CENG 311
Computer Architecture

Lecture 1

Introduction to Computer Architecture

Asst. Prof. Tolga Ayav, Ph.D.

Department of Computer Engineering
İzmir Institute of Technology

Computer Design

Instruction Set Design

- **Machine Language**
- **Compiler View**
- **"Computer Architecture"**
- **"Instruction Set Processor"**

"Building Architect"

Computer Hardware Design

- Machine Implementation\
- Logic Designer's View
- "Processor Architecture"
- "Computer Organization"

"Construction Engineer"

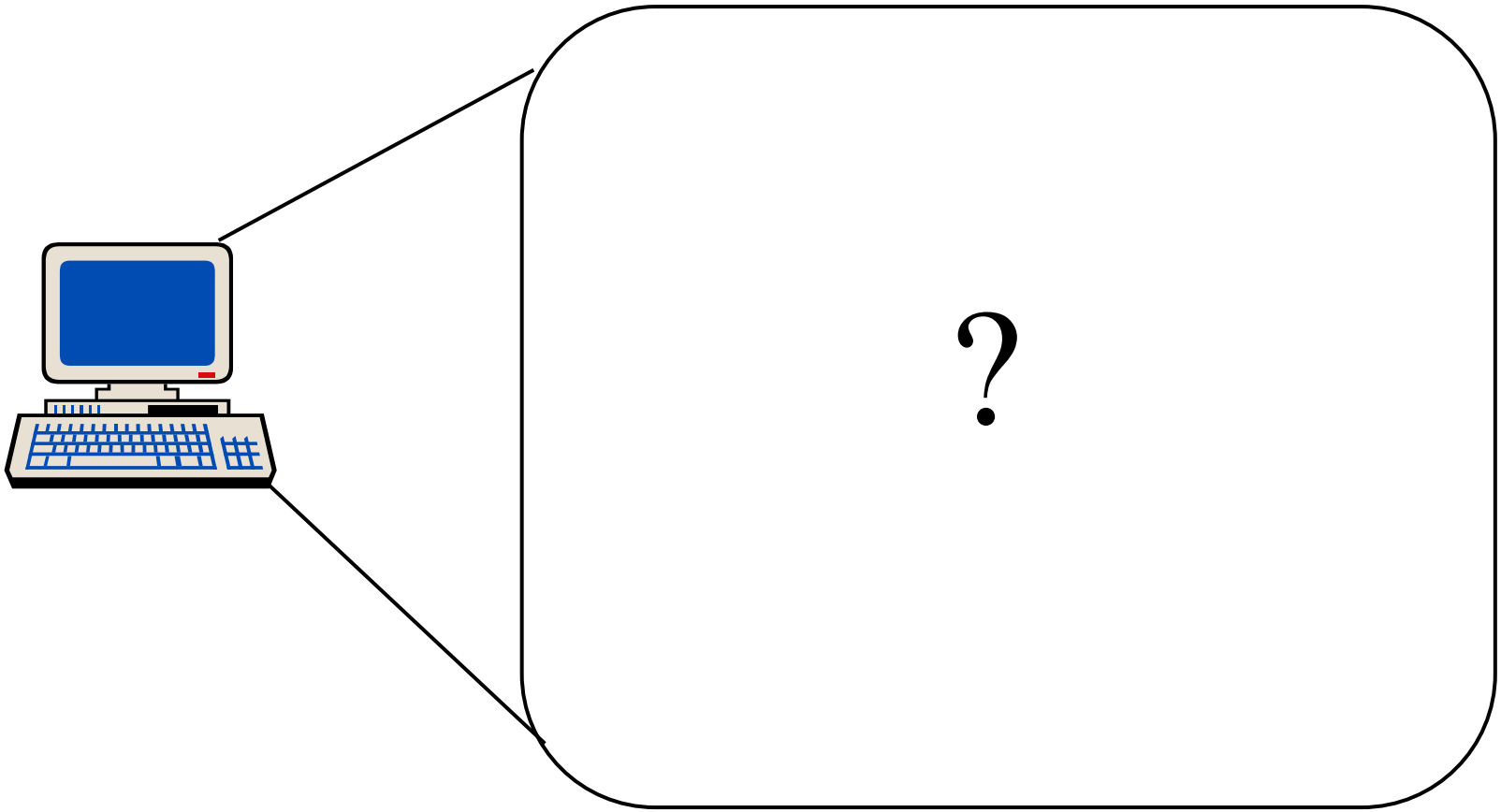
Few people design computers! Very few design instruction sets!

Many people design computer components.

Very many people are concerned with computer function, in detail.

