Instruction Set Architecture

Mengjia Yan
Computer Science & Artificial Intelligence Lab
M.I.T.
The IBM 650

The first mass produced computer in the world.
The IBM 650

The first mass produced computer in the world.

Side view of an IBM 650 Console Unit. First computer in Spain (1959) now at National Museum of Science and Technology in A Coruña. From Wikipedia.
The IBM 650 (1953-4)

Magnetic Drum (1,000 or 2,000 10-digit decimal words)

Active instruction (including next program counter)

Digit-serial ALU

20-digit accumulator

[From 650 Manual, © IBM]
Programmer’s view of a machine: IBM 650

A drum machine with 44 instructions

Instruction: 60 1234 1009
“Load the contents of location 1234 into the distributor; put it also into the upper accumulator; set lower accumulator to zero; and then go to location 1009 for the next instruction.”
Programmer’s view of a machine: IBM 650

A drum machine with 44 instructions

Instruction: 60 1234 1009
“Load the contents of location 1234 into the distributor; put it also into the upper accumulator; set lower accumulator to zero; and then go to location 1009 for the next instruction.”

• Programmer’s view of the machine was inseparable from the actual hardware implementation
Programmer’s view of a machine: IBM 650

A drum machine with 44 instructions

Instruction: 60 1234 1009
“Load the contents of location 1234 into the distributor; put it also into the upper accumulator; set lower accumulator to zero; and then go to location 1009 for the next instruction.”

- Programmer’s view of the machine was inseparable from the actual hardware implementation
- Good programmers optimized the placement of instructions on the drum to reduce latency!
Compatibility Problem at IBM

By early 60’s, *IBM had 4 incompatible lines of computers!*

<table>
<thead>
<tr>
<th>Model</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>701</td>
<td>7094</td>
</tr>
<tr>
<td>650</td>
<td>7074</td>
</tr>
<tr>
<td>702</td>
<td>7080</td>
</tr>
<tr>
<td>1401</td>
<td>7010</td>
</tr>
</tbody>
</table>
Compatibility Problem at IBM

By early 60’s, IBM had 4 incompatible lines of computers!

- 701 → 7094
- 650 → 7074
- 702 → 7080
- 1401 → 7010

Each system had its own

- Instruction set
- I/O system and Secondary Storage: magnetic tapes, drums and disks
- Assemblers, compilers, libraries,…
- Market niche business, scientific, real time, …
Compatibility Problem at IBM

By early 60's, *IBM had 4 incompatible lines of computers!*

- 701 → 7094
- 650 → 7074
- 702 → 7080
- 1401 → 7010

Each system had its own

- Instruction set
- I/O system and Secondary Storage: magnetic tapes, drums and disks
- Assemblers, compilers, libraries,...
- Market niche
  business, scientific, real time, ...

⇒ *IBM 360*
IBM 360: Design Premises
Amdahl, Blaauw, and Brooks, 1964

The design must lend itself to growth and successor Machines
IBM 360: Design Premises
Amdahl, Blaauw, and Brooks, 1964

The design must lend itself to *growth and successor Machines*

- General method for connecting I/O devices
- Total performance - answers per month rather than bits per microsecond ⇒ *programming aids*
- Machine must be capable of *supervising itself* without manual intervention
- Built-in *hardware fault checking* and locating aids to reduce down time
- Simple to assemble systems with redundant I/O devices, memories, etc. for *fault tolerance*
- Some problems required floating point words larger than 36 bits
Processor State and Data Types

“The information held in the processor at the end of an instruction to provide the processing context for the next instruction.”
Processor State and Data Types

“The information held in the processor at the end of an instruction to provide the processing context for the next instruction.”

Program Counter, Accumulator, ...

- The information held in the processor will be interpreted as having data types manipulated by the instructions.
Processor State and Data Types

“The information held in the processor at the end of an instruction to provide the processing context for the next instruction.”

Program Counter, Accumulator, ...

- The information held in the processor will be interpreted as having data types manipulated by the instructions.
- If the processing of an instruction can be **interrupted** then the *hardware* must save and restore the state in a transparent manner.
Processor State and Data Types

“The information held in the processor at the end of an instruction to provide the processing context for the next instruction.”

