Processor Architecture

Types of Processor Architecture

- Von-Neumann Architecture
- Harward Architecture
- CISC(Complex Instruction Set computer) Architecture
- RISC(Reduced Instruction Set computer) Architecture

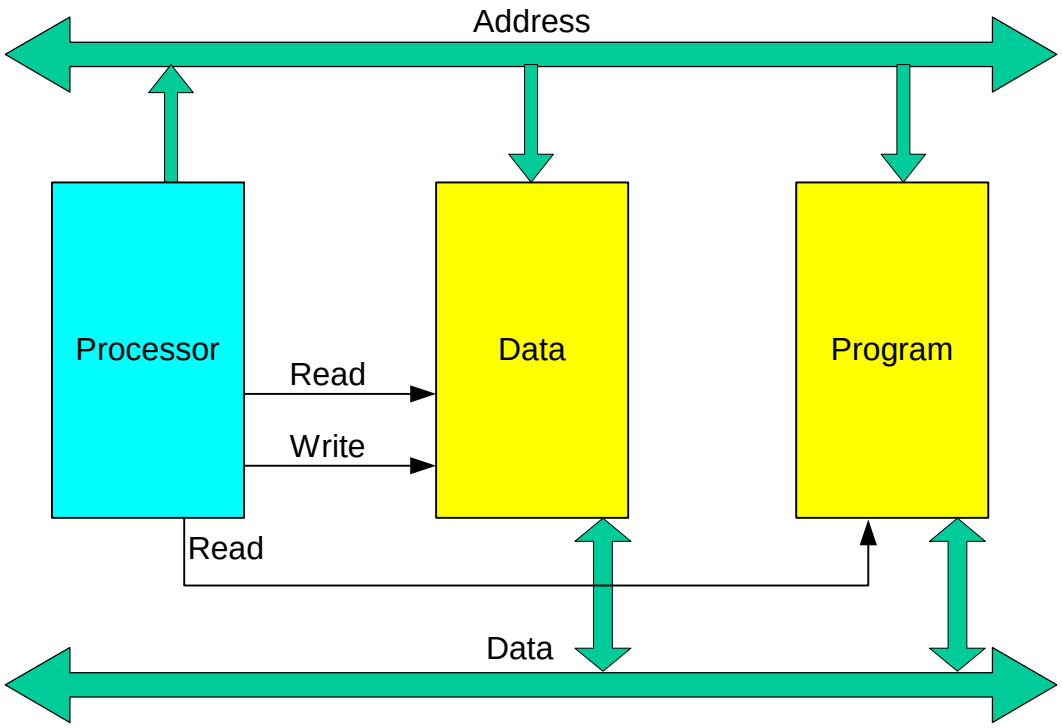
How to classify processors

- Categorized by memory organization
  - Von-Neumann architecture
  - Harvard architecture
- Categorized by instruction type
  - CISC
  - RISC

