2GA3 Tutorial #4

DATE: October 8th, 2021

TA: Jatin Chowdhary

Anonymous Feedback

- Spend ~4 Mins
  - Feedback on course/tutorial/content/etc.
- 100% Anonymous
  - I'm leaving the room
  - If you're paranoid about anonymity, write with your other hand
    - Then no one will be able to recognize your answers
- Be Honest
  - "The whole truth and nothing but the truth"
    - My cholesterol is high enough

Minor Stuff

- Expect Typos
  - Keyboard is breaking down
    - Keys are hard to press
  - The ***S*** & ***T*** keys
- Low Energy?
  - Let me know!
- Reading Week
  - Don’t waste your time
    - Study for the midterm
      - It’s gonna be long and hard

Major Stuff

- Note Taking?
  - No need
    - Everything is, or will be posted
  - Just participate
    - Because ***r*** _ _ _ _ _ _ _ _ _ _ _ _ _ is the language of the ***b*** _ _ _ _
    - *Participation* = ***R*** _ _ _ _ _ _ _ _ _ _ _ _
  - When you go home, then take notes
    - You'll be surprised at your retention rate

Review (1)

- **Question:** What are registers?
- **Options:**
  - A) Measure of how fast a processor performs operations
  - B) Where CPUs temporarily store and process information
  - C) The bridge between the CPU and RAM
  - D) Used to store money at the store
  - E) Secondary storage in a computer

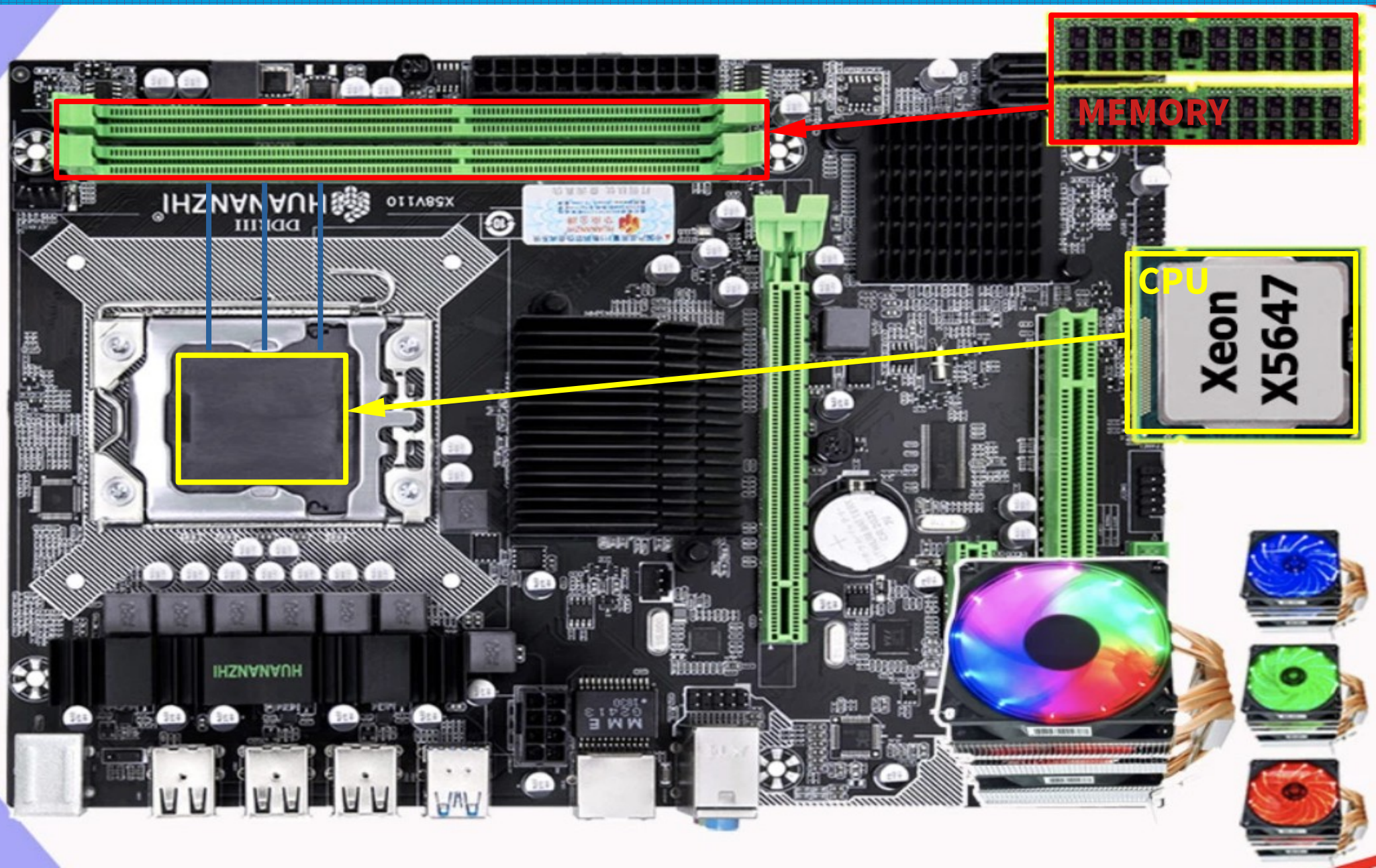
Review (2)

