Data Manipulation 2

Sabah Sayed

Topics

1. Machine Language.
2. Machine Instruction Types.
3. Decoding a Machine Instruction.
4. Program Execution.
5. Machine Cycle.
6. Decoding the JUMP Machine Instruction.

Chapter 2: Data Manipulation

- 2.1 Computer Architecture
- 2.2 Machine Language
- 2.3 Program Execution

Machine Language Terminology

- **Machine instruction: An instruction** (or command) encoded as a **bit pattern recognizable by** the **CPU**

- **Machine language / instruction set:** The **set of all instructions recognized by** a **machine**

Machine (M/C) Instruction Types

1. Data Transfer: **copy** data from one **location** to another
   - e.g.,
     - **Load** instruction: copy data from a memory cell to a CPU register
     - **Store** instruction: copy data from a CPU register to a memory cell
2. Arithmetic/Logic: use existing bit patterns to **compute** a new bit pattern
   - e.g.,
     - **Arithmetic**: Add, Sub, Mult, Div, Mod instruction
     - **Logical**: And, OR, XOR, NOT, SHIFT, ROTATE instruction
3. Control: **direct** the execution of the program
   - e.g.,
     - **Conditional Jump or unconditional** Jump instruction
     - Stop/Halt instruction

Identify the M/C Instruction Type

Figure 2.2 Adding values stored in memory

A=b+c

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.

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.

Step 4. Store the result in memory.

Step 5. Stop.

