CSE3421
Computer Architecture

Midterm Review

Xiaodong Zhang

Donald Knuth: 1974 Turing Award Winner

"One of the main characteristics of a computer science mentality is the ability to jump very quickly between levels of abstraction, between a low level and a high level, almost unconsciously".

Topics to be covered in the exam

- Basic concept in computer architecture
- Performance based on CPU cycles and clock rates
- The Instruction Set Architecture (no assembly coding)
- Computer Arithmetic (floating point representation)
- Basic components of a processor and its pipelines (no circuit reading)
- Cache design

Basic concepts

- We have studied several key concepts in the class
  - Bandwidth and latency
  - Spatial locality and temporal locality
  - Moore's Law and Dennard Scaling Law
  - Why do we need tags in cache, and how?
  - What is a cache set? What is a cache way?
  - What does a memory address mean?
  - Hardware dependent vs hardware independent instructions
  - ….

The Core Model of CPU Execution Time

- CPU execution time (or CPU time): time the CPU spends working on a task
  - Does not include time waiting for I/O or running other programs

$$\text{CPU execution time for a program} = \text{\# CPU clock cycles for a program} \times \text{clock cycle time}$$

or

$$\text{CPU execution time for a program} = \frac{\text{\# CPU clock cycles for a program}}{\text{clock rate}}$$

- Can improve performance by reducing either the length of the clock cycle or the number of clock cycles required for a program

Extending the Model: Clock Cycles per Instruction

- Not all instructions take the same amount of time to execute
  - One way to think about execution time is that it equals the number of instructions executed multiplied by the average time per instruction

$$\text{\# CPU clock cycles for a program} = \text{\# Instructions for a program} \times \text{Average clock cycles per instruction}$$

- Clock cycles per instruction (CPI) – the average number of clock cycles each instruction takes to execute
  - A way to compare two different implementations of the same ISA

	CPI for this instruction class		
	A	B	C
CPI	1	2	3

Effective (Average) CPI

- Computing the overall effective CPI is done by looking at the different types of instructions and their individual cycle counts and averaging

$$\text{Overall effective CPI} = \sum_{i=1}^{n} (CPI_i \times IC_i)$$

  - Where IC_i is the percentage of the number of instructions of class i executed
  - CPI_i is the (average) number of clock cycles per instruction for that instruction class
  - n is the number of instruction classes

- The overall effective CPI varies by instruction mix – a measure of the dynamic frequency of instructions across one or many programs

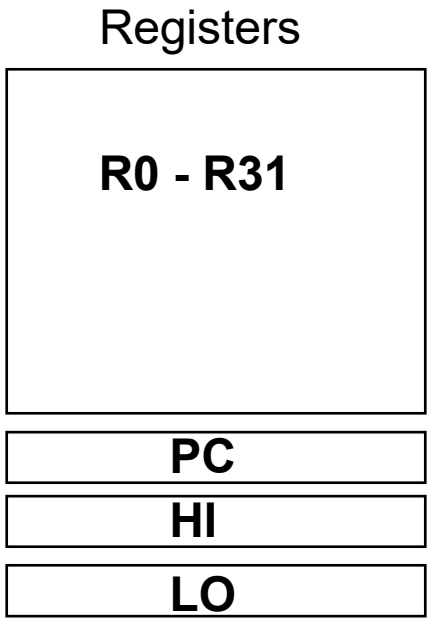
Two Key Principles of Machine Design

1. Instructions are represented as sequences of 0/1 and, as such, are indistinguishable from data
2. Programs are stored in alterable memory (that can be read or written to) just like data

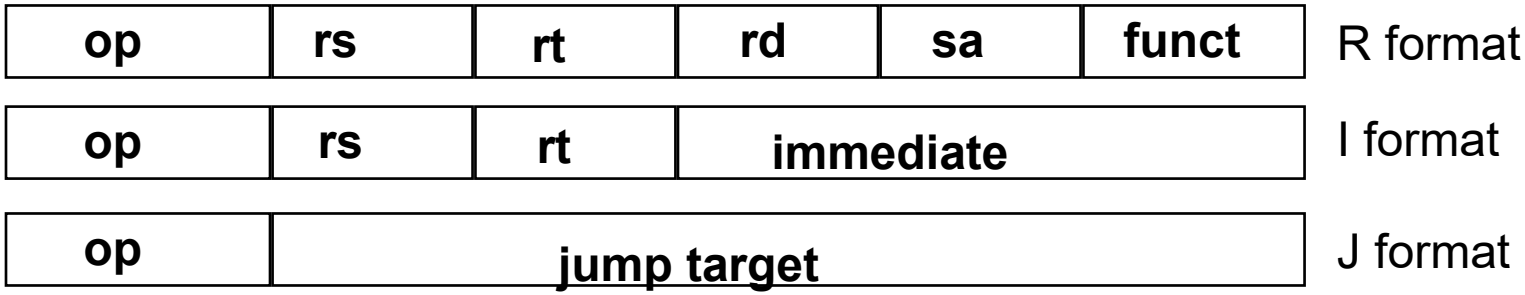
- Each instruction corresponds to a sequence of 0/1 signals
  - Human knows instruction, and machine understand 0/1
  - Can we regulate a set of instructions and the matched 0/1 signals?
  - We borrow the idea of "vocabulary and grammar" for any language
- Instruction set architecture is a set of basic instructions and 0/1
  - With ISA of the machine, one can write assembly code for computing