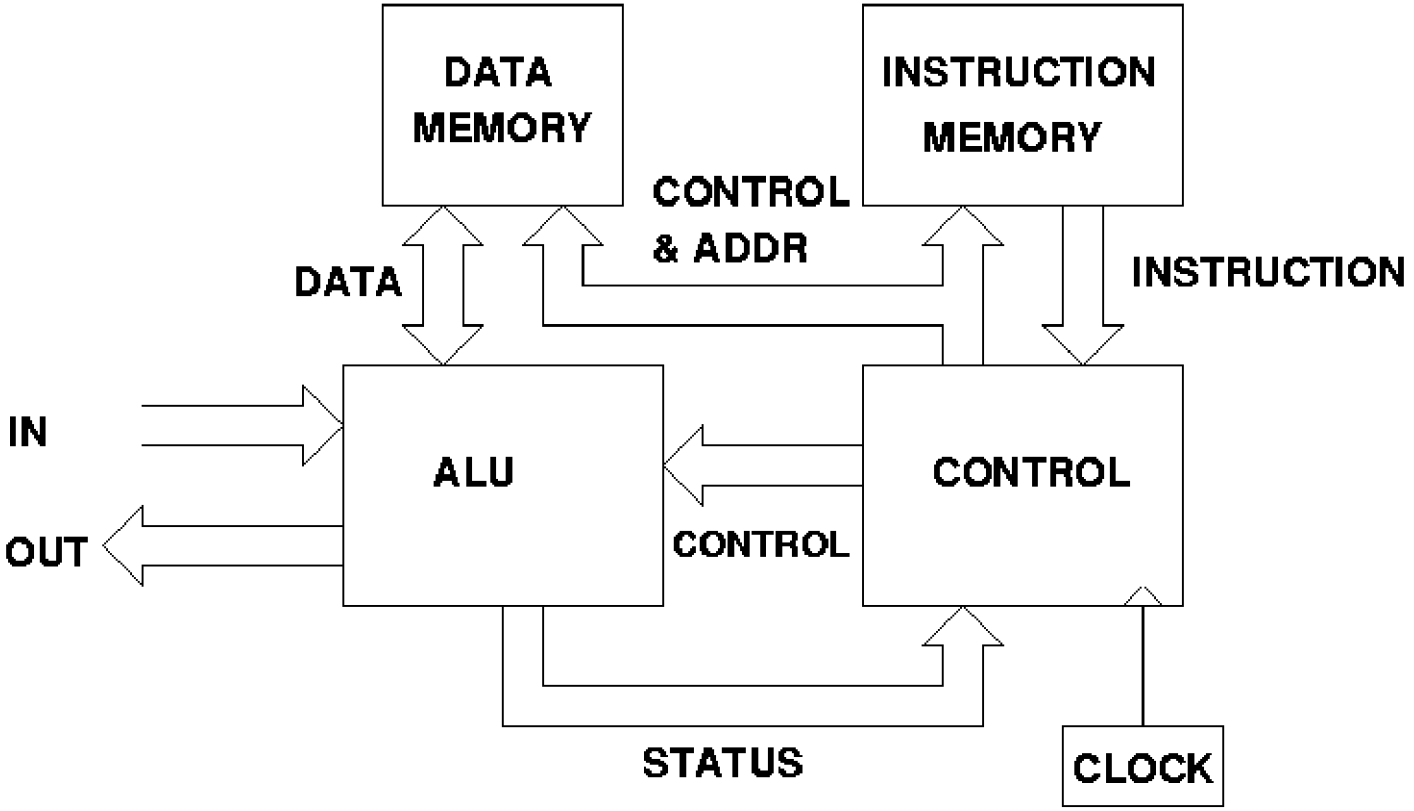
Harvard architecture

- Separate memory into 2 types
  - Program memory
  - Data memory
- Used in MCS-51, MIPS etc.

