2GA3 Tutorial #2

DATE: September 24th, 2021
TA: Jatin Chowdhary
TOPIC: RISC-V Instructions

When it's the second tutorial session and you still don't know what you are doing, so you're just going along

First Time

- Ask questions
- Give feedback
- Point out my mistakes
- Criticism is welcomed
- Don’t be shy/afraid
  - Not marking
- If I forget something, tell me!
  - Won’t nail every single point
  - Often times, it’ll come to me hours later (or in the shower)


Ideas Start Here...

Stay Safe

- Everyone I know is getting into car accidents
  - Cousin
    - 2 accidents in the span of 3 months
  - Neighbour
    - 1 week ago
  - Me
    - ~ 1 month ago
    - Got rear-ended on the highway
    - His fault
- Watch out and take care!

Other Stuff (1)

- Avoid sitting on the right side
  - Harder to see blackboard when I'm writing
    - I'm right handed
- When you come to class:
  - Be enthusiastic
  - Have a lot of questions
    - It's better to ask questions, than it is to take notes
      - Everything is, or will be, posted
  - **Sit up straight**
  - Phones on silent
  - Laptops tucked away

Other Stuff (2)

- Not (officially) taking attendance
  - But I will keep track, mentally
    - What does this mean?
- Still learning your names
  - Give it time
- Assignment questions
  - Direct them to Peter
- Haven’t seen the assignment
  - Not going to
  - Avoid bias

Quick Review

- **Question:** There are 3 processors, {i3, i5, i7}. The i3 processor has a clock rate of 4.5 GHz and a CPI of 1.5. The i5 processor has a clock rate of 3.0 GHz and a CPI of 1.5. The i7 processor has a clock rate of 3.8 GHz and a CPI of 1.5. Which processor has the highest performance? Assume all processors have the same architecture, and no other differences are present.

Answer

- The i3 processor is the fastest
  - It has the highest clock rate
  - CPI is irrelevant because they're all the same
    - No other differences are stated
  - No need to convert GHz to Hz
    - But only for this question
    - If this question had a part (b), then a conversion would probably be required
    - 10^9 is a constant, *c*

CPI

- CPI: Clock Cycles Per Instruction
- Recall that CPI differs based on:
  - Program
  - Processor
  - Architecture
  - *Implementation*

Implementation

- Why does implementation matter?
  - Simple example of copy/paste VS. duplicate and how it affects the clipboard
    - *Duplicate* does not affect the *clipboard*, and if it does, it reverts the changes