MIPS-32 ISA

- Instruction Categories
  - Computational
  - Load/Store
  - Jump and Branch
  - Floating Point
    - coprocessor
  - Memory Management (PC)
  - Special: HI and LO to store long word, high bits in HI, low bits in LO

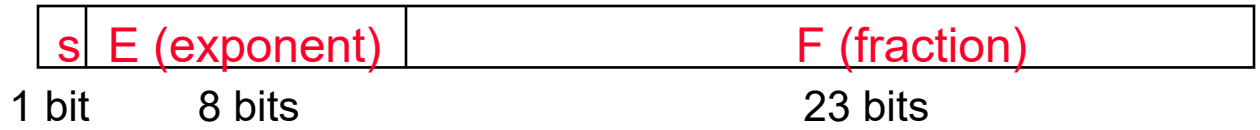


3 Instruction Formats: all 32 bits wide



Floating point representations in computers

- Floating point representation $(-1)^{sign} \times F \times 2^{E}$
  - Still have to fit everything in 32 bits (single precision)



  - The base (2, *not* 10) is hardwired in the design of the FPALU
  - More bits in the fraction (F) or the exponent (E) is a trade-off between precision (accuracy of the number) and range (size of the number)

Biased notation in binary representation

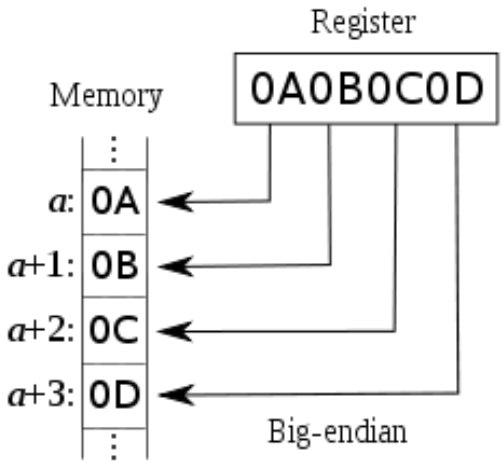
- Besides 2's complement representation for negative and positive number, another one is called **Biased Notation**
- For a given sequence of bits, the biased notation uses the smallest number to represent the most negative value and the biggest number to resent the most positive value
- For 8-bit Exponent, we have 0-255 possibilities. 0 is used for true zero
  - 00000001 (1) is used to represent most negative number (-126)
  - …
  - 01111110 (126) is the least negative number (-1)
  - 01111111 (127) is **not** used
  - 10000000 (128) is the least positive number (1)
  - …
  - 11111110 (254) represents the most positive number (127)
  - 11111111 (255) is **not** used

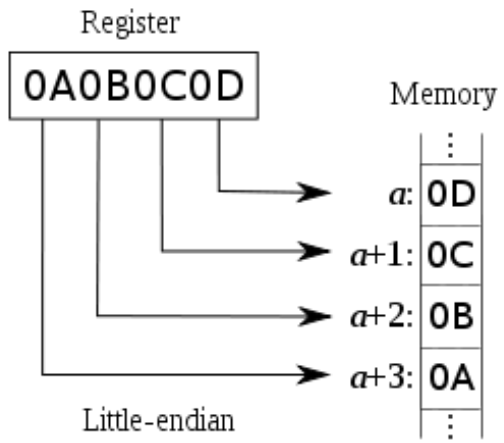
Big Endian vs Little Ending

- Alignment restriction - the memory address of a word must be on word boundaries (a multiple of 4 in MIPS)
  - same concept for House lot.
- Big Endian: most significant byte first