Data Transfer

Data Transfer

Arithmetic/Logical

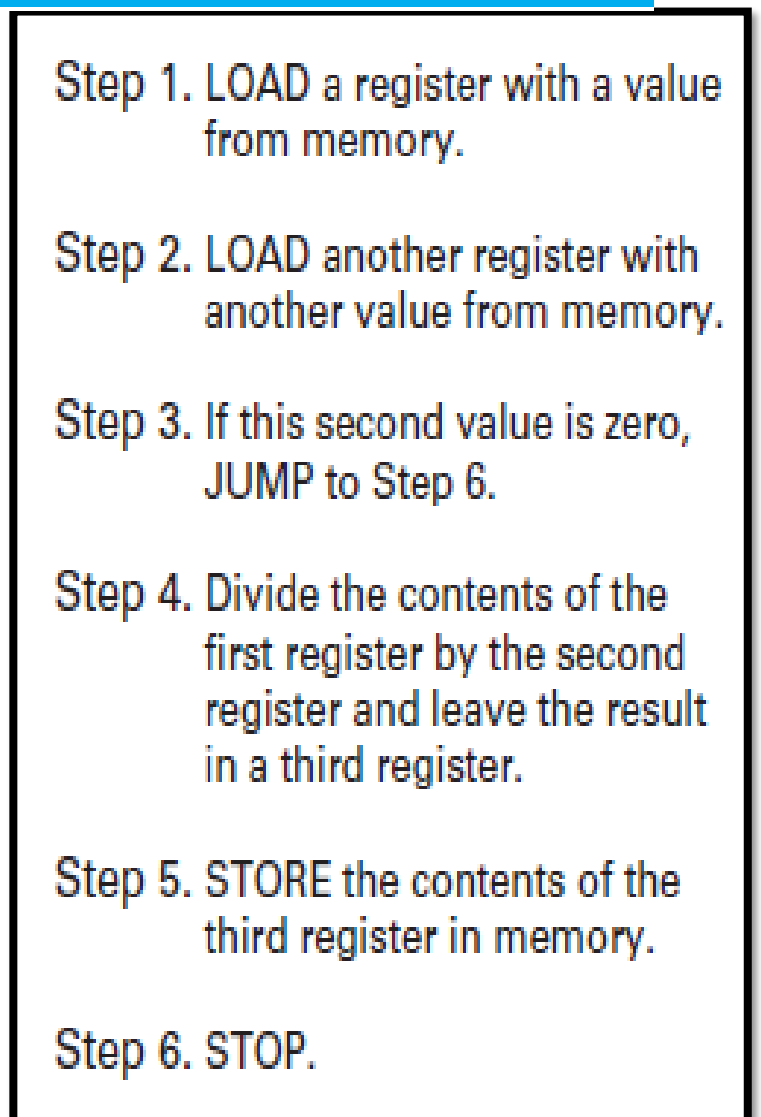
Data Transfer

Control

Identify the M/C Instruction Type

Figure 2.3 Dividing values stored in memory

A=b/c



Data Transfer

Data Transfer

Control

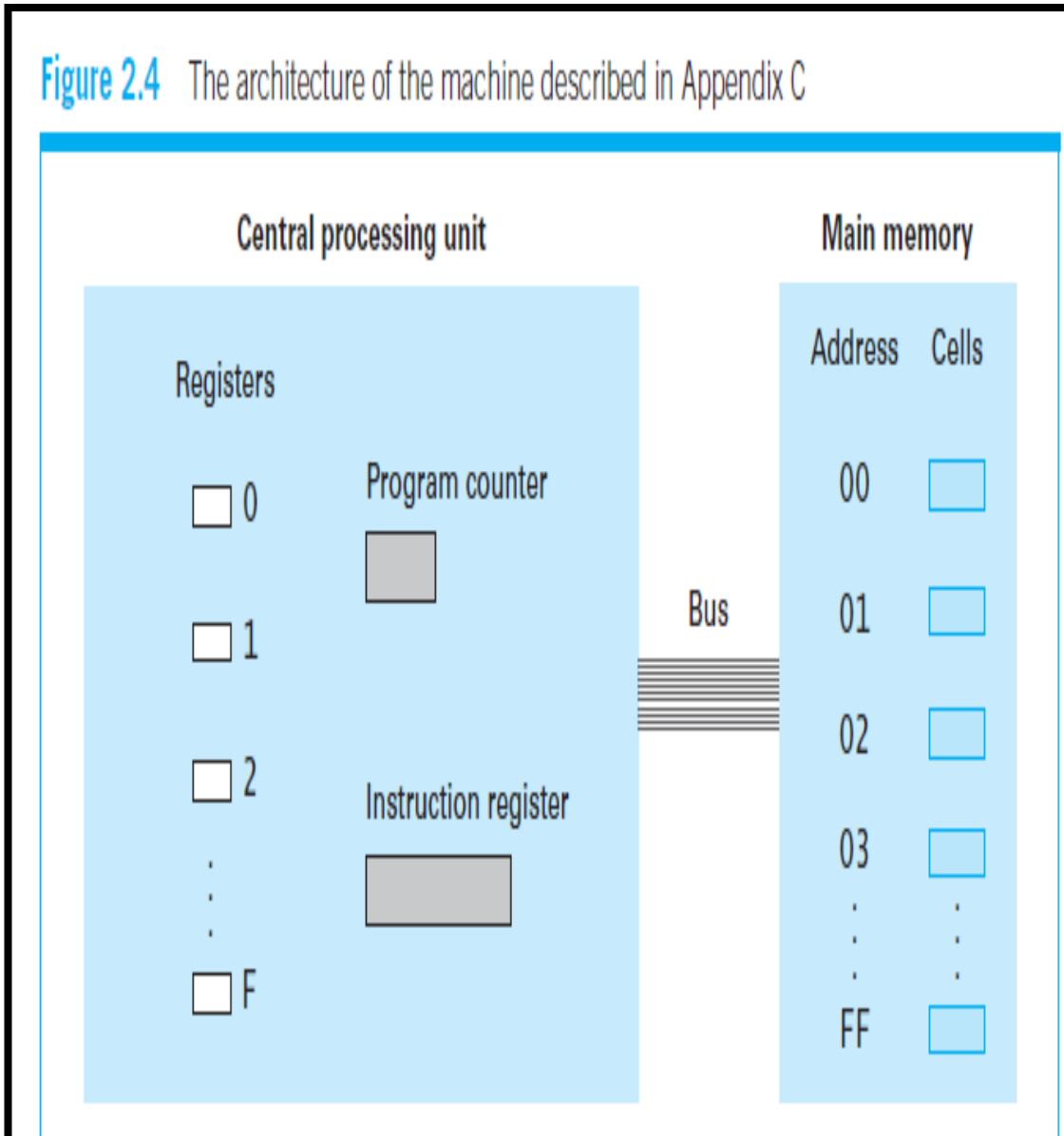
Arithmetic/Logical

Data Transfer

Control

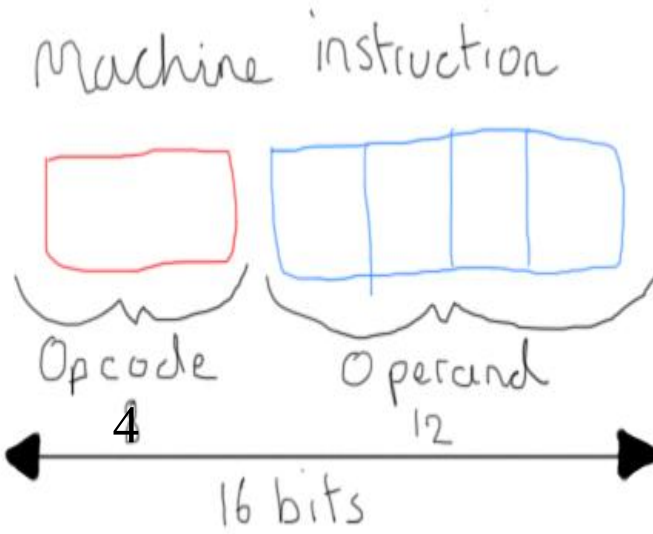
Simple M/C Architecture

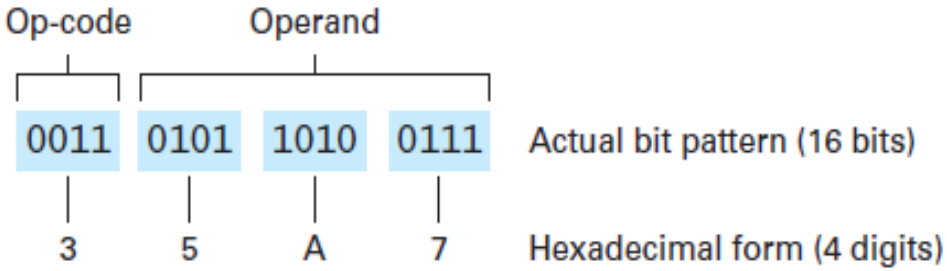
- This machine contains:
  - 16 general-purpose registers (from 0→F) and each register can store 1 byte
  - 256 memory cell addressed from (00→FF) (i.e., 0→255)

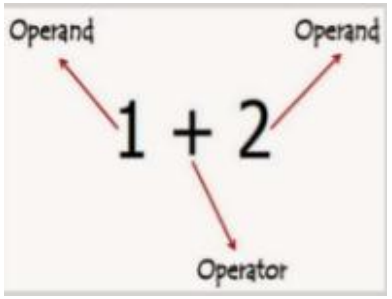


Machine Instruction is composed of 2 Main Parts

1. **Op-code (Operation code)**: Specifies which operation is requested by the instruction
   - Consists of 4 bits (i.e., 1 hex digit)
   - e.g., ADD, LOAD, STORE, XOR…
2. **Operand**: Specifies which data needed to execute the operation
   - Consists of 12 bits (i.e., 3 hex digit)
   - Interpretation of operand varies depending on op-code