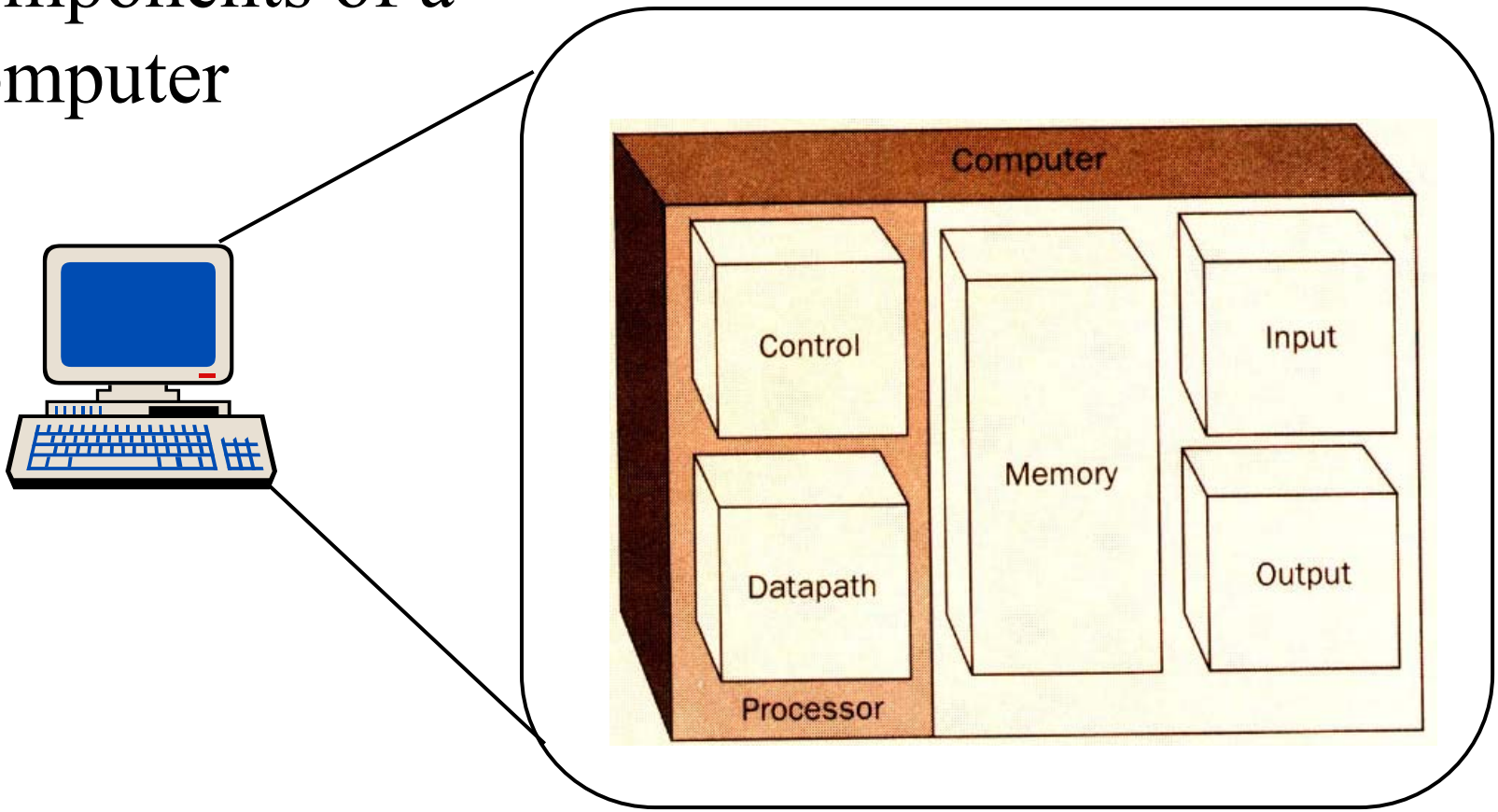
The Big Picture

- What is inside a computer?
- How does it execute my program?

