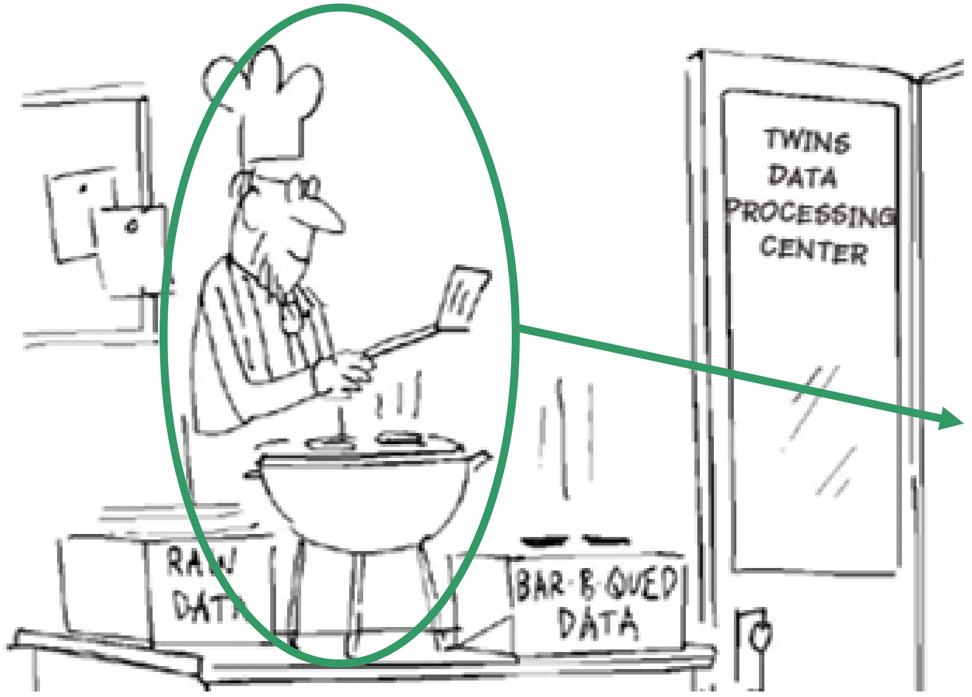
Computer Science- Elaboration

- **Science of ‘abstraction’:**

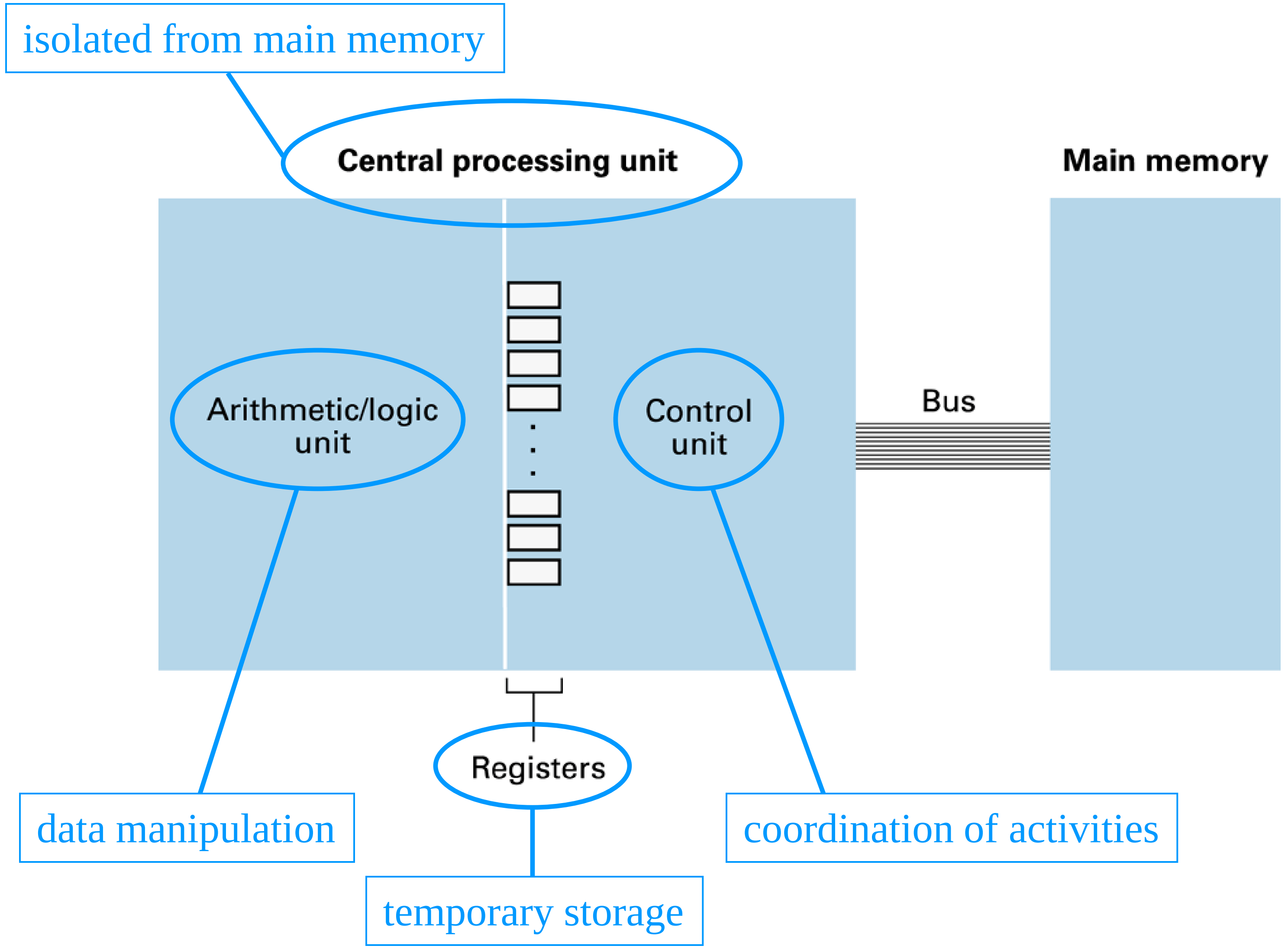
CHAPTER 2

Data Manipulation



Central Processing Unit (CPU)

