CSE-323
Computer Architecture

Reduced Instruction Set Computers (RISC)

Lecturer Afia Anjum
Military Institute of Science and technology

Semantic Gap

- Generally programmers develop or write code in HLL (High Level Language) and computer executes the code making use of LLL (Low Level language).
- The difference between HLL and LLL to execute the code is known as **semantic gap**.
- If the computers could directly execute the codes written with HLL without compilers, then there would be no semantic gaps.

Semantic Gap

Issues / Problems arising due to Semantic Gap:

- Execution inefficiency,
- Excessive machine program size,
- Compiler complexity.

Approaches to solve the Gap

- CISC Approach : In the past, engineers tried to bridge the semantic gap by making microprocessors more complex, by designing very complex architectures including a large number of complex instructions and addressing modes.
- RISC Approach : Later on, they Simplified the instruction set and adapted it to the real requirements of user programs.

To execute the models, from the architecture point of view, the microprocessor chips can be classified as :-

- Complex Instruction Set Computers (CISC)
- Reduced Instruction Set Computers (RISC)

CISC (Complex Instruction Set Computers)

- CISC has the ability to execute multi-step operations within one instruction set.
- It is the design of the CPU where one instruction performs many low-level operations.
- Called “complex” because of the complex work performed per instruction
- Hardware of the Intel is termed as Complex Instruction Set Computer (CISC)
- The CISC approach attempts to minimize the number of instructions per program, sacrificing the number of cycles per instruction.

CISC (Complex Instruction Set Computers)

- Computers based on the CISC architecture are designed to decrease the memory cost.
  - Large program means large amount of instructions, which is expensive
  - So the number of instructions per program is reduced by embedding the number of operations in a single instruction, thereby making the instructions more complex, on the other hand decreasing the memory cost

CISC (Complex Instruction Set Computers)

Examples of CISC PROCESSORS Not need to memorize

- IBM 370/168 – It was introduced in the year 1970. CISC design is a 32 bit processor and four 64-bit floating point registers.
- VAX 11/780 – CISC design is a 32-bit processor and it supports many numbers of addressing modes and machine instructions which is from Digital Equipment Corporation.
- Intel 80486 – It was launched in the year 1989 and it is a CISC processor, which has instructions varying lengths from 1 to 11 and it will have 235 instructions.

CISC Advantages

- Microprogramming is easy to implement and much less expensive than hard wiring a control unit.
- Because micro-program instruction sets can be written to match the constructs of high-level languages, the compiler does not have to be as complicated.
- This architecture makes the efficient use of main memory since the complexity (or more capability) of instruction allows to use less number of instructions to achieve a given task.

CISC Disadvantages

- Complex instruction set.
- Many specialized instructions aren't used frequently enough to justify their existence -approximately 20% of the available instructions are used in a typical program.
- The overall performance of the machine is reduced due to the different amount of clock time required by different instructions.
- Complex instruction decoding scheme & increased size of the control unit.

RISC (Reduced Instruction Set Computers)

- RISC is used in portable devices due to its power efficiency. For Example, Apple iPod and Nintendo DS.
- RISC is a type of microprocessor architecture that uses highly-optimized set of instructions.
- RISC does the opposite, reducing the cycles per instruction at the cost of the number of instructions per program.
- Pipelining is one of the unique feature of RISC.
- Called “reduced” because of the reduction of work performed by an instructions.
- RISC's original goals was to limit the number of instructions on the chip so that each could execute in one cycle.
- RISC instructions are easy to decode.

