Chap.4 The Processor

4.1 Introduction

An Overview of the Implementation

In Chapter 2, we looked at the core MIPS instructions, including the integer arithmetic-logical instructions, the memory-reference instructions, and the branch instructions. Much of what needs to be done to implement these instructions is the same, independent of the exact class of instruction. For every instruction, the first two steps are identical:

1. Send the program counter (PC) to the memory that contains the code and fetch the instruction from that memory.
2. Read one or two registers, using fields of the instruction to select the registers to read. For the load word instruction, we need to read only one register, but most other instructions require that we read two registers.

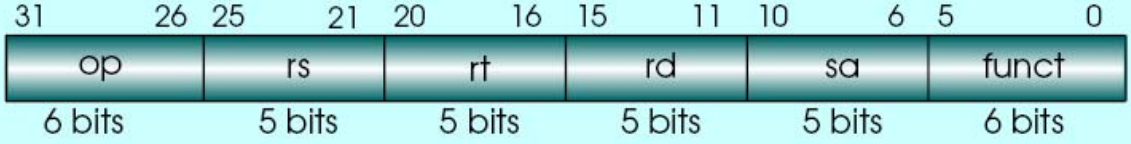
After these two steps, the actions required to complete the instruction depend on the instruction class. Fortunately, for each of the three instruction classes (memory-reference, arithmetic-logical, and branches), the actions are largely the same, independent of the exact instruction. The simplicity and regularity of the MIPS instruction set simplifies the implementation by making the execution of many of the instruction classes similar.

MIPS指令的功能分类

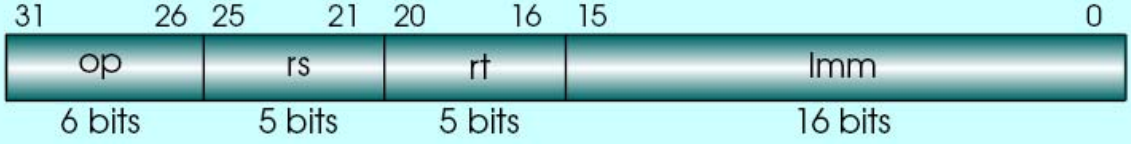
- 算術運算
  add, sub, addi, addu, mul, mulu, div, divu
- 邏輯運算
  and, or, andi, ori, sll, srl
- 數據傳送
  lw, sw, lb, lbu, sb
- 條件轉移
  beq, bne, bnez, slt, slti, sltu, sltiu
- 無條件跳轉
  j, jr, jal

MIPS指令的三种格式

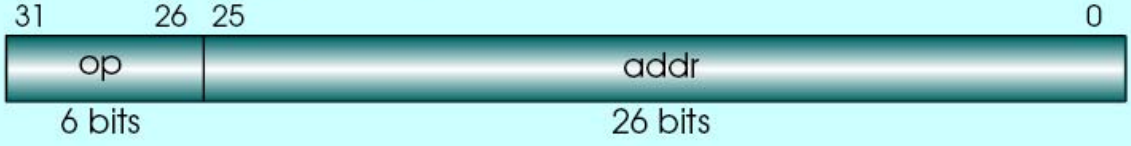
R 類型指令 (rs, rt, rd: 寄存器號, sa: 移位位數)