- **Question:** Which of the following are part of the main memory hierarchy?
- **Options:**
  - A) Registers
  - B) Cache
  - C) RAM
  - D) All of the above
  - E) None of the above

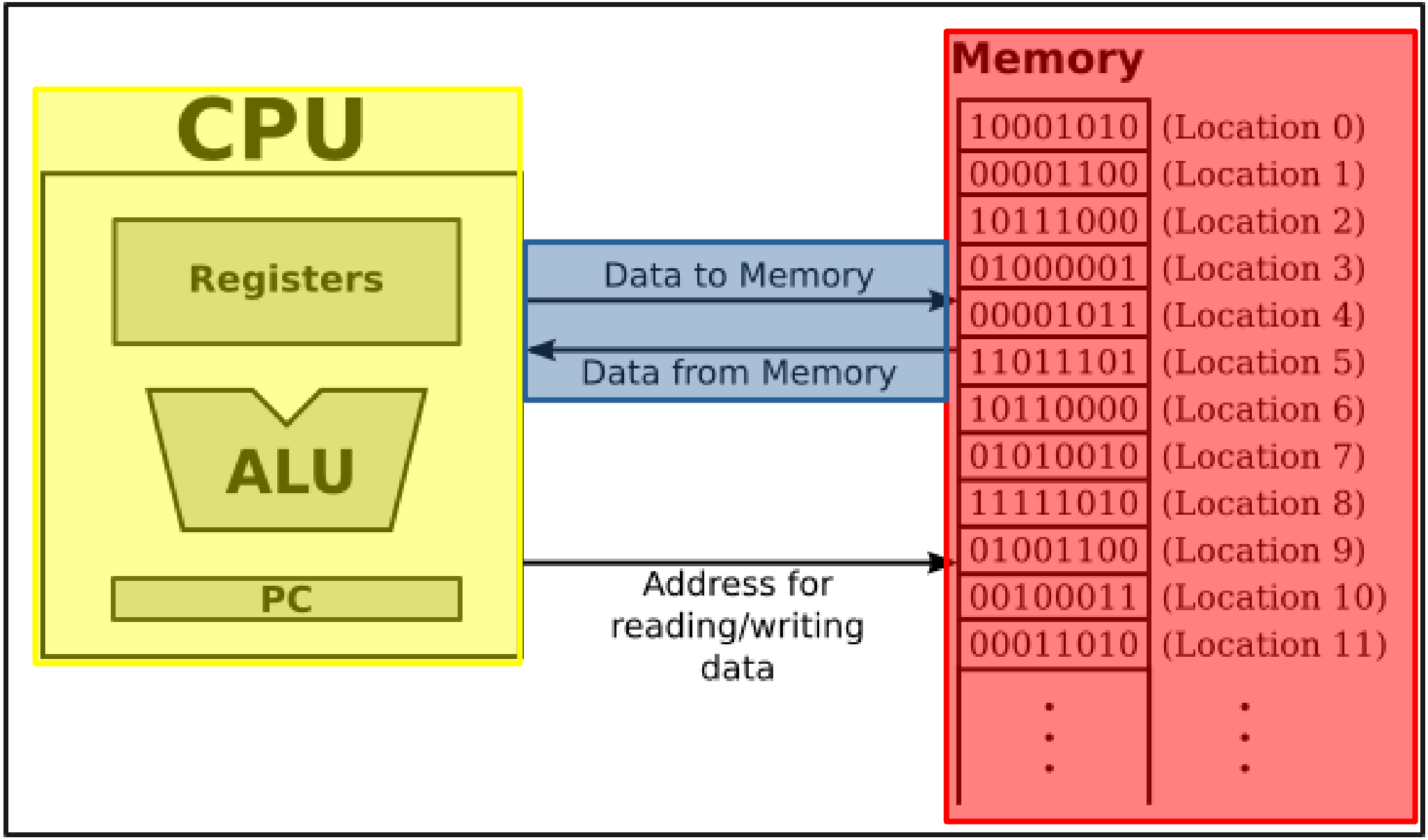
Review (3)