RISC Advantages

- The performance of RISC processors is often 2 to 4 times than that of CISC processors because of simplified instruction set.
- The speed of the operation can be maximized and the execution time can be minimized.
- Because the instruction set of a RISC processor is so simple, it has simple hardware requirements for decoding.
- RISC processors can be designed more quickly than CISC processors due to its simple architecture.
- Since RISC processors are simpler than corresponding CISC processors, they can complete there work in 1 clock cycle

RISC Disadvantages

- Mostly, the performance of the RISC processors depends on the programmer or compiler as the knowledge of the compiler plays a vital role.
- Since CISC machines perform complex actions with a single instruction, where RISC machines may require multiple instructions for the same action, code expansion can be a problem.
- Another problem that faces RISC machines is that they require very vast and fast memory systems to feed them instructions. RISC-based systems typically contain large memory caches, usually on the chip itself. This is known as a first-level cache

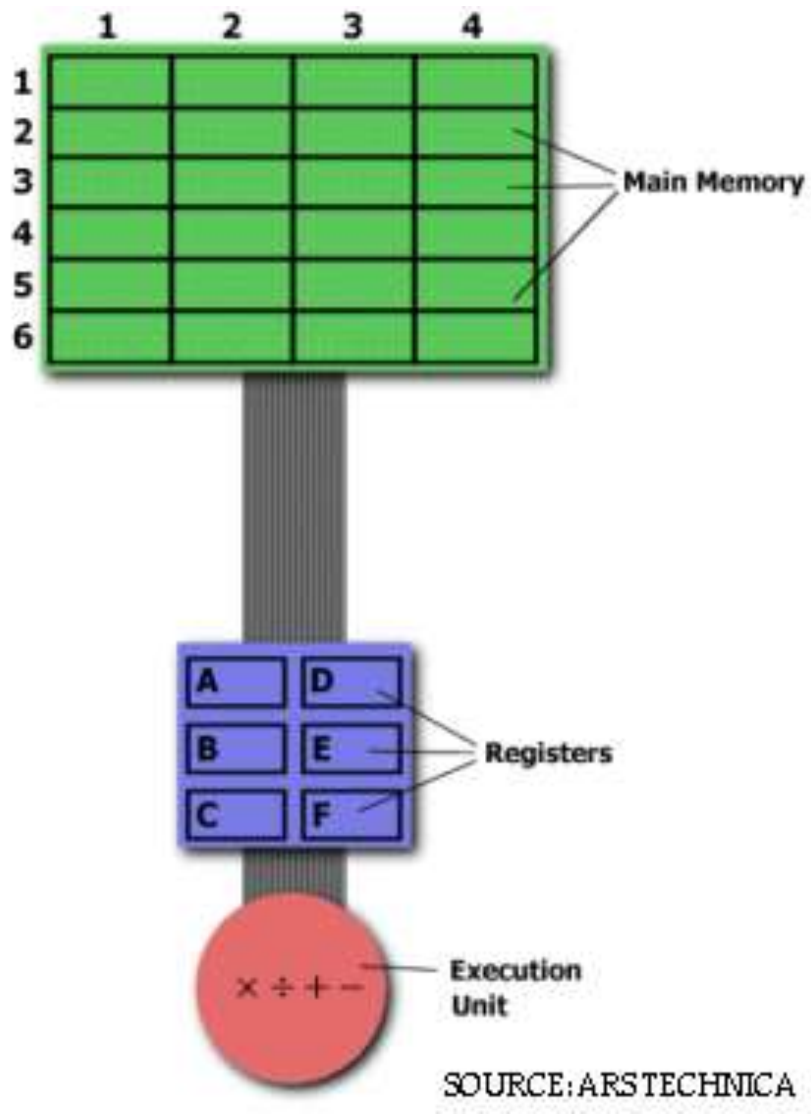
CISC Vs RISC

CISC	RISC
Many addressing mode	Fewer addressing mode
Includes complex instructions	Includes reduced instruction
Due to the complexity of the instructions of CISC architecture, the decoder is more complex	The decoder is simpler
Slower since instruction can take more than 1 clock cycle	Faster since instructions usually take 1 clock cycle
Main objective is less code	Main objective is more speed
CISC architecture do not support pipelining due to its complex instruction set	Pipelining is a common feature of the RISC architecture
Due to the higher number of transistors used in CISC architecture, it consumes more power.	Consumes less power
Micro programmed control unit	Hard wired control unit
Instruction size is mostly variable	Instruction size is always fixed

CISC Vs RISC (with example)

Multiplying Two Numbers in Memory

On the right is a diagram representing the storage scheme for a generic computer. The main memory is divided into locations numbered from (row) 1: (column) 1 to (row) 6: (column) 4. The execution unit is responsible for carrying out all computations. However, the execution unit can only operate on data that has been loaded into one of the six registers (A, B, C, D, E, or F). Let's say we want to find the product of two numbers - one stored in location 2:3 and another stored in location 5:2 - and then store the product back in the location 2:3.