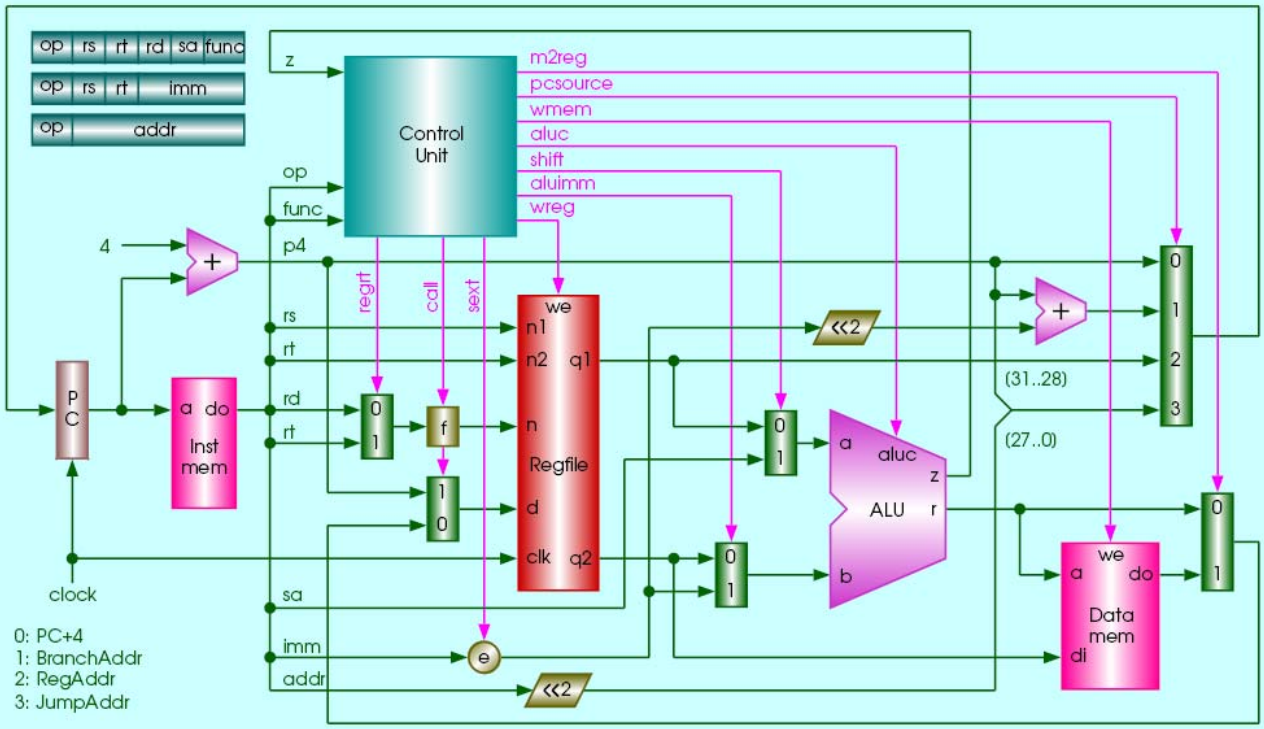
I 類型指令 (rs, rt: 寄存器號, imm: 立即數)



J 類型指令 (addr: 跳轉目標地址)



