

SYSC3601  
Microprocessor Systems

Unit 13:  
RISC/CISC Comparison

# Topics/Reading

---

## 1. Review:

- 8086 Instruction Format and addressing modes.
- 68000 Instruction Format and addressing modes.

## 2. RISC vs. CISC comparison

- Motivation for RISC
- Quantitative analysis of program execution
- RISC-enabled technologies

## 3. Power PC Instruction Format and addressing modes.

# RISC/CISC Comparison

---

- **8086 Instruction Format and addressing modes.**
- 1 to 7 bytes in size, any alignment.
- 16-bit Instruction Format:

Opcode++ 1-2 bytes	MOD-REG-R/M 0-1 byte	Displacement 0-2 bytes	Immediate 0-2 bytes
-----------------------	-------------------------	---------------------------	------------------------

- Encoded for space savings (see next slide).
- Opcode is typically 1 byte in size with the first six bits being the actual opcode.
- Opcodes accessing memory use MOD-REG-R/M byte to specify registers and addressing modes.
- In general, one operand is a register and the second operand can be memory.

# RISC/CISC Comparison

---

Table 1: MOV – Move Data

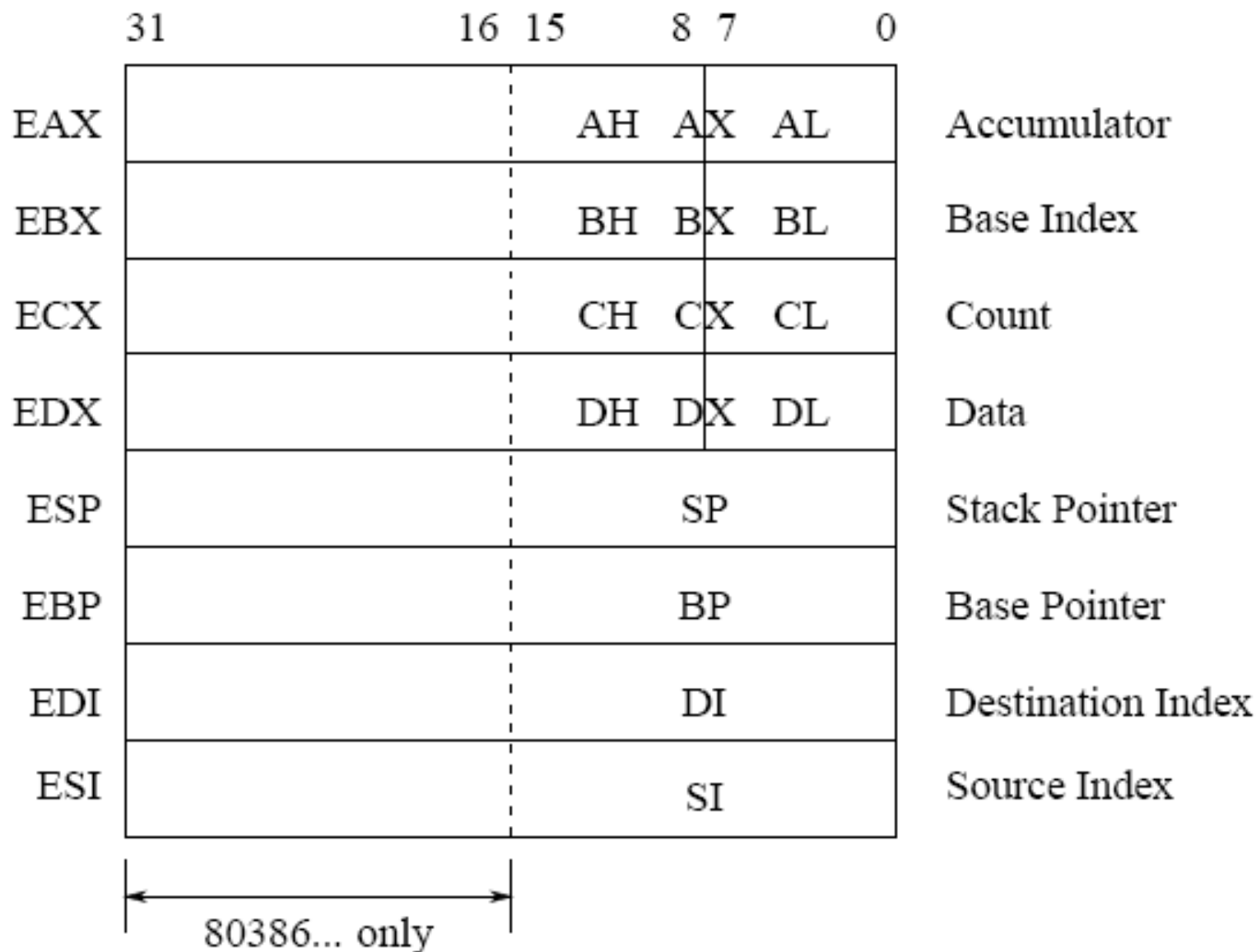
100010dw mod-reg-r/m disp		Clocks	
Format	Example	8086	8088
MOV reg,reg	MOV CX,DX	2	2
MOV mem,reg	MOV DATA,CX	9 + ea	13 + ea
MOV reg,mem	MOV CX,DATA	10 + ea	12 + ea