Program Counter, Accumulator, ...

- The information held in the processor will be interpreted as having data types manipulated by the instructions.
- If the processing of an instruction can be interrupted then the hardware must save and restore the state in a transparent manner

*Programmer’s machine model is a contract between the hardware and software*
Instruction Set

The control for **changing** the information held in the processor are specified by the instructions available in the instruction set architecture or ISA.
Some things an ISA must specify:

- A way to reference registers and memory
- The computational operations available
- How to control the sequence of instructions

The control for changing the information held in the processor are specified by the instructions available in the instruction set architecture or ISA.
Instruction Set

The control for *changing* the information held in the processor are specified by the instructions available in the instruction set architecture or ISA.

Some things an ISA must specify:

- A way to reference registers and memory
- The computational operations available
- How to control the sequence of instructions

- A binary representation for all of the above
Instruction Set

The control for changing the information held in the processor are specified by the instructions available in the instruction set architecture or ISA.

Some things an ISA must specify:
- A way to reference registers and memory
- The computational operations available
- How to control the sequence of instructions
- A binary representation for all of the above

ISA must satisfy the needs of the software:
- assembler, compiler, OS, VM
IBM 360: A General-Purpose Register (GPR) Machine

- **Processor State**
  - 16 General-Purpose 32-bit Registers
  - 4 Floating Point 64-bit Registers
  - A Program Status Word (PSW)
    - *PC, Condition codes, Control flags*

- **Data Formats**
  - 8-bit bytes, 16-bit half-words, 32-bit words,
    64-bit double-words
  - 24-bit addresses

- **A 32-bit machine with 24-bit addresses**
  - *No instruction contains a 24-bit address!*

- **Precise interrupts**
### IBM 360: Initial Implementations (1964)

<table>
<thead>
<tr>
<th></th>
<th>Model 30</th>
<th>. . .</th>
<th>Model 70</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Memory Capacity</strong></td>
<td>8K - 64 KB</td>
<td>. . .</td>
<td>256K - 512 KB</td>
</tr>
<tr>
<td><strong>Memory Cycle</strong></td>
<td>2.0µs</td>
<td>. . .</td>
<td>1.0µs</td>
</tr>
<tr>
<td><strong>Datapath</strong></td>
<td>8-bit</td>
<td>. . .</td>
<td>64-bit</td>
</tr>
<tr>
<td><strong>Circuit Delay</strong></td>
<td>30 nsec/level</td>
<td>. . .</td>
<td>5 nsec/level</td>
</tr>
<tr>
<td><strong>Registers</strong></td>
<td>in Main Store</td>
<td>. . .</td>
<td>in Transistor</td>
</tr>
<tr>
<td><strong>Control Store</strong></td>
<td>Read only 1µsec</td>
<td>. . .</td>
<td>Dedicated circuits</td>
</tr>
</tbody>
</table>

- Six implementations (Models, 30, 40, 50, 60, 62, 70)
- 50x performance difference across models
- **ISA completely hid the underlying technological differences between various models**

With minor modifications, IBM 360 ISA is still in use.
IBM 360: Fifty-five years later... z15 Microprocessor

- 9.2 billion transistors, 12-core design
- Up to 190 cores (2 spare) per system
- 5.2 GHz, 14nm CMOS technology

September 2019
Image credit: IBM
IBM 360: Fifty-five years later... z15 Microprocessor

- 9.2 billion transistors, 12-core design
- Up to 190 cores (2 spare) per system
- 5.2 GHz, 14nm CMOS technology

- 64-bit virtual addressing
  - Original 360 was 24-bit; 370 was a 31-bit extension
- Superscalar, out-of-order
  - 12-wide issue
  - Up to 180 instructions in flight
- 16K-entry Branch Target Buffer
  - Very large buffer to support commercial workloads
- Four Levels of caches
  - 128KB L1 I-cache, 128KB L1 D-cache
  - 4MB L2 cache per core
  - 256MB shared on-chip L3 cache
  - 960MB shared off-chip L4 cache
- Up to 40TB of main memory per system

September 2019
Image credit: IBM
Summary: Instruction Set Architecture (ISA) versus Implementation

