EC-252: COMPUTER ARCHITECTURE AND MICROPROCESSORS

Vaskar Raychoudhury

Indian Institute of Technology Roorkee

Major Advances in Computers(1)

- The family concept
  - IBM System/360 1964
  - DEC PDP-8
  - Separates architecture from implementation
- Microporgrammed control unit
  - Idea by Wilkes 1951
  - Produced by IBM S/360 1964
- Cache memory
  - IBM S/360 model 85 1969

Major Advances in Computers(2)

- Microprocessors
  - Intel 4004 1971
- Pipelining
  - Introduces parallelism into fetch execute cycle
- Multiple processors

The Next Step - RISC

- Reduced Instruction Set Computer

- Key features
  - Large number of general purpose registers
  - or use of compiler technology to optimize register use
  - Limited and simple instruction set
  - Emphasis on optimising the instruction pipeline

Comparison of processors

	Complex Instruction Set (CISC)Computer			Reduced Instruction Set (RISC) Computer		Superscalar		
Characteristic	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000	PowerPC	Ultra SPARC	MIPS R10000
Year developed	1973	1978	1989	1987	1991	1993	1996	1996
Number of instructions	208	303	235	69	94	225		
Instruction size (bytes)	2–6	2–57	1–11	4	4	4	4	4
Addressing modes	4	22	11	1	1	2	1	1
Number of general-purpose registers	16	16	8	40 - 520	32	32	40 - 520	32
Control memory size (Kbits)	420	480	246	—	—	—	—	—
Cache size (KBytes)	64	64	8	32	128	16-32	32	64

Driving force for CISC

- Software costs far exceed hardware costs
- Increasingly complex high level languages
- Semantic gap
- Leads to:
  - Large instruction sets
  - More addressing modes
  - Hardware implementations of HLL statements
    - e.g. CASE (switch) on VAX

Intention of CISC

- Ease compiler writing
- Improve execution efficiency
  - Complex operations in microcode
- Support more complex HLLs

Execution Characteristics

- Operations performed
- Operands used
- Execution sequencing
- Studies have been done based on programs written in HLLs
- Dynamic studies are measured during the execution of the program

Operations

- Assignments
  - Movement of data
- Conditional statements (IF, LOOP)
  - Sequence control
- Procedure call-return is very time consuming
- Some HLL instruction lead to many machine code operations

Weighted Relative Dynamic Frequency of HLL Operations [PATT82a]

	Dynamic Occurrence		Machine-Instruction Weighted		Memory-Reference Weighted	
	Pascal	C	Pascal	C	Pascal	C
ASSIGN	45%	38%	13%	13%	14%	15%
LOOP	5%	3%	42%	32%	33%	26%
CALL	15%	12%	31%	33%	44%	45%
IF	29%	43%	11%	21%	7%	13%
GOTO	—	3%	—	—	—	—
OTHER	6%	1%	3%	1%	2%	1%

Operands

- Mainly local scalar variables
- Optimisation should concentrate on accessing local variables

	Pascal	C	Average
Integer Constant	16%	23%	20%
Scalar Variable	58%	53%	55%
Array/Structure	26%	24%	25%

Procedure Calls

- Very time consuming
- Depends on number of parameters passed
- Depends on level of nesting
- Most programs do not do a lot of calls followed by lots of returns
- Most variables are local
- (c.f. locality of reference)

Implications

- Best support is given by optimising most used and most time consuming features
- Large number of registers
  - Operand referencing
- Careful design of pipelines
  - Branch prediction etc.
- Simplified (reduced) instruction set

Large Register File

- Software solution
  - Require compiler to allocate registers
  - Allocate based on most used variables in a given time
  - Requires sophisticated program analysis
- Hardware solution
  - Have more registers
  - Thus more variables will be in registers

Registers for Local Variables

- Store local scalar variables in registers
- Reduces memory access
- Every procedure (function) call changes locality
- Parameters must be passed
- Results must be returned
- Variables from calling programs must be restored