1100011w mod-000-r/m disp data		Clocks	
Format	Example	8086	8088
MOV mem,imm	MOV DATA,33H	10 + ea	14 + ea

1011wrrr data		Clocks	
Format	Example	8086	8088
MOV reg,imm	MOV CX,13H	4	4

101000dw disp		Clocks	
Format	Example	8086	8088
MOV mem,acc	MOV DATA,AX	10	14
MOV acc,mem	MOV AX,DATA	10	14

# Execution Unit – Multipurpose Registers



# RISC/CISC Comparison

---

- **Addressing modes**

Mode	Example	Size
Register addressing	MOV AX,BX	2
Imm addressing	MOV CX,3AH	3-4
Direct addressing	MOV [1234],AX	4
Register Indirect	MOV [BX],CL	2
Base plus index	MOV [BX+SI],BP	2
Register relative	MOV CL,[BX+4]	3-4
Base relative plus index	MOV ARRAY[BX+SI],DX	3-4
Immediate	MOV AX,1234	?

# History of IA - 32

---

- 1978: The Intel 8086 is announced (16 bit architecture)
- 1980: The 8087 floating point coprocessor is added
- 1982: The 80286 increases address space to 24 bits, +instructions
- 1985: The 80386 extends to 32 bits, new addressing modes
- 1989-1995: The 80486, Pentium, Pentium Pro add a few instructions (mostly designed for higher performance)
- 1997: 57 new “MMX” instructions are added, Pentium II
- 1999: The Pentium III added another 70 instructions (SSE)
- 2001: Another 144 instructions (SSE2)
- 2003: AMD extends the architecture to increase address space to 64 bits, widens all registers to 64 bits and other changes (AMD64)
- 2004: Intel capitulates and embraces AMD64 (calls it EM64T) and adds more media extensions

“This history illustrates the impact of the *golden handcuffs of compatibility*”

“adding new features as someone might add clothing to a packed bag”

“an architecture that is difficult to explain and impossible to love”

# RISC/CISC Comparison

---

- **68000 Instruction Format and addressing modes.**
- 1 to 11 words, aligned (mov instruction, 68020 addressing)
- Encoded for space savings.
- Opcode is 1 word in size with the first four bits being the actual opcode.
- In general, one operand is a register and the second operand can be memory (mov instruction allows two memory operands).