2.1: The 'Von Neumann Architecture': CPU and Main Memory connected via a Bus



2.1: Central role of the Control Unit

- To perform an operation on data stored in main memory, the control unit must
  - transfer the data from main memory into registers
  - inform the arithmetic/logic unit which registers hold the data
  - activate appropriate arithmetic/logic unit circuitry
  - tell the arithmetic/logic unit which register should receive the result
  - transfer the result from that register to main memory.

2.1: Flexibility of Execution

- Early computers were not flexible
  - operations were built into control unit
- Programs now encoded/stored in main memory
  - known as: *stored-program concept*
- As a consequence, the control unit must also
  - fetch (extract) the program from main memory
  - decode the set of instructions
  - execute each instruction in turn

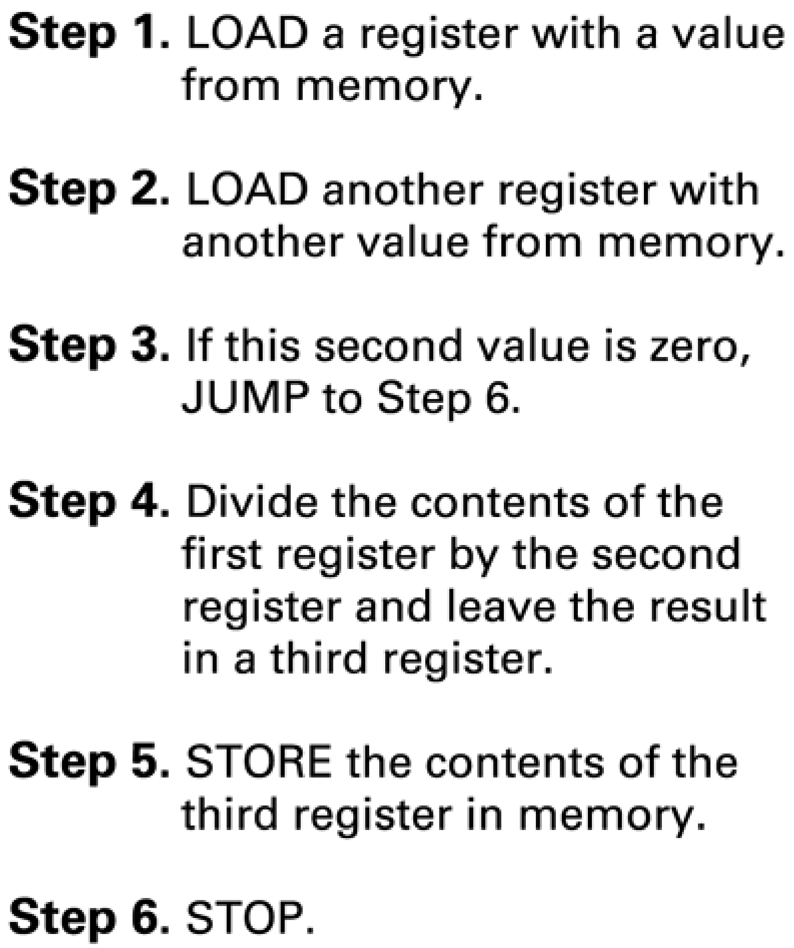
2.2: Machine Language

- To apply the stored-program concept:
  - CPUs must recognize encoded instructions
- Collection of instructions known as:
  - *Instruction set*, or
  - *Machine language*
- Essential set of instructions quite small
  - Additional instructions do not increase capabilities

2.2: Instruction categories

- Machine instructions classified into 3 categories:
  - (1) Data Transfer
    - move/copy data from one location to another
    - typical examples: LOAD, STORE
  - (2) Arithmetic/Logic
    - perform actual operations on data
    - typical examples: AND, OR, XOR, SHIFT, ROTATE
  - (3) Control
    - manipulate the execution of a program
    - typical examples: JUMP, HALT

2.2: Example: dividing 2 values stored in memory



Step 1. LOAD a register with a value from memory.

Step 2. LOAD another register with another value from memory.

Step 3. If this second value is zero, JUMP to Step 6.

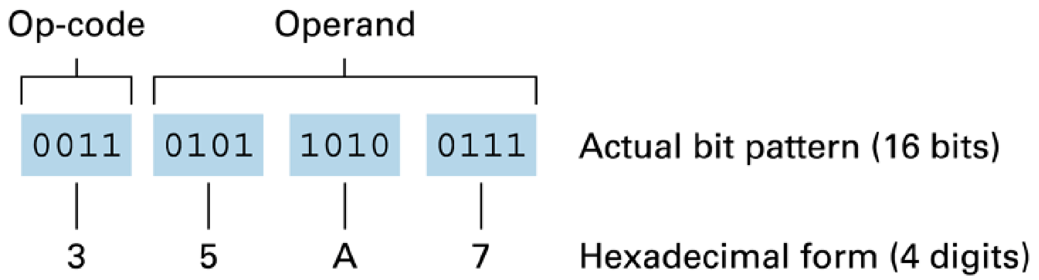
Step 4. Divide the contents of the first register by the second register and leave the result in a third register.

