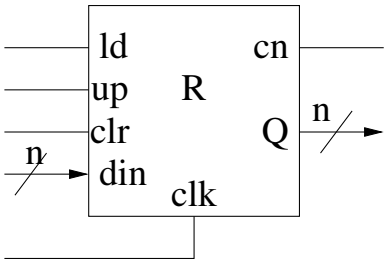
24.1 Multi-Function Sequential Devices

The basic storage register only provides one function: the storage of a binary sequence. Many registers can be augmented with additional capabilities such as resetting to 0, incrementing, decrementing and shifting.

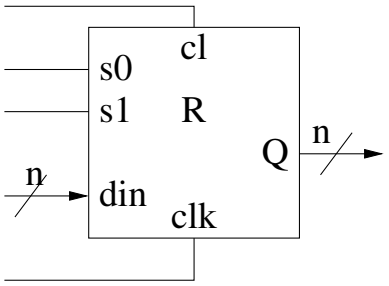
1. UP Counter (with parallel load and clear):



clk	cl	ld	up	function
↑	0	0	0	"none", R unchanged
↑	0	0	1	$R \leftarrow (R + 1) \bmod 2^n$
↑	0	1	X	$R \leftarrow din$
↑	1	X	X	$R \leftarrow 0$

$$cn = (Q = 2^n - 1)\ \&\&\ (up = 1)$$

2. Shift Register (with parallel load and clear):



clk	cl	s1	s0	function
↑	0	0	0	"none", dout = R
↑	0	0	1	$R \leftarrow R << din \bmod n$
↑	0	1	0	$R \leftarrow R >> din \bmod n$
↑	0	1	1	$R \leftarrow din$
↑	1	X	X	$R \leftarrow 0$

25 MEMORY COMPONENTS

The digital systems for implementing external memory and internal memory are different. External memory employs SRAM and DRAM technology to provide large amounts of random access memory at low cost. The price paid for this economy is a slow access time and is the source of the von Neumann bottleneck.

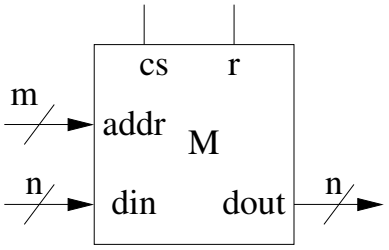
Because of the small size of internal memory, fast bipolar technologies can be employed and this memory is typically implemented with a register file. This device is able to output the contents of two locations while simultaneously storing a value in a third location. This property matches the the structure of the typical binary instruction; that is, one with two source operands and a destination operand and is in fact the motivation for its design.

Both external and internal memory can be modelled abstractly by a 1-dimensional array, where addresses provide the subscripts that specify the locations where values are stored in the array.

The behavioural description of digital systems that are used to represent memory are as follows:

1. **External Memory (M)**

   Entity Definition:



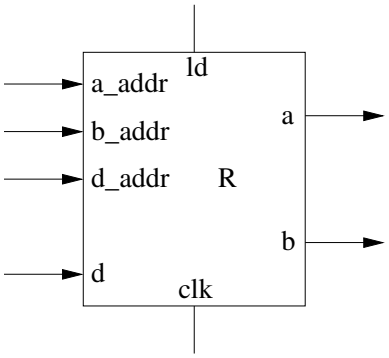
   Functional Specification

cs	r	function
0	0	dout = ZZ...Z
0	1	dout = ZZ...Z
1	0	M[addr] ← din
1	1	dout ← M[addr]

2. **INTERNAL MEMORY**

   In the simplest designs, individual storage registers can be used to provide the temporary working storage within the CPU. However, modern CPU's provide a larger number of storage locations (16 or 32) using a "**Register File**" defined as defined on the next page:

Entity Definition:



Functional Specification:

clk	ld	function
↑	0	a = R[a_addr], b = R[b_addr]
↑	1	R[d_addr] ← d a = R[a_addr], b = R[b_addr]

