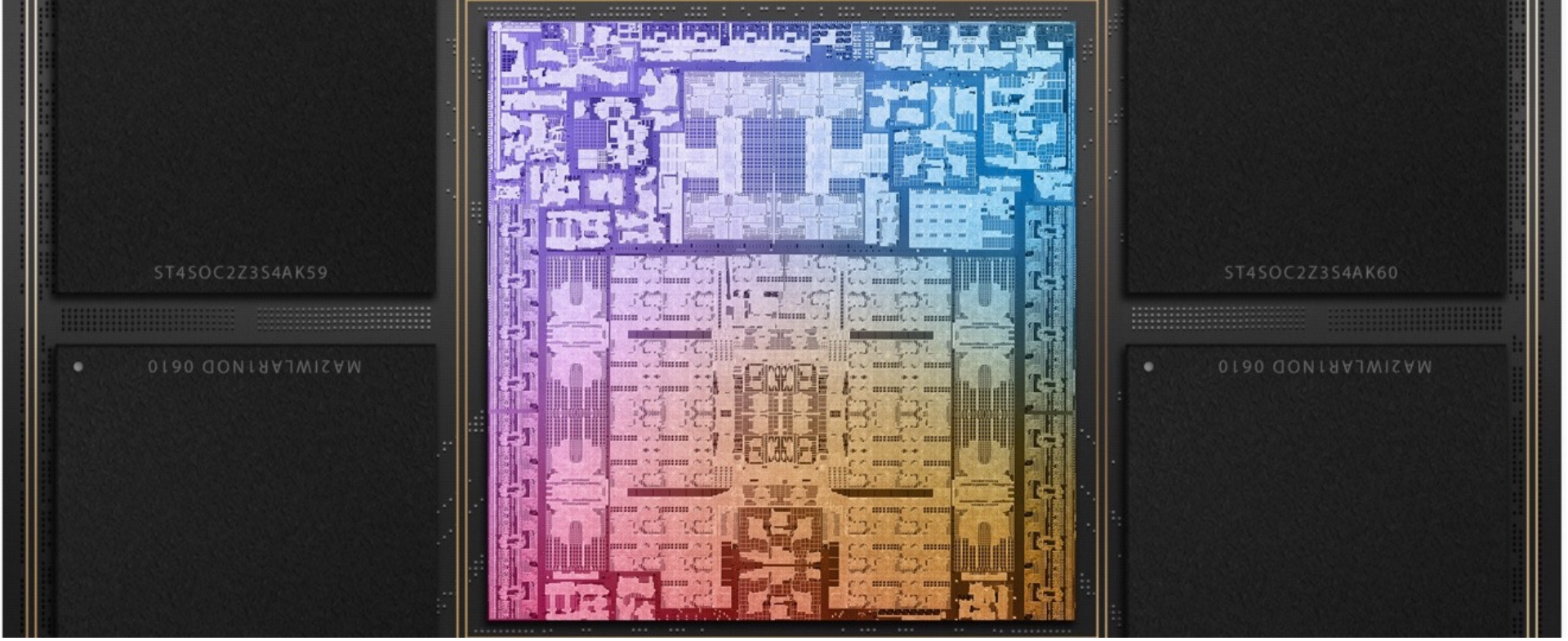


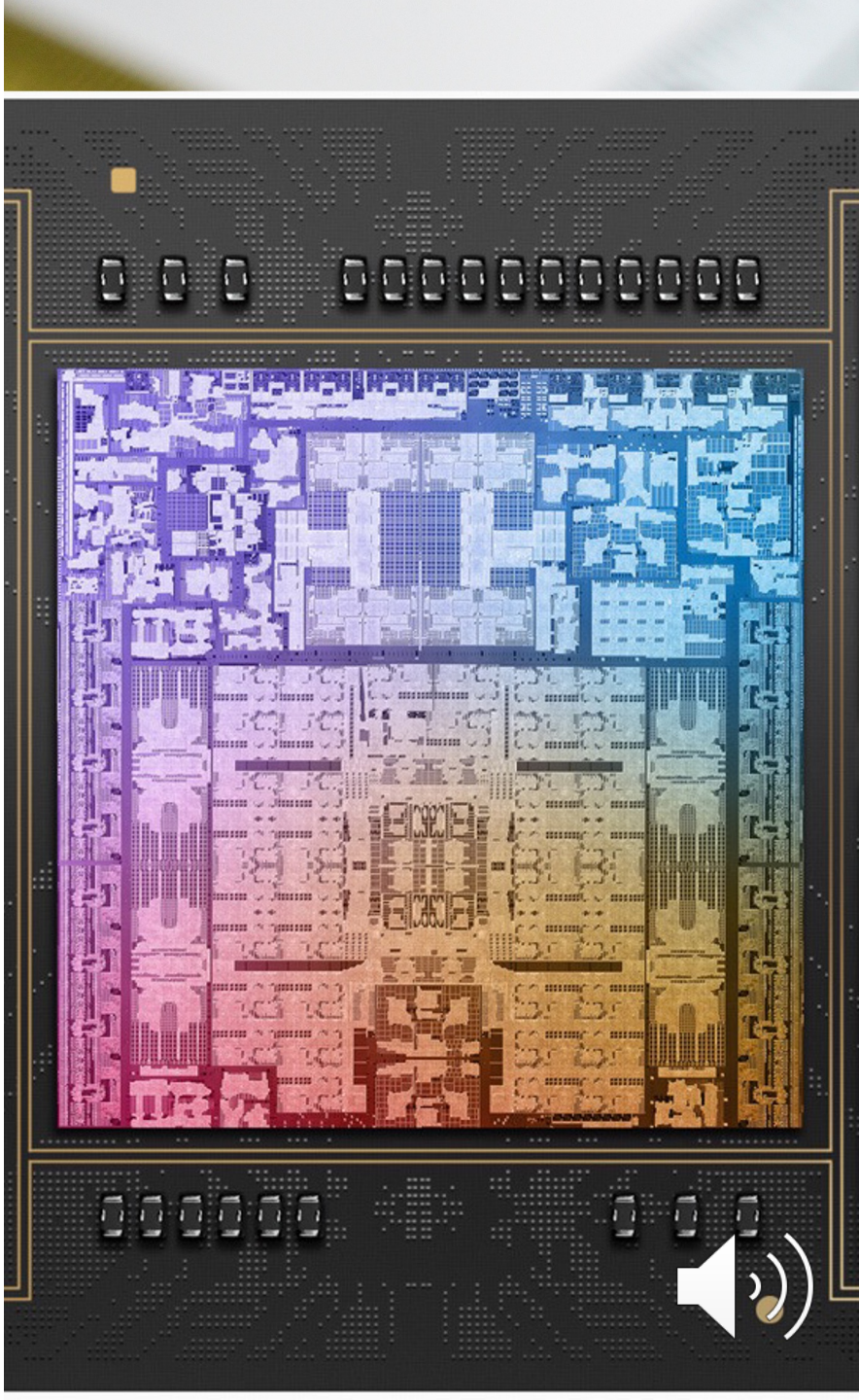
Central Processing Unit

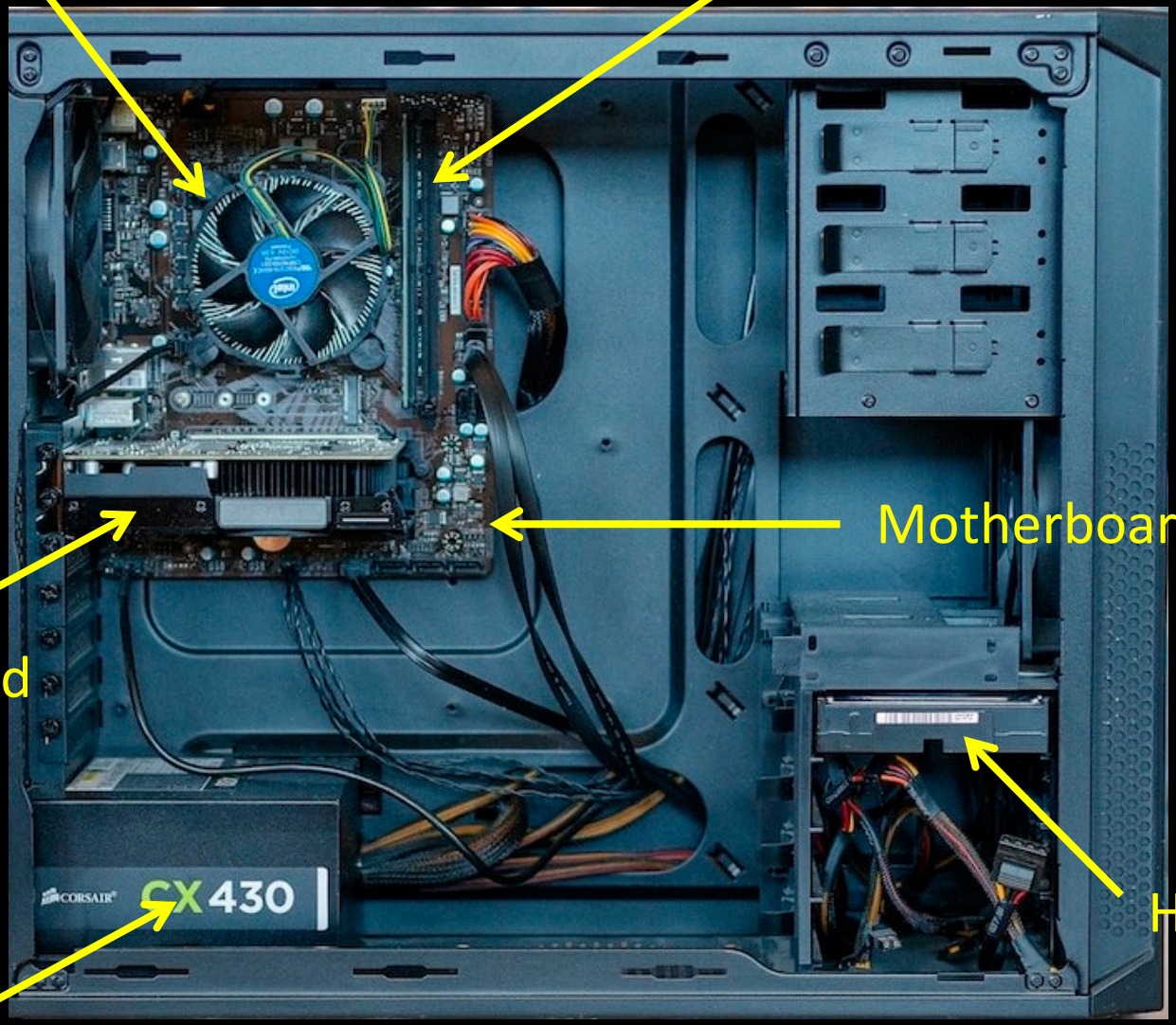
Asst. Prof. Natawut Nupairoj, Ph.D.
Department of Computer Engineering
Chulalongkorn University
natawut.n@chula.ac.th

CHULA ENGINEERING
Foundation toward Innovation

Central Processing Unit

Overview

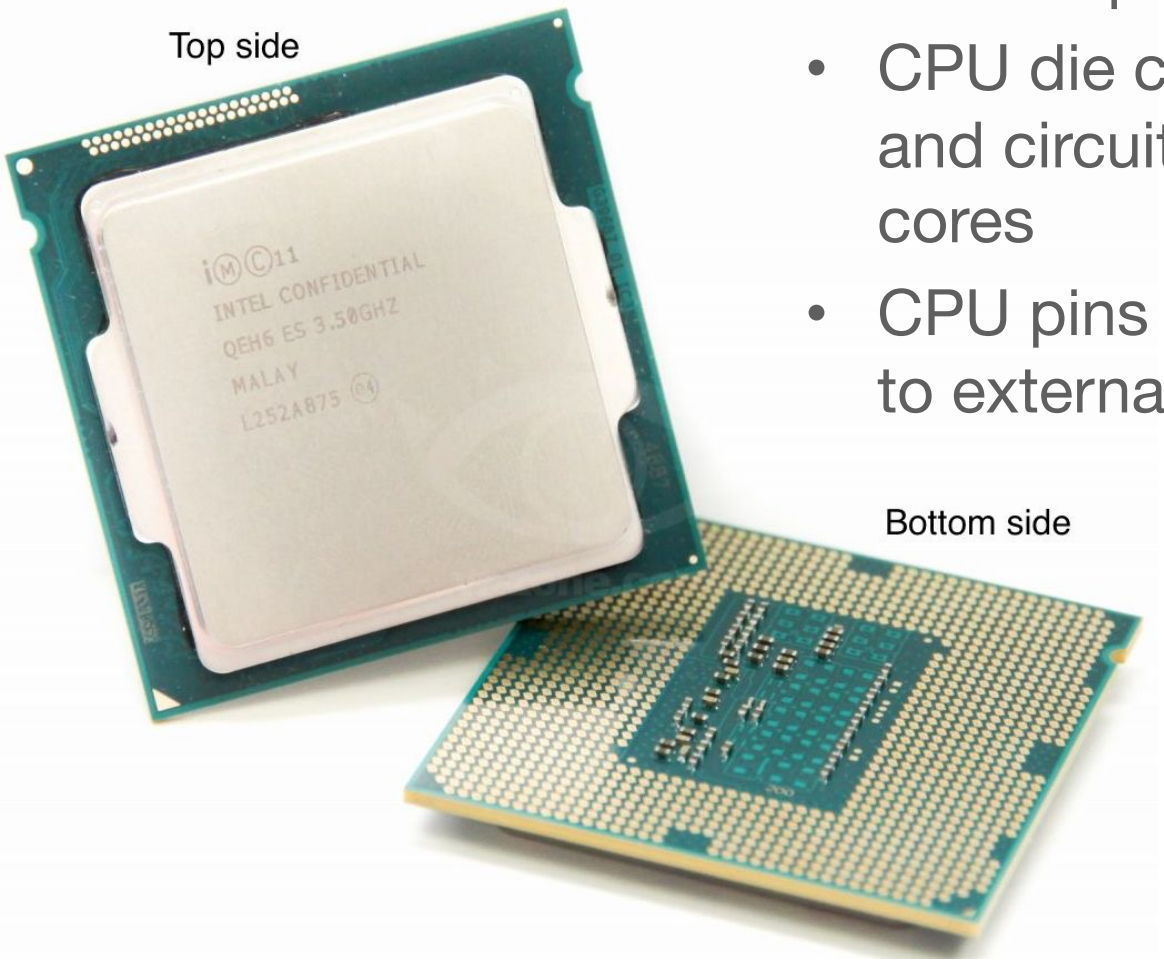




CPU = Central Processing Unit

- Consists of lots (billions) of transistors working synchronously in clock cycle (higher clock usually means faster CPU)
- Main purpose is to execute instructions (in low-level machine language) of computer program
- Some CPUs have multiple cores (processing units), some have special purpose cores (GPU, neural)