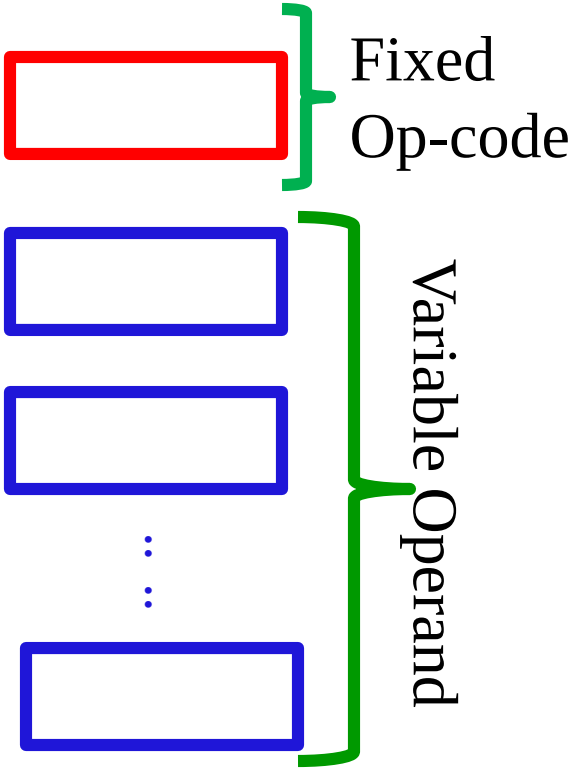




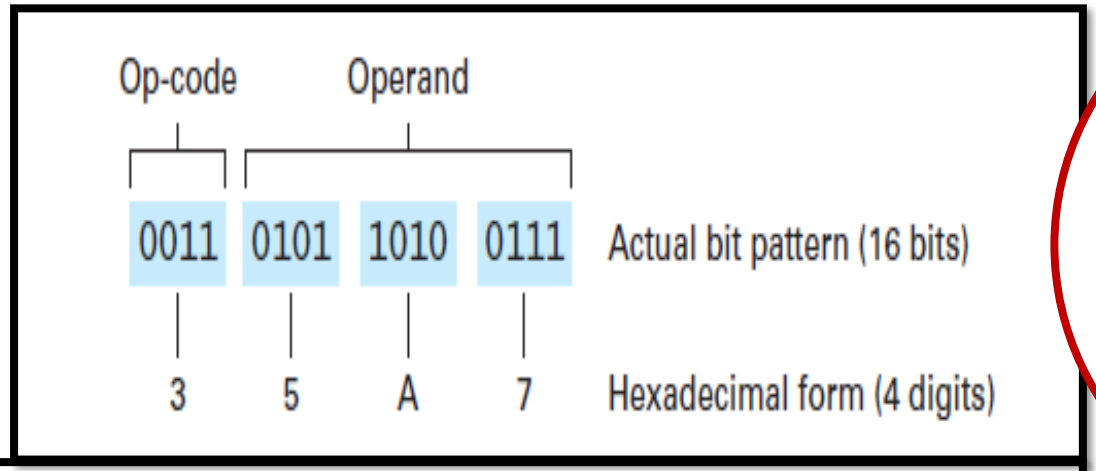


Variable length Machine Instruction

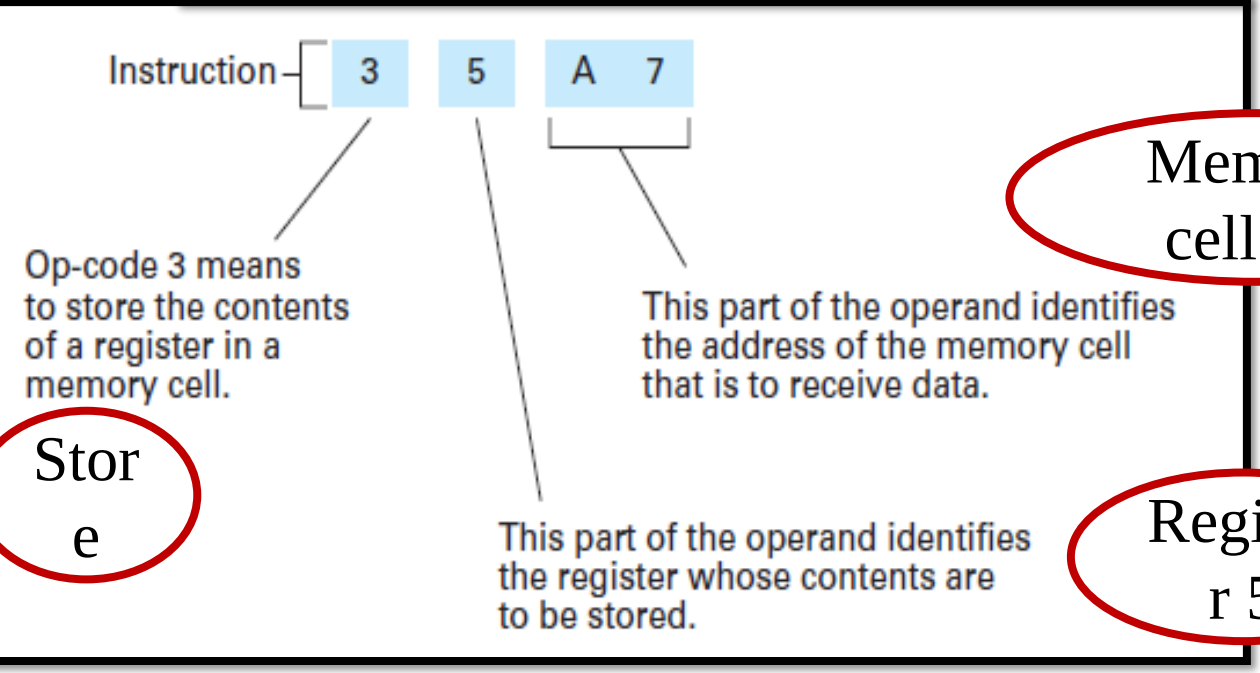
- Some machine languages whose instructions **vary in length**.
- Today's Intel processors, have instructions that range from one-byte to multiple-byte instructions.
- Length depends on the exact use of the instruction.
  1. CPU fetches the op-code of the instruction
  2. Then, based on the bit pattern received, fetches other parts from memory to obtain the rest of the instruction.



Decoding a Machine Instruction (STORE)



Op-code = 3, operand =5A7 means STORE the bit pattern found in register 5 in the memory cell whose address is A7

