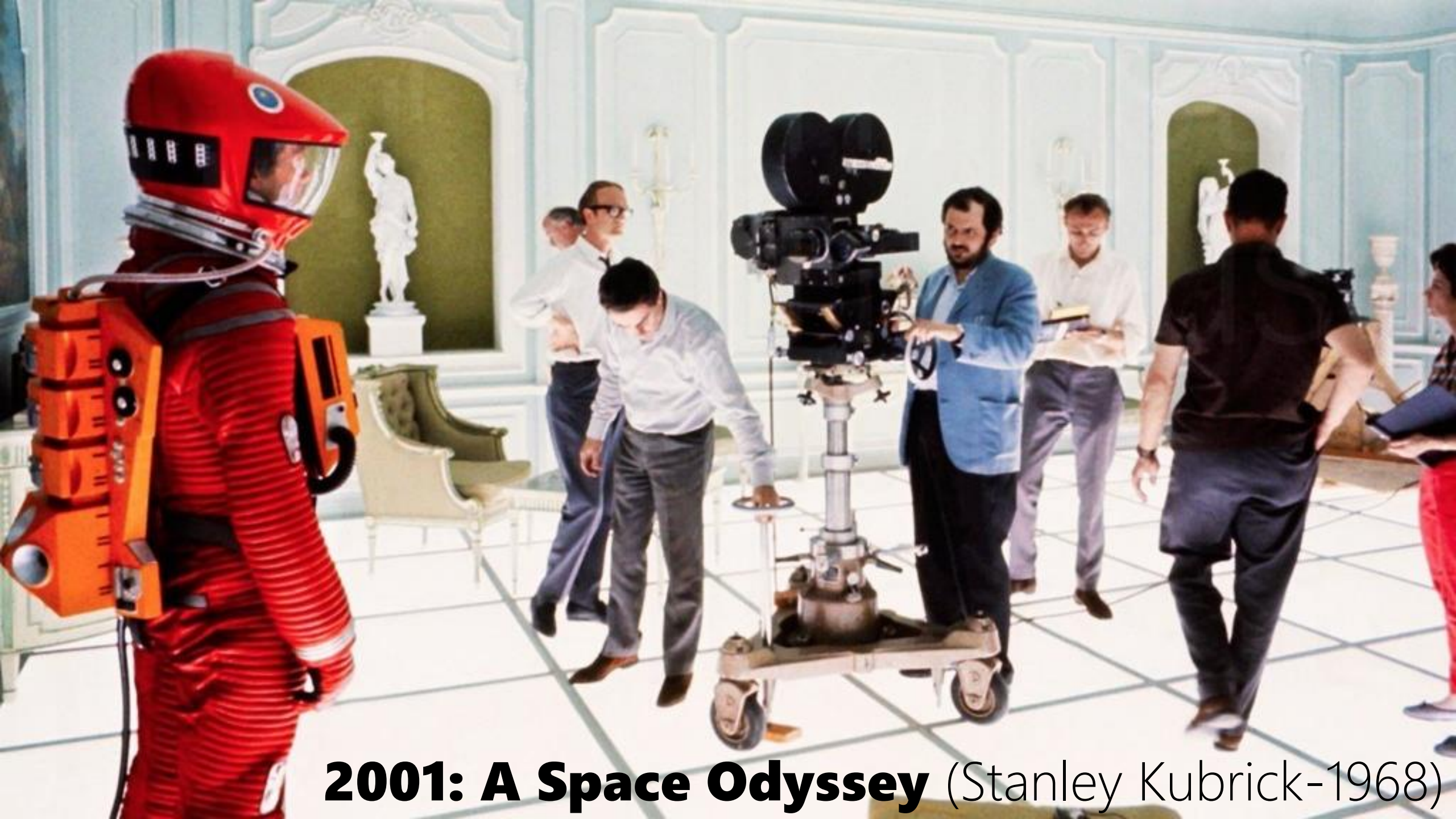
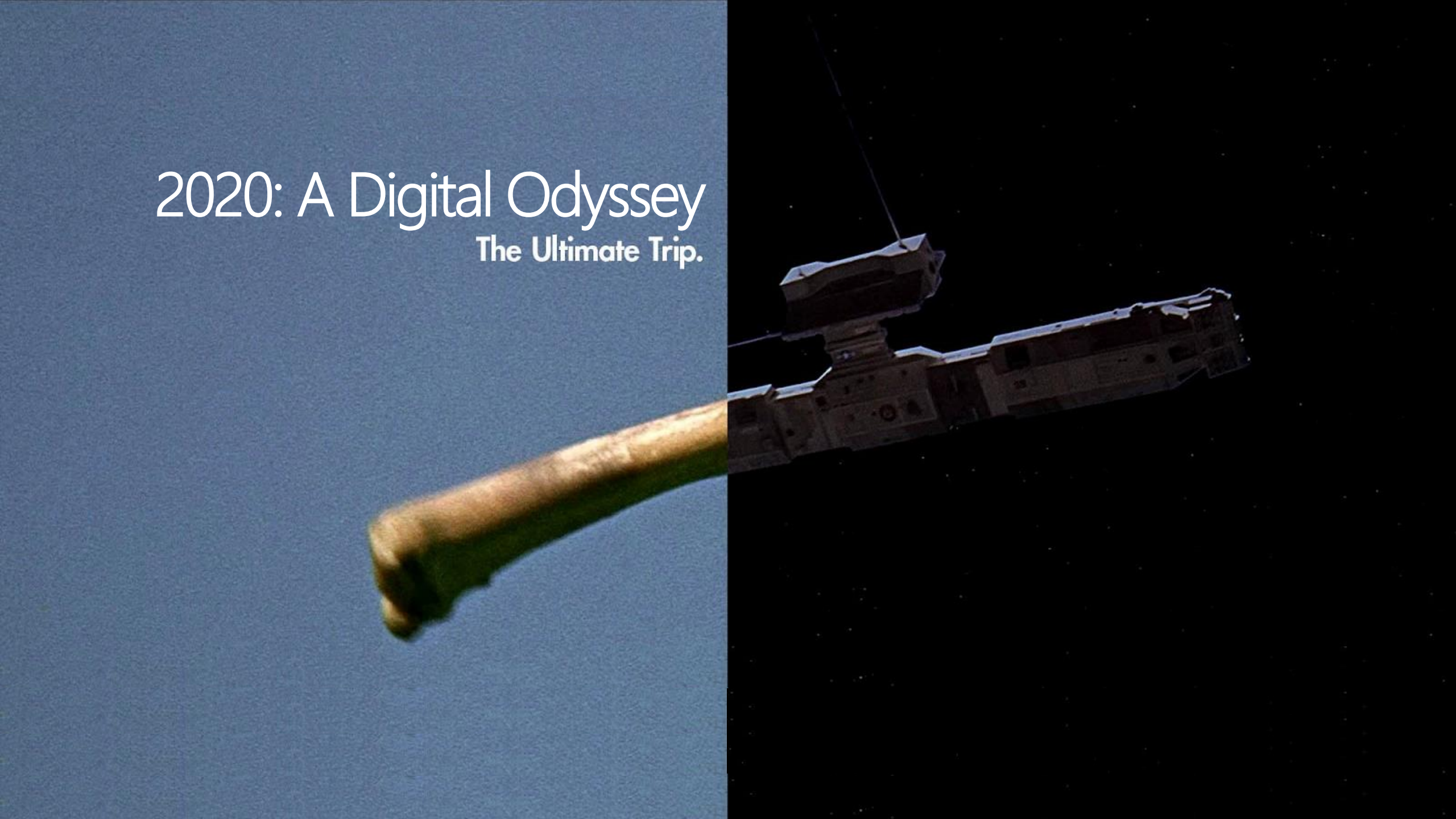


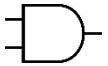
2001: A Space Odyssey (Stanley Kubrick-1968)

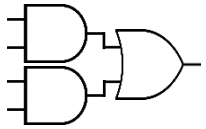
2020: A Digital Odyssey

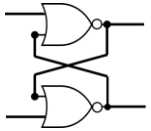
The Ultimate Trip.

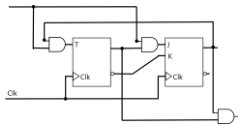


Number Systems | $(12)_{10} \rightarrow (1100)_2$

| Logic Gates | 

| Combinational Logic | 

| Flip-Flops | 

| Sequential Logic | 

| Computer Systems

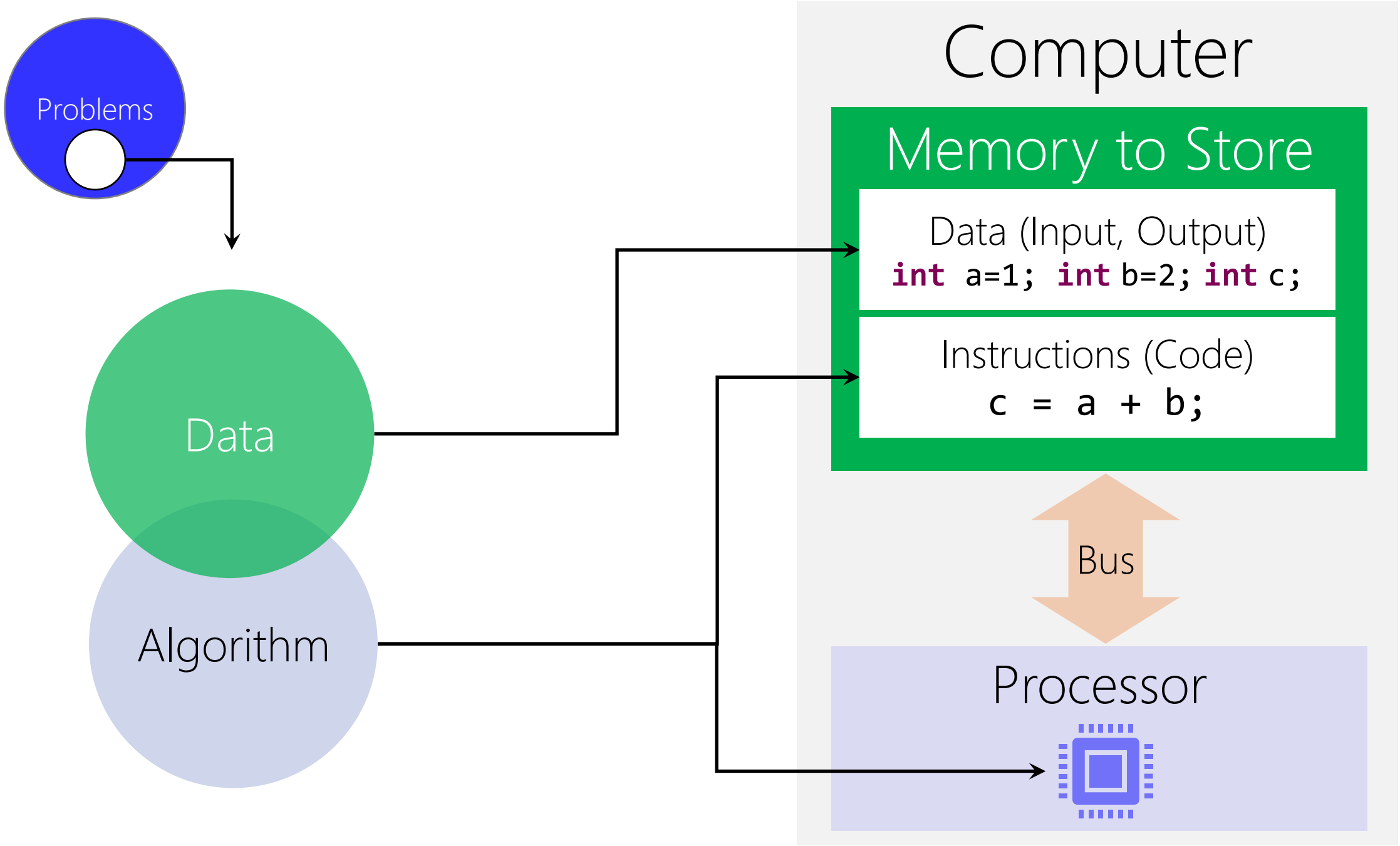
Design a Computer System

John von Neumann

(/vɒn ˈnɔɪmən/)
1903 –1957
Mathematician, Physicist, Computer Scientist, Engineer

Polymath
He integrated pure and applied sciences. He made major contributions to many fields, including:

- Mathematics
- Physics
- Economics (game theory)
- Computing
- Statistics


Problems
Data
Algorithm
Computer
Memory to Store
Data (Input, Output)
int a=1; int b=2; int c;
Instructions (Code)
c = a + b;
Bus
Processor

von Neumann Architecture

Principles

- Data and instructions are both stored in the main memory
- The content of the memory is addressable by location (regardless of what is stored in that location)
- Instructions are executed sequentially unless the order is explicitly modified

Computer System

Computer

Memory to Store

Data (Input, Output)

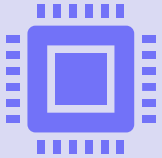
```
int a=1; int b=2; int c;
```

Instructions (Code)

```
c = a + b;
```

Bus

Processor



Input/Output Devices

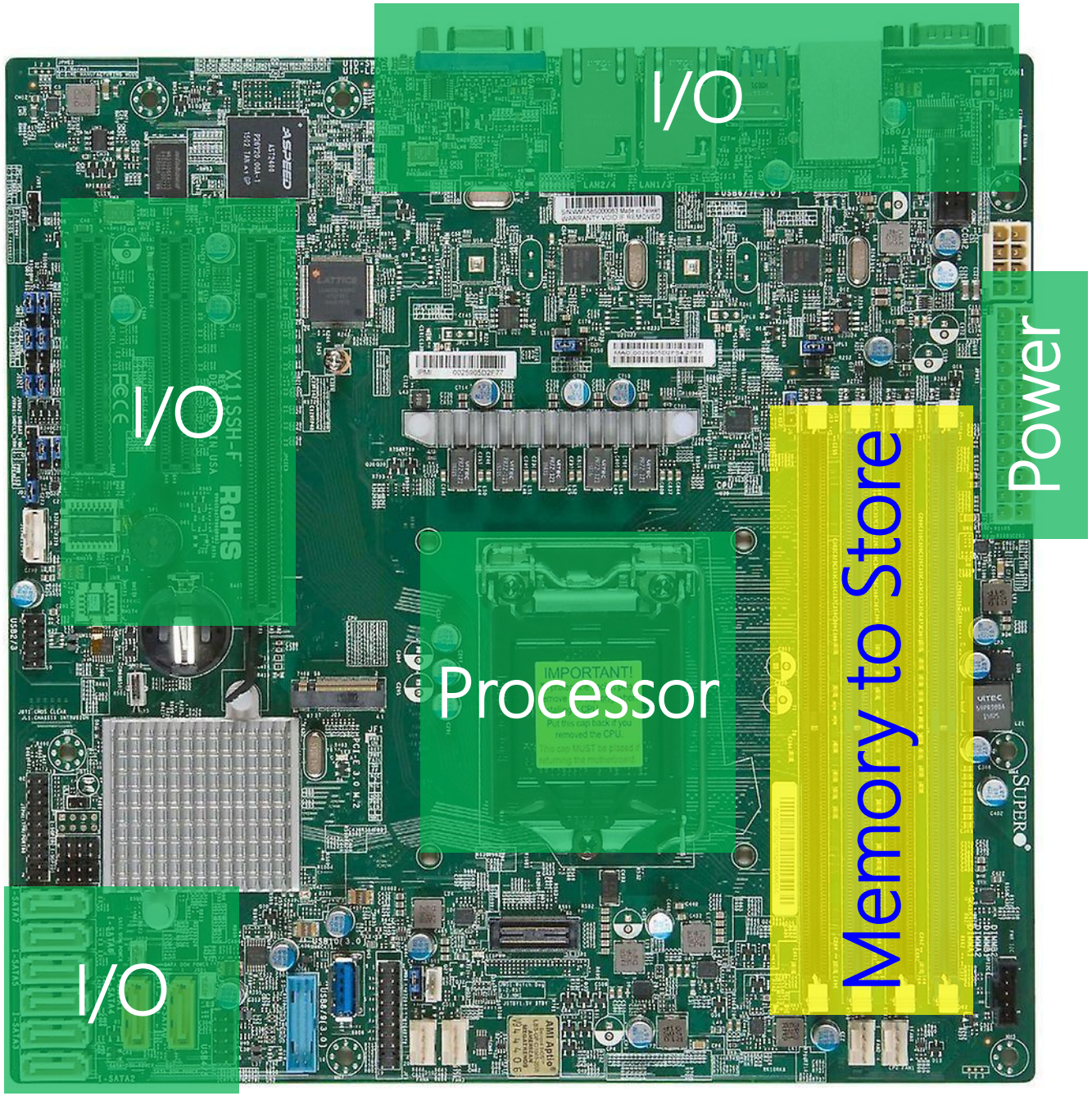
```
scanf("%d", &a);
scanf("%d", &b);
printf("%d", c);
```