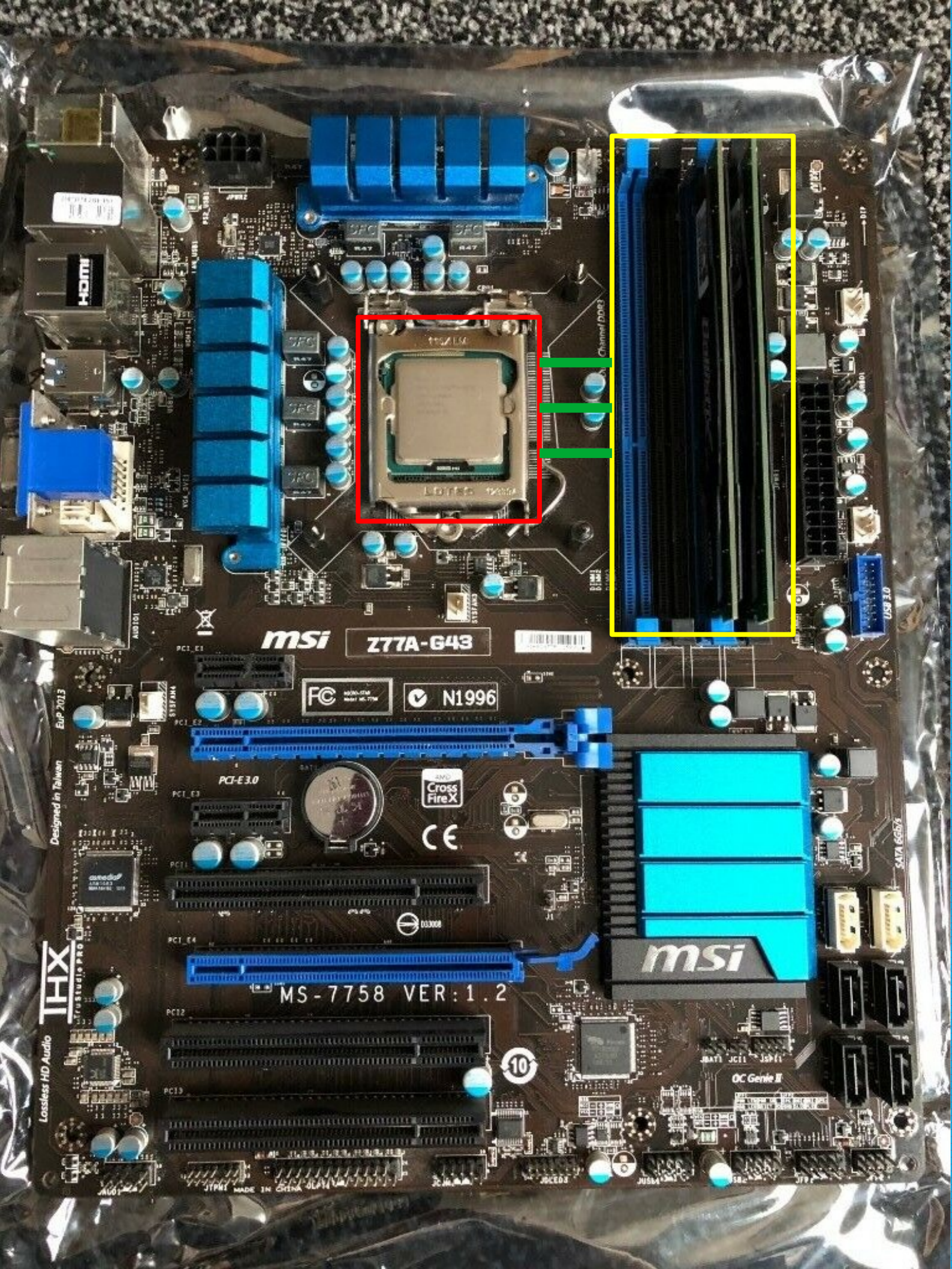
Lecture Question

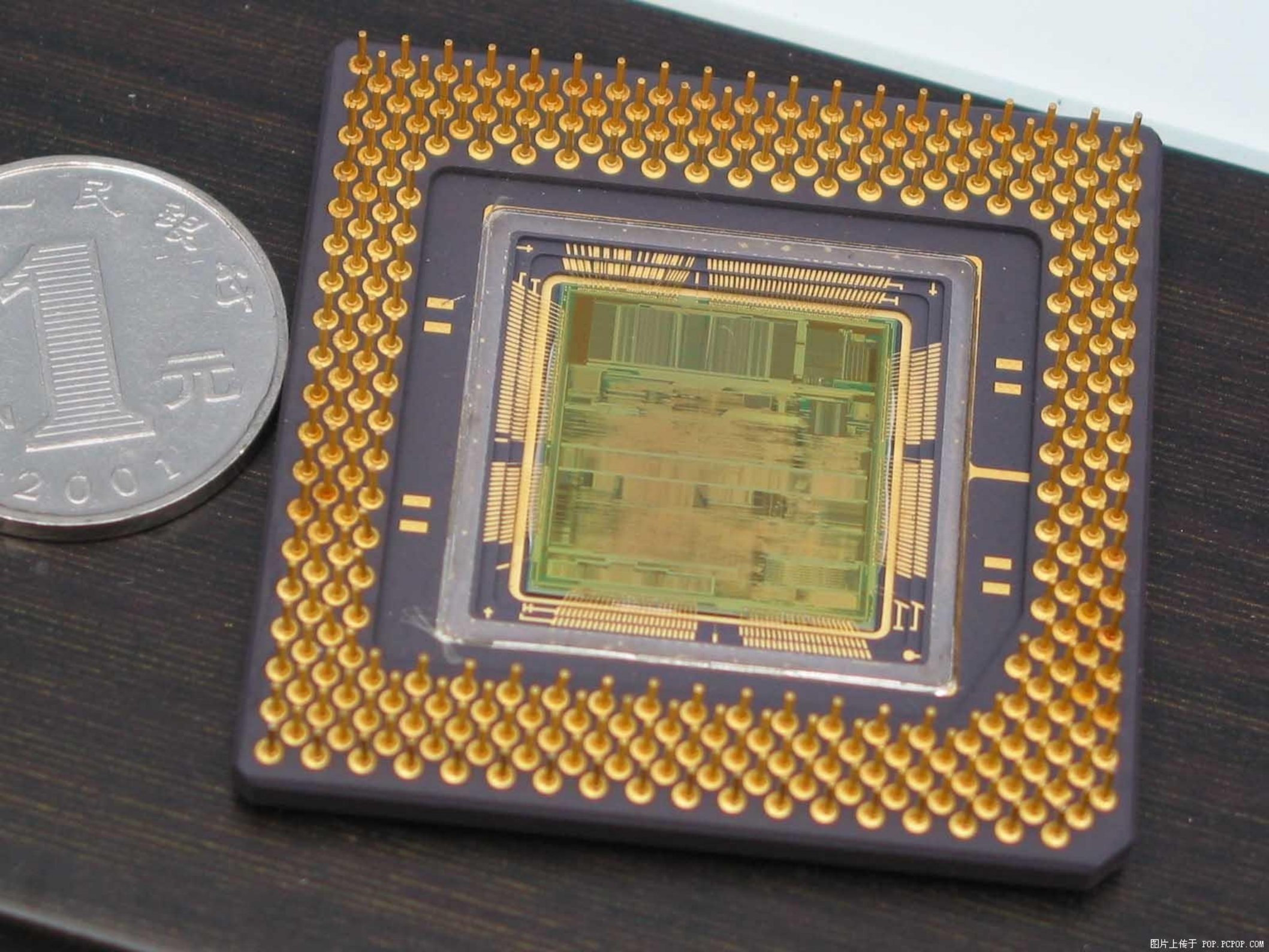
- **Question:** Between *lw* and *add*, which instruction takes more time? Assume that *lw* is used to load data from memory, and *add* is used to add data already in registers.
- **Options:**
  - A) Load
  - B) Add
  - C) Same time
  - D) Depends on the weather outside
- Answer: D