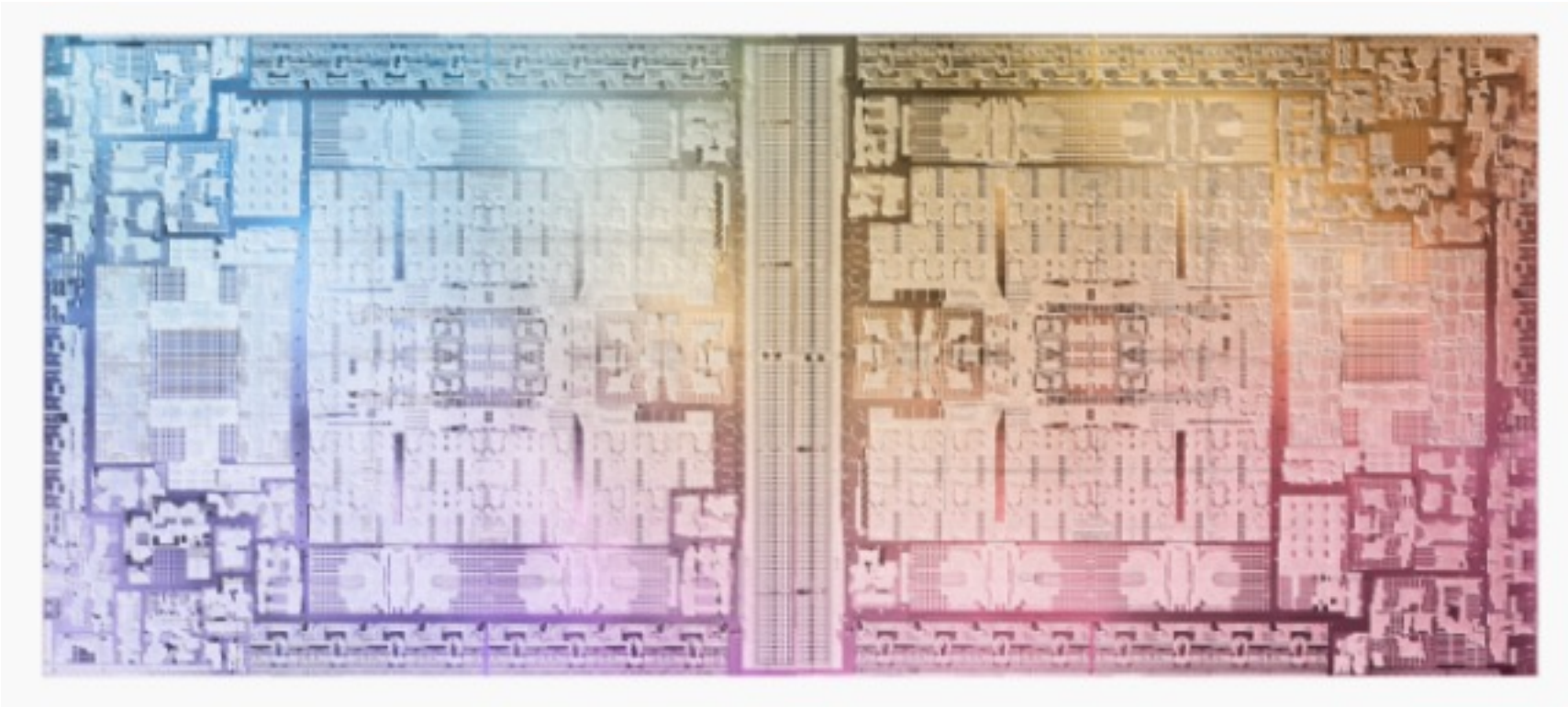
CPU Package



- CPU comes in package: CPU die and pins
- CPU die contains transistors and circuit of processor cores
- CPU pins connect CPU die to external components

CPU Die

Source: https://macdailynews.com/2023/06/05/apple-debuts-m2-ultra-with-up-to-192gb-of-memory/



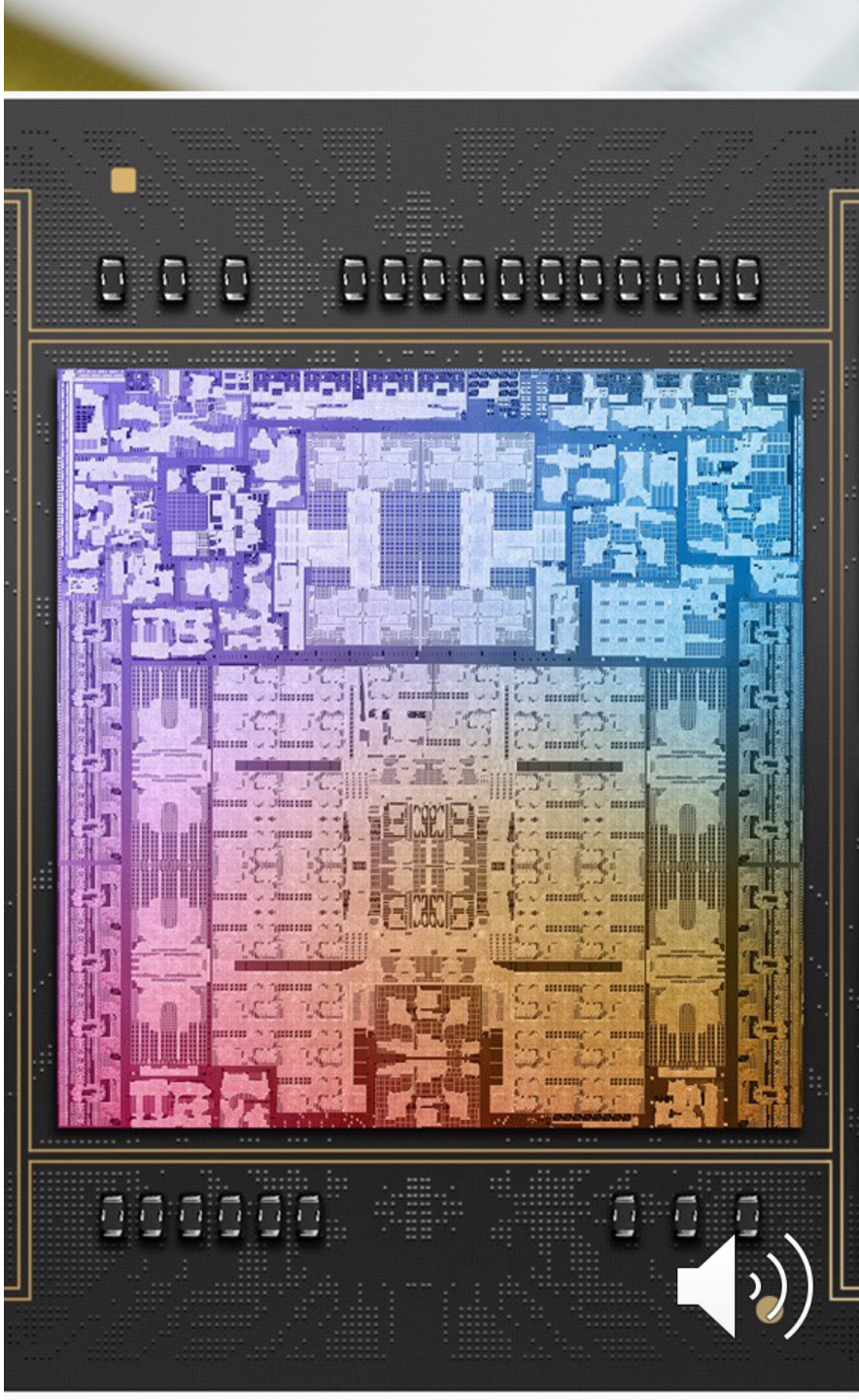
Apple M2 Ultra

- 134,000,000,000 transistors (5nm)
- 24 CPU cores / 32 neural cores / 76 GPU cores

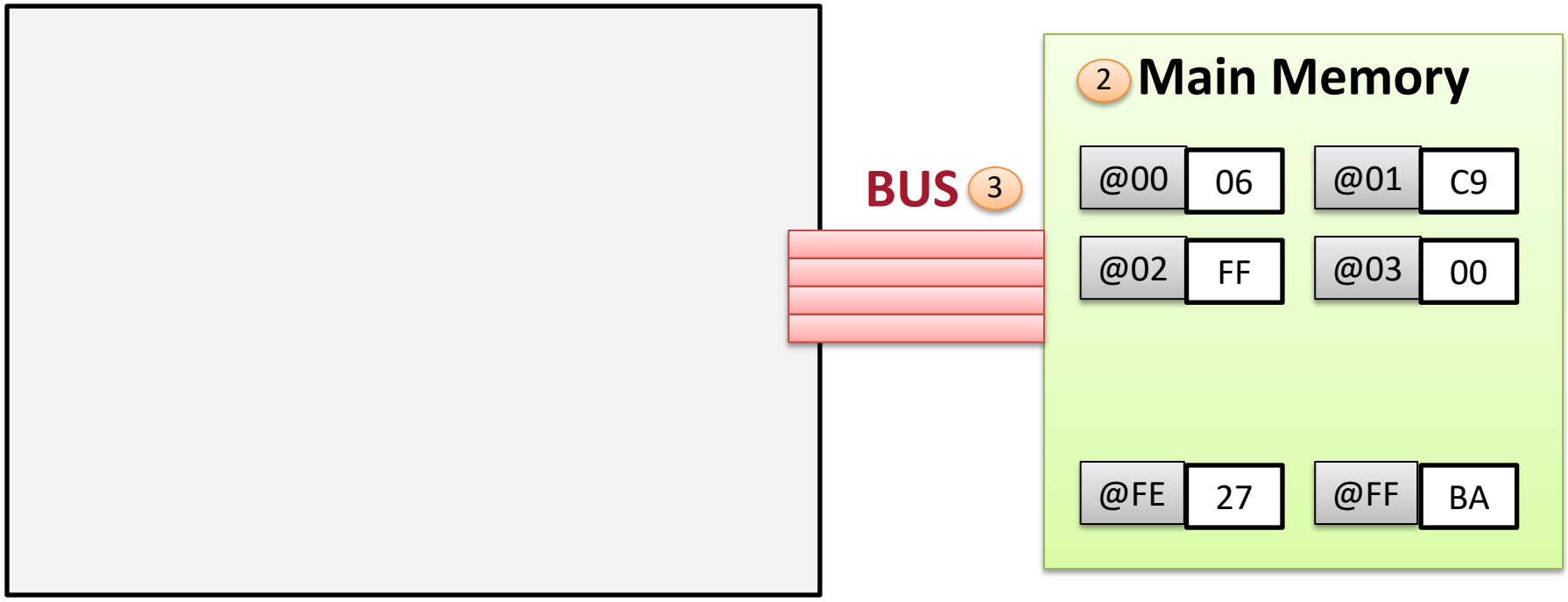
CPU	Year	Max Clock Speed	Cores	Transistor Counts	Max Memory
Intel 4004	1971	750 kHz	1 CPU	2.3 K	4 KB
Intel 8086	1978	10 MHz	1 CPU	29 K	1 MB
AMD Zen 2 AMD RDNA 2 (Sony PS5)	July 2019	3.5 GHz	8 CPU 36 GPU	3.8 B	64 GB
Nvidia GA102 (Ampere)	Sep 2020	1.7 GHz	10496 GPU	28 B	48 GB
Apple S8 (Apple Watch Ultra)	Sep 2022	1.8 GHz	2 CPU 1 GPU	N/A	1 GB
Intel Core i9 13900KS	Sep 2022	6.0 GHz (Turbo Mode)	24 CPU	14.2 B	128 GB
Apple M2 Ultra	June 2023	3.5 GHz	24 CPU 32 Neural 76 GPU	134 B	192 GB