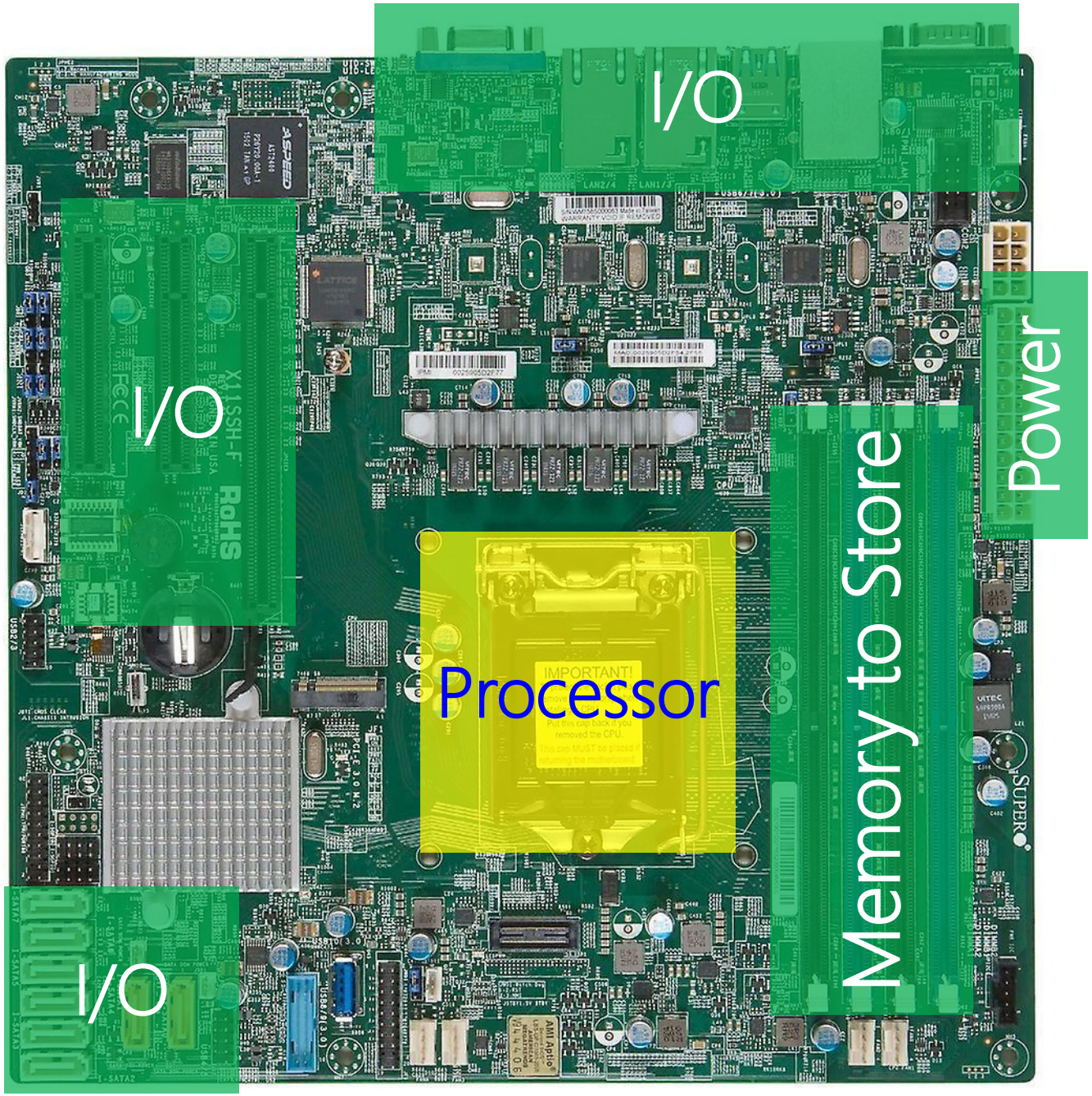
Bus

Bus

Permanent Storage

```
fprintf()
fscanf()
fread()
fwrite()
fseek()
```


I/O
I/O
Power
Processor
Memory to Store
I/O


I/O
I/O
Power
Memory to Store
Processor
I/O

Design Processor

Do Calculation

Design Processor

Calculation on Data → Calculation on Numbers

Design Processor

Number → Binary

Computer

Memory to Store

Data (Input, Output)

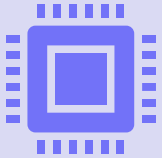
```
int a=1; int b=2; int c;
```

Instructions (Code)

```
c = a + b;
```

Bus

Processor



1. We instruct the processor what to do!
2. Write instructions (programs)
3. Processor does our instructions

Design Processor

Instruction Set

1. Cannot instruct a processor to do whatever we want!
2. Any processors have limitations.
    1. Some can only do addition,
    2. Some can do both addition and subtraction, but no division

Intel® 64 and IA-32 Architectures Software Developer's Manual

Volume 2 (2A, 2B, 2C & 2D): Instruction Set Reference, A-Z

https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual-325383.pdf

NOTE: The Intel 64 and IA-32 Architectures Software Developer's Manual consists of three volumes: *Basic Architecture*, Order Number 253665; *Instruction Set Reference A-Z*, Order Number 325383; *System Programming Guide*, Order Number 325384. Refer to all three volumes when evaluating your design needs.

Design Processor

Assembly Language

Writing programs using the Instruction Set of a particular processor

- RISC (/rɪsk/): Reduced Instruction Set Computer
- CISC (/ˈsɪsk/): Complex Instruction Set Computer

Design Processor

RISC

Small but highly optimized set of instructions
e.g., integer calculation

https://en.wikipedia.org/wiki/Reduced_instruction_set_computer

Design Processor

CISC

Large and NOT highly optimized set of instructions
e.g., integer and floating-point calculations

https://en.wikipedia.org/wiki/Complex_instruction_set_computer

Current Processors

x86, x64

By Intel and AMD

instruction set size: ~981

How Many x86-64 Instructions Are There Anyway?

Human Natural Language	Type	Example
High Level Programing Language	Imperative (What)	SQL `SELECT * FROM Students`

Middle Level Programming Language	Declarative (What + How)	C, C++, Java, Python
		`int a = 1;`
		`a = a + 1;`

Low Level Programming Language (Assembly Language)		Assembly x86, Assembly Z80
		`MOV 1, AX`
		`INC AX`

Binary Digits (Machine Language)		`00010011` `00001011`

Human Natural Language		Example
High Level Programing Language	Impera	SQL `SELECT * FROM Students`

Middle Level Programming Language	Declar ow)	C, C++, Java, Python
		`int a = 1;`
		`a = a + 1;`

Low Level Programming Language (Assembly Language)		Assembly x86, Assembly Z80
		`MOV 1, AX`
		`INC AX`

Binary Digits (Machine Language)		`00010011` `00001011`

Compilers
Interpreters

C/C++

Hossein's Computer

eclipse-workspace - COMP2650_W13_hfani/src/COMP2650_W13_hfani.c - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Debug | COMP2650_W13_hfani.exe

Debug | Project Explorer

- COMP2650_W13_hfani.exe [C/C++ Application]
  - COMP2650_W13_hfani.exe [13164]
    - Thread #1 [COMP2650_W13_hfani] 0 (Suspende
      - main() at COMP2650_W13_hfani.c:2 0x10040
    - Thread #2 0 (Suspended : Container)
    - Thread #3 0 (Suspended : Container)
    - Thread #4 [sig] 0 (Suspended : Container)
  - gdb (8.3.1)

COMP2650_W13_hfani.c | main() at COMP2650_W13_hfani.c:2 0x10040108d

```c
1 int main(void) {
2     int a=1;
3     int b=2;
4     int c;
5     c = a + b;
6
7     return 0;
8 }
9
```

Variables | Breakpoints | Expressions | Modules | Disassembly

Enter location here

```
000000010040107e:   nop
000000010040107f:   nop
                    main:
0000000100401080:   push    %rbp
0000000100401081:   mov     %rsp,%rbp
0000000100401084:   sub     $0x30,%rsp
0000000100401088:   callq   0x1004010d0 <__main>
000000010040108d:   movl    $0x1,-0x4(%rbp)
0000000100401094:   movl    $0x2,-0x8(%rbp)
000000010040109b:   mov     -0x4(%rbp),%edx
000000010040109e:   mov     -0x8(%rbp),%eax
00000001004010a1:   add     %edx,%eax
00000001004010a3:   mov     %eax,-0xc(%rbp)
00000001004010a6:   mov     $0x0,%eax
00000001004010ab:   add     $0x30,%rsp
00000001004010af:   pop     %rbp
00000001004010b0:   retq
00000001004010b1:   nop
00000001004010b2:   nop
00000001004010b3:   nop
```

Console | Registers | Problems | Executables | Debugger Console | Memory

COMP2650_W13_hfani.exe [C/C++ Application]

Writable | Smart Insert | 3 : 1 [10]

eclipse-workspace - COMP2650_W13_hfani/src/COMP2650_W13_hfani.c - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Debug | COMP2650_W13_hfani.exe

Debug | Project Explorer

- COMP2650_W13_hfani.exe [C/C++ Application]
  - COMP2650_W13_hfani.exe [13164]
    - Thread #1 [COMP2650_W13_hfani] 0 (Suspende
      - main() at COMP2650_W13_hfani.c:2 0x10040
    - Thread #2 0 (Suspended : Container)
    - Thread #3 0 (Suspended : Container)
    - Thread #4 [sig] 0 (Suspended : Container)
  - gdb (8.3.1)

COMP2650_W13_hfani.c | main() at COMP2650_W13_hfani.c:2 0x10040108d

```c
1 int main(void) {
2     int a=1;
3     int b=2;
4     int c;
5     c = a + b;
6
7     return 0;
8 }
9
```

Variables | Breakpoints | Expressions | Modules | Disassembly

Enter location here

```
000000010040107e:   nop
000000010040107f:   nop
                    main:
0000000100401080:   push    %rbp
0000000100401081:   mov     %rsp,%rbp
0000000100401084:   sub     $0x30,%rsp
0000000100401088:   callq   0x1004010d0 <__main>
000000010040108d:   movl    $0x1,-0x4(%rbp)
0000000100401094:   movl    $0x2,-0x8(%rbp)
000000010040109b:   mov     -0x4(%rbp),%edx
00000001004010a1:   add     %edx,%eax
00000001004010a3:   mov     %eax,-0xc(%rbp)
00000001004010a6:   mov     $0x0,%eax
00000001004010ab:   add     $0x30,%rsp
00000001004010af:   pop     %rbp
00000001004010b0:   retq
00000001004010b1:   nop
00000001004010b2:   nop
```

Console | Registers | Problems | Executables | Debugger Console | Memory

COMP2650_W13_hfani.exe [C/C++ Application]

Writable | Smart Insert | 3 : 1 [10]

eclipse-workspace - COMP2650_W13_hfani/src/COMP2650_W13_hfani.c - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Debug | COMP2650_W13_hfani.exe

Debug | Project Explorer

- COMP2650_W13_hfani.exe [C/C++ Application]
  - COMP2650_W13_hfani.exe [13164]
    - Thread #1 [COMP2650_W13_hfani] 0 (Suspende
      - main() at COMP2650_W13_hfani.c:2 0x10040
    - Thread #2 0 (Suspended : Container)
    - Thread #3 0 (Suspended : Container)
    - Thread #4 [sig] 0 (Suspended : Container)
  - gdb (8.3.1)

COMP2650_W13_hfani.c | main() at COMP2650_W13_hfani.c:2 0x10040108d

```c
1 int main(void) {
2     int a=1;
3     int b=2;
4     int c;
5     c = a + b;
6
7     return 0;
8 }
9
```

Variables | Breakpoints | Expressions | Modules | Disassembly

Enter location here

```
000000010040107e:   nop
000000010040107f:   nop
                    main:
0000000100401080:   push    %rbp
0000000100401081:   mov     %rsp,%rbp
0000000100401084:   sub     $0x30,%rsp
0000000100401088:   callq   0x1004010d0 <__main>
000000010040108d:   movl    $0x1,-0x4(%rbp)
0000000100401094:   movl    $0x2,-0x8(%rbp)
000000010040109b:   mov     -0x4(%rbp),%edx
000000010040109e:   mov     -0x8(%rbp),%eax
00000001004010a1:   add     %edx,%eax
00000001004010a3:   mov     %eax,-0xc(%rbp)
00000001004010a6:   mov     $0x0,%eax
00000001004010ab:   add     $0x30,%rsp
00000001004010af:   pop     %rbp
00000001004010b0:   retq
00000001004010b1:   nop
00000001004010b2:   nop
00000001004010b3:   nop
```

