CMSC 11:
Introduction to Computer Science

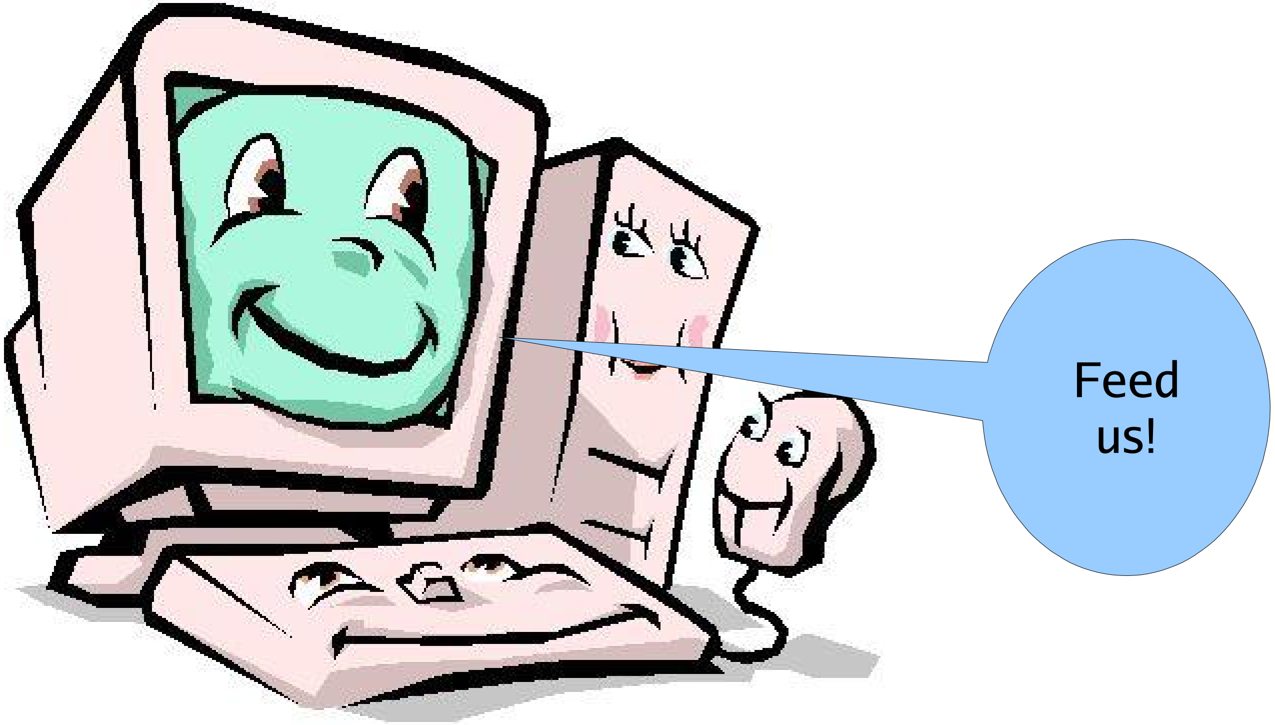
Jaderick P. Pabico <jppabico@uplb.edu.ph>
Institute of Computer Science, CAS, UPLB

Review

- Control
- Assembly language

Controlling a program

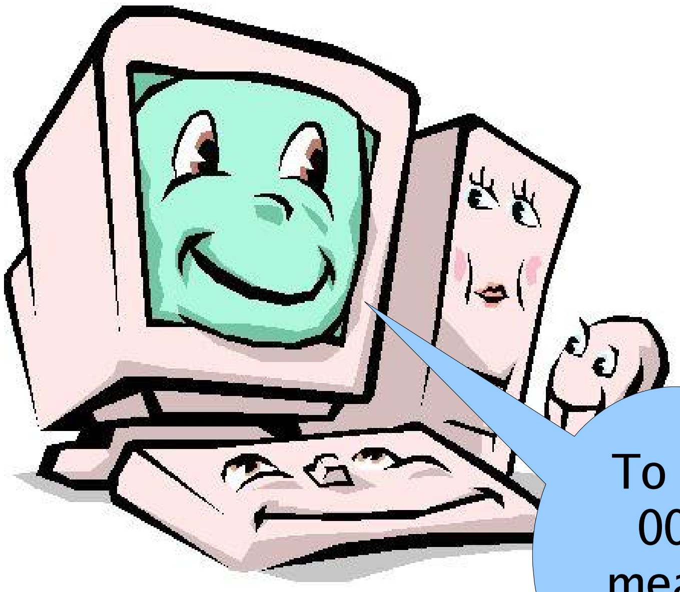
- Now that we have an assembly language program, how do we feed it to the machine – which only understands 0's and 1's?



Controlling a program

- The answer is clear: within the machine, each operator is encoded as a string of bits called its “OP-CODE”.
- Here is some simple samples:

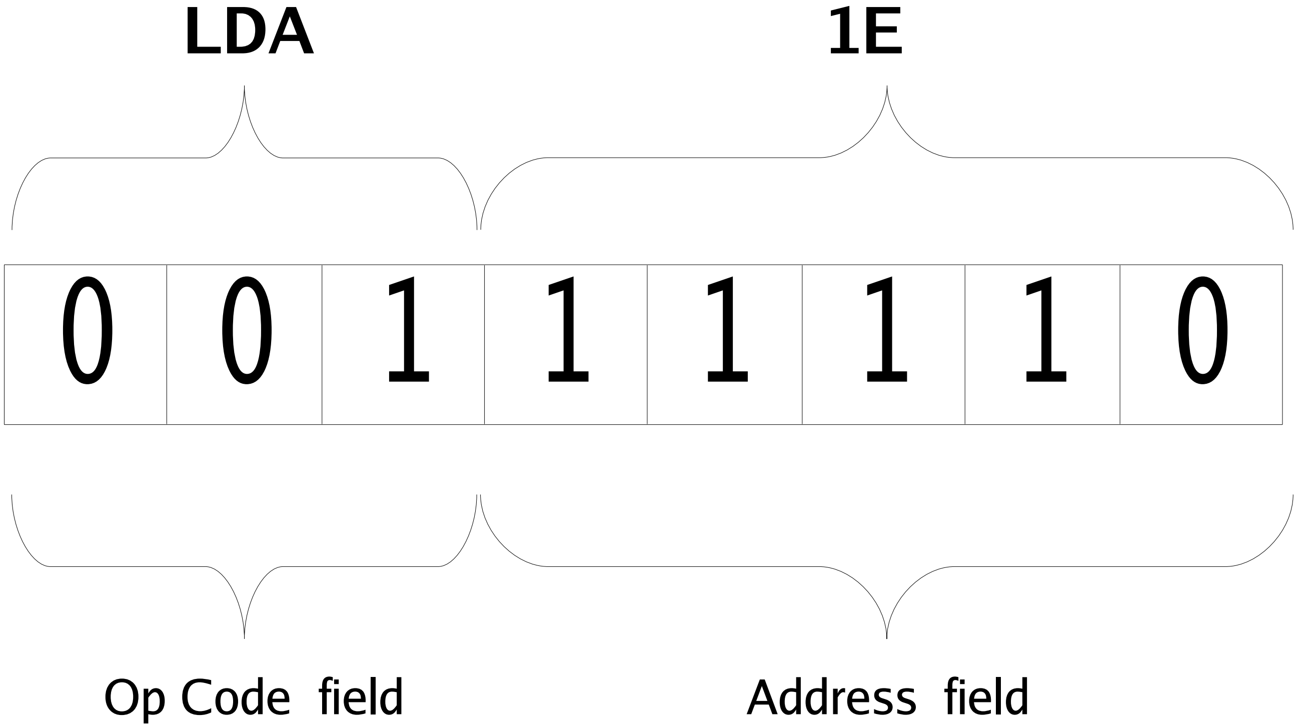
Operator	*Op Code*
LDA	001
ADD	010
OUT	110
HALT	111



To us, 001 means LDA!