Central Processing Unit

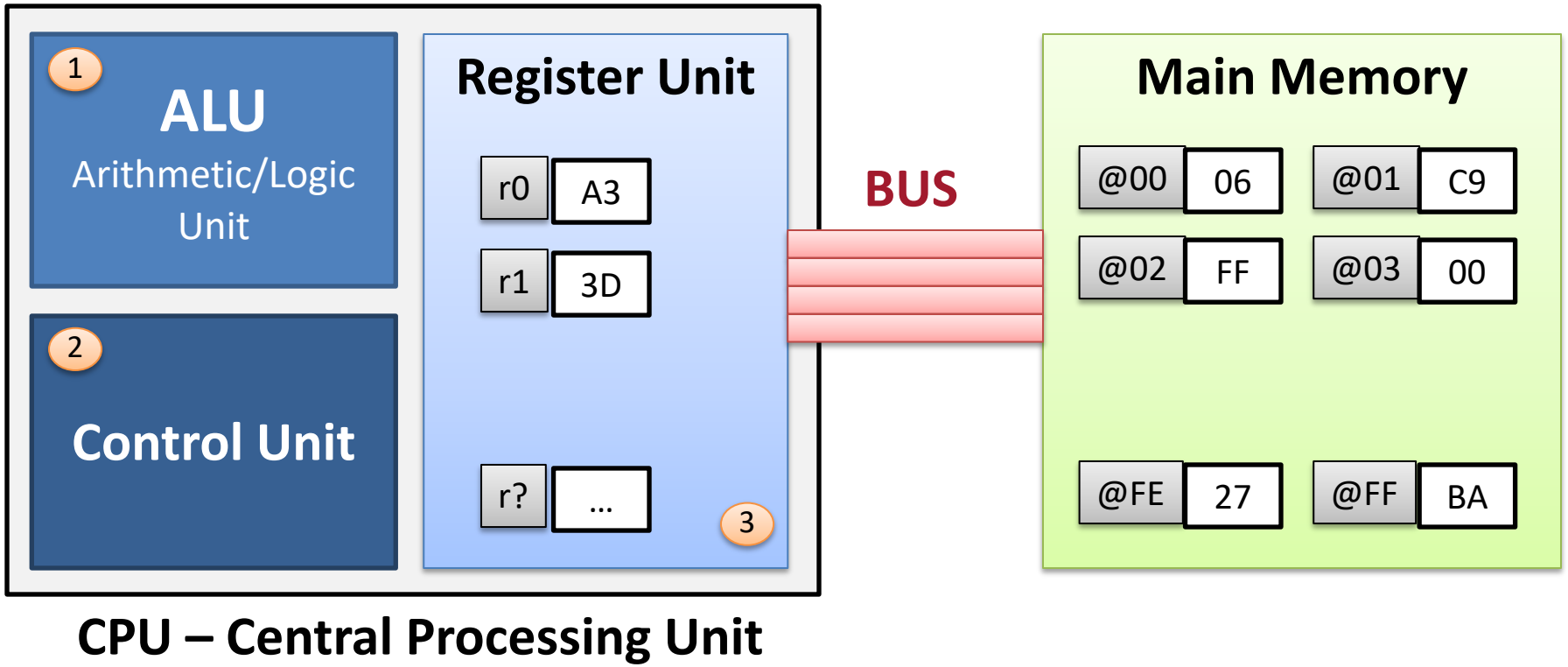
Basic CPU Components



Basic CPU Components



Basic CPU Components

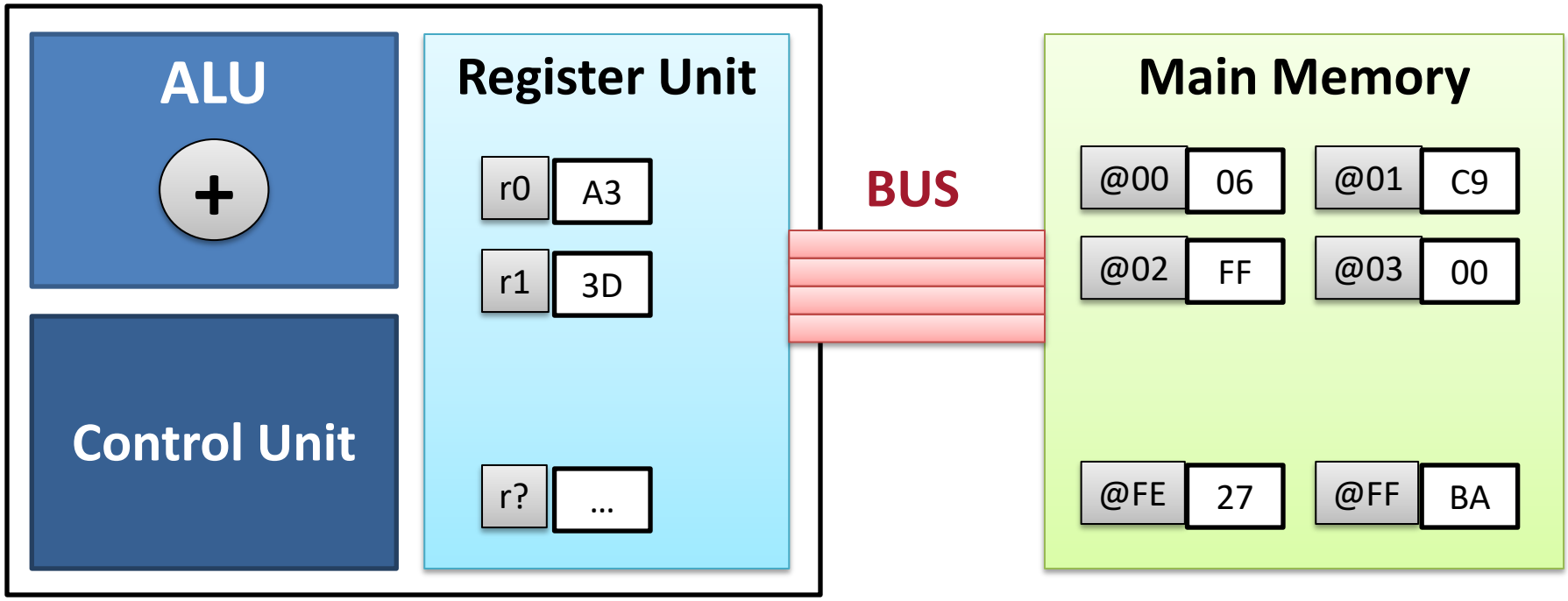


CPU – Central Processing Unit

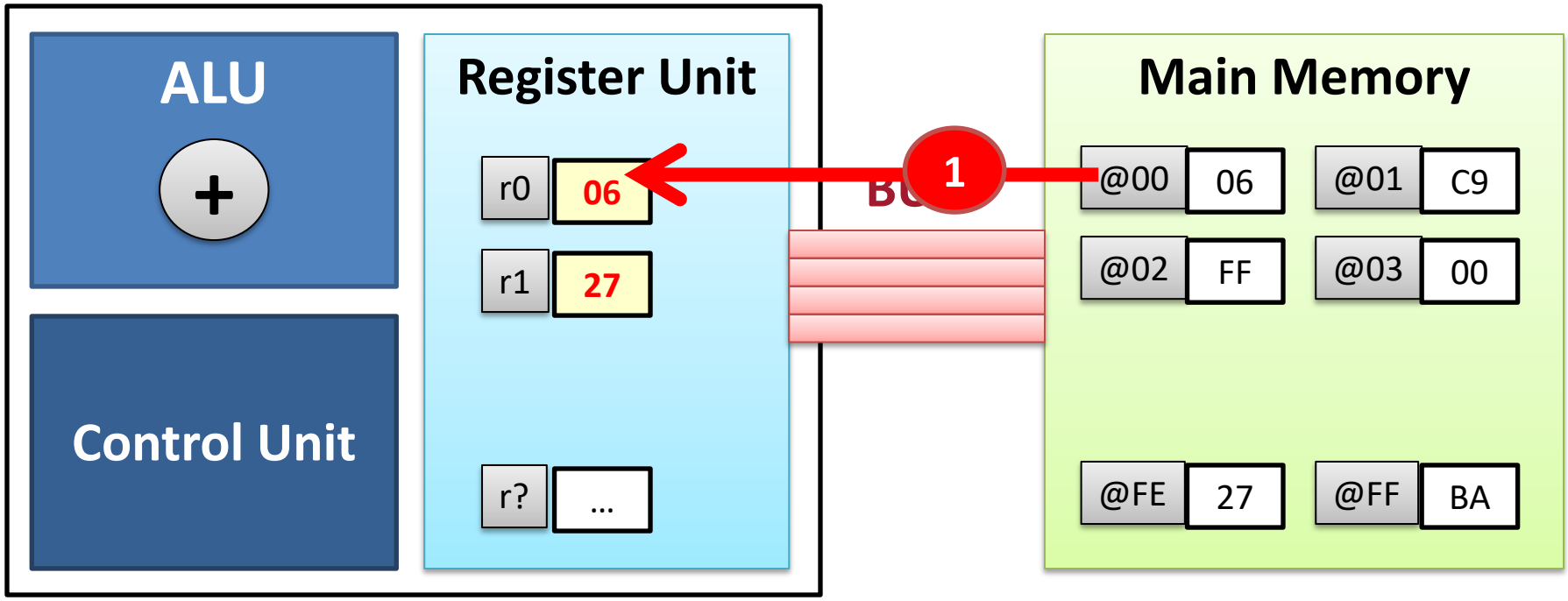
Typical CPU Behaviors

- All calculations in ALU must be done using values from registers; we cannot calculate using data in memory directly
- Thus, we usually load data from memory to registers, calculate placing result in a register, and store results back to memory
- This is called “Load-Store Architecture”