- **Question:** Main memory is…
- **Options:**
  - A) Fast and non-volatile
  - B) Slow and non-volatile
  - C) Fast and volatile
  - D) Slow and volatile
  - E) None of the above

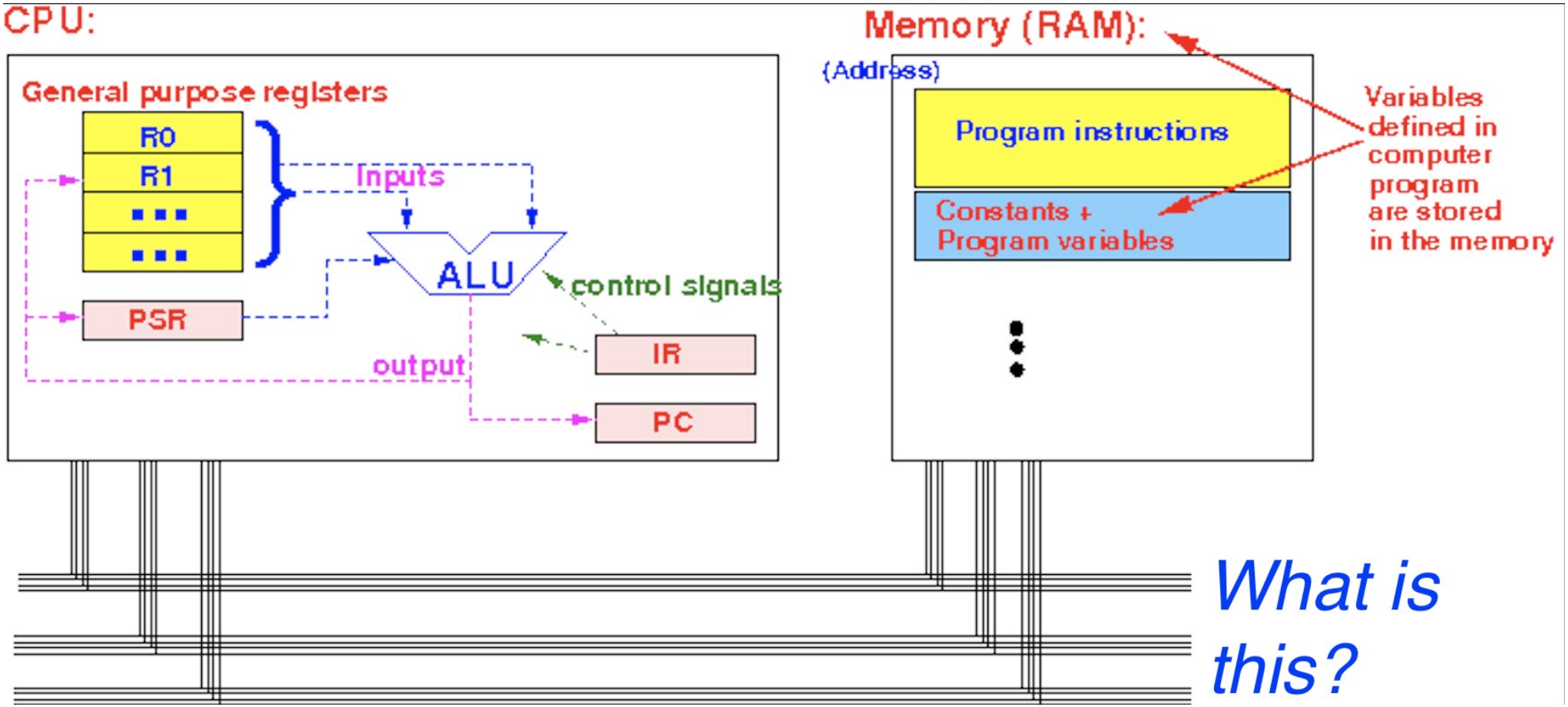
CPU & Memory



Abstract View



More Abstraction



- Everything in memory is either a 0 or a _
- Information is sent over the _ _ _
- R _ _ _ _ _ _ _ _ _ are found in the CPU

- ALU stands for…?
- RAM stands for…?
- PC stands for…?

Review (4)

- **Question:** Which of the following base-number systems are invalid?
- **Options:**
  - A) Base-4
  - B) Base-12
  - C) Base-20
  - D) Base-60
  - E) All of the above
  - F) None of the above

Review (5)

- **Question:** Can we use *base-3* or *base-4* number systems instead of *base-2*, in a computer? In other words, instead of a high/low voltage, there'll be 3, 4, etc. voltage(s)
- **Options:**
  - A) Yes
  - B) No
  - C) Maybe
  - D) I don't know
  - E) Can you repeat the question?
- *Next slide…*

Base ‘X’

- **Question:** Can we use *base-3* or *base-4* number systems instead of *base-2*, in a computer?
  - **Answer:** Yes we can.
    - But we don’t, because:
      - Harder to implement
      - More complexity
      - Boolean logic was established before computers
      - Noise
        - Need a clear distinction between 0 (false) and 1 (true)

Why Bother With Hex?

- **Question:** What is the point of hexadecimal?
  - Asked by Emily
- **Answer:** It's a better way to represent binary-coded values
  - Each hexadecimal "character" represents 4 bits
    - *Bit = **Bi**nary Digi**t***
  - Reading hex numbers is easier than reading binary numbers
  - Used by computer scientists, chip designers, *hackers*, etc.
