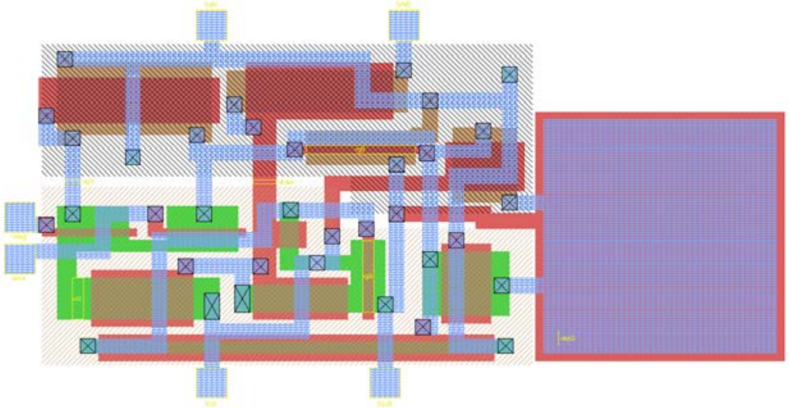
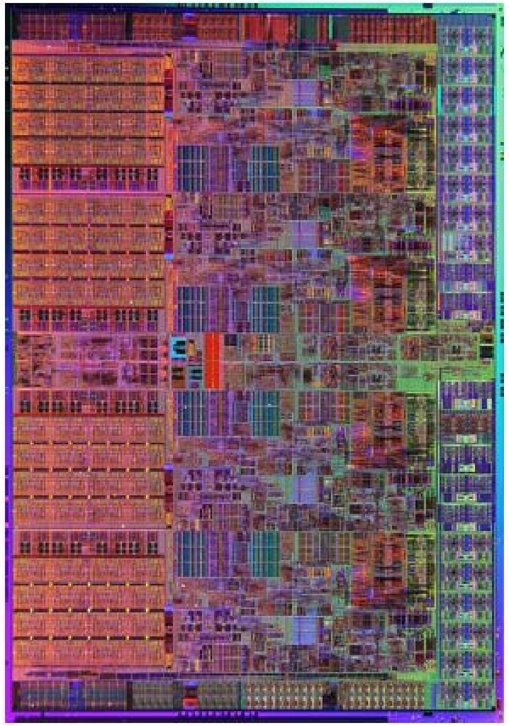
Three ways to look at a microprocessor

- What a VLSI (very large-scale integration) chip designer sees:



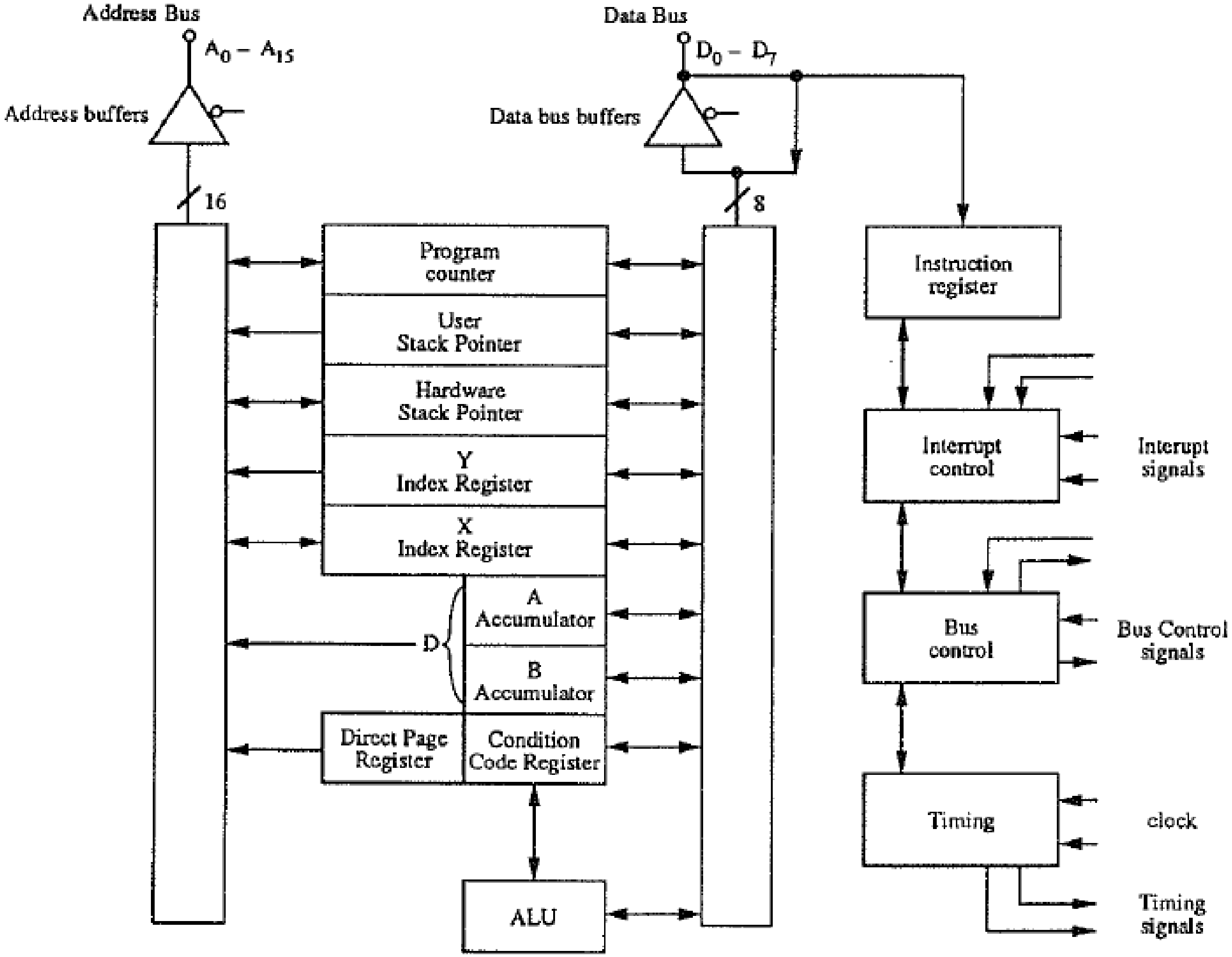
VLSI Design

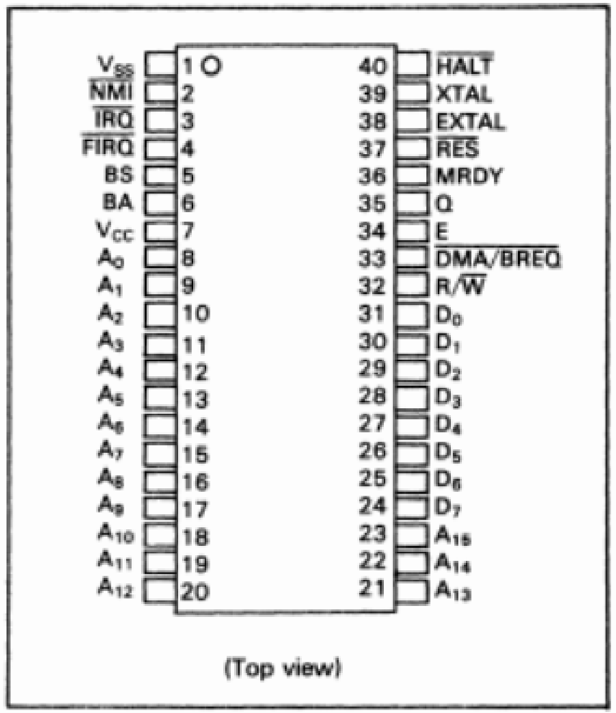


Core i7 die

- What a PCB (printed circuit board) designer sees:

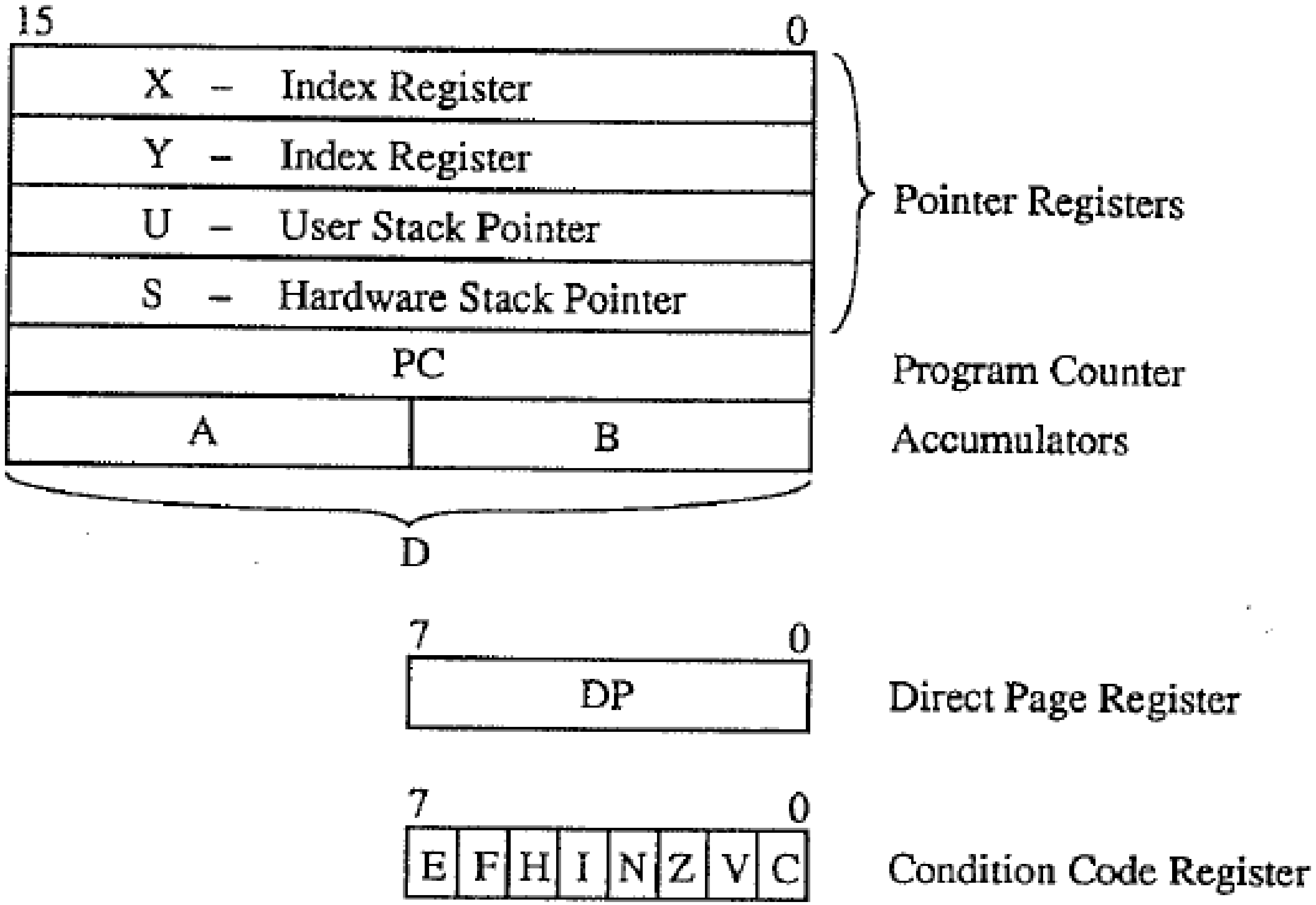
Block diagram of a 6809 uP