- ISA is the hardware/software interface
- Many possible implementations of one ISA
Summary: Instruction Set Architecture (ISA) versus Implementation

- ISA is the hardware/software interface
  - Examples: MIPS, RISC-V, Alpha, x86, IBM 360, VAX, ARM, JVM

- Many possible implementations of one ISA
Summary: Instruction Set Architecture (ISA) versus Implementation

- ISA is the hardware/software interface
  - Examples: MIPS, RISC-V, Alpha, x86, IBM 360, VAX, ARM, JVM

- Many possible implementations of one ISA
  - 360 implementations: model 30 (c. 1964), z15 (c. 2019)
  - x86 implementations: 8086 (c. 1978), 80186, 286, 386, 486, Pentium, Pentium Pro, Pentium-4, Core i7, AMD Athlon, AMD Opteron, Transmeta Crusoe, SoftPC
  - MIPS implementations: R2000, R4000, R10000, ...
  - JVM: HotSpot, PicoJava, ARM Jazelle, ...
Summary: Instruction Set Architecture (ISA) versus Implementation

- ISA is the hardware/software interface
  - Examples: MIPS, RISC-V, Alpha, x86, IBM 360, VAX, ARM, JVM

- Many possible implementations of one ISA
  - 360 implementations: model 30 (c. 1964), z15 (c. 2019)
  - x86 implementations: 8086 (c. 1978), 80186, 286, 386, 486, Pentium, Pentium Pro, Pentium-4, Core i7, AMD Athlon, AMD Opteron, Transmeta Crusoe, SoftPC
  - MIPS implementations: R2000, R4000, R10000, ...
  - JVM: HotSpot, PicoJava, ARM Jazelle, ...

What potential drawbacks with ISAs?
Memory and Caches

Mengjia Yan
Computer Science and Artificial Intelligence Laboratory
M.I.T.
Memory Technology

- Early machines used a variety of memory technologies
  - Manchester Mark I used CRT Memory Storage
  - EDVAC used a mercury delay line

![Diagram of mercury delay line](image)

store acoustic signals in mercury
Memory Technology

- Early machines used a variety of memory technologies
  - Manchester Mark I used CRT Memory Storage
  - EDVAC used a mercury delay line

![Image showing a mercury delay line](image.png)

store acoustic signals in mercury
Memory Technology

• Early machines used a variety of memory technologies
  – Manchester Mark I used CRT Memory Storage
  – EDVAC used a mercury delay line

• Core memory was first large scale reliable main memory
  – Invented by Forrester in late 40s at MIT for Whirlwind project
  – Bits stored as magnetization polarity on small ferrite cores threaded onto 2 dimensional grid of wires

From Wikipedia
Memory Technology

- Early machines used a variety of memory technologies
  - Manchester Mark I used CRT Memory Storage
  - EDVAC used a mercury delay line

- Core memory was first large scale reliable main memory
  - Invented by Forrester in late 40s at MIT for Whirlwind project
  - Bits stored as magnetization polarity on small ferrite cores threaded onto 2-dimensional grid of wires

From Wikipedia
Memory Technology

- Early machines used a variety of memory technologies
  - Manchester Mark I used CRT Memory Storage
  - EDVAC used a mercury delay line

- Core memory was first large scale reliable main memory
  - Invented by Forrester in late 40s at MIT for Whirlwind project
  - Bits stored as magnetization polarity on small ferrite cores threaded onto 2 dimensional grid of wires

- First commercial DRAM was Intel 1103
  - 1Kbit of storage on single chip
  - Charge on a capacitor used to hold value

- Semiconductor memory quickly replaced core in 1970s
  - Intel formed to exploit market for semiconductor memory
Memory Technology

• Early machines used a variety of memory technologies
  – Manchester Mark I used CRT Memory Storage
  – EDVAC used a mercury delay line

• Core memory was first large scale reliable main memory
  – Invented by Forrester in late 40s at MIT for Whirlwind project
  – Bits stored as magnetization polarity on small ferrite cores threaded onto 2 dimensional grid of wires

• First commercial DRAM was Intel 1103
  – 1Kbit of storage on single chip
  – charge on a capacitor used to hold value