CISC Vs RISC (with example)

The CISC Approach:

The entire task of multiplying two numbers can be completed with one instruction:

MULT 2:3, 5:2

- When executed, this instruction loads the two values into separate registers, multiplies the operands in the execution unit, and then stores the product in the appropriate register and loads into memory.
- This is why, MULT is what is known as a "complex instruction”
- It closely resembles a command in a higher level language. For instance, if we let "a" represent the value of 2:3 and "b" represent the value of 5:2, this command is identical to the C statement "a = a * b.“
- The compiler has to do very little work to translate a high-level language statement into assembly.
- Because the length of the code is relatively short, very little RAM is required to store instructions.

CISC Vs RISC (with example)

The RISC Approach:

- RISC processors only use simple instructions that can be executed within one clock cycle.
- Thus, the "MULT" command described above could be divided into three separate commands: "LOAD," which moves data from the memory bank to a register, "PROD," which finds the product of two operands located within the registers, and "STORE," which moves data from a register to the memory banks. In order to perform the exact series of steps described in the CISC approach, a programmer would need to code four lines of assembly:

```
LOAD A, 2:3
LOAD B, 5:2
PROD A, B
STORE 2:3, A
```

- At first, this may seem like a much less efficient way of completing the operation. Because there are more lines of code, more RAM is needed to store the assembly level instructions. The compiler must also perform more work to convert a high-level language statement into code of this form.

Pipeline Architecture

- Technique used in advanced microprocessors where the microprocessor begins executing a second instruction before the first has been completed.
- Next instructions is fetched while the processor is performing arithmetic operations.
- A Pipeline is a series of stages, where some work is done at each stage.
- Work is not finished until it has passed through all stages.
- For pipelining to work effectively, each instruction needs to have similarities to other instructions, at least in terms of relative instruction complexity.
- At every clock cycle, new instruction can be inserted for processing.
- CISC instructions do not fit pipelined architectures very well.

Pipeline Architecture (Car scenario)

- 1 employee in the company

Time	Engine	Structure	Wheels	paint
5 min	Car 1			
10 min		Car 1		
15 mins			Car 1	
20 mins				Car 1
25 mins	Car 2			
30 mins		Car 2		
35 mins			Car 2	
40 mins				Car 2

Pipeline Architecture (Car scenario)

- 4 employee in the company

Time	Engine (Emp 1)	Structure (Emp 2)	Wheels (Emp 3)	Paint (Emp 4)
5 min	Car 1			
10 min	Car 2	Car 1		
15 mins	Car 3	Car 2	Car 1	
20 mins	Car 4	Car 3	Car 2	Car 1
25 mins		Car 4	Car 3	Car 2
30 mins			Car 4	Car 3
35 mins				Car 4

Pipeline Architecture of RISC

- An instruction pipeline works the same way. Each station in the pipeline represents a different step in the load-store architecture of RISC.
- There are 5 stages:
  - **Instruction fetch (IF) -** Get instruction from memory
  - **Instruction Decode (ID) -** Translate opcode into control signals and read registers.
  - **Execute (EX) -** Perform ALU operation, compute jump/branch target
  - **Memory (MEM) -** Access memory if needed
  - **Writeback (WB) -** Update register file