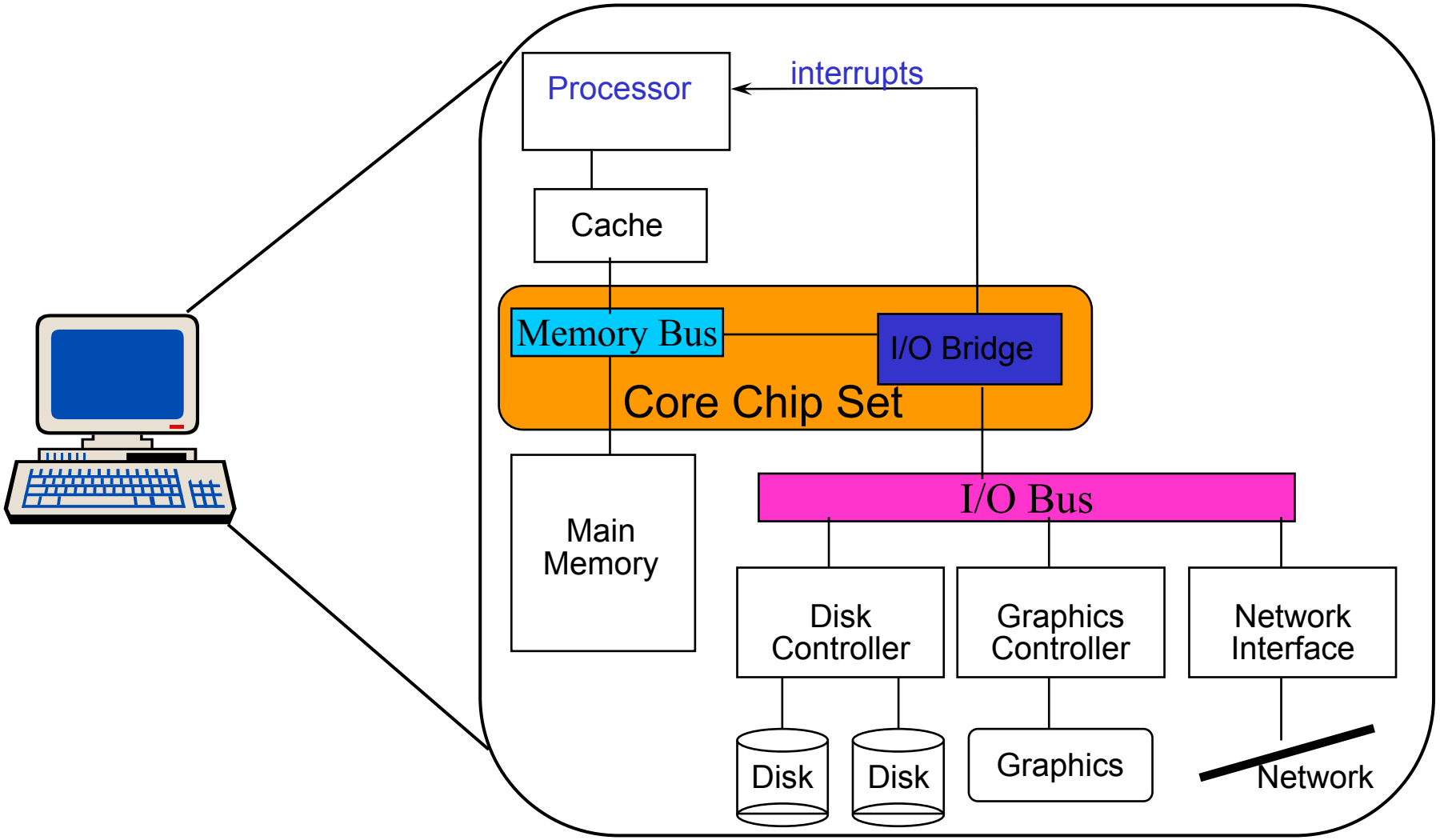


The Big Picture

- The Five Classic Components of a Computer

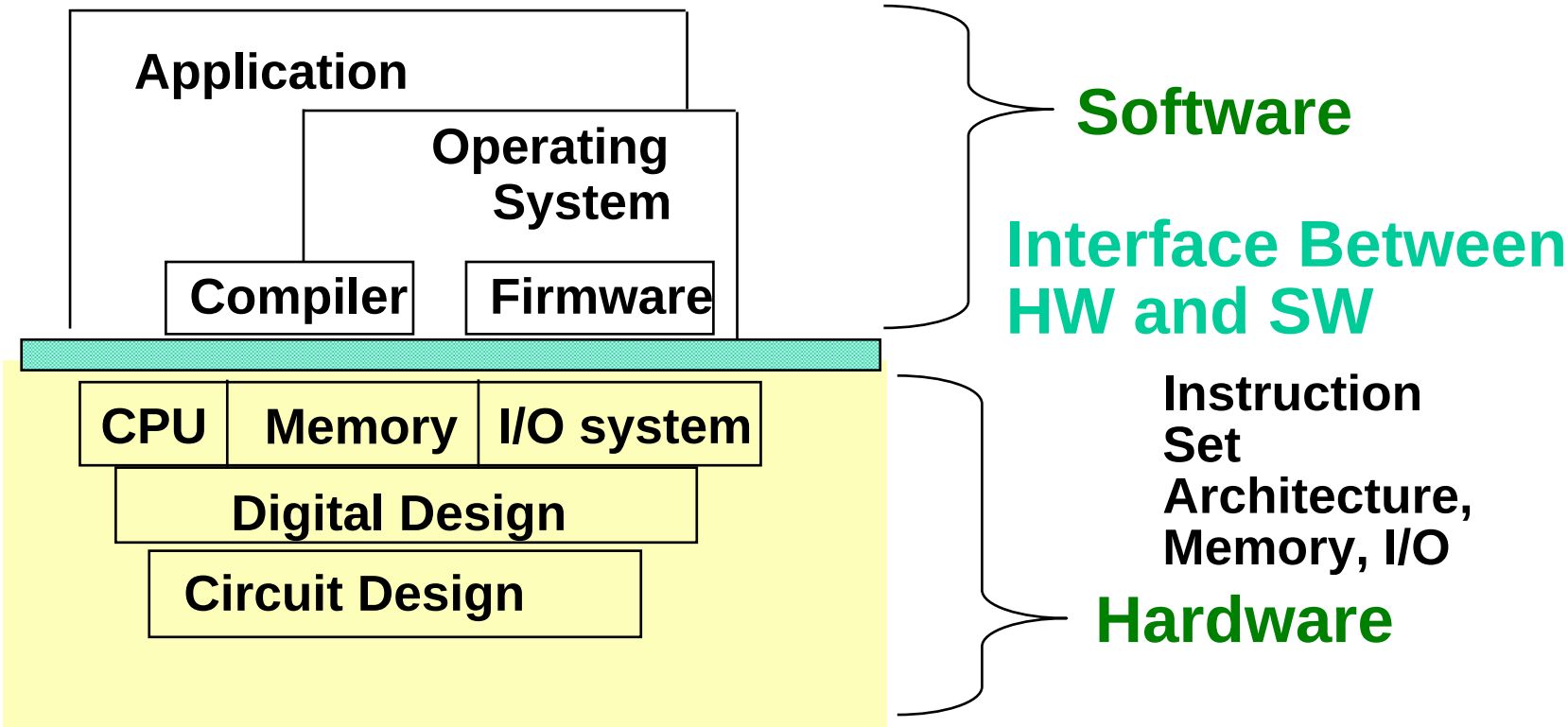


System Organization



What is Computer Architecture?

- **Coordination of levels of abstraction**



- **Under a set of rapidly changing *Forces***

Levels of Representation

High Level Language Program

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

Compiler

Assembly Language Program

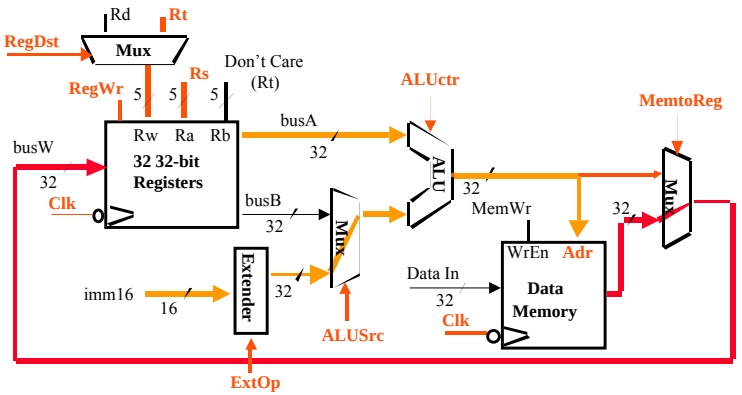
```
lw $15,   0($2)
lw $16,   4($2)
sw $16,   0($2)
sw $15,   4($2)
```