Step 5. STORE the contents of the third register in memory.

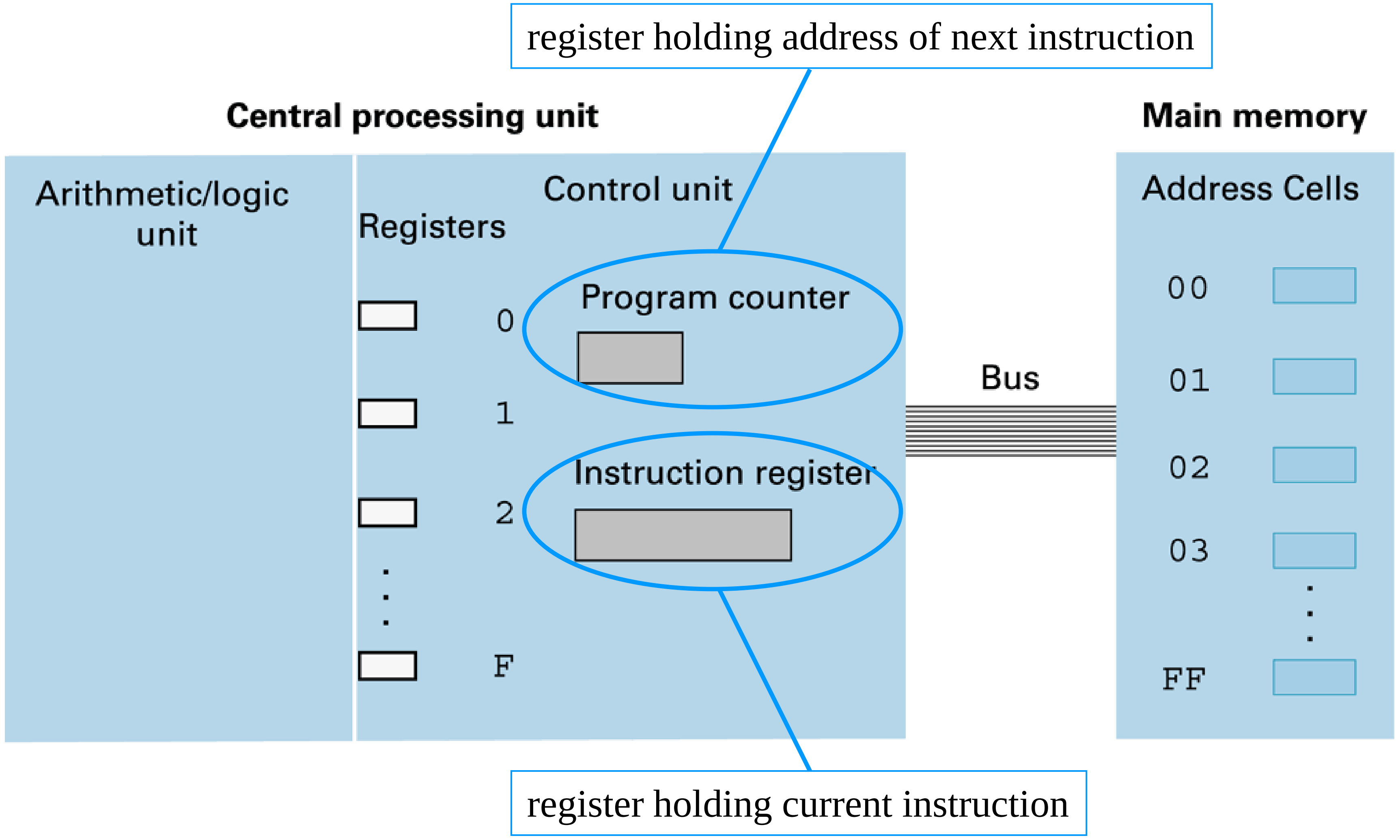
Step 6. STOP.