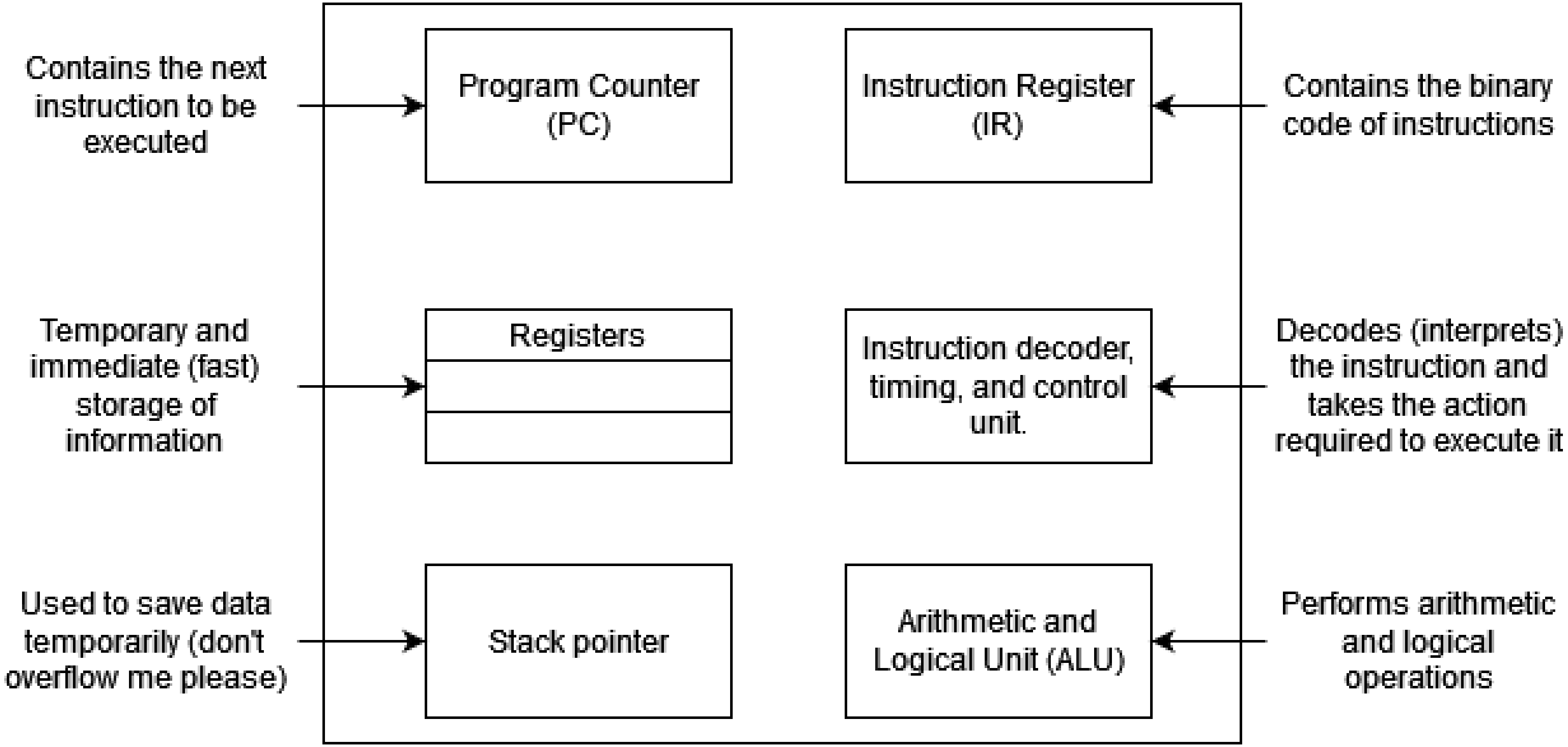
- What a programmer sees:

Programming model of a 6809 microprocessor



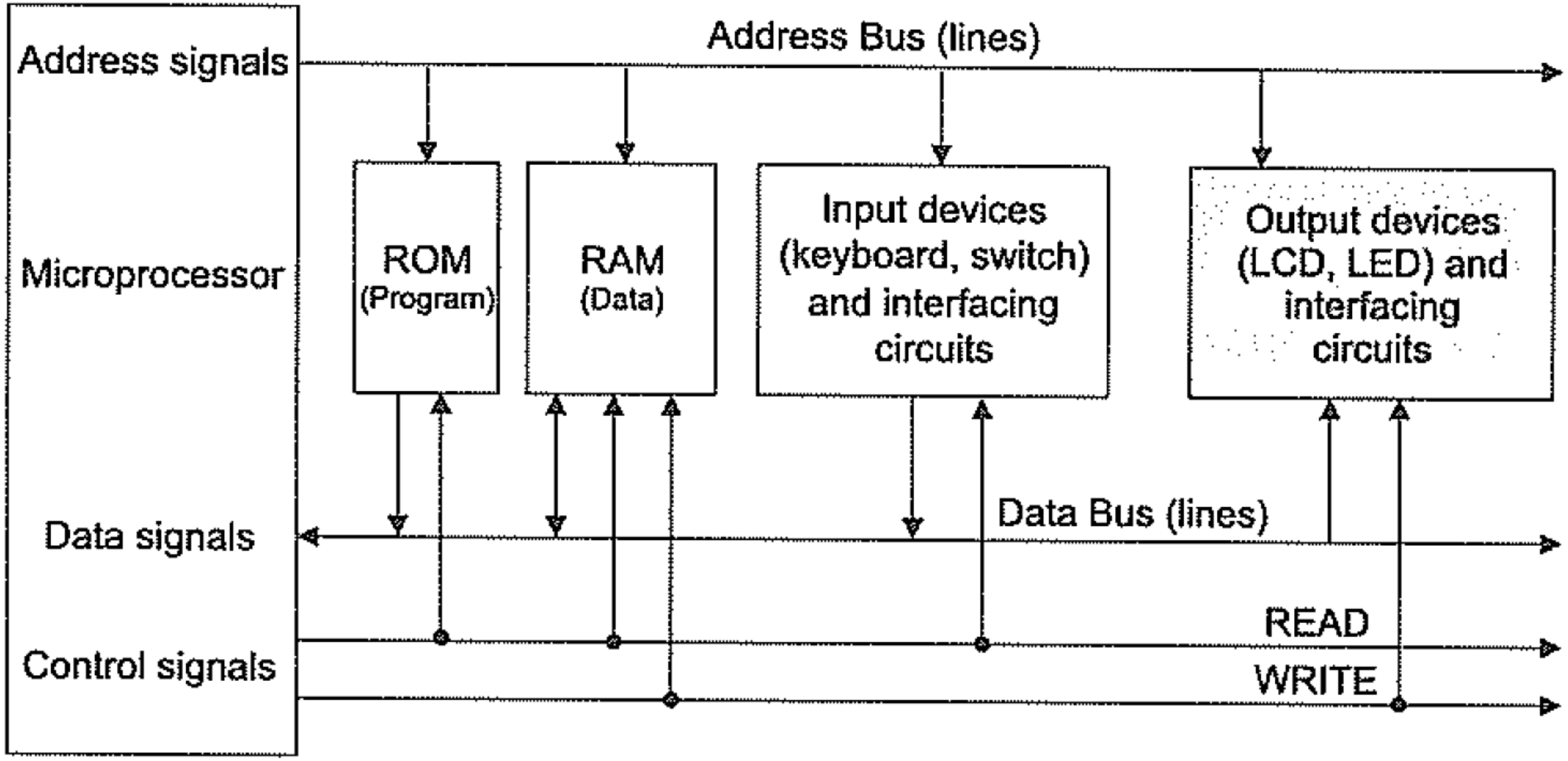
General block diagram of a microprocessor core.

A microprocessor is made up of a series of interconnected functional units. All controlled by a central timing and control unit that behaves like a finite state machine.


Contains the next instruction to be executed
Program Counter (PC)
Instruction Register (IR)
Contains the binary code of instructions
Temporary and immediate (fast) storage of information
Registers
Instruction decoder, timing, and control unit.
Decodes (interprets) the instruction and takes the action required to execute it
Used to save data temporarily (don't overflow me please)
Stack pointer
Arithmetic and Logical Unit (ALU)
Performs arithmetic and logical operations