Harvard architecture



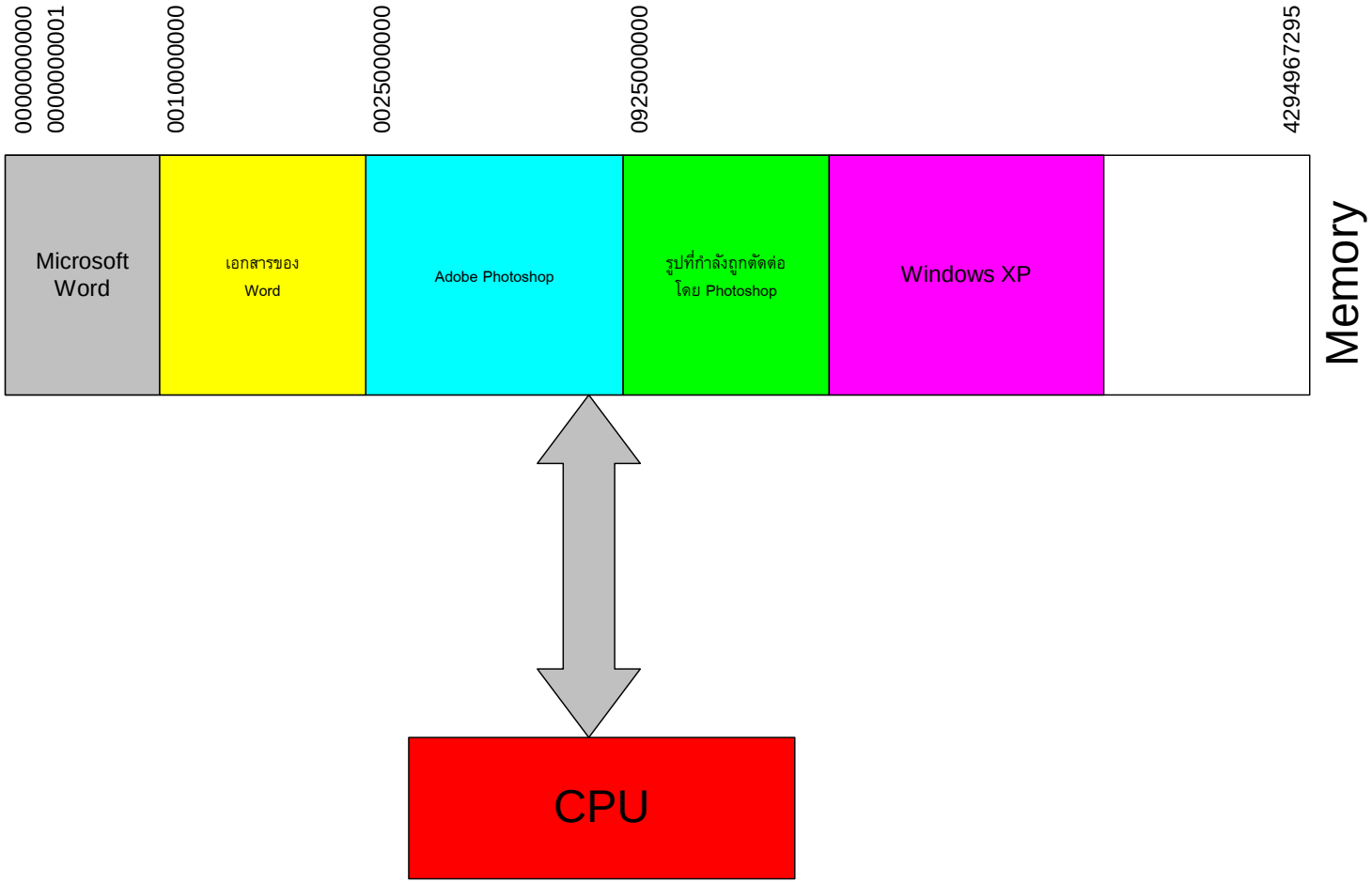
Harvard architecture



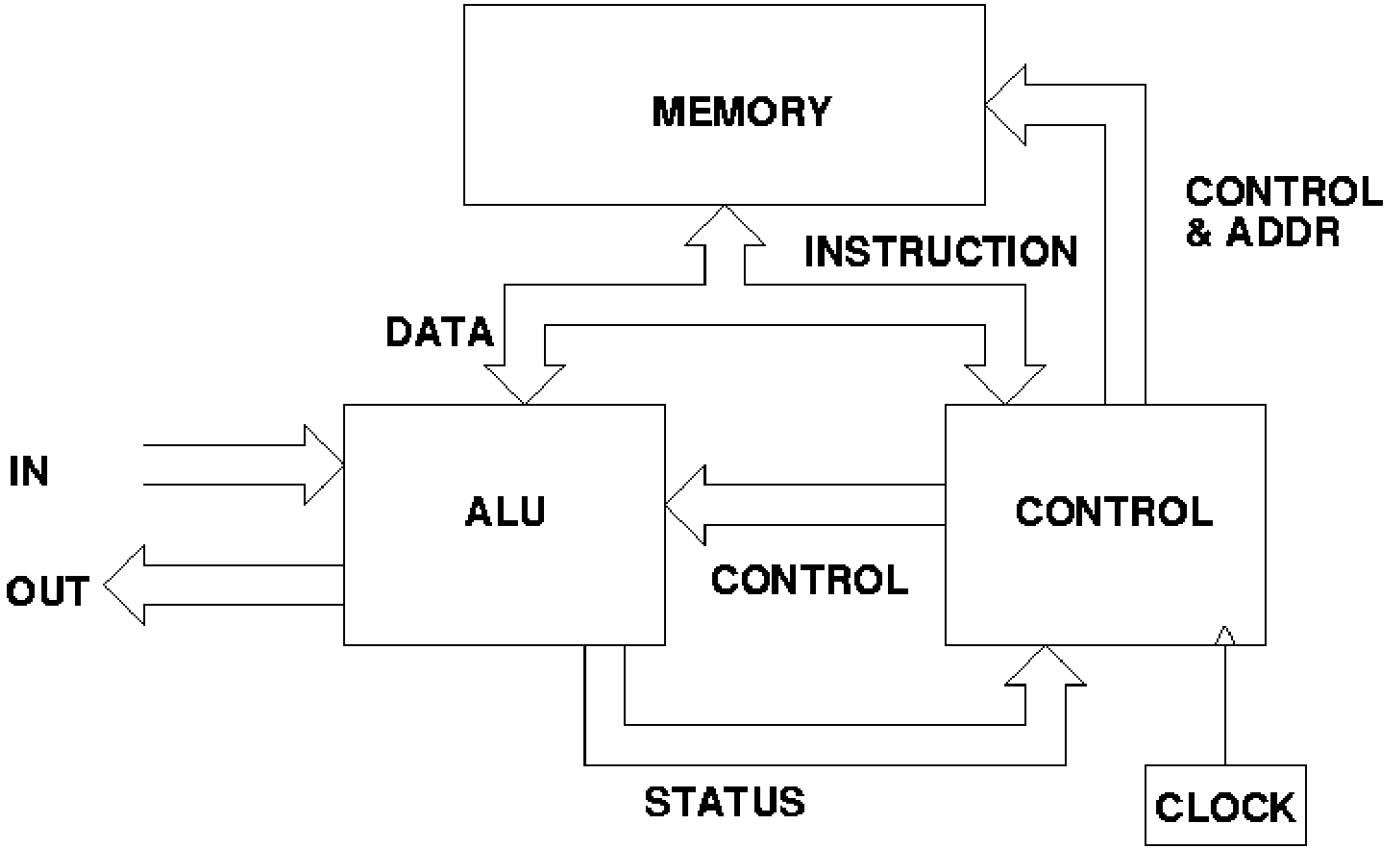
Von-Neumann architecture

- Combine program and data in 1 chunk of memory
- Example : 80x86 architecture

Von-Neumann architecture



Von-Neumann architecture



Features CISC

- Complex instruction set computer
- Large number of instructions (~200-300 instructions)
- Specialized complex instructions
- Many different addressing modes
- Variable length instruction format
- Memory to memory instruction
- For Example : 68000, 80x86

