More Machines

Murray Cole

Implementing Instructions

Consider the MIPS R2000 assembler instruction

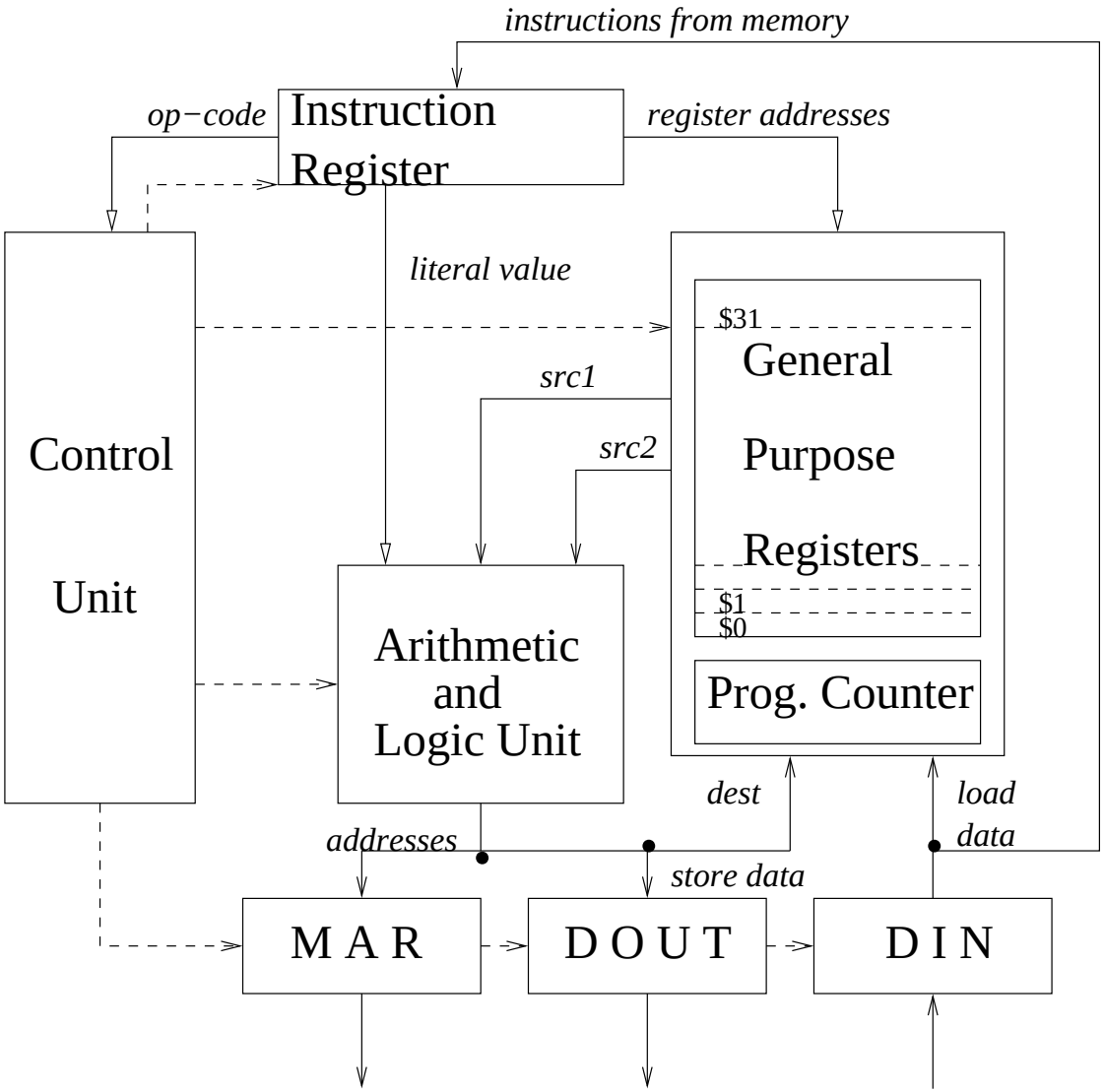
```
add $3, $1, $2
```

or

```
00000000001000100001100000100000
```

in its binary form.

How does the processor actually make this happen?


instructions from memory
op−code
Instruction Register
register addresses
literal value
Control Unit
$31
General Purpose Registers
$1
$0
src1
src2
Arithmetic and Logic Unit
Prog. Counter
dest
load data
addresses
store data
M A R
D O U T
D I N

Generating Control

The key is the control unit.

