EE-216

Computer Architecture

Lecture 1

Introduction to Computer Architecture

Engr. Erum Rehman

Department of Electrical Engineering

College of Engineering &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”

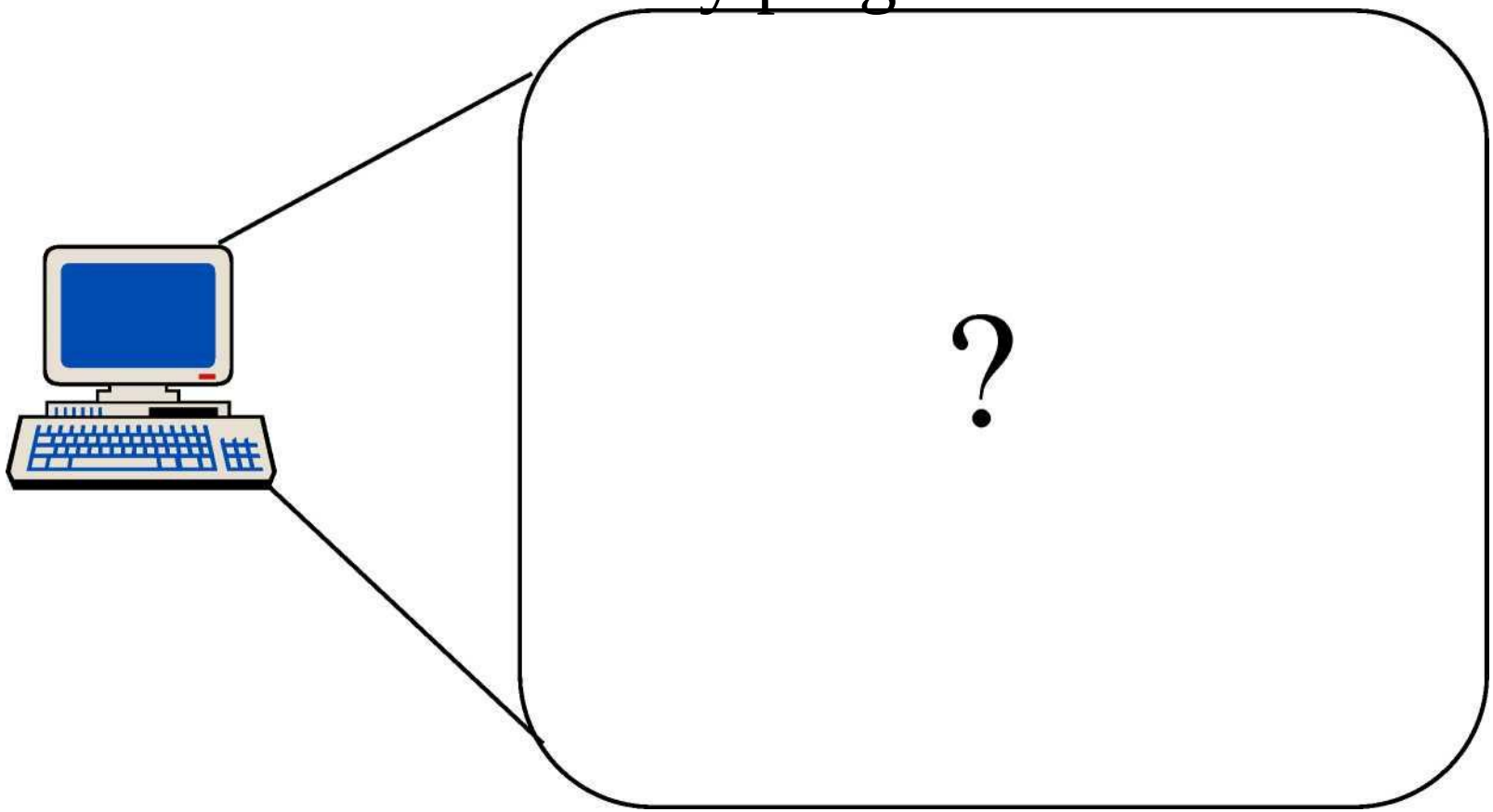
Few people design computers! Very few design instruction sets!

Many people design computer components.