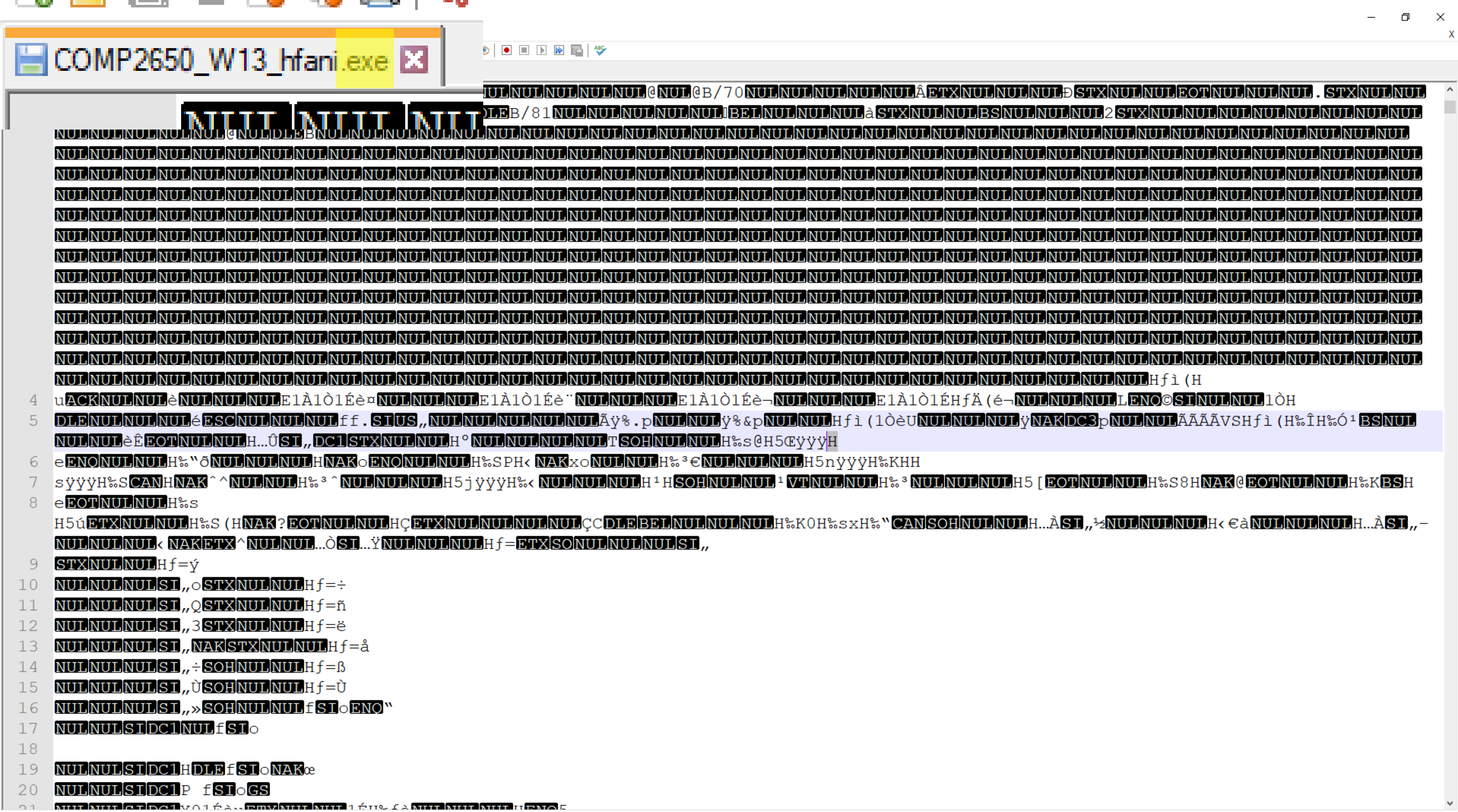
Memory Addresses

Console | Registers | Problems | Executables | Debugger Console | Memory

COMP2650_W13_hfani.exe [C/C++ Application]

To see the actual binary machine language (OP Code) for each instruction, open your executable file that the compiler makes. E.g., *.exe in Windows systems.

Writable | Smart Insert | 3 : 1 [10]



COMP2650_W13_hfani.exe

```
ULNULNULNULNUL@NUL@B/70NULNULNULNULNULÂETXNULNULNULÐSTXNULNULEOTNULNULNUL.STXNULNUL
LEB/81NULNULNULNULNUL|BELNULNULNULàSTXNULNULBSNULNULNUL2STXNULNULNULNULNULNULNULNUL
NULNULNULNULNUL@NULDLEBNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNUL
NULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNUL
NULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNUL
NULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNUL
NULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNUL
NULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNUL
NULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNUL
NULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNUL
NULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNUL
NULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNUL
NULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNUL
NULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNUL
NULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULHƒì(H
uACKNULNULèNULNULNULE1À1Ò1Éè¤NULNULNULE1À1Ò1Éè¨NULNULNULE1À1Ò1Éè¬NULNULNULE1À1Ò1ÉHƒÄ(é¬NULNULNULLENQ©SINULNUL1ÒH
DLENULNULNULéESCNULNULNULff.SIUS„NULNULNULNULNULÃÿ%.pNULNULÿ%&pNULNULHƒì(1ÒèUNULNULNULÿNAKDC3pNULNULÃÃÃÃVSHƒì(H‰ÎH‰Ó¹BSNUL
NULNULèÊEOTNULNULH…ÛSI„DC1STXNULNULH°NULNULNULNULTSOHNULNULH‰s@H5ŒÿÿÿH
eENQNULNULH‰"ðNULNULNULHNAKoENQNULNULH‰SPH‹NAKxoNULNULH‰³€NULNULNULH5nÿÿÿH‰KHH
sÿÿÿH‰SCANHNAK^^NULNULH‰³^NULNULNULH5jÿÿÿH‰‹NULNULNULH¹HSOHNULNUL¹VTNULNULH‰³NULNULNULH5[EOTNULNULH‰S8HNAK@EOTNULNULH‰KBSH
eEOTNULNULH‰s
H5úETXNULNULH‰S(HNAK?EOTNULNULHÇETXNULNULNULNULÇCDLEBELNULNULNULH‰K0H‰sxH‰"CANSOHNULNULH…ÀSI„½NULNULNULH‹€àNULNULNULH…ÀSI„–
NULNULNUL‹NAKETX^NULNUL…ÒSI…ŸNULNULNULHƒ=ETXSONULNULNULSI„
STXNULNULHƒ=ý
NULNULNULSI„oSTXNULNULHƒ=÷
NULNULNULSI„QSTXNULNULHƒ=ñ
NULNULNULSI„3STXNULNULHƒ=ë
NULNULNULSI„NAKSTXNULNULHƒ=å
NULNULNULSI„÷SOHNULNULHƒ=ß
NULNULNULSI„ÙSOHNULNULHƒ=Ù
NULNULNULSI„»SOHNULNULfSIoENQ"
NULNULSIDC1NULfSIo

NULNULSIDC1HDLEfSIoNAKœ
NULNULSIDC1P fSIoGS
```

Program vs. Process

- Program: dead body of the instructions stored in permanent place (flash drive, ssd, hard disk, paper!)
- Process: live body of the instructions stored in memory, *ready to be fetched by the Processor and be executed*!

Program vs. Process

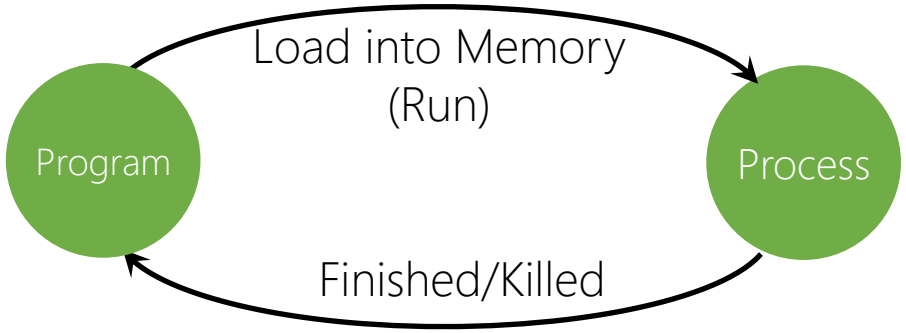
C, C++, Java, Python

```
int a = 1;
a = a + 1;
```

Assembly x86, Assembly Z80

```
MOV 1, AX
INC AX
```

```
00010011
00001011
```




```
00010011
00001011
```

```
00010011
00001011
```

```
00010011
00001011
```

Compilers vs. Interpreters

Instruction Encoders

Compilers

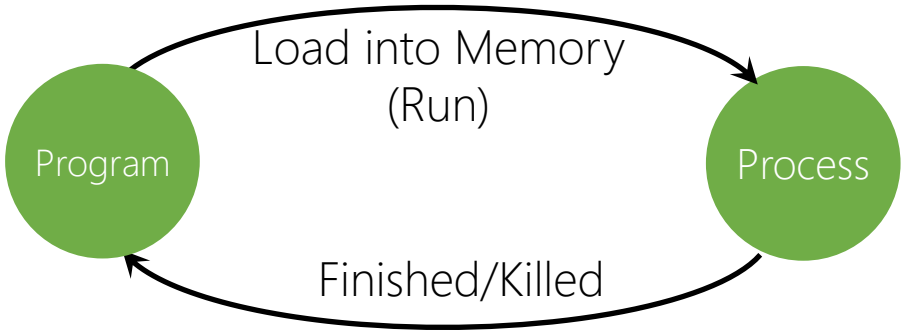
C, C++, Java, Python

```
int a = 1;
a = a + 1;
```

Assembly x86, Assembly Z80

```
MOV 1, AX
INC AX
```

```
00010011
00001011
```



Interpreters

```
int a = 1; → 00010011
a = a + 1; → 00001011
```

Instruction	Operation Code (OP Code)
c = a + b	000 XXXX YYYY ZZZZ
c = a – b	001 XXXX YYYY ZZZZ
c = a / b	010 XXXX YYYY ZZZZ
c = a * b	011 XXXX YYYY ZZZZ
c = {number}	100 XXXX XXX XXXX
…	…

Instruction Encoders very simplified version!

Design Processor

Processor can only see binary-coded instructions

Processor only understand machine language

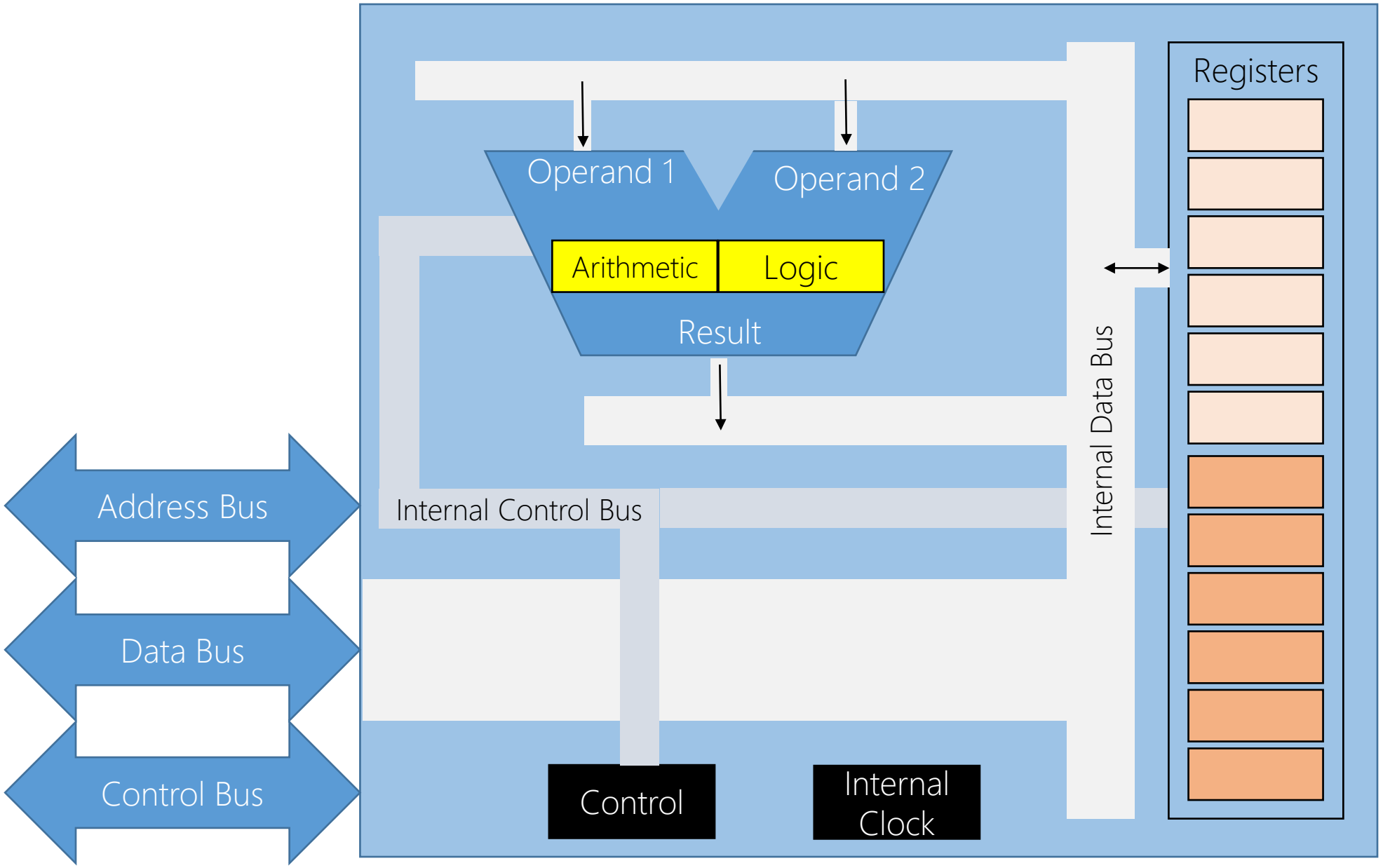
Instruction Set →Instruction Decoder

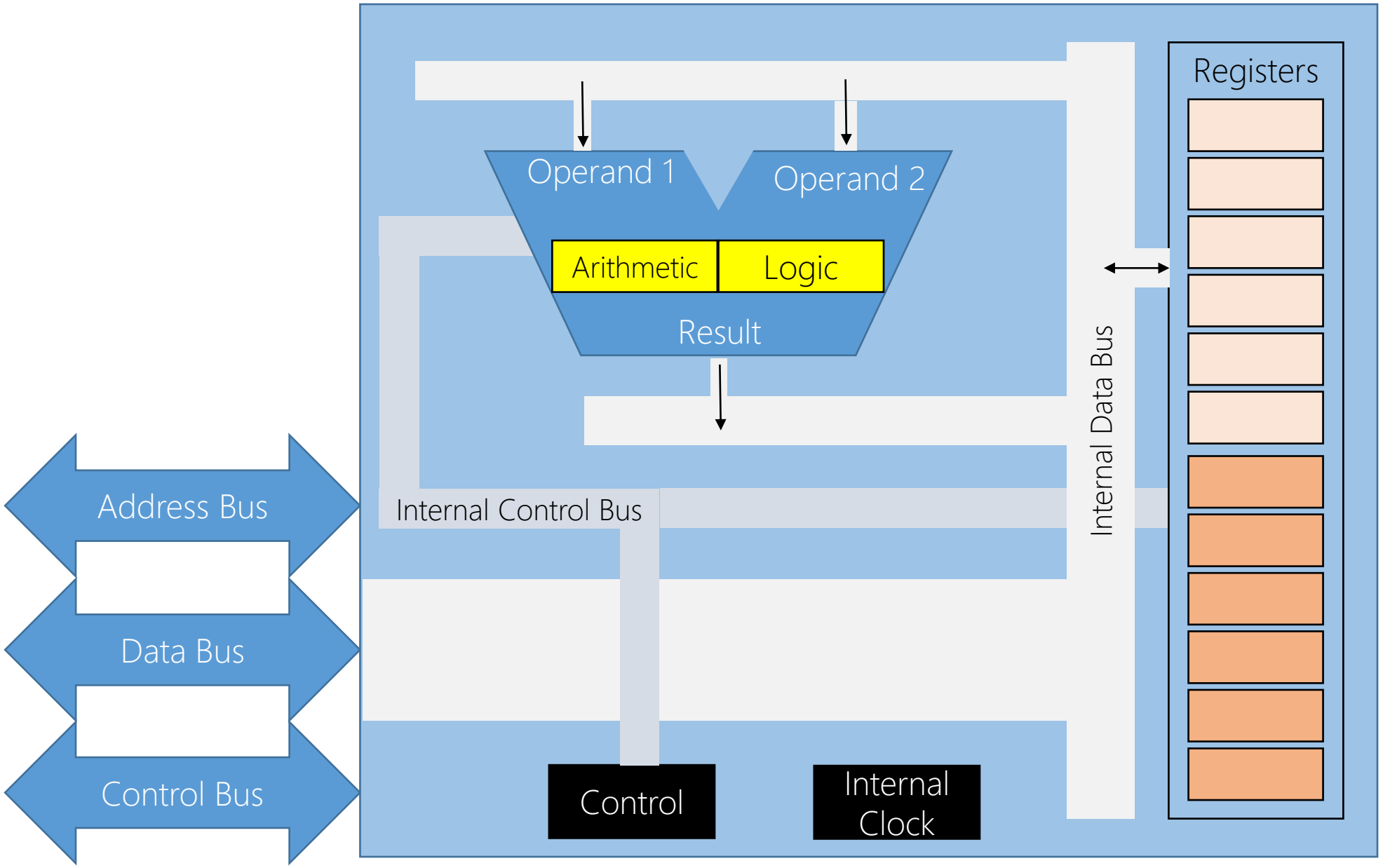
Operation Code (OP Code)	What to do?
000 XXXX YYYY ZZZZ	1) Fetch the first operand from memory at XXXX address 2) Store the first operand inside somewhere (AX) 3) Fetch the second operand from memory at YYYY address 4) Store the first operand inside somewhere else (BX) 5) Use the n-bit Adder to add AX and BX 6) Store the result inside somewhere else (CX) 7) Push CX to memory at ZZZZ address
001 XXXX YYYY ZZZZ	
010 XXXX YYYY ZZZZ	
011 XXXX YYYY ZZZZ	
100 XXXX XXX XXXX	
…	

Instruction Decoder very simplified version!