Controlling a program

- Then, a machine instruction consists of an op code segment or “field”, followed by an address field giving the operand in binary:



LDA

1E

0	0	1	1	1	1	1	0

Op Code field

Address field

Controlling a program

- So, here's our program translated into machine language:

Assembly Language	Machine Language
LDA 1E	001 11110
ADD 1F	010 11111
OUT	110 xxxxx
HALT	111 xxxxx

Any 5 bits are OK for these address fields, as they'll be ignored

Controlling a program

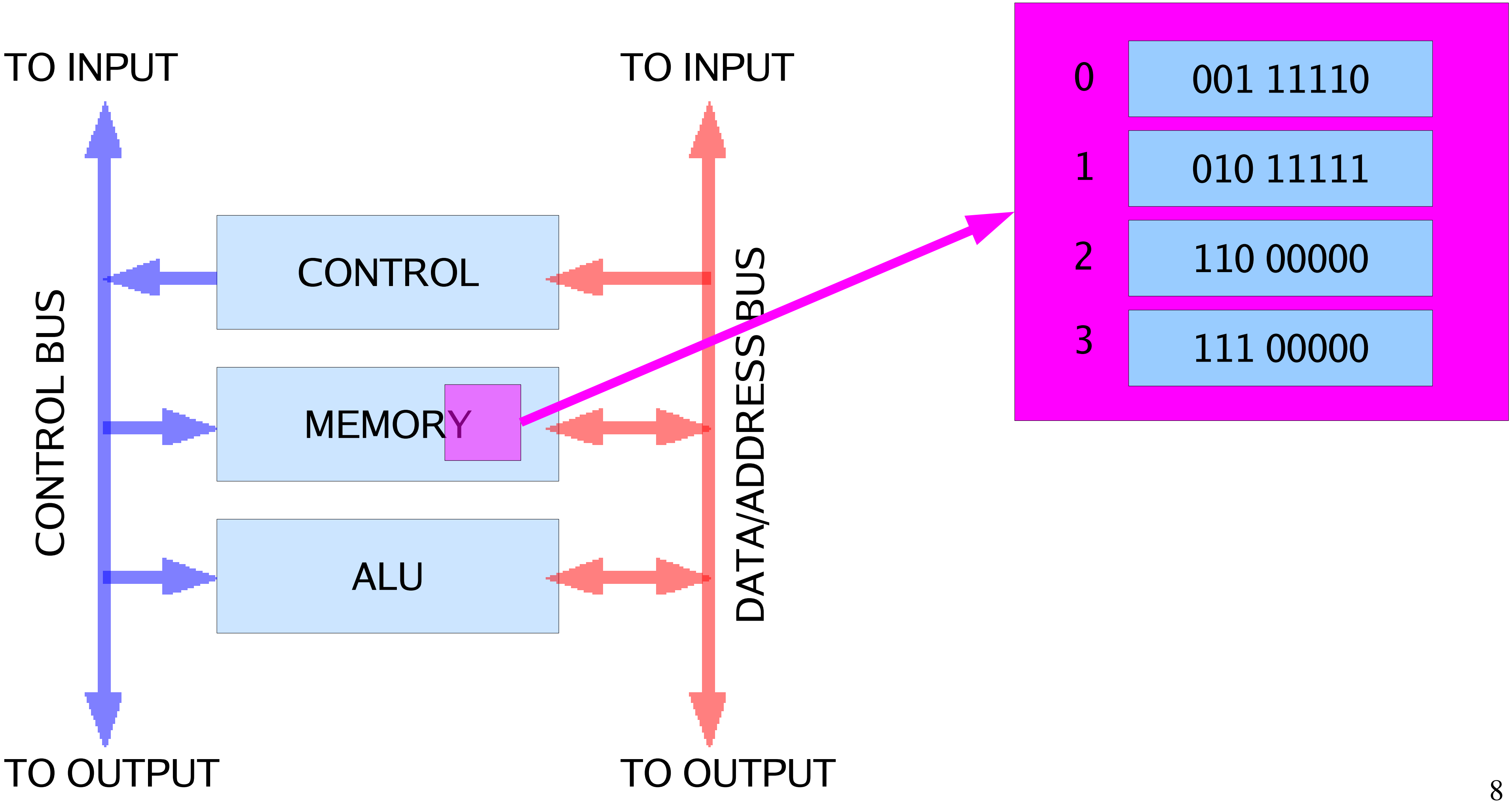
- Now, assuming there is an input device, the program steps are read into consecutive memory addresses, beginning with 0.
- The contents of memory are then:

Address	Contents
0	001 11110
1	010 11111
2	110 00000
3	111 00000

Note that the program step number is the address where it's stored

Controlling a program

- Rooms in memory hotel



0	001 11110
1	010 11111
2	110 00000
3	111 00000

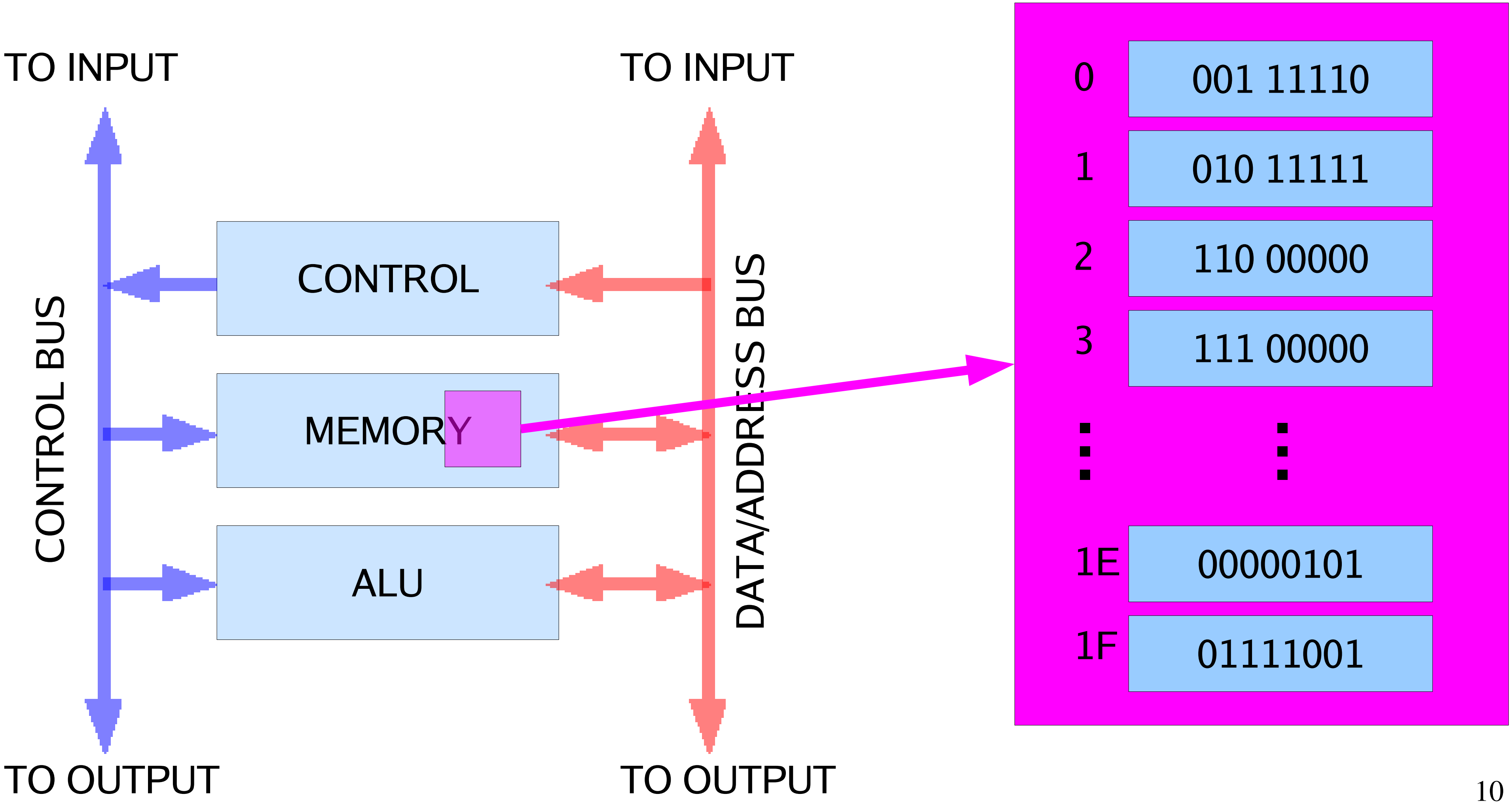
Controlling a program

- And, we also need to enter the data: The two numbers to be added. Any two numbers will do, say 5 and 121
- They go in addresses 1E and 1F:

Address	Contents		
1E	00000101	←	5
1F	01111001	←	121

Controlling a program

- More rooms in memory hotel



TO INPUT

TO INPUT

CONTROL

MEMORY

ALU

CONTROL BUS

DATA/ADDRESS BUS

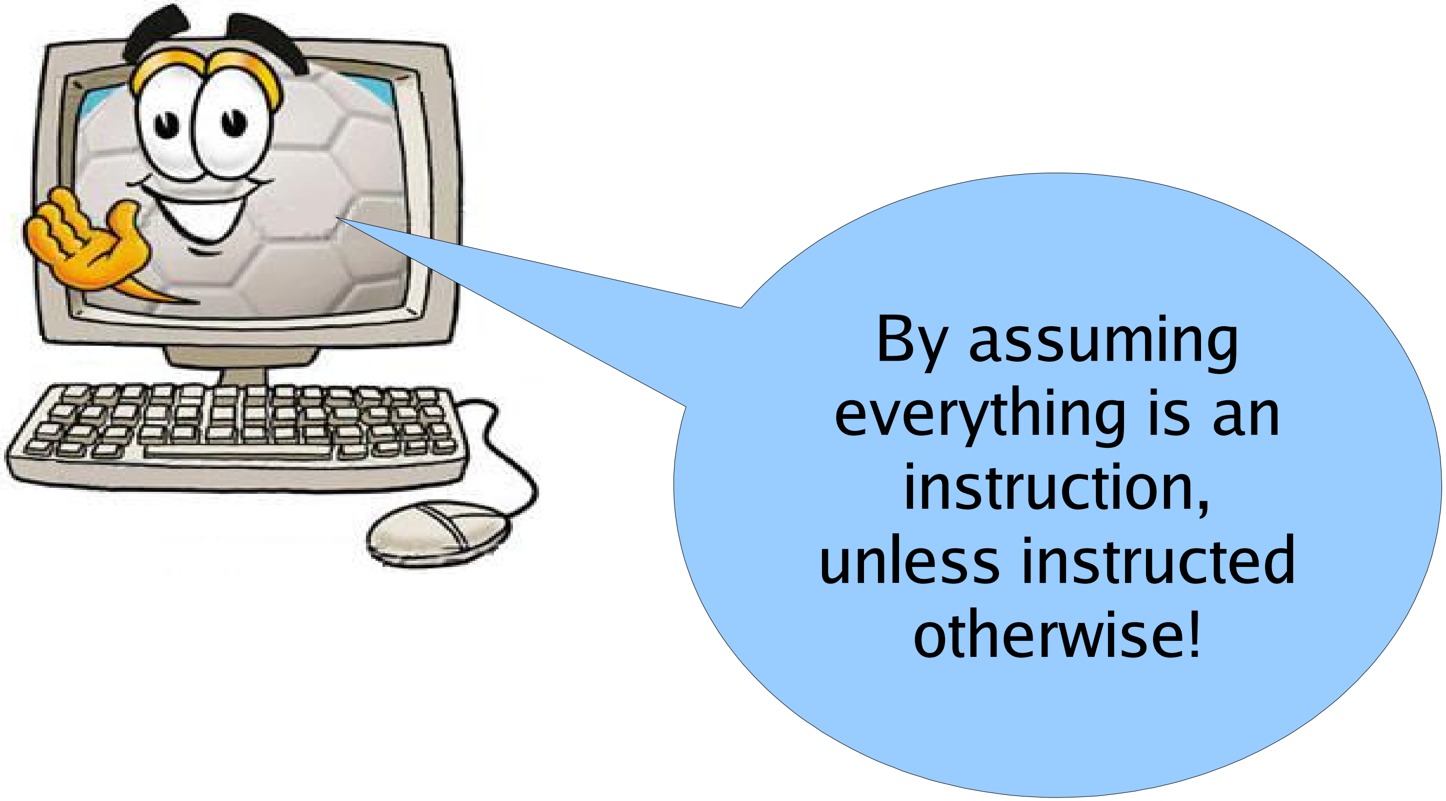
TO OUTPUT

TO OUTPUT

0	001 11110
1	010 11111
2	110 00000
3	111 00000
⋮	⋮
1E	00000101
1F	01111001

Controlling a program

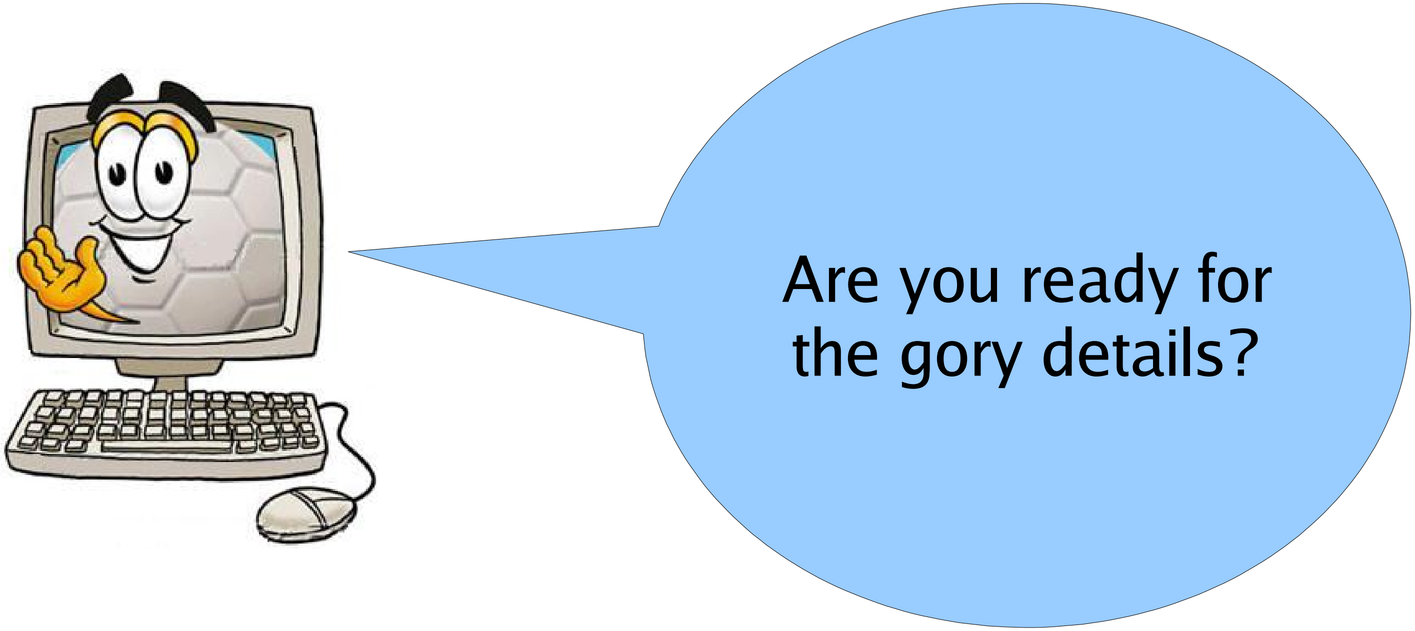
- How can the computer distinguish data from instructions?



By assuming everything is an instruction, unless instructed otherwise!

