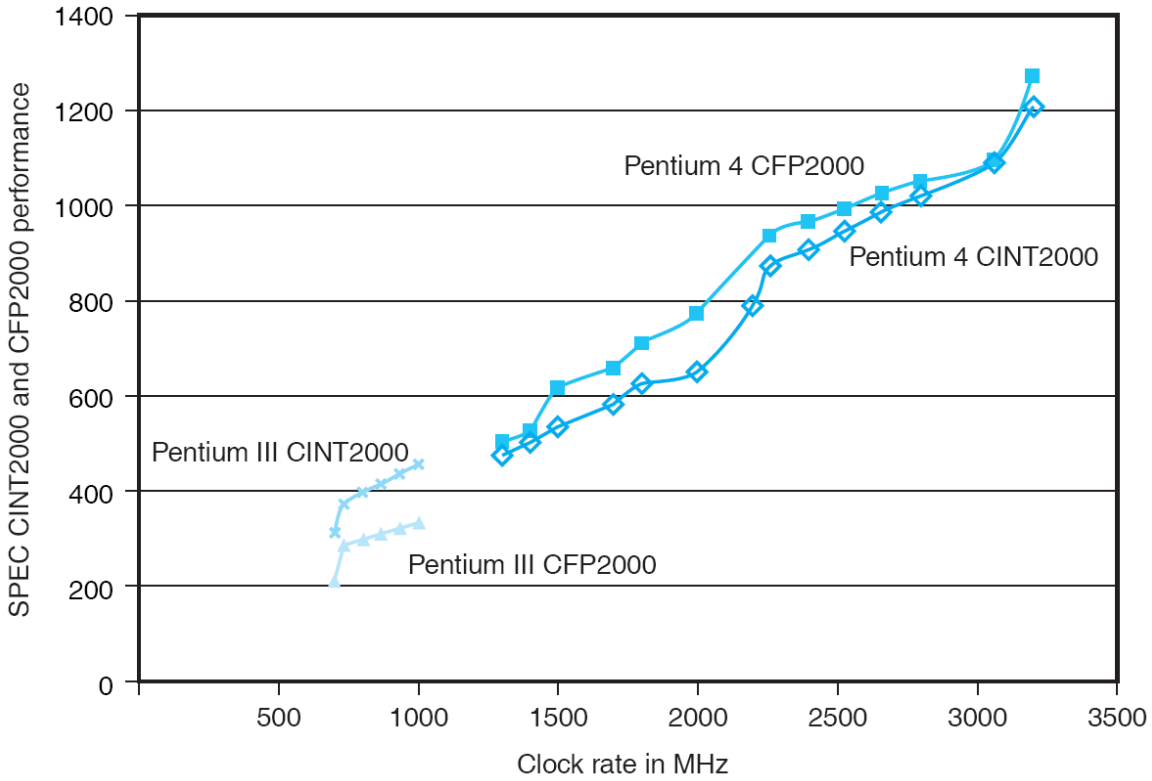
# EECE 321: Computer Organization

Mohammad M. Mansour
*Dept. of Electrical and Compute Engineering*
*American University of Beirut*

Lecture 17: Performance

# SPEC Ratings for Pentium Processors

- Dell Precision desktop computers




- Does doubling the clock rate double the performance?
- Can a machine with a slower clock rate have better performance?

# Amdahl's Law

- **Common Pitfall**: Expecting the improvement of one aspect of a machine to increase performance by an amount proportional to the size of the improvement.

- **Example**:
  **"Suppose a program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time. How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?"**

- **Solution**:
  - Execution time before improvement = $E_b$ = 80 + 20 sec.
  - Execution time after improvement = $E_a$ = 100/4 = 25 sec.
  - $E_a = T_{mult}$ + Time unaffected = $T_{mult}$ + 20 =>$T_{mult}$ = 5
  - Improvement in speed of multiplication: 100 ---> 5 or 20 times faster.

  - How about making it 5 times faster?
    $E_a$ = 100/5 = $T_{mult}$ + 20 =>$T_{mult}$ = 0.
    Hence no improvement in multiplication alone can make application run 5 times faster.