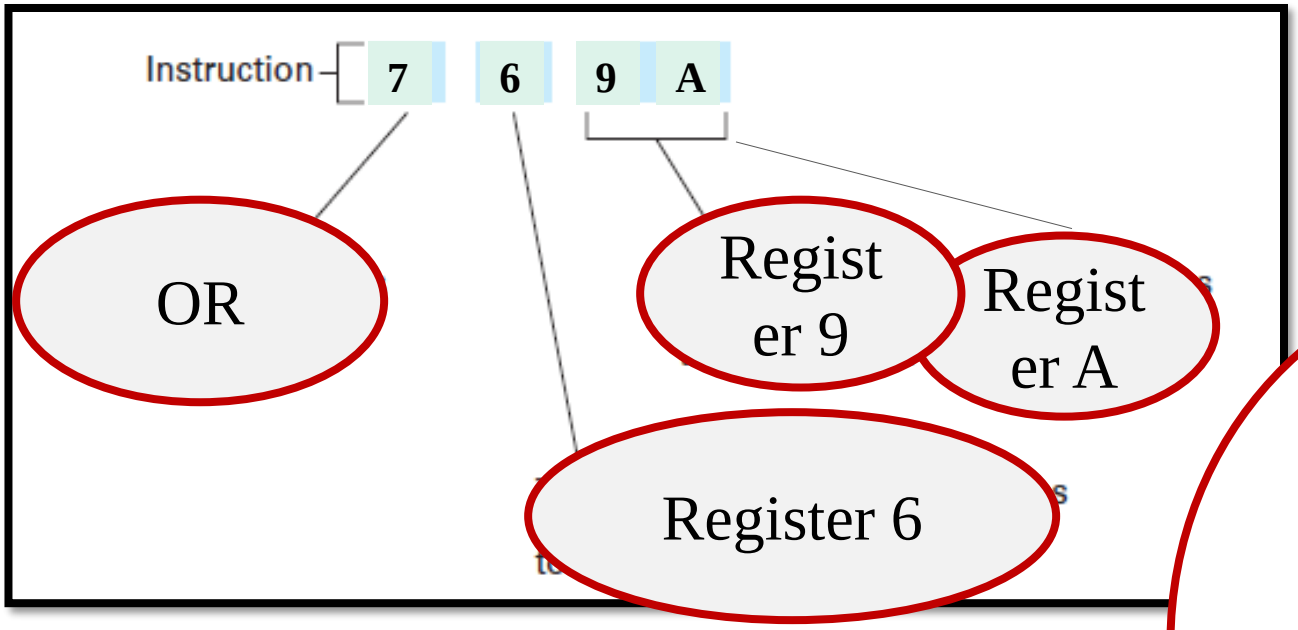


Memory cell A7

Stor e

Registe r 5

Decoding a Machine Instruction (OR)



Op-code = 7, operand =69A means ORing the bit patterns found in registers 9 and A and place the result in register 6

Check this table in Appendix c

Machine Language

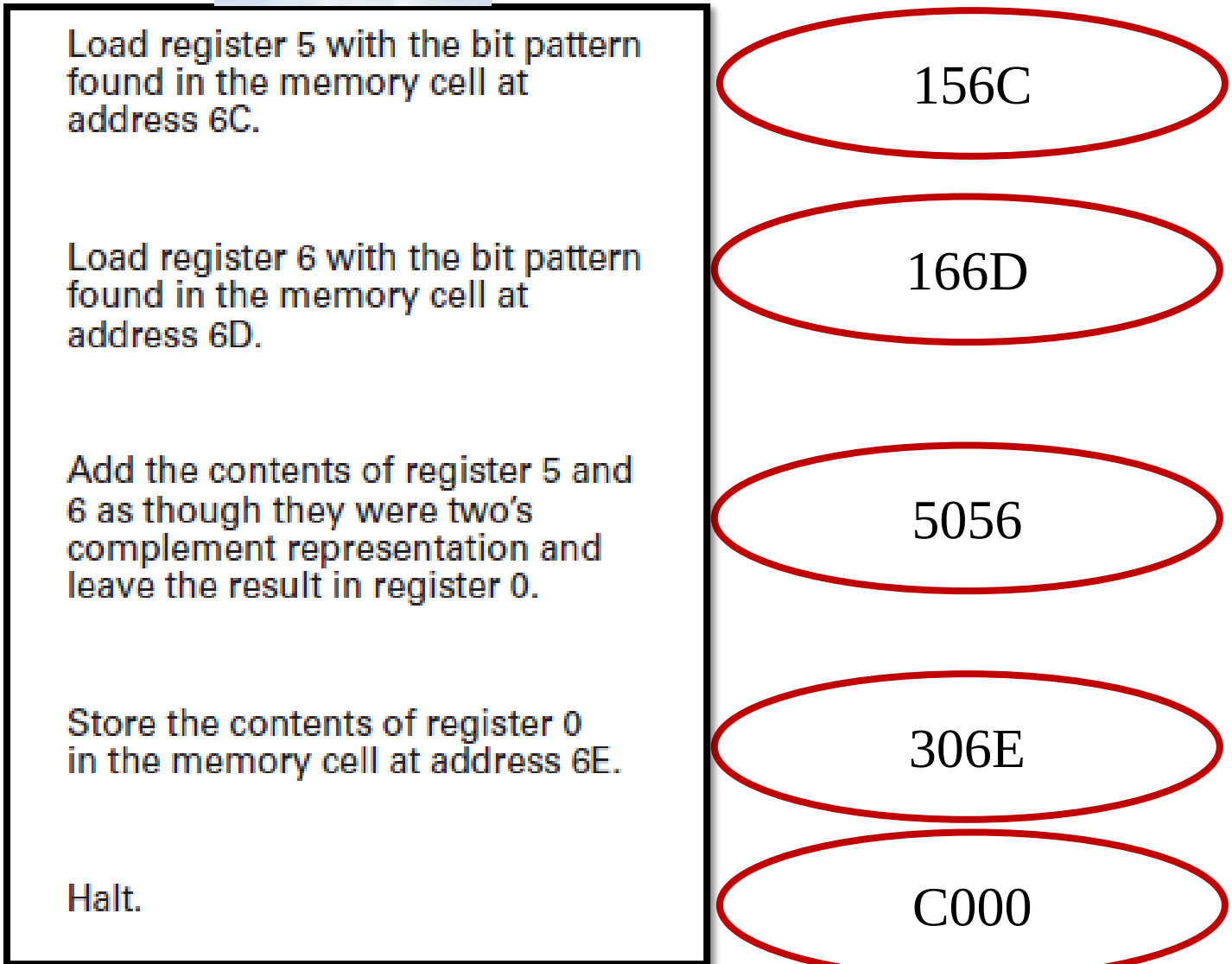
Op-code	Operand	Description
1	RXY	LOAD the register R with the bit pattern found in the memory cell whose address is XY. *Example:* 14A3 would cause the contents of the memory cell located at address A3 to be placed in register 4.
2	RXY	LOAD the register R with the bit pattern XY. *Example:* 20A3 would cause the value A3 to be placed in register 0.
3	RXY	STORE the bit pattern found in register R in the memory cell whose address is XY. *Example:* 35B1 would cause the contents of register 5 to be placed in the memory cell whose address is B1.
3	R00	**STORE to location 00, which is a memory mapping for the screen. Writing to 00 is writing to screen.**
4	0RS	MOVE the bit pattern found in register R to register S. *Example:* 40A4 would cause the contents of register A to be copied into register 4.
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. *Example:* 5726 would cause the binary values in registers 2 and 6 to be added and the sum placed in register 7.
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. *Example:* 634E would cause the values in registers 4 and E to be added as floating-point values and the result to be placed in register 3.
7	RST	OR the bit patterns in registers S and T and place the result in register R. *Example:* 7CB4 would cause the result of ORing the contents of registers B and 4 to be placed in register C.
8	RST	AND the bit patterns in register S and T and place the result in register R. *Example:* 8045 would cause the result of ANDing the contents of registers 4 and 5 to be placed in register 0.
9	RST	EXCLUSIVE OR the bit patterns in registers S and T and place the result in register R. *Example:* 95F3 would cause the result of EXCLUSIVE ORing the contents of registers F and 3 to be placed in register 5.