Assembler

Machine Language Program

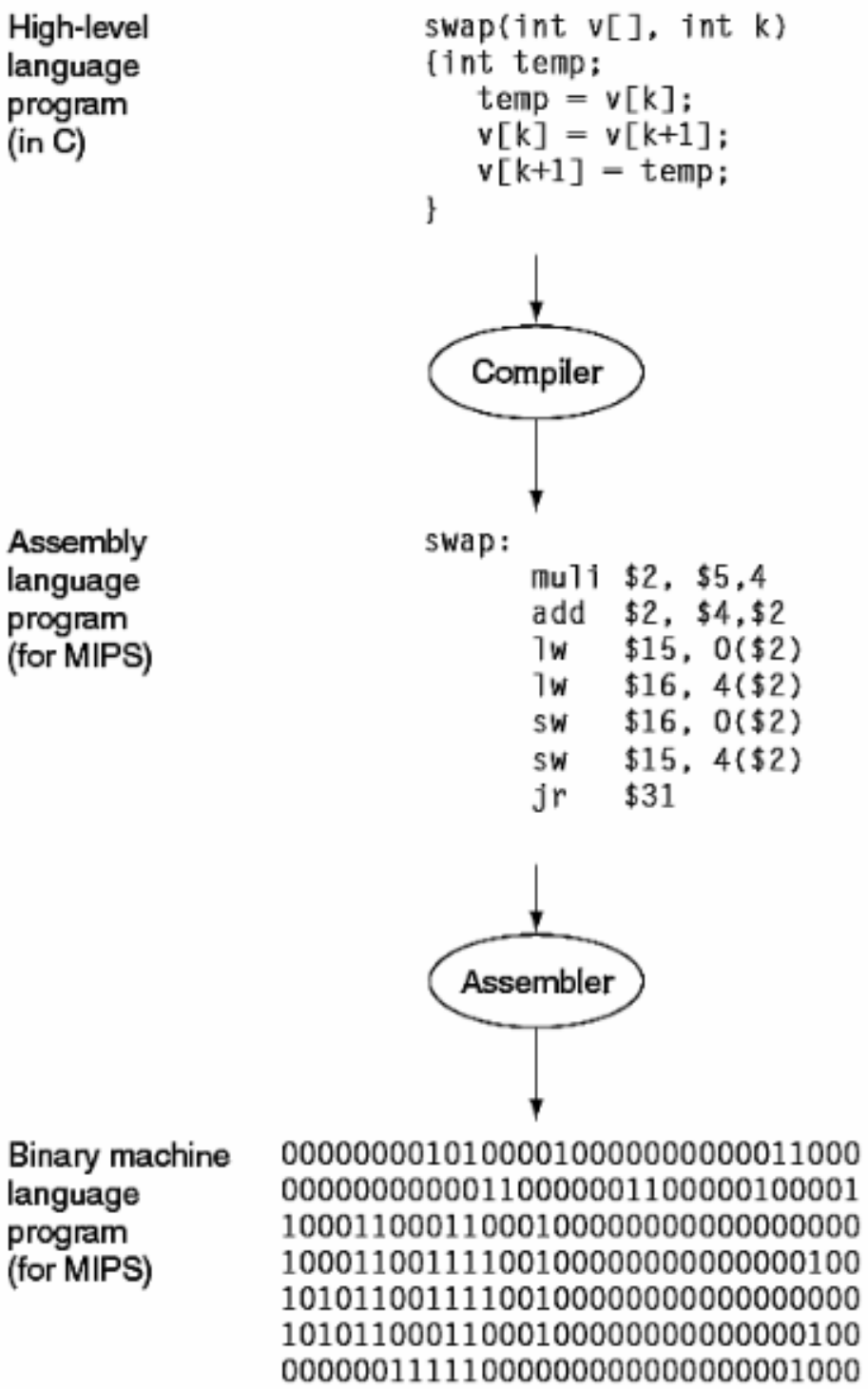
```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

Machine Interpretation

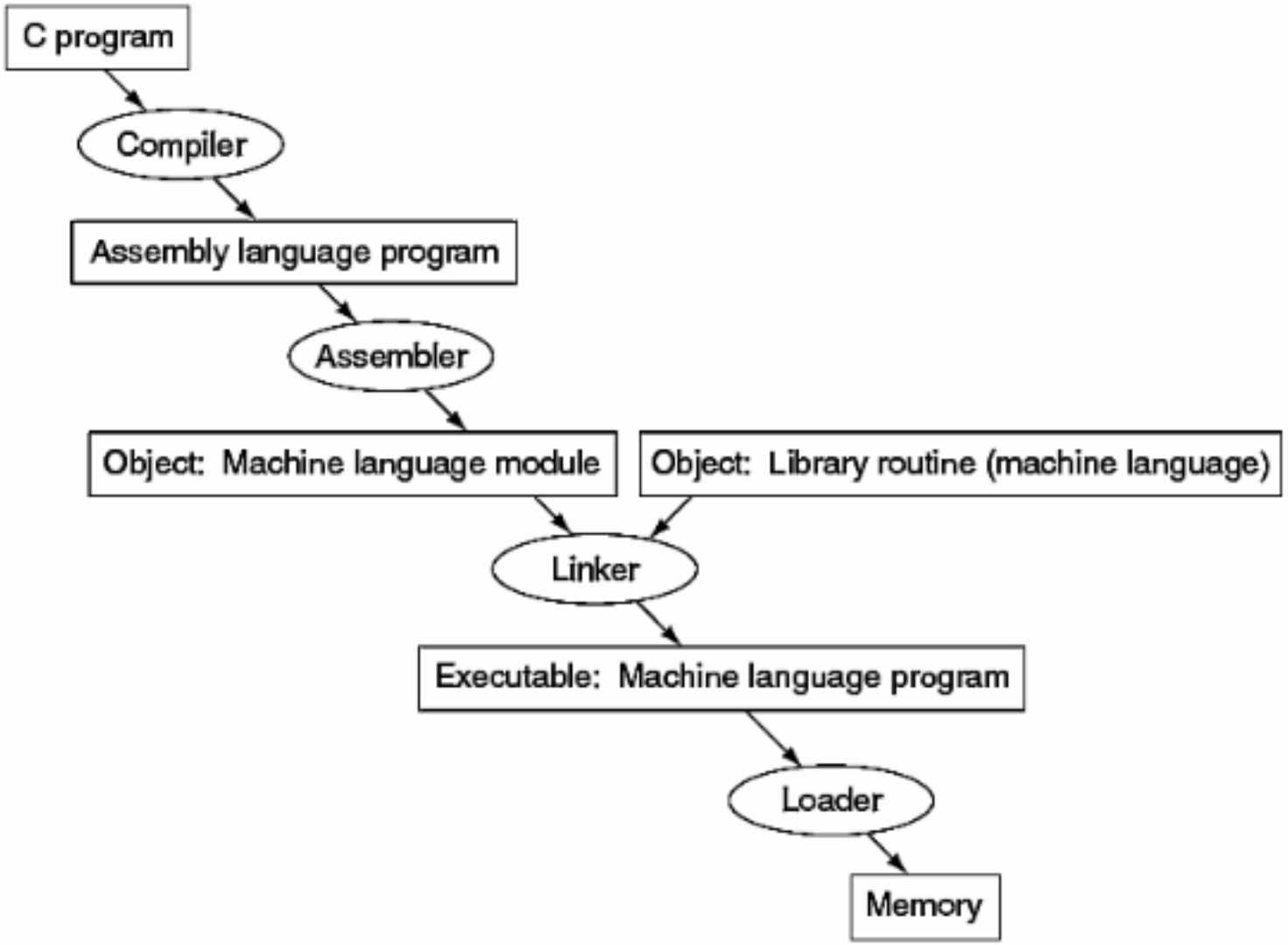


Control Signal Specification

Compiler-Assembler



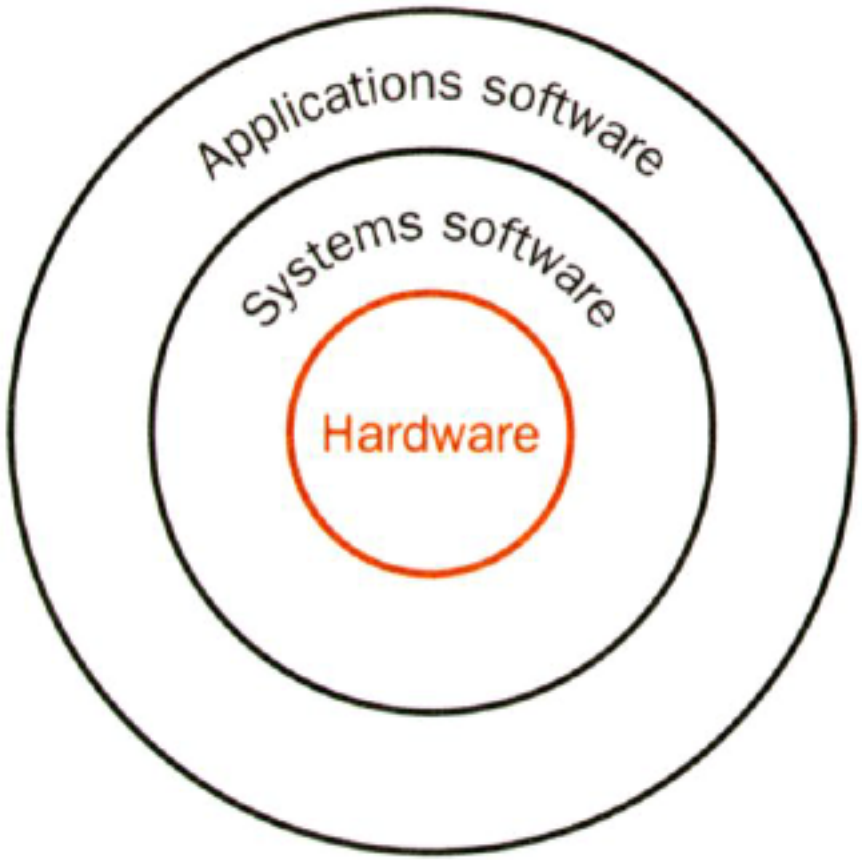
Translation hierarchy for C



Basic Elements

Functional Levels:

- Application Layer
- System Software
- Hardware Layer

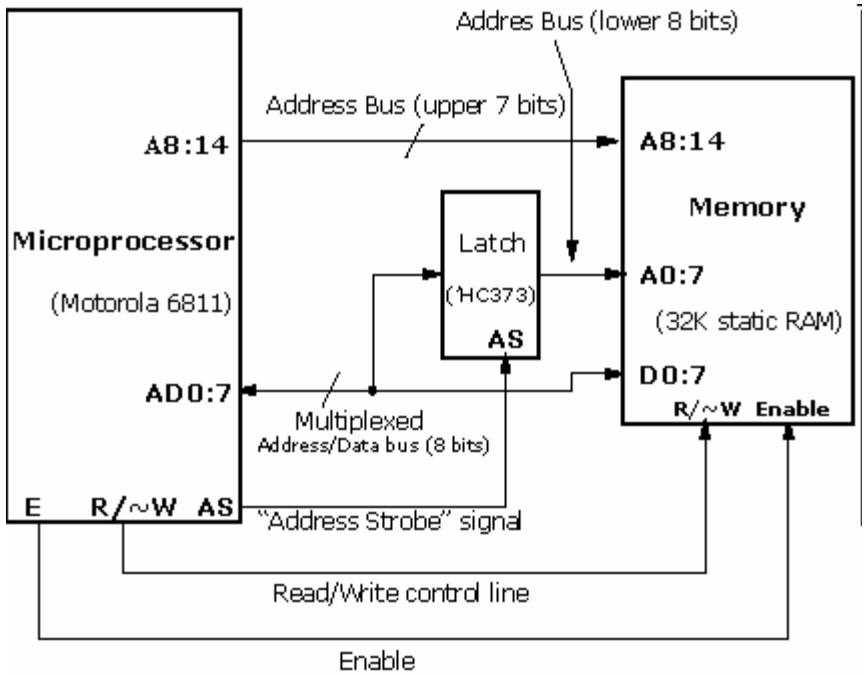


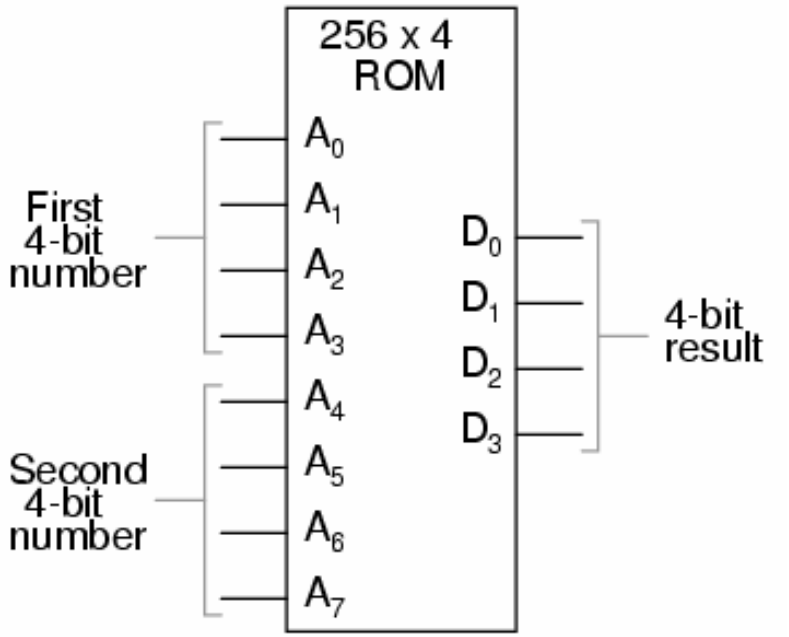
MIPS Assembly

- move $t0, $t1
- add $t0, $zero, $t1
- sll $t1, $a1, 2   (reg $t1=k*4)
- lw $t0=4($t1)   (reg $t0=v[k+1])
- ...

EPROM as a Programmable Logic Device

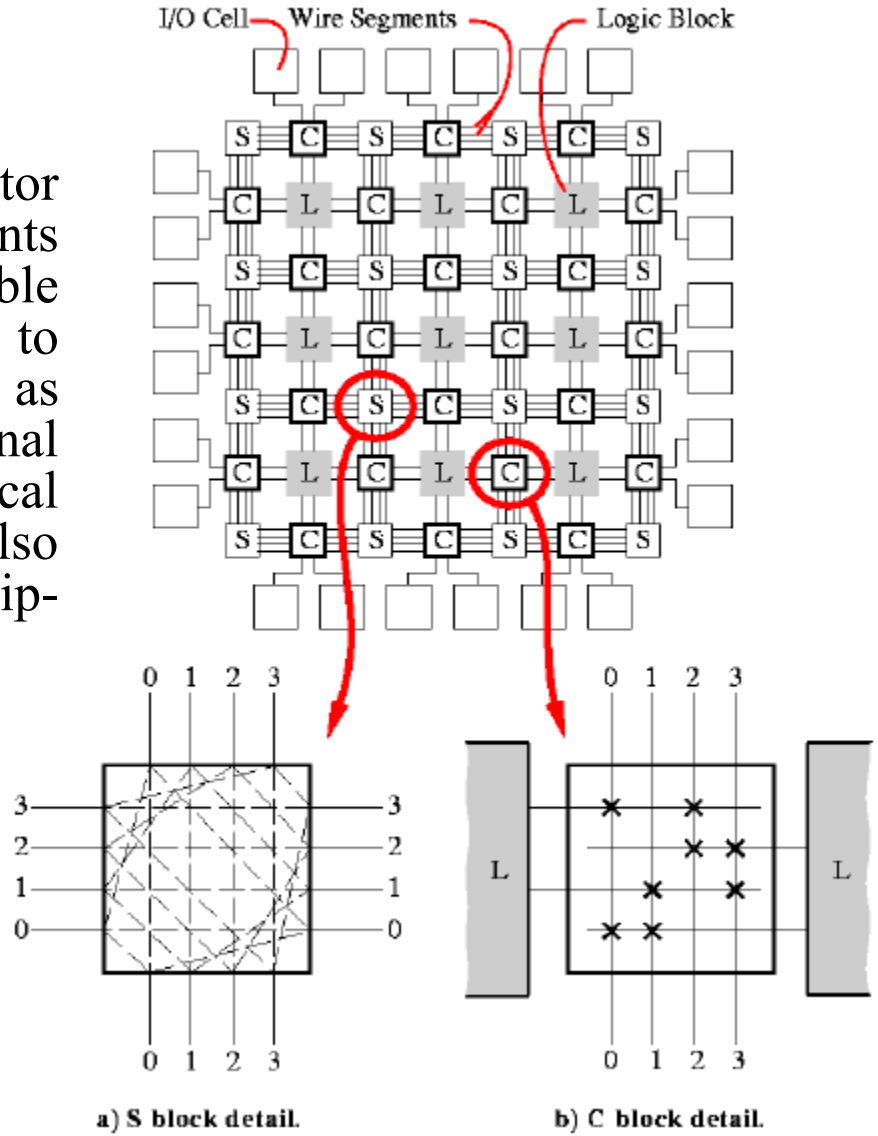
- ROMs are required for applications in which large amount of information needs to be stored in a nonvolatile manner. (Storage for microprocessor programs, fixed table of data, etc.) Another common application of the ROM is for the systematic realization of complex combinational circuits.



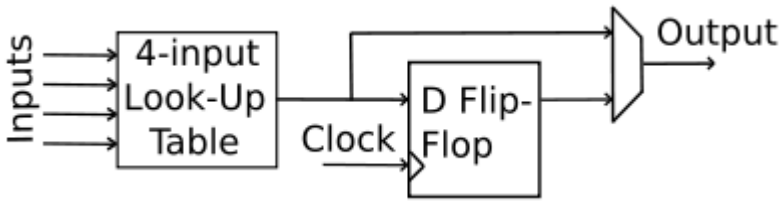


FPGA Design