• Semiconductor memory quickly replaced core in 1970s
  – Intel formed to exploit market for semiconductor memory

• Flash memory
  – Slower, but denser than DRAM. Also non-volatile, but with wearout issues

• Phase change memory (PCM, 3D XPoint)
  – Slightly slower, but much denser than DRAM and non-volatile
DRAM Architecture

Memory cell (one bit)
• Bits stored in 2-dimensional arrays on chip
DRAM Architecture

- Bits stored in 2-dimensional arrays on chip
- Modern chips have around 8 logical banks on each chip
  - Each logical bank physically implemented as many smaller arrays
Processor-DRAM Gap (latency)
Four-issue 2GHz superscalar accessing 100ns DRAM could execute 800 instructions during time for one memory access!
Four-issue 2GHz superscalar accessing 100ns DRAM could execute 800 instructions during time for one memory access!

How to bridge this gap?
Little’s Law

Throughput ($T$) = Number in Flight ($N$) / Latency ($L$)
**Little’s Law**

Throughput \((T) = \frac{\text{Number in Flight (N)}}{\text{Latency (L)}}\)

Example:

--- Assume infinite-bandwidth memory
--- 100 cycles / memory reference
--- 1 + 0.2 memory references / instruction

How many inflight accesses do we need to hold to keep the CPU busy?
Little’s Law

**Throughput** \( (T) = \frac{\text{Number in Flight} \ (N)}{\text{Latency} \ (L)} \)

---

**Example:**

- Assume infinite-bandwidth memory
- 100 cycles / memory reference
- 1 + 0.2 memory references / instruction

How many inflight accesses do we need to hold to keep the CPU busy?

\[ \Rightarrow \text{Table size} = 1.2 \times 100 = 120 \text{ entries} \]

120 independent memory operations in flight!
Basic Static RAM Cell

6-Transistor SRAM Cell

word (row select)
Basic Static RAM Cell

6-Transistor SRAM Cell

• Write:
  1. Drive bit lines (bit=1, \overline{bit}=0)
  2. Select word line
Basic Static RAM Cell

6-Transistor SRAM Cell

- Write:
  1. Drive bit lines (bit=1, \overline{bit}=0)
  2. Select word line
- Read:
  1. Precharge bit and \overline{bit} to Vdd
  2. Select word line
  3. Cell pulls one bit line low
  4. Column sense amp detects difference between bit & \overline{bit}
The Concept of Caching

CPU

Small, Fast Memory (RF, SRAM) hold frequently used data

Big, Slow Memory (DRAM)
The Concept of Caching

On a data access:
- data ∈ fast memory ⇒ low latency access
- data ∉ fast memory ⇒ long latency access (DRAM)

holds frequently used data
The Concept of Caching

On a data access:
- data ∈ fast memory ⇒ low latency access
- data ∉ fast memory ⇒ long latency access (DRAM)

- size: Register << SRAM << DRAM  
- latency: Register << SRAM << DRAM
- bandwidth: on-chip >> off-chip

Small, Fast Memory (RF, SRAM) holds frequently used data

Big, Slow Memory (DRAM)
Multilevel Memory

Strategy: Reduce average latency using small, fast memories called caches.

Caches are a mechanism to reduce memory latency based on the empirical observation that the patterns of memory references made by a processor are often highly predictable:
Multilevel Memory

Strategy: Reduce average latency using small, fast memories called caches.

Caches are a mechanism to reduce memory latency based on the empirical observation that the patterns of memory references made by a processor are often highly predictable:

```
...  PC
   96

Loop:  add x2, x1, x1   100
       addi x3, x3, -1  104
       bne  x3, x0, loop 108
...      112
```
Typical Memory Reference Patterns

- Address
- Instruction fetches

n loop iterations

Time
Typical Memory Reference Patterns

Address vs. Time

Instruction fetches

Stack accesses

n loop iterations
Typical Memory Reference Patterns

Address

Instruction fetches

Stack accesses

Time

n loop iterations
Typical Memory Reference Patterns

Address

Instruction fetches

Stack accesses

n loop iterations

subroutine call

Time
Typical Memory Reference Patterns

- Instruction fetches
- Stack accesses

Address vs. Time

n loop iterations

subroutine call

argument access

September 11, 2023
Typical Memory Reference Patterns

- Instruction fetches
- Stack accesses

Address

Time

- n loop iterations
- subroutine call
- subroutine return
- argument access
Typical Memory Reference Patterns