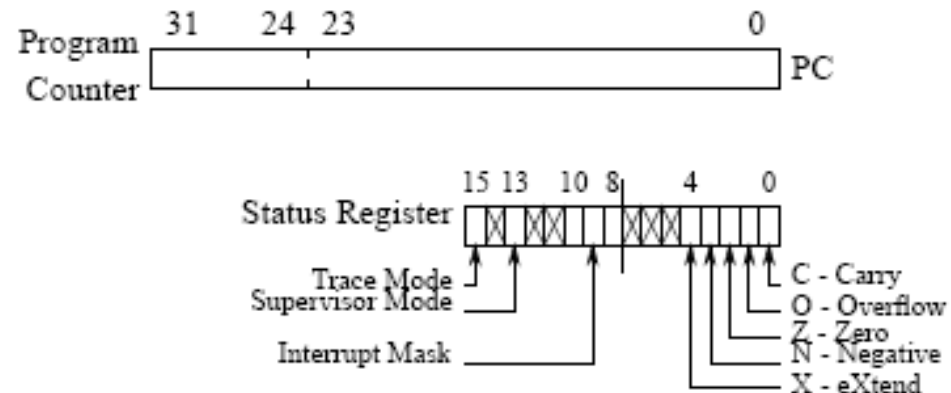
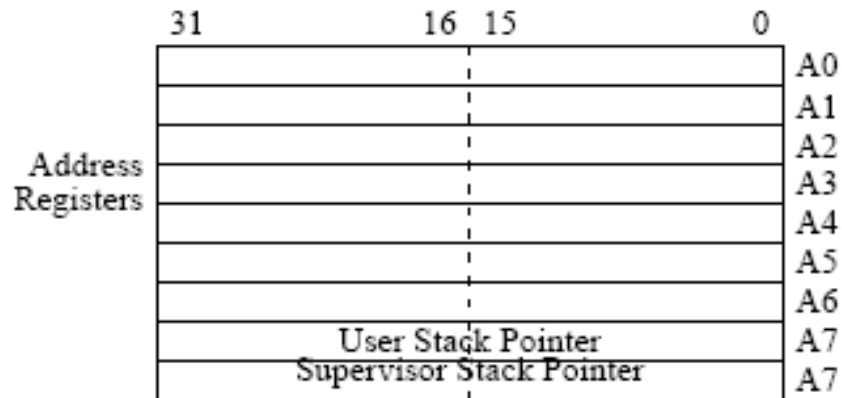
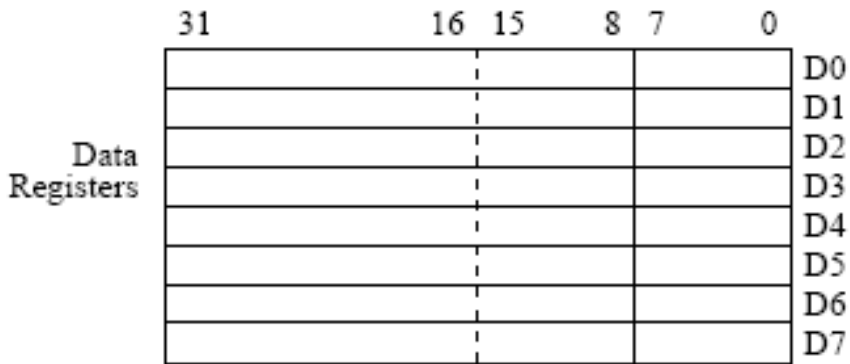
Opcode 1 word	Special 0-2 words	Imm/Source 0-6 words	Destination 0-6 words
------------------	----------------------	-------------------------	--------------------------

Figure 2: Instruction Word Specification Format



# RISC/CISC Comparison

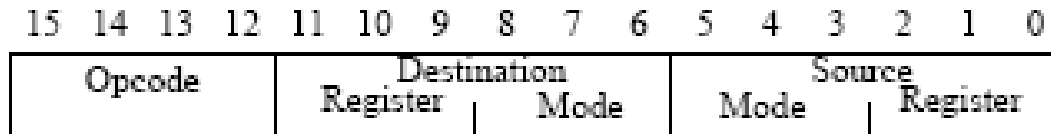
- **68000 General Registers**



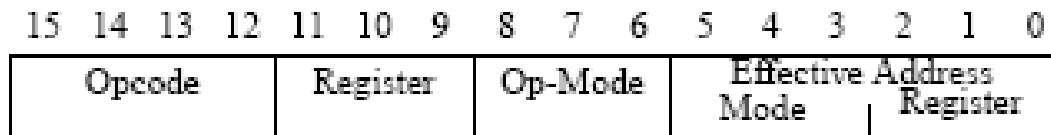
# RISC/CISC Comparison

- 68000 Instruction Encoding

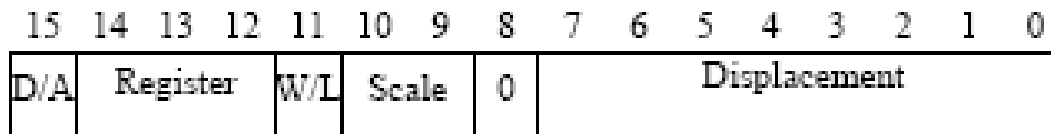
- Six values for “op-mode” for size (byte, word, long), and direction.
- Other two values represent “other” instructions!