A field-programmable gate array is a semiconductor device containing programmable logic components called "logic blocks", and programmable interconnects. Logic blocks can be programmed to perform the function of basic logic gates such as AND, and XOR, or more complex combinational functions such as decoders or simple mathematical functions. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.



A classic FPGA logic block consists of a 4-input lookup table (LUT), and a flip-flop:



Soft Processors

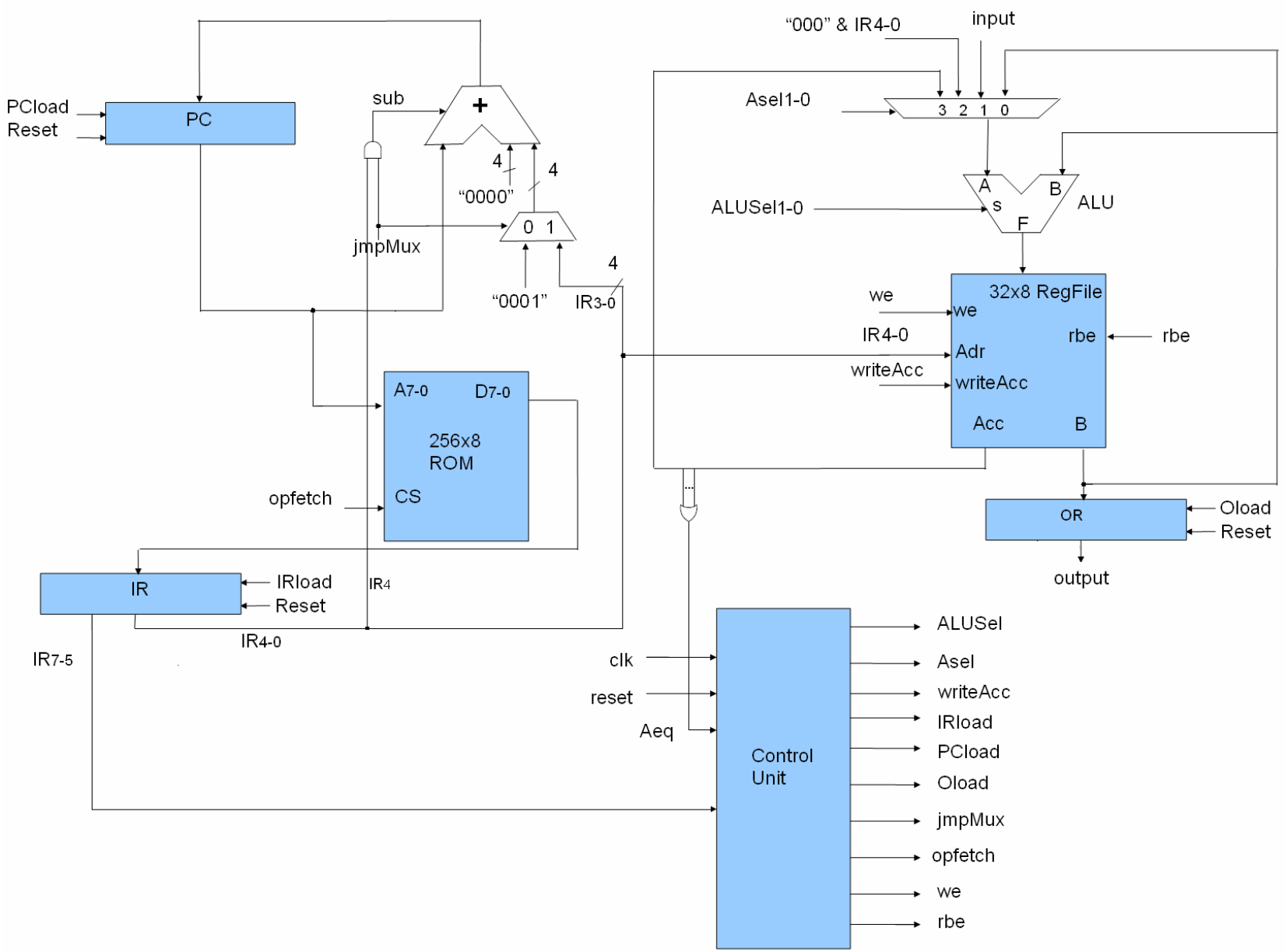
A soft microprocessor (also called softcore microprocessor or a soft processor) is a microprocessor core that can be wholly implemented using logic synthesis. It can be implemented via different semiconductor devices containing programmable logic (e.g., FPGA, CPLD).

Notable soft microprocessors include:

- MicroBlaze
- Nios II

Processor	Developer	Open Source	Bus Support	Notes	Project Home
MicroBlaze	Xilinx	no	OPB, FSL, LMB		Xilinx MicroBlaze
PicoBlaze	Xilinx	no			Xilinx PicoBlaze
Nios, Nios II	Altera	no			Altera Nios II
Cortex-M1	Arm	no			[1]
Mico32	Lattice	yes			LatticeMico32
AEMB	Shawn Tan	yes	Wishbone	MicroBlaze EDK 3.2 compatible Verilog core	AEMB
OpenFire	Virginia Tech CCM Lab	yes	OPB, FSL	Binary compatible with the MicroBlaze	VT OpenFire