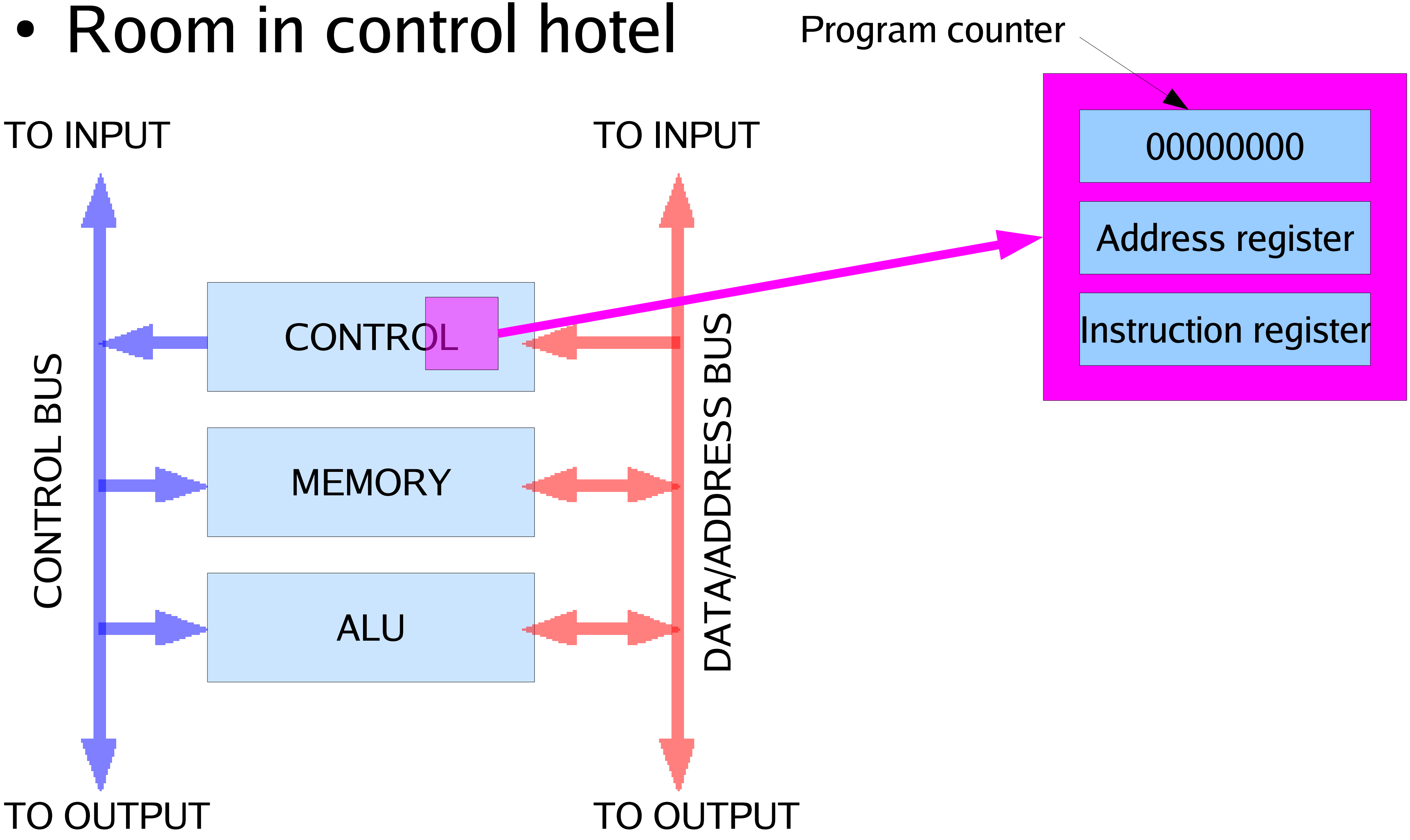
Controlling a program

- Once the program is stored, control can begin execution, in a series of even more primitive steps called **microinstructions**
- One microinstruction occuring with each clock pulse



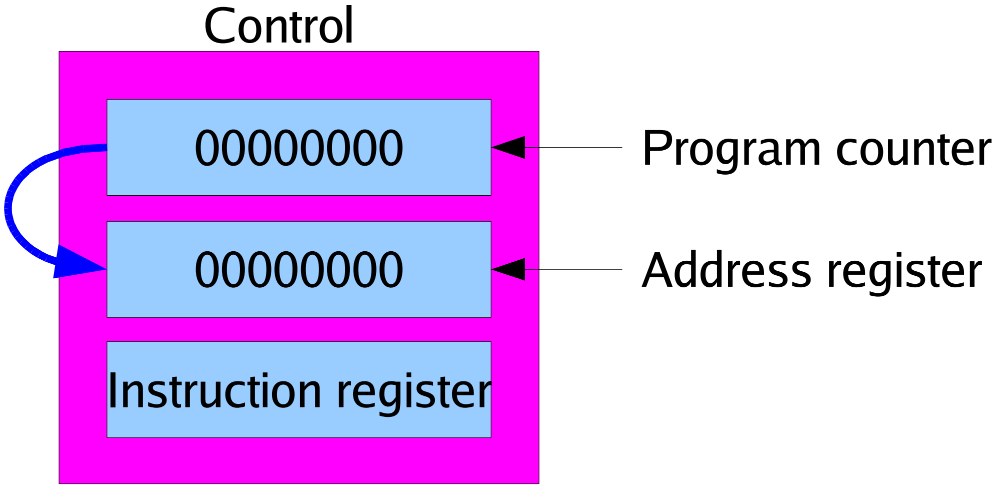
Controlling a program

- Room in control hotel



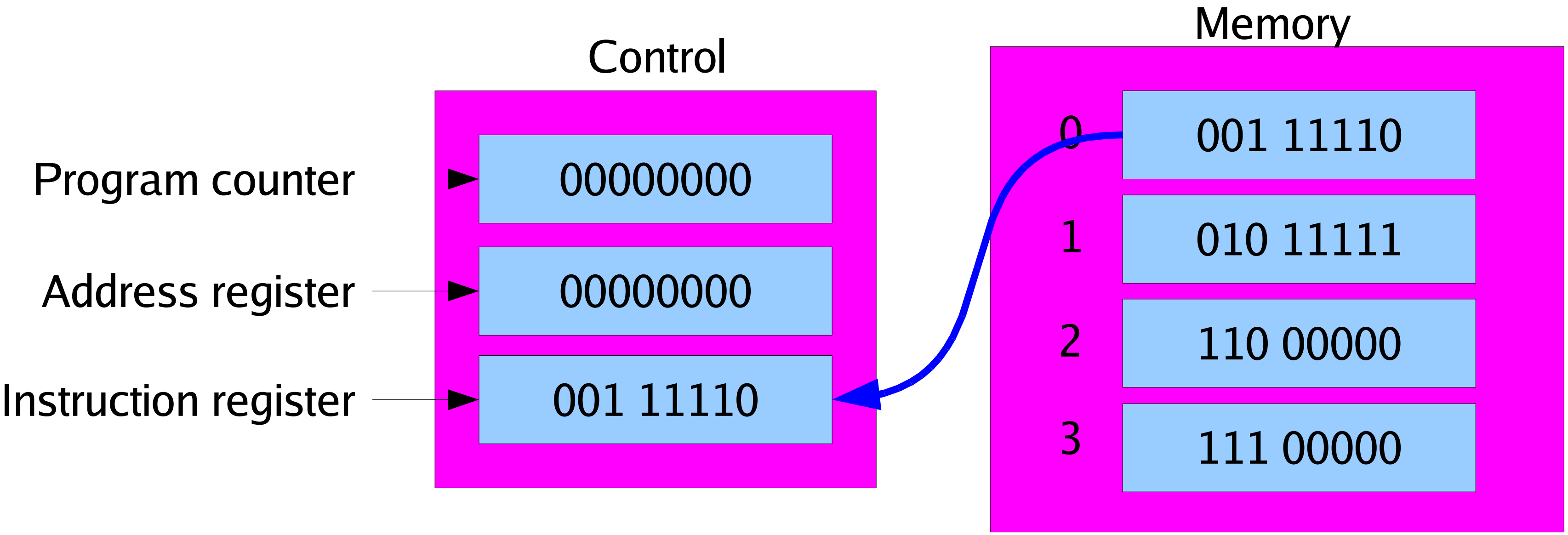
Running a program

- Control begins by fetching the first instruction
- STEP 0.0: Control moves the contents of program counter (00000000 to begin with) to address register