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

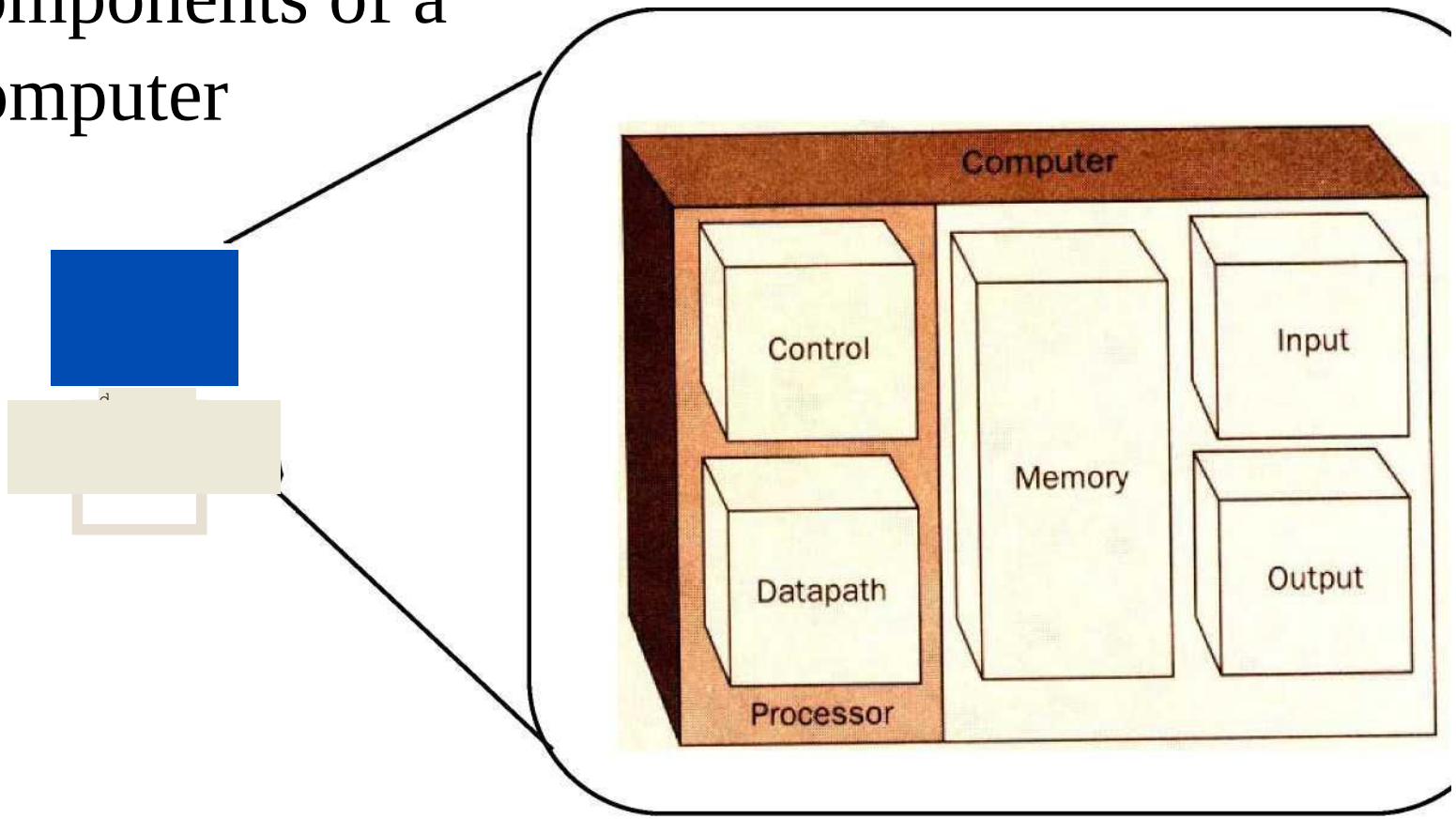
The Big Picture

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

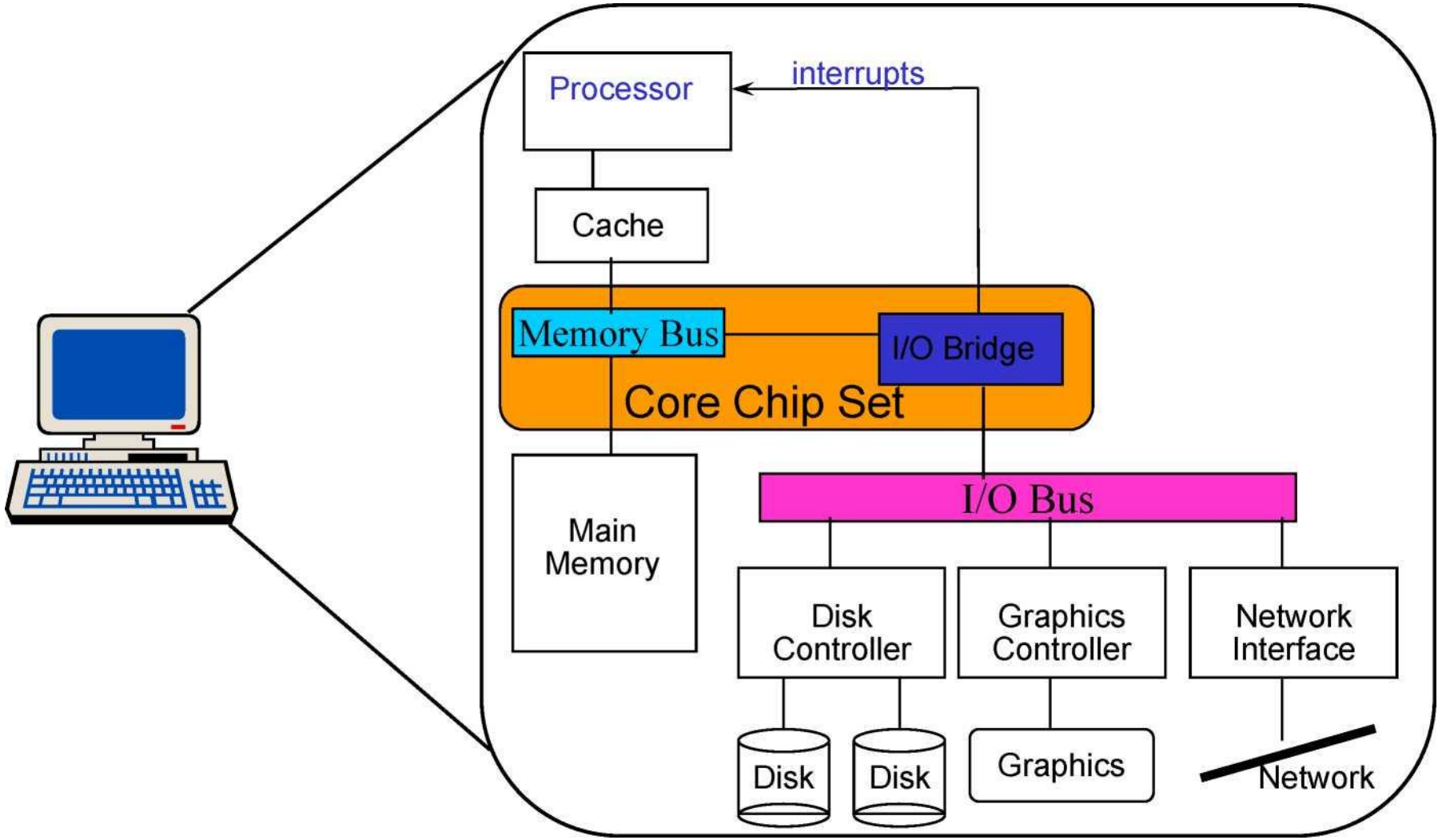


The Big Picture

The Five Classic Components of a Computer

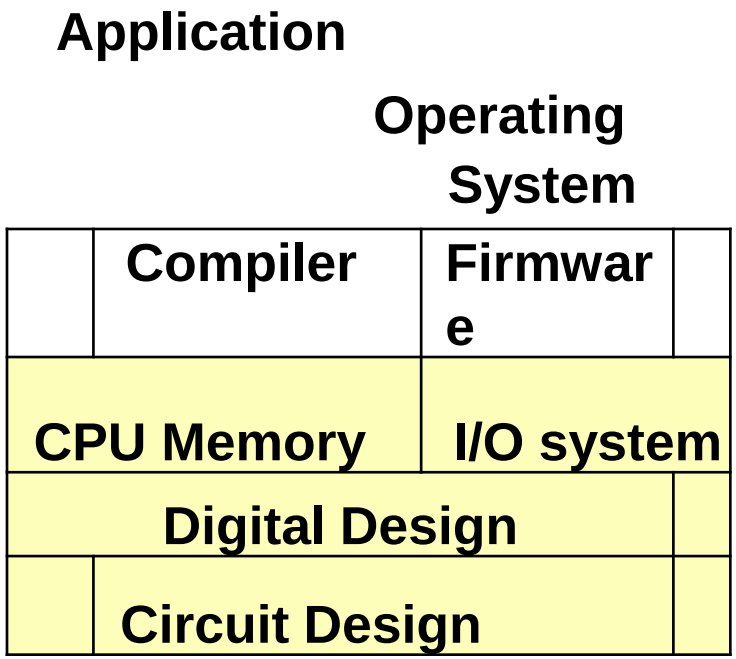


System Organization



What is Computer Architecture?