Operation Code (OP Code)	What to do?
000 XXXX YYYY ZZZZ	1) Fetch the first operand from memory at XXXX address 2) Store the first operand inside somewhere (AX) 3) Fetch the second operand from memory at YYYY address 4) Store the first operand inside somewhere else (BX) 5) Use the n-bit Adder to add AX and BX 6) Store the result inside somewhere else (CX) 7) Push CX to memory at ZZZZ address
001 XXXX YYYY ZZZZ	
010 XXXX YYYY ZZZZ	
011 XXXX YYYY ZZZZ	
100 XXXX XXX XXXX	
…	

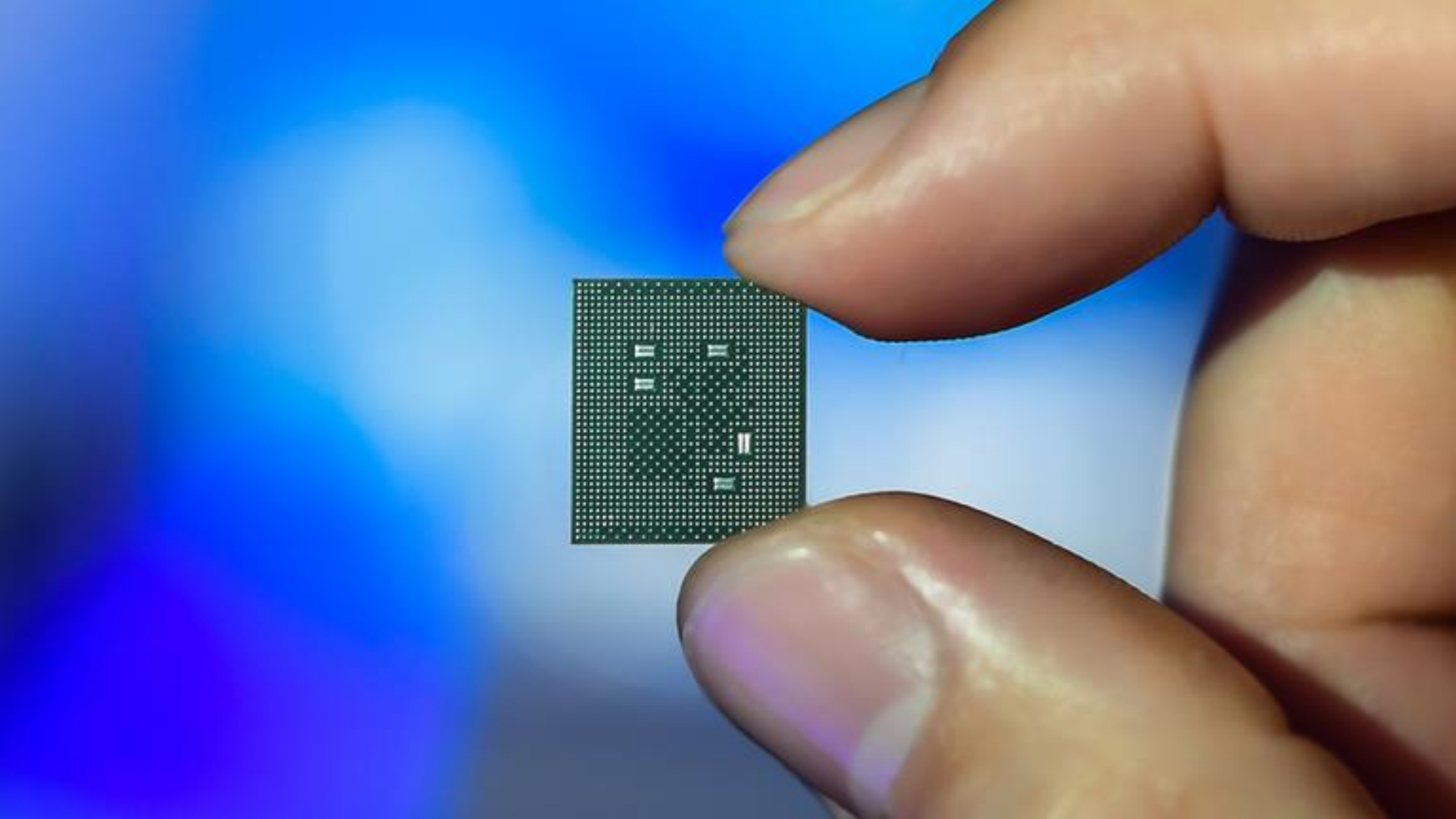
Instruction Decoder
very simplified version!


Registers
Operand 1
Operand 2
Arithmetic
Logic
Result
Internal Data Bus
Address Bus
Internal Control Bus
Data Bus
Control Bus
Control
Internal Clock



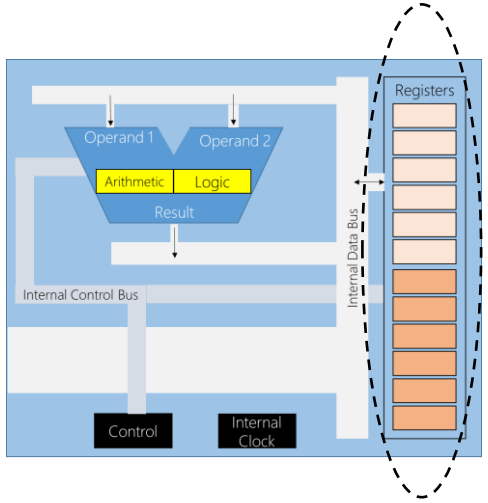
City

- Streets
  - 1-way
  - 2-way
- Gates
- Bridges
- Warehouses
- Policing



Registers

Some vectors of FFs



Instruction Pointer (IP)

aka. Program Counter (PC), Instruction Address Register (IAR), Instruction Counter

Keep track of the address of next instruction in memory

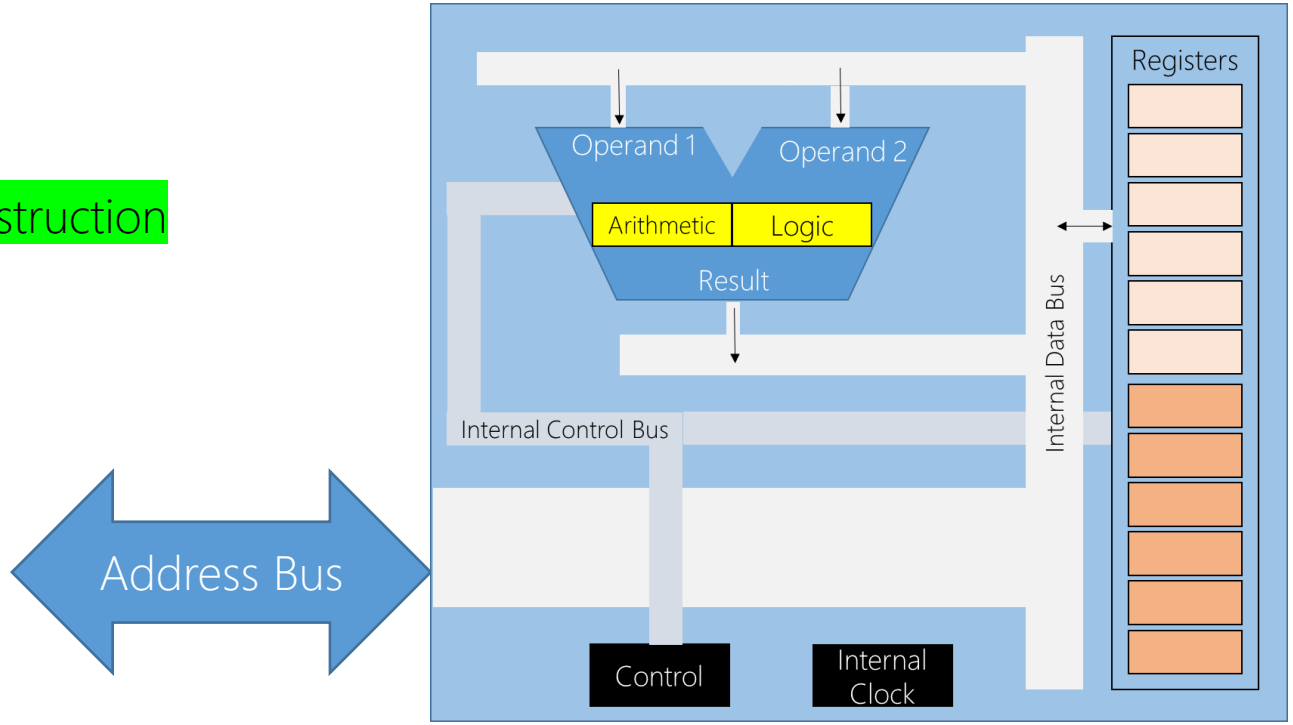
Instruction Pointer (IP)

aka. Program Counter (PC), Instruction Address Register (IAR), Instruction Counter

Keep track of the address of next instruction in memory

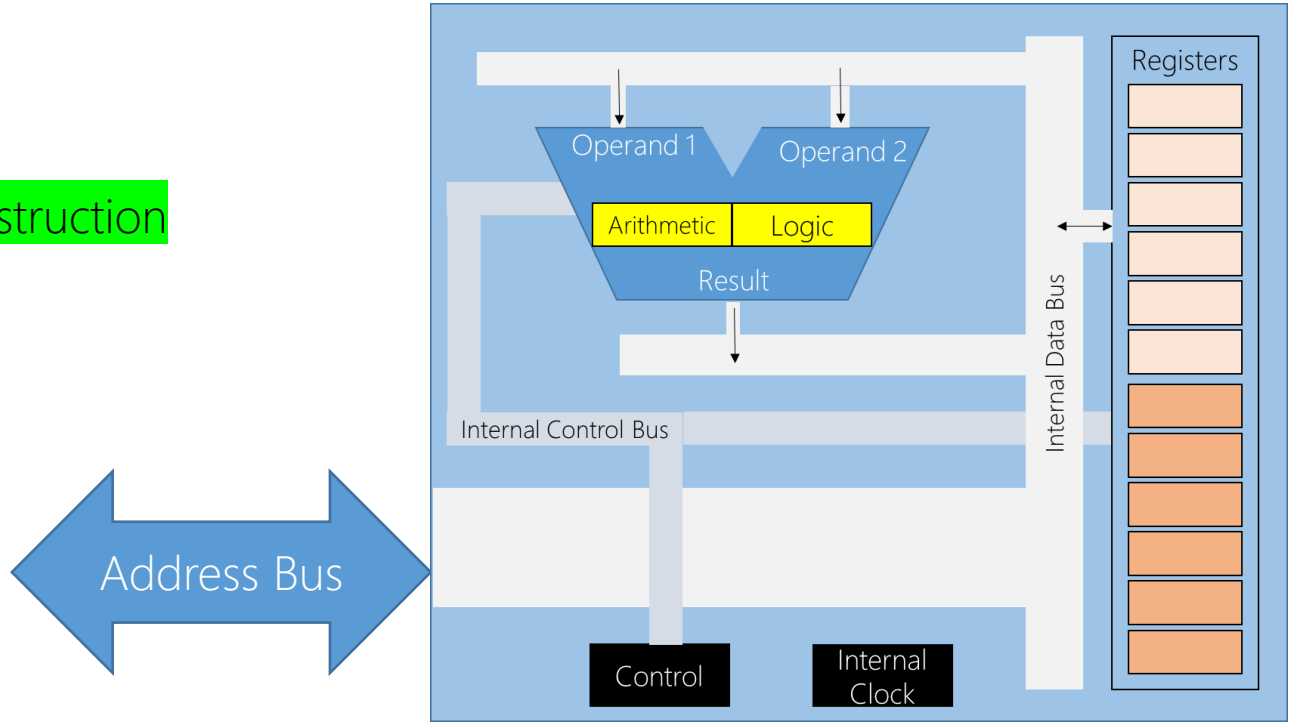
Why is it called ‘counter’?

Address	Memory	High Level Instr.
0000	100 0111 0001 0000	int a = 1;
0001	100 1000 0010 0000	int b = 2; Current Instruction
0010	100 1001 0000 0000	int c;
0011	001 0111 1000 1001	c = a + b;
0100	...	
...	....	



Where is the next instruction to be fetched and execute?

Address	Memory	High Level Instr.
0000	100 0111 0001 0000	int a = 1;
0001	100 1000 0010 0000	int b = 2; Current Instruction
0010	100 1001 0000 0000	int c;
0011	001 0111 1000 1001	c = a + b;
0100	100 0111 0001 0000	…
…	….	…



In von Neumann architecture, instructions are executed sequentially!
So, the next would be the address of current + 1

Instruction Pointer (IP)

aka. Program Counter (PC), Instruction Address Register (IAR), Instruction Counter

We already designed a 3-bit counter

00, 01, 10, 11

Instruction Pointer (IP)

aka. Program Counter (PC), Instruction Address Register (IAR), Instruction Counter

IP is a n-bit counter. What is n?

Instruction Pointer (IP)

aka. Program Counter (PC), Instruction Address Register (IAR), Instruction Counter

IP is a n-bit counter. What is n?

- IP contains a memory address.
- Address bus contains a memory address at a time.