PacoBlaze	Pablo Bleyer	yes		Compatible with the PicoBlaze processors	PacoBlaze

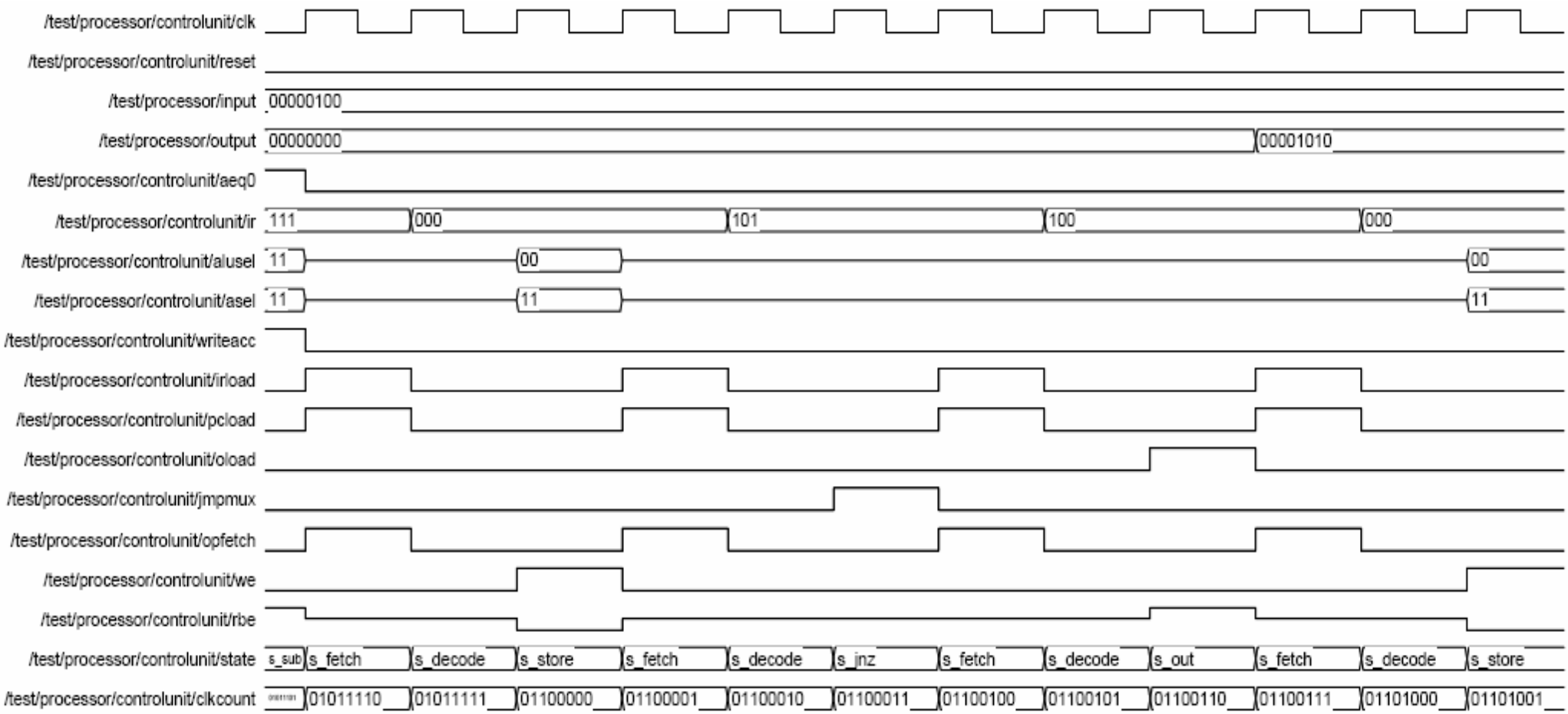
µPabs


PCload
Reset
PC
sub
+
4
4
"0000"
jmpMux
0 1
"0001"
4
IR3-0
"000" & IR4-0
input
Asel1-0
3 2 1 0
A
B
s
F
ALU
ALUSel1-0
we
32x8 RegFile
we
IR4-0
rbe
rbe
Adr
writeAcc
writeAcc
Acc
B
A7-0
D7-0
256x8
ROM
CS
opfetch
OR
Oload
Reset
output
IR
IRload
Reset
IR4
IR4-0
IR7-5
clk
reset
Aeq
Control
Unit
ALUSel
Asel
writeAcc
IRload
PCload
Oload
jmpMux
opfetch
we
rbe

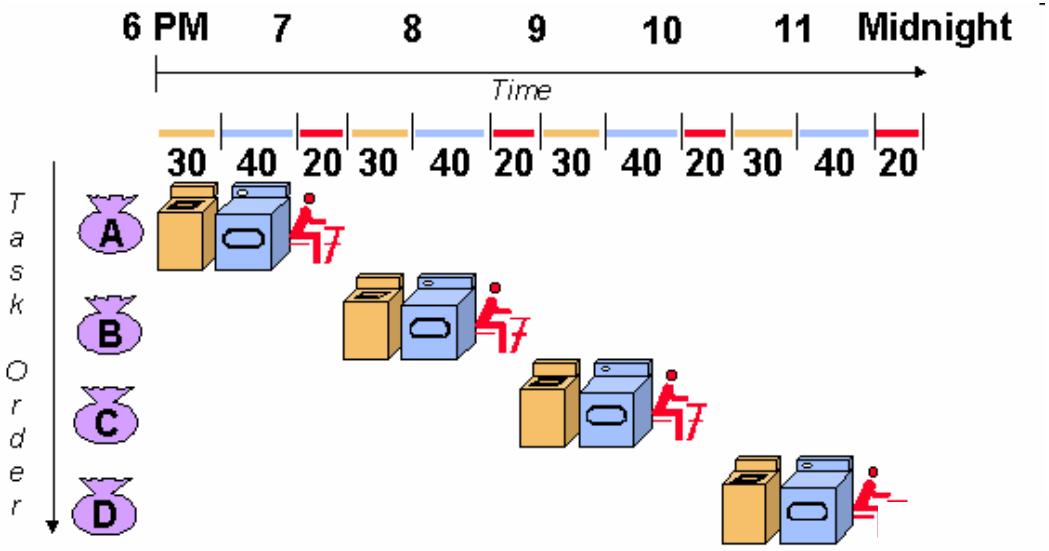
Implementation in VHDL

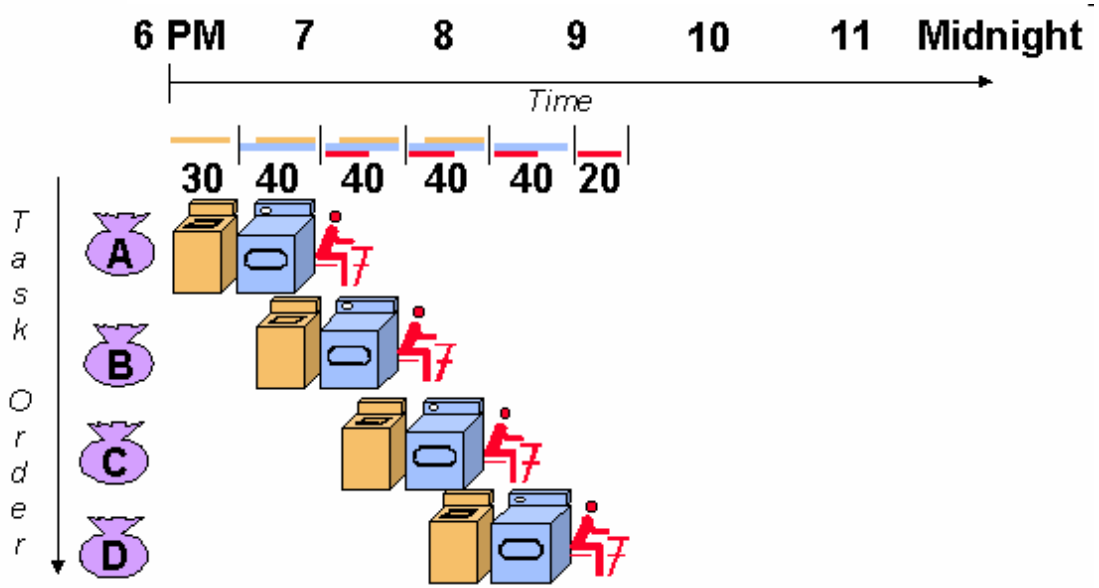
```
23  library ieee;
24  use ieee.std_logic_1164.all;
25  use ieee.std_logic_unsigned.all;
26  use ieee.numeric_std.all;
27
28  entity datapath is port(
29     clk:         in std_logic;
30     reset:       in std_logic;
31     input:       in std_logic_vector(7 downto 0);
32     output:      out std_logic_vector(7 downto 0);
33     -- status signals
34     AeqO:        out std_logic;
35     IROut:       out std_logic_vector(7 downto 5);
36     -- control signals
37     ALUSel:      in std_logic_vector(1 downto 0);