- **Coordination of levels of abstraction**



Software

Interface Between HW and SW

Instruction Set Architecture, Memory, I/O

Hardware

- **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

1001 1100 0110 1010 1111 0101 1000

1111 0101 1000 0000 1001 1100 0110

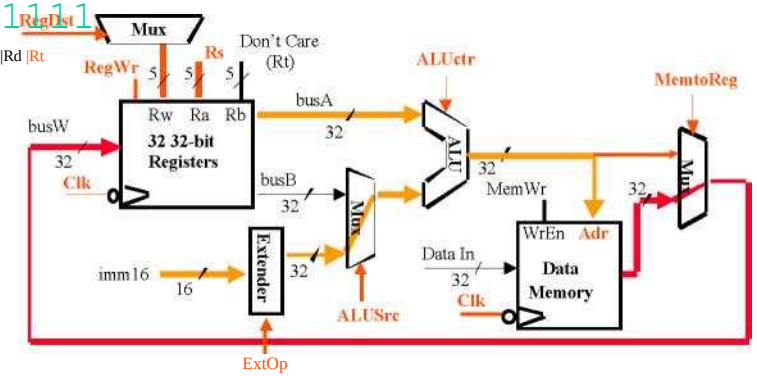
0110 1010 1111 0101 1000 0000 1001