单周期CPU+指令存储器+数据存储器



4.1 Introduction

MIPS子集的实现的抽象结构 - 包括主要的功能单元和它们之间的联系

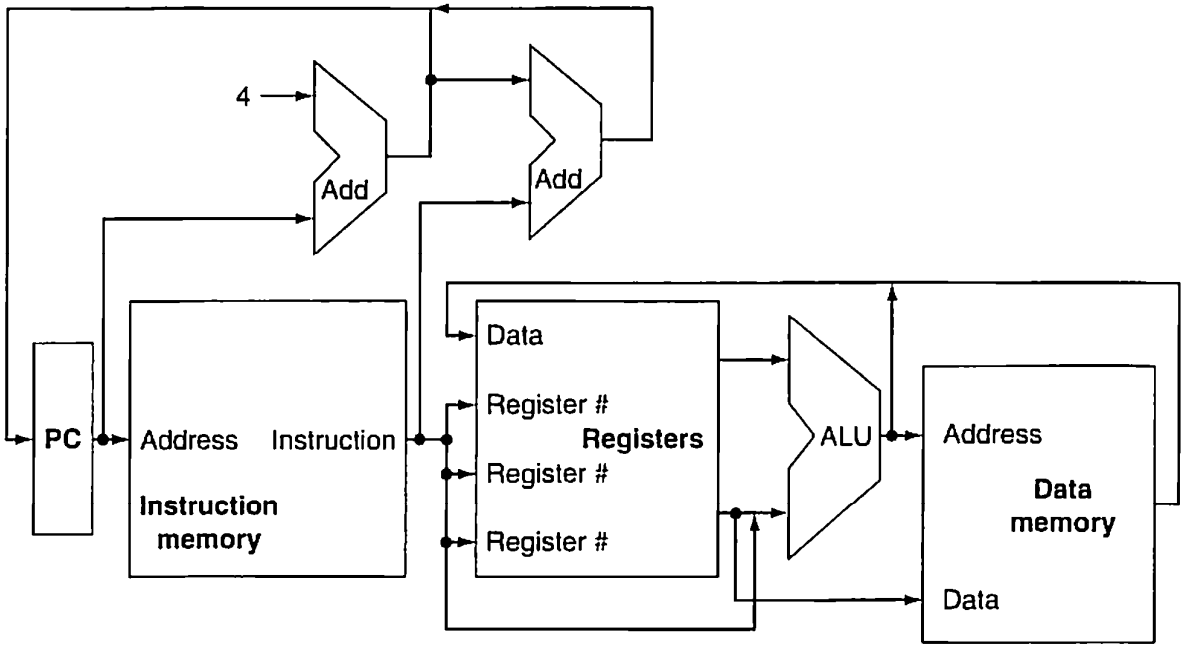
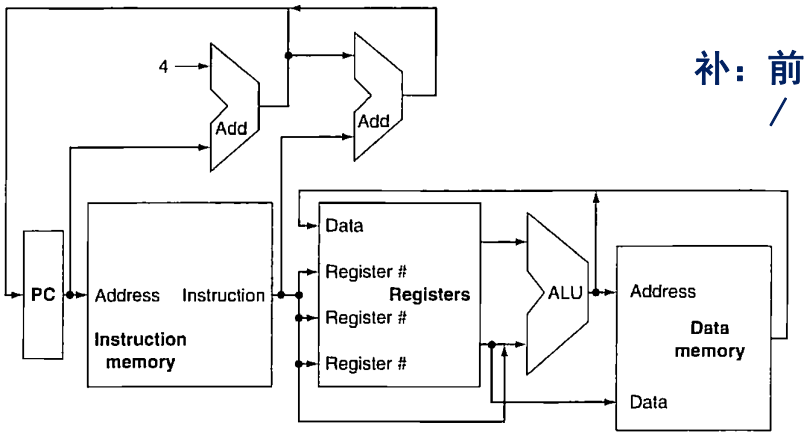


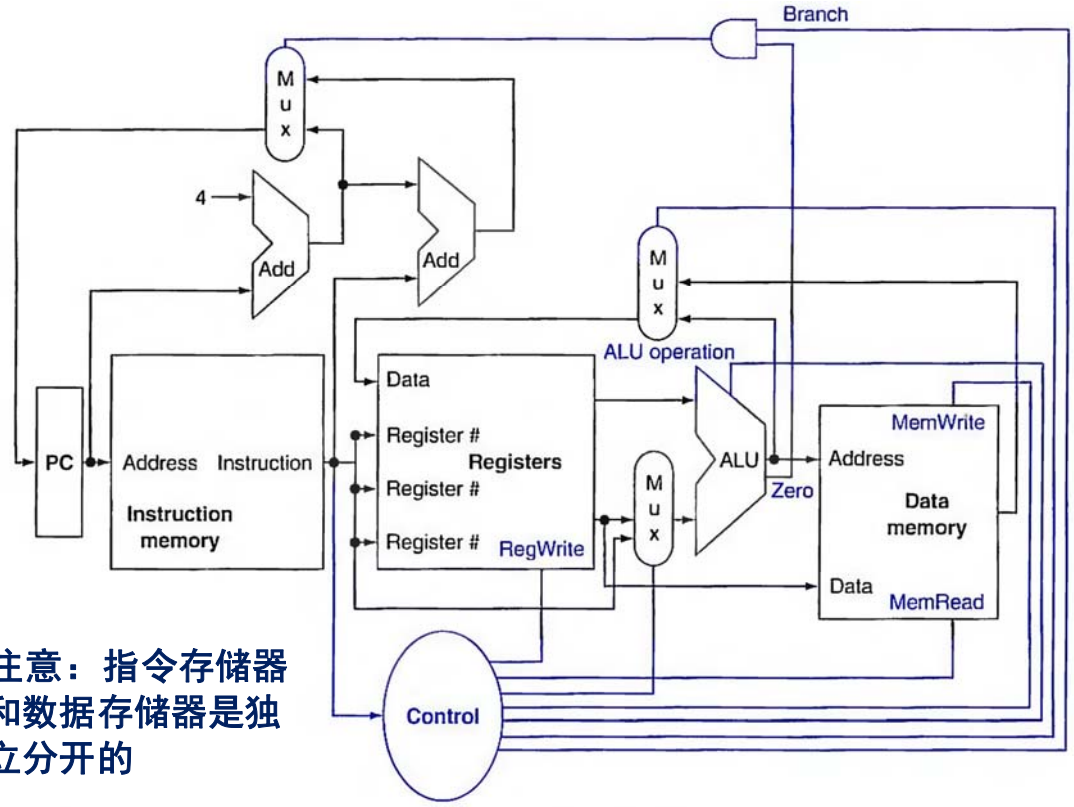
FIGURE 4.1 An abstract view of the implementation of the MIPS subset showing the major functional units and the major connections between them. All instructions start by using