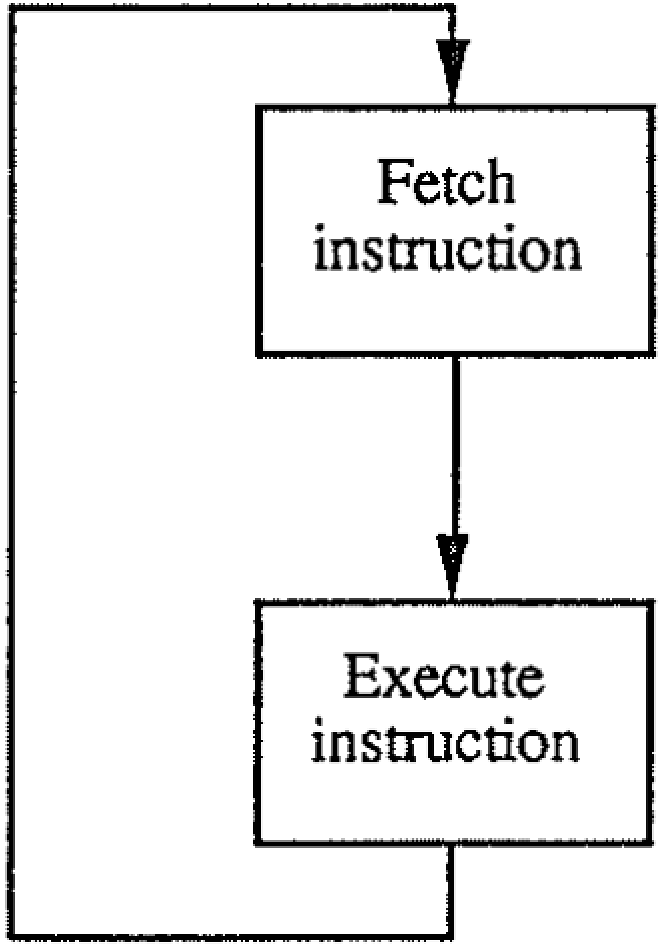
Components of a microcomputer

- A microcomputer is built up from a microprocessor, some memory, and several I/O peripherals. These all communicate via a parallel bus which consists of an **Address** bus, a **Data** bus, and a **Control** bus.



How does a microprocessor work?

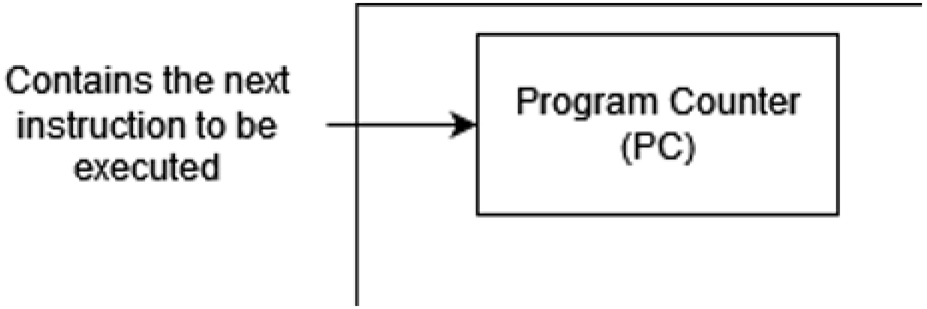
1. The microprocessor fetches an instruction.
   We will ignore how it does this for now.
2. This instruction tells the microprocessor what to do.
   For example, it might perform an XOR operation.
3. The microprocessor executes the instruction, and this process repeats where a new instruction is fetched.



Fetching an instruction

How do we fetch an instruction?

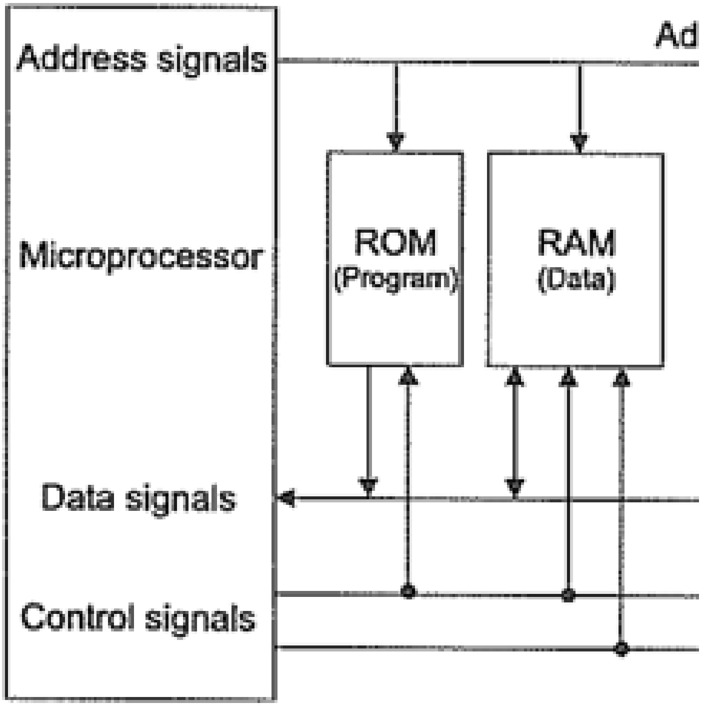
In the general block diagram we had a **program counter**.