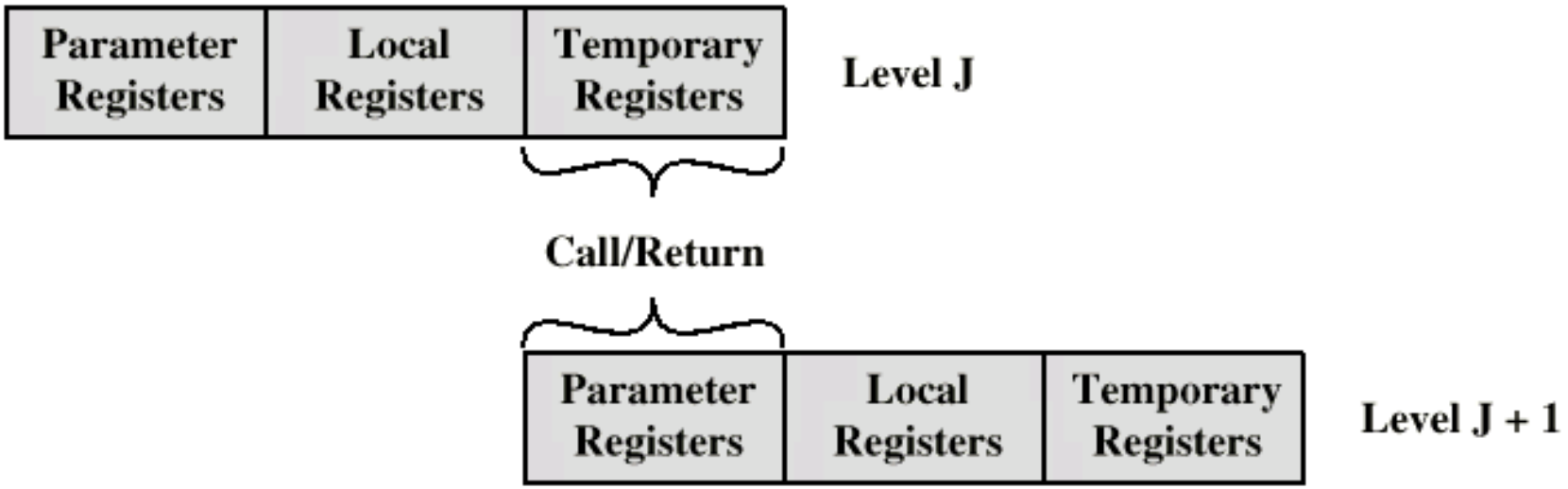
Register Windows

- Only few parameters
- Limited range of depth of call
- Use multiple small sets of registers
- Calls switch to a different set of registers
- Returns switch back to a previously used set of registers