    - Hackers use disassemblers to study how a program works, find vulnerabilities, and recreate the original code (as much as they can)
      - i.e. IDA *(on next slide)*

IDA (Pro)

IDA View-A | Hex View-A | Exports | Imports | Names | Strings | Structures | Enums | Functions

```
.text:1000D02E
.text:1000D02E hinstDLL        = dword ptr  4
.text:1000D02E fdwReason       = dword ptr  8
.text:1000D02E lpvReserved     = dword ptr  0Ch
.text:1000D02E
.text:1000D02E                 mov     eax, [esp+fdwReason]
.text:1000D032                 dec     eax
.text:1000D033                 jnz     loc_1000D107
.text:1000D039                 mov     eax, [esp+hinstDLL]
.text:1000D03D                 push    ebx
.text:1000D03E                 mov     ds:hModule, eax
.text:1000D043                 mov     eax, off_10019044
.text:1000D048                 push    esi
.text:1000D049                 add     eax, 0Dh
.text:1000D04C                 push    edi
.text:1000D04D                 push    eax             ; char *
.text:1000D04E                 call    strlen
.text:1000D053                 mov     ebx, ds:CreateThread
.text:1000D059                 mov     esi, ds:_strnicmp
.text:1000D05F                 xor     edi, edi
.text:1000D061                 pop     ecx
.text:1000D062                 test    eax, eax
.text:1000D064                 jz      short loc_1000D089
.text:1000D066                 mov     eax, off_10019044
.text:1000D06B                 push    7               ; size_t
.text:1000D06D                 add     eax, 0Dh
.text:1000D070                 push    offset aHttp    ; "http://"
.text:1000D075                 push    eax             ; char *
```