2.2: Composition of Machine Instructions

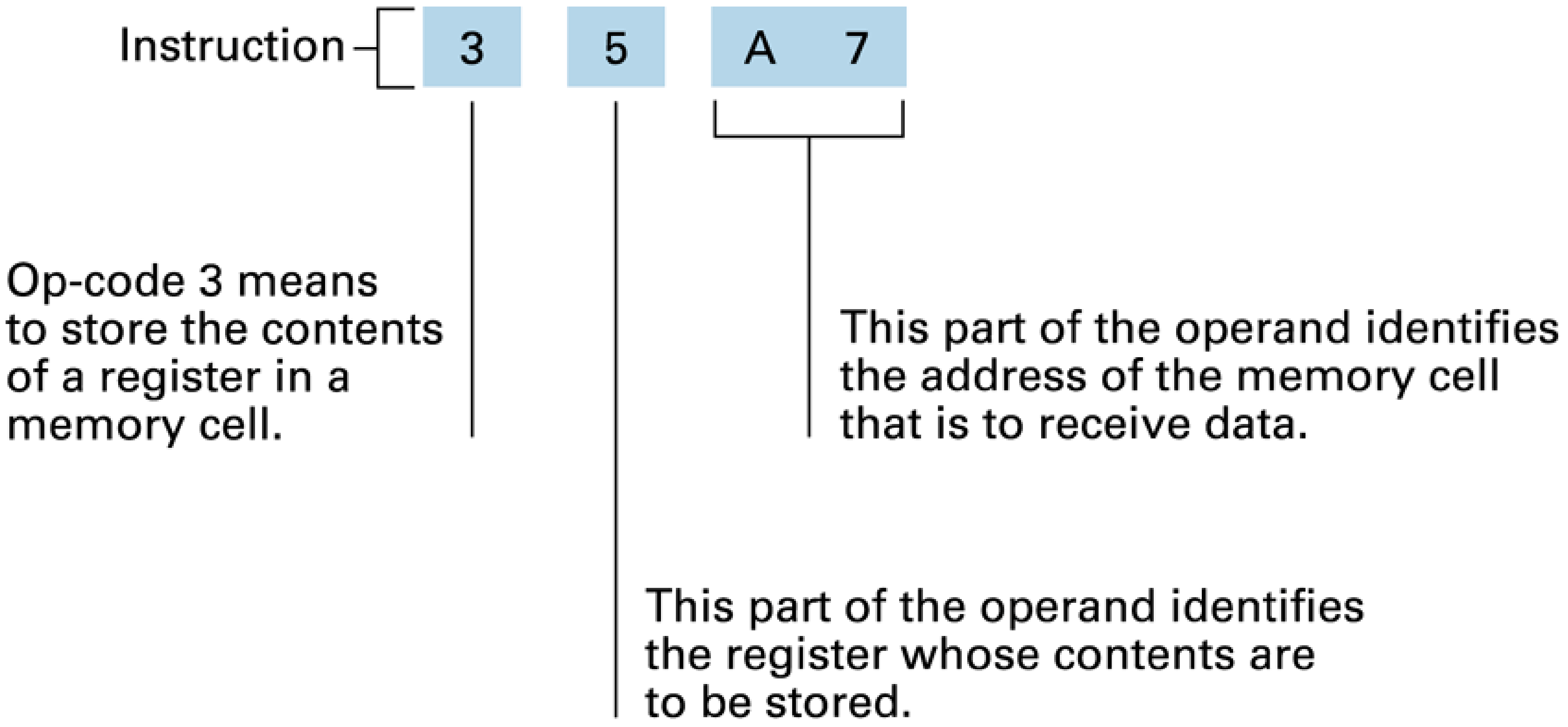
- Machine instructions typically consist of 2 parts:
  - (1) Op-code (operation code)
    - indicates which operation (such as STORE, SHIFT, XOR) is requested by the instruction
  - (2) Operand
    - provides more information about the operation specified by the op-code (e.g. registers/memory cells to be used)



2.2: Simple Machine Architecture (Appendix C)



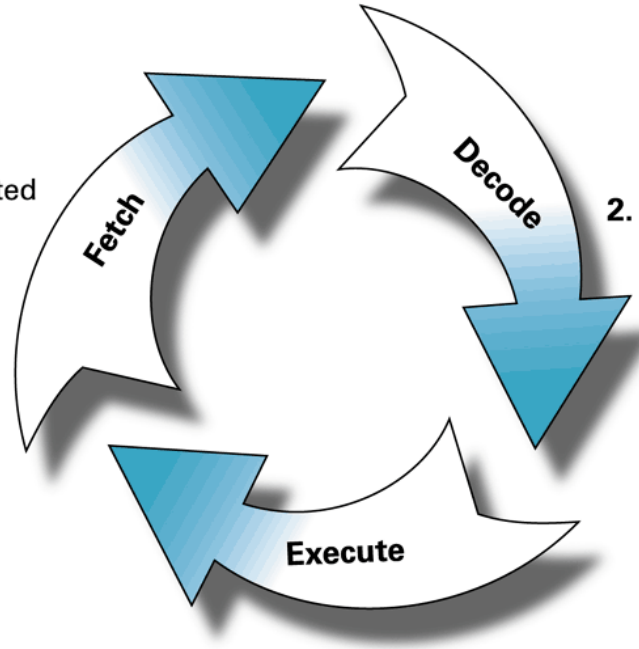
2.2: Example: decoding instruction 35A7 (App. C)


Instruction
3
5
A 7
Op-code 3 means to store the contents of a register in a memory cell.
This part of the operand identifies the address of the memory cell that is to receive data.
This part of the operand identifies the register whose contents are to be stored.

2.2: Adding two values stored in main memory

Encoded instructions	Translation
156C	Load register 5 with the bit pattern found in the memory cell at address 6C.
166D	Load register 6 with the bit pattern found in the memory cell at address 6D.
5056	Add the contents of register 5 and 6 as though they were two's complement representation and leave the result in register 0.
306E	Store the contents of register 0 in the memory cell at address 6E.
C000	Halt.

2.3: The Machine Cycle



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

(and store the instruction in the instruction register)