This uses the bits in the instruction to generate signals which control the operations of the other hardware components and the communications of data between them.

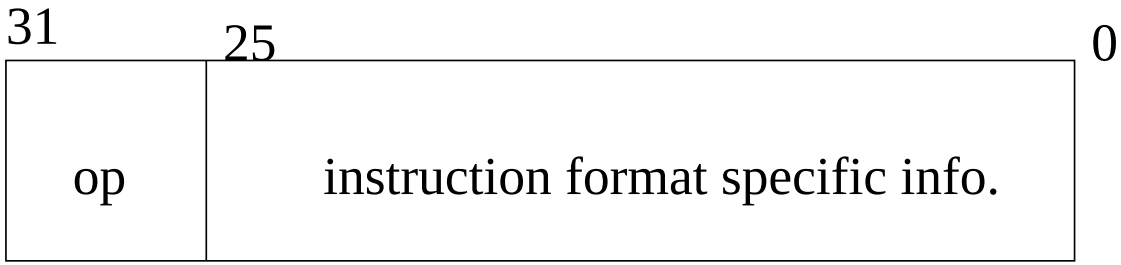
It is a transducer style FINITE STATE MACHINE!

To understand it as an FSM we must first look at its inputs and outputs in more detail.

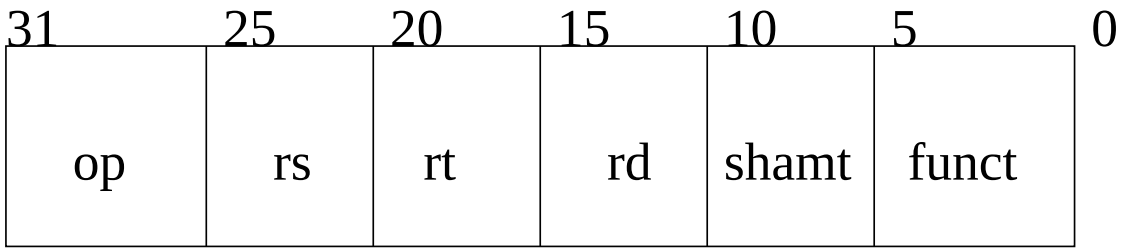
Control FSM inputs

The main input to the control FSM is the 32 bit current instruction.

To understand this, we consider its format, which explains how all the many possible instructions are encoded into 32 bits.

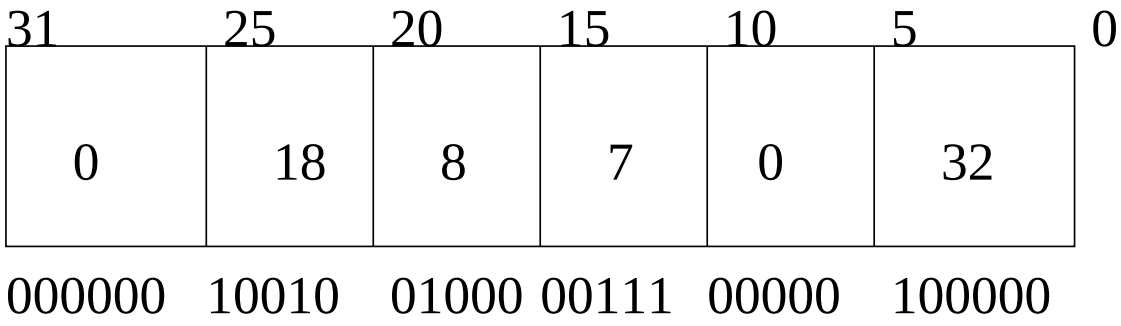


Arithmetic Instruction Format



An Add Instruction

add $7, $18, $8



MIPS R2000 Micro-architecture

Yet another layer!