38     Asel:        in std_logic_vector(1 downto 0);
39     writeAcc:    in std_logic;
40     IRload:      in std_logic;
41     PCload:      in std_logic;
42     Oload:       in std_logic;
43     jmpMux:      in std_logic;
44     opfetch:     in std_logic;
45     we:          in std_logic;
46     rbe:         in std_logic);
47  end datapath;
48
49  architecture imp of datapath is
50
51  signal dp_ROMData, dp_IR, dp_IR2, dp_ALU_Out:   std_logic_vector(7 downto 0);
52  signal dp_PC, dp_PCnext, dp_Adder_Out:    std_logic_vector(7 downto 0);
53  signal dp_regfile_A, dp_regfile_B:  std_logic_vector(7 downto 0);
54  signal dp_mux4_Out:  std_logic_vector(7 downto 0);
55  signal dp_mux2_Out:  std_logic_vector(3 downto 0);
56  signal dp_mux2_Out8: std_logic_vector(7 downto 0);
57  signal f_unsigned_overflow:  std_logic;
58  signal sub_jmp: std_logic;
59
60  begin
61  AeqO <= dp_regfile_A(0) or dp_regfile_A(1) or dp_regfile_A(2) or dp_regfile_A(3)
62          or dp_regfile_A(4) or dp_regfile_A(5) or dp_regfile_A(6) or dp_regfile_A(7);
63  dp_IR2 <= "000" & dp_IR(4 downto 0);
64  Bus_Select:          entity work.mux4 port map(Asel, dp_regfile_B, Input, dp_IR2, dp_regfile_A, dp_mux4_Out);