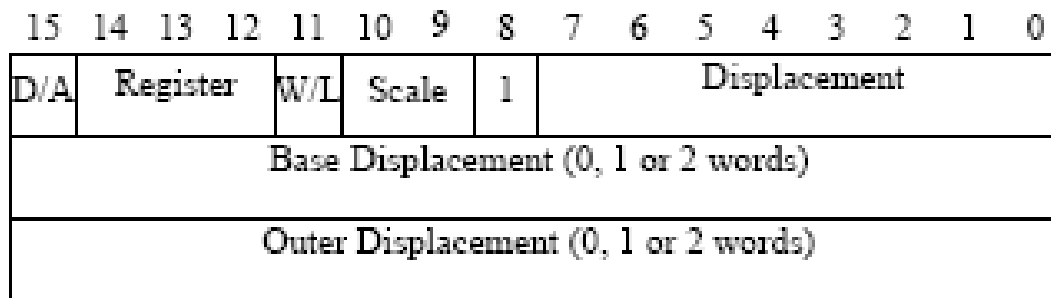
MOV instruction



General Format



Brief Extension Word Format



# RISC/CISC Comparison

---

- 68000 Addressing modes

Addr Mode	Mode	Register	Addr Mode	Mode	Register
Dn	000	reg-num:Dn	(xxx).W	111	000
An†	001	reg-num:An	(xxx).L	111	001
(An)	010	reg-num:An	#<data>	111	100
(An)+	011	reg-num:An			
-(An)	100	reg-num:An			
(d16,An)	101	reg-num:An	(d16,PC)	111	010
(d8,An,Xn)	110	reg-num:An	(d8,PC,Xn)	111	011

†Word and long only.