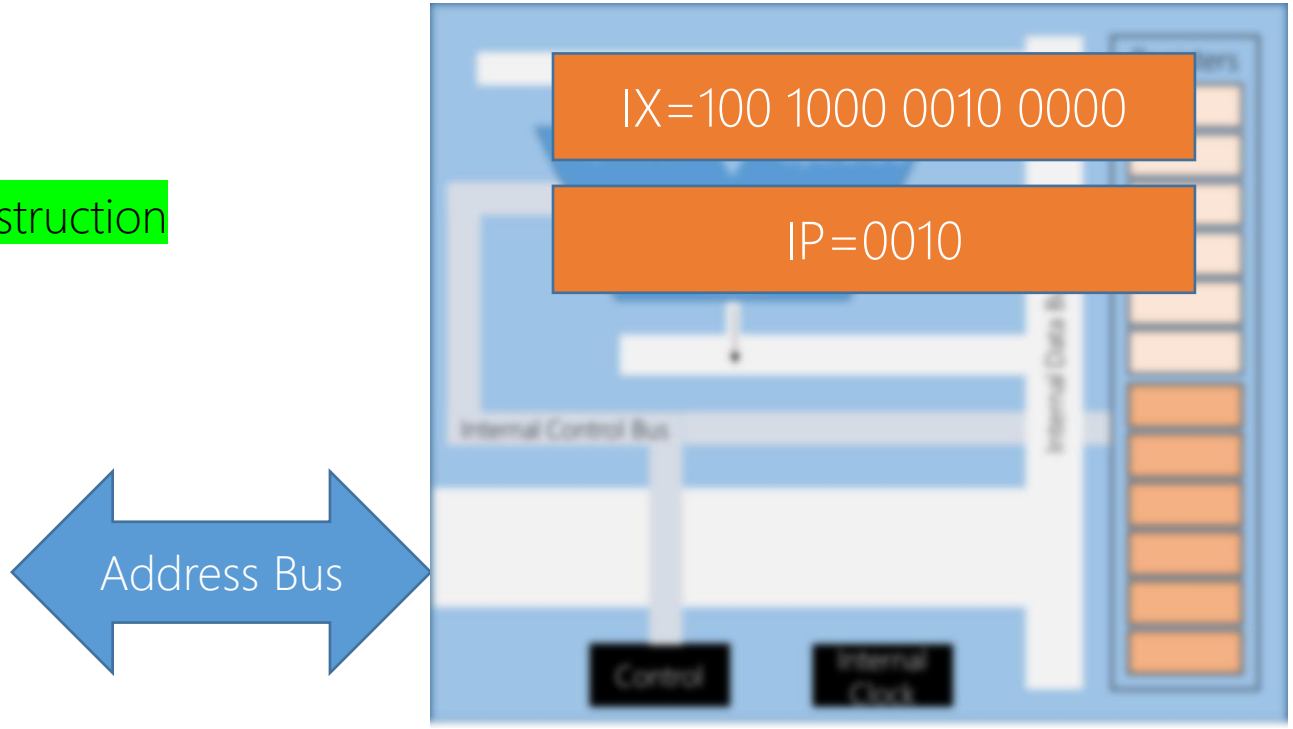
So, IP is the same size as address bus. Here, 4-bit address bus, we have 4-bit IP.

Where is the current instruction?

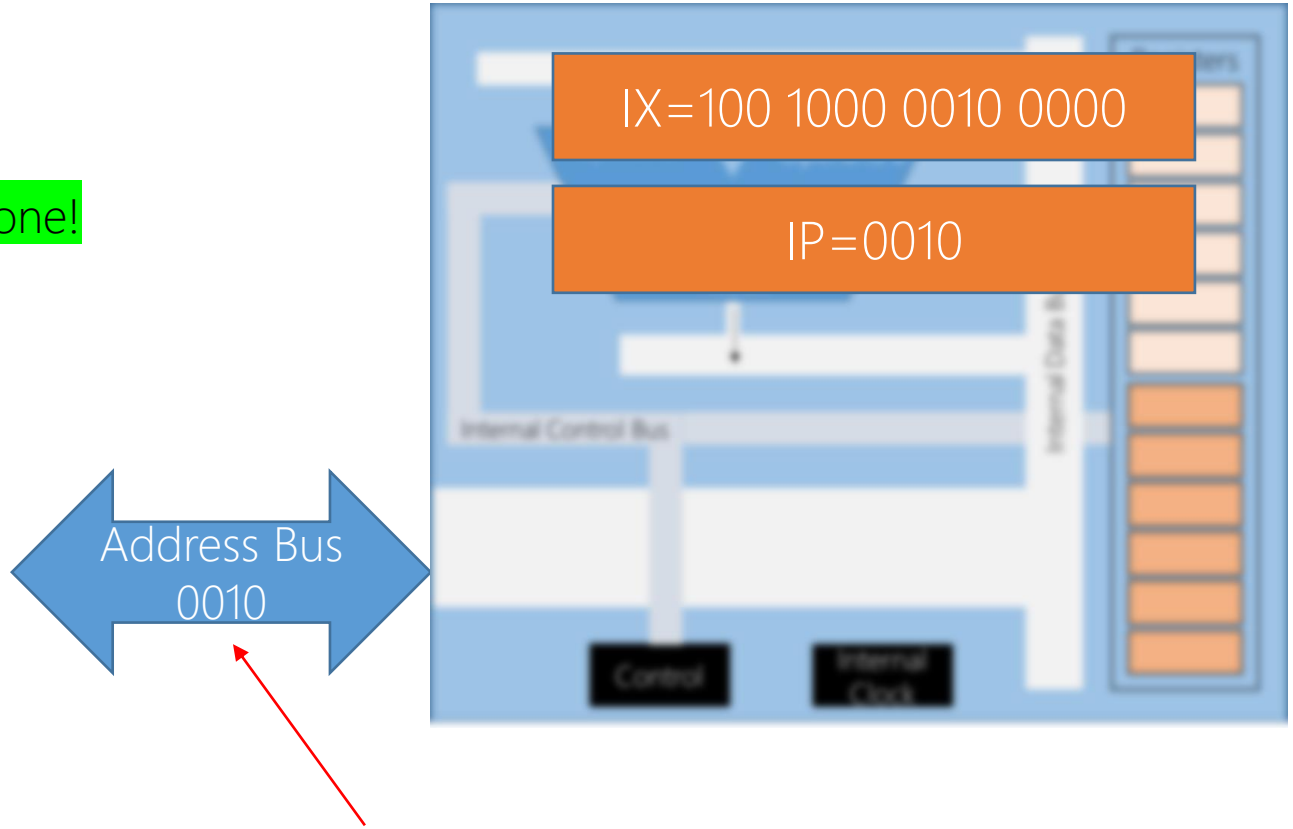
Instruction Register (IR or IX)

- Holds the current instruction that is fetched from memory and is just about to be executed.
- As soon as the instruction placed in IR, the processor starts executing the instruction and IP++ to point to the next instruction to be executed next.

Address	Memory	High Level Instr.
0000	100 0111 0001 0000	int a = 1;
0001	100 1000 0010 0000	int b = 2; Current Instruction
0010	100 1001 0000 0000	int c;
0011	001 0111 1000 1001	c = a + b;
0100	100 0111 0001 0000	…
…	….	…

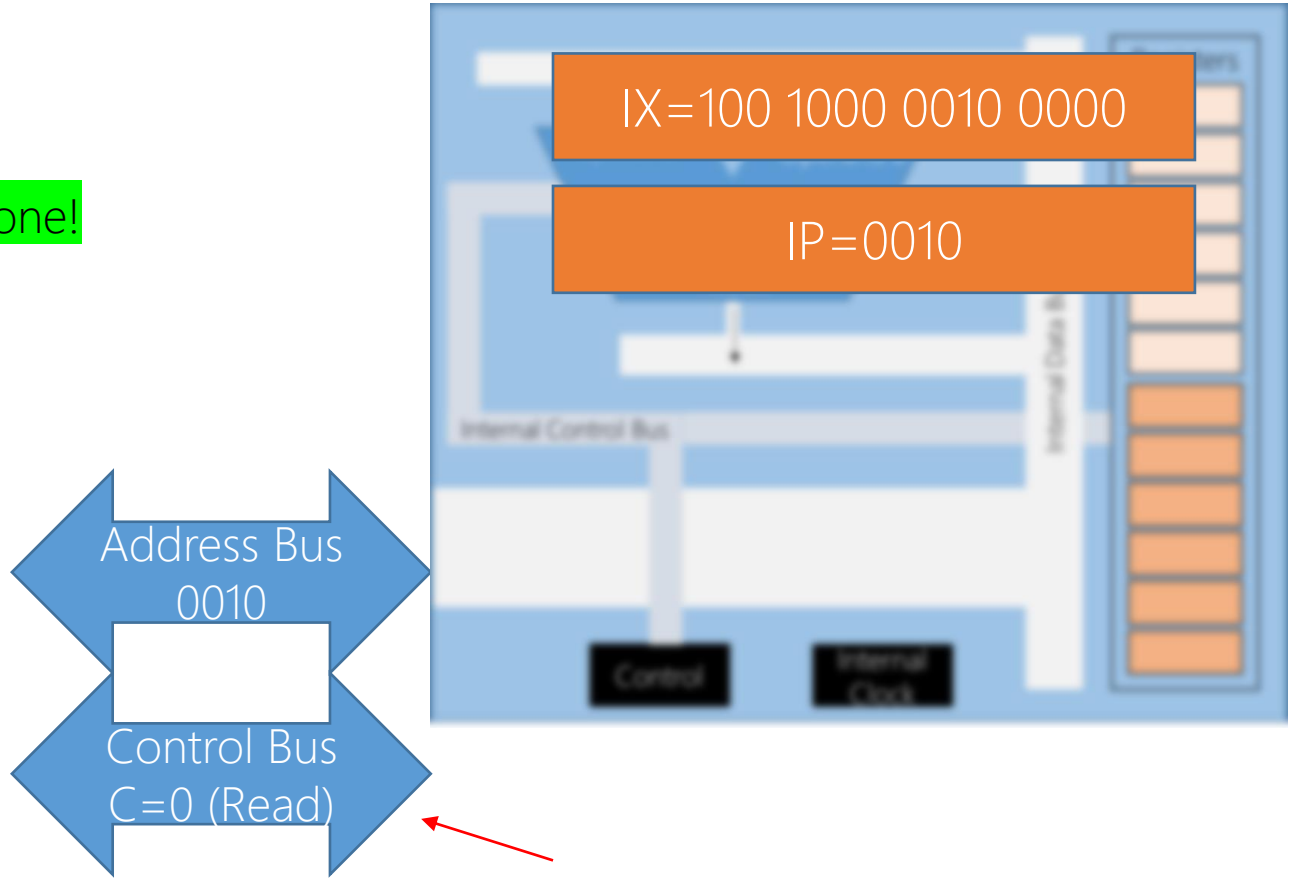


Address	Memory	High Level Instr.
0000	100 0111 0001 0000	int a = 1;
0001	100 1000 0010 0000	int b = 2; Current Done!
0010	100 1001 0000 0000	int c;
0011	001 0111 1000 1001	c = a + b;
0100	100 0111 0001 0000	…
…	….	…



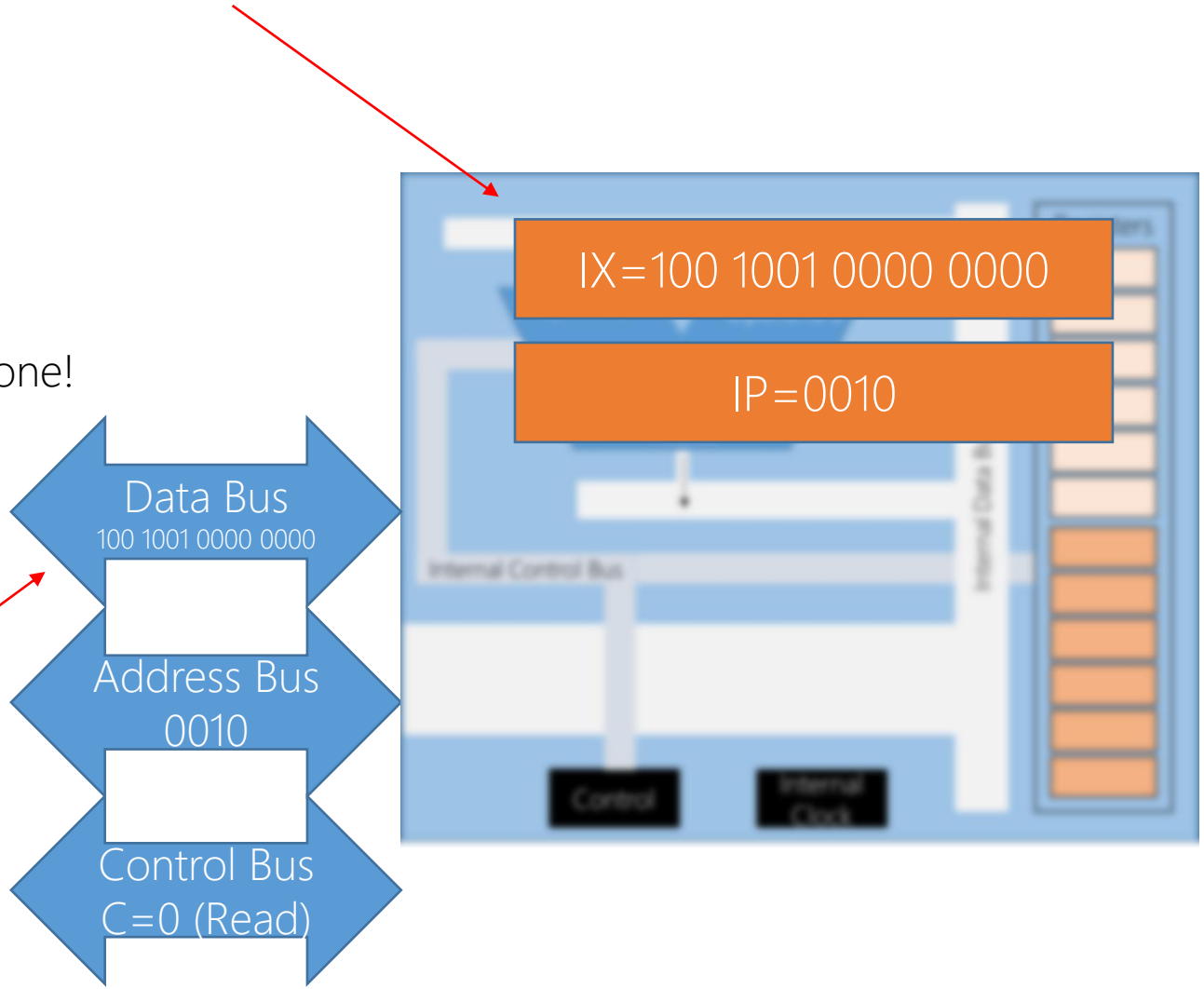
1) Load address bus with IP

Address	Memory	High Level Instr.
0000	100 0111 0001 0000	int a = 1;
0001	100 1000 0010 0000	int b = 2; Current Done!
0010	100 1001 0000 0000	int c;
0011	001 0111 1000 1001	c = a + b;
0100	100 0111 0001 0000	…
…	….	…