0000C42E | 1000D02E: DllMain(x,x,x)

More Hexadecimal

- 4 bits (**bi**nary digi**t**s) can be represented by a *single* hexadecimal "character"
  - Much easier to read ***0x000A***, than it is to read ***0b0000000000001010***
    - Imagine how long a 32-bit or 64-bit number would be. Using hex makes sense
- **Question:** Which of the following is a hexadecimal number?
- **Options:**
  - 0x10101
  - 0b00AB
  - $(00FF)_{10}$
  - "ABC"
  - $(54321)_6$

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

Review (6)

- **Question:** What is the difference between *add* and *addi*?
- **Options:**
  - A) There is no difference; it's the same thing
  - B) *add* is for large numbers; *addi* is for small numbers
  - C) *add* is for 32-bit; *addi* is for 64-bit
  - D) *add* adds the values in 2 registers; *addi* adds a constant value to a register
  - E) None of the above

Which Add?

- **Question:** Which instruction syntax corresponds to *add* & *addi*?
  - *rd*, *rs1*, *rs2* is the syntax for *XXX*
  - *rd*, *rs1*, *imm* is the syntax for *XXX*
- Note: *XXX* is the instruction

Registers Abbreviated

- rs = Register Source
- rd = Register Destination
- imm = Immediate
  - Also known as *c* _ _ _ _ _ _ _ _
  - Example?

