

CS1Bh Lecture Note 24

Machines, continued

24.1 Introduction

The previous lecture introduced the von Neumann machine as the basic model for most general-purpose computers of the past 50 years. It relies on a distinction between instructions and memory, and uses the fetch-execute cycle to run machine-code programs. We looked at a typical processor — the MIPS R2000 — and briefly examined aspects of its instruction set. In this lecture we examine instructions in more detail, and look at how a processor executes machine instructions by a series of *micro-instructions*, again using the MIPS R2000 as an exemplar.

24.2 Instruction formats

Depending on the type of instruction encountered, the processor performs different actions or micro-instructions. The binary format of an instruction depends on its type. The exact format of instructions is tightly coupled with factors like the chip design, word size and register provision of the processor. The exact reasoning behind each particular format is beyond the scope of this course. In the MIPS R2000 instruction set, all instructions have the basic format shown in Figure 24.1, where *op* refers to the type of instruction, e.g. add, load, jump, etc.

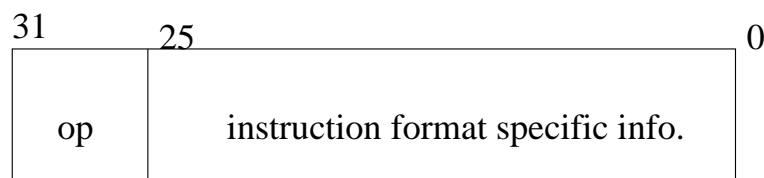


Figure 24.1: MIPS R2000 basic instruction format

There is a connection from the processor's instruction register to the control unit for transferring the six bits making up the *op* field. Based on this information, the control unit instructs the remainder of the processor to act appropriately.

Now, in more detail, arithmetic instructions only involving registers have the format shown in Figure 24.2, where *op* is the six-bit operation code as already seen, *rs* is a

five-bit source register specifier (`src1`), `rt` is a five-bit target register specifier (`src2`), `rd` is a five-bit destination register specifier (`dest`), `shamt` is a five-bit shift amount and `funct` is a six-bit operation field.

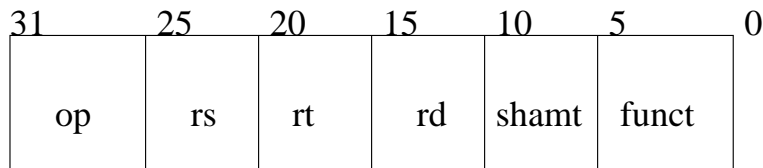


Figure 24.2: MIPS R2000 arithmetic instruction format

The `op` field determines what sort of instruction is to be executed. In the case of arithmetic instructions it is set to zero, which tells the control unit that this is an arithmetic instruction and it should look in the `funct` field to determine what sort of arithmetic instruction it is.

To illustrate this, consider the instruction `add $7, $18, $8`. It is represented in memory as shown in Figure 24.3.

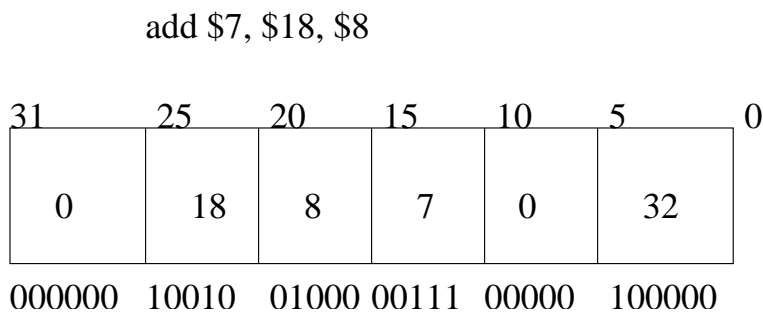


Figure 24.3: MIPS R2000 example add instruction format

In the `op` field, there is zero, showing it is an arithmetic instruction, and the `funct` field is 32 (100000 in binary), to denote that it is an add operation. The other fields refer to the registers used, while the `shamt` field is set to zero as there are no shifts involved in this operation.

24.3 Micro-Instructions

To execute one machine instruction, at least three different operations have to be carried out: (i) update the program counter; (ii) fetch the next instruction; and (iii) execute the instruction. Thus, internally, several *micro-instructions* have to be performed.

The actual number of micro-instructions performed depends on the type of instruction encountered. Data movement operations to and from memory require more micro-instructions than arithmetic operations for instance. To keep things simple, here we only consider the micro-instructions performed for arithmetic and logical instructions.

24.4 The datapath

In this section, we examine in a little more detail what happens on the datapath during each micro-instruction. One possible surprise will be that more than one operation can happen simultaneously. Thus, although from an instruction point of view instructions are executed sequentially, inside the processor there is some parallelism.