1000 0000 1001 1100 0110 1010 1111

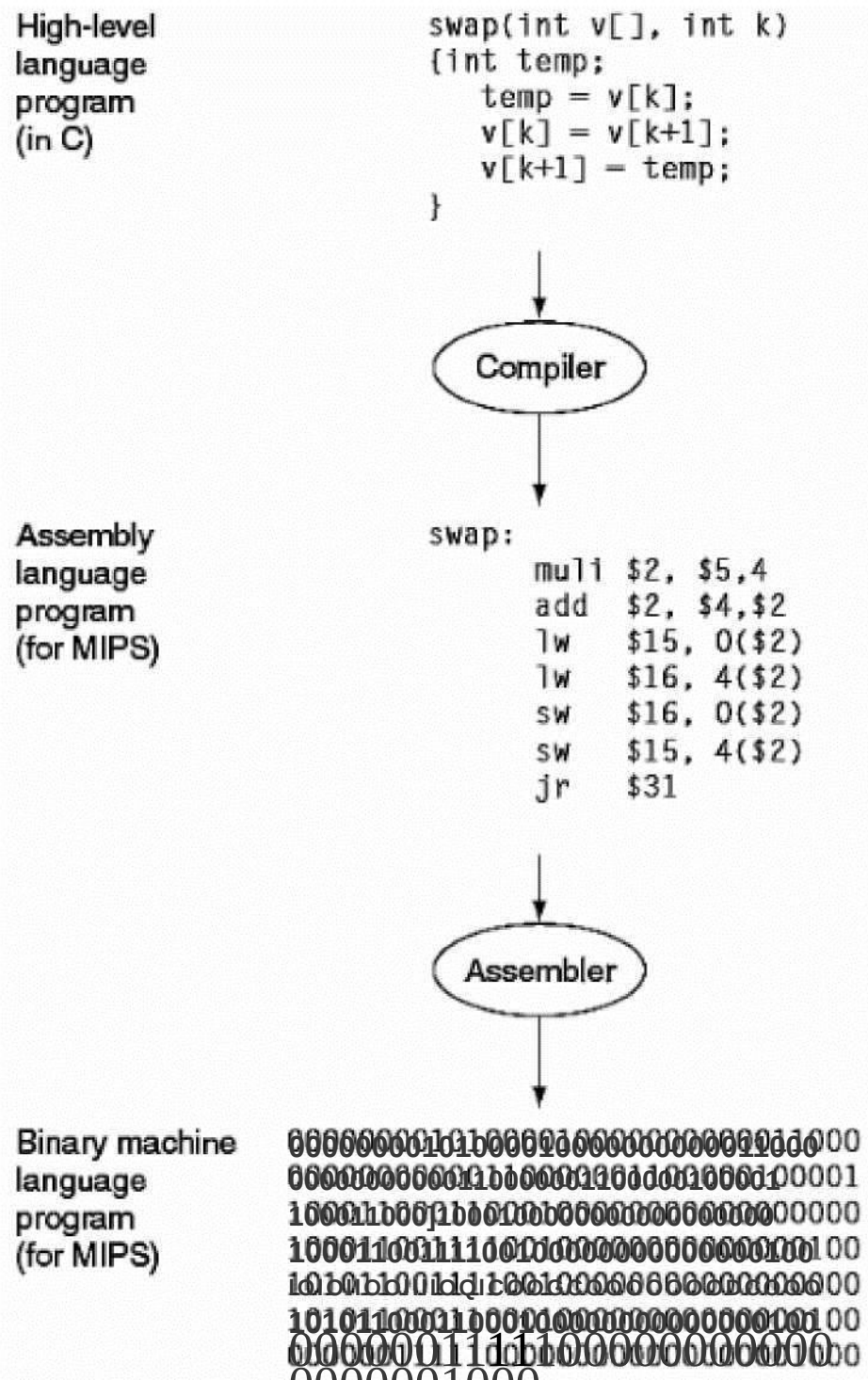
0000 1010 1100 0101

Machine Interpretation

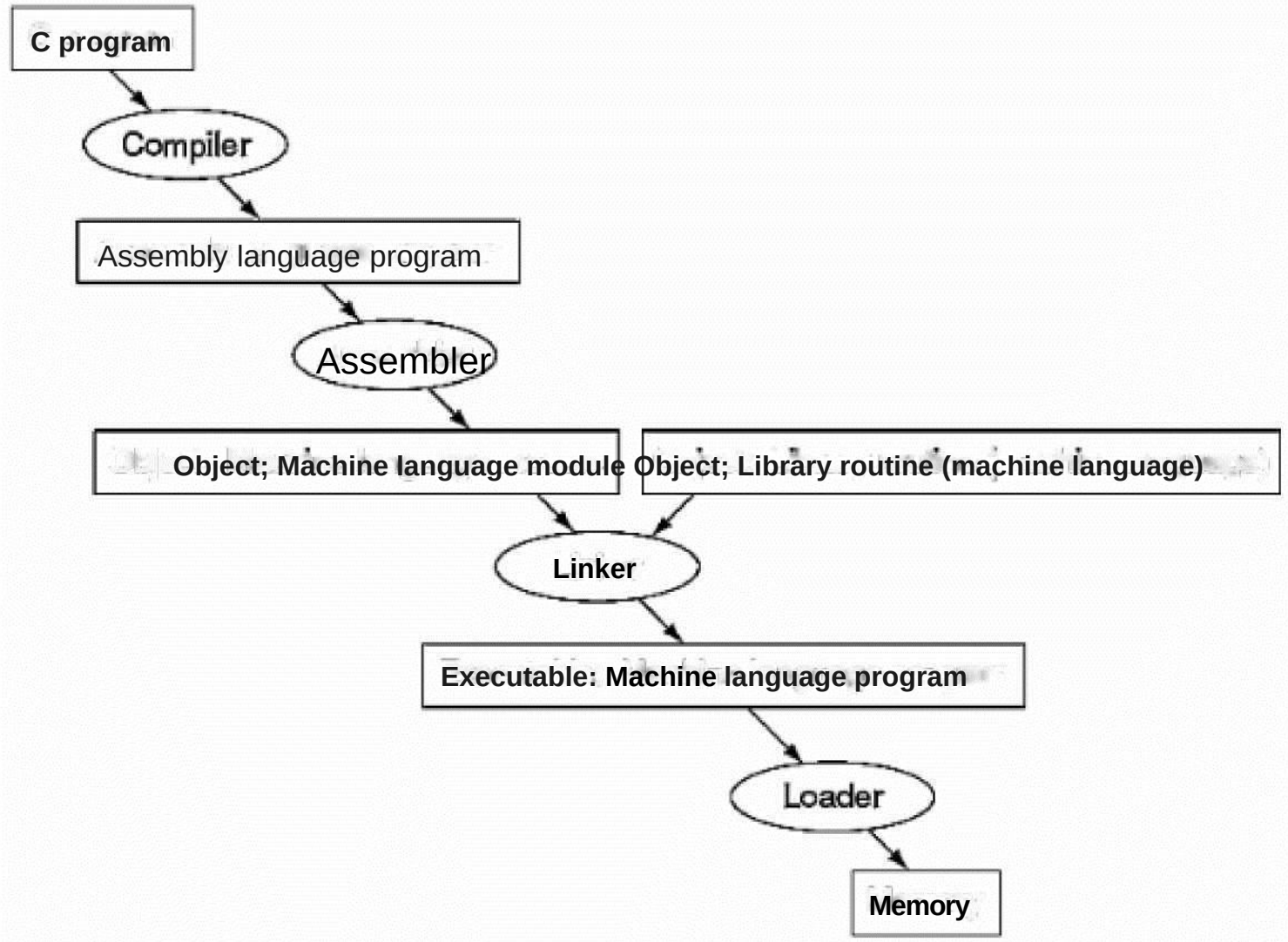
Control Signal Specification



Compiler-Assembler



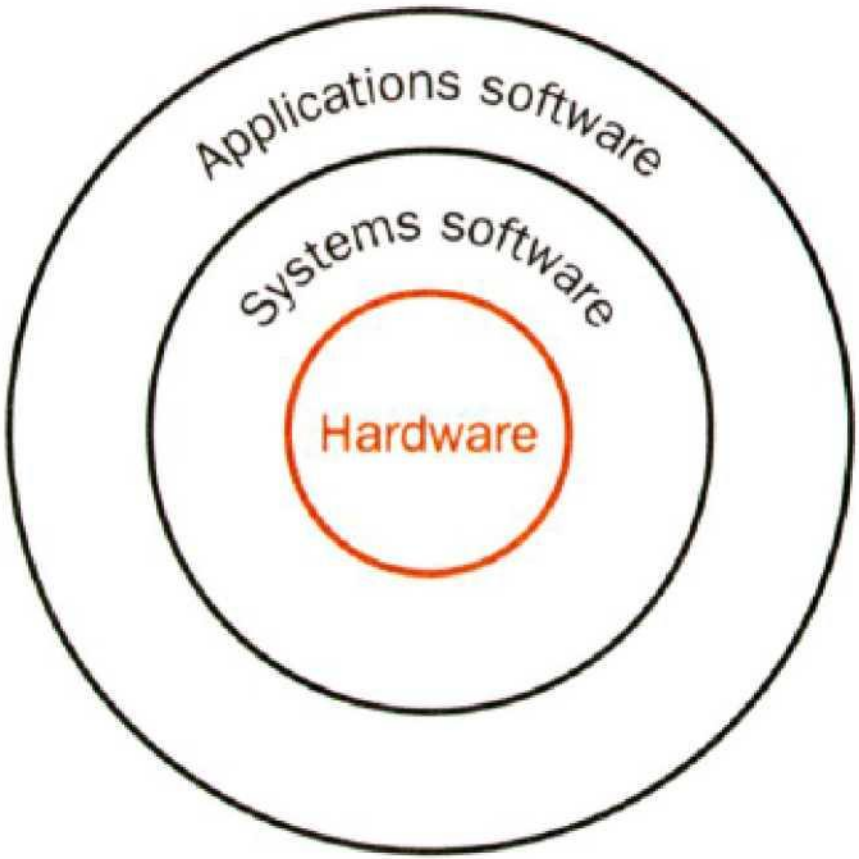
Translation hierarchy for C



Basic Elements

Functional Levels:

.B Application Layer .B System Software B Hardware Layer



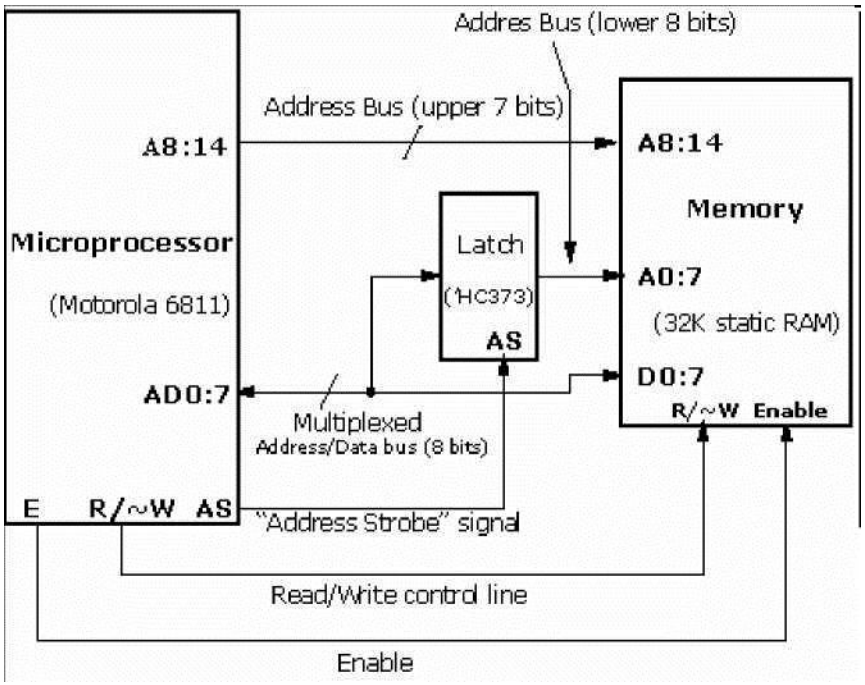
MIPS Assembly

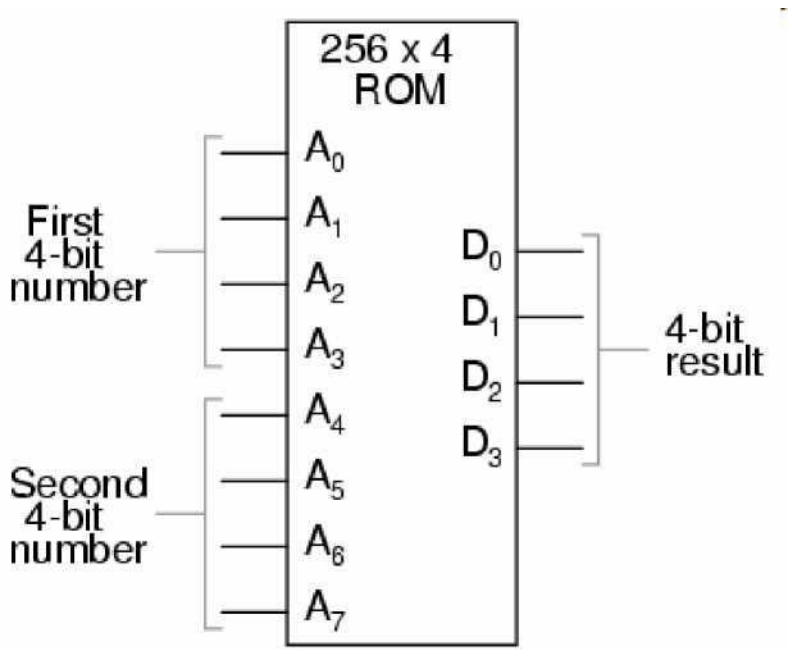
- move $t0, $t1
- add $t0, $zero, $t1
- sll $t1, $a1, 2 (reg $t1=k*4)
- lw $t0=4($t1) (reg $t0=v[k+1])
- ...

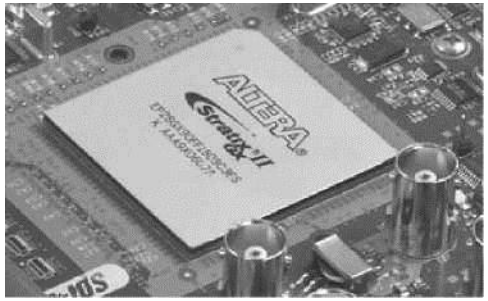
EPROM as a Programmable Logic Device

- **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.