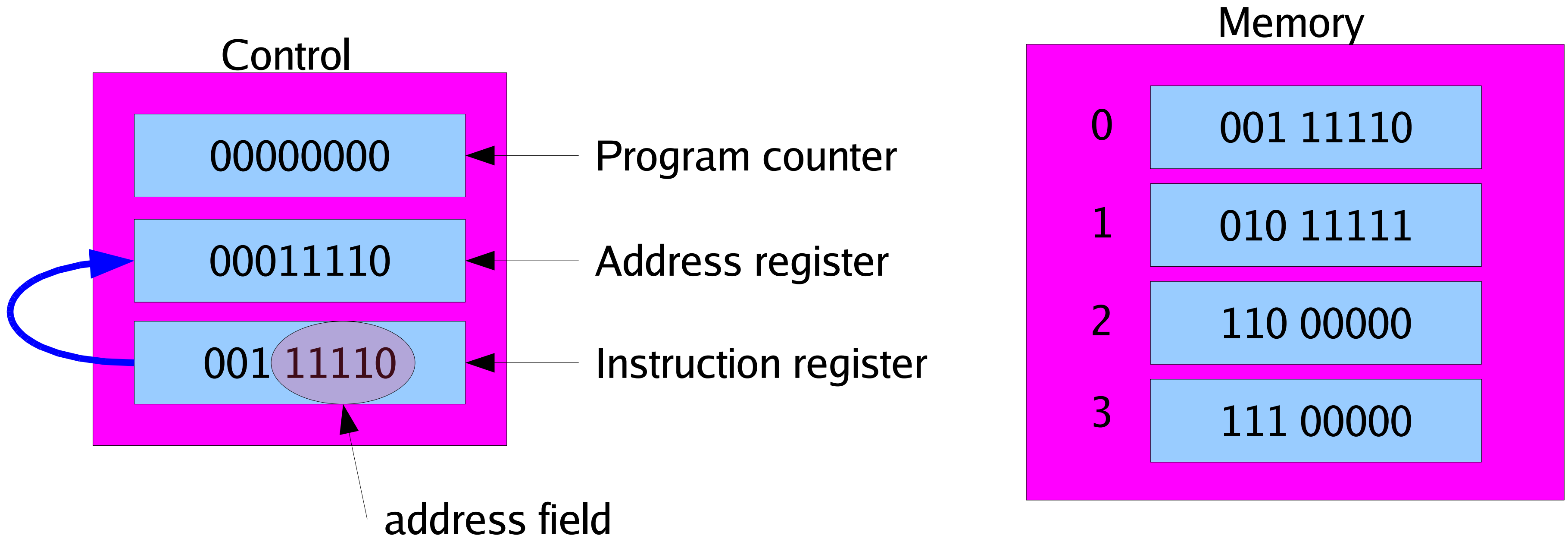
Running a program

- Then,
- STEP 0.1: Control moves the contents of memory address (that is 00000000) to instruction register



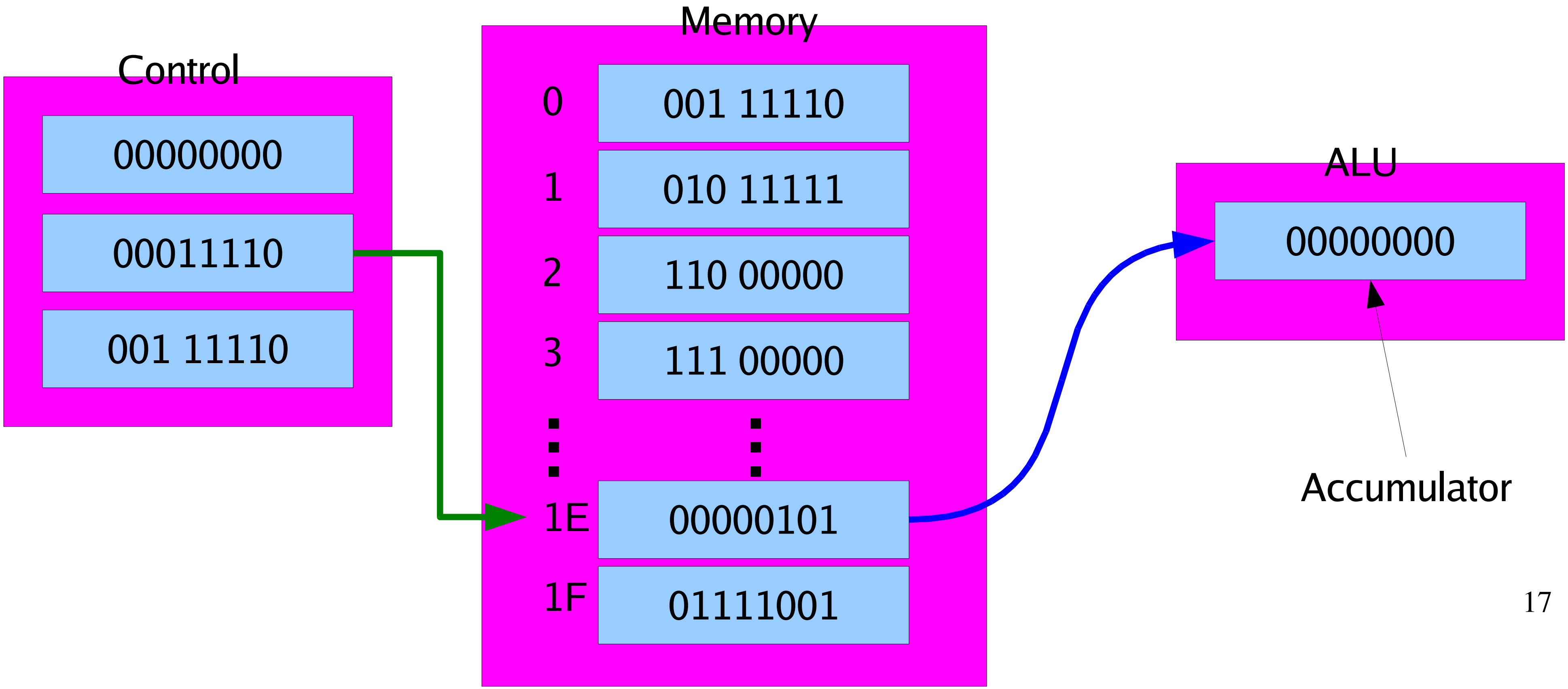
Running a program

- The instruction register now holds the first instruction. Control reads it and,
- STEP 0.2: Control moves the instruction register's address field to address register



Running a program

- Then,
- STEP 0.3: Control moves the contents of that memory address to accumulator



Running a program

- The accumulator is now loaded with the first piece of data. One microinstruction remains:
- STEP 0.4: Increment the program counter by 1.