The program counter is a register (16-bits wide for small microprocessors) which contains the **address** in memory of the next instruction to be executed.

Fetching an instruction

How do we fetch an instruction?



To fetch an instruction

the following process is used;

1. the microprocessor places the address contained in the program counter onto the address bus.
2. A read signal is then set in the control bus and the memory responds by placing the data (instruction) onto the data bus.
3. The microprocessor then loads this instruction into the instruction register. This completes the fetch.

Executing an instruction

In order to execute the instruction, the microprocessor must first **decode** the instruction and then perform the appropriate operations.

For example, the instruction may have been to increment (add 1) the number contained in the accumulator.

For the 8051 series microprocessor, this instruction called a “**machine code**”, has the hexadecimal code 04. This has a corresponding abbreviated description, called a **Mnemonic**, “INCA”.

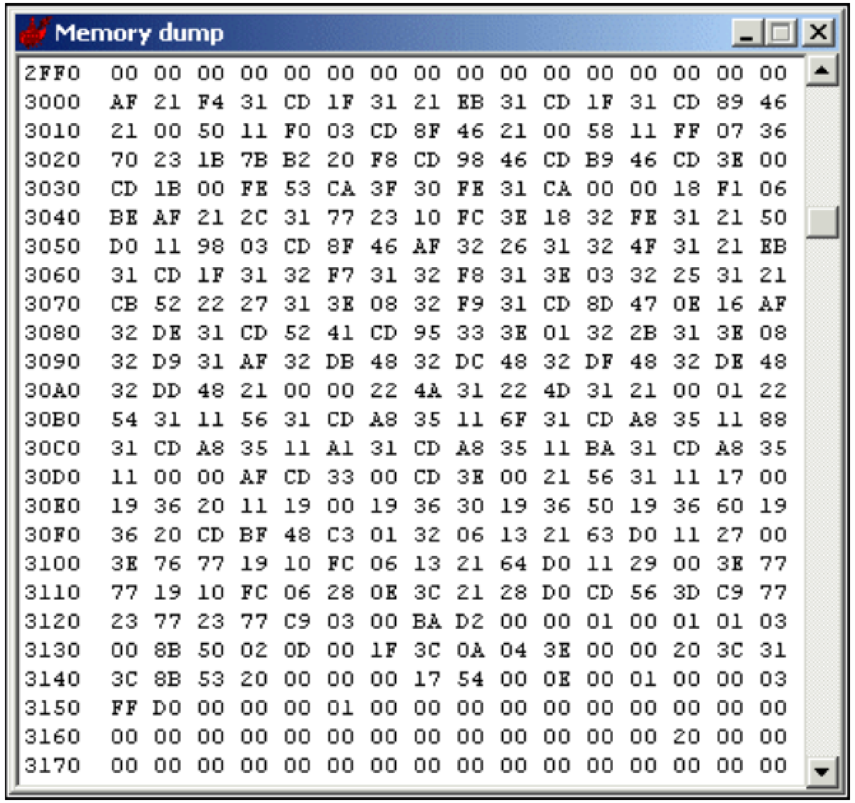
Executing an instruction

There are several different types of instructions that are possible, and these are usually summarised in a table with corresponding Mnemonics. (at_c51ism.pdf)

Mnemonics are a more convenient way of describing the program operations and are the main part of an assembly language.

