Lecture 05

Processor Architecture & ISA

ALU · Control Unit · ISA · Addressing Modes · CPU Performance · Amdahl's Law

SENG 21213 · Computer Architecture & Operating Systems

Learning Objectives — By the end of this lecture you should be able to:

1. Describe ALU operations and the status flags produced by each operation group
2. Compare hardwired and microprogrammed control units on speed, flexibility, and use case
3. Explain MIPS R-type, I-type, and J-type instruction formats with field widths and uses
4. Apply all 8 addressing modes to resolve effective addresses from given operands
5. Calculate CPU execution time using T = N × CPI × T_clock and find optimal design trade-offs
6. Apply Amdahl's Law to predict parallel speedup and derive required parallelisable fraction

Section 1 · ALU and Control Unit Design

📖 **Stallings Reference**

- Chapter 3: A Top-Level View of Computer Function and Interconnection
- Chapter 11: Instruction Sets — Addressing Modes and Formats

1.1 Arithmetic Logic Unit (ALU)

The ALU is the computational engine of the processor. It accepts two n-bit operands (A and B), a function select code (op), and produces an n-bit result along with status flags.

Operation Group	Operations	Status Flags Produced
Integer Arithmetic	ADD, ADDC (add with carry), SUB, SUBB (subtract with borrow), MUL, DIV, INC, DEC, NEG	Zero (Z): result=0; Negative (N): MSB=1; Carry (C): unsigned overflow; Overflow (V): signed overflow
Logical	AND, OR, NOT, XOR, NAND, NOR	Z: result=0; N: MSB=1 (logical shift left sets C)
Shift / Rotate	SHL (shift left), SHR (shift right), SAR (arithmetic right), ROL, ROR, RCL, RCR	C: last bit shifted out; Z: result=0
Comparison	CMP (performs SUB; discards result), TEST	All flags set — flags are the purpose

Operation Group	Operations	Status Flags Produced
	(performs AND; discards result)	
Bit manipulation	BIT (test bit), SET (set bit), CLEAR (clear bit), FLIP (toggle bit)	Z: tested bit state

1.2 Control Unit — Hardwired vs. Microprogrammed

Attribute	Hardwired Control Unit	Microprogrammed Control Unit
Implementation	Combinational logic decodes opcode → control signals in one step	Each machine instruction maps to a sequence of microinstructions in a Control Store (ROM)
Speed	Fast — one gate delay per control signal	Slower — one extra memory read per microinstruction
Flexibility	Rigid — changing instruction set requires hardware redesign	Flexible — new instructions added by updating control store firmware
Complexity	Simple for RISC (small ISA); grows exponentially with ISA size	Manageable for CISC — each complex instruction decomposed into simple microops
Usage	All modern RISC processors (MIPS, ARM, RISC-V)	x86 CPUs (Intel uses microcode internally for complex instructions)
Bug correction	Requires hardware respin	Fix by updating microcode (Intel has done this for Spectre/Meltdown mitigations)

Section 2 · Instruction Set Architecture (ISA)

2.1 MIPS Instruction Formats (RISC Example)

★ MIPS Three Instruction Formats

- R-type (Register): OP[6] | rs[5] | rt[5] | rd[5] | shamt[5] | funct[6] = 32 bits
- Used for: ADD, SUB, AND, OR, SLT — all three operands are registers
- I-type (Immediate): OP[6] | rs[5] | rt[5] | immediate[16] = 32 bits
- Used for: ADDI, LOAD (LW), STORE (SW), BEQ, BNE — constant or memory address
- J-type (Jump): OP[6] | target[26] = 32 bits
- Used for: J (jump), JAL (jump and link for function calls)
- Fixed 32-bit width: every instruction takes exactly 1 clock cycle to fetch — enables clean pipelining.

2.2 Eight Addressing Modes

Mode	Syntax (x86)	Effective Address (EA)	Typical Use
Immediate	MOV EAX, 42	EA = none; operand = 42 (literal)	Loading constants, initialisers
Register	ADD EAX, EBX	EA = none; operand = register EBX	Fast local variable operations
Direct (Absolute)	MOV EAX, [0x1000]	EA = 0x1000	Global/static variable access
Register Indirect	MOV EAX, [EBX]	EA = EBX	Pointer dereferencing (*ptr)
Register + Displacement	MOV EAX, [EBX + 8]	EA = EBX + 8	Struct field access (struct.offset)
Base + Index	MOV EAX, [EBX + ESI]	EA = EBX + ESI	Array traversal with pointer+index
Base + Index + Displacement	MOV EAX, [EBX + ESI*4 + 8]	EA = EBX + ESI×4 + 8	Array-of-structs element access
PC-Relative	JMP label	EA = PC + signed offset	Branches, position-independent code

⚠ Load/Store Architecture (RISC)

- On MIPS/ARM, ONLY LW (Load Word) and SW (Store Word) instructions access memory.
- ALL other operations (ADD, SUB, AND, ...) work exclusively on register operands.
- This means: to add a value from memory, you MUST: LW $t0, 0($s0) first, then ADD $t1, $t1, $t0.
- x86 (CISC) can do ADD EAX, [mem] directly — but this makes pipelining harder because the memory access latency is variable.