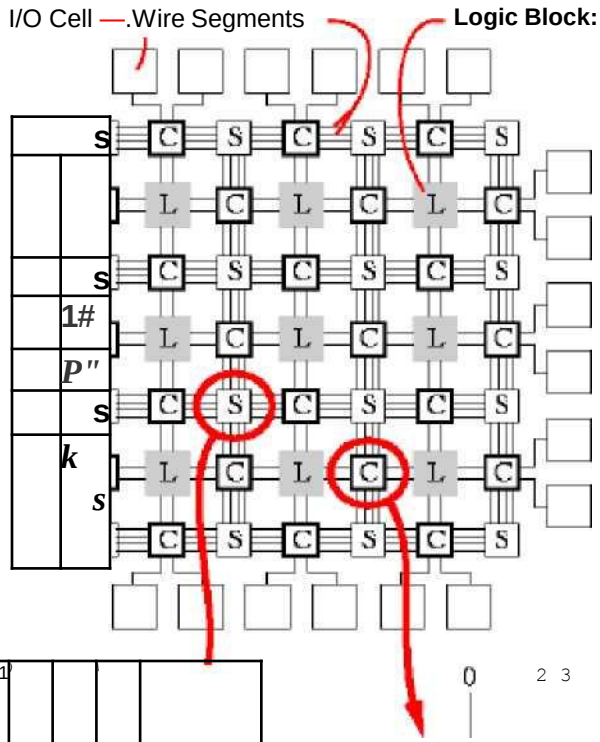




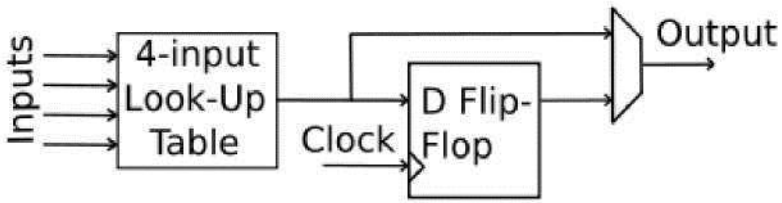


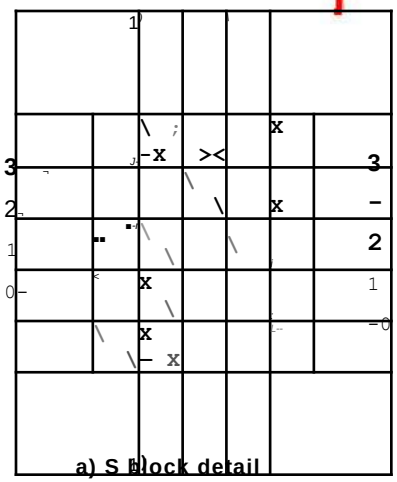
FPGA Design

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

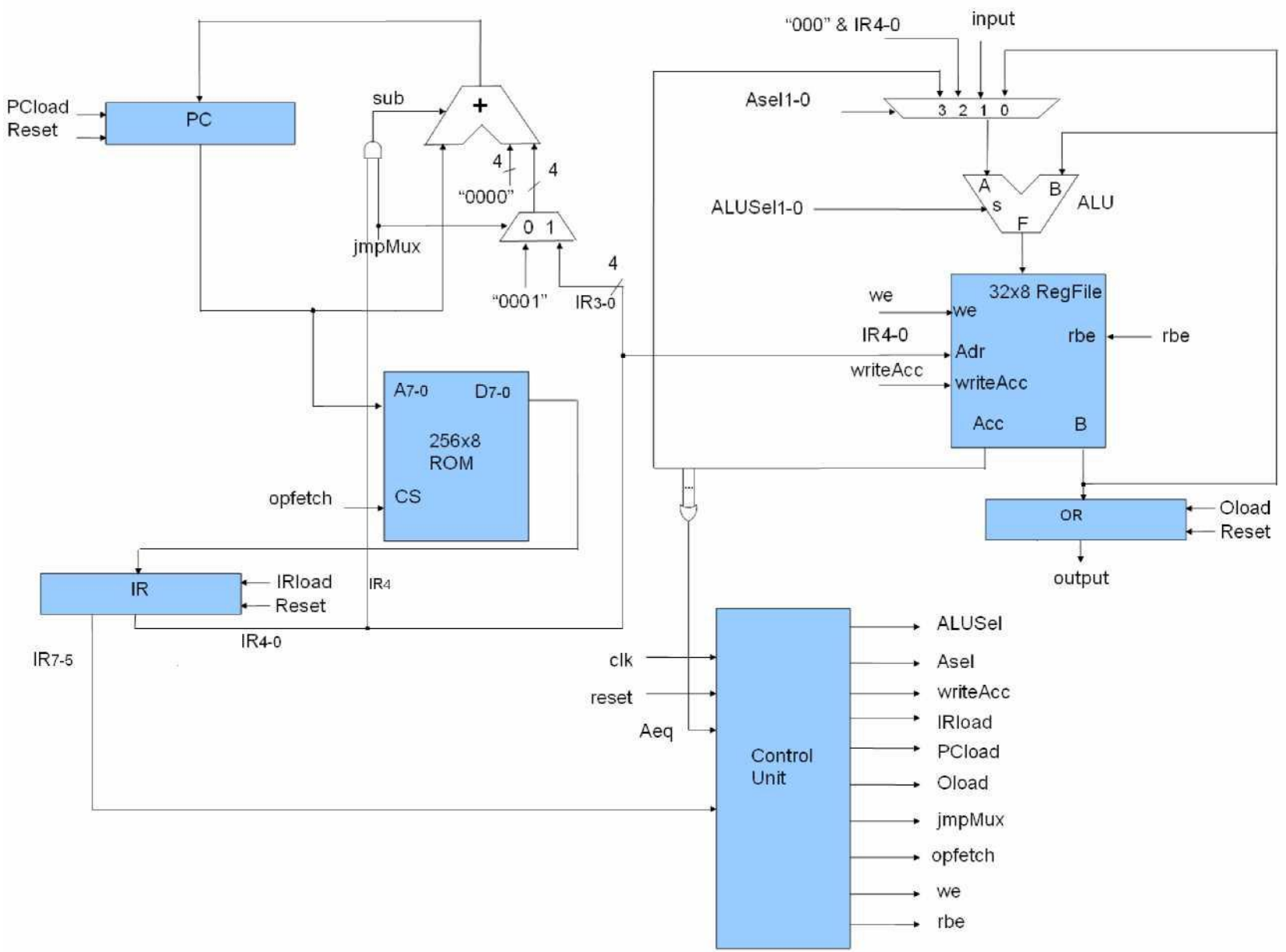
A soft microprocessor (also called softcore microprocessor or a soft processor) is a microprocessor core that can be wholly implemented using logic synthesis. It can be implemented via different semiconductor devices containing programmable logic (e.g., FPGA, CPLD).

Notable soft microprocessors include:

MicroBlaze ■*Nios II

Processor	Developer	Open Source	Bus Support	Notes	Project Home
MicroBlaze	Xilinx	no	OPB. FSL. LMB		Xilinx MicroBlaze ^
Pico Blaze	Xilinx	no			Xilinx PicoBlaze r^1
Nios, Nios II	Altera	no			Altera Nios II
CortBX-M 1	Arm	no			[1]®
Mi co 32	Lattice	yes			Lattice Mico 32
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 Pico Blaze processors	PacoBlaze i*?