Check this table in Appendix c

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. *Example:* A403 would cause the contents of register 4 to be rotated 3 bits to the right in a circular fashion.
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.) *Example:* B43C would first compare the contents of register 4 with the contents of register 0. If the two were equal, the pattern 3C would be placed in the program counter so that the next instruction executed would be the one located at that memory address. Otherwise, nothing would be done and program execution would continue in its normal sequence.
C	000	HALT execution. *Example:* C000 would cause program execution to stop.

Example: Encoding the following into Machine instructions (based on the previous table)

Figure 2.7



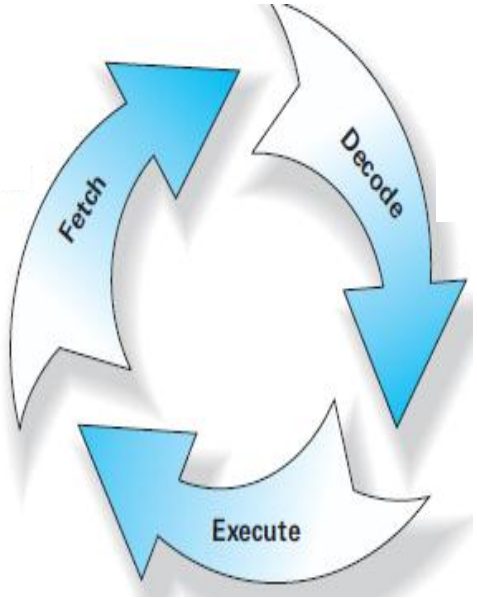
A=b+c

Chapter 2: Data Manipulation

- 2.1 Computer Architecture
- 2.2 Machine Language
- 2.3 Program Execution

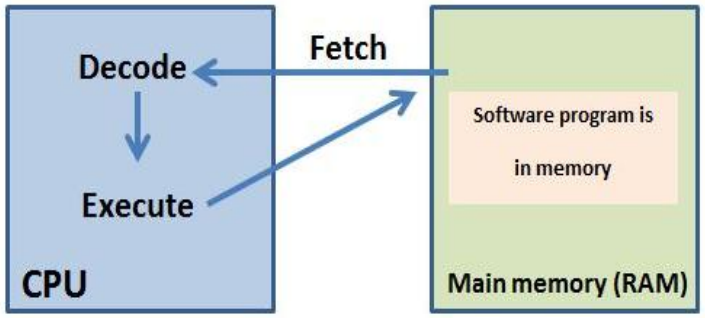
Program Execution

- Controlled by two special-purpose registers
  - Program counter (PC): store the address of next instruction
  - Instruction register (IS): store the current instruction
- Machine Cycle
  - **F**etch
  - **D**ecode
  - **E**xecute



Machine Cycle

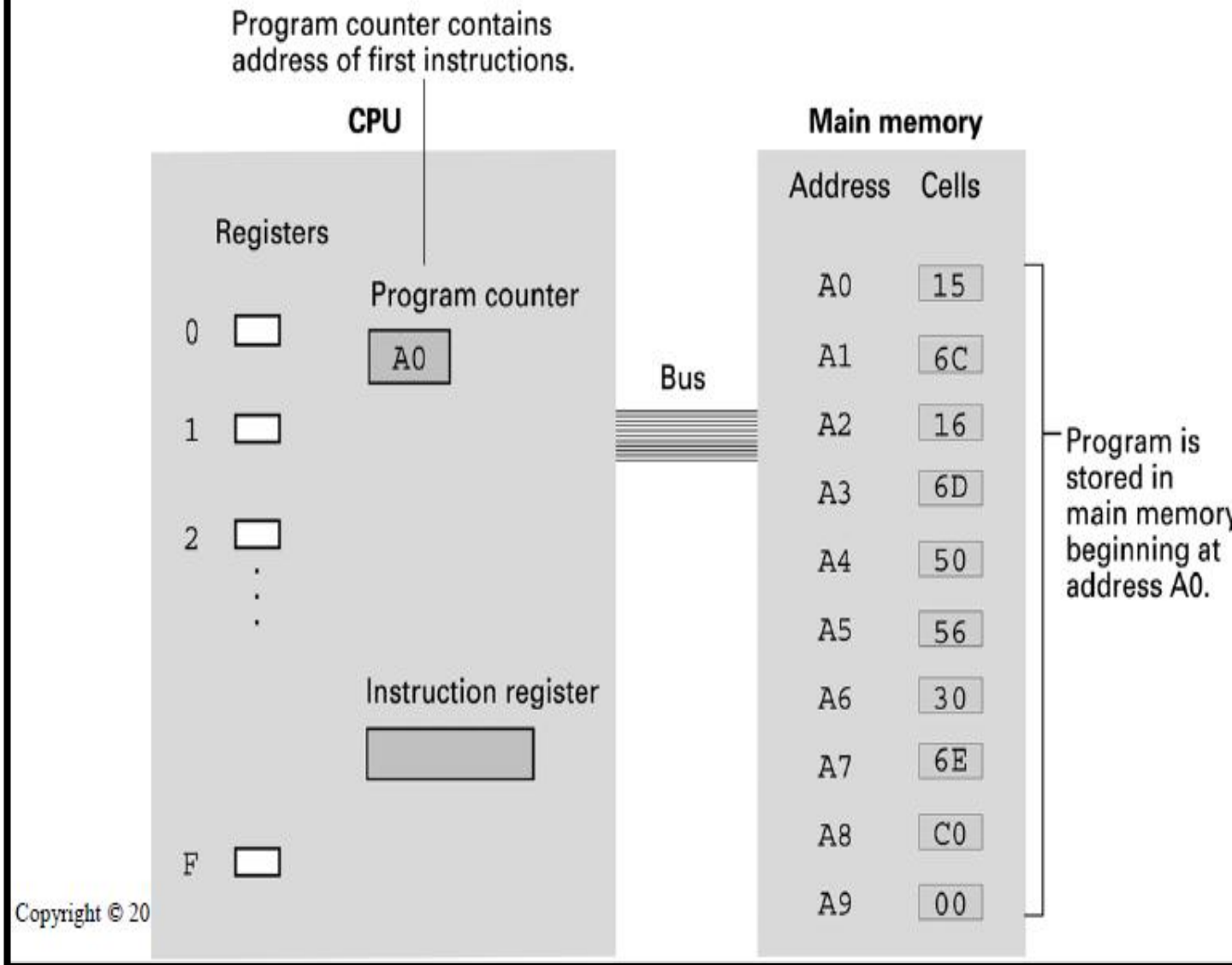
- **To execute a program**:
  - **CPU makes a Machine cycle which is composed of three main Steps (Fetch, Decode, Execute) :**
    1. **Fetch:** get an instruction from Main Memory that its address stored in the PC and store it in the IR, then increment the PC
    2. **Decode**: Decode the instruction, which exists in the IR, based on the instruction's op-code
    3. **Execute**: execute the action required by the instruction
  - Then **repeat** the machine cycle again …
  - The **order** for executing these steps is **very important**