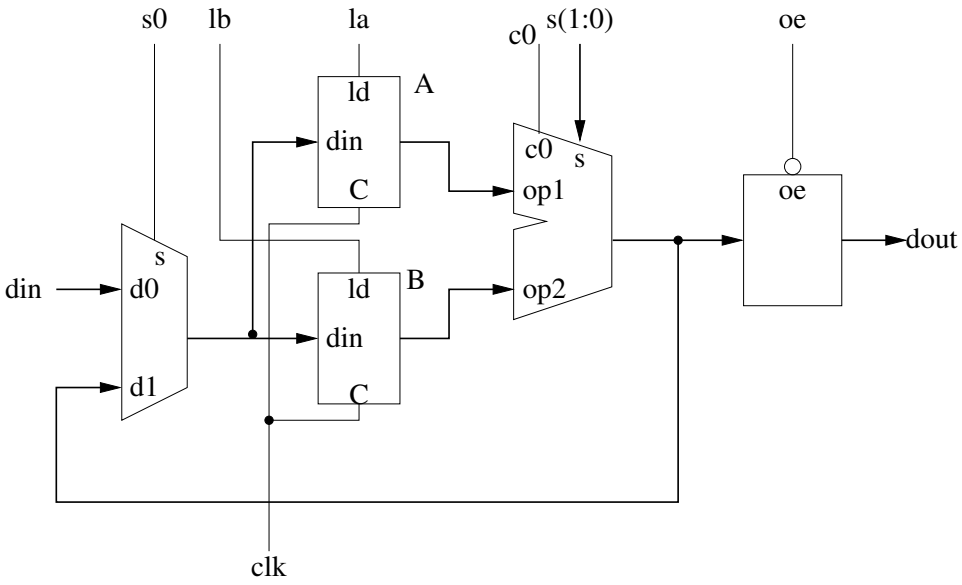
Inspection of this behavioural description reveals that this memory device can store a value and simultaneously retrieve two values from its memory. Therefore three address ports are required: `a_addr, b_addr, d_addr`. The first two determine which two registers are selected for output at porst `a` and `b` respectively. The third specifies the register destination of the data input value on input port `d`.

26 ANALYSIS of COMPLEX SYSTEMS

Analysis of a complex digital system addresses the problem of constructing the function select table that describes the behaviour of the system, given a schematic. A systematic approach to this is:

1. Identify the behavioural descriptions of all components.
2. Label all register and memory components (use capital letters).
3. Label all external signals (use small letters)
4. Construct the control word format for the system. This is an ordered list of all the control inputs in the schematic. Any order is acceptable, but once an order has been defined, it is important to view that order as identifying the role of each bits in a binary sequence.
5. For each control word, express the system's behaviour in register transfer notation.

EXAMPLE:



In this example, the components are a 2 × n multiplexer, two storage registers, an ALU, and a bus buffer. All of these components except the bus buffer are defined above. The functional specification of the bus buffer is given by:

oe	function
0	dout = din
1	dout = ZZ...Z

This device is useful for disconnecting the output of the ALU from the external output `dout`.

The following table represents a partial functon select table as it only identifies the effect of some of the 2^6 possible assignment of values to the control inputs:

oe	s0	lb	la	c0	s(1)	s(0)	function
0	0	0	0	0	0	0	dout = A + B
1	0	0	0	0	0	1	dout = ZZ...Z
1	0	1	0	0	0	1	dout = ZZ,,,Z, B ← din
1	1	1	0	0	0	1	dout = ZZ...Z, B ← A + B
1	1	0	1	0	0	1	dout = ZZ...Z, A ← A + B
1	1	0	1	1	0	1	dout =ZZ...Z, A ← A + $\overline{\text{B}}$ + 1 = A − B
1	0	1	0	1	0	1	B = din

27 CPU DESIGN

CPU design begins with a specified instruction set architecture. To illustrate the design techniques, the following instruction set architecture will be used. It is based on a small subset of a commercial chip, the x86-64.

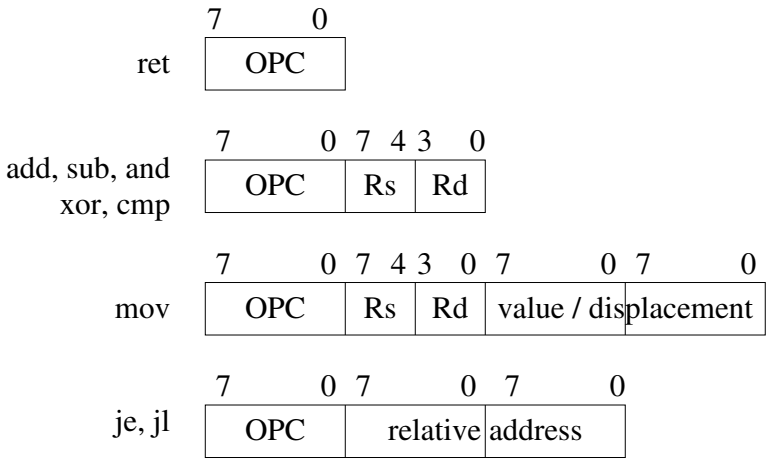
The following summarizes the features of the instruction set architecture to be used:

1. 2-operand machine
2. Instruction set size: 256

3. External memory size: $2^{64} \times 8$ (M)
4. Internal memory size: $2^4 \times 64$ (R)
5. Instruction Set:

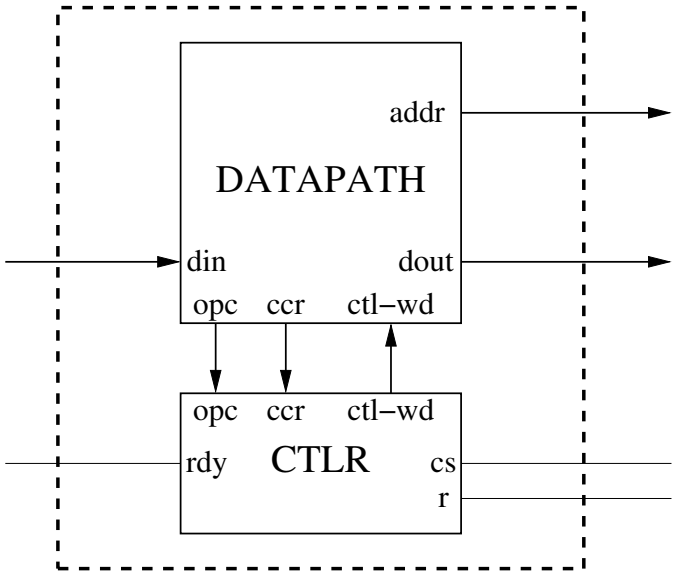
OPC	INST	OPNDS	MODE	SEMANTICS
0x40	mov	$value, Rd		R[d] ← value
0x41	mov	Rs, value(Rd)		M[value + R[d]] ← R[s]
0x42	mov	value(Rd), Rs		R[s] ← M[value + R[d]]
0x60	add	Rs, Rd		R[d] ← R[d] + R[s]
0x61	sub	Rs, Rd		R[d] ← R[d] - R[s]
0x62	and	Rs, Rd		R[d] ← R[d] & R[s]
0x63	xor	Rs, Rd		R[d] ← R[d] ^ R[s]
0x72	jl	addr		if SF then PC ← PC + value where value = addr−(*+3)
0x73	je	addr		if ZF then PC ← PC + value where value = addr−(*+3)
0x80	cmp	Rs, Rd		SF ← msb(R[d] - R[s]) ZF ← (R[d] = R[s])
0x90	ret			PC ← R[sp]

6. Instruction formats:



28 BASIC CPU ORGANIZATION

The typical complex digital system, including a CPU, is partitioned into two components, the "datapath" and the "controller." In the following diagram, the hatched box represents the CPU entity.



The interface between the datapath and controller components consists of three significant buses:

- The `opc` bus allows the datapath to pass the opcode of an instruction to be passed to the controller after it is input to the datapath vain the `din` bus.
- The `car` bus provides the controller with status information from the "flags" or "condition codes" register that is part of the datapath.
- The **ctl-wd** bus provides an appropriate control word to the datapath so that only selected components of the dat path are enabled, namely those whose control inputs match bits of the control word that have been set to 1. Thus a control word must be constructed by the controller that corresponds to enabling those components able to execute the function expressed by the opcode.

28.1 Datapath Organization

The datapath provides all components required to perform any computational task specified by the instruction set. Typiical components that make up a date path are:

REGISTERS : to provide internal storage.

- **Storage register**: For storing binary sequences
- **Counter**: For incrementing/decrementing binary sequences
- **Register File**: To provide an internal memory array

MULTIPLEXERS : Digital switches for selecting from multiple inputs

FA : An adder for computing effective addresses

ALU : Arithmetic Logic Unit
Provides most of the arithmetic and logic functions

EXT : Sign extender

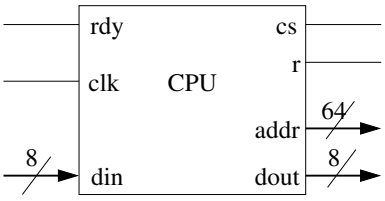
28.2 Controller Organization

As the name suggests, the controller controls the selection of components to be enabled in a given step of the instruction execution algorithm. It also determines the order in which the set of steps that define the execution algorithm are performed.

- **Storage Register**: For tracking the current step of the instruction execution algorithm.
- **Gates**: For computing the next step of the controller.
- **Read Only Memory**: For storing the required control word for each step.

28.3 Behavioral Description of the CPU

Given an instruction set architecture, a behavioral description of the CPU is defined. First the entity definition follows from how the CPU is to be interfaced with other external components. From the diagram above, the entity definition is:



The functional specification of the CPU is provided by the instruction Execution Algorithm. The details for how each instruction will be handled at each step are detailed in the following register transfer notation pseudo-code. In what follows, new symbols are introduced. Some of these will subsequently be implemented as registers and some as buses:

IC : The Instruction Code is the most significant 4-bits of the opcode. Note that the opcodes assigned to the instructions can be used to classify instructions:

- All move instructions begin with "4",
- All arithmetic/logic functions begin with "6",
- All branch instructions begin with "7"
- The compare instruction begins with "8"
- The return instruction begins with "9"

IF : The Instruction Function distinguishes among different instructions within a given IC type.

valC , **rA, rB**, store the values obtained from parsing an instruction into its operand fields.