65  Instruction_Register:entity work.IR port map(clk, reset, IRload, dp_ROMData, dp_IR);
66  ProgramCounter:      entity work.PC port map(clk, reset, PCload, dp_PCnext, dp_PC);
67  PC_Mux:              entity work.mux2 port map(jmpMux, "0001", dp_IR(3 downto 0), dp_mux2_Out);
68  dp_mux2_Out8 <= "0000" & dp_mux2_Out;
69  sub_jmp <= jmpMux and dp_IR(4);
70  Adder_8_bit:         entity work.addsub8_pc port map(dp_PC, dp_mux2_Out8, dp_PCnext, sub_jmp);
71  ProgramMemory:       entity work.rom_256_8 port map(opfetch, dp_PC, dp_ROMData);
72  RegisterFile:        entity work.regfile port map(clk, reset, we, writeAcc,
73                                        dp_IR(4 downto 0), dp_ALU_Out, rbe, dp_regfile_A, dp_regfile_B);
74  ALU8:                entity work.ALU port map(ALUSel, dp_mux4_Out, dp_regfile_B,
75                                        dp_ALU_Out, f_unsigned_overflow);
76  OutputRegister:      entity work.OReg port map(clk, reset, Oload, dp_regfile_B, output);
77  IROut <= dp_IR(7 downto 5);
78  end imp;
```

Running the CPU



Pipelining





fetch	decode	execute		
	fetch	decode	execute	
		fetch	decode	execute