RISC-I

RISC-I Registers: (10-6-10-6)

|  |  |
| --- | --- |
| R0 | =0 |
| R1 | Function Result |
| R2 | Stack Pointer |
| R3 | Global Variables |
| R4 |
| R5 |
| R6 |
| R7 |
| R8 |
| R9 |
| R10 | Parameters for  Next Function Call |
| R11 |
| R12 |
| R13 |
| R14 |
| R15 |
| R16 | Local Variables /  Intermediate Results |
| R17 |
| R18 |
| R19 |
| R20 |
| R21 |
| R22 |
| R23 |
| R24 |
| R25 |
| R26 | Parameters for  This Function |
| R27 |
| R28 |
| R29 |
| R30 |
| R31 |

### Sample Instructions

|  |  |
| --- | --- |
| **ARM x86** (L ← R) | **RISC-I** (L → R) |
| MOV R1, R2  LDR R1, =20  LDR R1, =23456  CMP R1, R2  TEST R1, R2  NEG R1, R1  NOT R1, R1  LDR R1, [R2, #20]  LDR R1, [#20]  STR R1, [R2, #20] | ADD R0, R2, R1 ; R1 = 0 + R2  ADD R0, #20, R1 ; -212 <= N <= 212-1  ADD R0, #23456, R1 ; Bits 12:0  LDHI #23456, R1 ; Bits 31:13  SUB R2, R1, R0, {C} ; Set flags  AND R1, R2, R0, {C}  SUB R0, R1, R1  XOR R1, #-1, R1  LDL (R2)#20, R1 ; Load 32-bit long  LDL (R0)#20, R1  STL (R2)#20, R1 |

RISC-I has a two-stage pipeline, so NOPs are needed when branching:

|  |
| --- |
| CALLR R25, function ; Call function and save return address in R25.  XOR R0, R0, R0 ; NOP in delay slot.  function:  ; Function body  ; Result stored in R1  RET R25, R0 ; Return to address in R25.  XOR R0, R0, R0 ; NOP in delay slot. |

### Register Windows

Since the RISC-I chip is so small, multiple **register sets** can be implemented.

Each nested function call allocates a new **register window** from a circular register file.

If more windows are needed in a program than the number physically available, the **oldest** register window must be pushed onto the stack in main memory.

**Register Overflow** occurs if functions nest deeper than the number of windows available.

**Register Underflow** occurs on a return if a window needs to be retrieved from stack.

Pipelining

Delayed jumps occur in RISC-I because it is not possible to calculate the destination address and fetch the destination instruction within one clock cycle.

### DLX MIPS Processor

The DLX Processor has a 5 stage pipeline:

1. IF Instruction Fetch
2. ID Instruction Decode & Register Fetch
3. EX Execute & Effective Address Calculation
4. MA Memory Access
5. WB Register Write Back

An instruction is issued (ID → EX) when it can be executed without stalling.

A non-pipelined DLX requires IF & MA every 5 clock cycles.

A pipelined DLX requires IF & MA every clock cycle.

This is helped by a **Harvard Architecture** where there is a **separate** instruction and data cache.

### Data Hazards

|  |  |
| --- | --- |
| Example | Solutions |
| ADD R1, R2, R3  SUB R4, R1, R5 | Pipeline Forwarding  Two-Phase Clocking |

### Pipeline Forwarding

The ALU result from the previous instruction can be forwarded to the ALU inputs before being written back to the register file in the next stage.

This occurs during the **execution phase** rather than the decode phase.

### Two-Phase Clocking

DLX Register File can be written to and read from in a single clock cycle.

1. Write during the first half of the cycle (WB phase)
2. Read during the second half of the cycle (ID phase)

### Branching

A three cycle penalty is incurred when a branch is taken. The pipeline is stalled until the branch target is known at the end of MA.

This three cycle penalty is improved to a one cycle penalty by using a **set conditional instruction** followed by a BEQZ or BNEZ. It also uses additional hardware to resolve branches during the ID stage.

Example:

|  |
| --- |
| SLT R1, R2, R3 ; R1 = (R2 < R3)? 1 : 0  BEQZ R1, L ; Branch to L if R1 == 0 |

DLX branching can be improved by assuming the branch is **not** taken. This means that the pipeline only stalls when the branch is taken.

### Branch Prediction

Implementing a **Branch Target Buffer** can resolve branches during the IF phase. The BTB is searched for the current PC, and if it is found, uses the predicted PC to fetch the next instruction.

If the branch is **incorrectly** predicted, a one cycle stall occurs as the correct instruction is fetched. The BTB is updated.