Example: Adding 2 Numbers $@03 \leftarrow @00 + @FE$

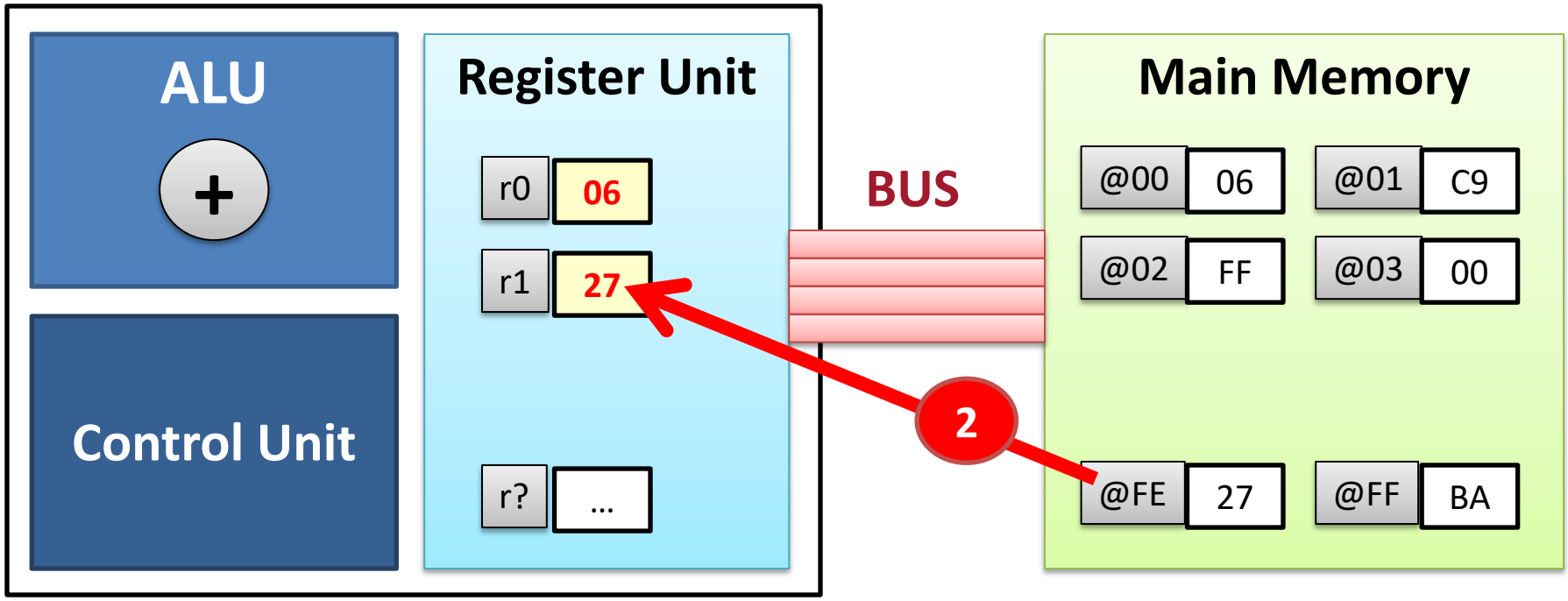


Example: Adding 2 Numbers $@03 \leftarrow @00 + @FE$



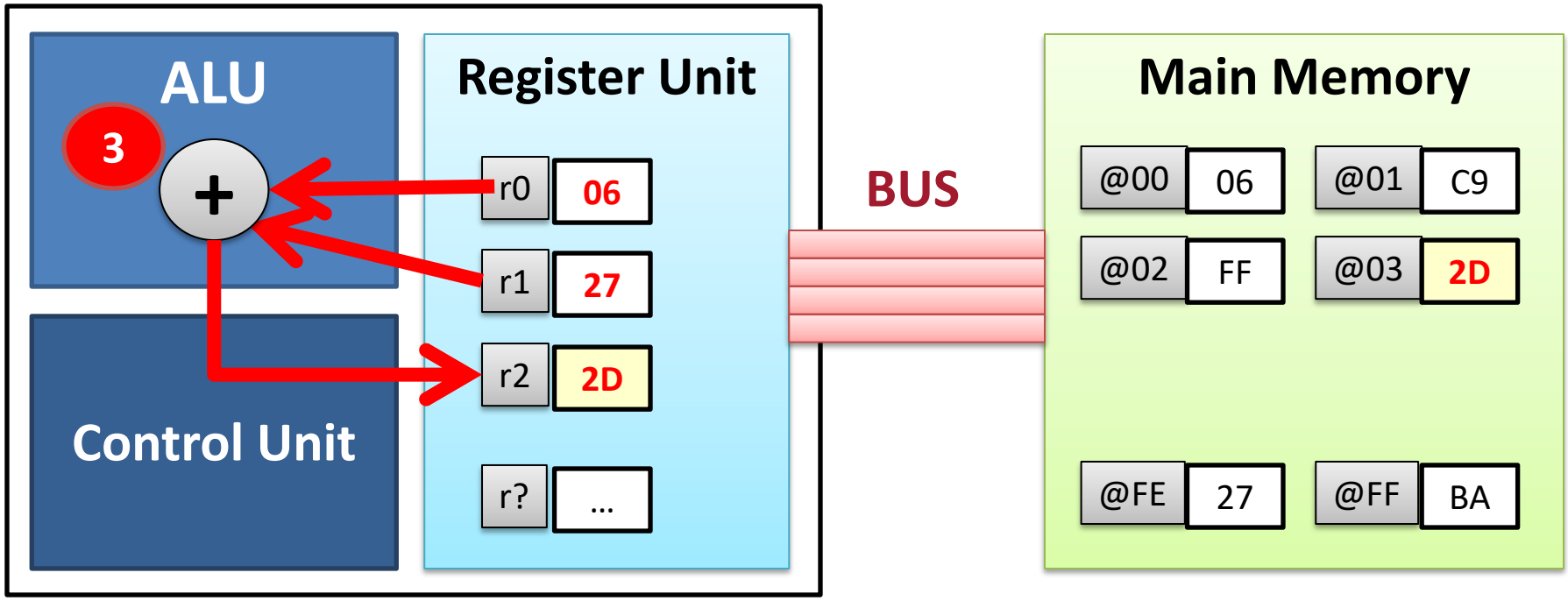
Step 1: Get one of the values to be added from memory and place it in a register
Step 2: Get the other value to be added from memory and place it in another register

Example: Adding 2 Numbers $@03 \leftarrow @00 + @FE$



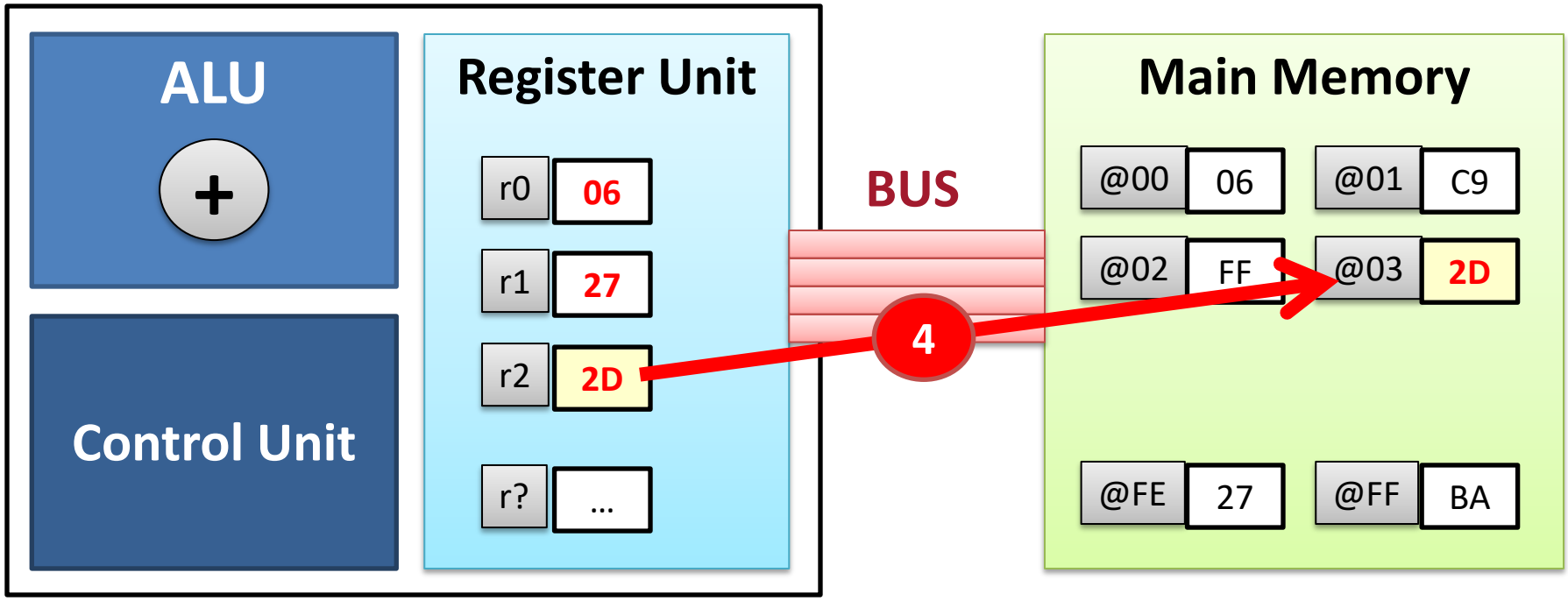
Step 2: Get the other value to be added from memory and place it in another register

Example: Adding 2 Numbers $@03 \leftarrow @00 + @FE$



Step 3: Activate the addition circuitry with the registers used in Steps 1 and 2 as inputs and another register designated to hold the result

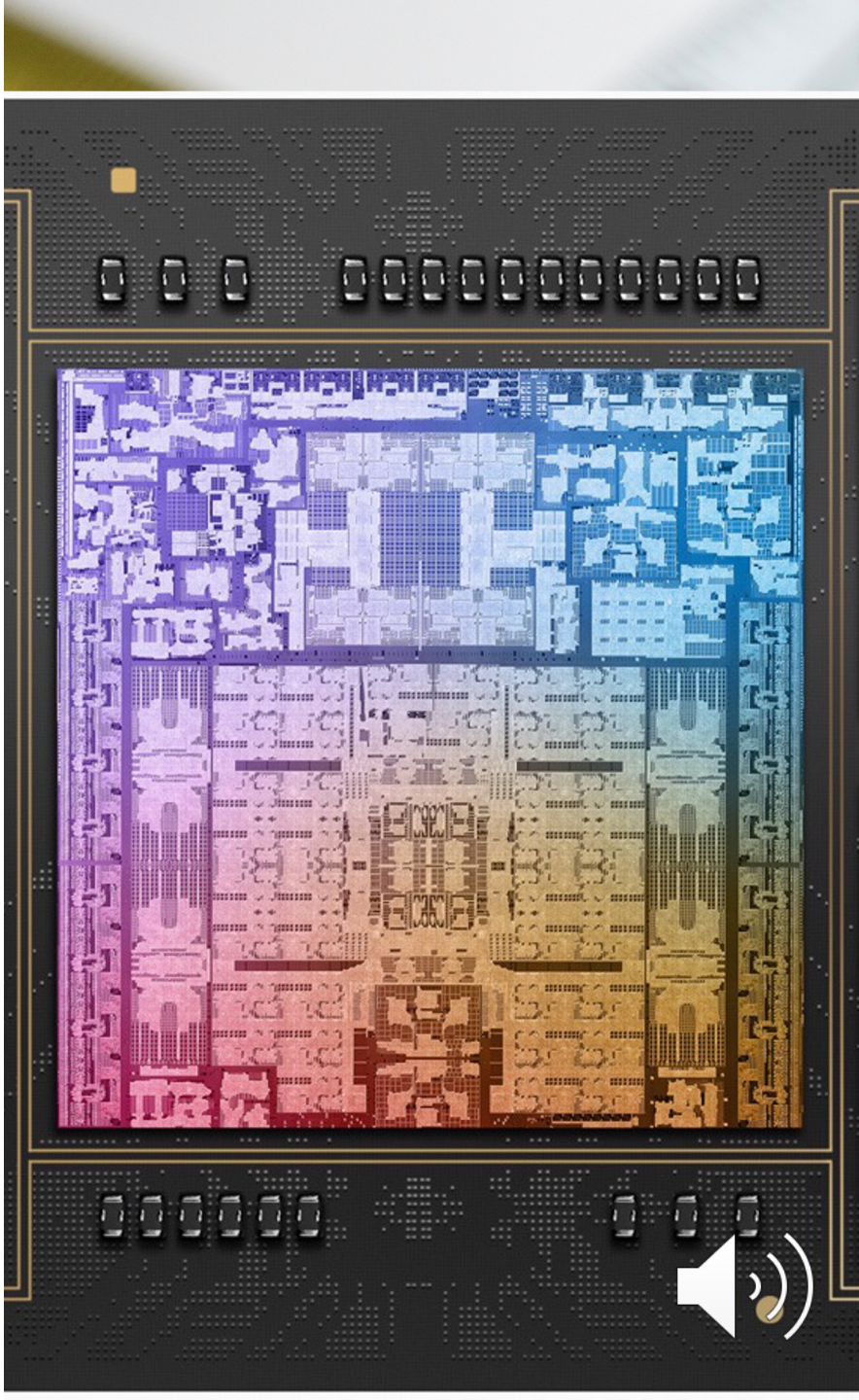
Example: Adding 2 Numbers $@03 \leftarrow @00 + @FE$



Step 4: Store the result in memory

Central Processing Unit

CPU Execution

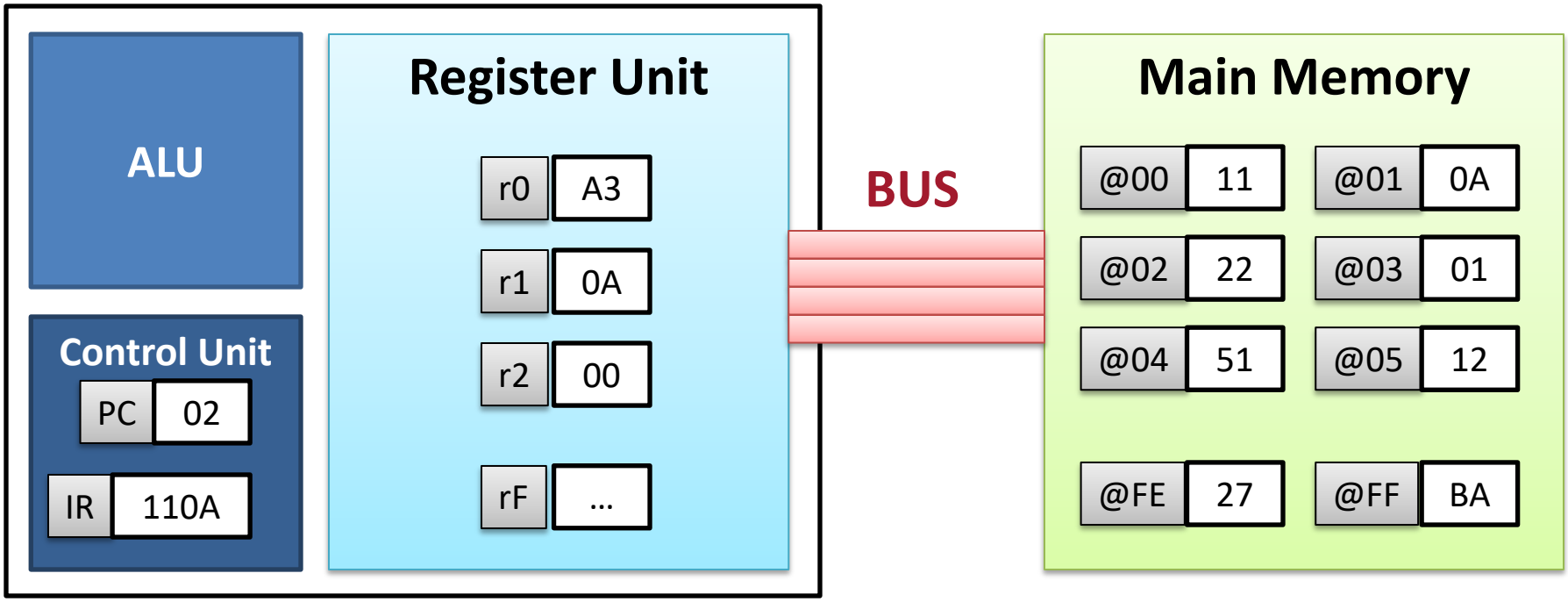


CPU Instruction Set

- To make a CPU runs, we must create a program using CPU instruction set or machine language
- Machine language is a low-level programming language, which can be executed directly by CPU
- We usually write a program in “high-level programming language” e.g. Python, Java, C, C++, Rust, Go, etc.
- Thus, all programs in high-level languages must be “translated” (compiled or interpreted) into machine language