In the first step of the micro-instruction sequence, we both load the instruction register with the instruction to be executed *and* update the program counter. This is usefully represented using a *register transfer notation*, that shows which components of the processor are used at different stages:

Step 0: $PC \leftarrow PC+1; MAR \leftarrow PC$

The “;” denotes that two actions occur together. Here, the program counter (PC) is incremented, and its new contents are placed in the Memory Address Register (MAR).

In the next step, an instruction stored in memory at the position stored in the memory address register (MAR), is transferred to the instruction register (IR), i.e. the instruction is loaded.

Step 1: $IR \leftarrow \text{Memory}[MAR]$

Finally, the ALU performs the operation specified by the *funct* field in IR (i.e. the last six bits of the instruction in the instruction register $IR[5..0]$) on the contents of *rs* and *rt*, and then writes the result to register *rd*.

Step 2. $rd \leftarrow rs \text{ funct } rt$

This is all that takes place when executing an arithmetic operation. Note that two steps are required to load the instruction but only one to do the actual work. The first two steps are executed regardless of the instruction involved, i.e. we always have to load an instruction regardless of its type. The remaining step(s) depend on its type and are determined by the control unit.

Now, we will examine a slightly simplified version of the MIPS 2000 processor and see what happens to different components of the datapath during each micro-instruction. The main difference between the following diagrams of the MIPS datapath and the diagram shown in the previous lecture note is the form of interconnections between units. Instead of the multiple connections between different units, there are, in fact, three main interconnection highways or *buses* in the MIPS R2000. These buses are named the S, T and D buses, because the S-bus acts as source (*src1*) to the ALU as does the T-bus (*src2*), while the D-bus carries data to several destinations (*dest*).

Figure 24.4 shows which components of the datapath are active during Step 0 of the above micro-instruction sequence. The value of the PC and the number 1 flow along the S and T buses respectively, and they are added together within the ALU. The result of this is transferred to the memory address register (MAR) and back into the PC, updating its value.

During Step 1, the appropriate instruction in memory is transferred on to the D-bus and then into the instruction register (IR), as shown in Figure 24.5.