# Amdahl's Law

- General formula for Amdahl's law:

$$Exec.\ Time\ After\ Improvement = Exec.\ Time\ Unaffected + \frac{Exec.\ Time\ Affected}{Amount\ of\ improvement}$$

- *Principle: Make the common case fast*
  - *This is better than optimizing the rare case.*

# Example 1

- Suppose we enhance a machine by making all floating-point instructions run five times faster. If the execution time of some benchmark before the floating-point enhancement is 10 seconds, what will the speedup be if half of the 10 seconds is spent executing floating-point instructions?

  **Solution**:

$$E_b = 10\ sec.$$

$$E_a = T_{unaffected} + \frac{T_{affected}}{Improvement} = 5 + \frac{5}{5} = 6\ sec.$$

$$Speedup = \frac{Perf_a}{Perf_b} = \frac{E_b}{E_a} = \frac{10}{6} = 1.67$$

# Example 2

- We are looking for a benchmark to show off the new floating-point unit described in the previous example, and want the overall benchmark to show a speedup of 3. One benchmark we are considering runs for 100 seconds with the old floating-point hardware. How much of the execution time would floating-point instructions have to account for in this program in order to yield our desired speedup on this benchmark?

- **Solution**:

$$E_b = 100 = T_{unaffected} + T_{FP}$$

$$E_a = \frac{E_b}{3} = \frac{100}{3} = T_{unaffected} + \frac{T_{FP}}{5}$$

$$=> \ T_{FP} = \frac{250}{3} \ sec.$$

$$T_{unaffected} = \frac{50}{3} \ sec.$$

# Points to Remember

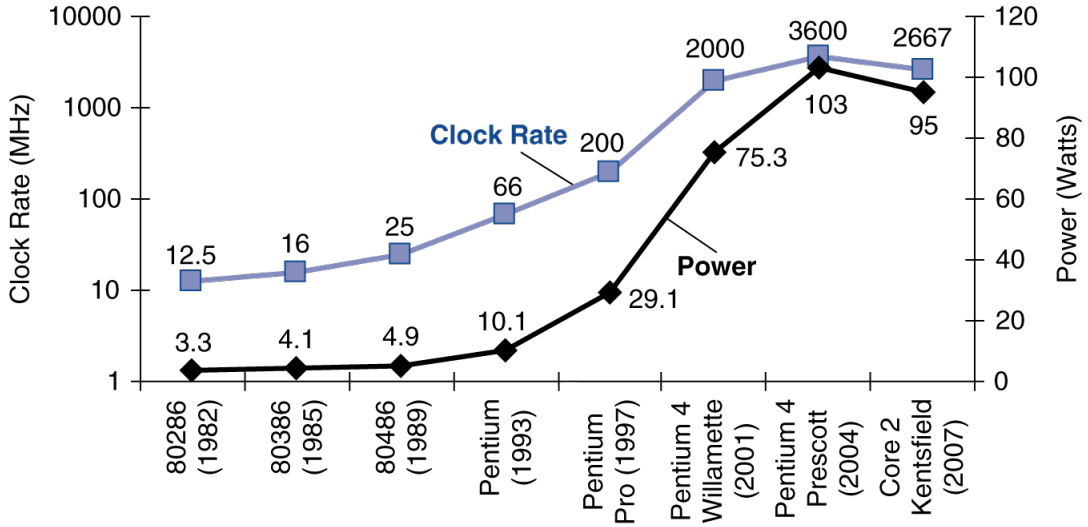
- Performance is specific to a particular set of programs
  - Total execution time is a consistent summary of performance

- For a given architecture performance increases come from:
  - Increases in clock rate (without adverse CPI affects)
  - Improvements in processor organization that lower CPI
  - Compiler enhancements that lower CPI and/or instruction count

- Pitfall: Expecting improvement in one aspect of a machine's performance to affect the total performance

- You should not always believe everything you read! Read carefully!