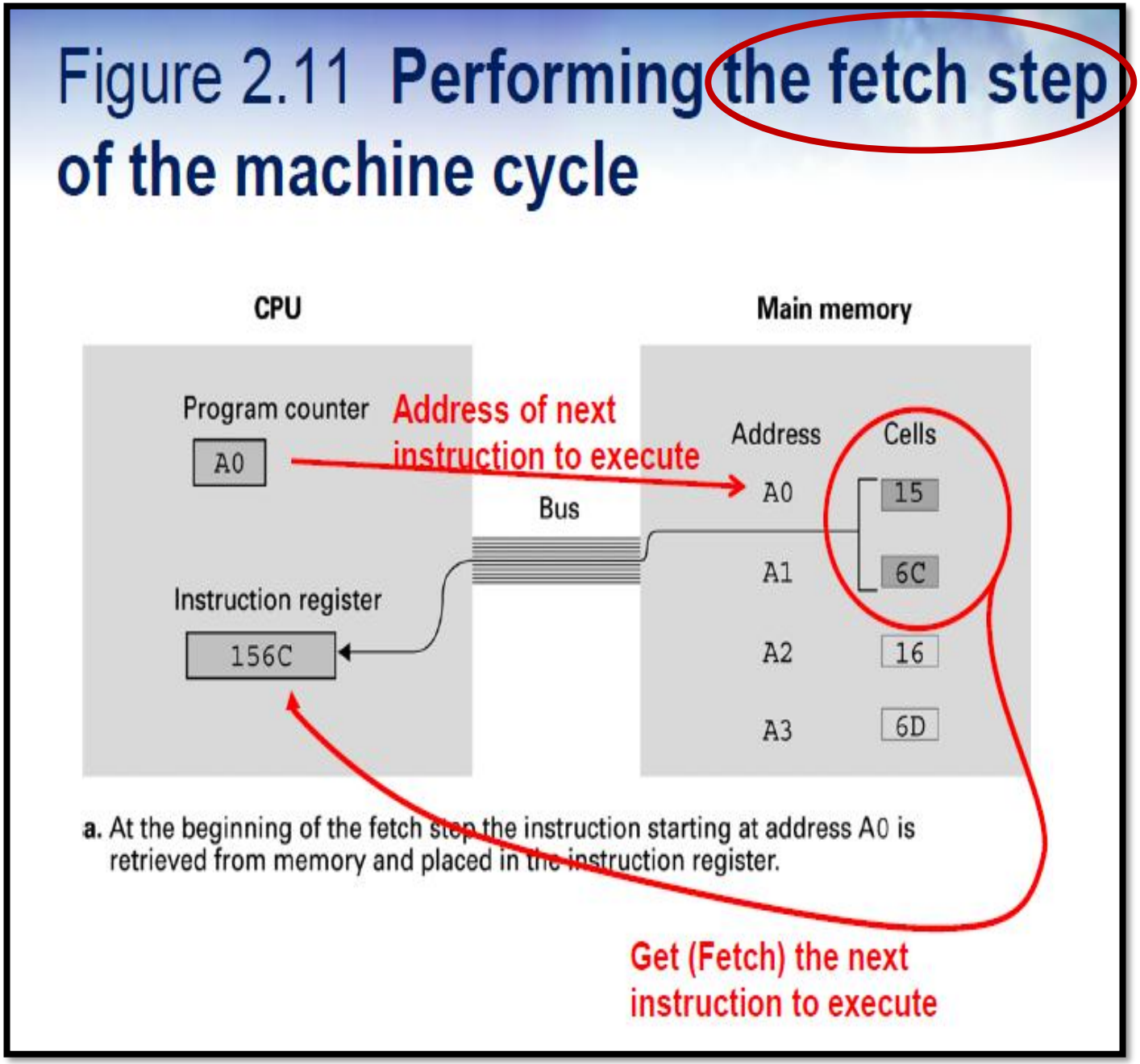
- An example of program execution

Note: each instruction composed of two memory cells

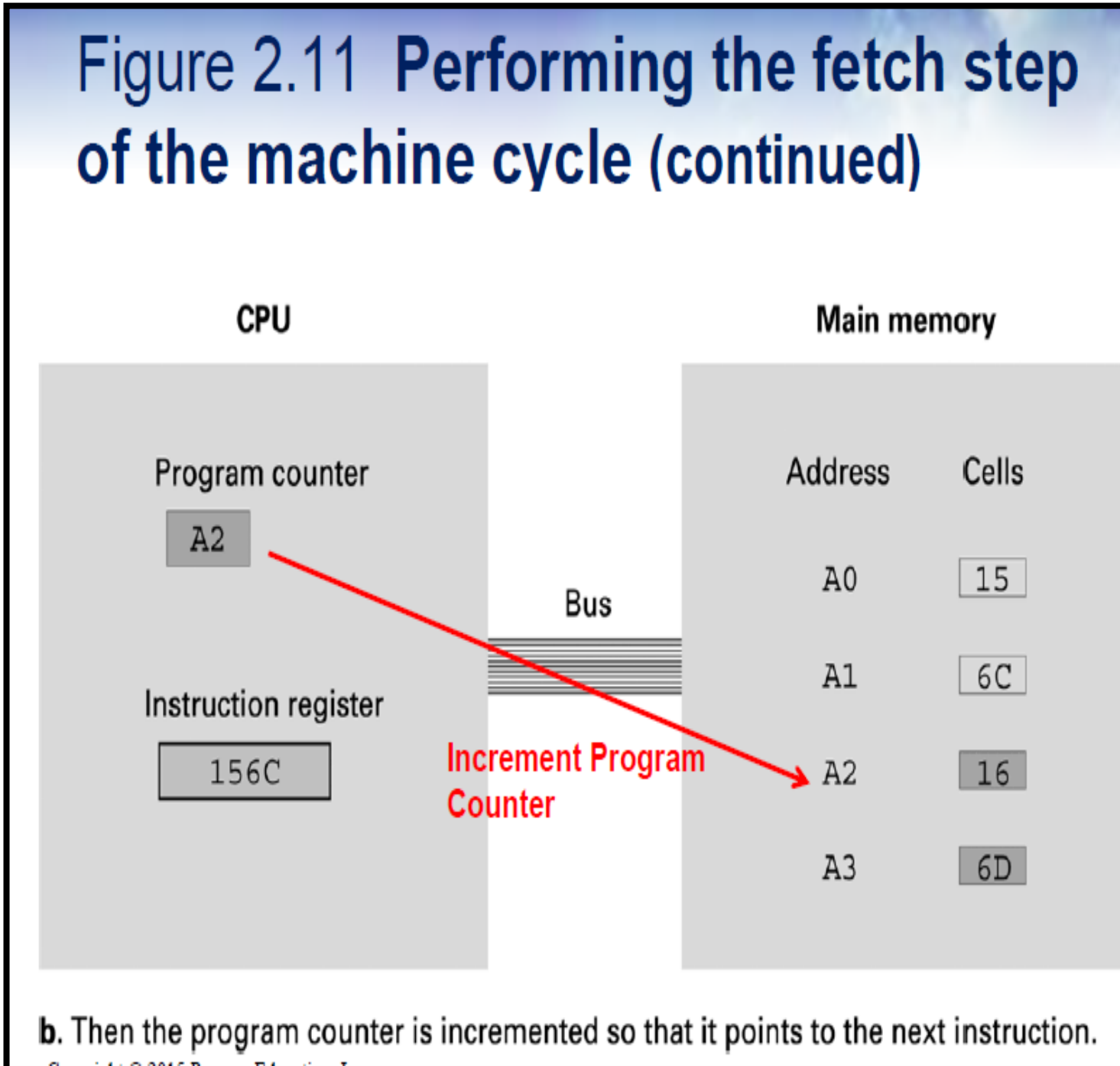
Figure 2.10 The program from Figure 2.7 stored in main memory ready for execution