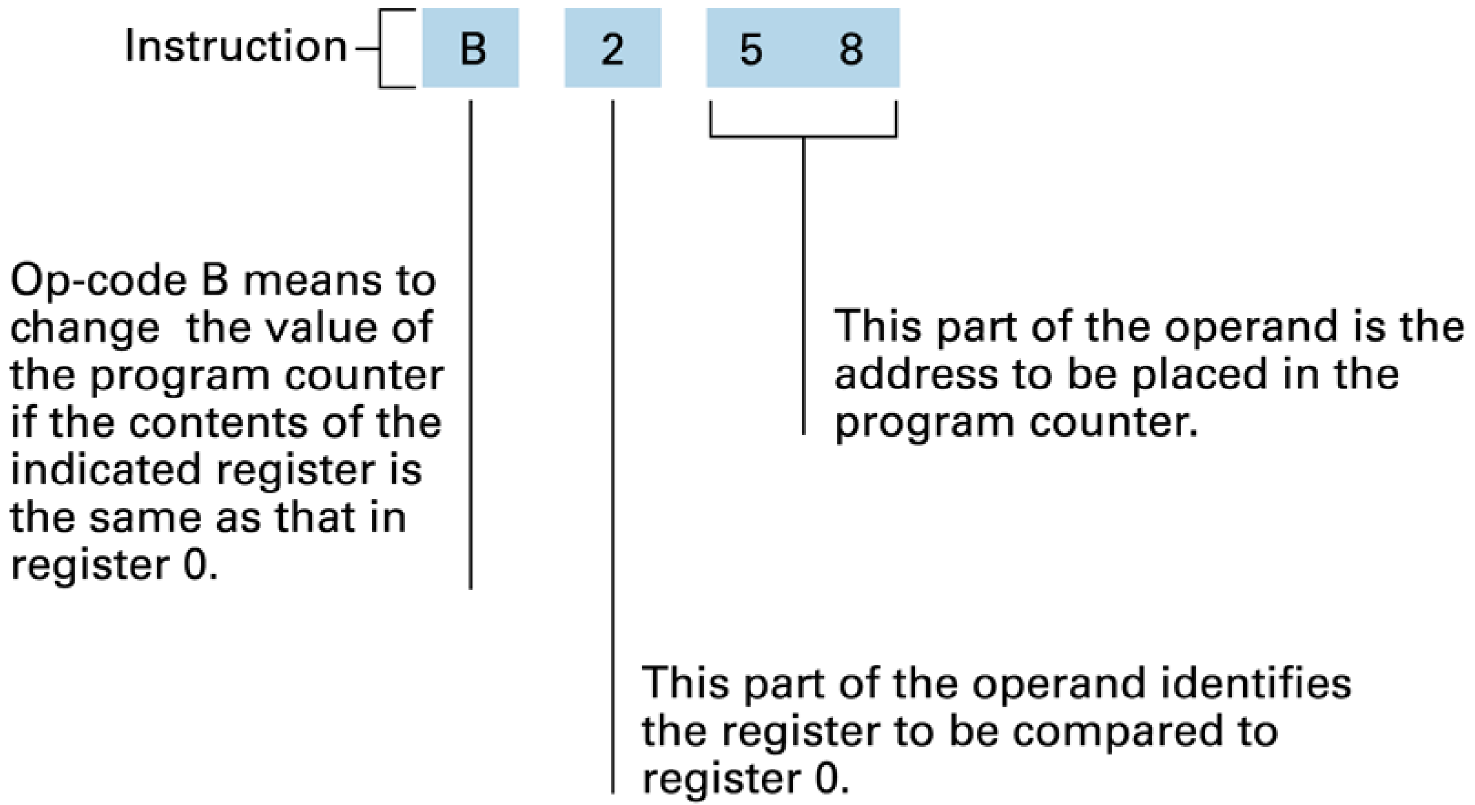
2. Decode the bit pattern in the instruction register.

(i.e.: break the operand field into its proper components based on the instruction's op-code)

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

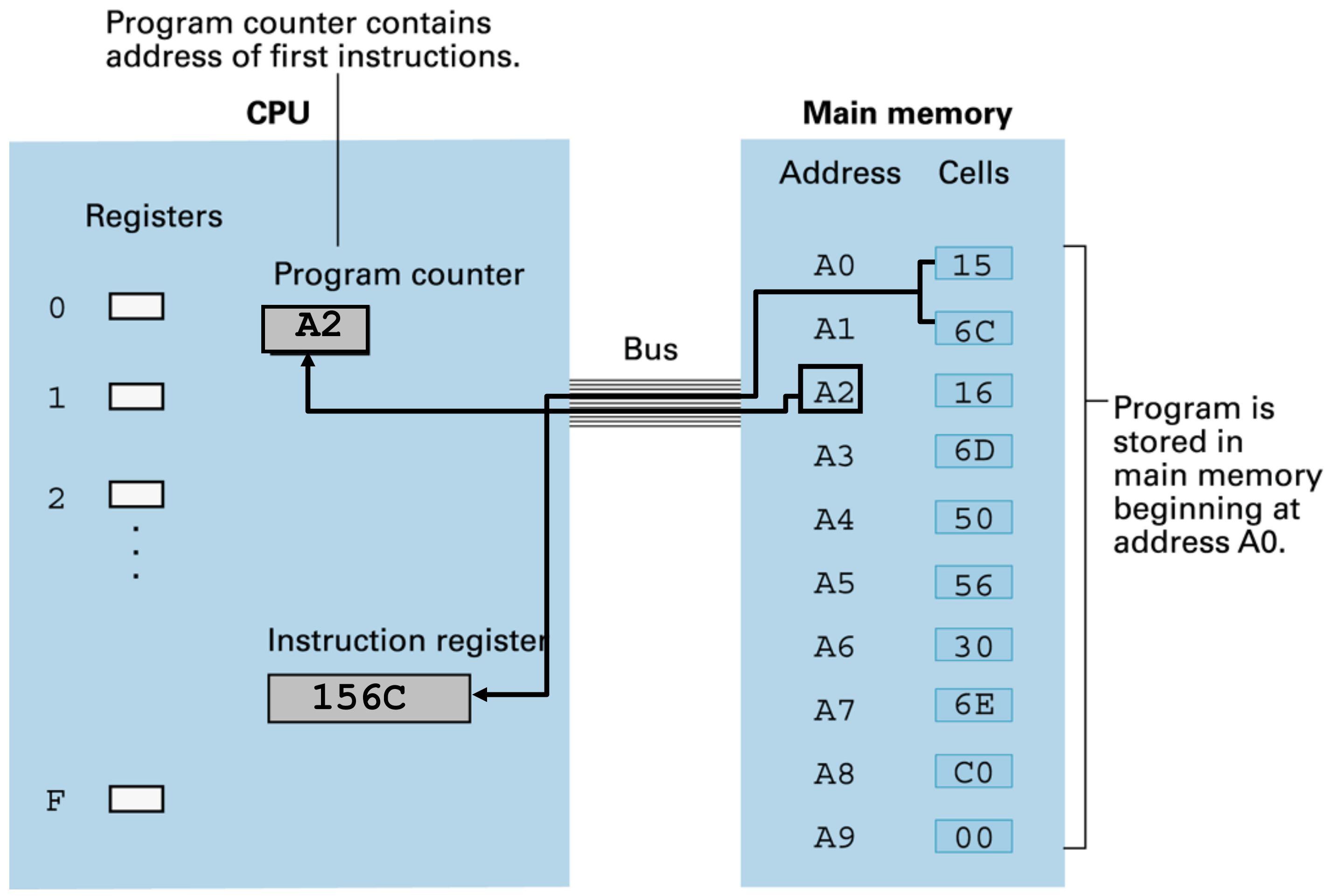
- Continually repeated by Control Unit until HALT
- Special case: JUMP (i.e: may change program counter)