# Power Consumption --- Barrier to Performance Scaling




- In CMOS IC technology

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

×30 | 5V → 1V | ×1000

# Reducing Power

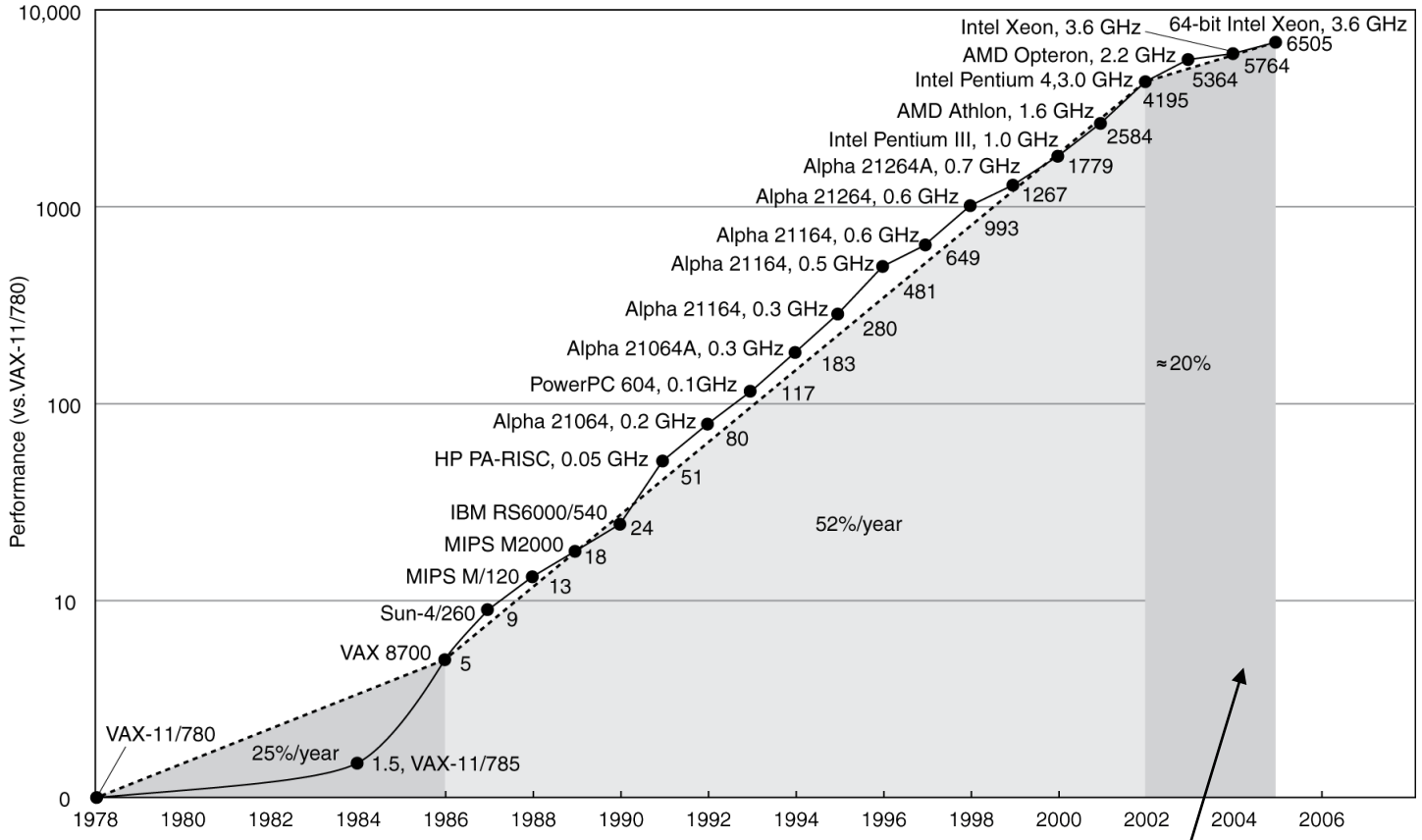
- Suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times V_{old}^{\ 2} \times F_{old}} = 0.85^4 = 0.52$$

- The power wall
  - We can't reduce voltage further
  - We can't remove more heat

- How else can we improve performance?