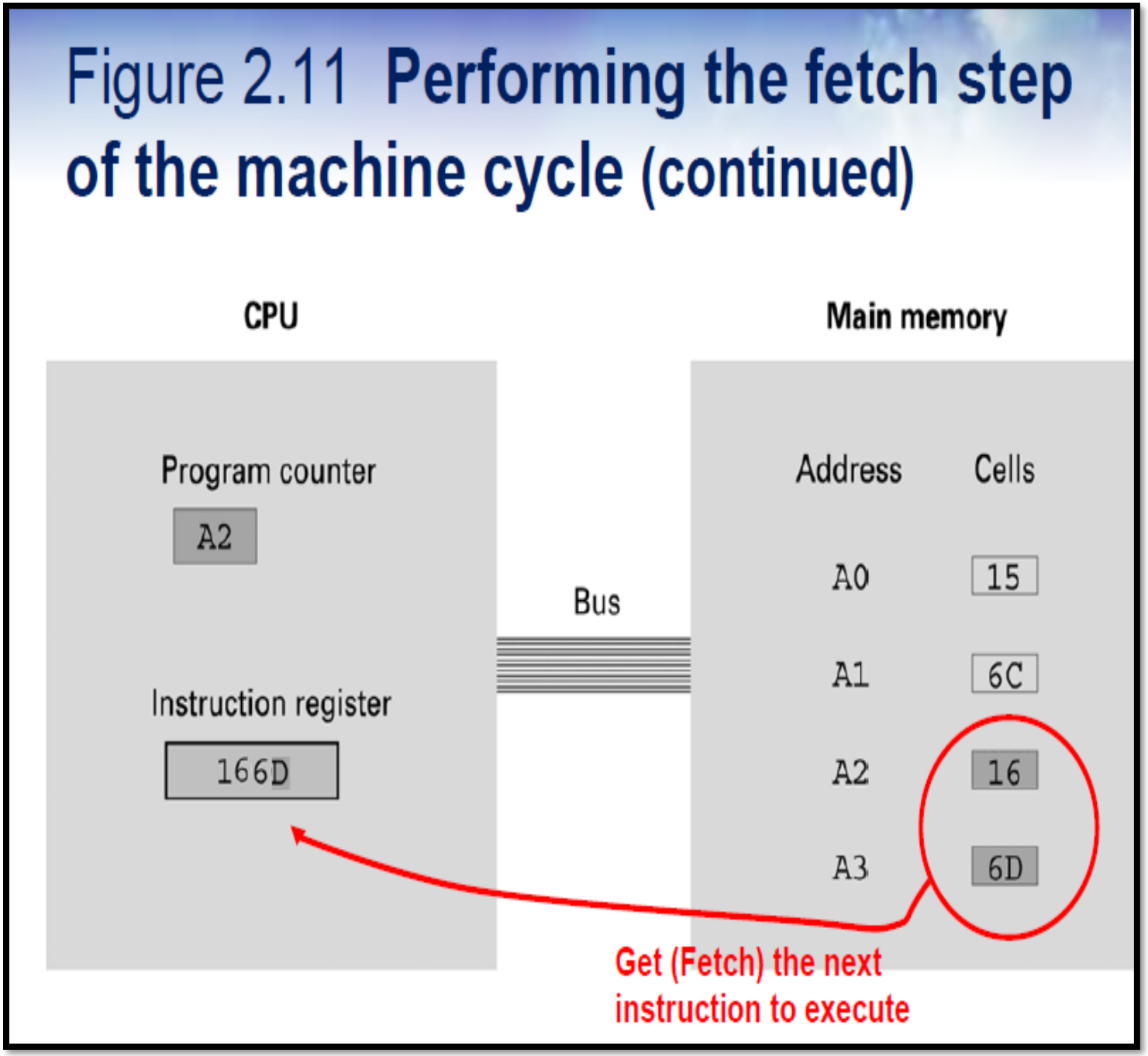
- An example of program execution



- An example of program execution

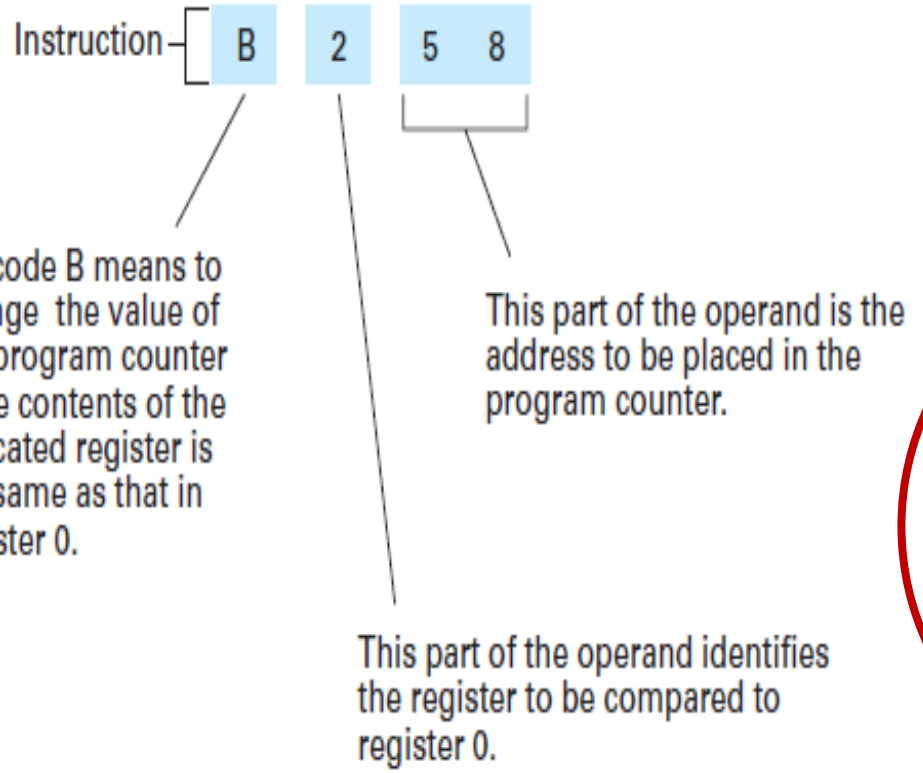


- An example of program execution



Decoding the JUMP Machine Instruction

Figure 2.9 Decoding the instruction B258



Note: JUMP instruction can change the address in the PC

Op-code = B, operand =258 means If R0==R2, put 58 in the PC register

Note: B0XY will cause a jump to be executed to the memory location XY regardless of the contents of register 0 → we can consider it as an example of unconditional JUMP

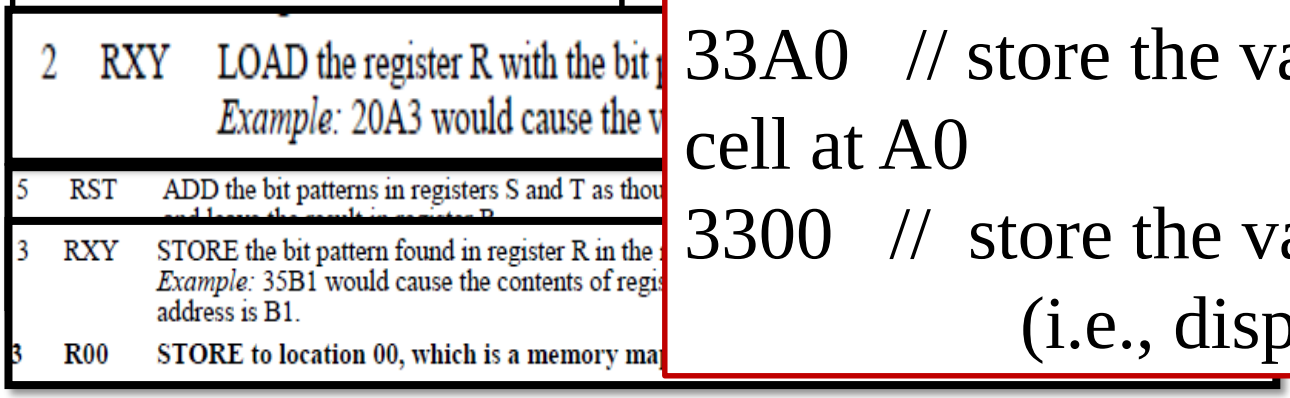
Example: Encode the following into m/c instructions

- Assume that we have three Registers (1,2,3) to store the variables x,y,z
- Finally, store the result in the memory cell at address A0
- Also, display the result on the screen

```
x = 5
y = -14
z = x + y
```

Solution

2105 // Load R1 with value 05
22F2 // Load R2 with value F2
5312 // Add R1 to R2 and place the resu
in R3
33A0 // store the value of R3 to memory
cell at A0
3300 // store the value of R3 to screen
(i.e., display result)



Example: Encode the following program into m/c instructions

- **Note:** each value used in the program, must be stored in a register before we using it
- So, we need to store values of x=0, 1 (increment), 5 (stopping condition) in registers