2.3: Decoding JUMP instruction B258


Instruction
B
2
5 8
Op-code B means to change the value of the program counter if the contents of the indicated register is the same as that in register 0.
This part of the operand is the address to be placed in the program counter.
This part of the operand identifies the register to be compared to register 0.

- Translates as follows:
  - If contents registers 0 & 2 equal: place 58 in program counter
  - Otherwise: do nothing

2.3: Example of Program Execution (Fetch)



2.4: Arithmetic/Logic Instructions

- Boolean operations:

```
        10011010              10011010              10011010
AND     11001001      OR      11001001      XOR     11001001
---------------      ---------------       ---------------
        10001000              11011011              01010011
```

- AND used to force 0's into certain positions:

```
        00001111
AND     10101010
---------------
        00001010
```

- OR used to force 1's into certain positions:

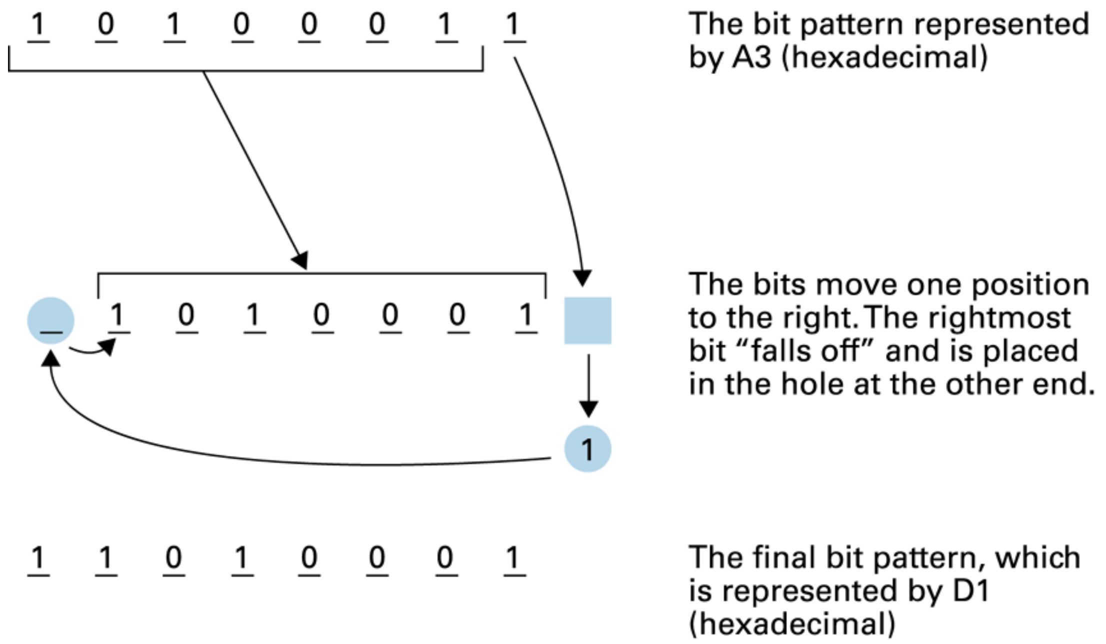
```
        00001111
OR      10101010
---------------
        10101111
```

- XOR used to form bit-string complement:

```
        11111111
XOR     10101010
---------------
        01010101
```

2.4: Arithmetic/Logic Instructions (cont'd)