Address

Instruction fetches

Stack accesses

Data accesses

n loop iterations

subroutine call

argument access

subroutine return

Time
Typical Memory Reference Patterns

Address

Instruction fetches

Stack accesses

Data accesses

Time

n loop iterations

subroutine call

subroutine return

argument access
Typical Memory Reference Patterns

- Instruction fetches
- Stack accesses
- Data accesses

- Address
- Time

- n loop iterations
- subroutine call
- subroutine return
- argument access
- vector access
Typical Memory Reference Patterns

- Instruction fetches
- Stack accesses
- Data accesses

Address vs. Time

- n loop iterations
- subroutine call
- subroutine return
- argument access
- vector access
- scalar accesses

September 11, 2023
Common Predictable Patterns

Two predictable properties of memory references:

- *Temporal Locality:* If a location is referenced, it is likely to be referenced again in the near future.

- *Spatial Locality:* If a location is referenced, it is likely that locations near it will be referenced in the near future.
Temporal vs. Spatial Reuse?

- **Instruction fetches**
- **Stack accesses**
- **Data accesses**

Temporal? Spatial?

- Subroutine call
- Subroutine return
- Argument access
- Vector access
- Scalar accesses

n loop iterations
Data Orchestration Techniques

Two approaches to controlling data movement in the memory hierarchy:

– *Explicit*: Manually at the direction of the programmer using instructions

– *Implicit*: Automatically by the hardware in response to a request by an instruction, but transparent to the programmer.
Data Orchestration Techniques

Two approaches to controlling data movement in the memory hierarchy:

– *Explicit*: Manually at the direction of the programmer using instructions

– *Implicit*: Automatically by the hardware in response to a request by an instruction, but transparent to the programmer.
Data Orchestration Techniques

Two approaches to controlling data movement in the memory hierarchy:

- **Explicit**: Manually at the direction of the programmer using instructions

- **Implicit**: Automatically by the hardware in response to a request by an instruction, but transparent to the programmer.
Inside a Cache

Processor → CACHE ← Main Memory

Address → Data → Address

Data → Address → Data
Inside a Cache

Processor -> CACHE -> Main Memory

Address -> Data

Copy of main memory location 100

Copy of main memory location 101

Address Tag

Data Block

100 | Data Byte | Data Byte | ------ | ------
--- | -------- | -------- | ------ | ------
304 | Data Byte | -------- | ------ | ------
6848 | -------- | -------- | ------ | ------
416  | -------- | -------- | ------ | ------
Inside a Cache

Q: How many bits needed in tag? ___________________________
Inside a Cache

Q: How many bits needed in tag? Enough to uniquely identify block
Cache Algorithm (Read)

Look at Processor Address, search cache tags to find match. Then either
Cache Algorithm (Read)

Look at Processor Address, search cache tags to find match. Then either

Found in cache
a.k.a. HIT

Return copy
of data from
cache
Cache Algorithm (Read)

Look at Processor Address, search cache tags to find match. Then either

- Found in cache
  - a.k.a. HIT
  - Return copy of data from cache

- Not in cache
  - a.k.a. MISS
  - Read block of data from Main Memory
    - Wait ...
    - Return data to processor and update cache
Cache Algorithm (Read)

Look at Processor Address, search cache tags to find match. Then either

- Found in cache
  - a.k.a. HIT
  - Return copy of data from cache
- Not in cache
  - a.k.a. MISS
  - Read block of data from Main Memory
    - Wait ...
    - Return data to processor and update cache

Which line do we replace?
Direct-Mapped Cache

![Diagram of Direct-Mapped Cache]

- V: Valid bit
- Tag
- Data Block

2^k lines
Direct-Mapped Cache

Block number | Block offset
--------------|--------------
Tag           | Index        | Offset

$V^k$ lines

V | Tag | Data Block | $2^k$
---|-----|------------|-------
|||
Direct-Mapped Cache