Control

00000001

00011110

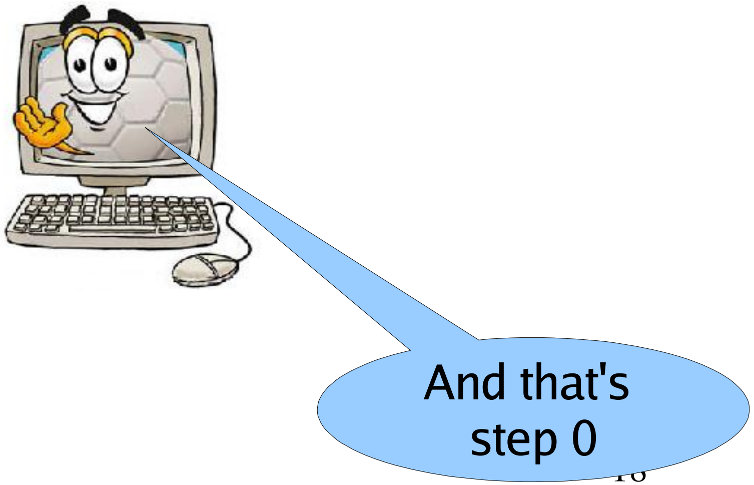
001 11110

Memory

0	001 11110
1	010 11111
2	110 00000
3	111 00000
⋮	⋮
1E	00000101
1F	01111001

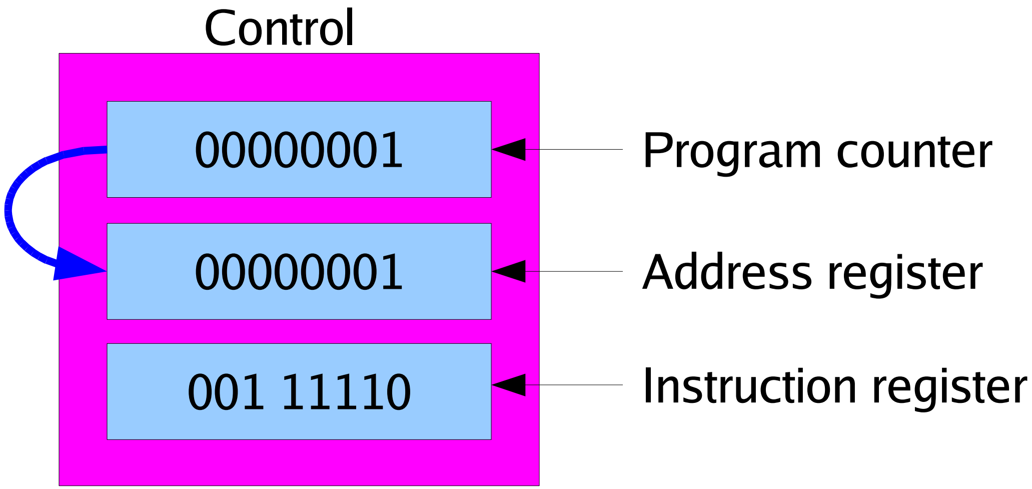
ALU

00000000



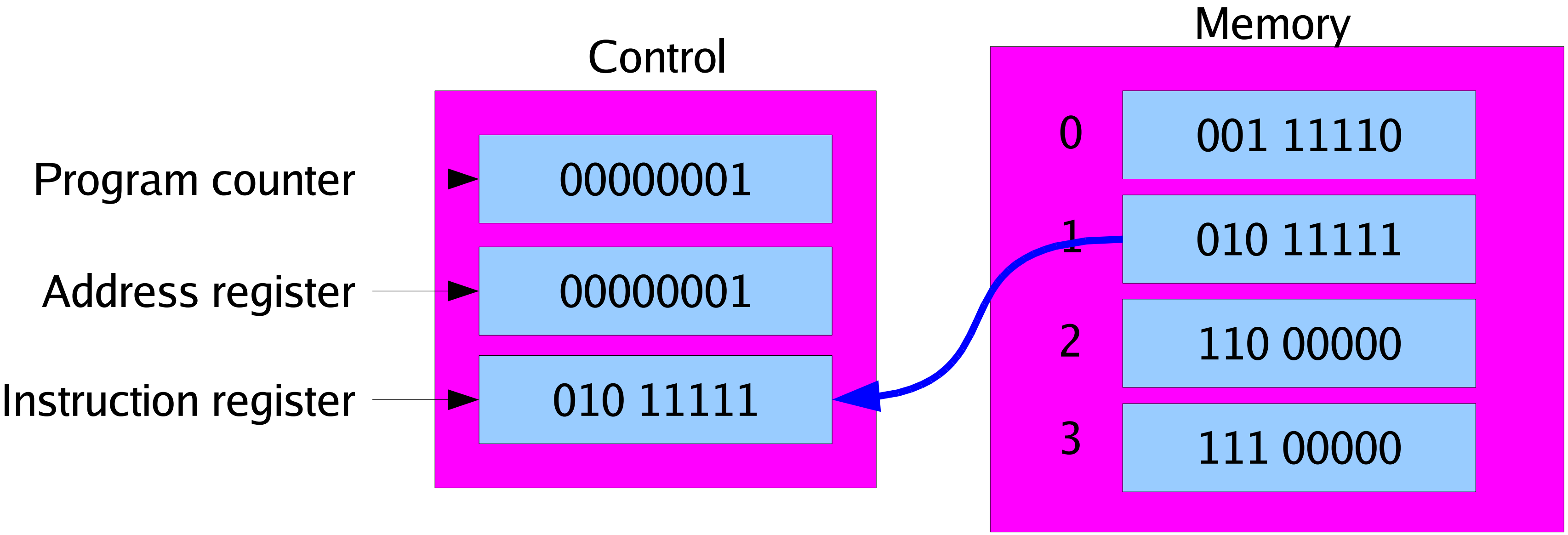
Running a program

- A bit confused? Let's go through it again with the next step: ADD
- Again control begins with a “fetch phrase”
- STEP 1.0: Control moves the contents of program counter (now 00000001) to address register



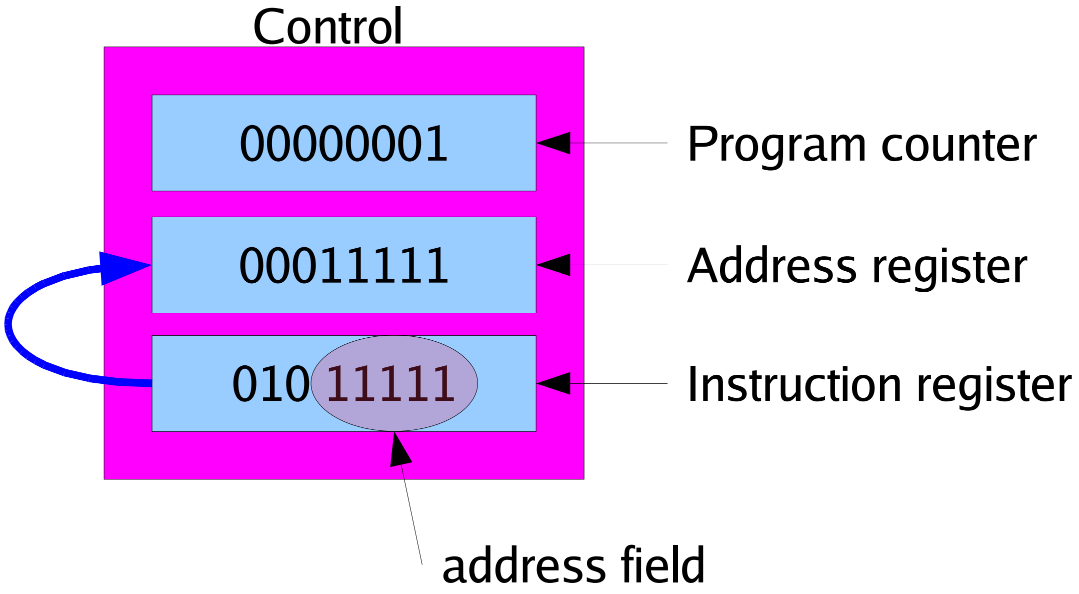
Running a program

- Then,
- STEP 1.1: Control moves the contents of that address to instruction register



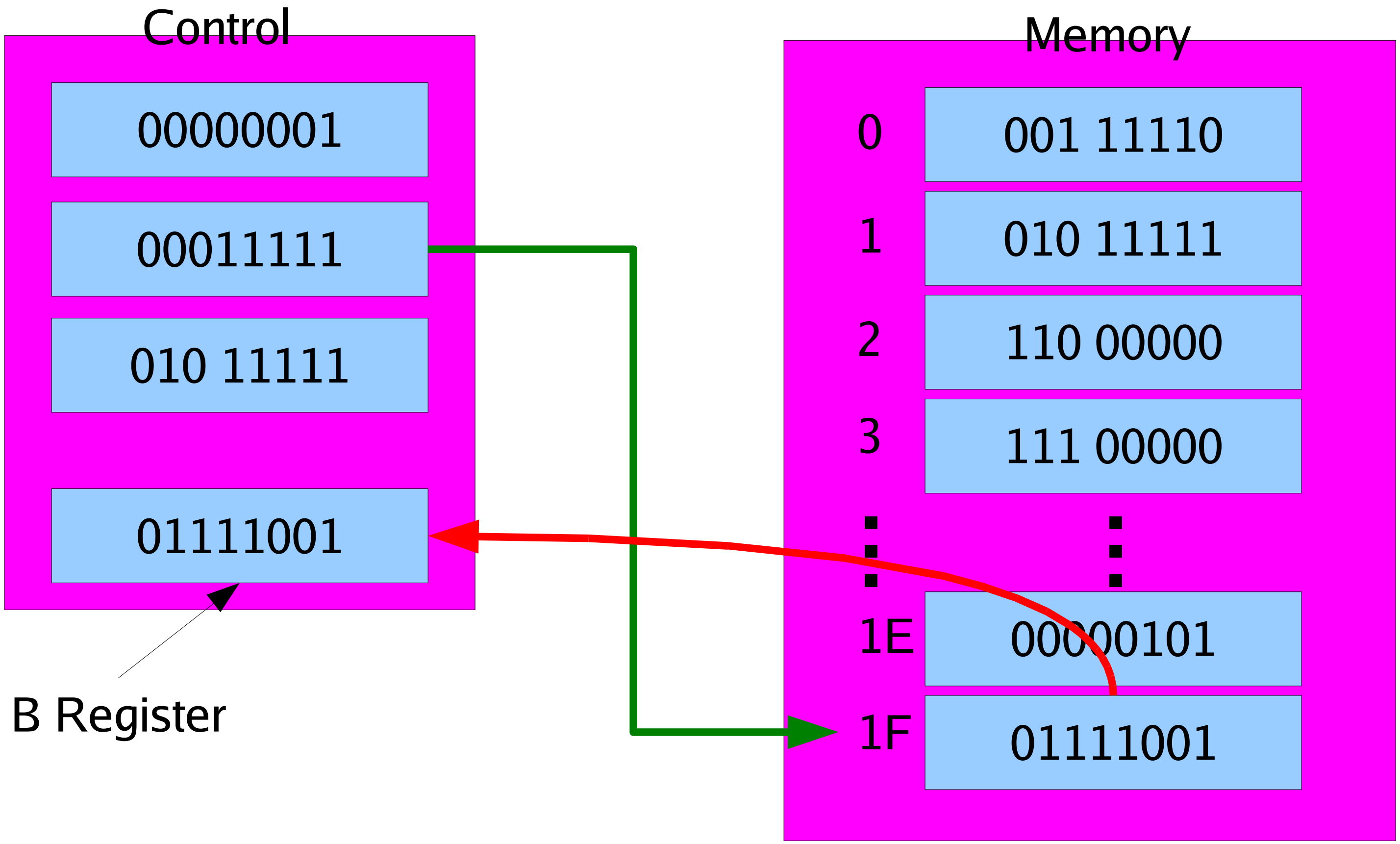
Running a program

- The instruction in instruction register, 010 11111, causes control to
- STEP 1.2: Control moves the address from instruction register to address register