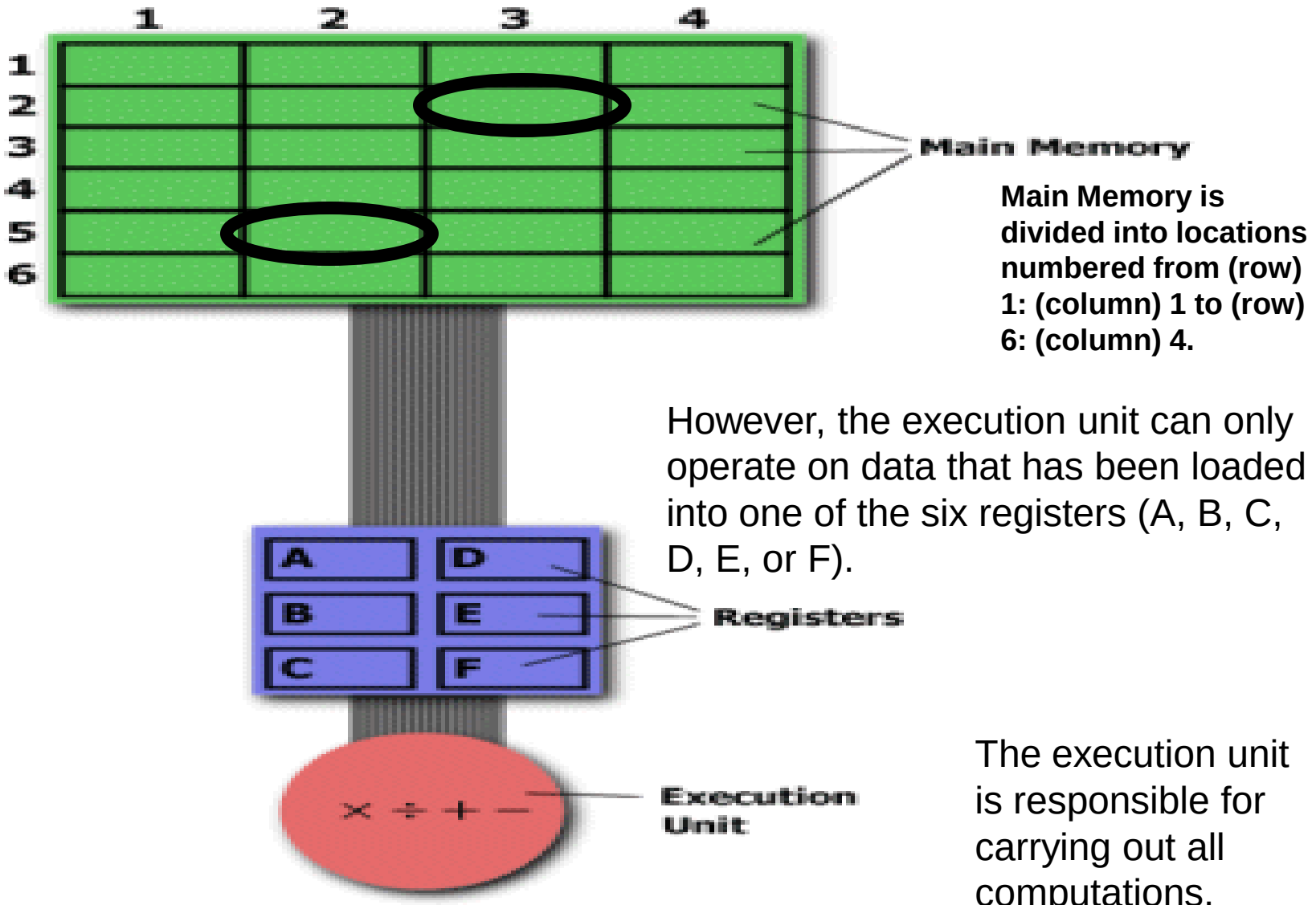
Features RISC

- Reduced instruction set computer
- Relatively few number of instructions (~50)
- Basic instructions
- Relatively few different addressing modes
- Fixed length instruction format
- Only load/store instructions can access memory

Features RISC

- Large number of registers
- Hardwired rather than micro-program control
- For Example : MIPS, Alpha, ARM etc.

Multiplying Two Numbers in Memory



Main Memory is divided into locations numbered from (row) 1: (column) 1 to (row) 6: (column) 4.