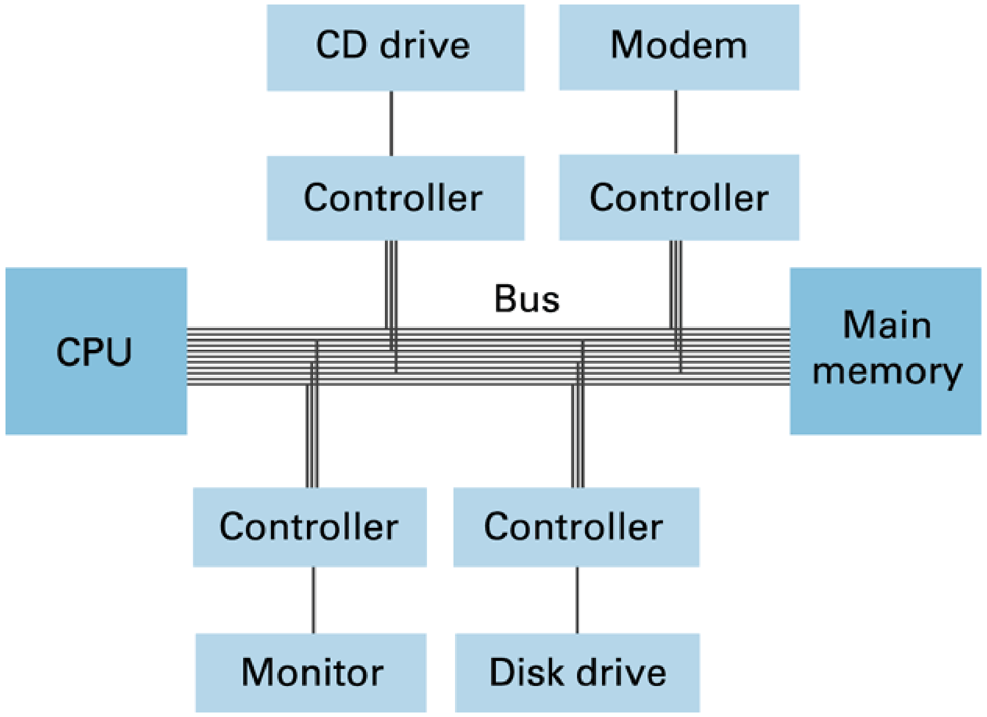
- Rotation and Shift:
  - Move bits within a register to left or right
  - e.g. for obtaining the mantissa in floating-point values
- Example:



1 0 1 0 0 0 1 1
The bit pattern represented by A3 (hexadecimal)
1 0 1 0 0 0 1
1
The bits move one position to the right. The rightmost bit "falls off" and is placed in the hole at the other end.
1 1 0 1 0 0 0 1
The final bit pattern, which is represented by D1 (hexadecimal)

2.5: Communication with other devices

- Handled through intermediary device: ‘*controller*’
- CPU communicates with controllers in the same way that it communicates with main memory



2.6: ‘Von Neumann Architecture’- Problem

- A problem of speed…?…!
  - electric pulses can go no faster than speed of light (approx. 30 cm per nanosecond)
- For average machine:
  - fetch-decode-execute cycle takes several nano secs.
- So: increasing speed becomes problem of scale
  - small, smaller, … stuck
- In other words: there appears to be a speed limit!

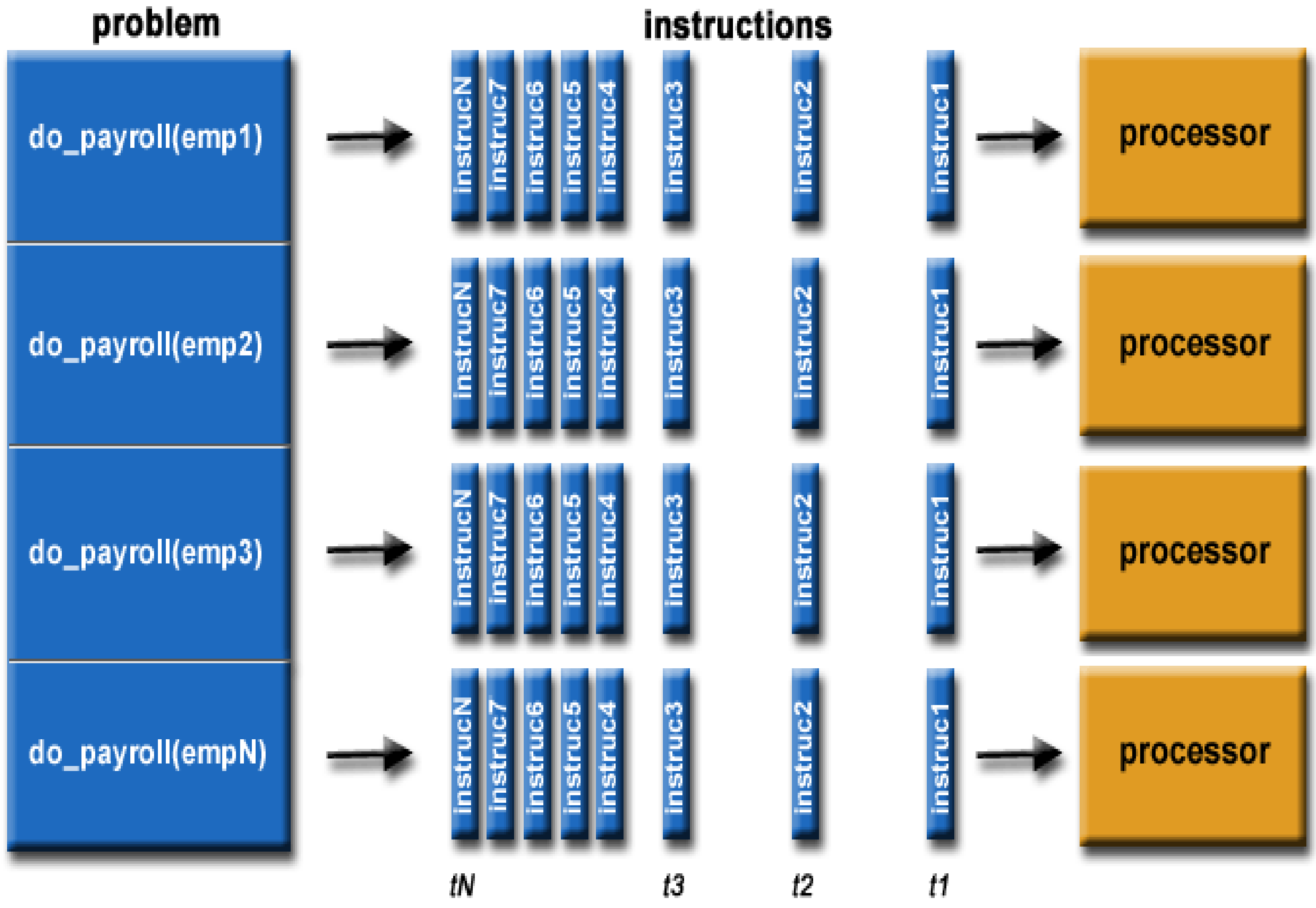
2.6: ‘Von Neumann Architecture’- Alternatives (1)