补：前图图示说明
/ 指令的实现执行过程

FIGURE 4.1 An abstract view of the implementation of the MIPS subset showing the major functional units and the major connections between them. All instructions start by using the program counter to supply the instruction address to the instruction memory. After the instruction is fetched, the register operands used by an instruction are specified by fields of that instruction. Once the register operands have been fetched, they can be operated on to compute a memory address (for a load or store), to compute an arithmetic result (for an integer arithmetic-logical instruction), or a compare (for a branch). If the instruction is an arithmetic-logical instruction, the result from the ALU must be written to a register. If the operation is a load or store, the ALU result is used as an address to either store a value from the registers or load a value from memory into the registers. The result from the ALU or memory is written back into the register file. Branches require the use of the ALU output to determine the next instruction address, which comes either from the ALU (where the PC and branch offset are summed) or from an adder that increments the current PC by 4. The thick lines interconnecting the functional units represent buses, which consist of multiple signals. The arrows are used to guide the reader in knowing how information flows. Since signal lines may cross, we explicitly show when crossing lines are connected by the presence of a dot where the lines cross.

包含必需的多路复用器和控制线的MIPS子集的基本实现



注意：指令存储器和数据存储器是独立分开的

FIGURE 4.2 The basic implementation of the MIPS subset, including the necessary multiplexors and control lines.

4.2 Logic Design Conventions

Clocking Methodology 时钟同步方法

Figure 4.3 shows the two state elements surrounding a block of combinational logic, which operates in a single clock cycle: all signals must propagate from state element 1, through the combinational logic, and to state element 2 in the time of one clock cycle. The time necessary for the signals to reach state element 2 defines the length of the clock cycle.

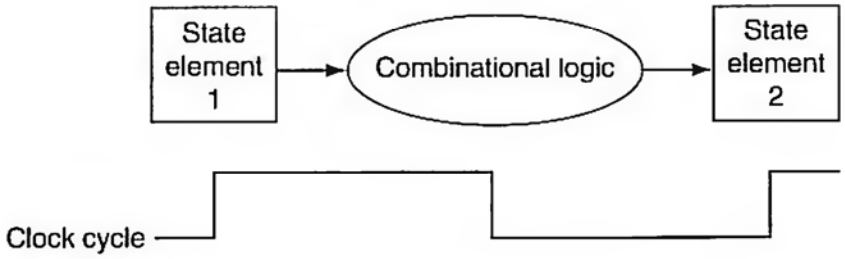


FIGURE 4.3 Combinational logic, state elements, and the clock are closely related. In a synchronous digital system, the clock determines when elements with state will write values into internal storage. Any inputs to a state element must reach a stable value (that is, have reached a value from which they will not change until after the clock edge) before the active clock edge causes the state to be updated. All state elements in this chapter, including memory, are assumed to be edge-triggered.