# RISC/CISC Comparison

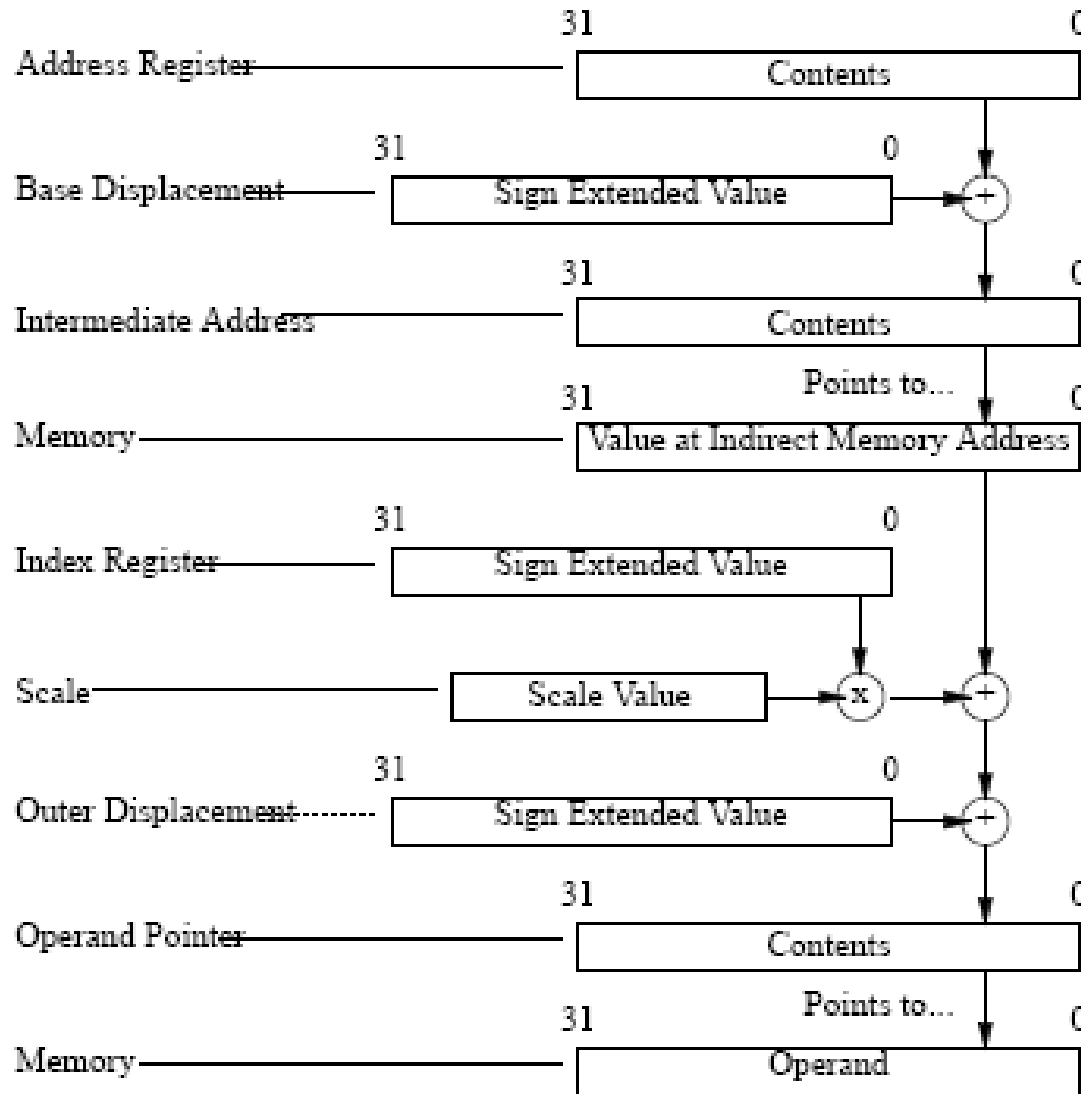
---

- 68020, 030, 040 Addressing modes

Addr Mode	Mode	Register
(bd,An,Xn)	110	reg-num:An
([bd,An,Xn],od)	110	reg-num:An
([bd,An],Xn,od)	110	reg-num:An
(bd,PC,Xn)	111	011
([bd,PC,Xn],od)	111	011
([bd,PC],Xd,od)	111	011

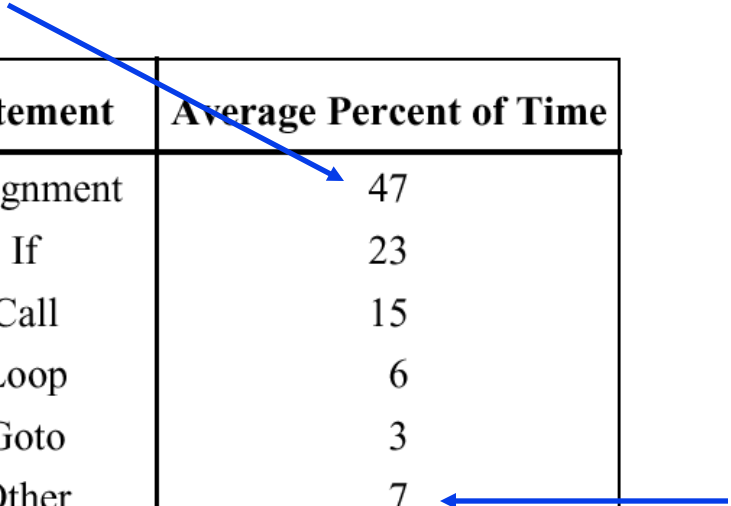
# RISC/CISC Comparison

- Example: Post Index:  $EA = (An+bd) + Xn * SCALE + od$



# RISC/CISC Comparison

- Prior to late 1970s, computer architects just focused on the complexity of instructions and addressing modes to improve the computer performance. → **Complex Instruction Set Computer (CISC)**
- Knuth showed that most of the statements used in a typical Fortran program are simple instructions.



Statement	Average Percent of Time
Assignment	47
If	23
Call	15
Loop	6
Goto	3
Other	7

**Arithmetic and other powerful instructions account for only 7%.**

- Later research by Hennessy and Patterson showed that most complex instructions and addressing modes are not used in a typical program. They coined the use of program analysis and **benchmarking** to evaluate the impact of architecture upon performance.

# RISC/CISC Comparison

---

- More quantitative metrics: All these metrics show that there is no or little payoff in increasing the complexity of the instructions.

	Percentage of Number of Terms in Assignments	Percentage of Number of Locals in Procedures	Percentage of Number of Parameters in Procedure Calls
0	–	22	41
1	80	17	19
2	15	20	15
3	3	14	9
4	2	8	7
≥ 5	0	20	8

- Moreover, analysis showed that compilers usually do not take advantage of complex instructions and addressing modes.
- These observations, brought an evolution from CISC to **Reduced Instruction Set Computer (RISC)**.
- The focus is to **make the frequent case fast and simple**.
  - make assignments fast.
  - use only **LOAD** and **STORE** to access memory.