- **Note:** The value of the stopping condition should be stored in register 0, because of JUMP instruction

```
x = 0
while x<5:
    x = x + 1
print x
```

Solution

```
10      2005   //Load R0 with value 05
12      2100   // Load R1 with value 0 (i.e., x=0)
14      2201   // Load R2 with value 1 (i.e., increment)
16      B11C    // if R1==R0, jump memory cell at 1C (to print x)
18      5112   // Add R1 to R2 and store result in R1
1A      B016   //  Jump to memory cell at 16 (i.e., Go to the beginning
of the loop)
1C       3100   // store R1 to screen (i.e., print x)
1E      C000   // Halt
```

Example: Complete the following m/c Instructions

- The following machine language program that
  - reads x from memory cell A0 and y from memory cell A1.
  - If x = y, store 1 in cell A2 and if x != y, store 2 in cell A2.
  - Explain your steps beside each step as shown with the first two steps.

Address	Instruction	
10	10A0	// Load *x* in R0
12	11A1	// Load *y* in R1
14	B11A	// if R1=R0, go to instruction 1A
16	2202	// Load R2 with value 02 (to be used later)
18	B01C	// Go to instruction IC to store value of R2 in A2
1A	2201	//Load R2 with value 01
1C	32A2	//store the value of R2 in memory cell A2
1E	C000	// Halt

H/W: Decoding m/c instructions

- The following are instructions written in the machine language described in Appendix C. Rewrite them in English.

1. 368A
2. BADE
3. 803C
4. 40F4

- Solve Question 6 & 7 page 103, in the text book