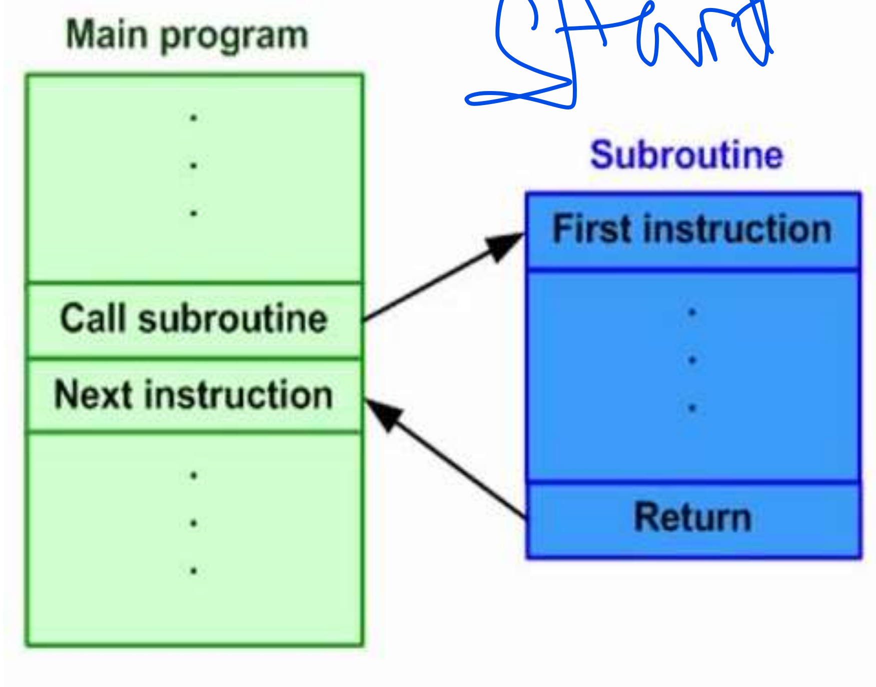
Pipeline Architecture of RISC

Clock Cycle	IF	ID	EX	MEM	WB
1	Inst 1				
2	Inst 2	Inst 1			
3	Inst 3	Inst 2	Inst 1		
4	Inst 4	Inst 3	Inst 2	Inst 1	
5		Inst 4	Inst 3	Inst 2	Inst 1
6			Inst 4	Inst 3	Inst 2
7				Inst 4	Inst 3
8					Inst 4

Overlapping Register Window Exam

Stanford

- CALL and RETURN occurs in HL programming languages. When translated into machine language, a procedure **CALL** produces a sequence of instructions that:
  - Save register values in the memory.
  - Pass parameters needed for the procedure.
  - Call a subroutine to execute the body of the procedure.



- After a procedure **RETURN**, the program will:
  - 2 Restore the old register values from the memory,
  - 1 Pass results to the calling program,
  - 3 Return from the subroutine.

Overlapping Register Window

But saving and restoring registers and passing parameters and results involve time-consuming operations. To overcome this, one of the following techniques must be used:

- **Using multiple-register banks:** each procedure is allocated its own bank of registers. This will eliminate the need for saving and storing register values
- **Using overlapped register windows:** It can be used to provide the passing of parameters and avoid the need for saving and restoring register values.

The 1st method is expensive as it requires a lot of registers.

The 2nd one is applicable in RISC architecture. We'll discuss the 2nd one briefly.

Overlapping Register Window

- The main concept is to divide the register into a set of fixed-sized windows.
- Each register window is assigned to procedure.
- Windows for adjacent procedures are overlapped to allow parameter passing.
- The window is divided into three fixed-size areas:
  1. **Parameter registers:** hold parameters passed down from the procedure that called the current procedure and hold results to be passed back up.
  2. **Local registers:** used for local variables.
  3. **Temporary registers:** used to exchange parameters and results with the next lower level.