Block number \rightarrow Tag \rightarrow Index \rightarrow Offset

Data Block

V^k \text{Tag} \quad 2^k \text{lines}

\text{Offset}
Direct-Mapped Cache

Block number

Block offset

Tag

Index

Offset

$V_k$

$t$

Hit

Data Block

$2^k$ lines

September 11, 2023

MIT 6.5900 Fall 2023
Direct-Mapped Cache

Block number

Block offset

Tag

Index

Offset

V

Tag

Data Block

k

b

2^k lines

HIT

Data Word or Byte

September 11, 2023

MIT 6.5900 Fall 2023

L02-29
Q: What is a bad reference pattern?  ____________________
Direct-Mapped Cache

Block number

Block offset

Tag

Index

Offset

Data Word or Byte

2^k lines

Q: What is a bad reference pattern?  **Strided at size of cache**
Direct Map Address Selection
*higher-order vs. lower-order address bits*

![Diagram of direct map address selection]

**Q:** Why might this be undesirable? ________________________
Direct Map Address Selection
higher-order vs. lower-order address bits

Q: Why might this be undesirable? ____________

Spatially local blocks conflict
Hashed Address Mapping

**Question:** What are the tradeoffs of hashing?
Hashed Address Mapping

Q: What are the tradeoffs of hashing?

Good: Regular strides don’t conflict
Bad: Hash adds latency
Tag is larger
2-Way Set-Associative Cache

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
<th>Block Offset</th>
</tr>
</thead>
</table>

V  Tag  Data Block

V  Tag  Data Block

- - -
- - -
- - -

- -
- -
- -
2-Way Set-Associative Cache

- Tag
- Index
- Block Offset

V Tag Data Block

V Tag Data Block
2-Way Set-Associative Cache

- Tag
- Index
- Block Offset

For a hit, the tag and data block are compared to determine if the requested data is present.
2-Way Set-Associative Cache

- Tag
- Index
- Block Offset

Data Word or Byte

HIT

September 11, 2023
Q: Where are the index bits?
Q: Where are the index bits? Not needed
### Placement Policy

#### Block Number
```
0 1 2 3 4 5 6 7 8 9
1 1 1 1 1 1 1 1 1 1
0 1 2 3 4 5 6 7 8 9
2 2 2 2 2 2 2 2 2 2
0 1 2 3 4 5 6 7 8 9
3
0 1
```

#### Memory

#### Set Number
```
0 1 2 3 4 5 6 7
```

#### Cache

Block 12 can be placed only into block 4 \((12 \mod 8)\).
Placement Policy

Block Number

Memory

Set Number

Cache

Direct Mapped

block 12 can be placed only into block 4 
\((12 \mod 8)\)

(2-way) Set Associative

anywhere in set 0 
\((12 \mod 4)\)

block 12 can be placed only into block 4 
\((12 \mod 8)\)

(2-way) Set Associative

anywhere in set 0 
\((12 \mod 4)\)
Placement Policy

Block Number

Memory

Set Number

Cache

Direct Mapped only into block 4 (12 mod 8)

(2-way) Set Associative anywhere in set 0 (12 mod 4)

Fully Associative anywhere

block 12 can be placed
Cache Performance

Average memory access time (AMAT) = 
Hit time + Miss rate × Miss penalty
Cache Performance

Average memory access time (AMAT) =
Hit time + Miss rate \times Miss penalty

To improve performance:
• reduce the hit time
• reduce the miss rate (e.g., larger, better policy)
• reduce the miss penalty (e.g., L2 cache)

What is the simplest design strategy?
Cache Performance

Average memory access time (AMAT) =
Hit time + Miss rate x Miss penalty

To improve performance:
• reduce the hit time
• reduce the miss rate (e.g., larger, better policy)
• reduce the miss penalty (e.g., L2 cache)

What is the simplest design strategy?