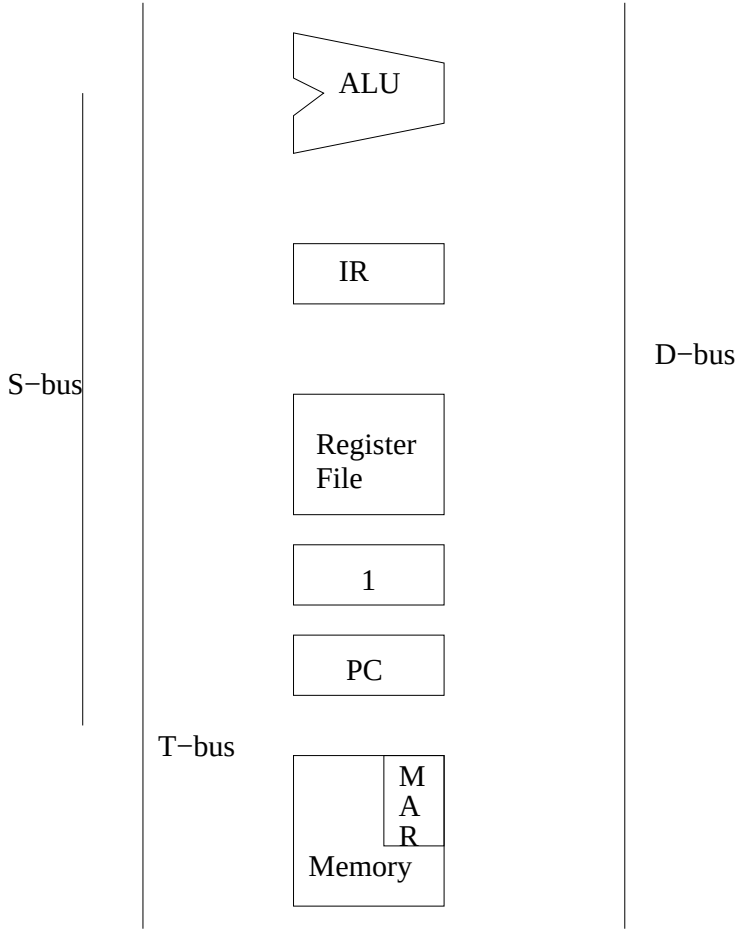
Circuit blocks which do the work or store the data.

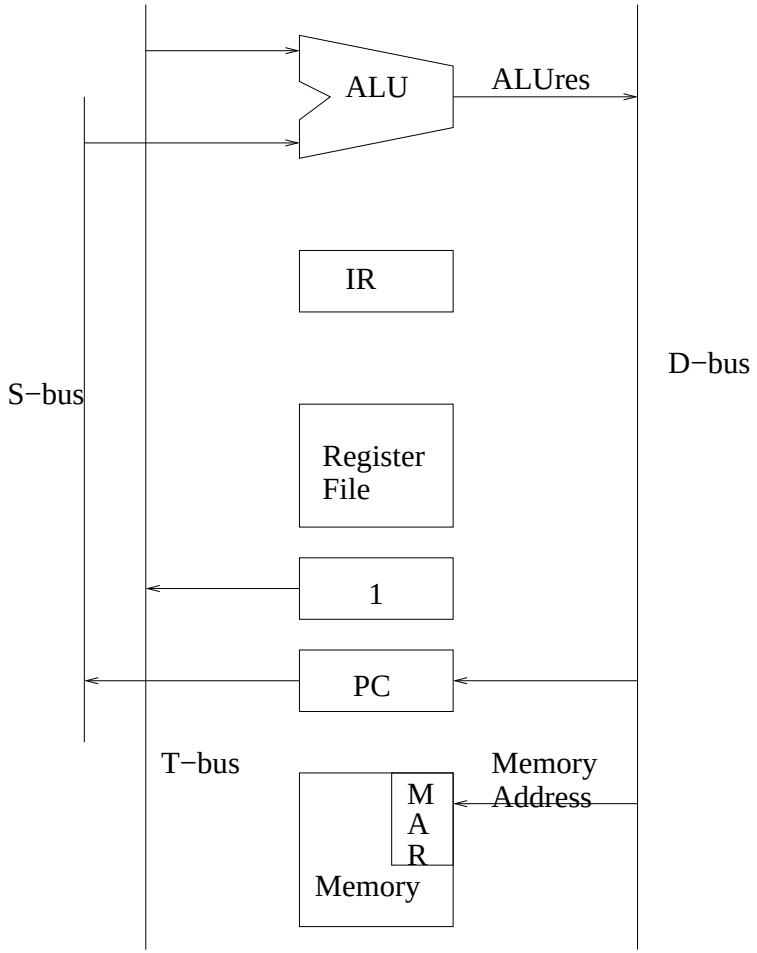
Buses (effectively wires and control logic) which connect the blocks and allow data to flow.