# Uniprocessor Performance




Constrained by power, instruction-level parallelism, memory latency

# Multiprocessors

- Answer: Use multi-cores

- Multi-core microprocessors
  - More than one processor core per chip

- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
  - Hard to do
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization

# SPEC Power Benchmark

- SPEC also offers benchmark sets to test server power consumption
  - SPECpower benchmarks

- Power consumption of server at different workload levels
  - Performance: ssj_ops/sec (business operations per second)
  - Power: Watts (Joules/sec)

$$\text{Overall ssj\_ops per Watt} = \left(\sum_{i=0}^{10} \text{ssj\_ops}_i\right) \Big/ \left(\sum_{i=0}^{10} \text{power}_i\right)$$

# SPECpower_ssj2008 Running on an AMD Processor

| Target Load % | Performance (ssj_ops/sec) | Average Power (Watts) |
|---|---|---|
| 100% | 231,867 | 295 |
| 90% | 211,282 | 286 |
| 80% | 185,803 | 275 |
| 70% | 163,427 | 265 |
| 60% | 140,160 | 256 |
| 50% | 118,324 | 246 |
| 40% | 920,35 | 233 |
| 30% | 70,500 | 222 |
| 20% | 47,126 | 206 |
| 10% | 23,066 | 180 |
| 0% | 0 | 141 |
| Overall sum | 1,283,590 | 2,605 |
| ∑ssj_ops/ ∑power | | 493 |

# Fallacy: Low Power at Idle

- Look back at X4 power benchmark
  - At 100% load: 295W
  - At 50% load: 246W (83%)
  - At 10% load: 180W (61%)

- Google data-center
  - Mostly operates at 10% – 50% load
  - At 100% load less than 1% of the time

- Consider designing processors to make power proportional to load

# Intel x86 ISA

# The Intel x86 ISA

- An example of a CISC ISA
  - Provides more powerful instructions than MIPS
  - Goal is the reduce number of instructions executed by a program
  - Cons: Complex; slow because instructions take longer to execute

- Evolution with backward compatibility
  - 8080 (1974): 8-bit microprocessor
    - Accumulator, plus 3 index-register pairs
  - 8086 (1978): 16-bit extension to 8080
    - Complex instruction set (CISC)
  - 8087 (1980): floating-point coprocessor
    - Adds FP instructions and register stack
  - 80286 (1982): 24-bit addresses, Memory Management Unit (MMU)
    - Segmented memory mapping and protection
  - 80386 (1985): 32-bit extension (now IA-32)
    - Additional addressing modes and operations
    - Paged memory mapping as well as segments

