What is CISC?

- CISC means Complex Instruction Set Computer chips that are easy to program and which make efficient use of memory. Since the earliest machines were programmed in assembly language and memory was slow and expensive, the CISC philosophy was commonly implemented in large computers as the PDP-11 and the DECsystem 10 and 20 machines.
- Most common microprocessor designs such as the Intel 80x86 and Motorola 68K series followed the CISC philosophy.
- CISC was developed to make compiler development simpler. It shifts most of the burden of generating machine instructions to the processor. For example, instead of having to make a compiler write long machine instructions to calculate a square-root, a CISC processor would have a built-in ability to do this.

CISC Attributes

CISC instructions sets have some common characteristics:

- A 2-operand format, where instructions have a source and a destination. Register to register, register to memory, and memory to register commands.
- Variable length instructions where the length often varies according to the addressing mode
- Instructions which require multiple clock cycles to execute.

E.g. Pentium is considered a modern CISC processor

Most CISC hardware architectures have several characteristics in common:

- Complex instruction-decoding logic, driven by the need for a single instruction to support multiple addressing modes.
- A small number of general purpose registers. This is the direct result of having instructions which can operate directly on memory and the limited amount of chip space not dedicated to instruction decoding, execution, and microcode storage.
- Several special purpose registers. Many CISC designs set special registers for the stack pointer, interrupt handling, and so on.
- A 'Condition code" register which is set as a side-effect of most instructions. This register reflects whether the result of the last operation is less than, equal to, or greater than zero and records if certain error conditions occur.

At the time of their initial development, CISC machines used available technologies to optimize computer performance.

- Microprogramniing is as easy as assembly language to implement, and much less expensive than hardwiring a control unit.
- The ease of microcoding new instructions allowed designers to make CISC machines upwardly compatible: a new computer could run the same programs as earlier computers because the new computer would contain a superset of the instructions of the earlier computers.
- As each instruction became more capable, fewer instructions could be used to implement a given task. This made more efficient use of the relatively slow main memory.
- Because microprogram instruction sets can be written to match the constructs of high-level languages, the compiler does not have to be as complicated.

Complex Instruction Set Computer (CISC) Characteristics

- Major characteristics of a CISC architecture
  - »1) A large number of instructions - typically from 100 to 250 instruction
  - »2) Some instructions that perform specialized tasks and are used infrequently
  - »3) A large variety of addressing modes - typically from 5 to 20 different modes
  - »4) Variable-length instruction formats
  - »5) Instructions that manipulate operands in memory (RISC in register)

What is RISC?

- **RISC?**
  RISC, or *Reduced Instruction Set Computer*. is a type of microprocessor architecture that utilizes a small, highly-optimized set of instructions, rather than a more specialized set of instructions often found in other types of architectures.
- **History**
  The first RISC projects came from IBM, Stanford, and UC-Berkeley in the late 70s and early 80s. The IBM 801, Stanford MIPS, and Berkeley RISC 1 and 2 were all designed with a similar philosophy which has become known as RISC. Certain design features have been characteristic of most RISC processors:
  - *one cycle execution time*: RISC processors have a CPI (clock per instruction) of one cycle. This is due to the optimization of each instruction on the CPU and a technique called PIPELINING
  - *pipelining*: a techique that allows for simultaneous execution of parts, or stages, of instructions to more efficiently process instructions;
  - *large number of registers*: the RISC design philosophy generally incorporates a larger number of registers to prevent in large amounts of interactions with memory

Reduced Instruction Set Computer (RISC)

- Major characteristics of a RISC architecture
  - »1) Relatively few instructions
  - »2) Relatively few addressing modes
  - »3) Memory access limited to load and store instruction
  - »4) All operations done within the registers of the CPU
  - »5) Fixed-length, easily decoded instruction format
  - »6) Single-cycle instruction execution
  - »7) Hardwired rather than microprogrammed control

– RISC Instruction

- Only use **LOAD** and **STORE** instruction when communicating between memory and CPU
- All other instructions are executed within the registers of the CPU without referring to memory

Program to evaluate $X = (A + B) * (C + D)$

LOAD	**R1, A**	$R1 \leftarrow M[A]$
LOAD	**R2, B**	$R2 \leftarrow M[B]$
LOAD	**R3, C**	$R3 \leftarrow M[C]$
LOAD	**R4, D**	$R4 \leftarrow M[D]$
ADD	**R1, R1, R2**	$R1 \leftarrow R1 + R2$
ADD	**R3, R3, R4**	$R3 \leftarrow R3 + R4$
MUL	**R1, R1, R3**	$R1 \leftarrow R1 * R3$
STORE	**X, R1**	$M[X] \leftarrow R1$

•Load instruction transfers the operand from memory to CPU Register.

•Add and Multiply operations are executed with data in the registers without accessing the memory.

•Result is then stored in the memory with store information.