# Quantitative Analysis of Program Execution

---

- **Load/Store machine:** is a typical RISC architecture. Only these two instructions can communicate with memory. Others should use registers.
  - Access to memory can be overlapped as there is less side effect.
  - A large number of registers is needed.
  - ➔ A simple instruction set results in a simpler CPU, which frees up space on microprocessor to be used for other purposes, such as registers, and caches.



# CISC vs RISC

---

- **CISC: Complex Instruction Set Computer**
  - Many complex instructions and addressing modes. Suitable for when the memory access times were very high and number of registers in CPU low.
  - Some instructions take many steps to execute.
  - Not suitable for **pipelining** because it has different instructions with different complexity.
- **RISC: Reduced Instruction Set Computer**
  - All instructions are of fixed length (typically 32 bits).
    - This simplifies fetch and decoding.
  - Few, simple instructions and addressing modes.
    - A **Load-Store** architecture. All operands must be in registers.
  - There should be a large number of registers.
  - A program may have more number of instructions than CISC but runs faster as the instructions are simpler.
  - Use hardwired technique. Avoid microcode.
  - Let the compiler do the complex things.
  - RISC CPUs represent the vast majority of all CPUs in use (embedded CPUs are almost always RISC design).

# CISC vs RISC (Cont'd)

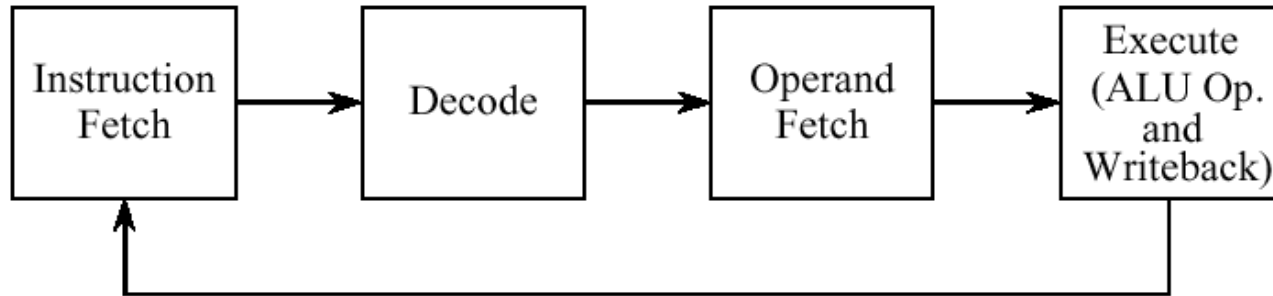
---

- Other RISC benefits:
  - Allows **prefetching** instructions to hide the latency of fetching instructions and data.
  - **Pipelining**: begin execution of an instruction before the previous instructions have completed.
    - Instructions can be issued into the pipeline at a rate of one per clock cycle. Pipelining allows different instructions to use different parts of the execution unit on each clock cycle and an instruction can be completed in every clock cycle.
  - **Superscalar operation**: issuing more than one instruction simultaneously (**Instruction-level parallelism: ILP**)
  - **Delayed loads, stores, and branches**: Operands may not be available when an instruction attempts to access them.
  - **Register windows**: ability to switch to a different set of CPU registers with a single command. Alleviates procedure call/return overhead.

# Pipelining

---

- Pipelining takes an assembly line approach to instruction execution.



- One instruction enters the pipeline at each clock cycle.
- At the end of each stage, with the clock, output is latched and is passed to the next stage.
- As soon as the pipeline is full, the instructions are completed in every clock cycle.
- Different pipelines may have different number of stages.

# Pipelining (Cont'd)

- What if the instruction is a branch? As soon as the pipeline is full and a branch is taken, then the pipeline has to be **flushed** by filling it with no-operations (nops). They are also called **pipeline bubbles**. This also allows us to delay until the branch is known to be taken or not taken.
- When a LOAD or STORE occurs, we may have to expand the execute phase from one clock cycle to two clock cycles. This is also known as **delayed load**.

