Chapter – 8

Reduced Instruction Set Computer (RISC)

Introduction:

The RISC architecture is dramatic departure from the historical trend in processor architecture. Analysis of RISC architecture brings into focus many of the important issues in computer organization and architecture.

The key elements in design of RISC architecture are:

a. Large number of general purpose registers and the use of compiler technology to optimize the register usage.
b. Limited and simple instruction set.
c. Emphasis on optimizing the instruction pipelining.

Features:

a. Limited and simple instruction
b. Large number of general purpose registers or the use of compiler technology
c. Emphasis on optimizing instruction pipeline
d. Single cycle operation
e. Load or store design

CISC (Complex Instruction Set Computer):

CISC architecture have a richer instruction sets which include a large number of instructions and more complex instructions.

Features:

a. Complex machine instructions
b. More hardware circuitry
c. More addressing modes.

RISC v/s CISC:

RISC	CISC
Emphasis on software.	Emphasis of hardware.
Instructions of same set with few formats.	Multiple instruction sizes and formats.
Uses more registers.	Uses less registers.
Fewer addressing modes.	More addressing modes.
Instructions take one cycle time that is single clock.	Instructions take varying cycle time that is multi-clock.
Pipelining is easy.	Pipelining is difficult.

INSTRUCTION PIPELINING:

First generation RISC processors achieve execution speed that approach one instruction per system clock cycle. To improve it, two classes of processors evolved to offer execution of multiple instructions per clock cycle i.e. super-scalar and super-pipelined architectures.

A super-scalar architecture replicates each of the pipeline stages so that two or more instructions at the same stage of pipelines. Similarly, super-pipelined architecture is one that makes use of more and finer grained pipeline stages.

All instructions follow the five pipeline stages:

1. Instruction fetch
2. Source operand fetch from register file
3. ALU operation or data operand address generation
4. Data memory reference
5. Write back into register file

The limitation of super-scalar is dependencies between instructions in different pipelines can slow down the system. With super-scaling there is overhead associated with transferring instructions form one stage to another.

PIPELINING NUMERICAL:

$$Speed\ up\ ration = \frac{time\ to\ execute\ 'n'\ instructionsin\ non\ pipelined\ design}{time\ to\ execute\ 'n'\ instructions\ in\ pipelined\ design}$$

$$= \frac{k*n*t}{k*n+(n-1)*t}$$

$$= \frac{k*n}{k+(n-1)}$$

Where,

- k = number of segments
- n = number of tasks
- t = time to process a sub-operation

EXAMPLE:

Assume that pipeline has k = 8 segments and execute n = 120 tasks in sequence. Let the time taken to process a sub-operation in each segment is 20 second calculate the speed of ration in the pipeline.

Solution,

$$speed\ up\ ration = \frac{k*n*t}{k*t+(n-1)*t}$$

$$= \frac{8*120*20}{8*20+(120-1)*20}$$

$$= \frac{8 * 120 * 20}{20(8 + 119)}$$

$$= \frac{8 * 120}{8 + 119}$$

$$= \frac{960}{127}$$

$$= 7.56$$

RISC PIPELINING:

Define two phase of execution for register based instruction:

- I: Instruction Fetch
- E: Execute
    - ALU operation with register input and output.

For load and store there will be three phases:

- I: Instruction fetch
- E: Execute
    - Calculate memory address
- D: Memory
    - Register to memory or memory to register operation

Load rA 🡐 M	I	E	D										
Load rB 🡐 M				I	E	D							
Add rC 🡐 rA + rB							I	E					
Store M 🡐 rC									I	E	D		
Branch X												I	E

Fig: Sequential Execution

Load rA 🡐 M	I	E	D								
Load rB 🡐 M		I		E	D						
Add rC 🡐 rA + rB				I		E					
Store M 🡐 rC						I	E	D			
Branch X							I		E		
NOOP										I	E

Fig: Two Stage Pipelining

CONFLICTS OR HAZARDS IN INSTRUCTION PIPELINING AND THEIR SOLUTIONS:

Pipelining hazards are the problems that may be created due to the use of pipelining. Different types of hazards in pipelining are:

1. Structural Hazards:

Attempt to use the same resource by two or more instructions. For example: Access at same time by memory stage.

Solution:

a. Delay the second access by one clock cycle.
b. Provide separate memory for instructions and data.

2. Control Hazards:

Attempt to make branching decision before branch condition is evaluated.

Solution:

a. Predict: Assume an outcome and continue fetching (undo) if prediction is wrong.
b. Delayed branch:
c. Stall: Stop loading instructions until result is available.

3. Data Hazards:

Attempt to use the data before it is ready.

Solution:

a. Read after write
b. Write after read
c. Write after write (out dependencies)

Register Window:

1. Use of large set of registers decreases the need to access memory.
2. Design task is to organize registers in such a way that this goal is achieved.
3. Multiple small set of registers are used, each assigned to a different procedure.
4. A procedure call automatically switches the processor to use a different fixed size window of registers.
5. At any time only one window of register is visible and addressable as if it were the only set of registers.
6. The window is divided into three fixed size area i.e. parameter register, local register, and temporary register.
7. Parameter register hold the parameters passed down from the procedure that call the current procedure and holds the result to be passed as a backup.
8. Local registers are used for local variables as assigned by the compiler.
9. Temporary registers are used to exchange parameters and result with the next lower level. The overlap permits parameters to be passed without actual movement of data.

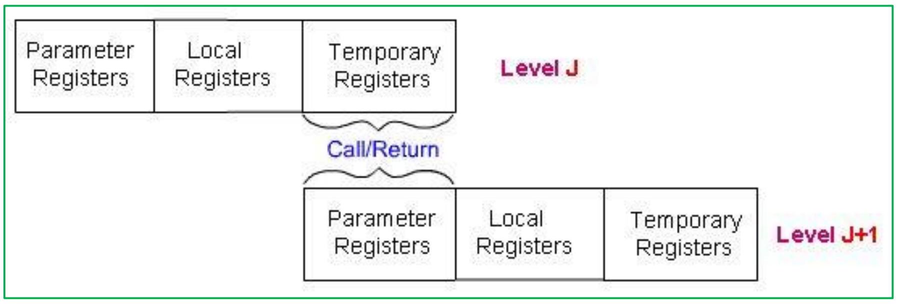

Fig: Overlapping Register Window

REGISTER RENAMING:

- ☑ When instructions are issued in sequence and complete in sequence, it is possible to specify the contents of each register at each point in the execution.
- ☑ When out of order techniques are used, the values of registers cannot be fully known at each point.
- ☑ Due to this effect, values are in conflict for the use of registers and processor must resolve these conflicts by the process of pipelining. Anti-dependencies and out dependencies are the examples of storage conflicts.
- ☑ Multiple instructions compete for the use of same register location generating pipeline constraints that retards performance.
- ☑ The method of coping with these type of conflict is duplication of resources called register renaming.