4.2 Logic Design Conventions

Clocking Methodology 时钟同步方法

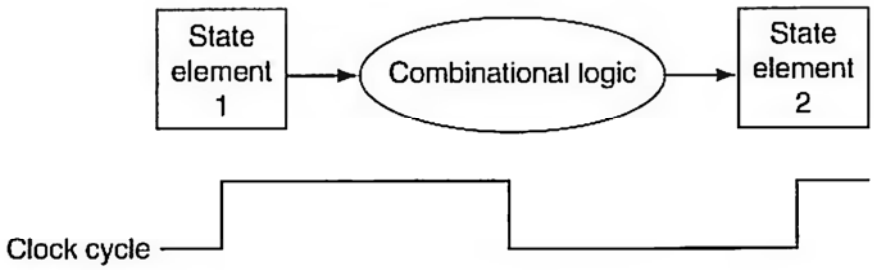


FIGURE 4.3 Combinational logic, state elements, and the clock are closely related. In a synchronous digital system, the clock determines when elements with state will write values into internal storage. Any inputs to a state element must reach a stable value (that is, have reached a value from which they will not change until after the clock edge) before the active clock edge causes the state to be updated. All state elements in this chapter, including memory, are assumed to be edge-triggered.

注意：DE2实验板中的FPGA支持同步MEM

- 要深刻理解组合逻辑（combinational logic）和时序逻辑（ sequential logic ）的区别和联系。
- 深刻理解边沿触发的含义，尤其是在同步数字逻辑中的作用。坐球比赛？
- 事实上，大系统中的时钟同步也是不易做到的。时钟信号也会有延迟。

Clocking Methodology 时钟同步方法

沿触发方式允许状态单元的写和读发生在同一个时钟周期里

而且不会产生不确定状态

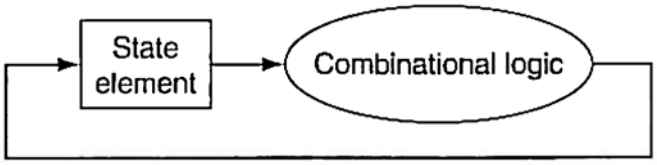


FIGURE 4.4 An edge-triggered methodology allows a state element to be read and written in the same clock cycle without creating a race that could lead to indeterminate data values. Of course, the clock cycle still must be long enough so that the input values are stable when the active clock edge occurs. Feedback cannot occur within one clock cycle because of the edge-triggered update of the state element. If feedback were possible, this design could not work properly. Our designs in this chapter and the next rely on the edge-triggered timing methodology and on structures like the one shown in this figure.

- 可以将state element理解为维持阻塞型的D触发器、或主从结构的J-K触发器等，理解电路结构决定了其不会发生因反馈导致的“空翻”。

combinational element An operational element, such as an AND gate or an ALU.

state element A memory element, such as a register or a memory.

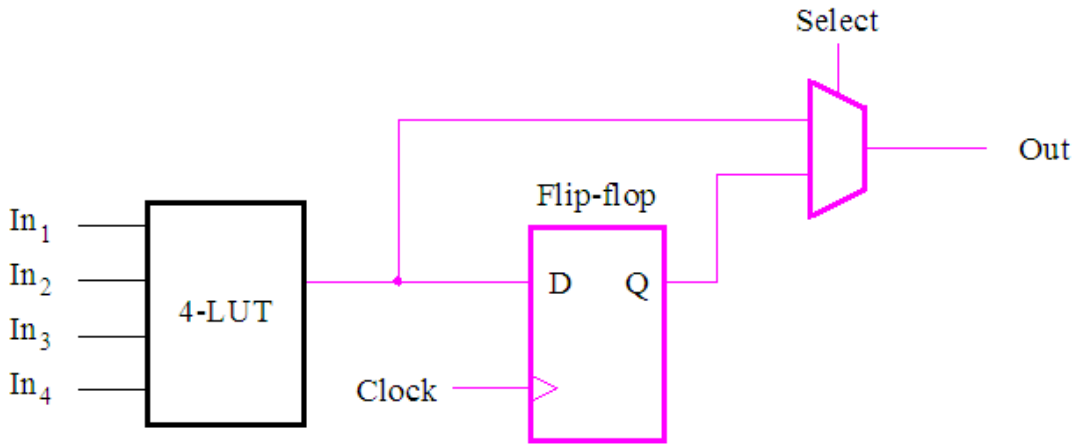
clocking methodology The approach used to determine when data is valid and stable relative to the clock.

edge-triggered clocking A clocking scheme in which all state changes occur on a clock edge.