- Other characteristics of a RISC architecture
  - 1) A relatively large number of registers in the processor unit
  - 2) Use of overlapped register windows to speed-up procedure call and return
  - 3) Efficient instruction pipeline
  - 4) Compiler support for efficient translation of high-level language programs into machine language programs

OVERLAPPED REGISTER WINDOWS

- There are three classes of registers:
  - Global Registers
    - Available to all functions
  - Window local registers
    - Variables local to the function
  - Window shared registers
    - Permit data to be shared without actually needing to copy it
- Only one register window is active at a time
  - 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
- When a function calls a new function, the high numbered registers of the calling function window are shared with the called function as the low numbered registers in its register window
- This way the caller's high and the called function's low registers overlap and can be used to pass parameters and results

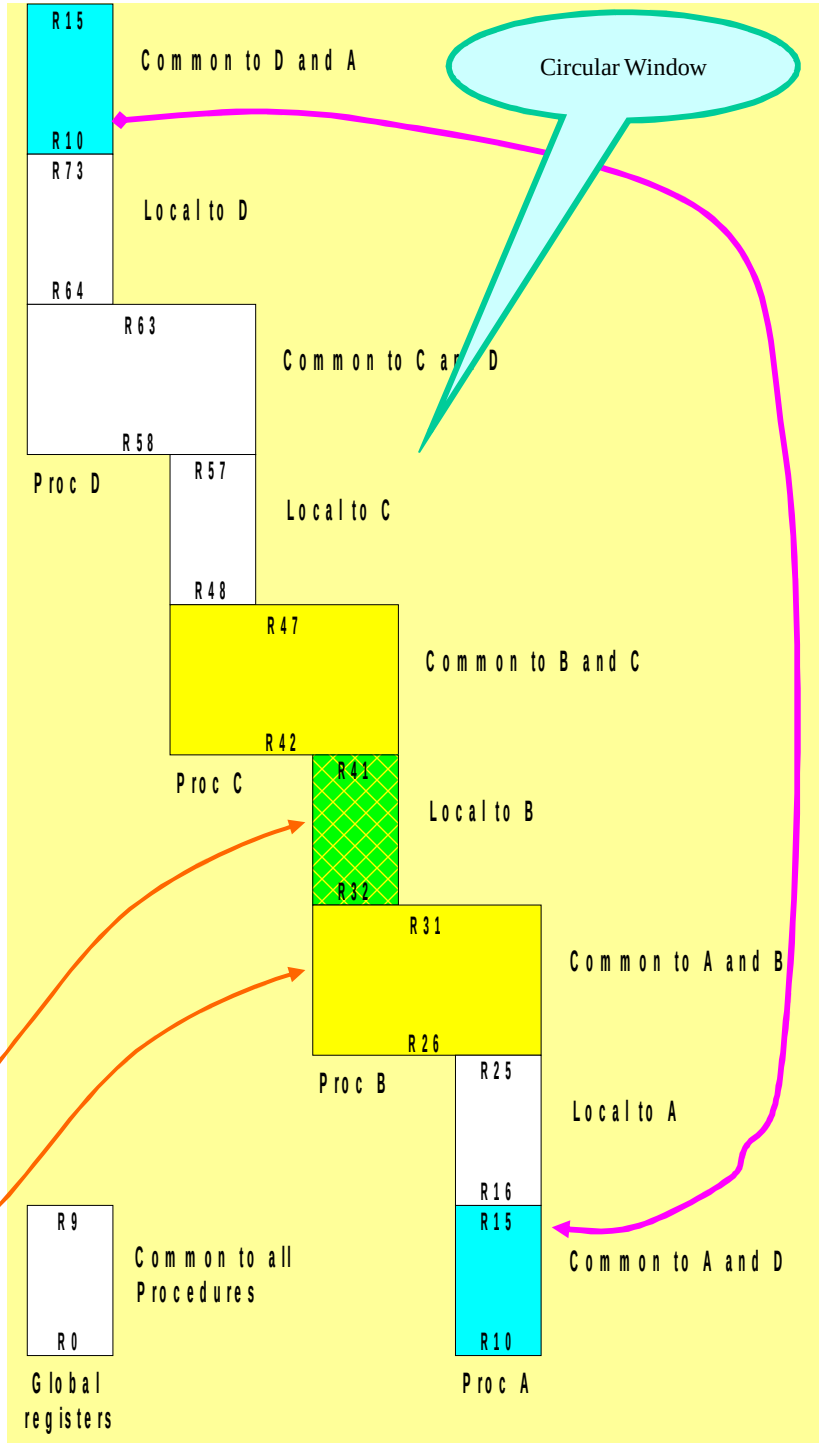
OVERLAPPED REGISTER WINDOWS

- In addition to the overlapped register windows, the processor has some number of registers, *G*, that are global registers
  - This is, all functions can access the global registers.