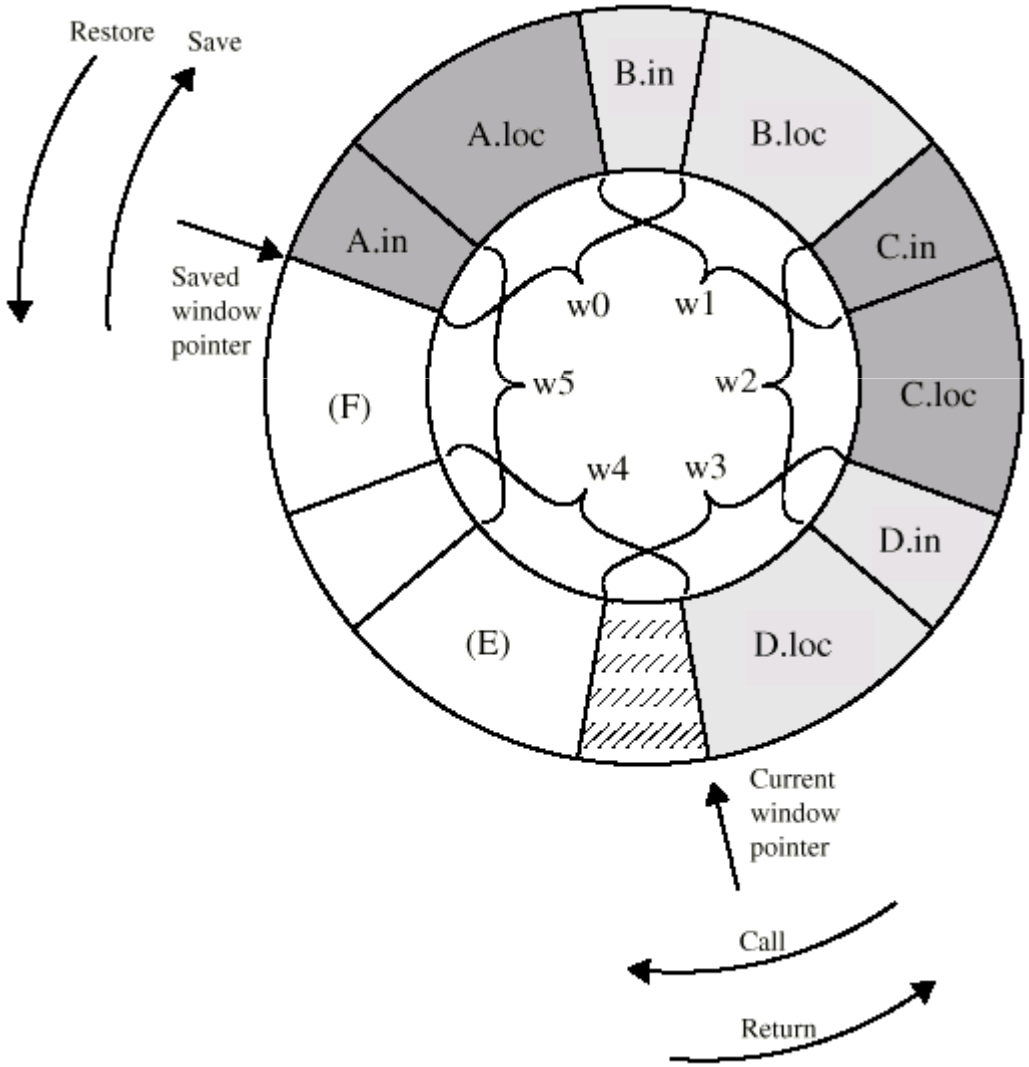
Register Windows cont.

- Three areas within a register set
  - Parameter registers
  - Local registers
  - Temporary registers
  - Temporary registers from one set overlap parameter registers from the next
  - This allows parameter passing without moving data

Overlapping Register Windows



Circular Buffer diagram



Operation of Circular Buffer

- When a call is made, a current window pointer is moved to show the currently active register window
- If all windows are in use, an interrupt is generated and the oldest window (the one furthest back in the call nesting) is saved to memory
- A saved window pointer indicates where the next saved windows should restore to

Global Variables

- Allocated by the compiler to memory
  - Inefficient for frequently accessed variables
- Have a set of registers for global variables

RISC Characteristics

- One instruction per cycle
- Register to register operations
- Few, simple addressing modes
- Few, simple instruction formats
- Hardwired design (no microcode)
- Fixed instruction format
- More compile time/effort

CISC vs RISC

- Hard to Distinguish Now. Boundary is getting vague.
- Academia don't Care
- Industry doesn't Care (Except for Advertisements)
- CISC
  - *Effectively realizes one* particular High Level Language Computer System *in HW* - recurring **HW development costs** when change needed
- RISC
  - *Allows effective realization of any* High Level Language Computer System *in SW* - recurring **SW development costs** when change needed

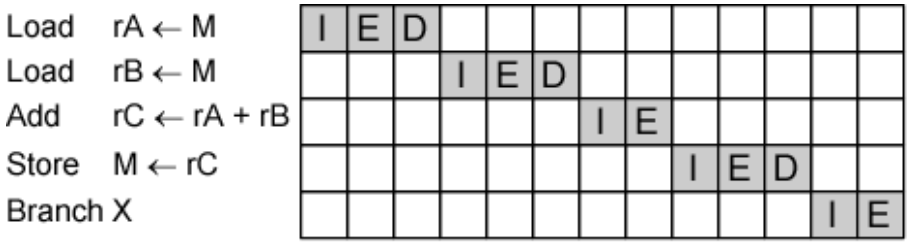
Registers v Cache (w.r.to RISC)

Large Register File	Cache
All local scalars	Recently-used local scalars
Individual variables	Blocks of memory
Compiler-assigned global variables	Recently-used global variables
Save/Restore based on procedure nesting depth	Save/Restore based on cache replacement algorithm
Register addressing	Memory addressing