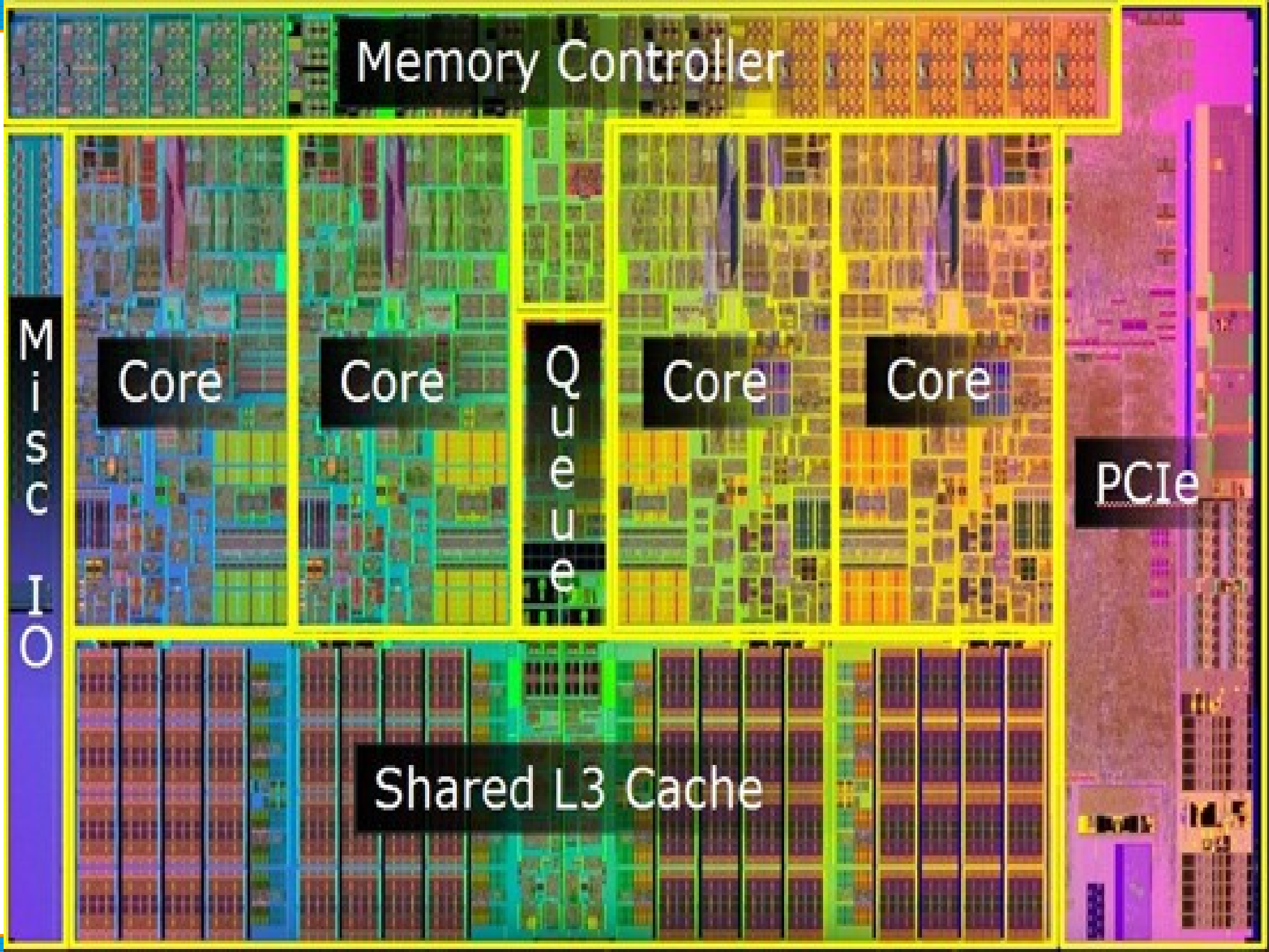
Register

- What is a register?
  - Best question from last week
  - Asked by Melissa
    - Thank you Melissa
  - The basis of everything
    - Simple question, but very broad
  - Pay attention
    - This is very useful!