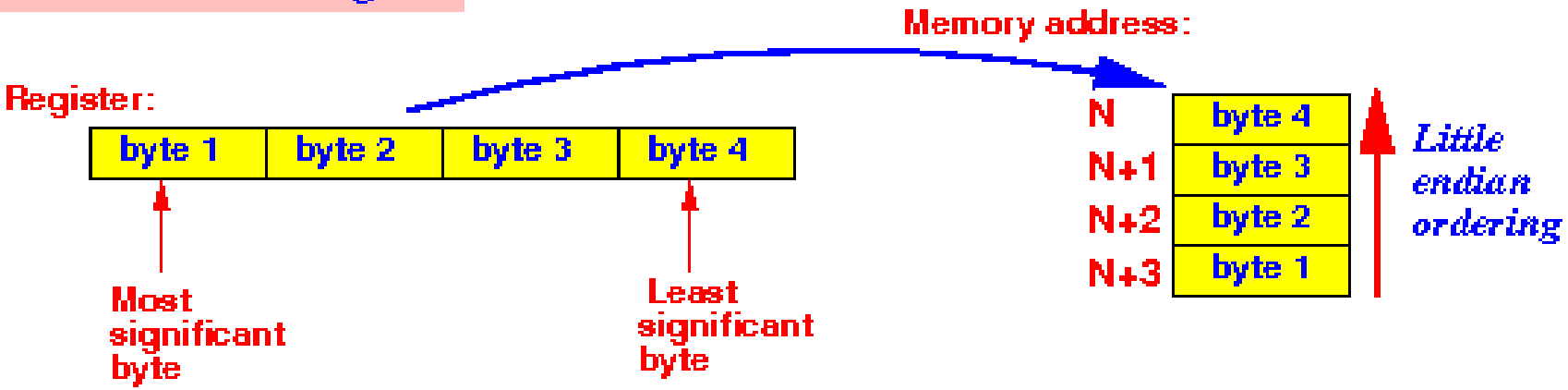
  IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA
- Little Endian: least significant byte first

  Intel 80x86, DEC Vax, DEC Alpha (Windows NT)





A 32-bit register contains 4 Bytes

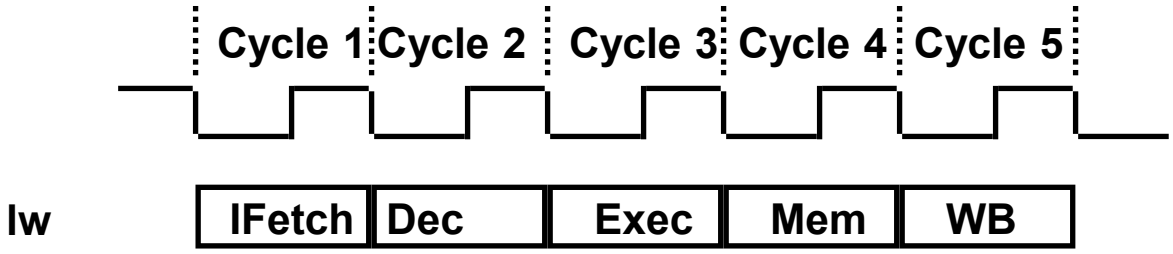


- The register has Most significant byte and least significant Byte
- For each byte-addressable memory access

  the starting address N is given
- Little Endian: least significant byte first

- How do you write the Big endian writing order between the register to the starting memory address N?

The Five Stages of Load Instruction



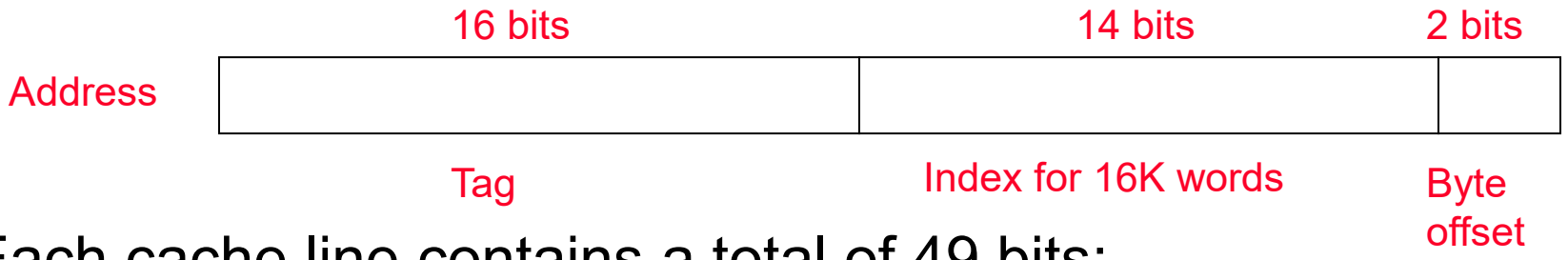
- IFetch: Instruction Fetch and Update PC
- Dec: Registers Fetch and Instruction Decode
- Exec: Execute R-type; calculate memory address
- Mem: Read/write the data from/to the Data Memory
- WB: Write the result data into the register file

Basics of Hardware Caches

- A data item is referenced by its memory address.
- It is first searched in the cache.
- Three questions cover all the cache operations:
  - How do we know it is in the cache?
  - If it is (a hit), how do we find it?
  - If it is not (a miss), how to replace the data if the location is already occupied?