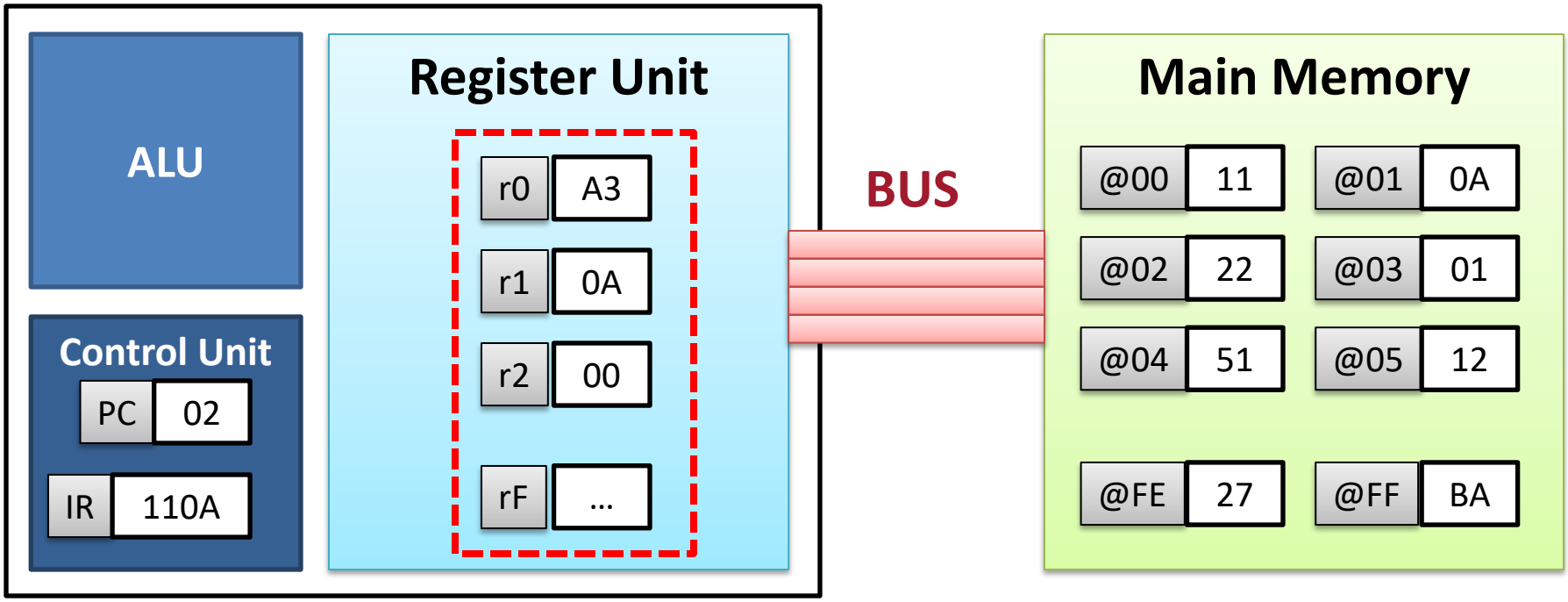
https://www.amazon.com/Computer-Science-Overview-Glenn-Brookshear/dp/0133760065

Brookshere's Simple Machine



https://www.amazon.com/Computer-Science-Overview-Glenn-Brookshear/dp/0133760065

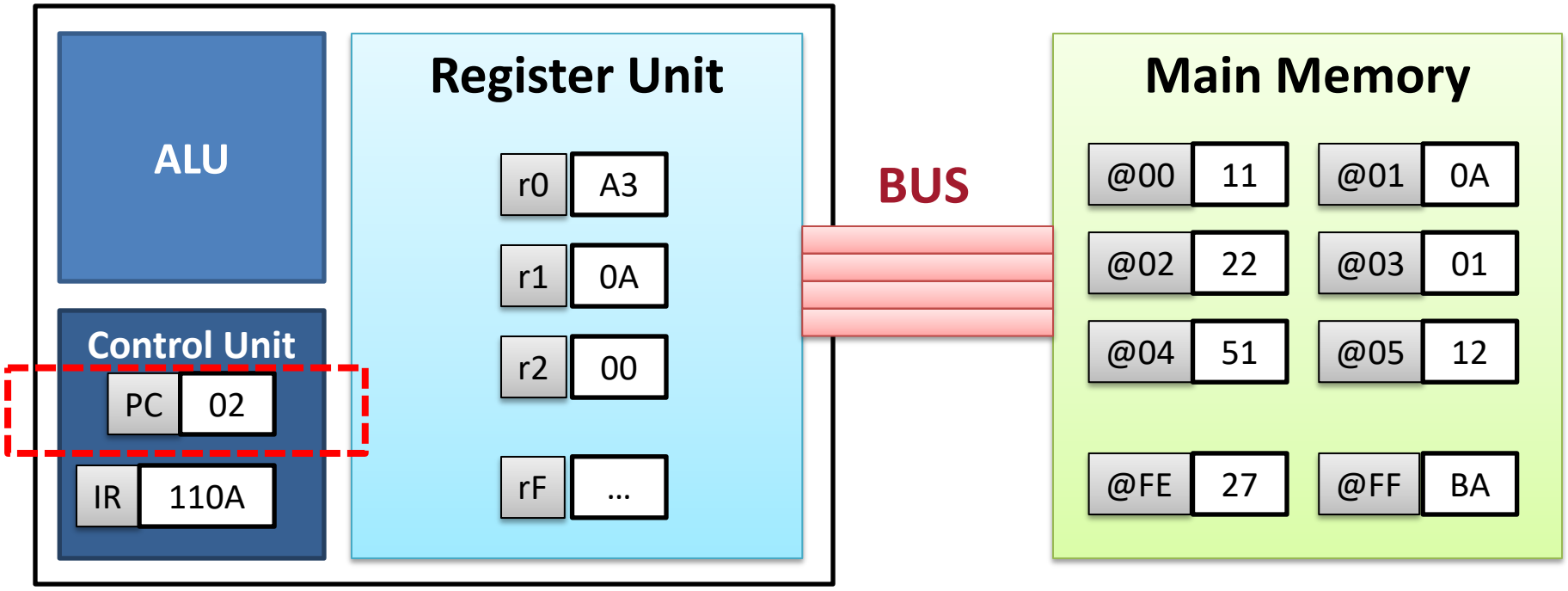
Brookshere's Simple Machine



16 general-purpose registers (r0 - rF)

https://www.amazon.com/Computer-Science-Overview-Glenn-Brookshear/dp/0133760065

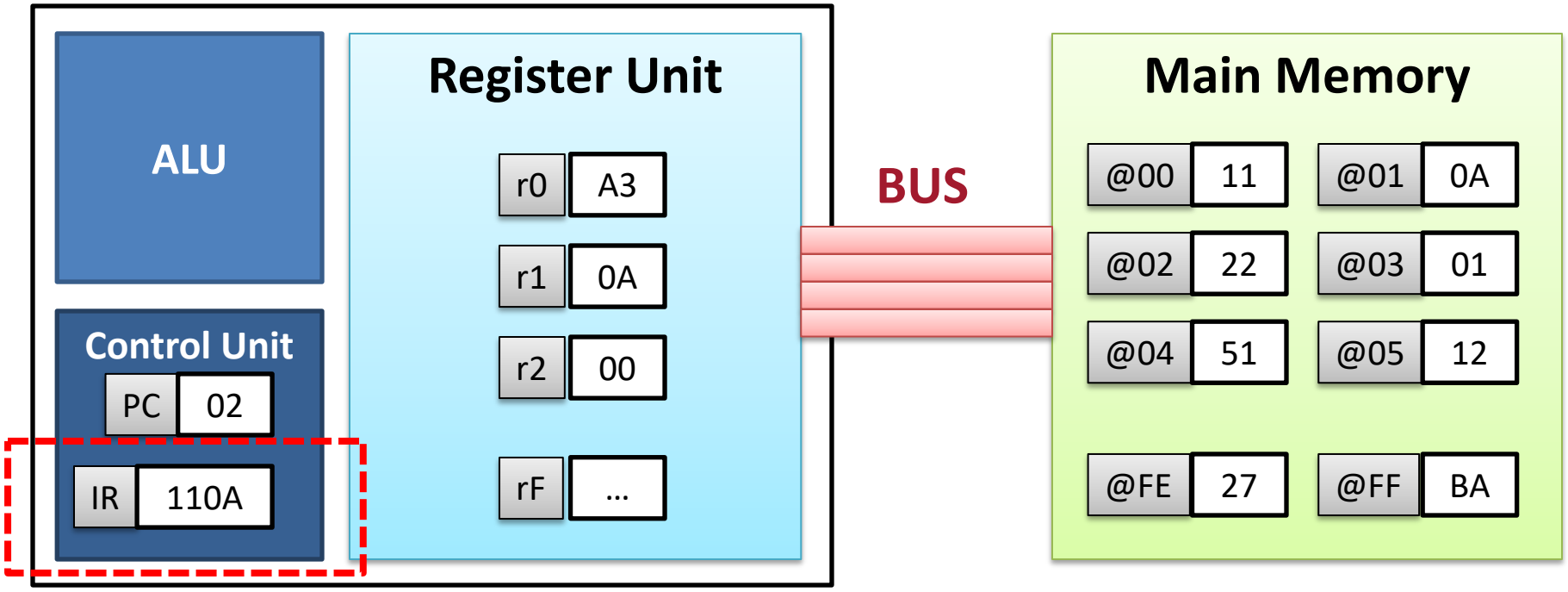
Brookshere's Simple Machine



16 general-purpose registers (r0 - rF)
Program Counter ($PC)

https://www.amazon.com/Computer-Science-Overview-Glenn-Brookshear/dp/0133760065

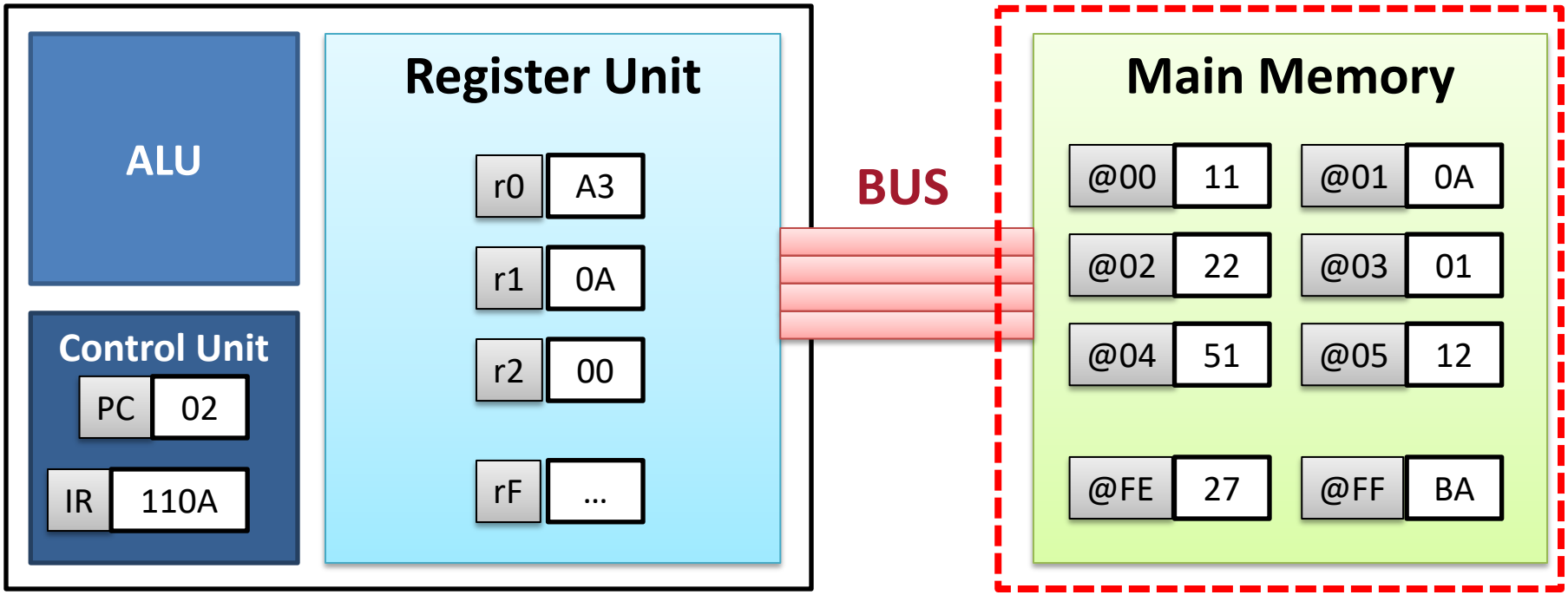
Brookshere's Simple Machine



16 general-purpose registers (r0 - rF)
Program Counter ($PC)
Instruction Register ($IR)