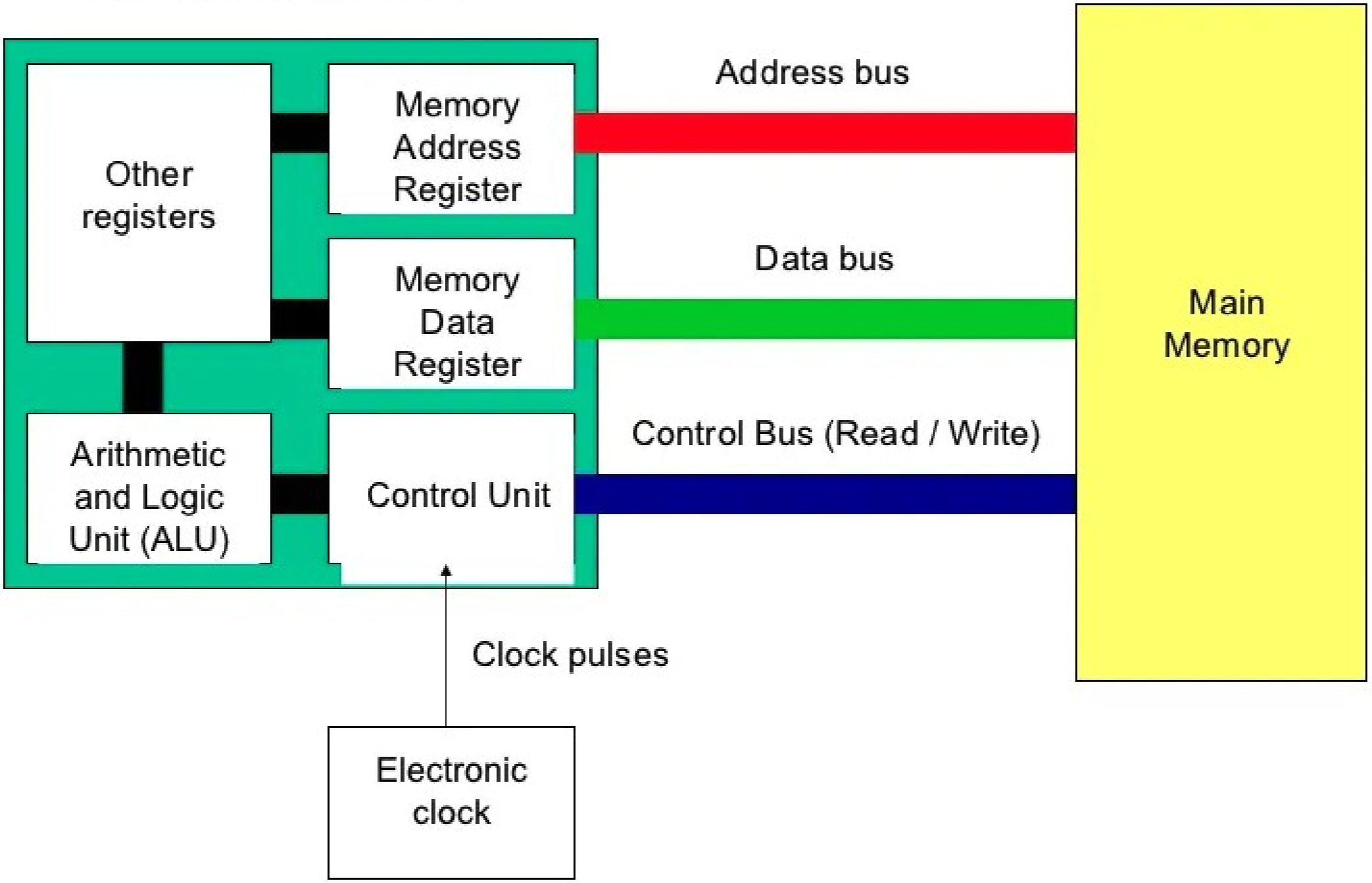

HDMI


msi
Z77A-G43
N1996
MS-7758 VER:1.2
Designed in Taiwan
PCI-E 3.0
Cross FireX
OC Genie II
Lossless HD Audio




Memory Controller
Misc IO
Core
Core
Queue
Core
Core
PCIe
Shared L3 Cache

Inside the CPU



Recap

- Registers are in the CPU
- Each core has its own set of registers
- Registers are used to store, accept, and transfer instructions and data
  - i.e. Modifying an image
  - i.e. Playing a video game
  - Anytime you use a computer, registers are used

Practical Example

- Now let's connect it all
  - Program In C → Compile to assembly
  - Assembly → Machine code (Instructions)
  - Machine code runs in the CPU
  - Information is exchanged over the BUS
    - BUS connects the RAM and CPU
      - This is not a school bus
        - But it is magical

CMOS/PMOS

- I’ll try to explain it next week
  - No time today
    - (We’re always running out of time)

Architecture

- 32-bit VS. 64-bit
  - Will be explained next week
  - Relates to register size
    - The bigger the *register*, the more information we can shove; we can do more
      - i.e. Moving houses (Use your car OR rent a U-Haul)