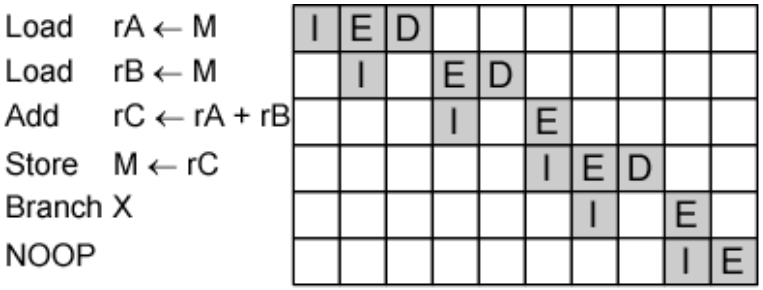
RISC Pipelining

- Most instructions are register to register
- Two phases of execution
  - I: Instruction fetch
  - E: Execute
    - ALU operation with register input and output
- For load and store
  - I: Instruction fetch
  - E: Execute
    - Calculate memory address
  - D: Memory
    - Register to memory or memory to register operation

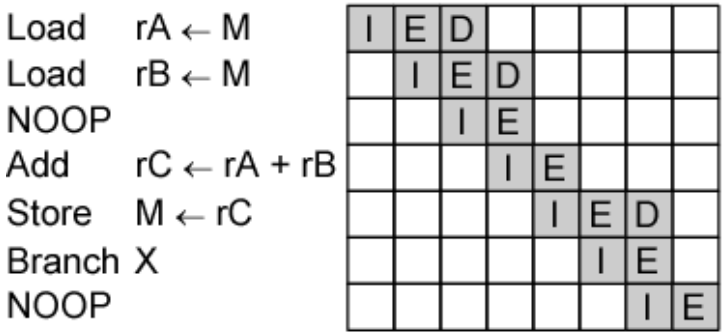
Effects of Pipelining



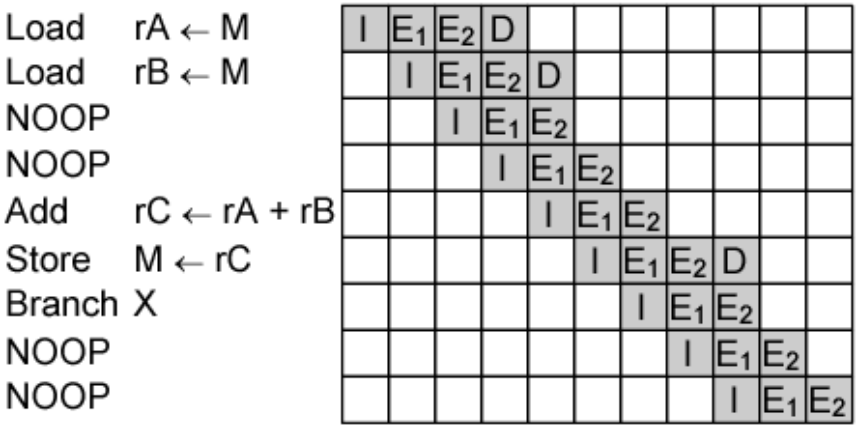
(a) Sequential execution



(b) Two-stage pipelined timing



(c) Three-stage pipelined timing



(d) Four-stage pipelined timing

Optimization of Pipelining

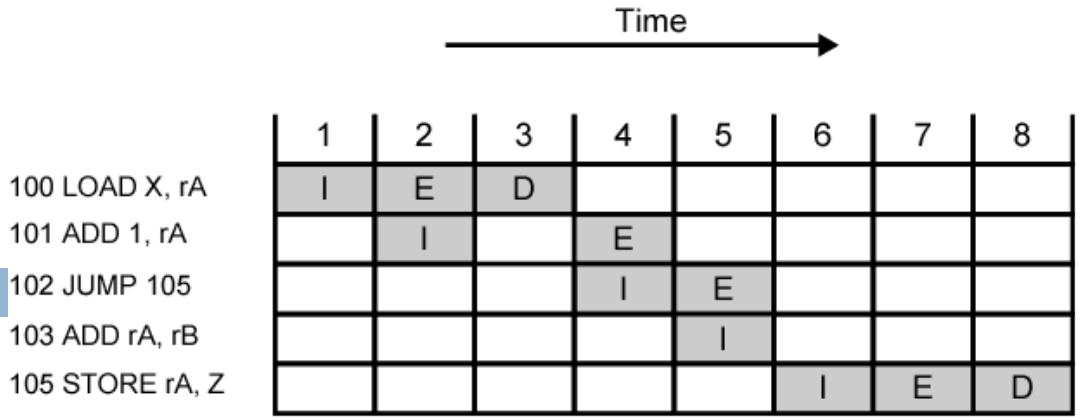
- Delayed branch
  - Does not take effect until after execution of following instruction
  - This following instruction is the delay slot
- Delayed Load
  - Register to be target is locked by processor
  - Continue execution of instruction stream until register required
  - Idle until load complete
  - Re-arranging instructions can allow useful work whilst loading
- Loop Unrolling
  - Replicate body of loop a number of times
  - Iterate loop fewer times
  - Reduces loop overhead
  - Increases instruction parallelism
  - Improved register, data cache or TLB locality

Normal and Delayed Branch

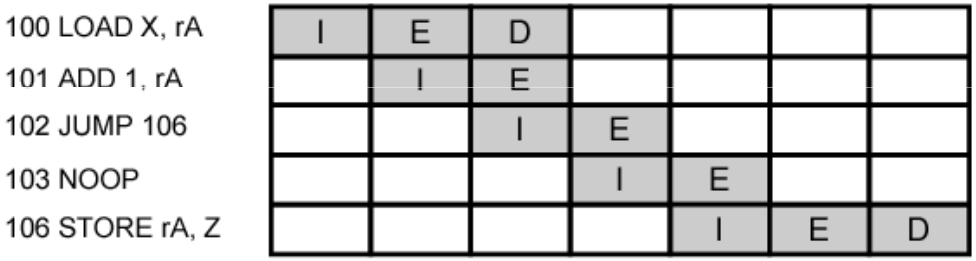