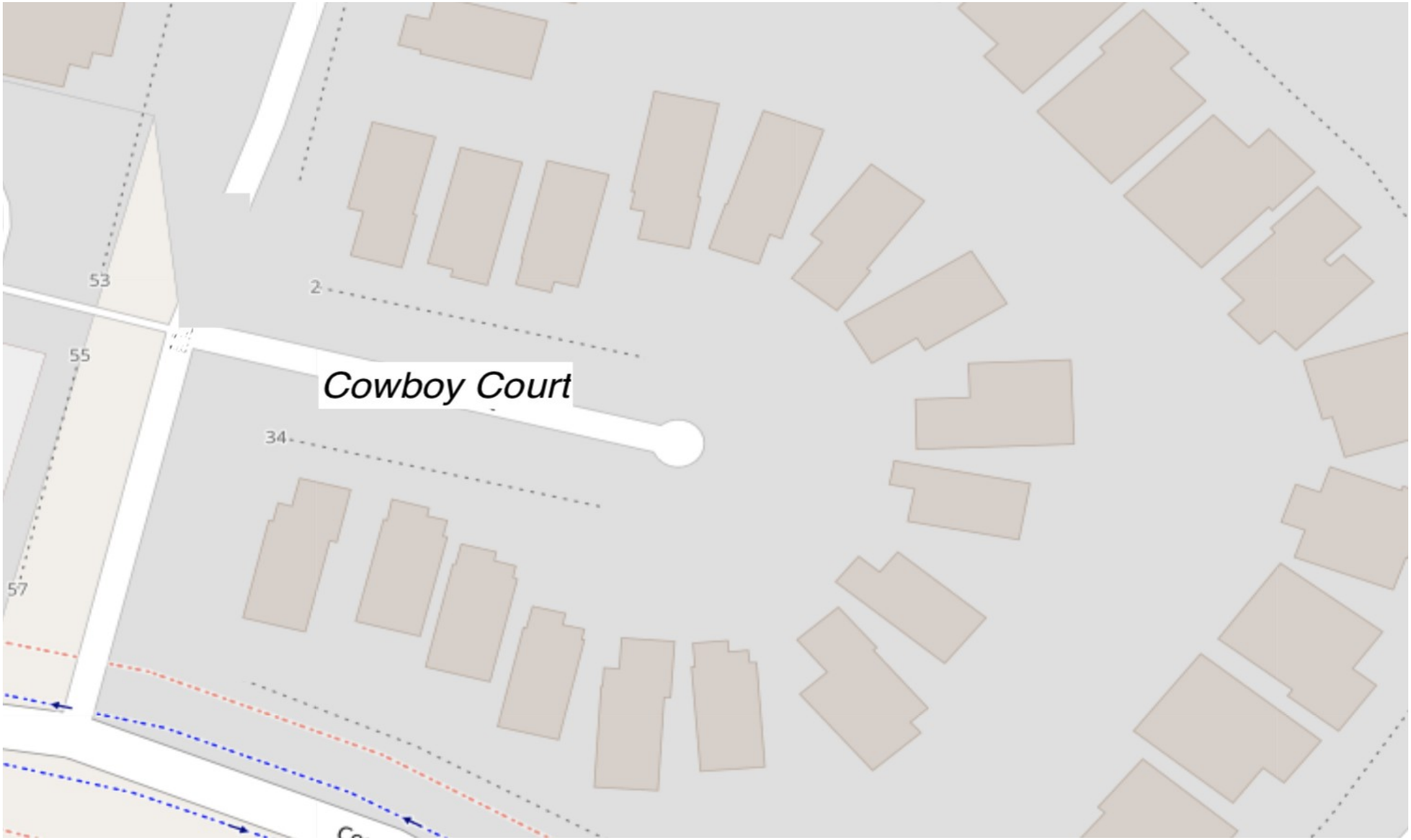
RISC-V Registers

- **Question:** Can the instruction list/set be used to determine which register is which?
  - Asked by Liam
- **Answer:** Nope, you should use the *GreenCard*, to determine which register is which. Then, use the *InstructionList* to convert RISC-V assembly to machine code
  - *Demo*