	Time							
	1	2	3	4	5	6	7	8
Instruction Fetch	addcc	ld	srl	subcc	be	nop	nop	nop
Decode		addcc	ld	srl	subcc	be	nop	nop
Operand Fetch			addcc	ld	srl	subcc	be	nop
Execute				addcc	ld	srl	subcc	be
Memory Reference						ld		

Wait for branch by inserting NOPs.

The other approach is to do **branch prediction** or speculatively execute the instructions after the branch.

Bubbles are also inserted when an interrupt occurs.

# Superscalar and VLIW Architectures

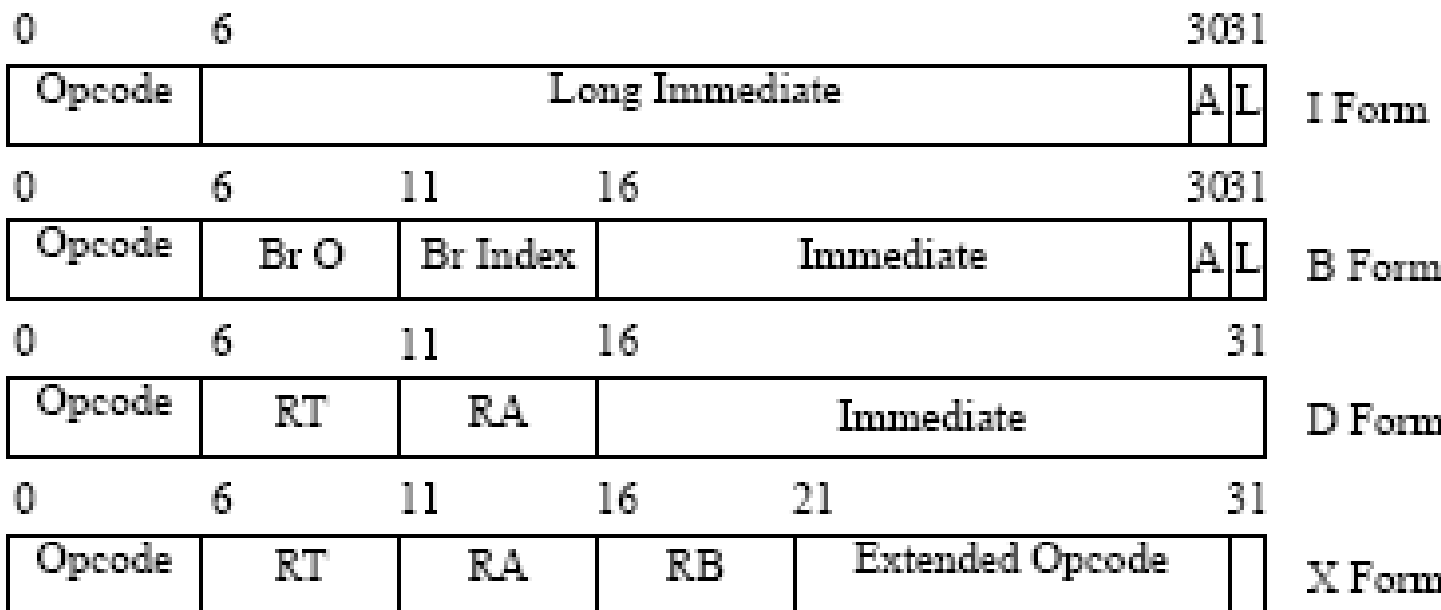
---

- **Superscalar architecture**: might have one or more separate integer units (IUs), floating point units (FPUs), and branch processing units (BPUs). Thus, with separate functional units, several instructions are executed at the same time.
  - Instructions should be scheduled into various execution units and might be executed **out of order**.
  - **Out-of-order execution**: means that instructions need to be examined prior to dispatching them to an execution unit, not only to determine which unit should execute them but also to determine whether executing them out of order would result in an incorrect program because of dependencies between the instructions.
    - Out-of-order issue, in-order issue, retiring instructions
- **Very Long Instruction Word (VLIW)**: where multiple operations are packed into a single instruction word. The compiler is responsible to organize multiple instructions into the instruction word.