Overlapping Register Window

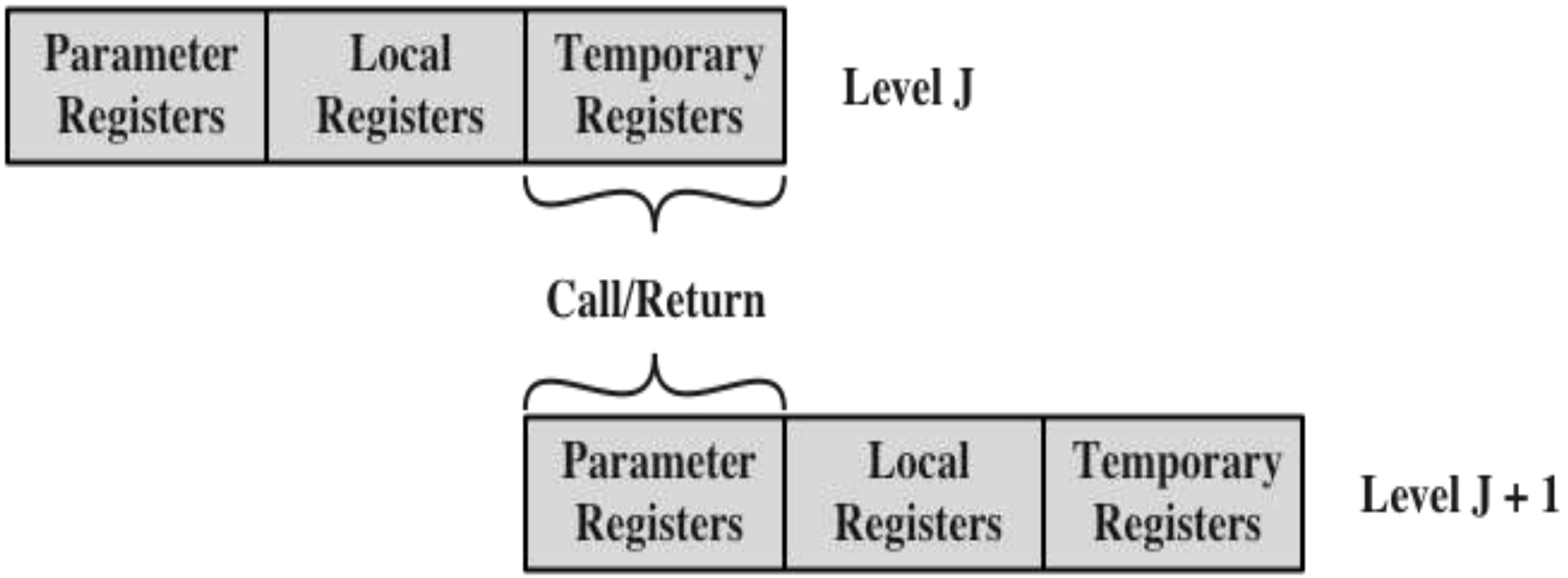
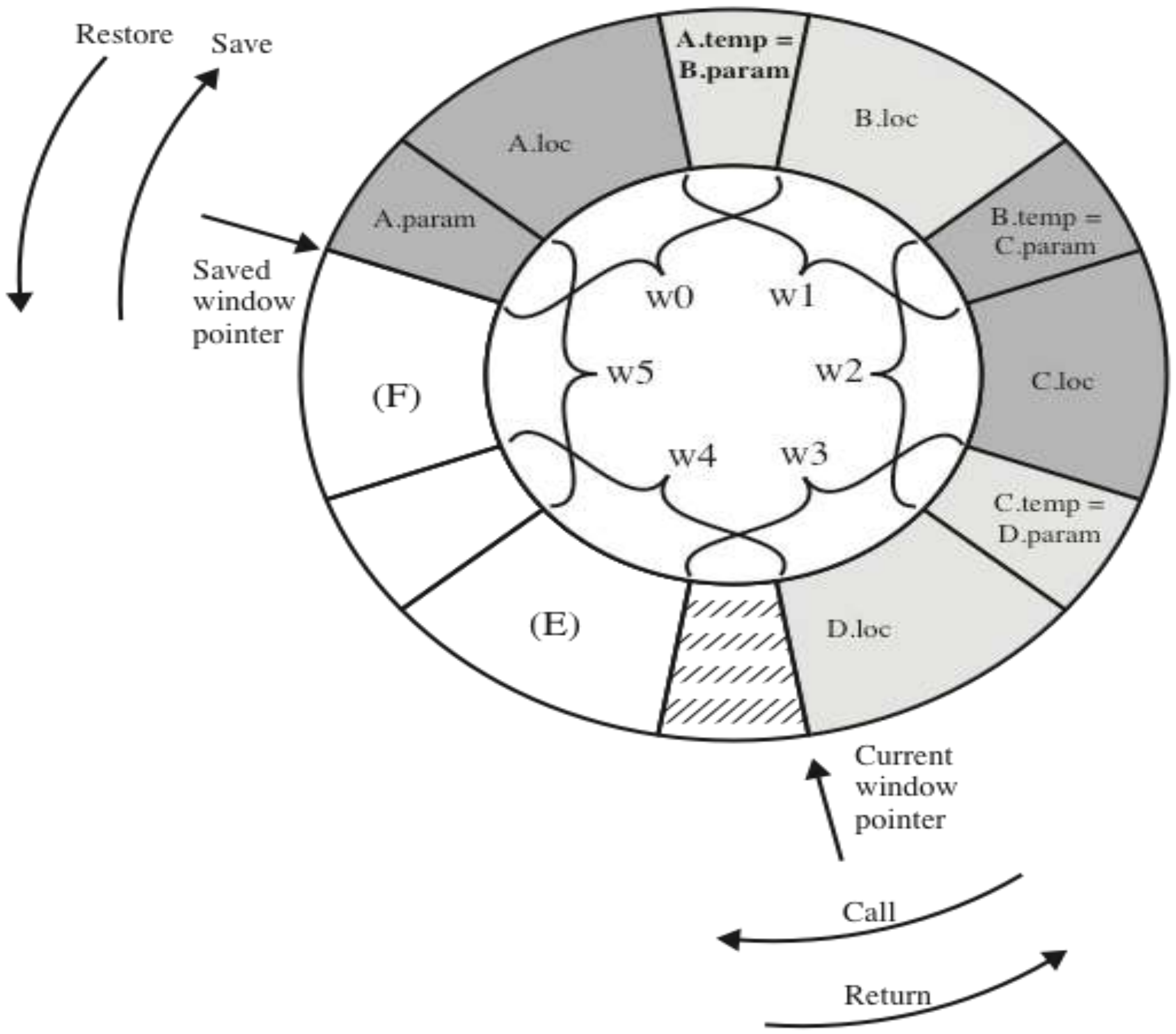