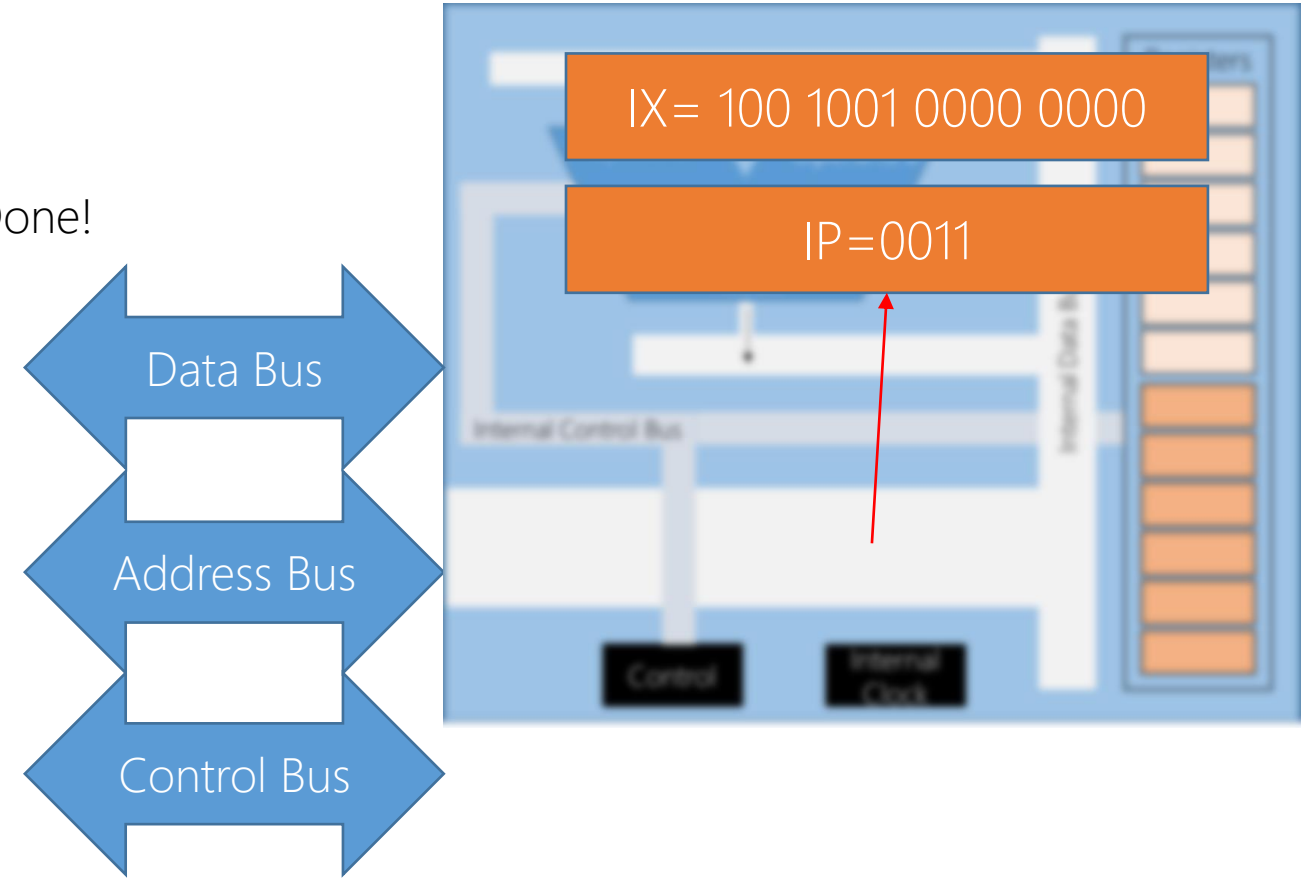
2) Load control bus to read from memory

Address	Memory	High Level Instr.
0000	100 0111 0001 0000	int a = 1;
0001	100 1000 0010 0000	int b = 2; Current Done!
0010	100 1001 0000 0000	int c;
0011	001 0111 1000 1001	c = a + b;
0100	100 0111 0001 0000	…
…	….	…



3) Read data bus and load it into IX

Address	Memory	High Level Instr.
0000	100 0111 0001 0000	int a = 1;
0001	100 1000 0010 0000	int b = 2; Current Done!
0010	100 1001 0000 0000	int c;
0011	001 0111 1000 1001	c = a + b;
0100	100 0111 0001 0000	…
…	….	…



4) Increment IP

Instruction Register (IR or IX)

n-bit Register. What is n?

Fetch Instruction Cycle

1) Load address bus with IP
2) Load control bus to read from memory
3) Read data bus and load it into IX
4) Increment IP

Memory Address Register (MAR)

Holds the address of memory location to Read/Write data from.

Memory Data Register (MDR)

Holds the actual value that is either

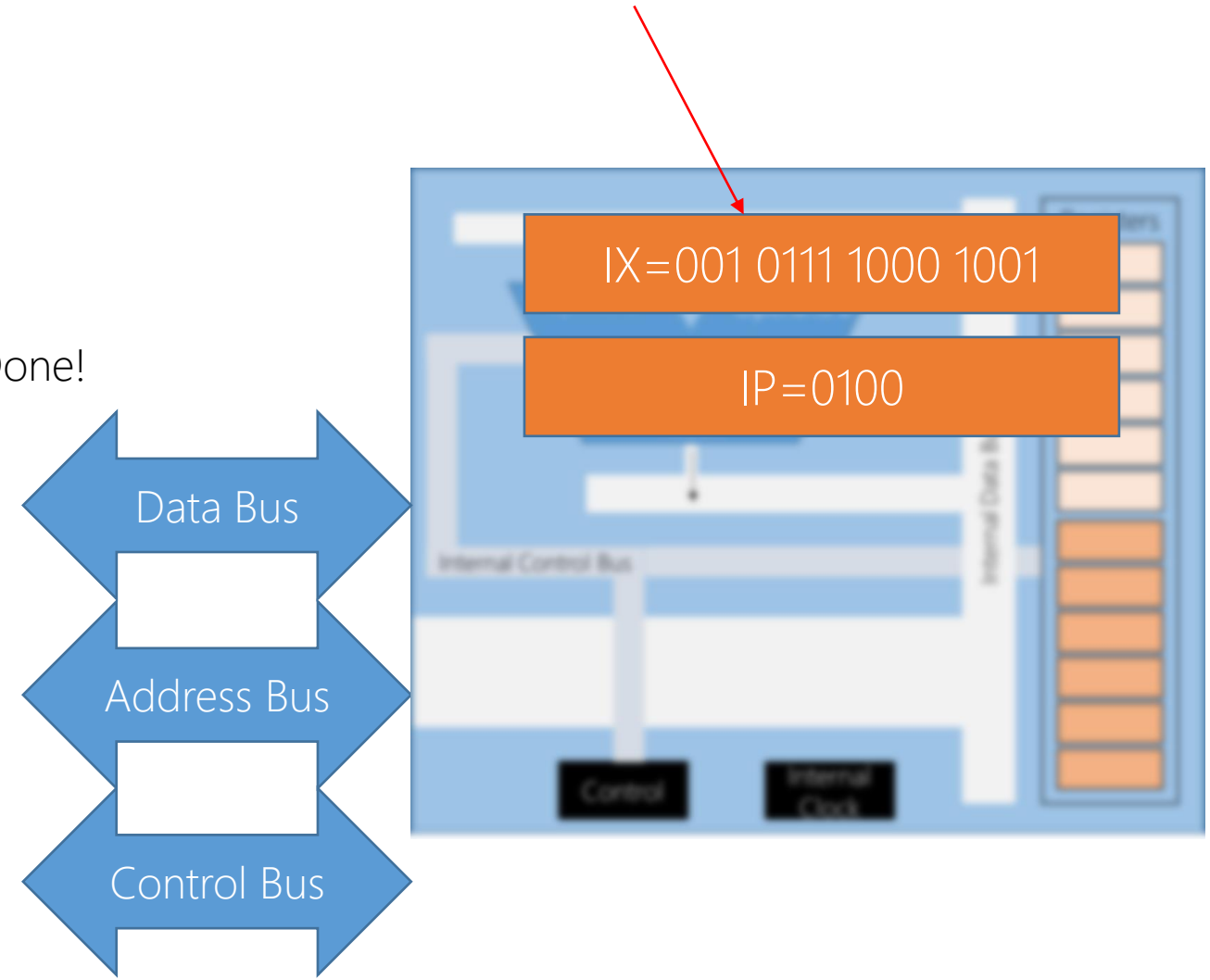
- Read from memory, or
- To be written to memory

Memory Address Register (MAR)
Memory Data Register (MDR)

Work hand-in-hand to facilitate the communication of the processor and the main memory

What are their sizes?

Address	Memory	High Level Instr.
0000	100 0111 0001 0000	int a = 1;
0001	100 1000 0010 0000	int b = 2; Current Done!
0010	100 1001 0000 0000	int c;
0011	001 0111 1000 1001	c = a + b;
0100	100 0111 0001 0000	…
…	….	…

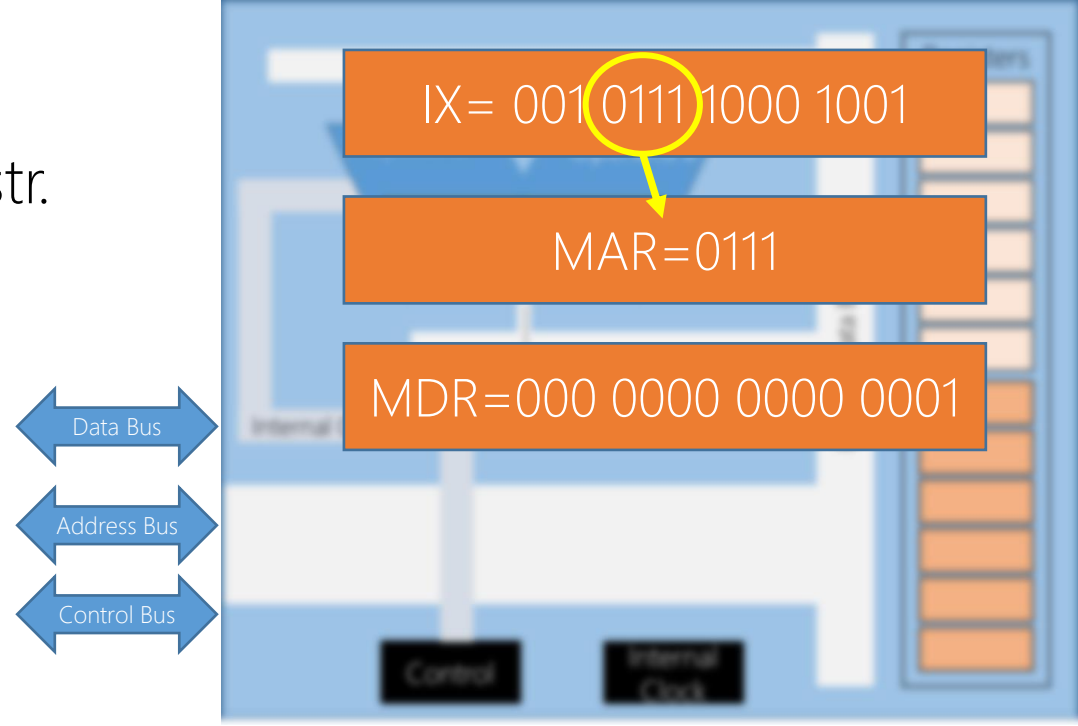


What are the steps to execute this instruction?

Address	Memory
0011	001 0111 1000 1001
0100	
0101	
0110	
0111	000 0000 0000 0001

High Level Instr.

c = a + b;

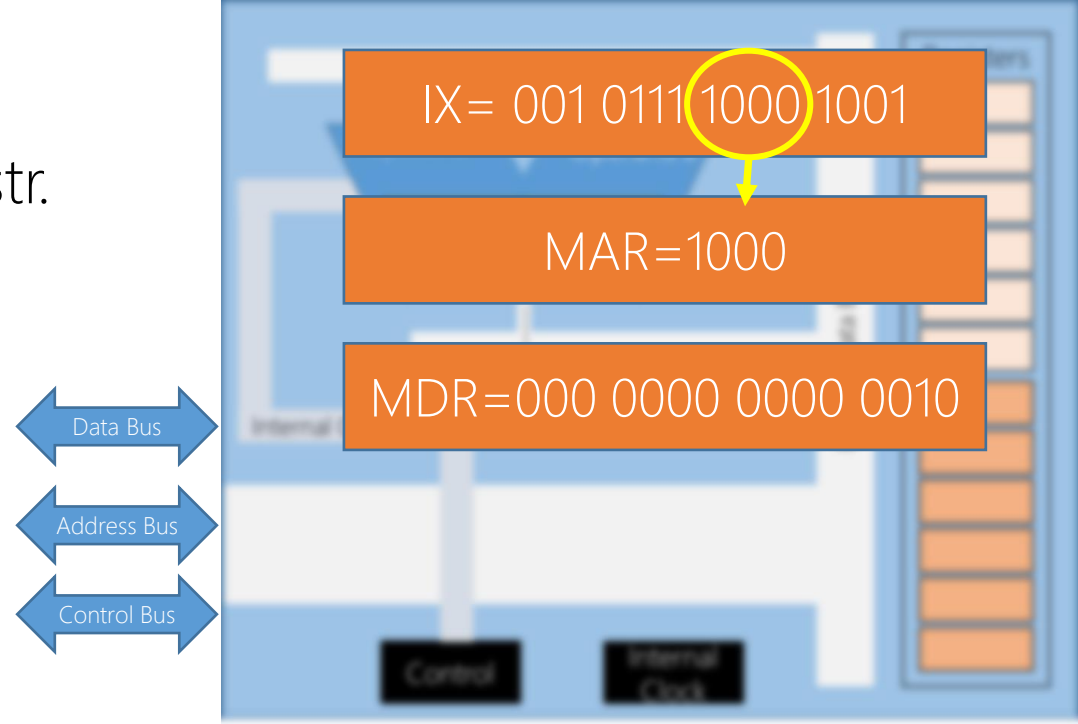


1) Fetch the first operand from memory
   I. Load MAR with the address of ‘a’ from the instruction op code
   II. Load address bus with MAR
   III. Load control bus with C=0 (Read)
   IV. Load MDR with the content of data bus

Address	Memory
0011	001 0111 1000 1001
0100	
0101	
0110	
0111	000 0000 0000 0001
1000	000 0000 0000 0010

High Level Instr.

c = a + b;



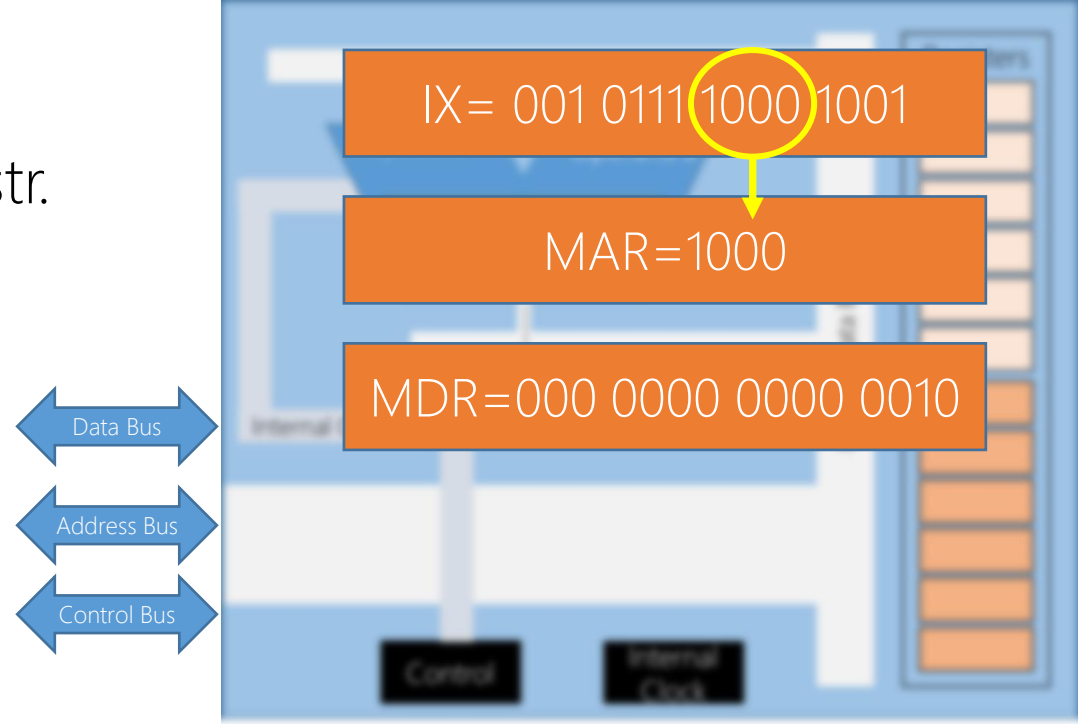
2) Fetch the second operand from memory

I. Load MAR with the address of 'b' from the instruction op code
II. Load address bus with MAR
III. Load control bus with C=0 (Read)
IV. Load MDR with the content of data bus