Address	Normal Branch	Delayed Branch	Optimized Delayed Branch
100	LOAD X, rA	LOAD X, rA	LOAD X, rA
101	ADD 1, rA	ADD 1, rA	JUMP 105
102	JUMP 105	JUMP 106	ADD 1, rA
103	ADD rA, rB	NOOP	ADD rA, rB
104	SUB rC, rB	ADD rA, rB	SUB rC, rB
105	STORE rA, Z	SUB rC, rB	STORE rA, Z
106		STORE rA, Z	

Use of Delayed Branch

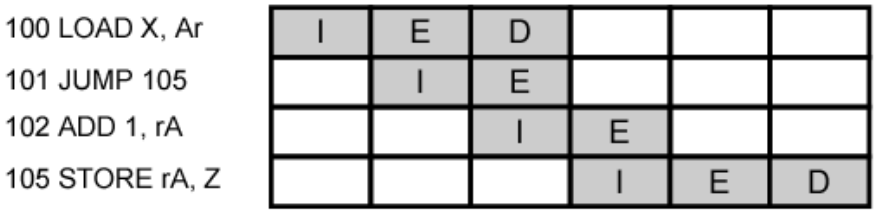
- The delayed branch means that
  - the instruction following the branch is always executed before the PC is modified to perform the branch



(a) Traditional Pipeline



(b) RISC Pipeline with Inserted NOOP



(c) Reversed Instructions

Delayed Branch and Delay Slot

- A delayed branch specifies that the
  - jump to a new location happens *after* the next instruction
  - that next instruction is the one (called, the delay slot) loaded after the branch
- Branch Delay Slot
  - Always fetch the instruction after the branch from the instruction cache, and always execute it, even if the branch is taken
- MIPS-X use a double branch delay slot
  - executes a pair of instructions following a branch instruction before the branch takes effect

Disadvantages of Delayed Branch

- Delayed branches have been criticized as a poor short-term choice in ISA design
  - Compilers typically have some difficulty finding logically independent instructions to place after the branch
  - Better to insert NOPs/NOOPs into the delay slots