https://www.amazon.com/Computer-Science-Overview-Glenn-Brookshear/dp/0133760065

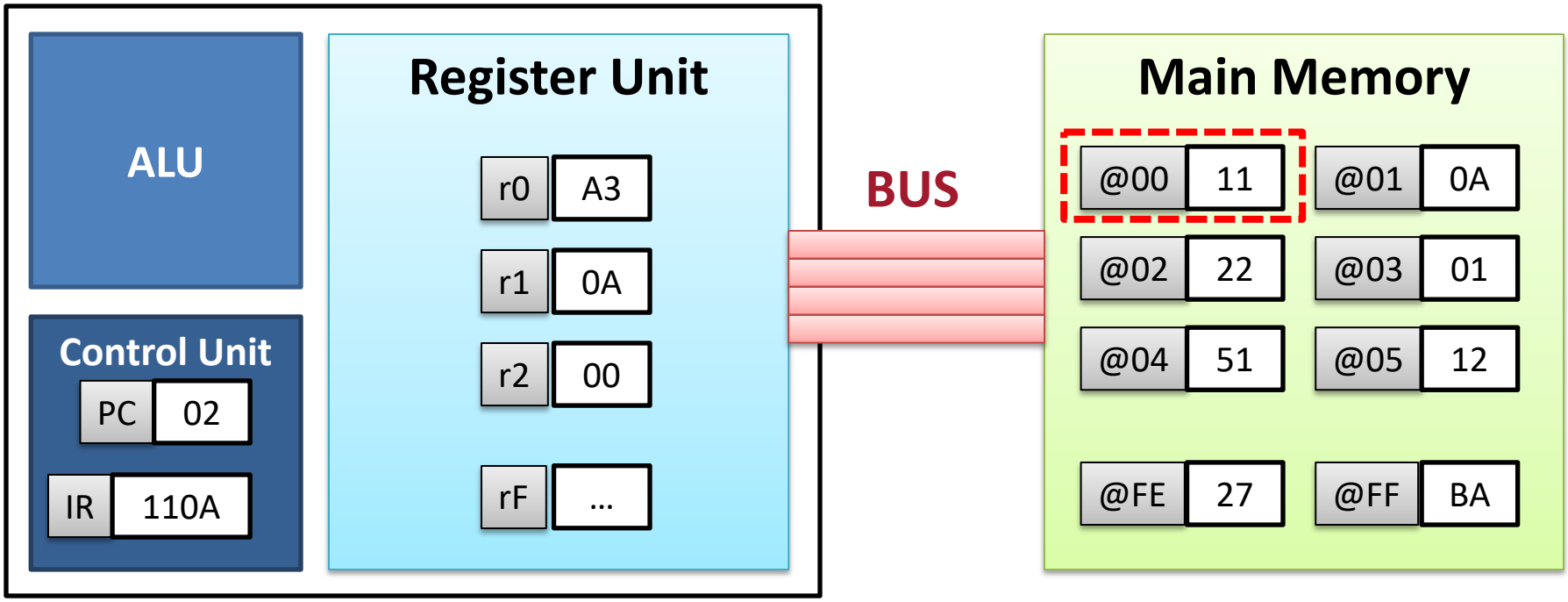
Brookshere's Simple Machine



16 general-purpose registers (r0 - rF)
Program Counter ($PC)
Instruction Register ($IR)
256 bytes main memory (@00 - @FF)

https://www.amazon.com/Computer-Science-Overview-Glenn-Brookshear/dp/0133760065

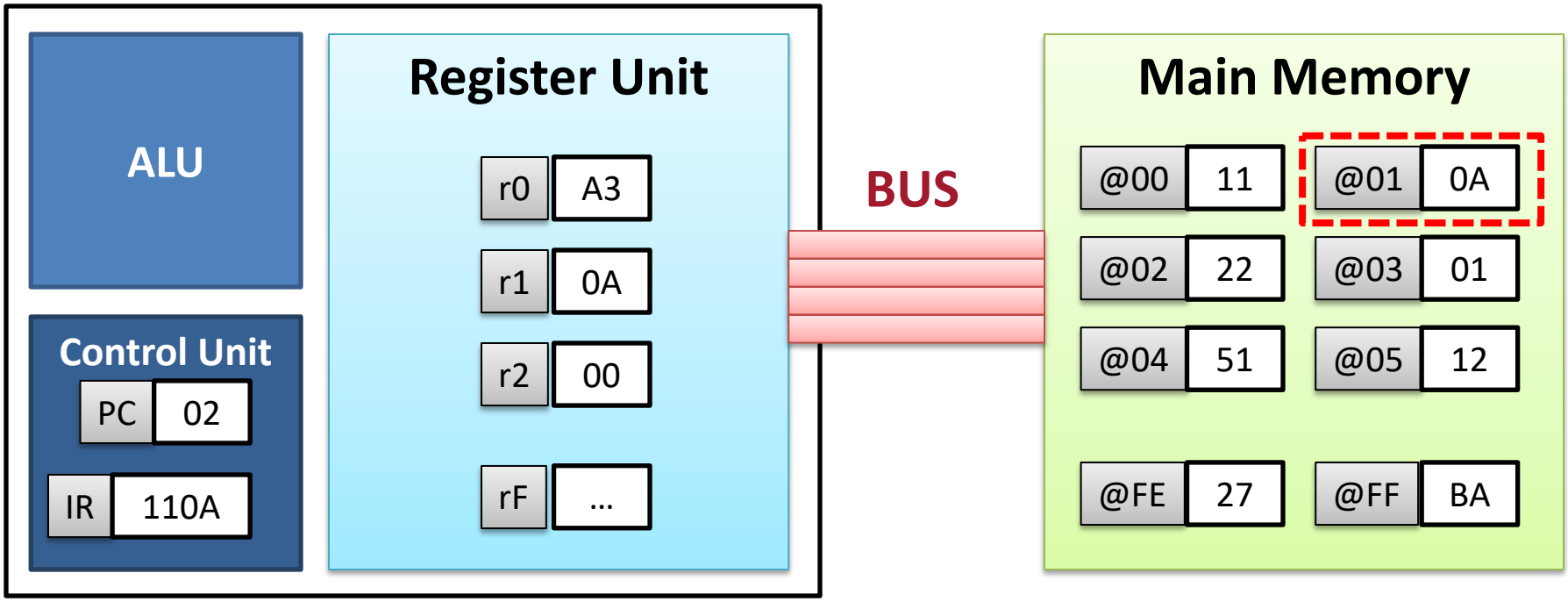
Brookshere's Simple Machine



16 general-purpose registers (r0 - rF)
Program Counter ($PC)
Instruction Register ($IR)
256 bytes main memory (@00 - @FF)

https://www.amazon.com/Computer-Science-Overview-Glenn-Brookshear/dp/0133760065

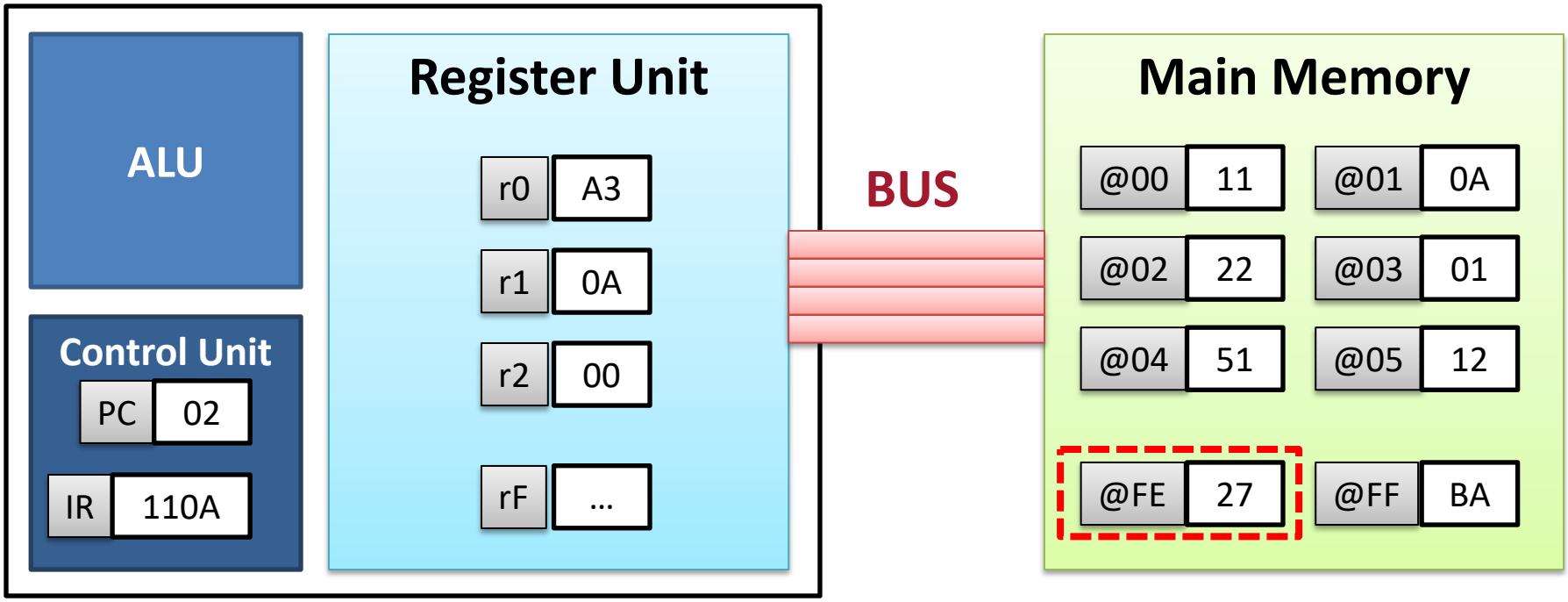
Brookshere's Simple Machine



16 general-purpose registers (r0 - rF)
Program Counter ($PC)
Instruction Register ($IR)
256 bytes main memory (@00 - @FF)

https://www.amazon.com/Computer-Science-Overview-Glenn-Brookshear/dp/0133760065

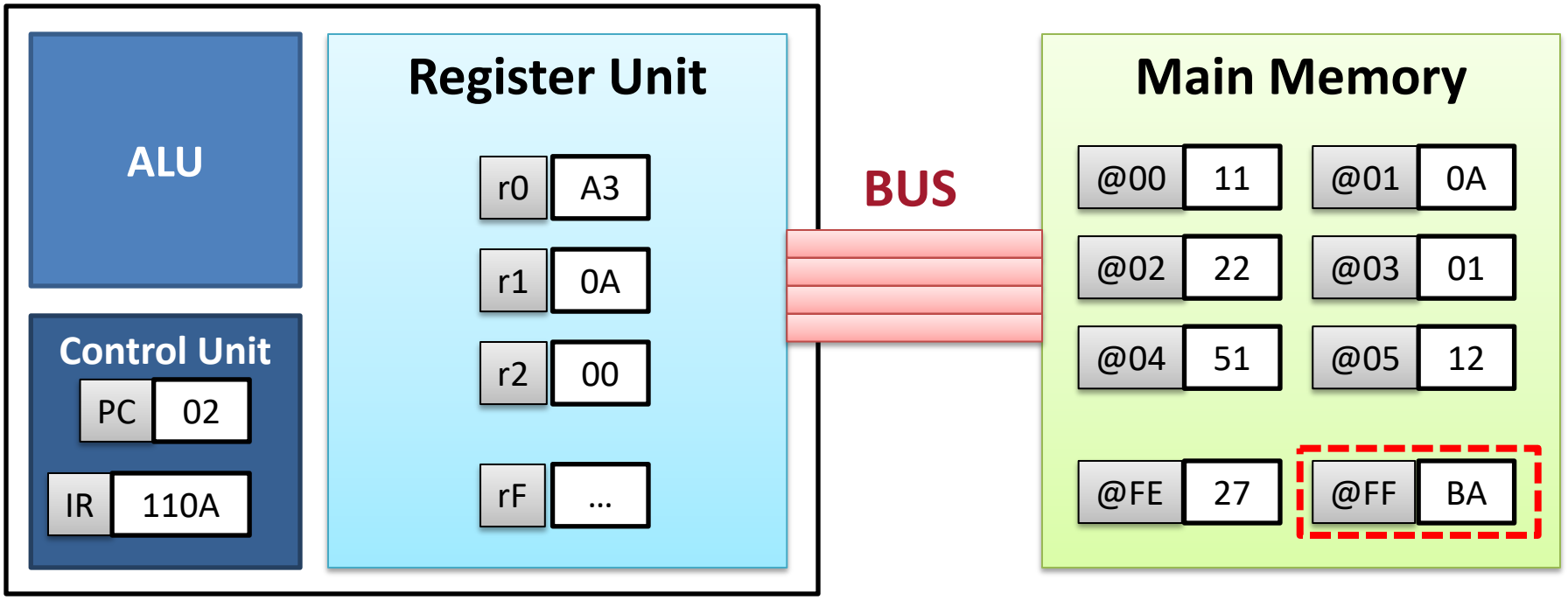
Brookshere's Simple Machine



16 general-purpose registers (r0 - rF)
Program Counter ($PC)
Instruction Register ($IR)
256 bytes main memory (@00 - @FF)

