and d+e
3	ADD R1, R1, R3					(a+b)+c
4	ADD R1, R1, R4					(a+b)+c+(d+e)

Execution Time: 4 Time Units !!!
Number of Registers: 5 !!!

JIT (Just-In-Time) and VM (Virtual Machine):

- JIT compiler compiles code on the fly, or in other words, just in time.
- The compiled code (bytecode sequences), are turned into a faster, more readable machine language, or native code.
- JIT compilers contrast different compiler types such as a traditional compiler, which will compile all code to a machine language before a program starts to run.
- Two common uses of JIT compilers include Java Virtual Machine (JVM) which is used in Java, as well as CLR (Common Language Runtime) which is used in C#.
- In the Java programming language and environment, a just-in-time (JIT) compiler turns Java bytecode -- a program that contains instructions that must be interpreted -- into instructions that can be sent directly to the processor.

A JIT compiler can also make relatively simple optimizations when compiling bytecode into a native machine language. e.g.

- Get rid of common sub-expressions.
- Reduce memory access in register allocations etc.

Advantages of JIT compilation include:

- JIT compilers need less memory usage.
- JIT compilers run after a program starts.
- Code optimization can be done while the code is running.
- Can utilize different levels of optimization etc.

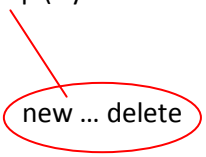
Disadvantages of just-in-time compilation:

- Startup time can take a noticeable amount of time.
- Heavy usage of cache memory.
- Increases the level of complexity in a program etc.

Runtime Environment:

Every program, when loaded into the memory, requires 4 basic independent memory chunks.

- Code
- Data (Global/Static Data)
- Stack
- Heap (Dynamic Memory Allocation)



Example of stack Usage:

main()
main() → f1()
main() → f1() → f2()
main() → f1() → f2() → f3()
main() → f1() → f2() → f3() → f4()
main() → f1() → f2() → f3()
main() → f1() → f2()
main() → f1()
main()

Notes:

- When f1 is called, some information of main is pushed onto the stack.
- Local variables of f1 are declared in the stack.