- The advantage of overlapped register windows is that the processor does not have to push registers on a stack to save values and to pass parameters when there is a function call
  - Conversely, pop the stack on a function return
- This saves
  - Accesses to memory to access the stack.
  - The cost of copying the register contents at all
- And, since function calls and returns are so common, this results in a significant savings relative to a stack-based approach

- Total 74 registers : R0 - R73
  - –R0 - R9 : Global registers
  - –R10 - R63 : 4 windows
    - »Window A
    - »Window B
    - »Window C
    - »Window D

10 Local registers
+
2 sets of 6 registers
(common to adjacent windows)

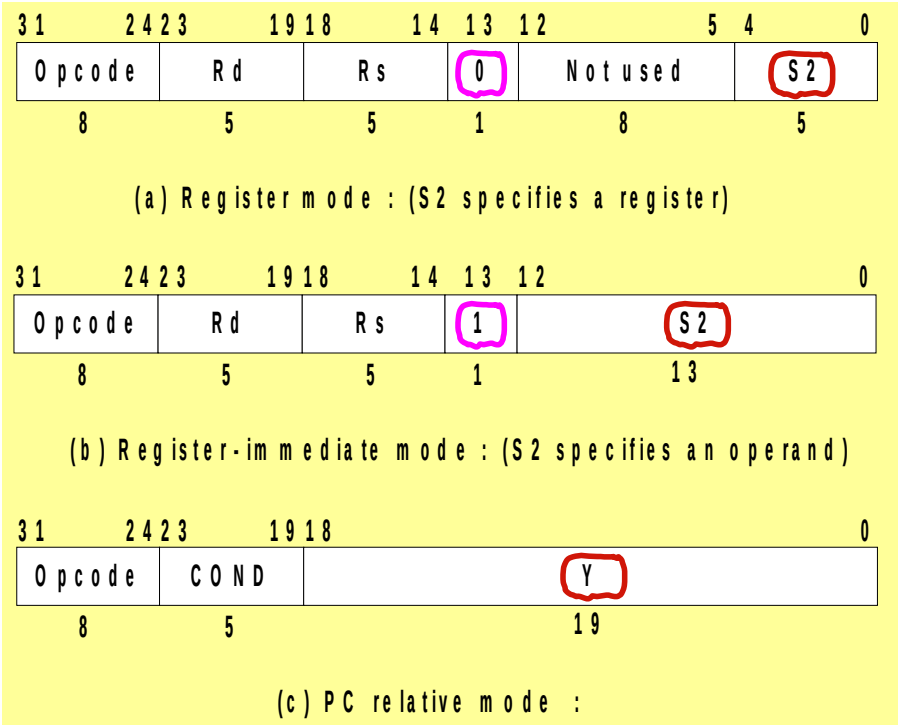


- ***Example)*** Procedure A calls procedure B
  - R26 - R31
    - » Store parameters for procedure B
    - » Store results of procedure B
  - R16 - R25 : Local to procedure A
  - R32 - R41 : Local to procedure B
- Window Size = L + 2C + G = 10 + (2 X 6) + 10 = 32 registers
- Register File (total register) = (L + C) X W + G = (10 + 6) X 4 + 10 = 74 registers
  - 여기서, **G** : Global registers = *10*
    L : Local registers = *10*
    C : Common registers = *6*
    W : Number of windows = *4*

– Berkeley RISC I

- RISC Architecture 의 기원 : 1980 년대 초
  - Berkeley RISC project : first project = **Berkeley RISC I**
  - Stanford MIPS project
- Berkeley RISC I
  - 32 bit CPU, 32 bit instruction format, 31 instruction
  - 3 addressing modes : register, immediate, relative to PC

- Instruction Set : ***Tab. 8-12***
- Instruction Format : ***Fig. 8-10***
- Register Mode : ***bit 13 = 0***
  - » S2 = register
  - » ***Example)*** **ADD R22, R21, R23**
    - **ADD Rs, S2, Rd :** Rd = Rs + S2
- Register Immediate Mode : ***bit 13 = 1***
  - » S2 = sign extended 13 bit constant
  - » ***Example)*** **LDL (R22)#150, R5**
    - **LDL (Rs)S2, Rd :** Rd = M[R22] + 150
- PC Relative Mode
  - » Y = 19 bit relative address
  - » ***Example)*** **JMPR COND, Y**
    - Jump to PC = PC + Y
  - » CWP (Current Window Pointer)
    - CALL, RET?stack pointer ??



- RISC Architecture Originator

Architecture	Originator	Licensees
Alpha	DEC	Mitsubishi, Samsung
MIPS	MIPS Technologies	NEC, Toshiba
PA-RISC	Hewlett Packard	Hitachi, Samsung
PowerPC	Apple, IBM, Motorola	Bul
Sparc	Sun	Fujitsu, Hyundai
i960	Intel	Intel only (Embedded Controller)

CISC versus RISC

CISC	RISC
Emphasis on hardware	Emphasis on software
Includes multi-clock complex instructions	Single-clock, reduced instruction only
Memory-to-memory: "LOAD" and "STORE" incorporated in instructions	Register to register: "LOAD" and "STORE" are independent instructions
Small code sizes, high cycles per second	Low cycles per second, large code sizes
Transistors used for storing complex instructions	Spends more transistors on memory registers