A machine code program

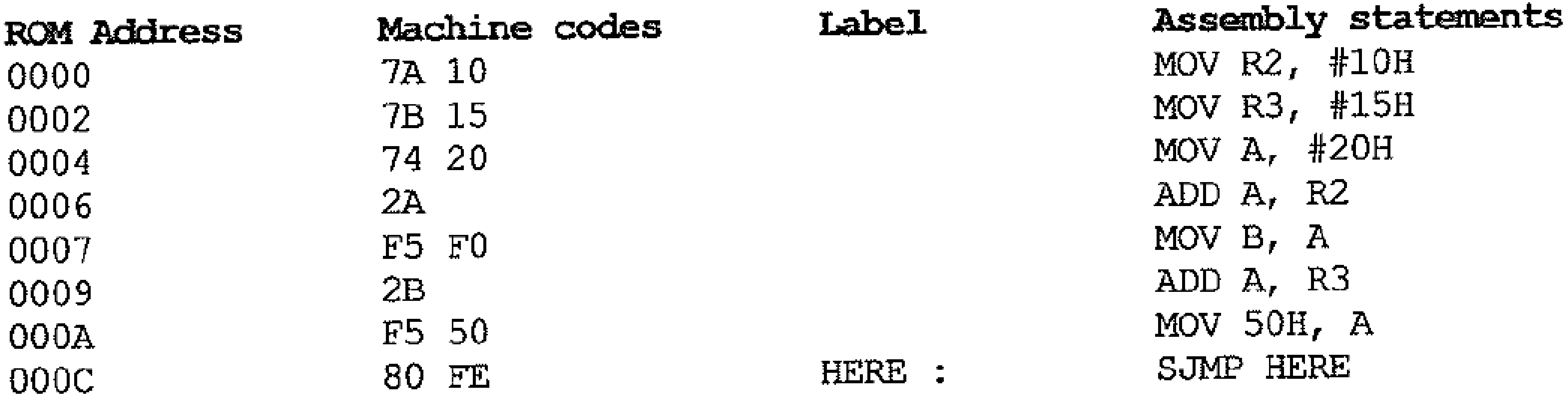
The program consists of a series of machine code instructions that are usually placed sequentially in memory.



```
2FF0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3000   AF 21 F4 31 CD 1F 31 21 EB 31 CD 1F 31 CD 89 46
3010   21 00 50 11 F0 03 CD 8F 46 21 00 58 11 FF 07 36
3020   70 23 1B 7B B2 20 F8 CD 98 46 CD B9 46 CD 3E 00
3030   CD 1B 00 FE 53 CA 3F 30 FE 31 CA 00 00 18 F1 06
3040   BE AF 21 2C 31 77 23 10 FC 3E 18 32 FE 31 21 50
3050   D0 11 98 03 CD 8F 46 AF 32 26 31 32 4F 31 21 EB
3060   31 CD 1F 31 32 F7 31 32 F8 31 3E 03 32 25 31 21
3070   CB 52 22 27 31 3E 08 32 F9 31 CD 8D 47 0E 16 AF
3080   32 DE 31 CD 52 41 CD 95 33 3E 01 32 2B 31 3E 08
3090   32 D9 31 AF 32 DB 48 32 DC 48 32 DF 48 32 DE 48
30A0   32 DD 48 21 00 00 22 4A 31 22 4D 31 21 00 01 22
30B0   54 31 11 56 31 CD A8 35 11 6F 31 CD A8 35 11 88
30C0   31 CD A8 35 11 A1 31 CD A8 35 11 BA 31 CD A8 35
30D0   11 00 00 AF CD 33 00 CD 3E 00 21 56 31 11 17 00
30E0   19 36 20 11 19 00 19 36 30 19 36 50 19 36 60 19
30F0   36 20 CD BF 48 C3 01 32 06 13 21 63 D0 11 27 00
3100   3E 76 77 19 10 FC 06 13 21 64 D0 11 29 00 3E 77
3110   77 19 10 FC 06 28 0E 3C 21 28 D0 CD 56 3D C9 77
3120   23 77 23 77 C9 03 00 BA D2 00 00 01 00 01 01 03
3130   00 8B 50 02 0D 00 1F 3C 0A 04 3E 00 00 20 3C 31
3140   3C 8B 53 20 00 00 00 17 54 00 0E 00 01 00 00 03
3150   FF D0 00 00 00 01 00 00 00 00 00 00 00 00 00 00
3160   00 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00
3170   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

An assembly languarge program

Large machine code programs are impossible to interpret so we use Mnemonics and formatting to show program structure.



ROM Address	Machine codes	Label	Assembly statements
0000	7A 10		MOV R2, #10H
0002	7B 15		MOV R3, #15H
0004	74 20		MOV A, #20H
0006	2A		ADD A, R2
0007	F5 F0		MOV B, A
0009	2B		ADD A, R3
000A	F5 50		MOV 50H, A
000C	80 FE	HERE :	SJMP HERE