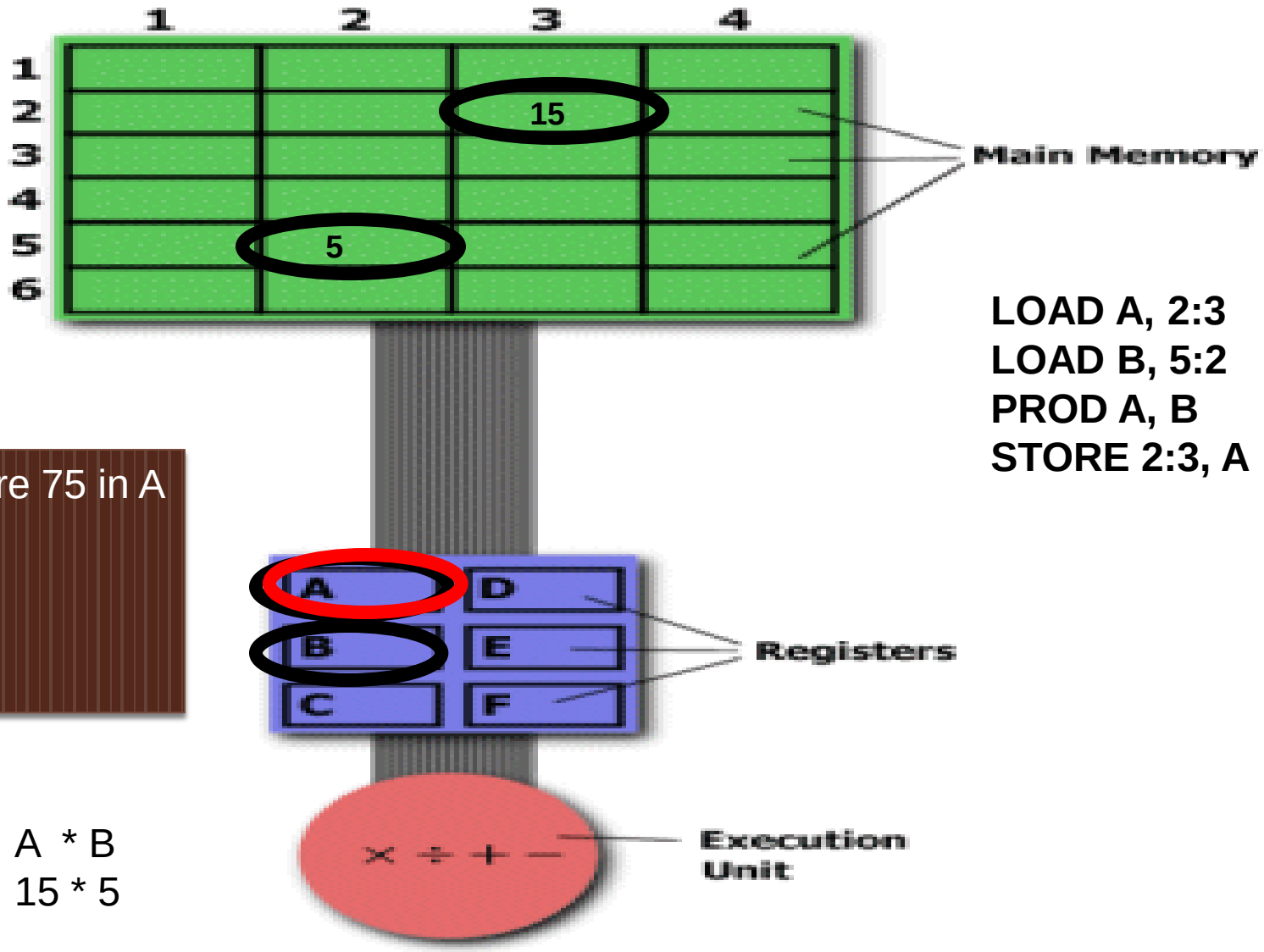
However, the execution unit can only operate on data that has been loaded into one of the six registers (A, B, C, D, E, or F).

The execution unit is responsible for carrying out all computations.

- Diagram represents the storage scheme for a generic computer
- The main memory is divided into locations numbered from (row) 1: (column) 1 to (row) 6: (column) 4.
- The execution unit is responsible for carrying out all computations. However, the execution unit can only operate on data that has been loaded into one of the six registers (A, B, C, D, E, or F).
- **Let's say we want to find the product of two numbers - one stored in location 2:3 and another stored in location 5:2 - and then store the product back in the location 2:3.**