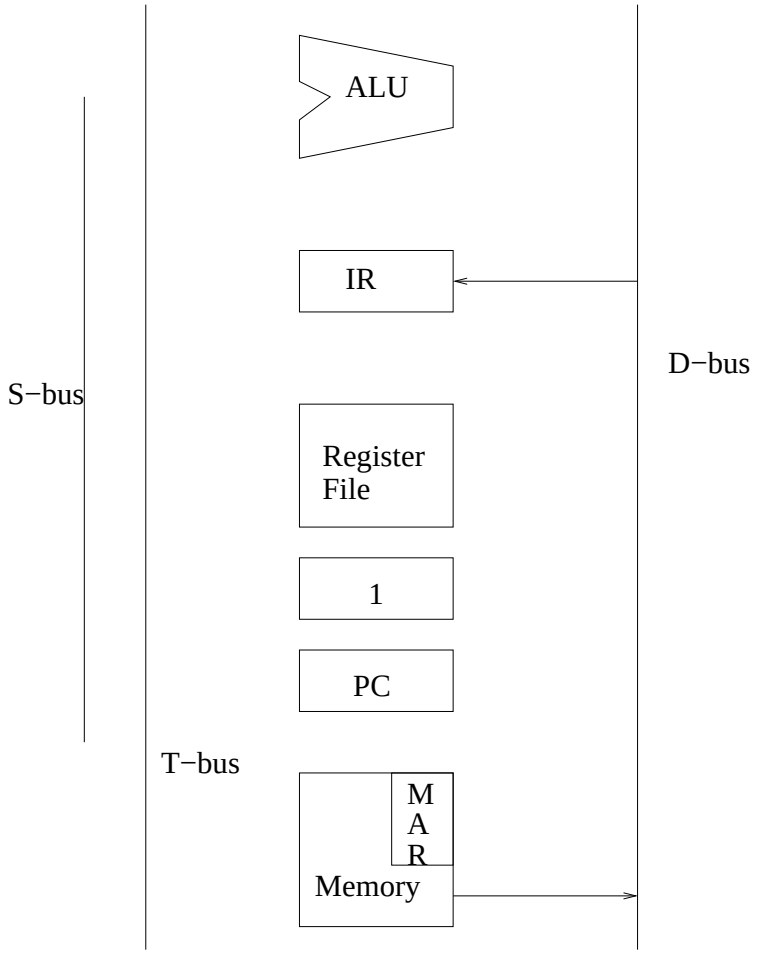
S-bus and T-bus deal with delivering inputs to the ALU.

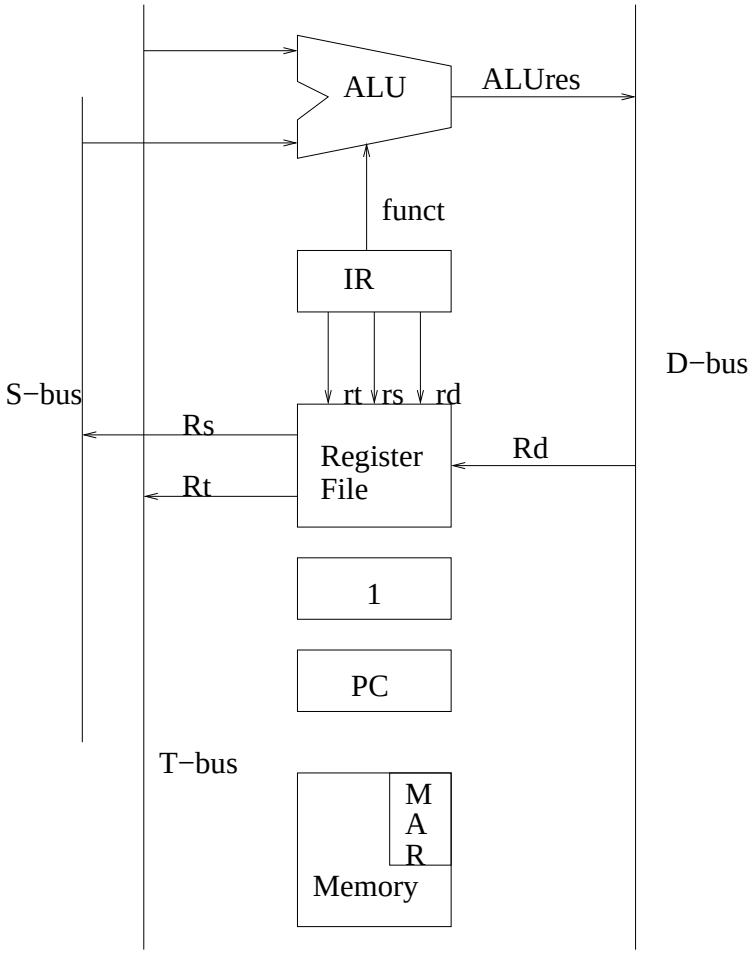
D-bus deal with delivering other data (often ALU outputs).