Figure 15.1 Overlapping Register Windows

Overlapping Register Window

- The temporary registers at one level are physically the same as the parameter registers at the next lower level.
- At any time only one window of registers is visible and is addressable. The active register window is indicated by a pointer
- When a function is called, a new register window is activated. This is done by incrementing the pointer.
- The register windows can be used to hold the few most recent procedure activations. Older activations must be saved in memory and later restored when the nesting depth decreases. Thus, the actual organization of the register file is as a circular buffer of overlapping windows.

Circular Buffer Organization of Overlapped Windows



Advantages of Overlapping Register Window

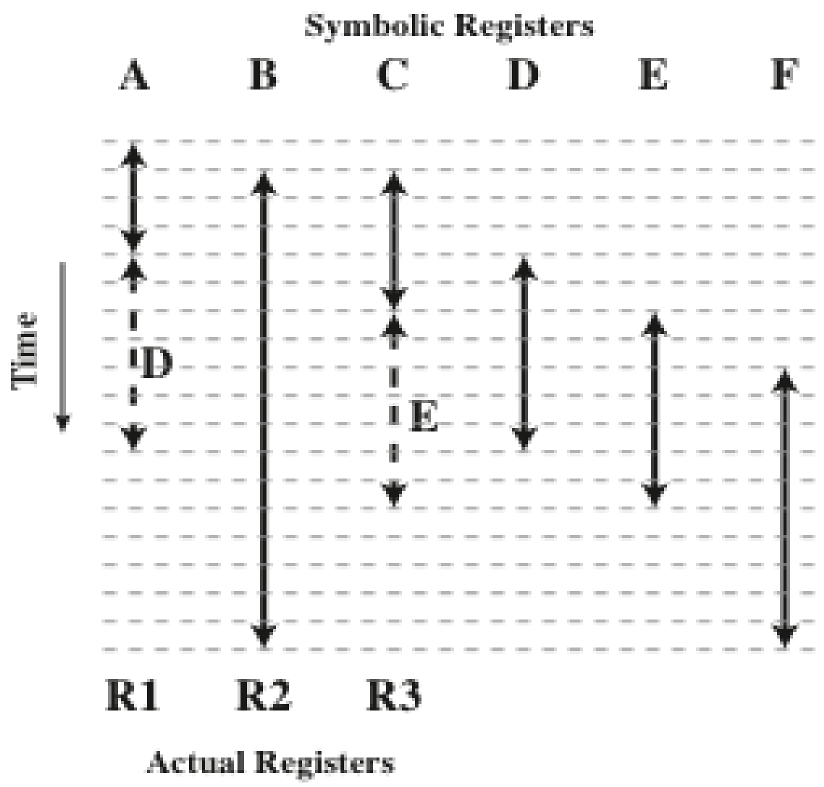
- This approach saves the time by
  - Simplifying the method of data sending to the function
  - Simplifying the method of data return
  - No need to save the value of register while calling the