RISC APPROACH

Multiplying Two Numbers in Memory



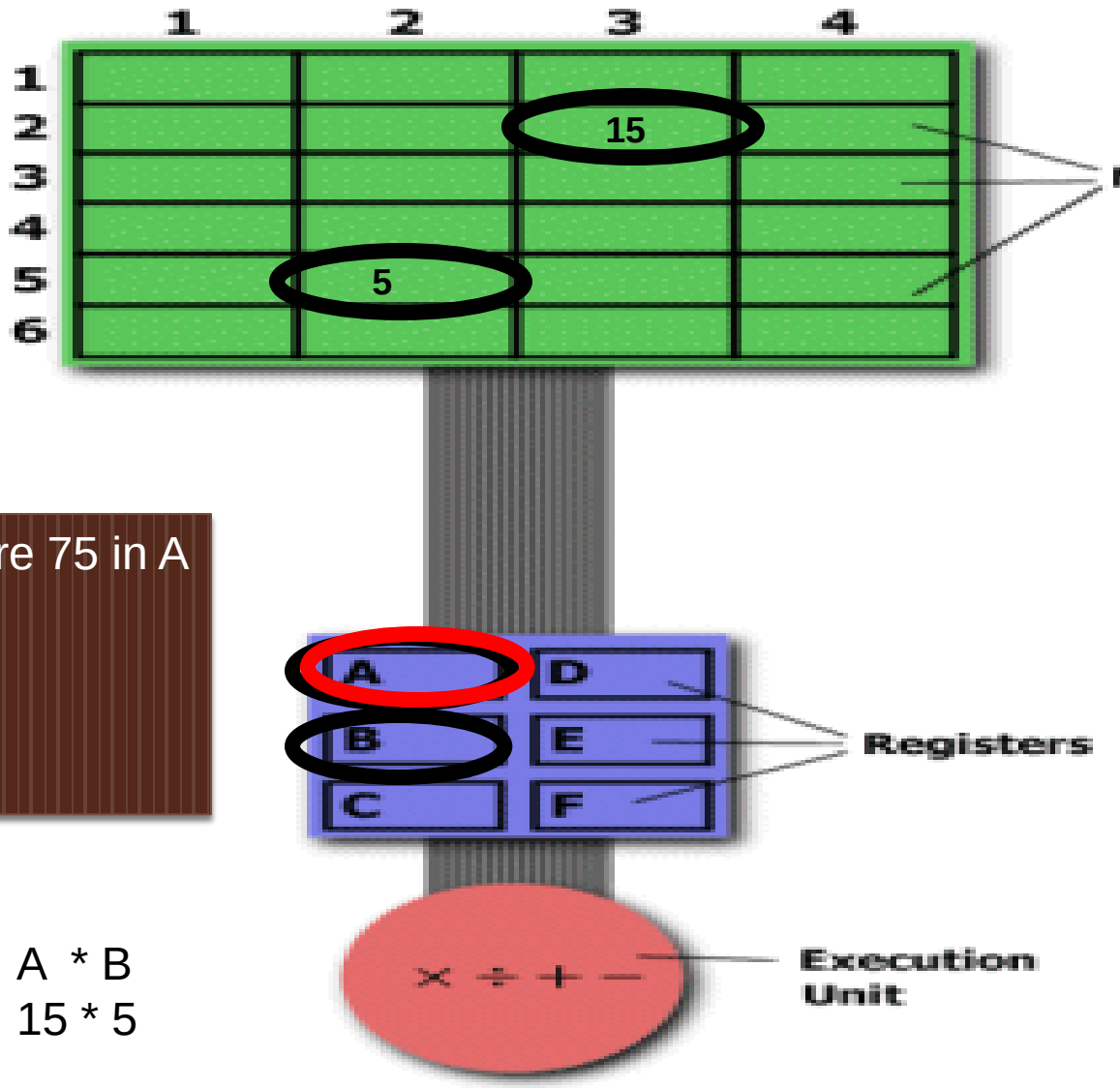
Store 75 in A

LOAD A, 2:3
LOAD B, 5:2
PROD A, B
STORE 2:3, A

A * B
15 * 5

CISC APPROACH

Multiplying Two Numbers in Memory



Store 75 in A

A * B
15 * 5

MULT 2:3, 5:2

"a = a * b."

The CISC Approach

- The primary goal of CISC architecture is **to complete a task in as few lines as possible.**
- This is achieved if **we build a processor hardware that is capable of understanding and executing a series of operations**.
- For this particular task, a CISC processor would come prepared with a specific instruction (we'll call it "**MULT**").
- **When executed, this instruction loads the two values into separate registers, multiplies the operands in the execution unit, and then stores the product in the appropriate register**
- Thus, the **entire task of multiplying two numbers can be completed with one instruction:**

MULT 2:3, 5:2

- MULT is what is known as a *"complex instruction."*
- **It operates directly on the computer's memory banks and does not require the programmer to explicitly call any loading or storing functions.**
- **It *closely resembles a command in a higher level language*.** For instance, if we let "a" represent the value of 2:3 and "b" represent the value of 5:2, then this command is identical to the C statement "a = a * b."
- **One of the primary advantages of this system is that *the compiler has to do very little work to translate a high-level language* statement into assembly. Because the length of the code is relatively short, *very little RAM is required to store instructions.***
- **The emphasis is put on building complex instructions directly into** the hardware.

The RISC Approach

- RISC processors only **use simple instructions that can be executed within one clock cycle.**

- Thus, the "**MULT**" command described above could be **divided** into **three separate commands:** "**LOAD**," which moves data from the memory bank to a register, "**PROD**," which finds the product of two operands located within the registers, and "**STORE**," which moves data from a register to the memory banks.

```
LOAD A, 2:3
LOAD B, 5:2
PROD A, B
STORE 2:3, A
```