Section 3 · CPU Performance Metrics

3.1 The CPU Performance Equation

★ CPU Execution Time Formula

- T_cpu = N × CPI × T_clock
- N = Instruction count (depends on program and ISA; lower is better)
- CPI = Cycles Per Instruction (average over all instruction types; lower is better)
- T_clock = 1/f (clock period; lower is better = higher frequency)
- Mixed CPI: CPI_avg = Σ(fraction_i × CPI_i) for each instruction type i
- Example: 60% INT (CPI=1), 20% FP (CPI=4), 20% MEM (CPI=2): CPI_avg = 0.6×1 + 0.2×4 + 0.2×2 = 1.8

3.2 Amdahl's Law

Amdahl's Law states that the maximum speedup from parallelising part of a program is strictly limited by the fraction that cannot be parallelised (the serial fraction):

★ Amdahl's Law

- Speedup = 1 / [s + (1−s)/p]
- s = serial fraction (0 ≤ s ≤ 1); p = number of processors
- Maximum speedup (p → ∞): Speedup_max = 1/s
- Example: s = 0.2 (20% serial). Max speedup = 1/0.2 = 5×, regardless of how many CPUs.
- Practical implication: Reduce the serial fraction, not just add CPUs.
- Gustafson's counterpoint: As problem size scales with processor count (weak scaling), near-linear speedup is achievable.

Serial Fraction (s)	Max Speedup (∞ CPUs)	Speedup (4 CPUs)	Speedup (16 CPUs)
0.5 (50% serial)	2×	1.60×	1.88×
0.2 (20% serial)	5×	2.50×	3.81×
0.1 (10% serial)	10×	3.08×	6.40×
0.05 (5% serial)	20×	3.48×	9.75×
0.01 (1% serial)	100×	3.88×	14.80×

Section 4 · Practice Exercises

✎ Exercise 5.1 [12 marks] — CPU Performance Equation

- (a) [4] A CPU runs at 3.2 GHz. A benchmark executes 4×10^9 instructions. The average CPI breakdown is: 50% integer ops (CPI=1), 25% memory ops (CPI=3), 25% FP ops (CPI=4). Calculate: (i) CPI_avg, (ii) total execution time.
- (b) [4] A new floating-point unit reduces FP CPI from 4 to 1.5. Recalculate CPI_avg and execution time. What is the speedup? Justify why the speedup is less than the improvement in FP CPI.
- (c) [4] Alternatively, the clock frequency is increased from 3.2 GHz to 4.0 GHz but this increases the average CPI from 1.875 to 2.2 (due to pipeline stalls at higher frequency). Which design (FP improvement or frequency increase) is faster? Show all calculations.

✎ Exercise 5.2 [10 marks] — Amdahl's Law

- (a) [3] A program has 40% of its execution time in a loop that can be perfectly parallelised. The remaining 60% is serial. Calculate the speedup with 2, 4, 8, and 16 processors. Plot a small table.
- (b) [3] What fraction of the program must be parallelisable to achieve a speedup of 4× with 8 processors? Solve for (1−s) algebraically from Amdahl's formula.
- (c) [4] A server application processes requests. Each request takes 100 ms: 20 ms serialised setup + 80 ms parallelisable computation. (i) With 4 cores, what is the speedup per request? (ii) With 4 cores handling 4 independent requests in parallel (each using 1 core), what is the system throughput improvement? Why is (ii) much better than (i)?

✎ Exercise 5.3 [8 marks] — Addressing Modes

- (a) [4] For each x86-64 instruction below, identify the addressing mode and compute the effective address EA where A=0x1000, B=0x2000, base register RBX=0x5000, index register RSI=0x3, scale=8: (i) MOV RAX, [0x1000] (ii) MOV RAX, [RBX] (iii) MOV RAX, [RBX + RSI*8 + 0x10] (iv) LEA RDX, [RIP + 0x200] (PC = 0x4000)
- (b) [4] Explain why PC-relative addressing is essential for Position-Independent Code (PIC) used in shared libraries (.so / .dll files). What would happen if a shared library used absolute addresses and was loaded at a different base address each time?