Actions of both circuits and buses are controlled by binary signals on control wires.


ALU
IR
S−bus
D−bus
Register
File
1
PC
T−bus
M
A
R
Memory


ALU
ALUres
IR
D−bus
S−bus
Register
File
1
PC
T−bus
Memory
Address
M
A
R
Memory


ALU
IR
D−bus
S−bus
Register
File
1
PC
T−bus
M
A
R
Memory


ALU
ALUres
funct
IR
D−bus
S−bus
rt
rs
rd
Rs
Rd
Register
File
Rt
1
PC
T−bus
M
A
R
Memory

Microinstructions

Each circuit or bus is controlled by a few signal bits, which must be generated by the control FSM in the right combination.

<S-bus>,<T-bus>,<ALUop>,<Memop>,{D-bus}

- <S-bus> defines which of the S-bus source register paths is enabled. If we look at the previous figures, we can see that there are only two possible sources — `rs` and PC — and therefore we have one of the following choices: {NONE,rs,PC}. The choice can be encoded in a two-bit field {00,01,10}.

- Similarly, <T-bus> defines which of the T-bus source register paths is enabled, i.e. one of {NONE,rt,literal,constant 1}. The choices can be encoded in a two-bit field {00,01,10,11}.

- <ALUop> defines the ALU operation to be performed if any, i.e. {NOP, add, subtract, funct, ==, !=, <, > }. The choices are encoded in a three-bit field {000,001,010,011,100,101,110,111}.

- <Memop> defines a memory operation to be performed if any, i.e. {NOP,read,write}. The choices are encoded in a two-bit field {00,01,10}.

- {D-bus} defines a *set* of the D-bus destination register paths enabled, i.e. none or more of {rd,IR,PC,MAR}. The choices are encoded in a four-bit field as more than one may be enabled at any one time.

Each of our microsteps is implemented by setting up these fields and sending out the resulting microinstruction on the control wires.

Step 0

S-bus	T-bus	ALUop	Memop	D-bus
10	11	001	00	0011

Step 1

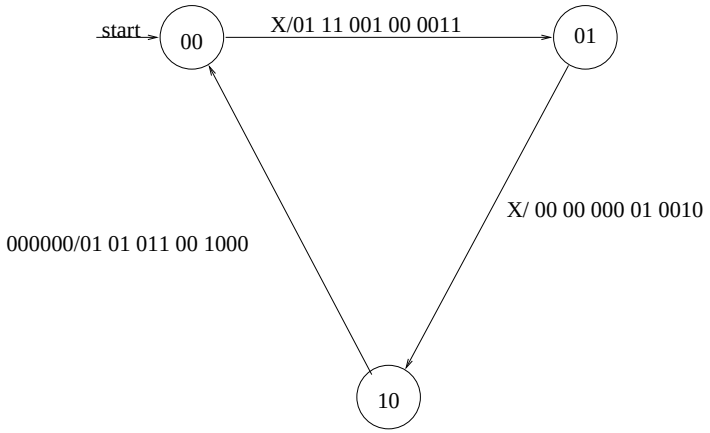
S-bus	T-bus	ALUop	Memop	D-bus
00	00	000	01	0100

Step 2

S-bus	T-bus	ALUop	Memop	D-bus
01	01	011	00	1000

Generating Control

Here is the part of the control FSM for simple arithmetic instructions. The input is the IR opcode (X means “don’t care”).



Other instructions would introduce new transitions after 10.

More Complex Processors

Processors like the MIPS R2000 are still used in millions of simple embedded systems, where high performance is not essential.

Contemporary processor designs (e.g. Pentium) are much more complicated in detail, but in many ways quite similar in principle.

Later courses will examine these advanced architectures.

One key technique is to execute many machine code level instructions at the same time, without breaking the expected rules of behaviour.

Discovering and exploiting this on-chip parallelism is one of the major challenges in processor (and compiler) research and design.