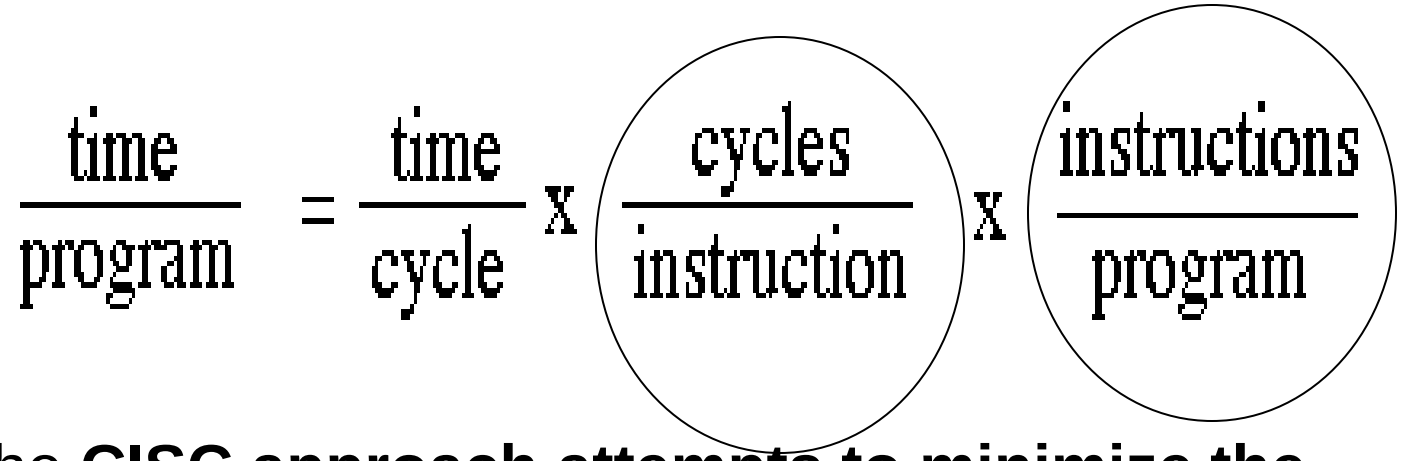
- This may seem like a much less efficient way of completing the operation. Because there are **more lines of code, more RAM** is needed to store the assembly level instructions. **The compiler must also perform more work to convert** a high-level language statement into code of this form.

- However, the **RISC strategy also brings some very important advantages**
- Because each instruction requires only one clock cycle to execute, the entire program will execute in approximately the same amount of time as the multi-cycle "MULT" command.

- These RISC "reduced instructions" **require less transistors of hardware space than the complex instructions**, leaving more room for general purpose registers

The Performance Equation

- The following equation is commonly used for expressing a computer's performance ability:

- 

- The **CISC approach attempts to minimize the number of instructions per program, sacrificing the number of cycles per instruction**.

- **RISC does the opposite, reducing the cycles per instruction at the cost of the number of instructions per program.**

- Separating the "LOAD" and "STORE" instructions actually reduces the amount of work that the computer must perform.

- After a **CISC-style "MULT" command is executed, the processor automatically erases the registers.** If one of the operands needs to be used for another computation, the processor must **re-load the data from the memory bank into a register.**

- In **RISC, the operand will remain in the register until another value is loaded in its plac**e.

- **CISC**
  - Emphasis on hardware
  - Includes multi-clock complex instructions
  - Memory-to-memory: "LOAD" and "STORE" incorporated in instructions
  - Small code sizes, high cycles per second
  - Less Transistors used for storing complex instructions
  - Easy to program
  - Efficient use of memory
- **RISC**
  - Emphasis on software
  - Single-clock, reduced instruction only
  - Register to register: "LOAD" and "STORE" are independent instructions
  - Low cycles per second, large code size
  - Spends more transistors on memory registers

Addressing Modes

- The operation field of the instruction specifies the operation to be performed.
- This operation must be executed on some data stored in computer registers on memory
- The way the operands are chosen during program execution is dependent on the addressing modes of the instruction
- So we say that addressing tells us where operands are stored

- Three major phases of instruction cycle.

- Fetch the instruction
- Decode
- Execute

Program counter

- Pc keep track of the instructions in the program stored in memory.
- PC holds the address of the instruction to be executed next and is incremented each time an instruction is fetched from memory.
- The decoding done in step 2 determines the operation to be performed, the addressing mode of the instruction and the location of the operands.
- The computer then executes the instruction and returns to step 1 to fetch the next instruction in sequence.

- Instruction format also consists of addressing mode field sometimes.

- The operation code specifies the operation to be performed
- The mode field is used to locate the operands needed for the operation
- There may or may not be an address field in the instruction
- If there is an address field it may designate a memory address or a processor register.
- There are two modes that need no address field at all
- Implied modes
- Immediate modes

Implied modes

- In this mode the operands are specified implicitly in the definition of the instructions.
- Ex: the instruction “ complement accumulator” is an implied mode instruction because the operand in the accumulator register is implied in the definition of the instruction.
- In fact all register reference instruction that use an accumulator are implied mode instruction

Immediate mode

- In this mode the operands is specified in the instruction itself.
- An immediate mode instruction has an operand field rather than an address field
- The operand field contains an actual operand
- Immediate modes instruction are useful for initializing register to the constant value.