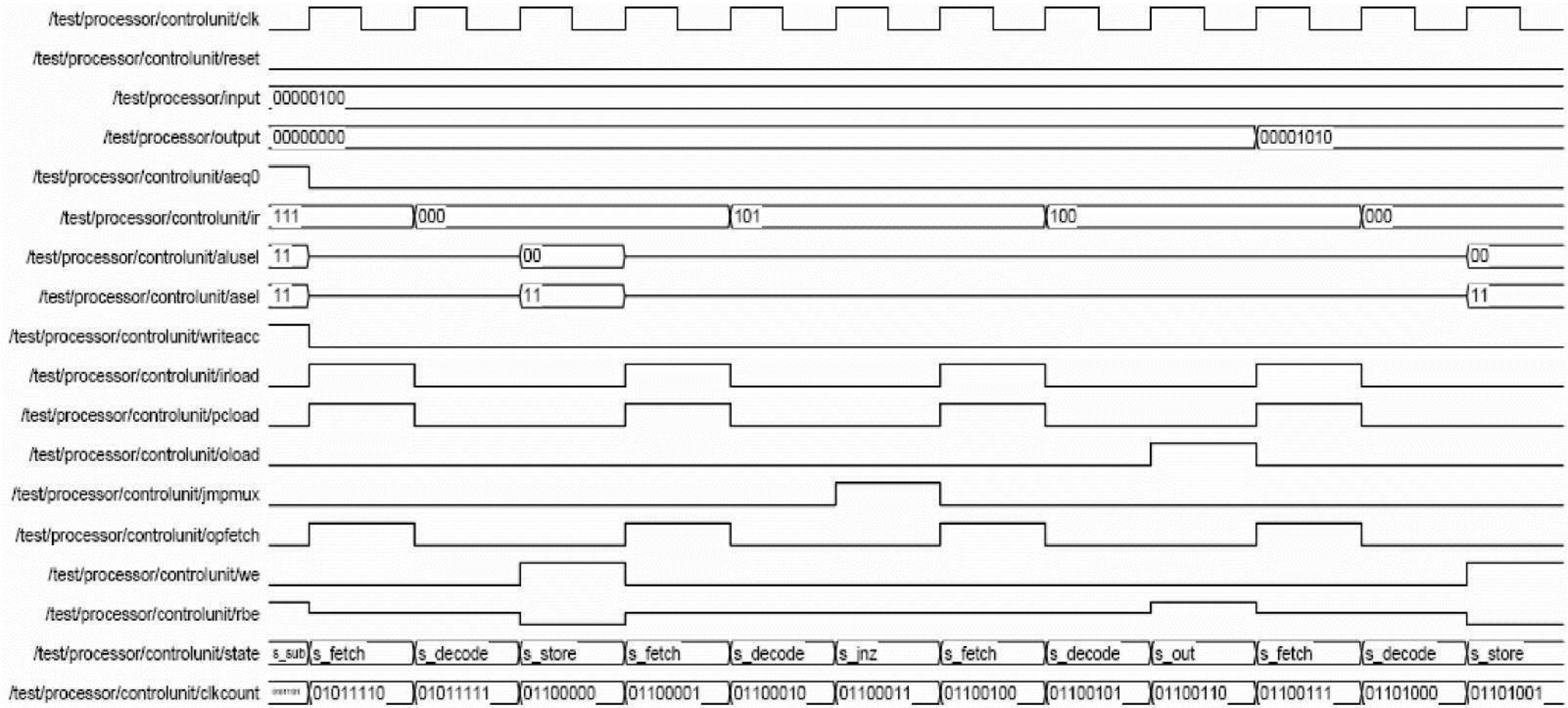
µPabs


PCload
Reset
PC
sub
+
4
4
"0000"
jmpMux
0 1
"0001"
4
IR3-0
"000" & IR4-0
input
Asel1-0
3 2 1 0
ALUSel1-0
A
B
s
F
ALU
we
32x8 RegFile
we
IR4-0
Adr
rbe
rbe
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