https://www.amazon.com/Computer-Science-Overview-Glenn-Brookshear/dp/0133760065

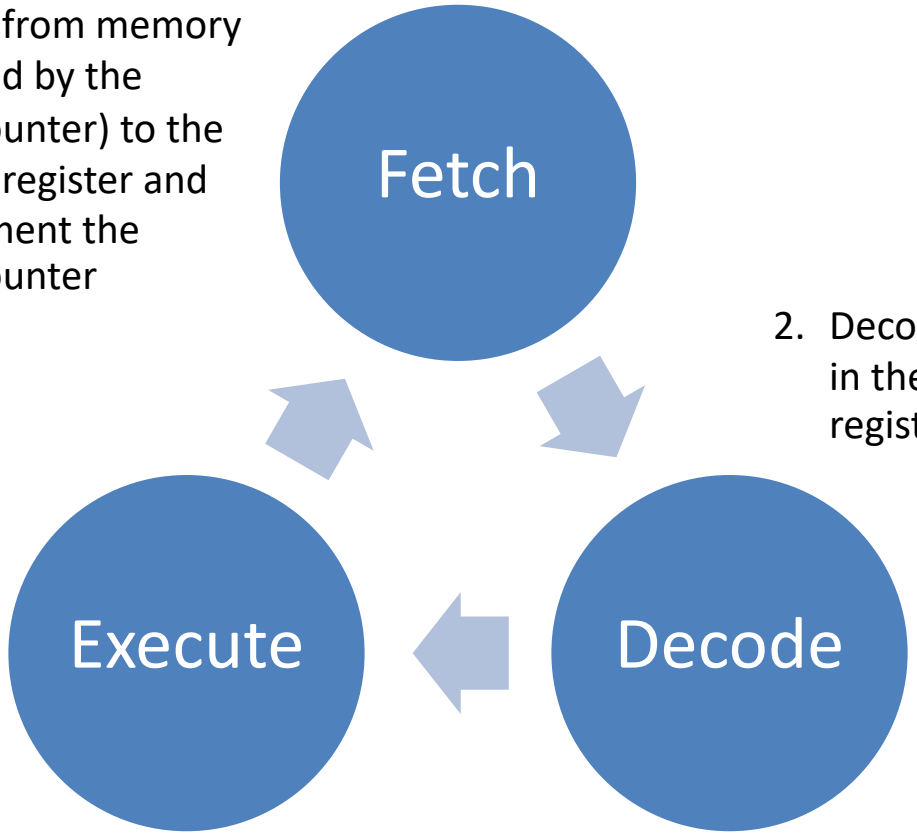
Brookshere's Simple Machine



16 general-purpose registers (r0 - rF)
Program Counter ($PC)
Instruction Register ($IR)
256 bytes main memory (@00 - @FF)

Machine Cycle: Fetch-Decode-Execute

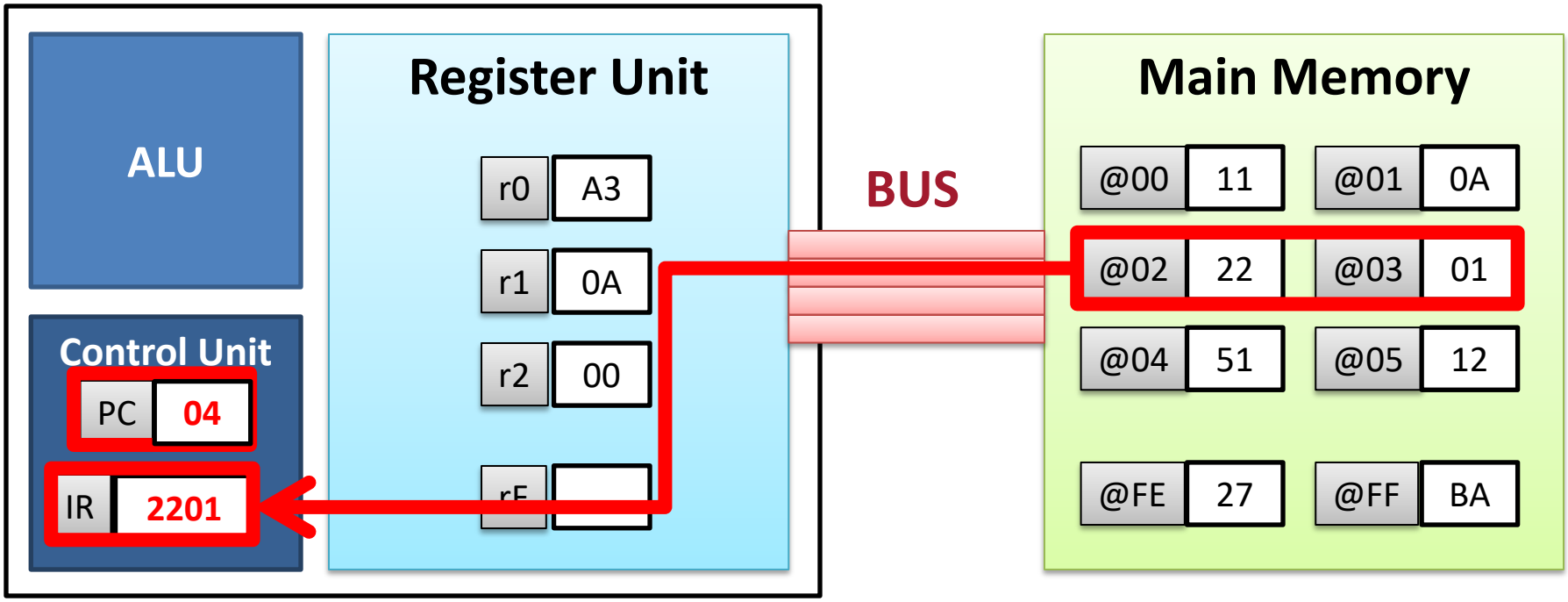
1. Retrieve the next instruction from memory (as indicated by the program counter) to the instruction register and then increment the program counter



2. Decode the bit pattern in the instruction register

3. Perform the action required by the instruction in the instruction register

Machine: Fetch

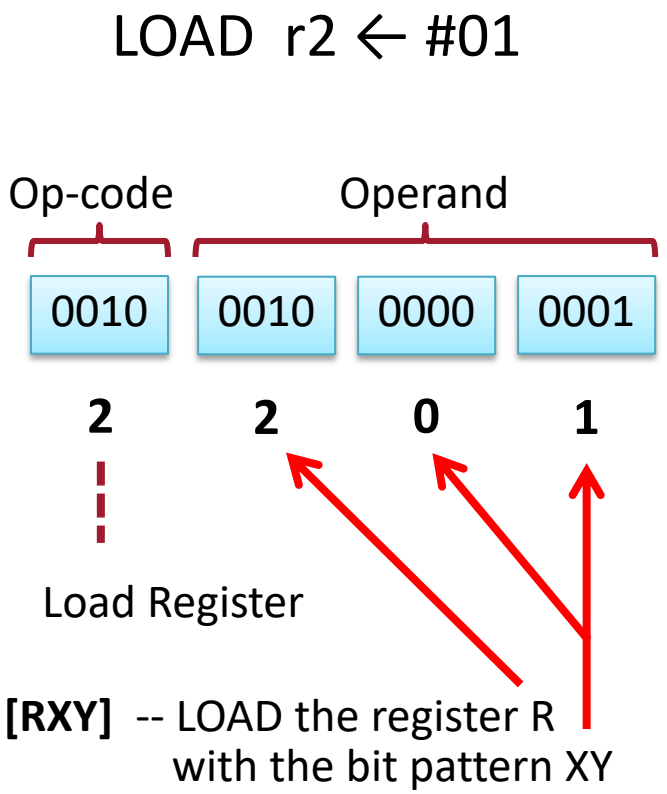


Instruction Types

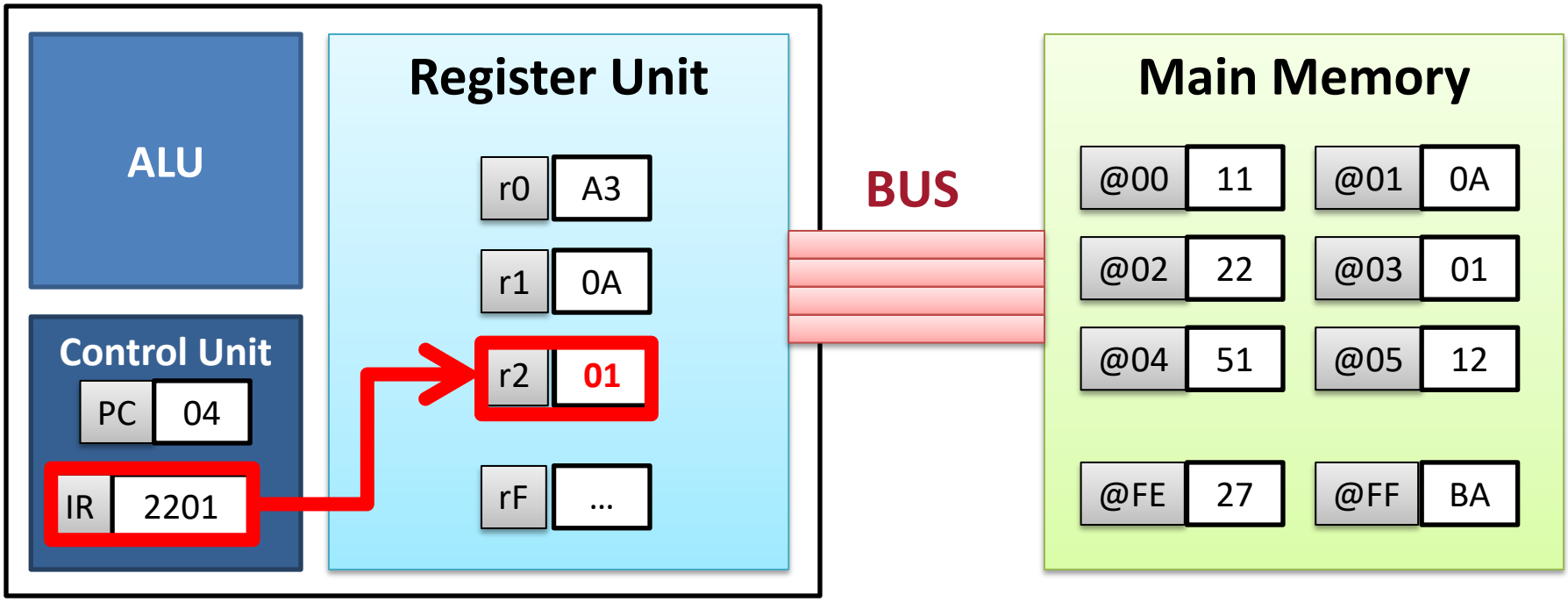
- Data Transfer
  - LOAD from memory, STORE to memory
  - LOAD from constant
  - MOVE between registers
- Arithmetic / Logic
  - Arithmetic operations (ADD integer, ADD floating points)
  - Boolean operations (AND, OR, XOR)
  - Bit-wise operations (ROTATE)
- Control
  - Conditional jumps/branches
  - HALT execution

Parts of Instruction

- Op-code (Operation code) – 4 bits
  - Specify the operation to execute
- Operand – 12 bits
  - Additional details about operation
    - Address, register, constant