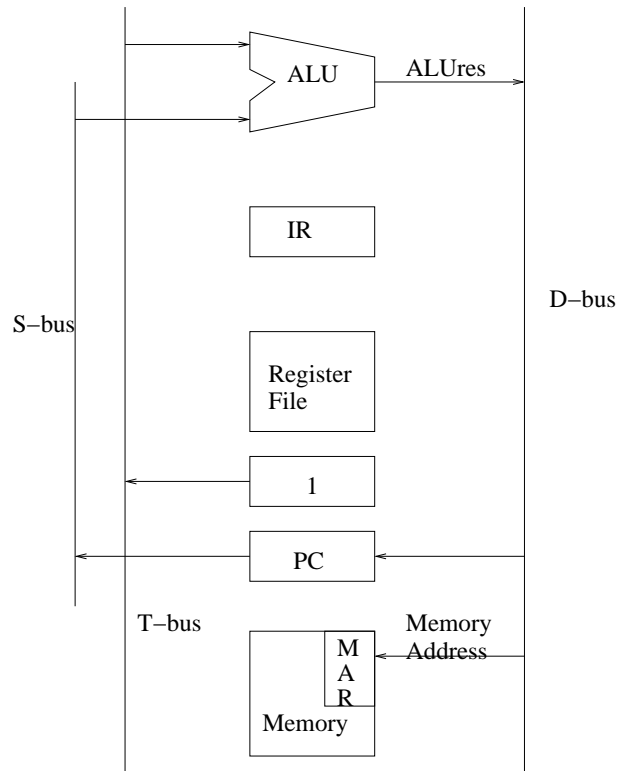


Figure 24.4: MIPS R2000 datapath during Step 0

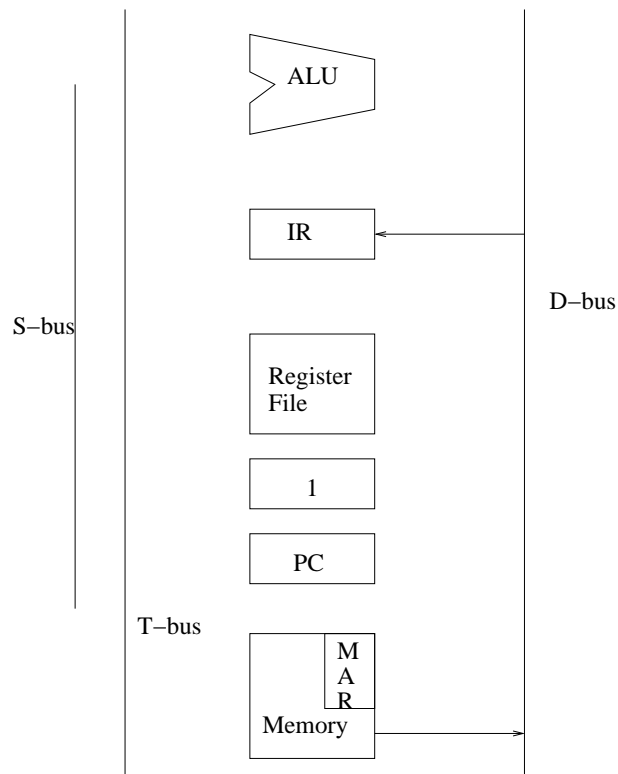


Figure 24.5: MIPS R2000 datapath during Step 1

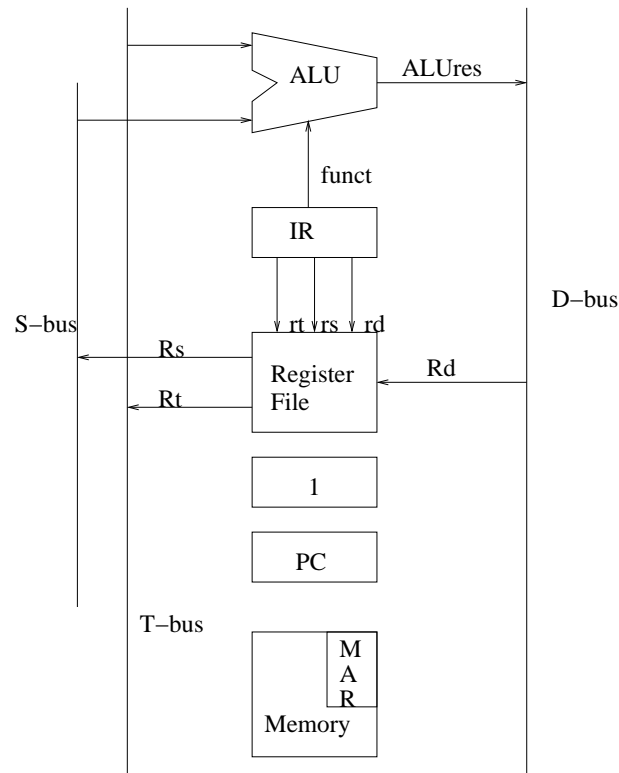


Figure 24.6: MIPS R2000 datapath during Step 2

The final step, Step 2, is illustrated in Figure 24.6. The names of the three registers involved in the operation are passed to the register file from the appropriate fields within the instruction currently stored in the IR. Similarly, the type of ALU operation is passed to the ALU from the IR. The actual values stored in registers *rs* and *rt* are transferred on to the S- and T-buses respectively, and passed to the ALU. The appropriate function, based on the *funct* field, is carried out and the result passed back to the appropriate register, *rd*, on the D-bus.

We have now completely described what happens during the loading and execution of an arithmetic operation. The key decision to be made at each micro-step is which buses and registers of the datapath are to be involved in that micro-step. This is determined by the control unit, and is the subject of the next section.

24.5 The control unit

The control unit decides, based on various inputs, which parts of the datapath are to be involved in a particular micro-step. In particular, it determines which, if any, of the buses are to be used and which registers on that bus are to be involved. It also instructs the ALU as to what operation to perform and what memory operation (if any) is to take place.

However, before we describe how the control unit makes and implements these decisions, it is worthwhile briefly mentioning how it can either connect or disconnect two registers via a bus.

Simplistically, the control unit sends a signal along a wire to the bus which either allows data to flow between two units or not — rather like turning a light switch on or off. It does this via a very simple electronic device called a multiplexer, the technical details of which are covered later on in the course. The main point to note is that selection of whether two registers are connected across a bus is trivial to implement, and therefore our main focus here can be on how the control unit makes such a decision in the first place.