Allocation of Tag, Index, and Offset Bits

- Cache size: 64 Kbytes
- Block size: 4 Bytes (2 bits for offset)
- 64 Kbytes = 16 K blocks = 2^{14} (14 bits for index)
- For a 32-bit memory address: 16 bits left for tag.



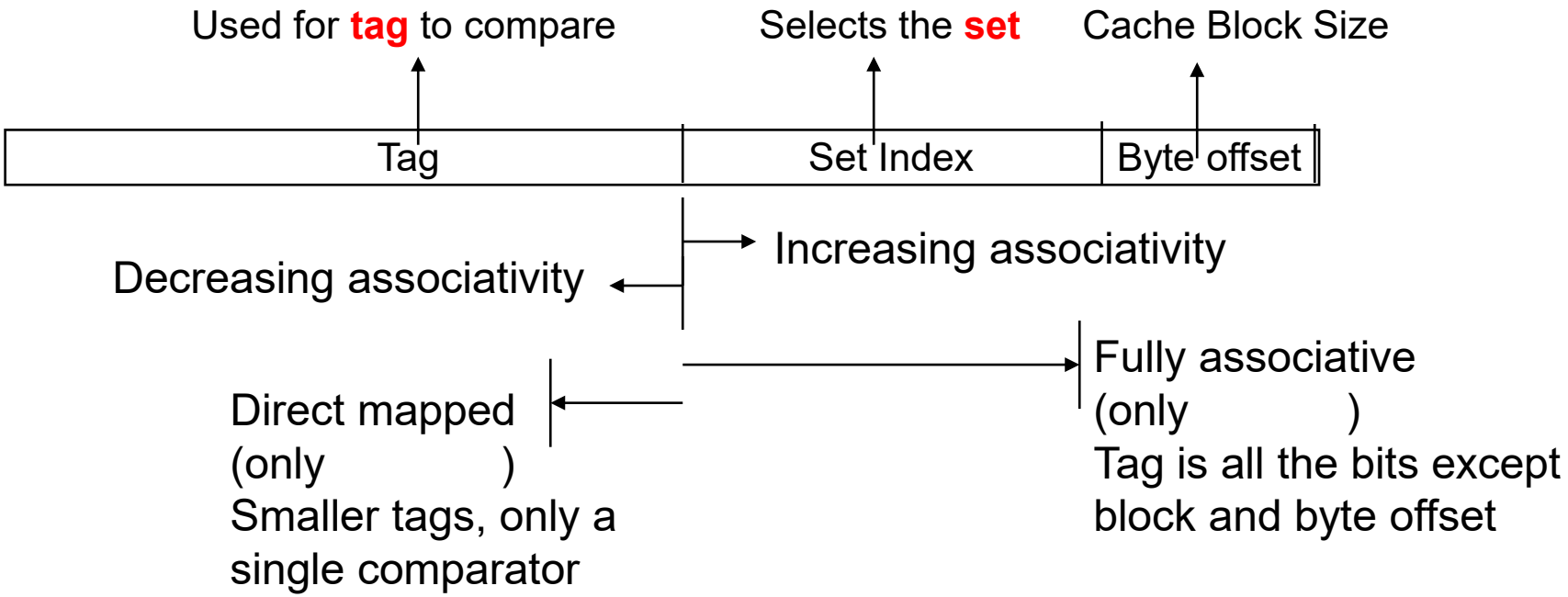
- Each cache line contains a total of 49 bits:
  - 32 bits (data of 4 Bytes)
  - 16 bits (tag)
  - 1 bit for valid bit.

Allocation of Tag, Index, and Offset Bits

Make sure to have understood the HW5 in details

Range of Set Associative Caches

- For a fixed size cache, each time the associativity increases it doubles the number of blocks per set (or the number of ways)
  - halves the number of sets – decreases the set index by 1 bit
  - increases the size of the tag by 1 bit
- Note: word offset is for multiple words in a block. word offset + byte offset = long cache block



Basic points of cache design

- From set associative cache of 1-way (direct-mapped) to a fully associative cache, how is the dividing line changed?
- As the block size changed from 1 Byte to multiple bytes, how are the two dividing lines changed?
- How to calculate the total number of bits for each cache line and for the whole cache?
- How to make a direct-mapping in set-associative cache (multicolumn cache)?