FPGA中的基本逻辑单元原理

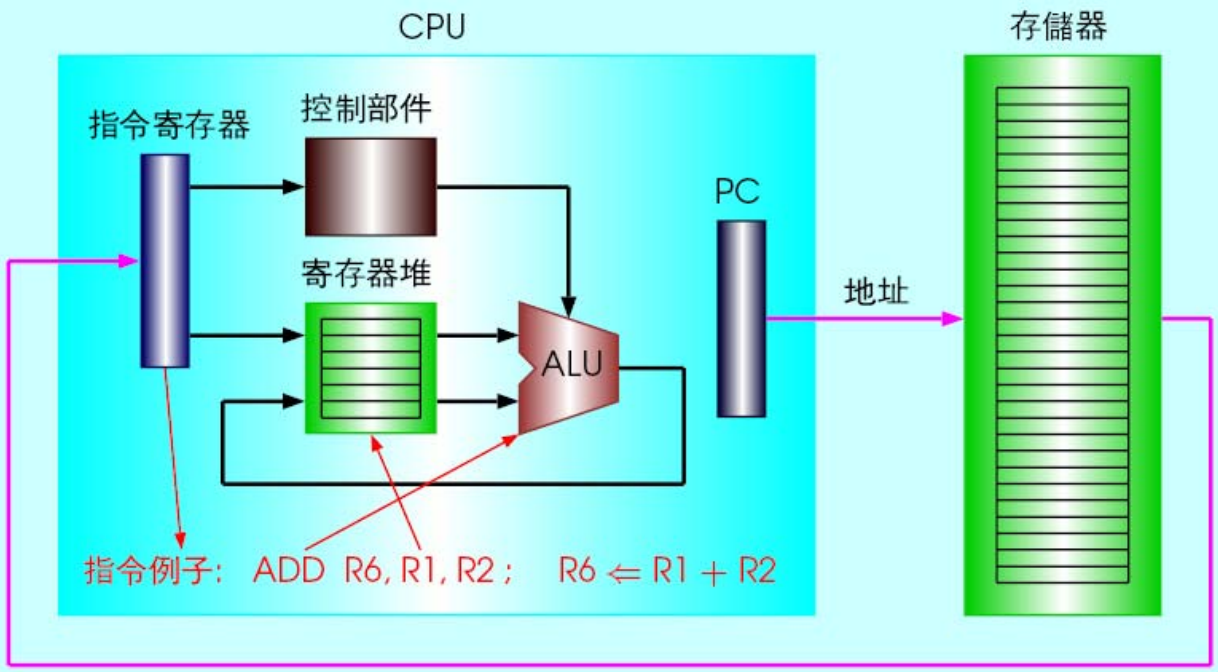
FPGA Architecture: Logic Element（LE）

- Lookup table (LUT) implements any 4-input logic function



- 该LE可构成组合逻辑或时序逻辑。
- 真正的LE要复杂的多。

4.3 Building a Datapath



4.3 Building a Datapath

部分名词

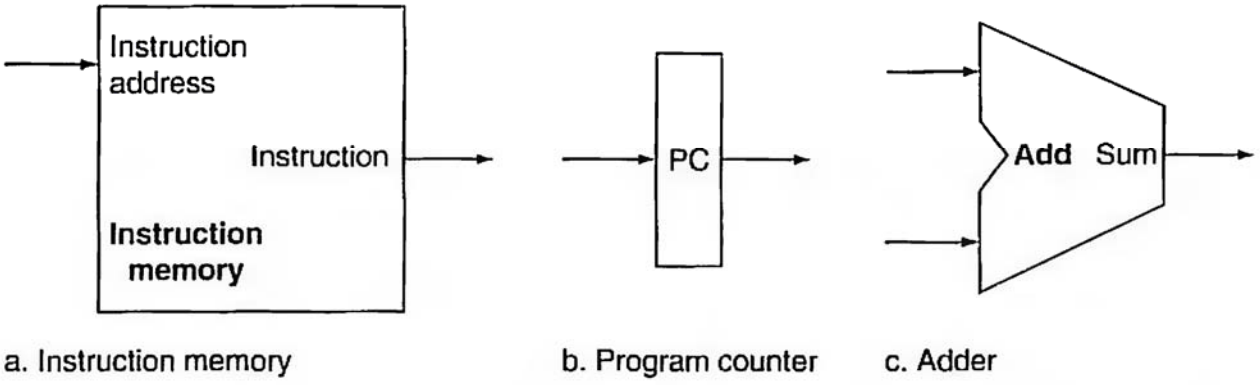
datapath element A unit used to operate on or hold data within a processor. In the MIPS implementation, the datapath elements include the instruction and data memories, the register file, the ALU, and adders.

program counter (PC) The register containing the address of the instruction in the program being executed.

- Ddatapath element，数据通路元素，处理器中用于操作或保持数据的单元。如存储器、寄存器、ALU、加法单元等。
- Program counter，包含当前正在执行指令的地址的寄存器。

主要功能单元实现技术

存取指令需要指令存储器和PC, 计算下一条指令地址需要一个加法器：



a. Instruction memory　　b. Program counter　　c. Adder

FIGURE 4.5 Two state elements are needed to store and access instructions, and an adder is needed to compute the next instruction address. The state elements are the instruction memory and