# The Intel x86 ISA

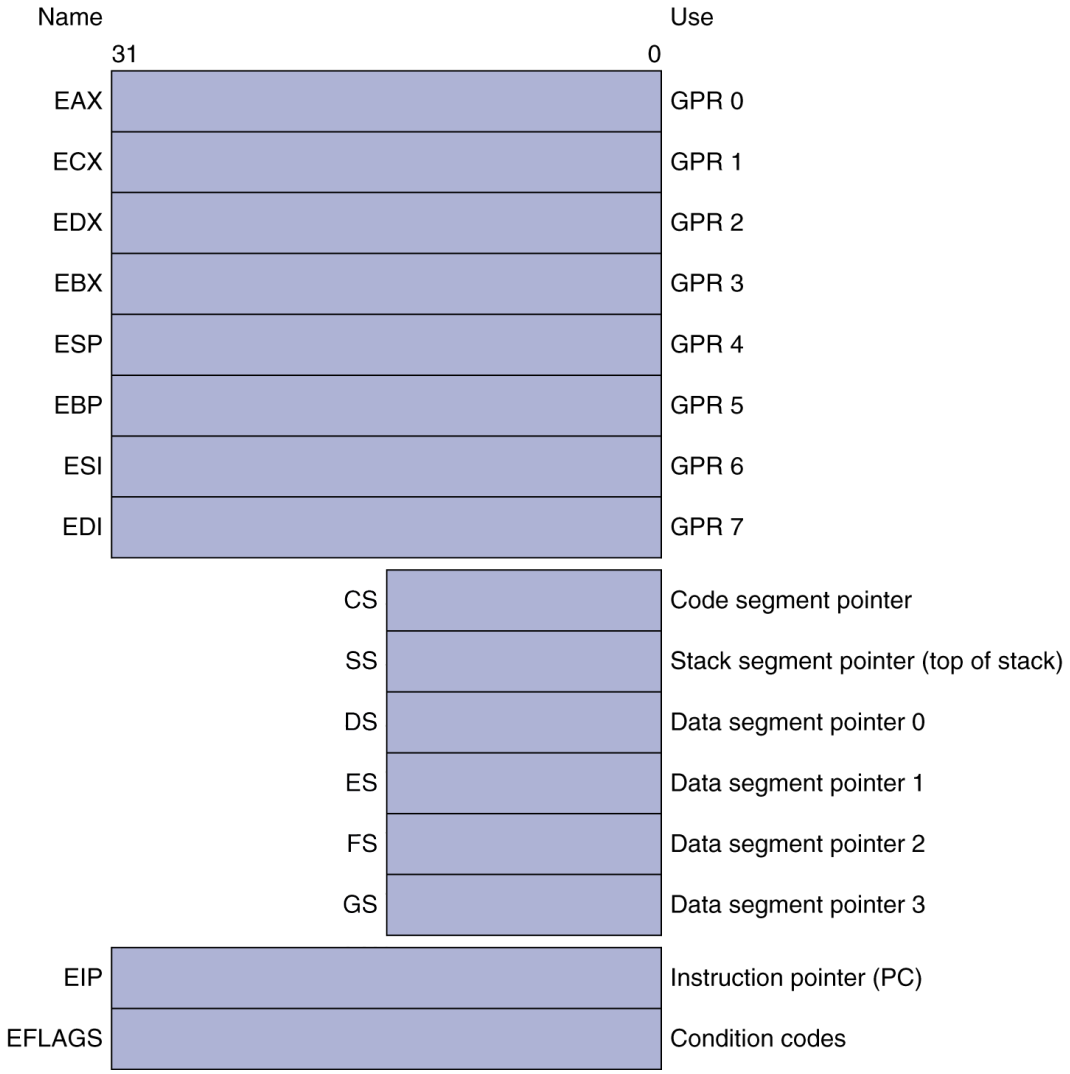
- Further evolution…
  - i486 (1989): pipelined, on-chip caches and FPU
    - Compatible competitors: AMD, Cyrix, …
  - Pentium (1993): superscalar, 64-bit datapath
    - Later versions added MMX (Multi-Media eXtension) instructions
    - The infamous FDIV bug
  - Pentium Pro (1995), Pentium II (1997)
    - New microarchitecture
  - Pentium III (1999)
    - Added SSE (Streaming SIMD Extensions) instructions and associated registers
    - 128-bit registers that can pack 4 single-precision FP numbers
  - Pentium 4 (2001)
    - New microarchitecture
    - Added SSE2 instructions: Pairs of double-precision FP numbers to operate in parallel

# The Intel x86 ISA

- And further…
  - AMD64 (2003): extended architecture to 64 bits
    - EM64T – Extended Memory 64 Technology (2004)
    - AMD64 adopted by Intel (with refinements)
    - Added SSE3 instructions
  - Intel Core (2006)
    - Added SSE4 instructions, virtual machine support
  - AMD64 (announced 2007): SSE5 instructions
    - Intel declined to follow, instead…
  - Advanced Vector Extension (announced 2008)
    - Longer SSE registers, more instructions

- If Intel didn't extend with compatibility, its competitors would!
  - Technical elegance ≠ market success

# Basic x86 Registers



# Basic x86 Addressing Modes

- Two operands per instruction

| Source/dest operand | Second source operand |
|---|---|
| Register | Register |
| Register | Immediate |
| Register | Memory |
| Memory | Register |
| Memory | Immediate |

