TO -> CESYA

What will you learn?

- Programs translated into machine learning
- Hardware - software interface
- Parallel processing

Grading

- Evaluate research papers
- Conduction a research survey } 45%
- Written exam (55%

Seven Great ideas

- abstraction
- Common case fast
- pipelining
- prediction } Performance
- Parallelism
- Hierarchy of memories
- Dependability

T1

Fundamentals of Computer Architectures

Computer Revolution

Computers went from old useless machines to smartphones

Below your program

- Application software → high-level language
- System Software → Compiler, OS
- Hardware → Processor, memory, etc

Levels of Program Code

- High-Level language (ex: C, Java)
- Assembly language
- Hardware Representation (binary digits)

Translating and Starting a program

C program → Compiler → Assembly language Program → Assembler → ML → Linker → .exe → Loader → Memory

Components of a computer

- Processor (gets data from memory)
- Memory
- Control (sends the signals)

Performance

→ Algorithm (nº of operations executed)
→ Programming language, compiler, architecture
→ Processor and memory system
→ I/O system (including OS)

Response Time → how long it takes to do a task
Throughput → Total work done per hour unit

$$\text{Performance} = \frac{1}{\text{Execution Time}}$$

Measuring Execution Time

→ Elapsed time → total response time
→ CPU time → time spent processing a job

CPU Clocking

- Clock period (duration of a clock cycle)
- Clock frequency (rate): cycles per second
- CPU Time = CPU clock cycles × Clock cycle time = $\frac{\text{CPU clock cycles}}{\text{Clock rate}}$

CPU performance equation

- Clock cycle = Instruction count × Cycles per Instruction
- CPU Time = Instruction count × CPI × Clock cycle Time = $\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock Rate}}$

MIPS → Millions of instructions per second

- MIPS = $\frac{\text{Instruction count}}{\text{Execution time} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$
- Execution time = $\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock Rate}}$

Performance Summary

- CPU time = $\frac{\text{Instructions}}{\text{Programs}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$

Performance depends

- Algorithm
- Programming
- Compiler
- Instruction set a

- Power = capacitive load × Voltage^2 × Frequency

Reducing power

- Suppose a new cpu has:
  - 85% of capacity load
  - 15% of voltage reduction
  - 15% of frequency reduction

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \cdot 0{,}85 \, (V_{old} \cdot 0{,}85)^2 \cdot (F_{old} \cdot 0{,}85)}{C_{old} \cdot V_{old}^2 \cdot F_{old}} = 0{,}52$$

↓
half th

Multiprocessors

- execute multiple instructions
- harder to | programming for performance
  | load balancing

Pitfall: Amdahl's Law

- $T_{improved} = \frac{T_{affected}}{\text{improvement factor}} + T_{unaffected}$

T2

Data Representation

- Most computers use blocks of eight bits, or bytes, as the smallest addressable unit of memory

Stored ~~Computer~~ Program Concept

- Processor → Memory
  - Accounting Program
  - Editor program
  - C compiler
  - } Machine code
  - Payroll data
  - Book text
  - Source code in C

Using bits to represent numbers

- Binary: 0000 0000 to 1111 1111
- Decimal: 0 to 255
- Hexadecimal: 00 to FF

Conversion table

Hex	Dec	Bin
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Unsigned Binary Integers

- $X = X_{n-1}2^{n-1} + X_{n-2}2^{n-2} + \dots + X_0 2^0$
- Range: 0 to $2^n - 1$

2's - complement Signed Integers

- $X = X_{n-1}2^{n-1} + X_{n-2}2^{n-2} + \dots + X_0 2^0$
- Range: -2^{n-1} to $2^{n-1} - 1$
- Most significant bit is a sign bit
- Complement and add 1

$X + \bar{X} = 1111\dots111_2 = -1$

$\bar{X} + 1 = -X$

Data representations

- limited numbers of bits
- operations can underflow/overflow
- everything is represented by numbers only

Data table

C data type	bits
char	1
short	2
int	4
long	4
long long	8
float	4
double	8
long double	10/12
pointer	4

Using Bits to represent code

- A program is a sequence of bytes
- Different machine types (Windows / Linux, etc) use diffent and incompatible instructions and encodings

Boolean Algebra

- True = 1
- False = 0

- And → A & B

&	0	1
0	0	0
1	0	1

- Not → ~A

~	
0	1
1	0

- Or → A | B

|	0	1
0	0	1
1	1	1

- Xor - A ^ B

^	0	1
0	0	1
1	1	0

Operate on bit vectors (Boolean)

- & → Intersection
- | → Union
- ^ → Symmetric difference
- ~ → Complement

ex:

```
  0110 1001
& 0101 0101
-----------
  0100 0001
```

```
  0110 1001
| 0101 0101
-----------
  0111 1101
```

```
  0110 1001
^ 0101 0101
-----------
  0011 1100
```

```
~ 0101 0101
-----------
  1010 1010
```

Bit level Operations

- implement masking operations
- Very useful in:
  - Hash tables
  - Controlling devices
  - Ip addressing
  - Image processing

Shift Operations

- shifting bit patterns to the left and to the right
- **Left Shift**: x << y
  - Fill with 0's on right
- **Right Shift**: x >> y
  - Fill with 0's on left
  - Replicate most significant bit on left

Machine Words

- **32-bit** → limit addresses to 4 GB (2^{32} bytes)
- **64-bit** → limit addresses to 16 PB (2^{64} bytes)

Pointers in C

- pointer is a reference to another variable
- pointer size is determined by word-size of machine
  - 32 bit architecture - 4 bytes
  - 64 bit architecture - 8 bytes

Byte Ordering

- multi-byte word ordered in memory:
  - **Big Endian** - least significant byte has highest address
    - Used in Internet
  - **Little Endian** - least significant byte has lowest address
    - Used by x86, Android, iOS

ex:
0x01234567

Big	01	23	45	67
Little	67	45	23	01