Biggest cache that doesn’t increase hit time past 1-2 cycles
(approx. 16-64KB in modern technology)
[design issues more complex with out-of-order superscalar processors]
Causes for Cache Misses

• **Compulsory:**
  First reference to a block *a.k.a.* cold start misses
  - misses that would occur even with infinite cache

• **Capacity:**
  cache is too small to hold all data the program needs
  - misses that would occur even under fully-associative placement & perfect replacement policy

• **Conflict:**
  misses from collisions due to block-placement strategy
  - misses that would not occur with full associativity
## Effect of Cache Parameters on Performance

|                        | Larger capacity cache | Higher associativity cache | Larger block size cache *
|------------------------|------------------------|-----------------------------|-----------------------------
| Compulsory misses      |                        |                             |                             |
| Capacity misses        |                        |                             |                             |
| Conflict misses        |                        |                             |                             |
| Hit latency            |                        |                             |                             |
| Miss latency           |                        |                             |                             |

* Assume substantial spatial locality
## Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>=</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capacity misses</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflict misses</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hit latency</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miss latency</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* *Assume substantial spatial locality*
# Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>=</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capacity misses</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflict misses</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hit latency</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miss latency</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality

---

September 11, 2023

MIT 6.5900 Fall 2023

L02-37
### Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capacity misses</td>
<td></td>
<td>▼</td>
<td></td>
</tr>
<tr>
<td>Conflict misses</td>
<td></td>
<td>▼</td>
<td></td>
</tr>
<tr>
<td>Hit latency</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miss latency</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
# Effect of Cache Parameters on Performance

|                        | Larger capacity cache | Higher associativity cache | Larger block size cache *
|------------------------|------------------------|-----------------------------|-----------------------------
| Compulsory misses      | =                      |                             |                             |
| Capacity misses        | ↓                      |                             |                             |
| Conflict misses        | ↓                      |                             |                             |
| Hit latency            | ↑                      |                             |                             |
| Miss latency           |                        |                             |                             |

* Assume substantial spatial locality
### Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>≈</td>
<td>≈</td>
<td>≈</td>
</tr>
<tr>
<td>Capacity misses</td>
<td>↓</td>
<td>▲</td>
<td>≈</td>
</tr>
<tr>
<td>Conflict misses</td>
<td>↓</td>
<td>▲</td>
<td>≈</td>
</tr>
<tr>
<td>Hit latency</td>
<td>↑</td>
<td>≈</td>
<td>≈</td>
</tr>
<tr>
<td>Miss latency</td>
<td>≈</td>
<td>≈</td>
<td>≈</td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
## Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>≈</td>
<td>≈</td>
<td>≈</td>
</tr>
<tr>
<td>Capacity misses</td>
<td>↓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflict misses</td>
<td>↓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hit latency</td>
<td>↑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miss latency</td>
<td>≈</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
## Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>=</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>Capacity misses</td>
<td>↓</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>Conflict misses</td>
<td>↓</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>Hit latency</td>
<td>↑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miss latency</td>
<td>=</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>=</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>Capacity misses</td>
<td>↓</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>Conflict misses</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td>Hit latency</td>
<td>↑</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>Miss latency</td>
<td>=</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
# Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>=</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>Capacity misses</td>
<td>↓</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>Conflict misses</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td>Hit latency</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>Miss latency</td>
<td>=</td>
<td>=</td>
<td>=</td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
Effect of Cache Parameters on Performance

|                        | Larger capacity cache | Higher associativity cache | Larger block size cache *
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>=</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>Capacity misses</td>
<td>↓</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>Conflict misses</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td>Hit latency</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>Miss latency</td>
<td>=</td>
<td>=</td>
<td></td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
## Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>ighting</td>
<td>ighting</td>
<td>ighting</td>
</tr>
<tr>
<td>Capacity misses</td>
<td>↓</td>
<td>ighting</td>
<td>ighting</td>
</tr>
<tr>
<td>Conflict misses</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td>Hit latency</td>
<td>↑</td>
<td>↑</td>
<td>↑</td>
</tr>
<tr>
<td>Miss latency</td>
<td>ighting</td>
<td>ighting</td>
<td>ighting</td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
## Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>=</td>
<td>=</td>
<td>▼</td>
</tr>
<tr>
<td>Capacity misses</td>
<td>▼</td>
<td>=</td>
<td>▼</td>
</tr>
<tr>
<td>Conflict misses</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
</tr>
<tr>
<td>Hit latency</td>
<td>▲</td>
<td>▲</td>
<td>▲</td>
</tr>
<tr>
<td>Miss latency</td>
<td>=</td>
<td>=</td>
<td>=</td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
# Effect of Cache Parameters on Performance