In order to describe the actions of the control unit accurately at each step, we must first examine another instruction format, namely the micro-instruction format. The general form is as follows:

$\langle S\text{-bus} \rangle, \langle T\text{-bus} \rangle, \langle \text{ALUop} \rangle, \langle \text{Memop} \rangle, \{D\text{-bus}\}$

Taken together, these items describe which components are connected, and what operations are to take place on the datapath:

- $\langle S\text{-bus} \rangle$ 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: $\{\text{NONE}, rs, \text{PC}\}$. The choice can be encoded in a two-bit field $\{00, 01, 10\}$.
- Similarly, $\langle T\text{-bus} \rangle$ defines which of the T-bus source register paths is enabled, i.e. one of $\{\text{NONE}, rt, \text{literal}, \text{constant } 1\}$. The choices can be encoded in a two-bit field $\{00, 01, 10, 11\}$.
- $\langle \text{ALUop} \rangle$ defines the ALU operation to be performed if any, i.e. $\{\text{NOP}, \text{add}, \text{subtract}, \text{funct}, =, \neq, <, >\}$. The choices are encoded in a three-bit field $\{000, 001, 010, 011, 100, 101, 110, 111\}$.
- $\langle \text{Memop} \rangle$ defines a memory operation to be performed if any, i.e. $\{\text{NOP}, \text{read}, \text{write}\}$. The choices are encoded in a two-bit field $\{00, 01, 10\}$.
- $\{D\text{-bus}\}$ defines a set of the D-bus destination register paths enabled, i.e. none or more of $\{rd, IR, \text{PC}, \text{MAR}\}$. The choices are encoded in a four-bit field as more than one may be enabled at any one time.

At each step of the micro-instruction sequence, the control unit is responsible for deciding what the values of S-bus, T-bus etc. are. We will now examine what happens at each step to these fields.

Step 0

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

The program counter (PC) places its value on the S-bus and the value 1 is placed on the T-bus. The ALU is instructed to add the two values, and the resulting value is placed on the D-bus and received by both the program counter and the memory address register. No memory operations take place during this step.

Step 1

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

No values are placed on the S or T-bus and the ALU is inactive. A read memory operation takes place, with the value placed on the D-bus and transferred to the instruction register (IR).

Step 2

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

Finally, the value of `rs` is placed on the S-bus and the value of `rt` is placed on the T-bus, and are transferred to the ALU which performs whatever function is defined in the `funct` field of the instruction. The result of this is placed on the D-bus and transferred to the destination `rd`. No memory operations take place during this step.

24.6 Control Unit as FSM

We now know what the control unit outputs at each stage, so the only remaining question is: how does it make such a decision?

The control unit outputs, which enable or disable parts of the datapath, are determined by just two inputs. These are the current contents of the IR — specifically the op-code field — and the current state of the processor. The state of the processor corresponds to the current step in the micro-instruction sequence, which can be 0, 1 or 2 in our running example.

The outputs of the control unit are: (i) the micro-instruction described above which enables/disables certain sections of the datapath, and (ii) the next step `S'`. This behaviour can be expressed in a table as follows:

Opcode	S	S-bus	T-bus	ALUop	Memop	D-bus	S'
X	00	01	11	001	00	0011	01
X	01	00	00	000	01	0010	10
000000	10	01	01	011	00	1000	00

This table is often known as a *state table*¹ as it describes the next state and outputs given the current state and input. The steps 0, 1 and 2 are represented by the values of S: 00, 01 and 10. Associated with each step is the corresponding micro-instruction, which determines those registers on the three buses that should be connected, the ALU operation and memory operations to be carried out and the next step, `S'`. This table can be represented in the more familiar form of the state diagram shown in Figure 24.7, where the arcs are labelled with input/output and the nodes are labelled with the current states.

As it stands, this state machine is only capable of performing ALU operations. However, it is relatively straightforward to work out what should happen for other types of instructions and hence to derive the corresponding revised FSM.

¹Note: “X” stands for ‘don’t care’.

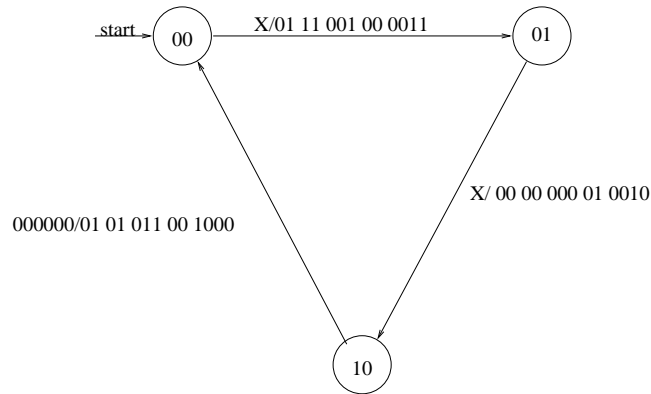


Figure 24.7: Control unit finite state machine

24.7 Summary

This lecture has examined in greater detail the internal workings of the MIPS R2000 processor. It has introduced the notion of micro-instructions and described their impact on the datapath and control unit. We have examined the role of the control unit and seen that it can be implemented as a finite state machine.

*Based on original material by Mike O'Boyle, Gordon Brebner.
Murray Cole, 29th April 2003.*