# RISC/CISC Comparison – PowerPC

---

- PowerPC Instruction Format
  - 4 bytes in size. Aligned to long-word.
  - Instruction format:



# RISC/CISC Comparison – PowerPC

---

- Addressing modes

1. Register Relative                     $EA \leftarrow (RA|0) + D$

2. Base Plus Index     $EA \leftarrow (RA|0) + RB$

3. Register Relative and Update       $EA \leftarrow (RA|0) + D; RA \leftarrow EA$

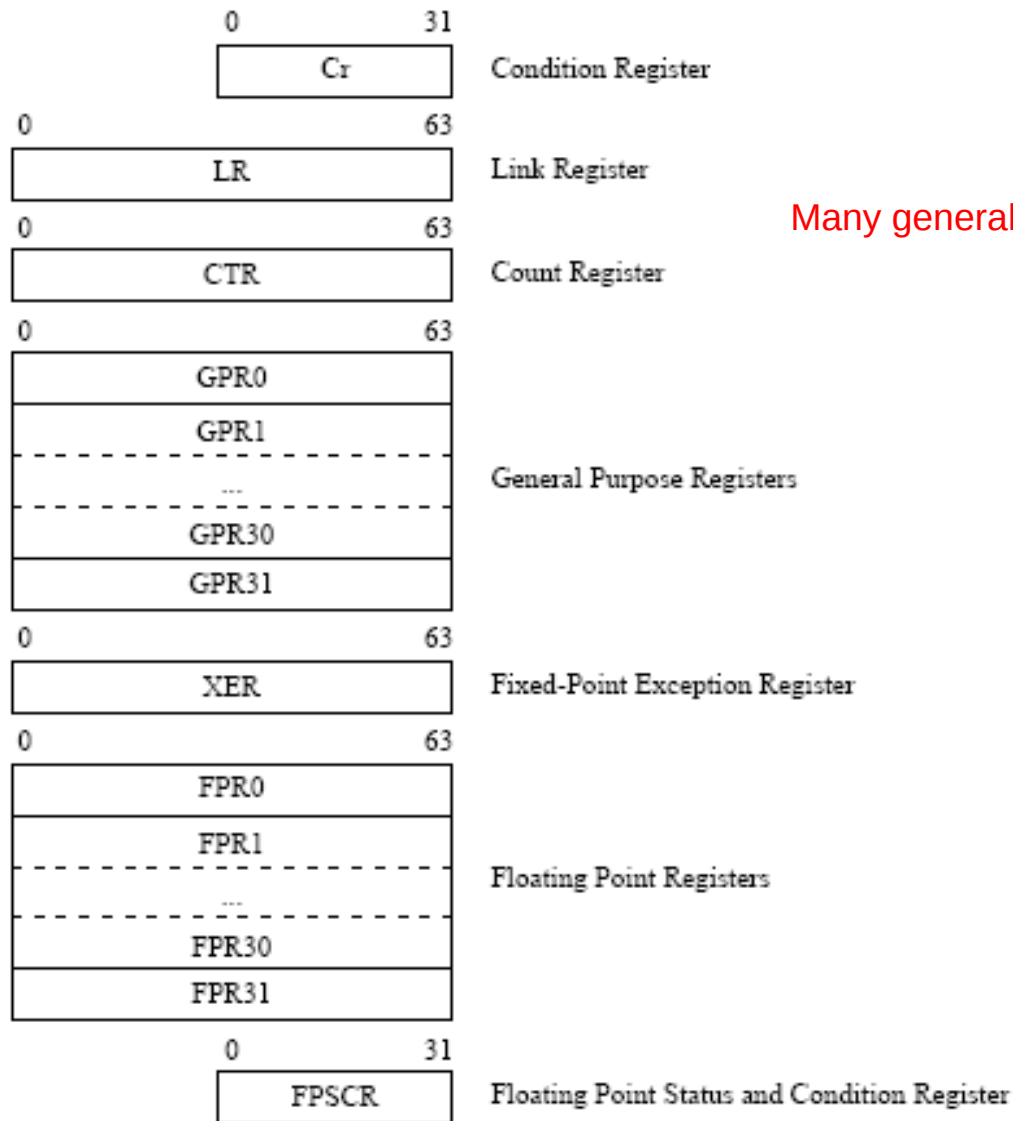
4. Base Plus Index and Update         $EA \leftarrow (RA|0) + RB; RA \leftarrow EA$

5. Immediate

- Keep it simple!

# RISC/CISC Comparison – PowerPC

- General Registers



Many general purpose registers