*Assume substantial spatial locality*

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>=</td>
<td>=</td>
<td>∇</td>
</tr>
<tr>
<td>Capacity misses</td>
<td>∇</td>
<td>=</td>
<td>∇</td>
</tr>
<tr>
<td>Conflict misses</td>
<td>∇</td>
<td>∇</td>
<td>?</td>
</tr>
<tr>
<td>Hit latency</td>
<td>↑</td>
<td>↑</td>
<td></td>
</tr>
<tr>
<td>Miss latency</td>
<td>=</td>
<td>=</td>
<td></td>
</tr>
</tbody>
</table>
### Effect of Cache Parameters on Performance

|                            | Larger capacity cache | Higher associativity cache | Larger block size cache *
|---------------------------|------------------------|-----------------------------|-----------------------------
| Compulsory misses        | ≈                      | ≈                           | ↓                           |
| Capacity misses           | ↓                      | ≈                           | ↓                           |
| Conflict misses           | ↓                      | ↓                           | ?                           |
| Hit latency               | ↑                      | ↑                           | ≈                           |
| Miss latency              | ≈                      | ≈                           |                             |

* Assume substantial spatial locality
## Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>=</td>
<td>=</td>
<td>↓</td>
</tr>
<tr>
<td>Capacity misses</td>
<td>↓</td>
<td>=</td>
<td>↓</td>
</tr>
<tr>
<td>Conflict misses</td>
<td>↓</td>
<td>↓</td>
<td>?</td>
</tr>
<tr>
<td>Hit latency</td>
<td>↑</td>
<td>↑</td>
<td>=</td>
</tr>
<tr>
<td>Miss latency</td>
<td>=</td>
<td>=</td>
<td>↑↓</td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
Backup Slides
Processor Performance

\[
\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}
\]

- Instructions per program depends on source code, compiler technology and ISA
- Cycles per instructions (CPI) depends upon the ISA and the microarchitecture
- Time per cycle depends upon the microarchitecture and the base technology
Processor Performance

\[
\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}
\]

- Instructions per program depends on source code, compiler technology and ISA
- Cycles per instructions (CPI) depends upon the ISA and the microarchitecture
- Time per cycle depends upon the microarchitecture and the base technology

<table>
<thead>
<tr>
<th>Microarchitecture</th>
<th>CPI</th>
<th>cycle time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcoded</td>
<td>&gt;1</td>
<td>short</td>
</tr>
<tr>
<td>Single-cycle unpipelined</td>
<td>1</td>
<td>long</td>
</tr>
<tr>
<td>Pipelined</td>
<td>1</td>
<td>short</td>
</tr>
</tbody>
</table>
CPU-Memory Metrics

CPU ➔ Memory

September 11, 2023
CPU-Memory Metrics

- Latency (time for a single access)
  Memory access time >> Processor cycle time
CPU-Memory Metrics

- **Latency** (time for a single access)
  Memory access time >> Processor cycle time

- **Bandwidth** (number of accesses per unit time)
  If memory latency is $t$ cycles, and we have $k$ banks,
  The memory bandwidth is $k/t$ requests per cycle
CPU-Memory Metrics

- **Latency (time for a single access)**
  Memory access time >> Processor cycle time

- **Bandwidth (number of accesses per unit time)**
  If memory latency is \( t \) cycles, and we have \( k \) banks,
  The memory bandwidth is \( \frac{k}{t} \) requests per cycle

- **Energy (nJ per access)**
Management of Memory Hierarchy

- **Small/fast storage, e.g., registers**
  - Address usually specified directly in instruction
  - Generally implemented using **explicit** data orchestration
    - e.g., directly as a register file
    - but hardware might do things behind software’s back, e.g., stack management, register renaming

- **Large/slower storage, e.g., memory**
  - Address usually computed from values in register
  - Generally implemented using **implicit** data orchestration
    - e.g., as a cache hierarchy where hardware decides what is kept in fast memory
    - but software may provide “hints”, e.g., don’t cache or prefetch
Thank you!

Next lecture:
Caches + Virtual memory