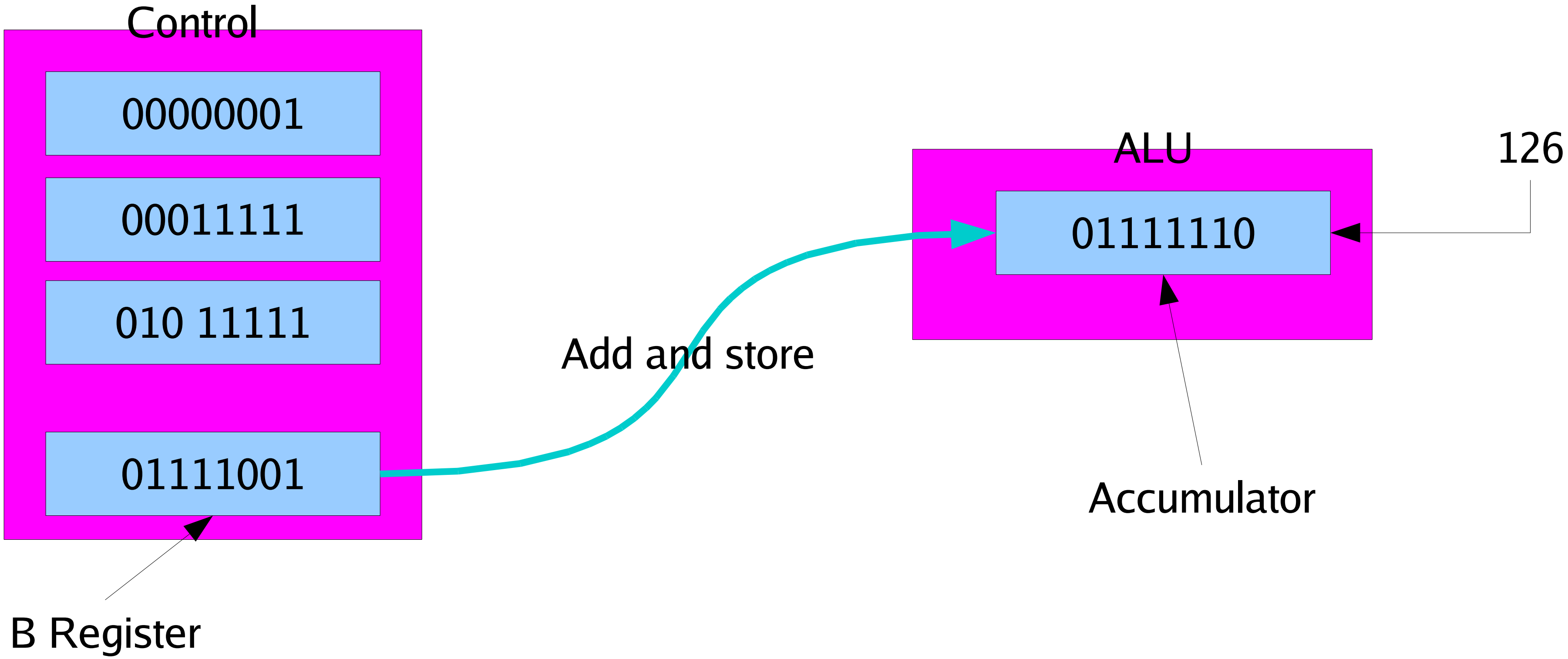
Running a program

- Then,
- STEP 1.3: Control moves the contents of that memory address to B register



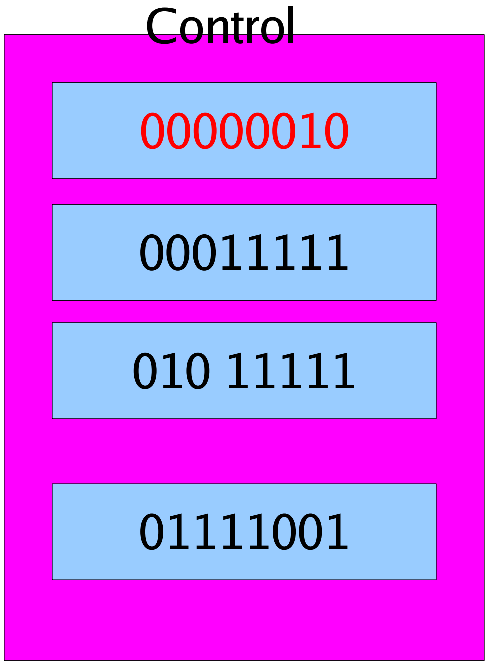
Running a program

- Then,
- STEP 1.4: Control signals the ALU to add the contents of B register and the accumulator and put the sum to accumulator



Running a program

- Again, there's one more step:
- STEP 1.5: Increment the program counter by 1.