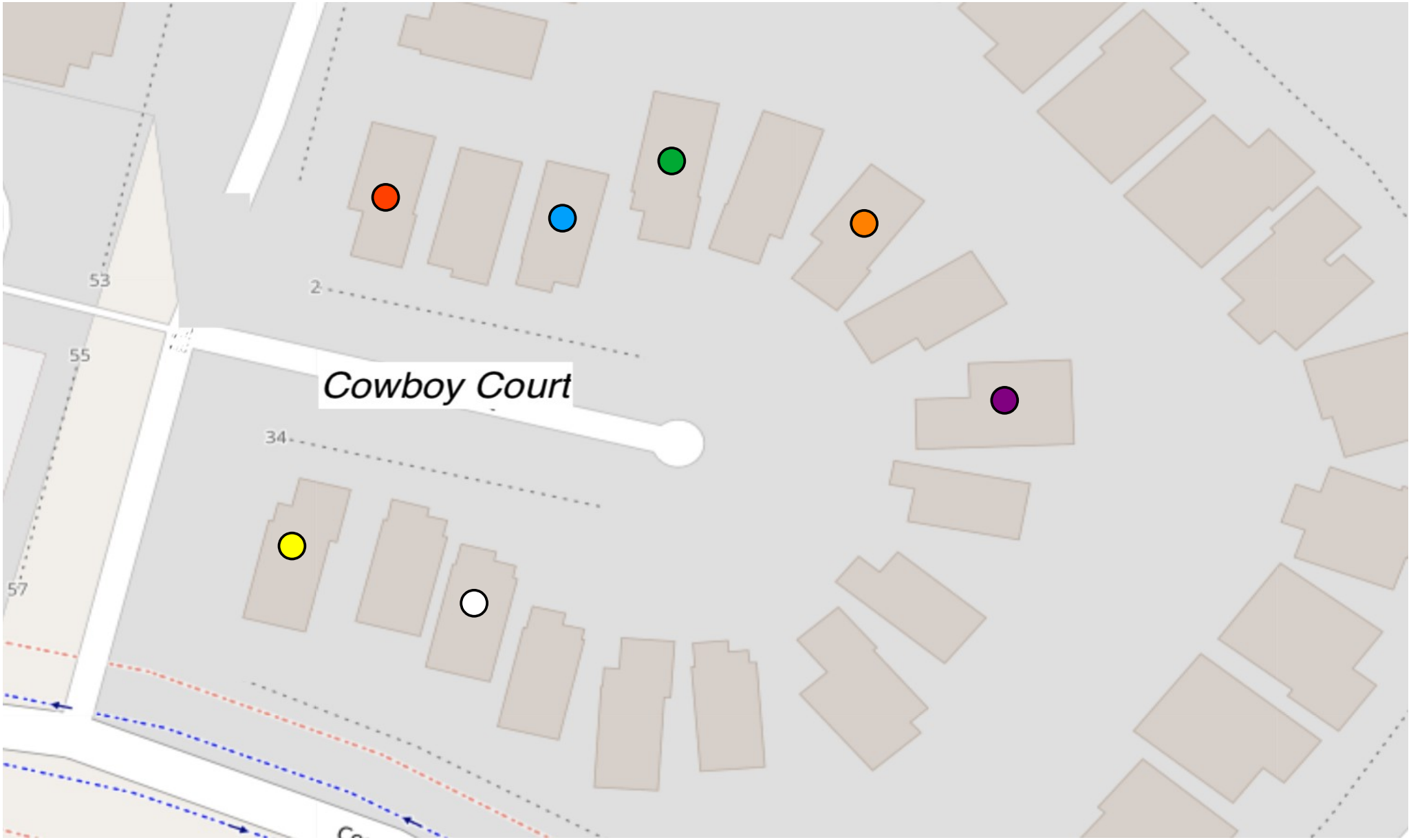
Base Address

- **Question:** *something something*… Base address… *something something*?
  - Asked by Liam
- **Answer:** Think about it like this
  - <*name*> takes my lunch money, and the only thing I know is that <*name*> lives at Cowboy Court, 5th house down, from where the Court starts
    - Where does <*name*> live?
      - *Next slide*

Cowboy Court



Cowboy Court (Labelled)



Abstraction → Application

- How does it work in practice?
  - i.e. *int.arrays.c*
    - i.e. *diagram.int.array.txt*
  - i.e. *double.arrays.c*
    - i.e. *diagram.double.array.txt*

Immediate(s)

- **Question:** Why is immediate split up?
  - Can't remember who asked this
- **Answer:** To keep the format/structure of the instruction the same
  - Makes decoding easier
    - Consistency is good
  - Size and place/location should be consistent

31 27 26 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]				rd	opcode	U-type
imm[20|10:1|11|19:12]				rd	opcode	J-type

Instruction Type

- **Question:** What type of instruction is ***sw***?
  - What does ***sw*** stand for?
    - Is ***sw*** used for *32-bit* or *64-bit* architecture?
- **Options:**
  - A) R-type
  - B) I-type
  - C) B-type
  - D) U-type
  - E) S-type
  - F) J-type

Question #1 (2.27)

- Translate the following RISC-V assembly code into C. Assume that the variable ***i*** is stored in register *x6*, ***result*** is stored in *x5*, and *x10* holds the base address of an array called ***MemArray***.
  - *Assembly code on next slide*

Question #1 (2.27)

```
.data
MemArray:
        .word   0
        .word   1
          …
        .word   2
        .word   99

.text
            addi x6, x0, 0
            addi x5, x0, 0
            addi x29, x0, 100
    LOOP:
            lw x7 0(x10)
            add x5, x5, x7
            addi x10, x10, 4
            addi x6, x6, 1
            blt x6, x29, LOOP
```

- Recall that:
  - **x6** = *i*
  - **x5** = *result*
  - **X10** = Base address of *MemArray*

Answer #1 (2.27)

- **Solution #1:**

```
int i;
int result = 0;
for (i = 0; i < 100; i++) {
    result += *MemArray;
    MemArray++;
}
return result;
```