Address	Memory
0011	001 0111 1000 1001
0100	
0101	
0110	
0111	000 0000 0000 0001
1000	000 0000 0000 0010

High Level Instr.

c = a + b;



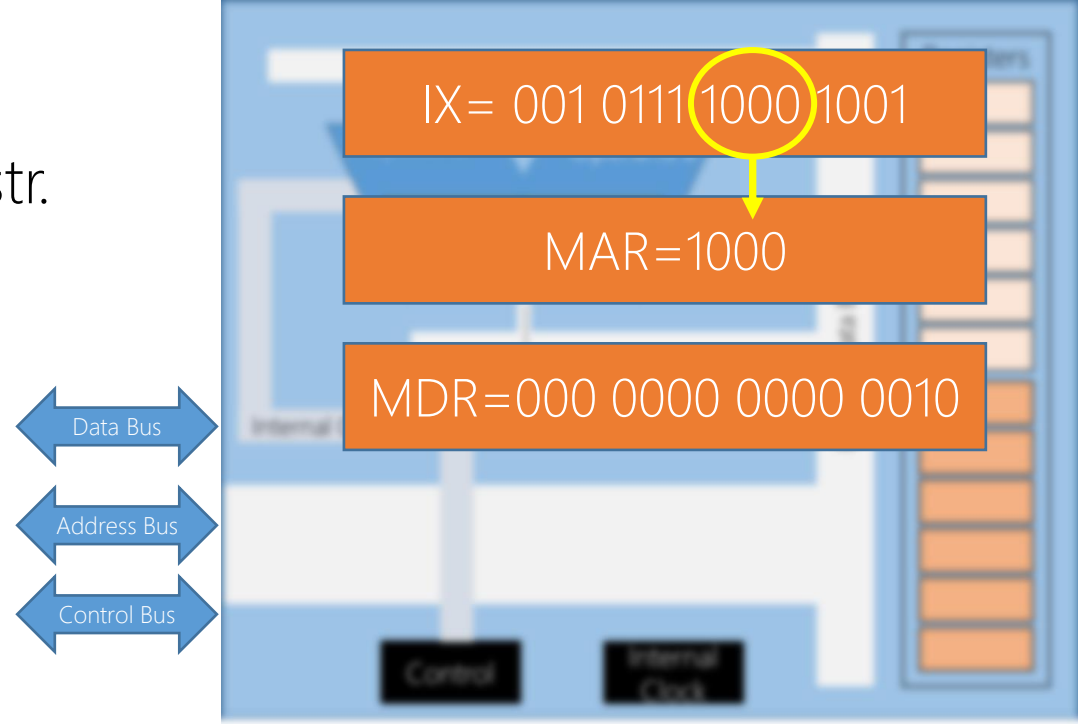
2) Fetch the second operand from memory

I. Load MAR with the address of 'b' from the instruction op code
II. Load address bus with MAR
III. Load control bus with C=0 (Read)
IV. Load MDR with the content of data bus. Wait! I lost the value of 'a' inside the processor!

Address	Memory
0011	001 0111 1000 1001
0100	
0101	
0110	
0111	000 0000 0000 0001
1000	000 0000 0000 0010

High Level Instr.

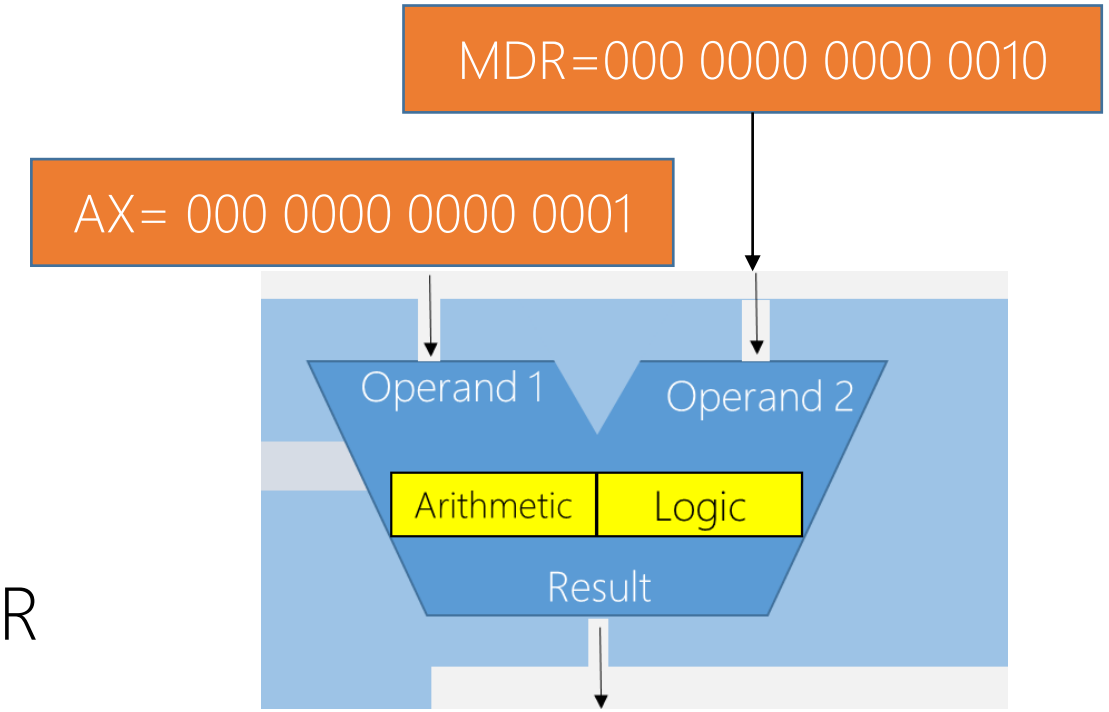
c = a + b;



2) Fetch the second operand from memory

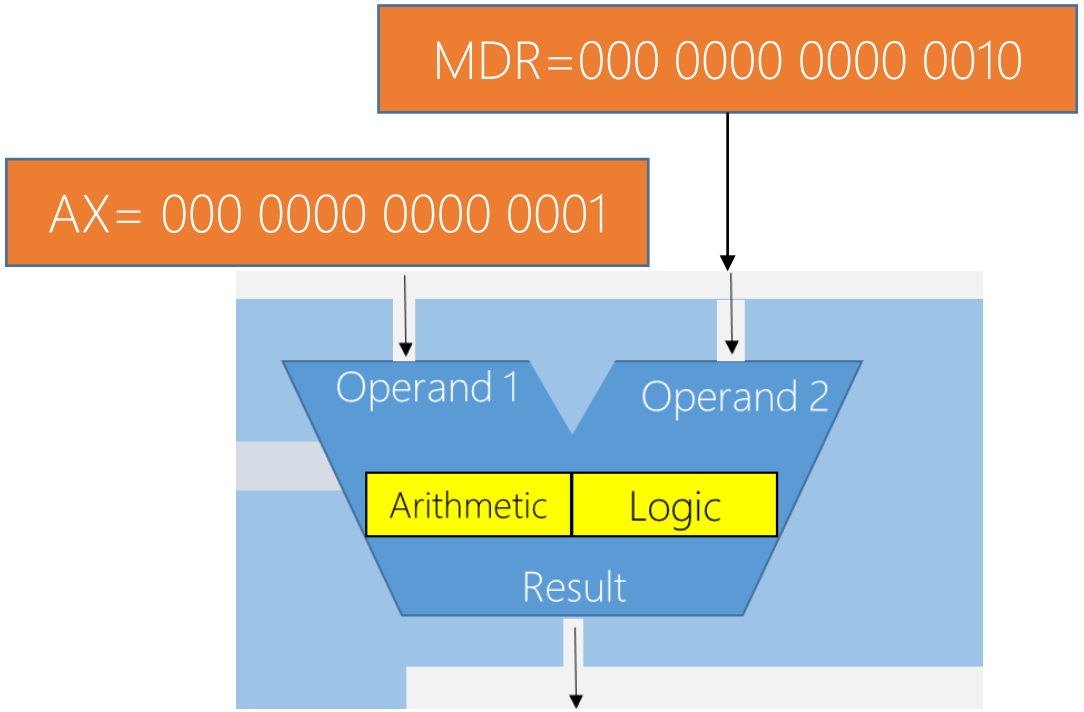
I. Move MDR (value of 'a') to somewhere (AX) first!
II. Load MAR with the address of 'b' from the instruction op code
III. Load address bus with MAR
IV. Load control bus with C=0 (Read)
V. Load MDR with the content of data bus.

Address	Memory	High Level Instr.
0011	001 0111 1000 1001	c = a + b;



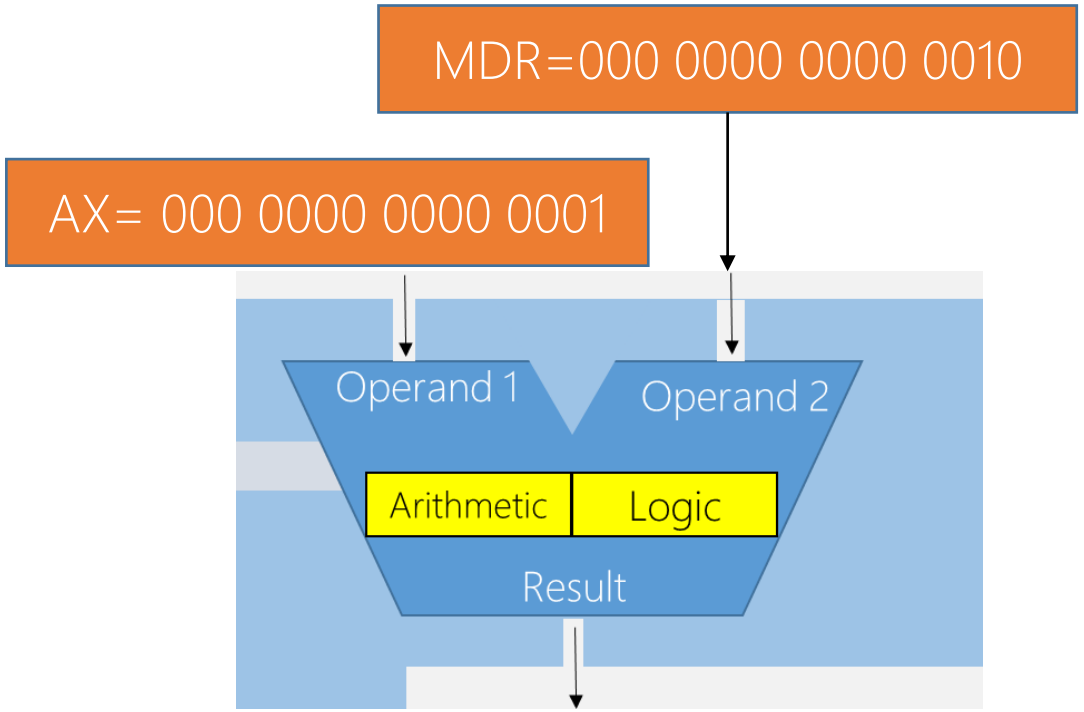
3) Use the n-bit Adder to add AX and MDR

Address	Memory	High Level Instr.
0011	001 0111 1000 1001	c = a + b;



4) Store the result somewhere. Where?

Address	Memory	High Level Instr.
0011	001 0111 1000 1001	c = a + b;

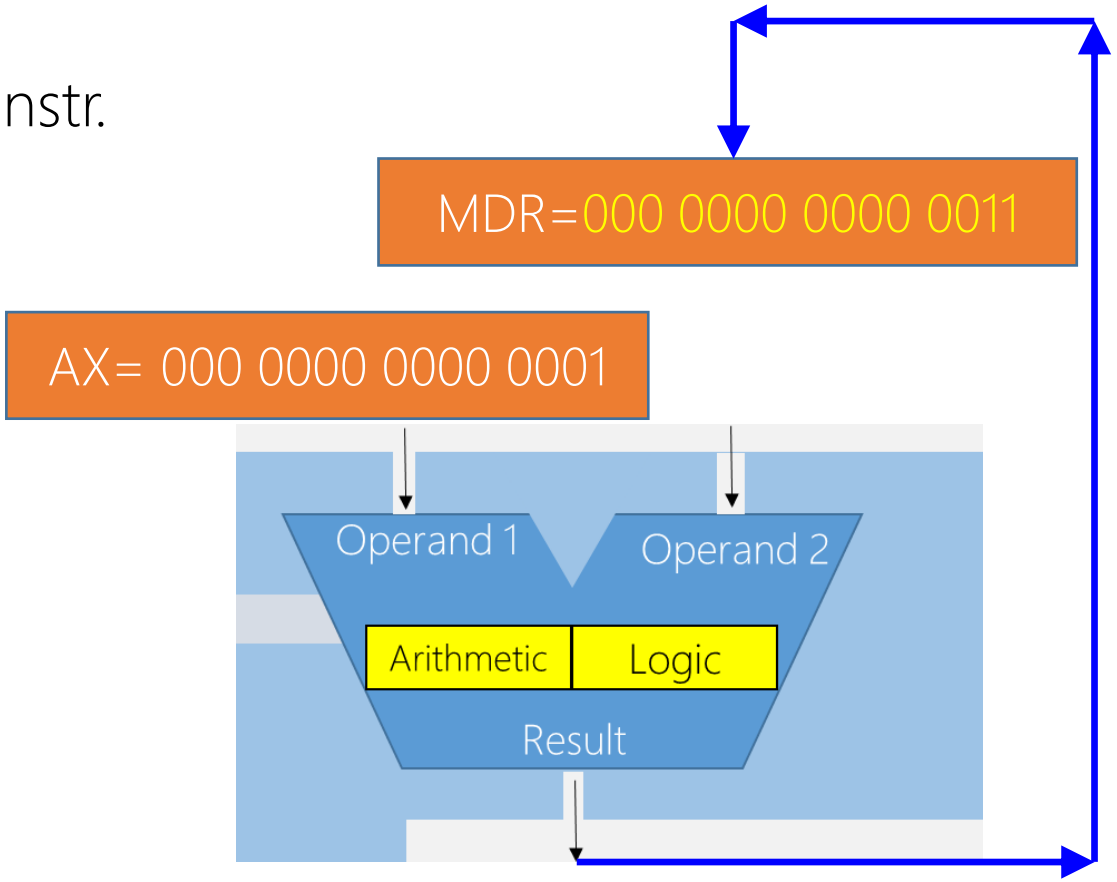


4) Store the result somewhere. Where?