Control

00000010

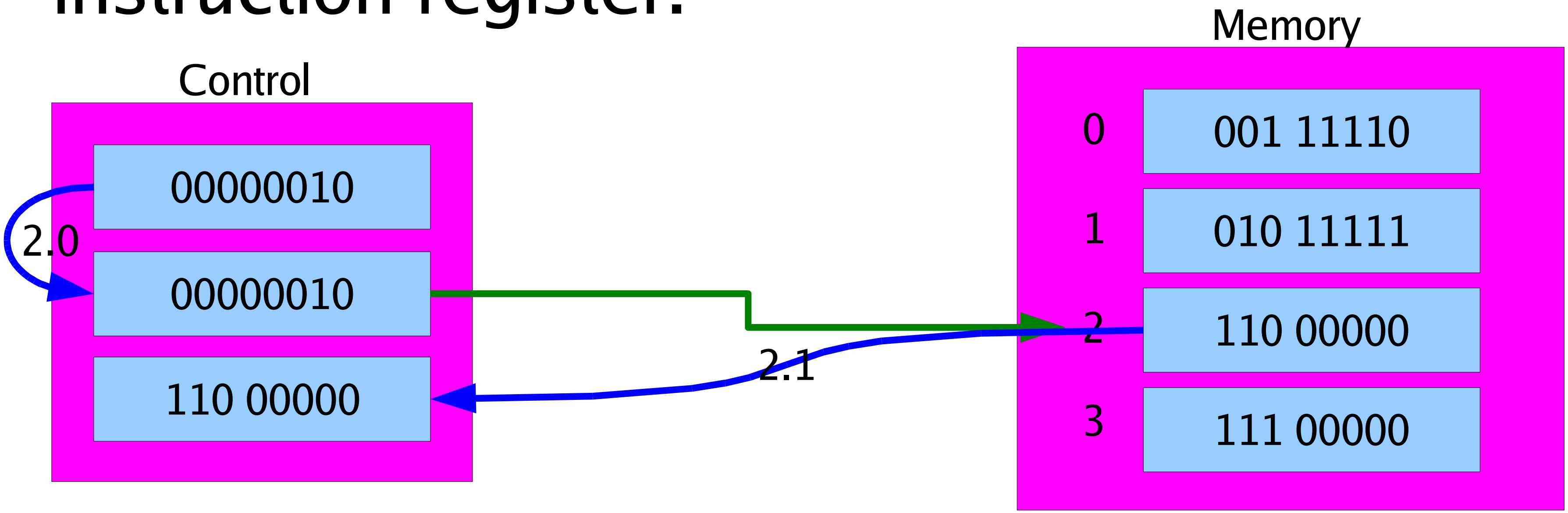
00011111

010 11111

01111001

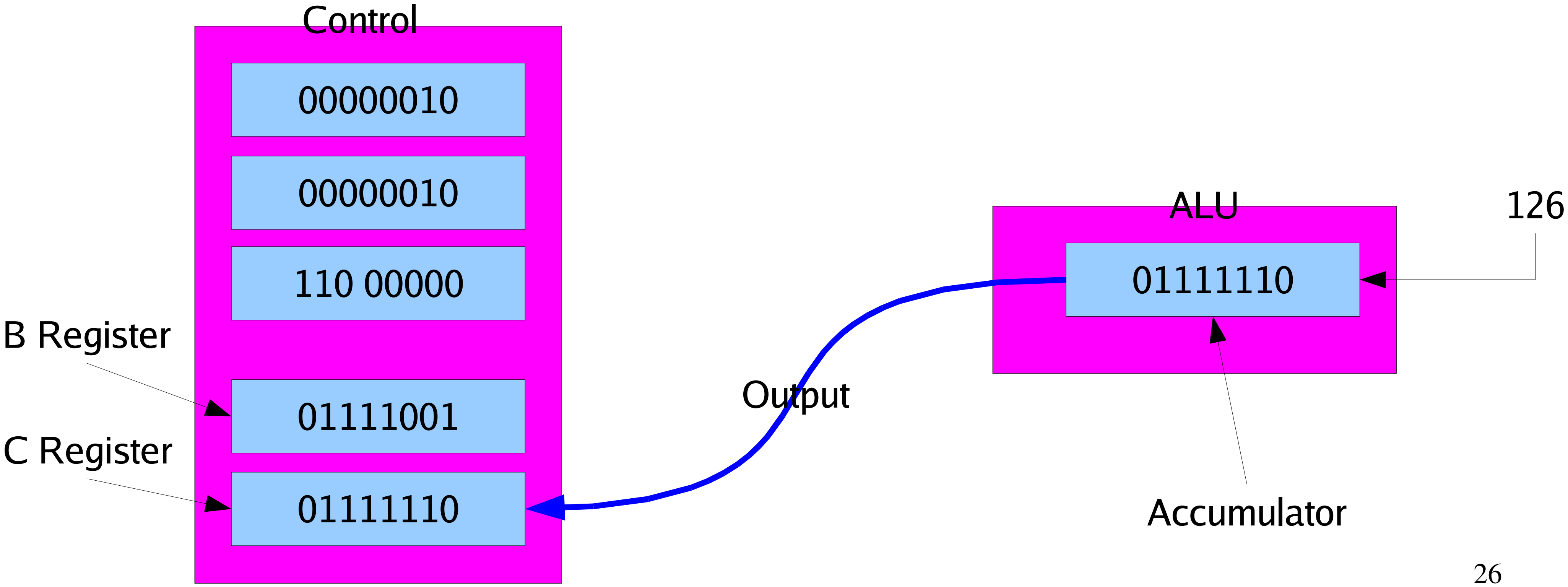
Running a program

- And finally?
- Well, luckily the last two instructions are easier:
- STEPS 2.0 & 2.1: The same fetch instructions as before, putting the instruction “OUT” in the instruction register.



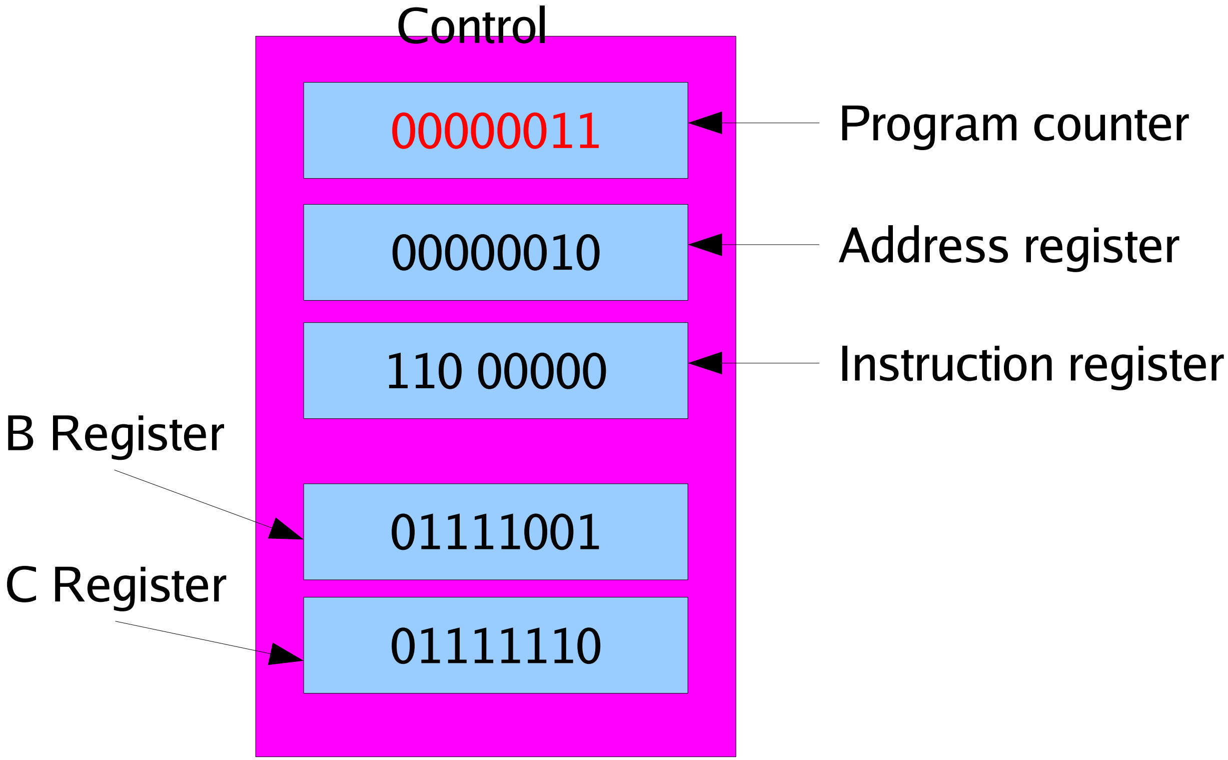
Running a program

- “OUT's” op code (110) causes control to:
- STEP 2.2: Control moves the contents of accumulator to C register



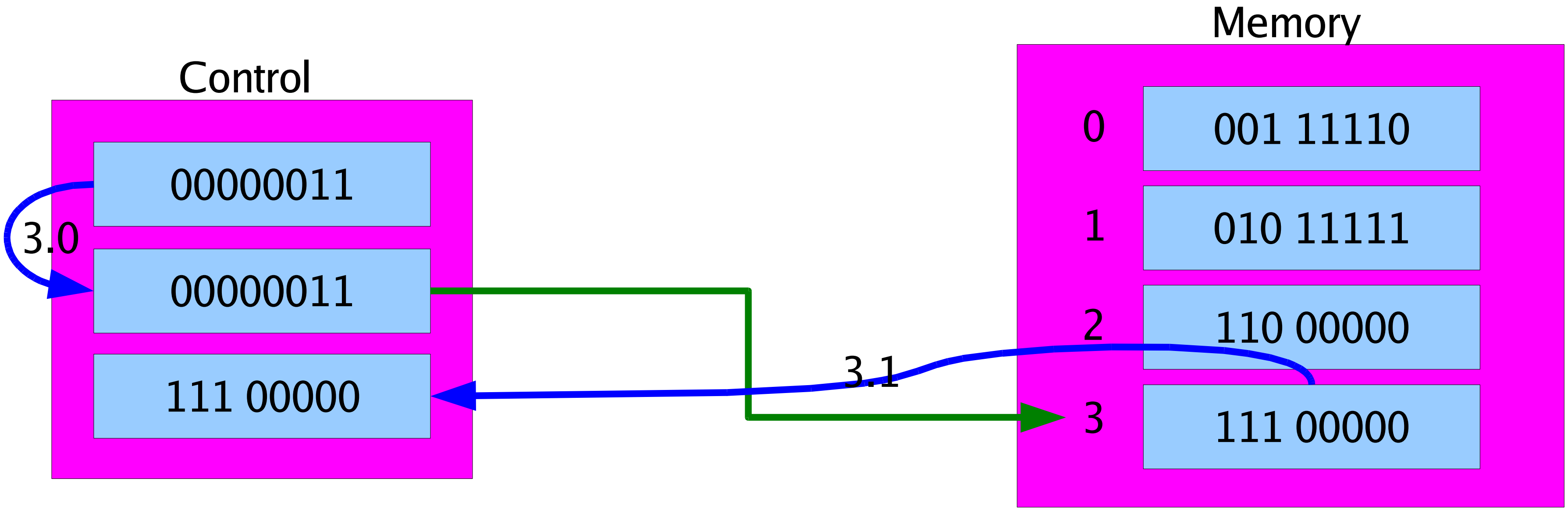
Running a program

- Then,
- STEP 2.3: Increment the program counter by 1.



Running a program

- Finally, control fetches the instruction “HALT” (op code of 111)
- This causes the control to
- STEP 3.2: Do nothing



Running a program

- Are you beginning to see what kind of beast control really is?