- One solution: Pipelining
  - allows steps in the machine cycle to overlap
  - during execution of one instruction, next is fetched
  - so: more than one instruction is ‘in the pipe’
- Result: same speed - but higher *throughput*
- Problems with JUMP instructions, of course…
  - pre-fetching becomes useless
- Modern CPUs:
  - fetch & execute multiple instructions at a time

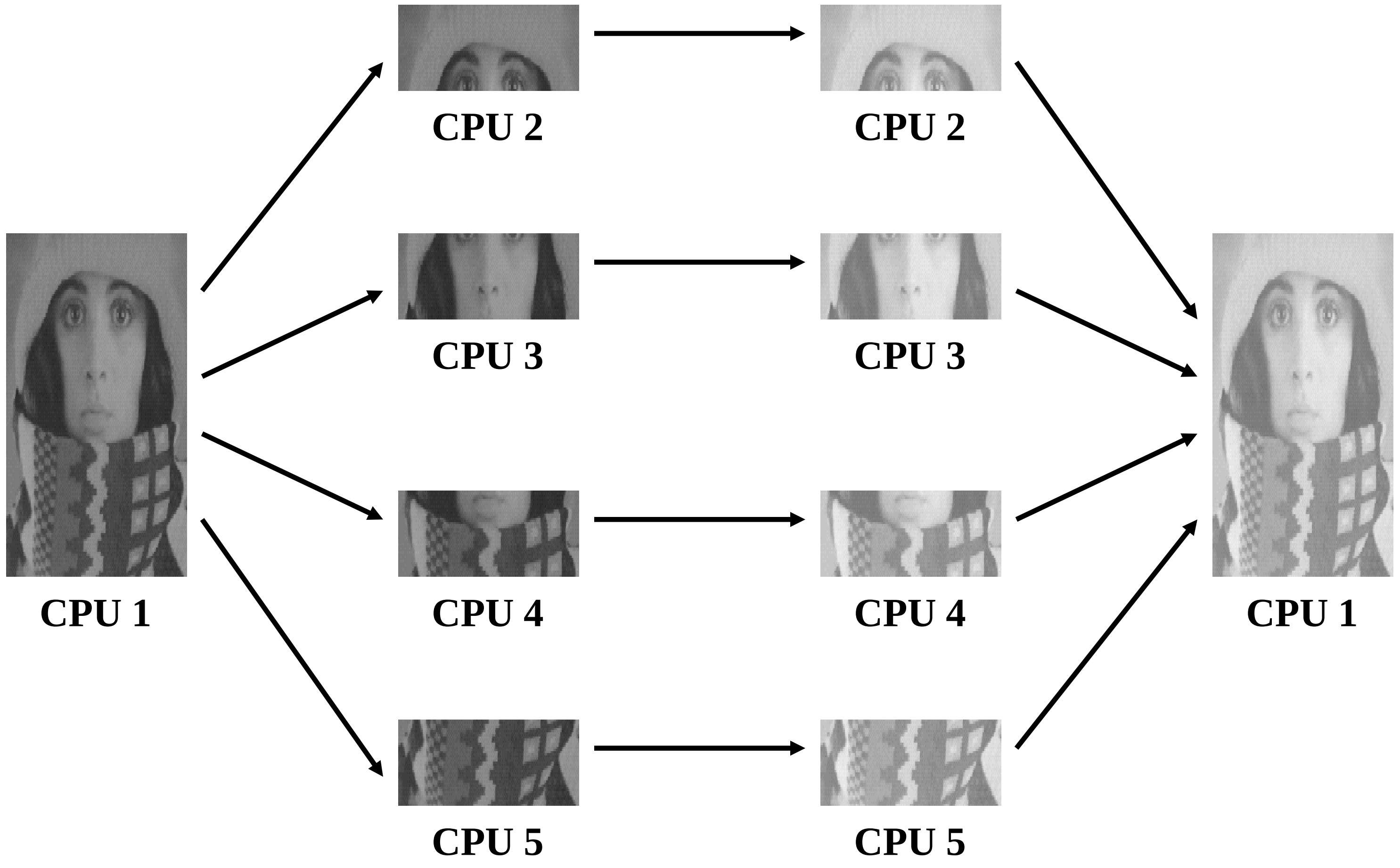
2.6: ‘Von Neumann Architecture’- Alternatives (2)

- Other solution: (true) Parallel Processing
  - performing several tasks at the same time
  - needs multiple processing units
- For example:
  - several complete PCs connected via fast network
    - each PC capable of independent execution
  - …

2.6: Parallel Processing: Example



- Parallel Image Processing:



Chapter 2 - Data Manipulation: Conclusions

- Today’s computers based on ‘Von Neumann’ Architecture
  - CPU connected to main memory via bus
- Both program & data stored in main memory
- Program execution based on Machine Cycle
  - fetch - decode - execute
- Peripheral devices connected via controllers
- Alternatives to ‘Von Neumann’ Architecture:
  - based on pipelining / true parallel processing

Quiz No.1 and Mid-term

- Course chapters 0, 1 and 2
- Topics / slides which covered in theory