Question 1

- For the following **C statement**, write the corresponding RISC-V assembly code. Assume that the C variables *f*, *g*, and *h*, have already been placed in registers *x5*, *x6*, and *x7*, respectively. Use a minimal number of RISC-V assembly instructions.

  f = g + (h – 5);

Solution

```
addi x5, x7, -5    # f = h – 5
add x5, x5, x6     # f = g + f
```

- *addi* stands for add immediate
  - Used for constants
    - i.e. 2, 3, 5, etc.
- *add* is used to sum values that are already in registers
  - Values must be loaded, via *lw*, before they can be added
- Note: There is no *subi*

Quick Review 1

- What is the largest *unsigned* value you can fit in a 32-bit register?
- Options:
  - 2^{32}
  - 2^{31}
  - $2^{32} - 1$
  - $2^{31} - 1$
  - None of the above
- *Example: C Program*

Question 2

- For the following C statement, write the corresponding RISC-V assembly code. Assume that the variables *f*, *g*, *h*, *i*, and *j* are assigned to registers *x5*, *x6*, *x7*, *x28*, and *x29*, respectively. Assume that the base address of the arrays *A* and *B* are in registers *x10* and *x11*, respectively.

  B[8] = A[i–j];

Solution (1)

- **32-bit**

```
sub x30, x28, x29     // Compute [i – j]
slli x30, x30, 2      // Multiply by 4 to get byte offset
add x10, x10, x30     // Calculate memory location
lw x30, 0(x10)        // Load A[i – j]
sw x30, 32(x11)       // Store in B[8]
```

- But wait, what about 64-bit?

Solution (2)

- **32-bit**

```
sub x30, x28, x29     // Compute [i – j]
slli x30, x30, 2      // Multiply by 4 to get byte offset
add x10, x10, x30     // Calculate memory location
lw x30, 0(x10)        // Load A[i – j]
sw x30, 32(x11)       // Store in B[8]
```

- ***64-bit***

```
sub x30, x28, x29     // Compute [i – j]
slli x30, x30, 3      // Multiply by 8 to get byte offset
add x10, x10, x30     // Calculate memory location
ld x30, 0(x10)        // Load A[i – j]
sd x30, 64(x11)       // Store in B[8]
```

Quick Review 2

- What is the largest *signed* value you can fit in a 64-bit register?
- Options:
  - 2^{64}
  - 2^{63}
  - $2^{64} - 1$
  - $2^{63} - 1$
  - None of the above
- *Example: C Program*

Question 3

- Translate the following C code to RISC-V. Assume that the variables *f*, *g*, *h*, *i*, and *j* are assigned to registers *x5*, *x6*, *x7*, *x28*, and *x29*, respectively. Assume that the base address of the arrays *A* and *B* are in registers *x10* and *x11*, respectively. Assume that the elements of the arrays *A* and *B* are 8-byte words.

  B[8] = A[i] + A[j];

Solution (1)

- **32-bit**
  - slli x28, x28, 2 # x28 = i * 4
  - add x28, x10, x28 # x28 = &A[0] + i * 4
  - lw x28, 0(x28) # x28 = A[i]
  - slli x29, x29, 2 # x29 = j * 4
  - add x29, x10, x29 # x29 = &A[0] + j * 4
  - lw x29, 0(x29) # x29 = A[j]
  - add x28, x28, x29 # x28 = A[i] + A[j]
  - sw x28, 32(x11) # store result in B[8]
- *Solution provided by Mingzhe Wang*

Solution (2)

- **64-bit**
  - slli x28, x28, 3 # x28 = i * 8
  - add x28, x10, x28 # x28 = &A[0] + i * 8
  - ld x28, 0(x28) # x28 = A[i]
  - slli x29, x29, 3 # x29 = j * 8
  - add x29, x10, x29 # x29 = &A[0] + j * 8
  - ld x29, 0(x29) # x29 = A[j]
  - add x28, x28, x29 # x28 = A[i] + A[j]
  - sd x28, 64(x11) # store result in B[8]
- *Solution provided by Mingzhe Wang*