- Memory addressing modes
  - Address in register
  - Address = $R_{base}$ + displacement
  - Address = $R_{base} + 2^{scale} \times R_{index}$ (scale = 0, 1, 2, or 3)
  - Address = $R_{base} + 2^{scale} \times R_{index}$ + displacement

# Basic x86 Addressing Modes

- The Base-plus-Scaled-Index addressing mode, not found in ARM or MIPS, is included to avoid the multiplies by 4 (scale factor of 2) to turn an index in a register into a byte address.
  - A scale factor of 1 is used for 16-bit data
  - A scale factor of 3 for 64-bit data
  - A scale factor of 0 means the address is not scaled
  - If the displacement is longer than 16 bits in the second or fourth modes, then the MIPS equivalent mode would need two more instructions

| Mode | Description | Register restrictions | MIPS equivalent |
|---|---|---|---|
| Register indirect | Address is in a register. | Not ESP or EBP | `lw $s0,0($s1)` |
| Based mode with 8- or 32-bit displacement | Address is contents of base register plus displacement. | Not ESP | `lw $s0,100($s1) # <= 16-bit # displacement` |
| Base plus scaled index | The address is Base + ($2^{Scale}$ x Index) where Scale has the value 0, 1, 2, or 3. | Base: any GPR Index: not ESP | `mul $t0,$s2,4`<br>`add $t0,$t0,$s1`<br>`lw $s0,0($t0)` |
| Base plus scaled index with 8- or 32-bit displacement | The address is Base + ($2^{Scale}$ x Index) + displacement where Scale has the value 0, 1, 2, or 3. | Base: any GPR Index: not ESP | `mul $t0,$s2,4`<br>`add $t0,$t0,$s1`<br>`lw $s0,100($t0) # ð16-bit # displacement` |

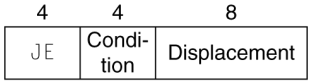
# x86 Instruction Encoding

- X86 encoding is complex with many instruction format
- Instruction length varies from 1 byte (no operands) up to 15 bytes
- Variable length encoding
  - Postfix bytes specify addressing mode
  - Prefix bytes modify operation
  - Operand length, repetition, locking, …

# x86 Instruction Encoding

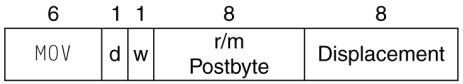
| Instruction | Function |
|---|---|
| je name | if equal(condition code) {EIP=name}; EIP-128 <= name < EIP+128 |
| jmp name | EIP=name |
| call name | SP=SP-4; M[SP]=EIP+5; EIP=name; |
| movw EBX,[EDI+45] | EBX=M[EDI+45] |
| push ESI | SP=SP-4; M[SP]=ESI |
| pop EDI | EDI=M[SP]; SP=SP+4 |
| add EAX,#6765 | EAX= EAX+6765 |
| test EDX,#42 | Set condition code (flags) with EDX and 42 |
| movsl | M[EDI]=M[ESI]; EDI=EDI+4; ESI=ESI+4 |

a. JE EIP + displacement

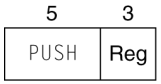



b. CALL

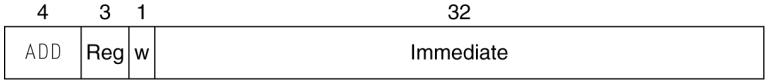

c. MOV EBX, [EDI + 45]



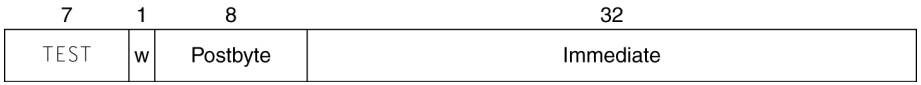

d. PUSH ESI




e. ADD EAX, #6765




f. TEST EDX, #42




•Many instructions contain the 1-bit field w, which says whether the operation is a byte or a double word.

•The d field in MOV is used in instructions that may move to or from memory and shows the direction of the move.

•The ADD instruction requires 32 bits for the immediate field, because in 32-bit mode, the immediates are either 8 bits or 32 bits.

•The immediate field in the TEST is 32 bits long because there is no 8-bit immediate for test in 32-bit mode.