Compiler Based Register Optimization using Graph Coloring Approach

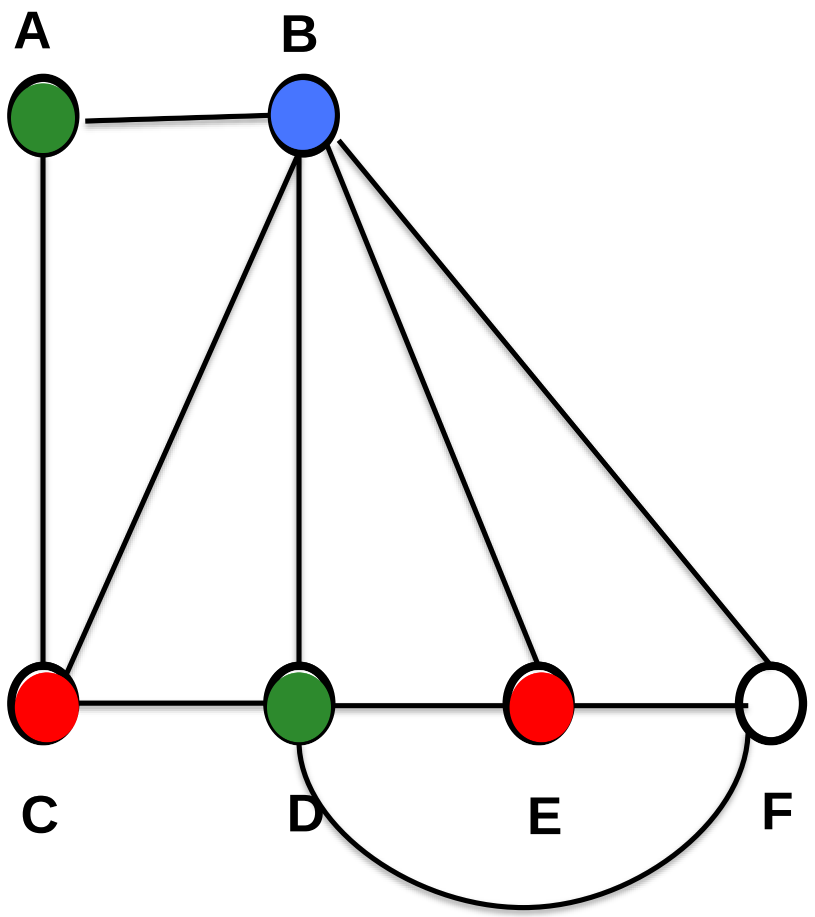
Rules:

- Nodes are symbolic registers. For simplicity, imagine symbolic registers as variables in a code
- Two symbolic registers that are used in the same time frame or same program fragment, are joined by an edge
- Assign a color to each node
- Adjacent nodes must have different colors
- Try to color the graph with n colors, where n is the number of real registers
- Nodes that can not be colored are placed in memory

Compiler Based Register Optimization using Graph Coloring Approach



(a) Time sequence of active use of registers



"Respect your parents,
They passed school without Google! 🤓"