We want to write it back to memory at location for ‘c’

So, better to store it in MDR

Address	Memory	High Level Instr.
0011	001 0111 1000 1001	c = a + b;

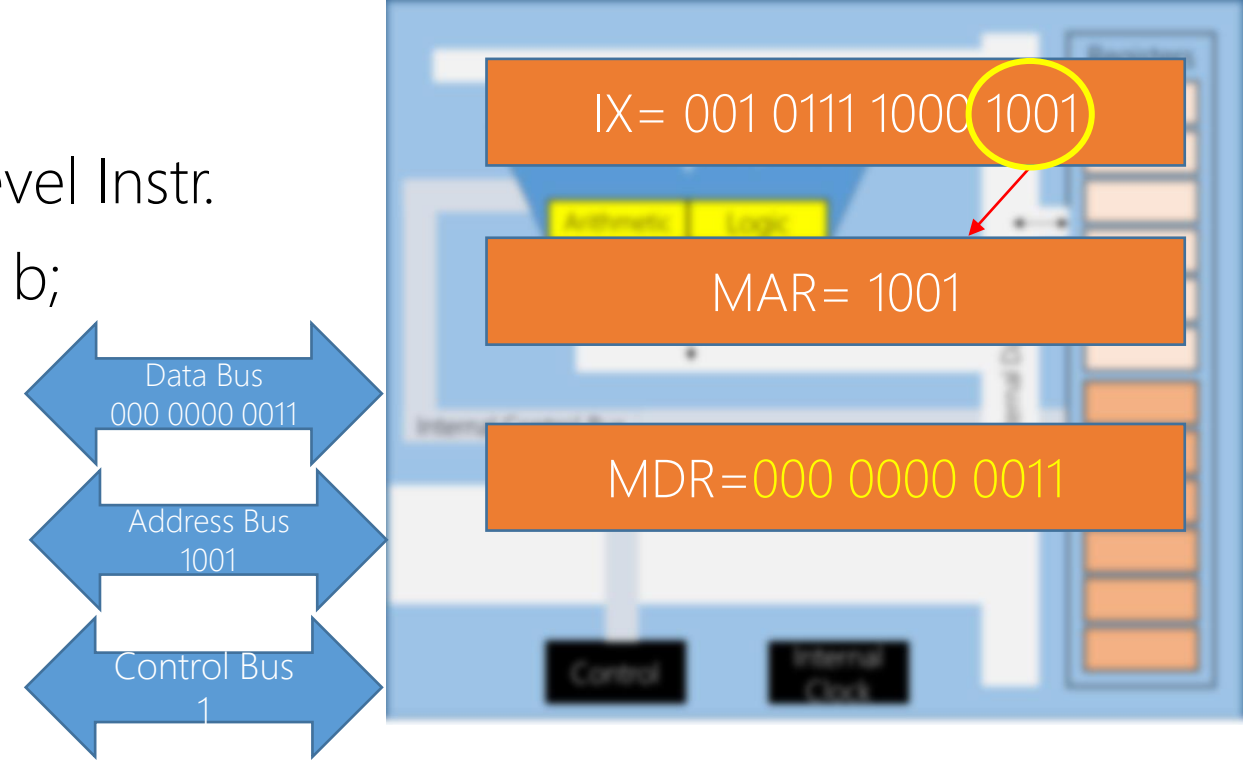


4) Store the result somewhere. Where?
What do we want to do next?
We want to write it back to memory at location for ‘c’
So, better to store it in MDR

Address	Memory
0011	001 0111 1000 1001
	…
1001	000 0000 0000 0011

High Level Instr.

c = a + b;



5) Write the result back to memory at location for ‘c’.

I. Find the address of ‘c’ in op code
II. Load it into MAR
III. Load address bus with MAR
IV. Load data bus with MDR
V. Load control bus with C=1 (Write)

Accumulator (AX)

This is the most frequently used register.
Any intermediate results
somewhere → AX

General Purpose Registers

$R_1, R_2, \ldots, R_i$

eclipse-workspace - COMP2650_W13_hfani/src/COMP2650_W13_hfani.c - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Debug | COMP2650_W13_hfani.exe

Debug | Project Explorer

- COMP2650_W13_hfani.exe [C/C++ Application]
  - COMP2650_W13_hfani.exe [19356]
    - Thread #1 [COMP2650_W13_hfani] 0 (Suspende
      - main() at COMP2650_W13_hfani.c:6 0x10040
    - Thread #2 0 (Suspended : Container)
    - Thread #3 0 (Suspended : Container)
    - Thread #4 [sig] 0 (Suspended : Container)
  - gdb (8.3.1)

COMP2650_W13_hfani.c | main() at COMP2650_W13_hfani.c:6 0x10040108d

```
1  int main(void) {
2  //  int a=1;
3  //  int b=2;
4      int c;
5  //  c = a + b;
6      c = 1 + 2;
7      return 0;
8  }
9
```

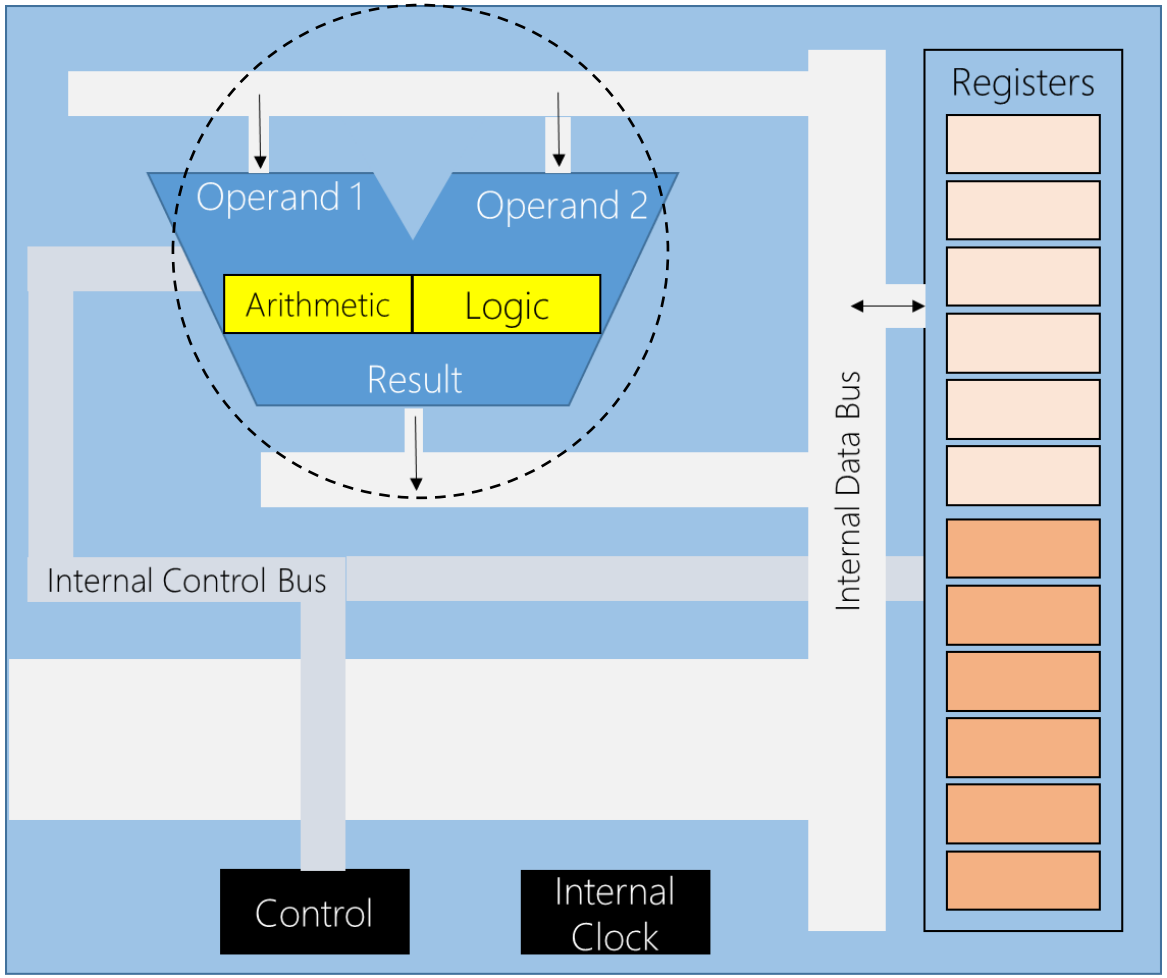
Variables | Breakpoints | Expressions | Modules | Disassembly

Enter location here

```
0000000100401088:   callq   0x1004010b0 <__main>
000000010040108d:   movl    $0x3,-0x4(%rbp)
0000000100401094:   mov     $0x0,%eax
0000000100401099:   add     $0x30,%rsp
000000010040109d:   pop     %rbp
000000010040109e:   retq
000000010040109f:   nop
                  __cxa_atexit:
00000001004010a0:   jmpq    *0x700e(%rip)        #
00000001004010a6:   nop
00000001004010a7:   nop
00000001004010a8:   nop
00000001004010a9:   nop
```

Console | Registers | Problems | Executables | Debugger Console | Memory

Name	Value	Description
General Registers		General Purpose and FPU Register Group
rax	0	
rbx	6444787584	
rcx	1	
rdx	0	
rsi	0	
rdi	32	
rbp	0xffffcbe0	
rsp	0xffffcbb0	
r8	0	
r9	0	
r10	4294953584	
r11	6442750706	
r12	4294954032	
r13	4294954048	
r14	4294954048	
r15	0	
rip	0x10040108d <main+13>	
eflags	[PF IF]	
cs	51	


Operand 1
Operand 2
Arithmetic
Logic
Result
Registers
Internal Data Bus
Internal Control Bus
Control
Internal Clock

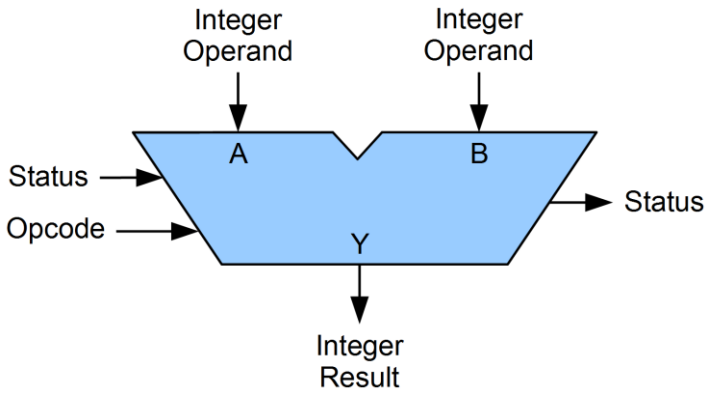
Arithmetic & Logic Unit (ALU)

Arithmetic Unit

Integer Arithmetic (Addition, Subtraction)

- Multiplication: multiple addition
- Division: multiple subtraction

Floating Point Arithmetic (FPU)

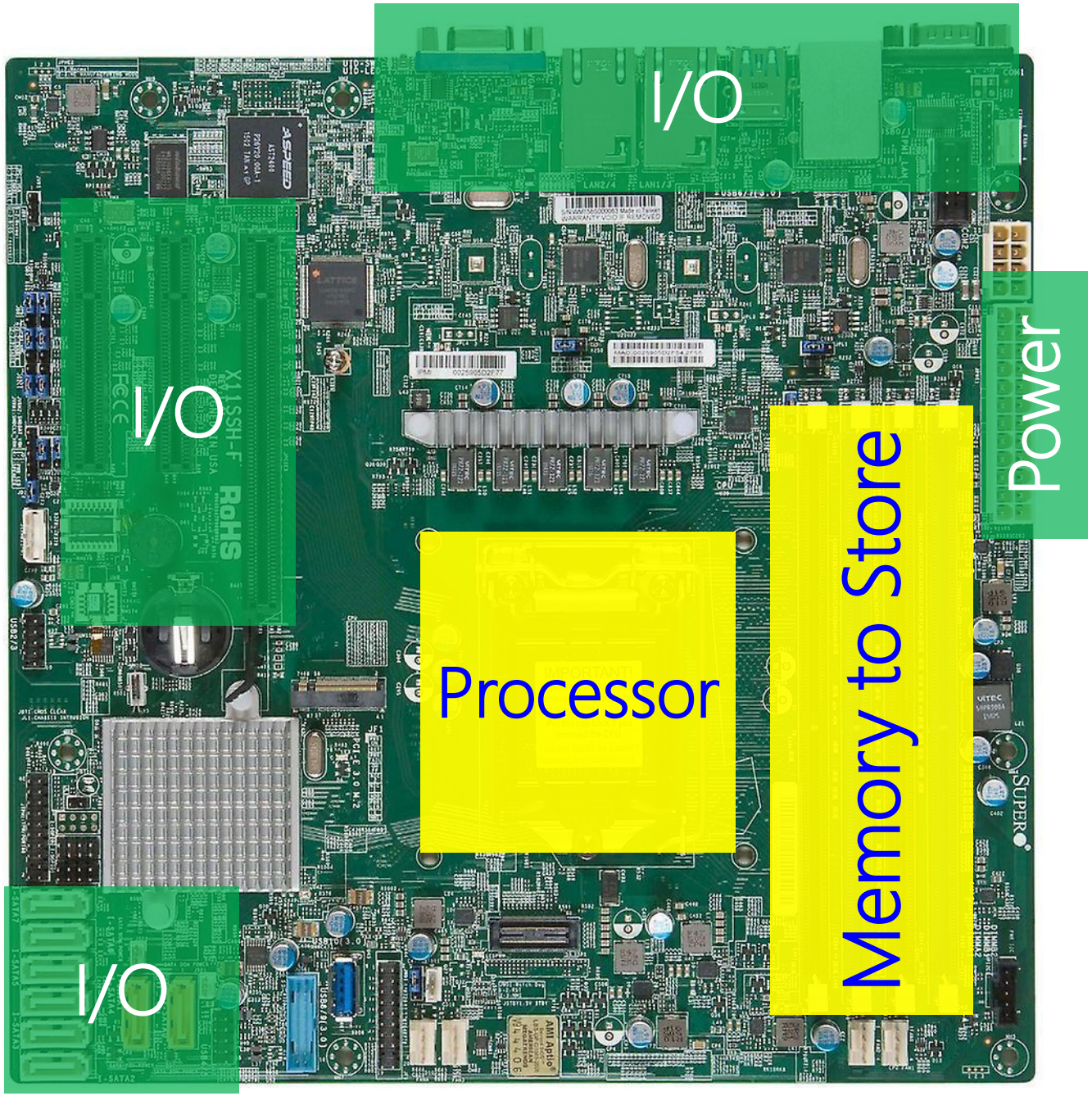


Logic Unit

Bitwise
AND, OR, XOR, 1's Comp.

Recap

1) Fetch Instruction Cycle
2) Execution Cycle
   - Depends on OP Code
     Arithmetic
     Logic


I/O
I/O
Power
Processor
Memory to Store
I/O

Microprocessor

Central Processing Unit (CPU)

Intel (Xeon), AMD (Ryzen)

Graphic Processing Unit (GPU)

nVidia (GeForce) for Video Card (Gaming)

AI Acceleration (Neural Nets)

Tensor Processing Unit (TPU)

Google (TensorFlow AI Lib for Neural Nets)

COMP-2660
Computer Architecture II
Microprocessor Programming

Postscript

Office Hours Before Week
- Practice on questions

Office Hours Final Exam
- Review exam questions