- ***Solution #2:***

```
int i;
int result = 0;
for (i = 0; i < 100; i++) {
    result += MemArray[i];
}
return result;
```

Clarifications #1 (2.27)

- **Question:** Can a *while loop* be used instead of a *for loop*?
  - **Answer:** Yes! Semantically speaking, there is little difference between a *for loop* and a *while loop*.
    - Programmers prefer *for loops* because they are quicker to write, <u>BUT</u> not quicker to execute on the CPU
- **Question:** Why are there 2 solutions?
  - **Answer:** Because *C* code to assembly isn't a 1:1 mapping. When you compile *C* code, the output depends on what you've written <u>AND</u> the what the compiler does (i.e. The compiler may make optimizations)

Question #2 (2.31)

- Translate function ***f*** into RISC-V assembly language. Assume the function declaration for ***g*** is ***int g(int a, int b)***.

  The code for function ***f*** is as follows:

```
int f (int a, int b, int c, int d) {
    return g(g(a,b), c+d);
}
```

- *Note: For a question like this, you will have to specify registers yourself. (i.e. I will assume that variable "abc" is stored in register "yyz".*
  - *Thus, you will need to reference the RISC-V Calling Convention*

Calling Convention

RISC-V Calling Convention			
Register	ABI Name	Saver	Description
x0	zero	---	Hard-wired zero
x1	ra	Caller	Return address
x2	sp	Callee	Stack pointer
x3	gp	---	Global pointer
x4	tp	---	Thread pointer
x5-7	t0-2	Caller	Temporaries
x8	s0/fp	Callee	Saved register/frame pointer
x9	s1	Callee	Saved register
x10-11	a0-1	Caller	Function arguments/return values
x12-17	a2-7	Caller	Function arguments
x18-27	s2-11	Callee	Saved registers
x28-31	t3-t6	Caller	Temporaries

Answer #2 (2.31)

f:

```
addi x2, x2, -8        // Allocate stack space for 2 words
sw x1, 0(x2)           // Save return address
add x5, x12, x13       // x5 = c + d
sw x5, 4(x2)           // Save x5, which is c + d, on the stack
jal x1, g              // Call x10 = g(a,b)
lw x11, 4(x2)          // Reload x11 = c + d, from the stack
jal x1, g              // Call x10 = g(g(a,b), (c + d))
lw x1, 0(x2)           // Restore return address
addi x2, x2, 8         // Restore stack pointer
jalr x0, 0(x1)         // Return to caller, f
```

Question #3 (2.37)

- **Question:** Write the RISC-V assembly code to implement the following C code as an atomic “set max” operation using the *lr.d* and *sc.d* instructions. Here, the argument *shvar* contains the address of a shared variable which should be replaced by *y* if *y* is greater than the value it points to. Assume *x10* is address of integer pointed by *shvar* and value of *y* is in *x11*.
- **Code:**
  - *Next slide*

Question #3 (2.37)

- **Code:**

```
void setmax(int* shvar, int y) {
    // Begin critical section
    if (y > *shvar)
        *shvar = y;
    // End critical section}
}
```

Answer #3 (2.37)

- **Solution:**

```
setmax:
    try:
        lr.d x5, (x10)          // Load-reserve *shvar
        bge  x5, x11, release // Skip update if *shvar > y
        addi x5, x11, 0         // *shvar = y;
    release:
        sc.d x7, x5, (x10)    // Check x5 before storing
        bne  x7, x0, try      // If store-conditional failed, try again
        jalr x0, 0(x1)        // Jump to the calling function
```

Question #3 (2.37)

- The best way to understand critial sections is to understand:
  - Mutual exclusion (mutexes)
  - Race conditions
  - Deadlock
  - Etc.
- Why all this extra stuff?
  - Because in computer science, the *how* is more important than the *why*
- Side note: All original Q/A document can be found on Teams
  - Posted by Mingzhe Wang

HAVE AN AMAZING READING WEEK

GOD BLESS YOU ALL

BUT MOSTLY ME

& STAY SAFE

HAVE FUN!