Register mode

- In this mode the operands are in registers that reside within the CPU

- The particular register is selected from a register field in the instruiction

Register indirect mode

- In this mode the instruction specifies the register in the CPU whose contents give the address of the operands in the memory

- The selected register contains the address of the operands rather than the operands itself

Effective address

- Is the address of the operand in a computational type instruction

Direct address mode

- In this mode the effective address is equal to the address part of the instruction

Indirect address mode

- In the mode the address field of the instruction gives the address where the effective address is stored in the memory

- Control fetches the instruction from memory and uses its address part to access memory again to read the effective address

Relative address mode

- In this mode the contentr ogf the program counter is added to the address [part of the instruction in order to obtain the effective address

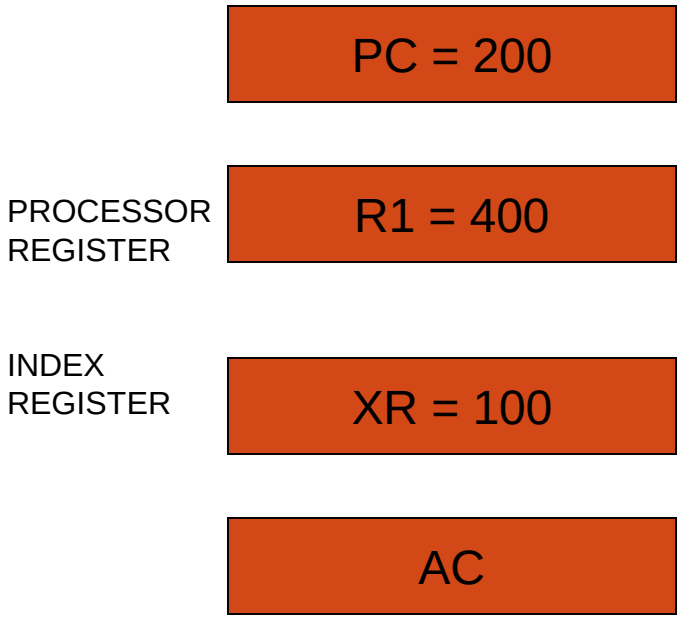
Index Register

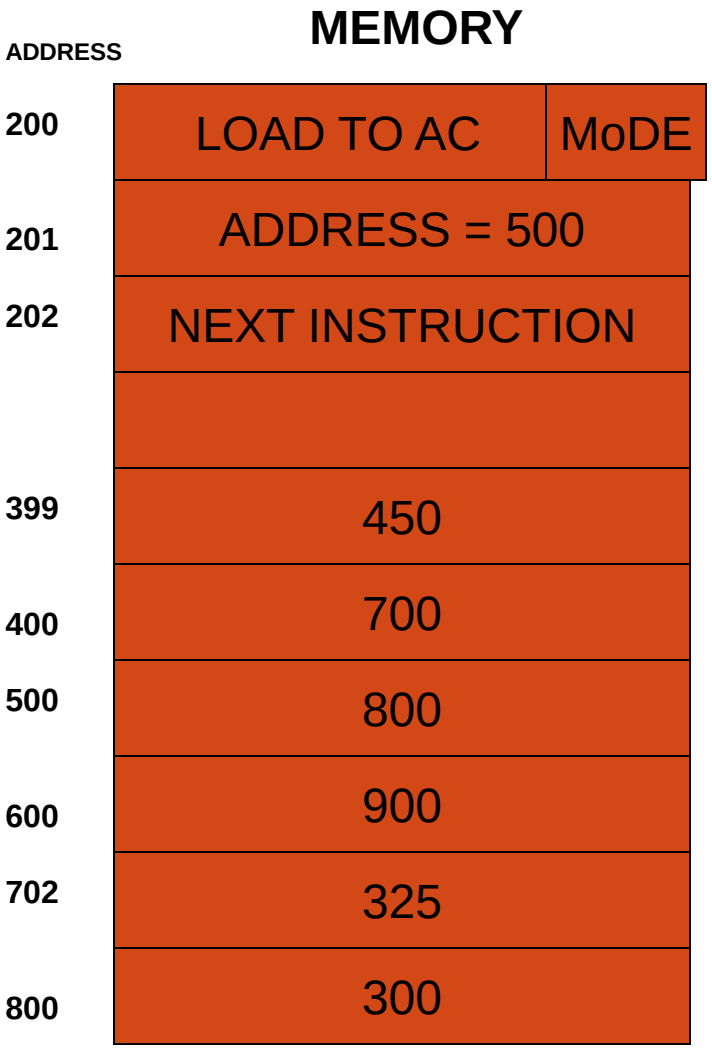
- An **index register** in a computer's CPU is a processor register used for modifying operand addresses during the run of a program

Indexed addressing mode

- In this mode the contents of the indexed register is added to the address part of the instruction to obtain the effective address
- The index register is the special CPU register that contain an index value
- The address field of the instruction defines the beginning address of the data array in memory
- The distance between the beginning address and the address of the operand is the index value stored in the index register.

Example





INSTRUCTION IS AT ADDRESS 200 & 201
LOAD TO AC
ADDRESS FIELD = 500

Find the effective address and content of AC if the mode is :

- Direct address mode
- Immediate operand
- Indirect address
- Relative address
- Indexed register
- Register
- Register indirect

Addressing modes	Effective address	Contents of AC
Direct address mode	500	800
Immediate operand	201	500
Indirect address	800	300
Relative address	702	325
Indexed register	600	900
Register	--	400
Register indirect	400	700