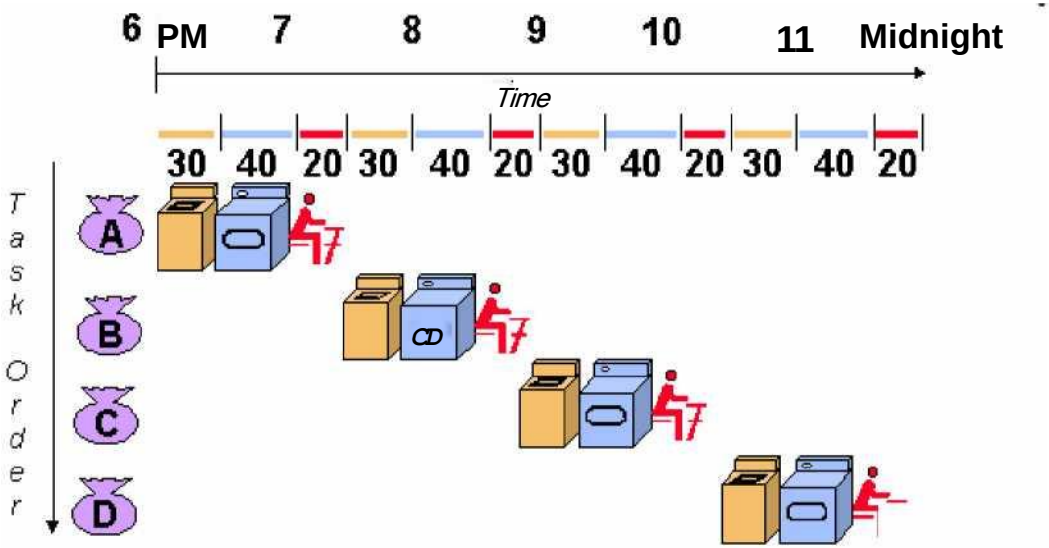
Implementation in VHDL

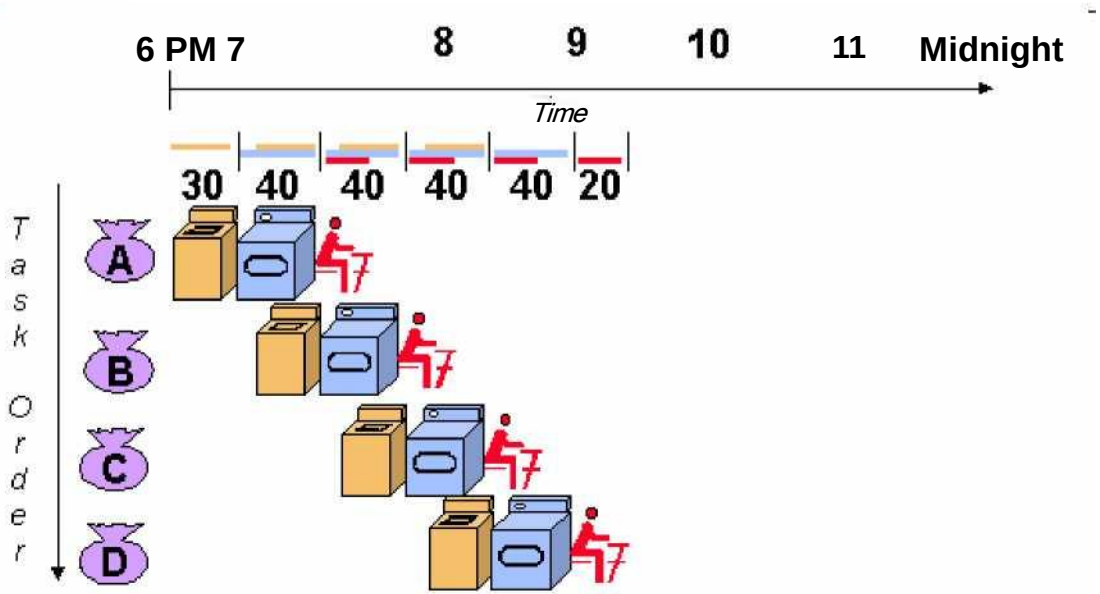
```
23 library ieee;
24  use ieee . std_ 1 ogic_ 1164 . a 11; use
25  ieee.std_logic_unsigned.a11; use ieee.numeric
26  std.all;
27
28  entity datapath is port(
29  elk:    in std_logic;
30  reset:    in std_logic;
31  input:    in std logic_vector(7 downto 0);
32  output:     o ut s t d_logic_vec t or(7     dounto
33  0)
34  —    stat us s i gna1s
35  Ae qO :    o ut s t d_ 1 o g i c;
36  IROut:    out std_logic_vector(7    downto    5)
37  —    c o ntr o1 si gna1s
38      ALUSel:     in std_logic_vector(1 downto 0) ;
39      Asel:       in std_logic_vector(1 downto 0) ;
40      writeAcc:   in std_logic;
41      IRload:     in std_logic;
42      PCload:     in std_logic;
43      Oload:      in std_logic;
44      jrepMux:    in std_logic;
45      opfetch:    in std_logic;
46      we: rtae:   in std_logic;
47  end datapath;   in std logic);
48
49
50  architecture irep of datapath is
51
52  signal dp_ROMData, dp_IR, dp_IR2, dp_ALU_Out:    std_logic_vector(7 downto 0);
53  signal dp_PC, dp_PCnext, dp_Adder_Out:    std_logic_vector(7 downto 0);
54  s i gna1 dp_r e gf i1e_A, dp_r e gf i1e_B: s t d_log i c_vector (7 downto 0); signal
55  dp_reux4_0ut: std_logic_vector(7 downto 0); s ignal dp_mux2_0ut: std_logic_vector(3
56  downto 0); signal dp raux2 Out8: std logic vector(7 downto 0);
57
58  signal  f_unsigned_overflow:  std logic;
59  signal sub jrep: std logic;
60
61   begin
62   AeqO <= dp_regfile_A(□) or    dp_regfile_A(1) or dp_regfile_A(2) or dp_regfile_A(3)
63            or dp_regfile_A(4)    or dp_regfile_A(5) or dp_regfile_A(6) or dp_regfile_A(7) ,
64   dp_IR2 <= "000" £ dp_IR[4    downto 0);
65   Bus_Select:    entity work.reux4 port reap(Asel, dp_regfile_B, Input, dp_IR2, dp_regfile_A, dp_reux4_0ut) ,
66   Instruction_Register:entity work.IR port reap(elk, reset, IRload, dp_ROMData, dp_IR);
67   ProgramCounter:    entity work.PC port reap(elk, reset, PCload, dp_PCnext, dp_PC);
68   PC_Mux:    entity work.reux2 port reap(jrepMux, "0001", dp_IR(3 downto 0), dp_reux2_Out);
69   dp_reux2_Out8 <= "0000" £ dp_reux2_0ut; sub jrep <= jrepMux and dp IR(4);
70
71  Adder_8_b it:          entity work.addsub8_pc port reap(dp_PC, dp_reux2_Out8, dp_PCnext, sub_jrep); entity
72  ProgramHereory:        work.rore_256_8 port reap(opfetch, dp_PC, dp_ROHData); entity work.regfile port reap(elk,
73  RegisterFile:          reset, we, writeAcc,
74                         dp_IR(4 downto 0), dp_ALU_0ut, rbe, dp_regfile_A, dp_regfile_B); entity work.ALU port
75  ALUS:                  reap(ALUSel, dp_reux4_0ut, dp_regfile_B, dp_ALU_Out, f_unsigned_overflow);
76                         entity work.OReg port reap(elk, reset, Oload, dp_regfile_B, output); downto 5);
77  OutputRegister:
78  IROut <= dp_IR(7
    end irep;
```

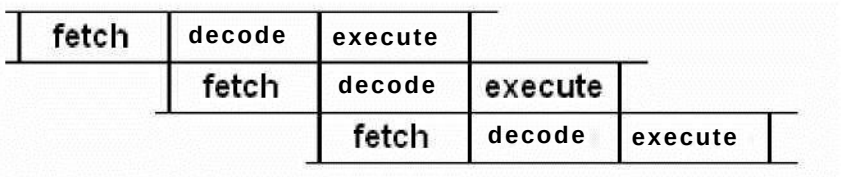
Running the CPU



Pipelining


6 PM
7
8
9
10
11
Midnight
Time
30 40 20 30 40 20 30 40 20 30 40 20
Task Order
A
B
C
D


6 PM 7
8
9
10
11
Midnight
Time
30 40 40 40 40 20
Task Order
A
B
C
D


fetch
decode
execute
fetch
decode
execute
fetch
decode
execute