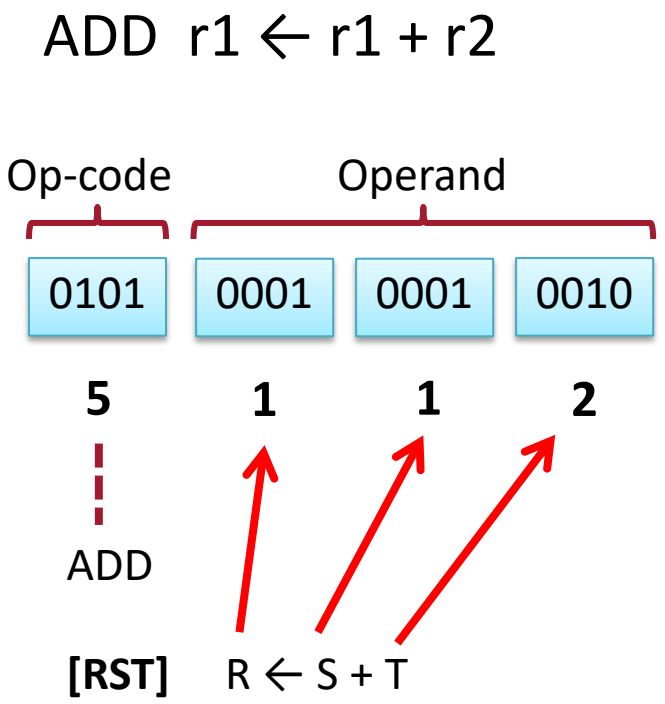


Machine: Execute

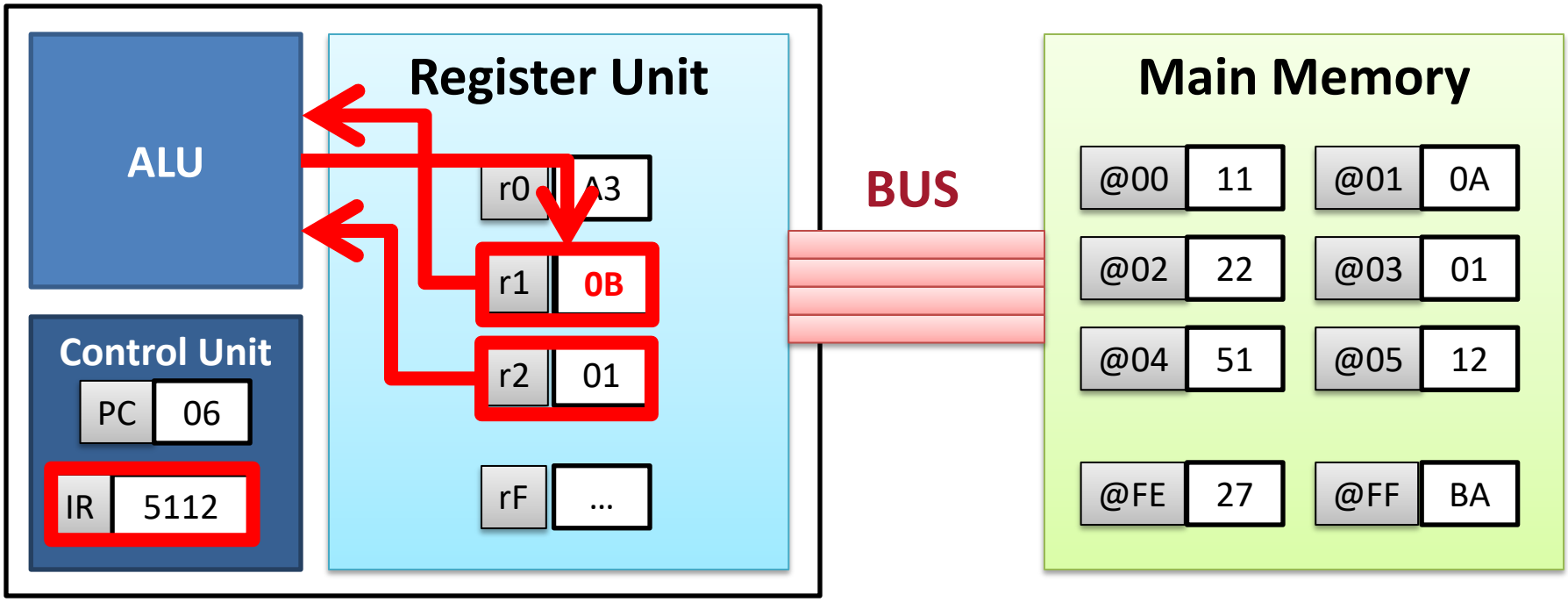


ADD Instruction

- Add values from 2 registers and save the result in a designated register
- This is a two's complements add e.g. you can add value 15 (0x0F) with -1 (0xFF) and the result is 14 (0x0E)



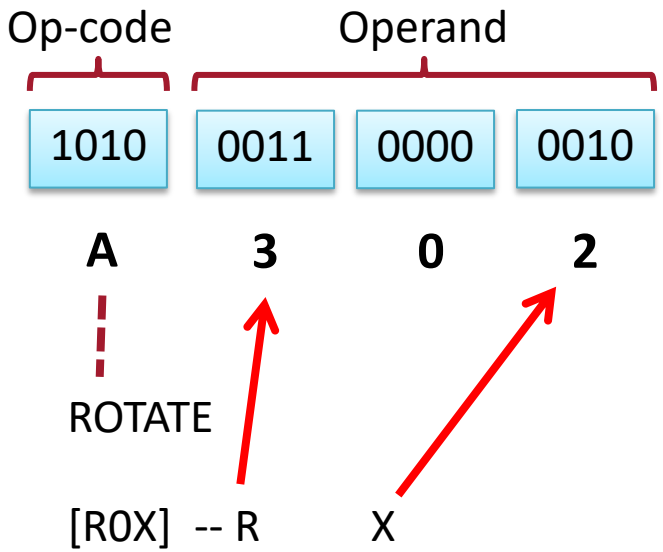
Machine: Execute



Op-code	Operand	Description
1	RXY	LOAD the register R with the bit pattern found in the memory cell whose address is XY.
2	RXY	LOAD the register R with the bit pattern XY.
3	RXY	STORE the bit pattern found in register R in the memory cell whose address is XY.
4	0RS	MOVE the bit pattern found in register R to register S.
5	RST	ADD the bit patterns in registers S and T as though they were two's complement representations and leave the result in register R.
6	RST	ADD the bit patterns in registers S and T as though they represented values in floating-point notation and leave the floating point result in register R.
7	RST	OR the bit patterns in registers S and T and place the result in register R.
8	RST	AND the bit patterns in registers S and T and place the result in register R.
9	RST	EXCLUSIVE OR the bit patterns in registers S and T and place the result in register R.
A	R0X	ROTATE the bit pattern in register R one bit to the right X times. Each time place the bit that started at the low-order end at the high-order end.
B	RXY	JUMP to the instruction located in the memory cell at address XY if the bit pattern in register R is equal to the bit pattern in register number 0. Otherwise, continue with the normal sequence of execution. (The jump is implemented by copying XY into the program counter during the execute phase.)
C	000	HALT execution.

ROTATE Instruction

- Rotate a bit pattern in register R one bit to the right X times
- For each rotation, place the bit started at the low-order end at the high-order end



	Value of r3	
Before Execution	10010101	95
Rotate once	11001010	CA
Rotate twice	01100101	65

JUMP Instruction

- If the value in register R equals to value in register 0, Jump to the instruction located at memory cell @XY
- Otherwise, continue execute next instruction

@20	B43A	JUMP r4, @3A
@22	5442	ADD r4, r4, r2
…	…	
@3A	C000	HALT

- Suppose PC = 0x20
- Case#1: r0 = 0x1F and r4 = 0x1F
  - Value in register 4 equals to register 0
  - Next instruction will be fetch from memory address 3A
- Case#1: r0 = 0x1F and r4 = 0x32
  - Value in register 4 does not equal to register 0
  - Next instruction will be fetch from memory address 22

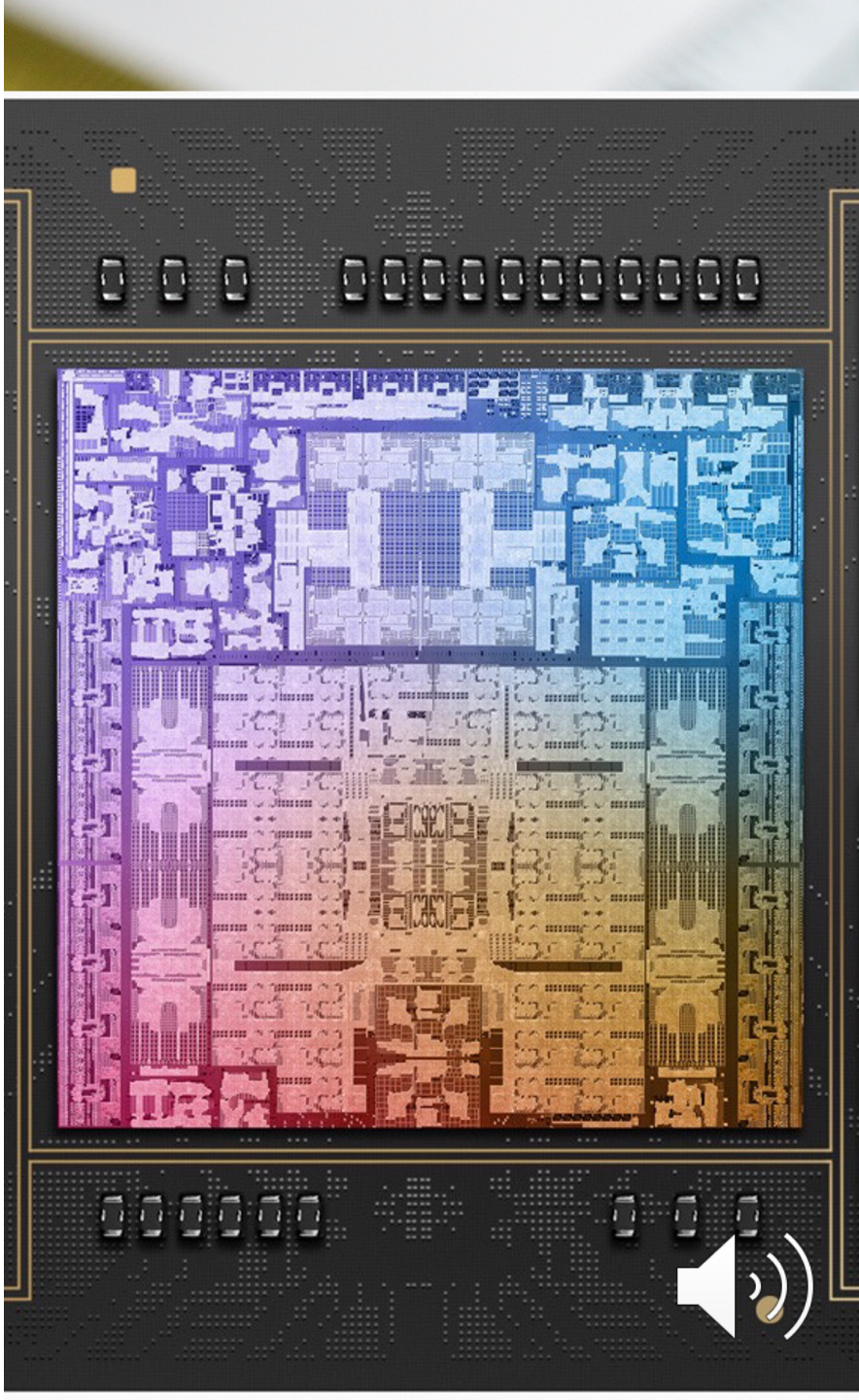
JUMP Instruction (Absolute)

- We can perform an abolute jump (always jump) by setting R in the instruction to be 0
- This will force the program to always jump to the target

@20	B**0**3A	JUMP **r0**, @3A
@22	5442	ADD r4, r4, r2
…	…	
@3A	C000	HALT

Central Processing Unit

Sample Program



Example: $@0A \leftarrow @0A + 1$

```
LOAD  r1, @0A          @00: 110A
LOADI r2, 0x01         @02: 2201
ADD   r1, r1, r2       @04: 5112
STORE r1, @0A          @06: 310A
HALT                   @08: C000
                       @0A: 1000
```

CHULA ENGINEERING
Foundation toward Innovation

SML Emulator

http://joeledstrom.github.io/brookshear-emu/

Shareable link for reproducing the current memory contents (opens in a new window/tab)

CPU

GPRs	
0	00
1	10
2	01
3	00
4	00
5	00
6	00
7	00
8	00
9	00
A	00
B	00
C	00
D	00
E	00
F	00

PC 04

IR 2201

Decoded LOADI r2 <- 0x01

FETCH | DECODE | EXECUTE

Clear and Run

Clear CPU

Run

Step

Memory

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	11	0A	22	01	51	12	31	0A	C0	00	10	00	00	00	00	00
1	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
3	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
4	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
5	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
6	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
7	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
9	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
B	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Inspector

address MEM: 0x0A

hex 0x10

binary 00010000

signed 16

float 0

Testing $@0A \leftarrow @0A + 1$ in Emulator

- We can type our program in hexadecimal numbers into emulator's memory and start the emulator (either run or step)
- We can also share emulator with altered memory using sharable link at the top
- http://joeledstrom.github.io/brookshear-emu/#110A22015112310AC00010
- Note that some notations in the emulator may be slightly different than those being used in this presentation e.g. memory address @00 vs. *(0x00)