the program counter. The instruction memory need only provide read access because the datapath does not write instructions. Since the instruction memory only reads, we treat it as combinational logic: the output at any time reflects the contents of the location specified by the address input, and no read control signal is needed. (We will need to write the instruction memory when we load the program; this is not hard to add, and we ignore it for simplicity.) The program counter is a 32-bit register that is written at the end of every clock cycle and thus does not need a write control signal. The adder is an ALU wired to always add its two 32-bit inputs and place the sum on its output.

主要功能单元实现技术

用于取指令和程序计数器PC自增的数据通路的一部分

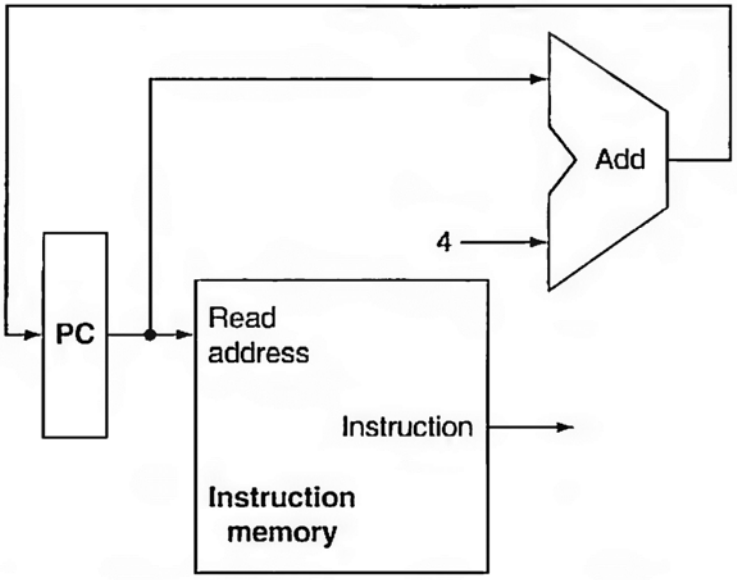


FIGURE 4.6 A portion of the datapath used for fetching instructions and incrementing the program counter. The fetched instruction is used by other parts of the datapath.

主要功能单元实现技术

实现 寄存器 - 寄存器 型运算需要寄存器堆（register file）和运算器ALU

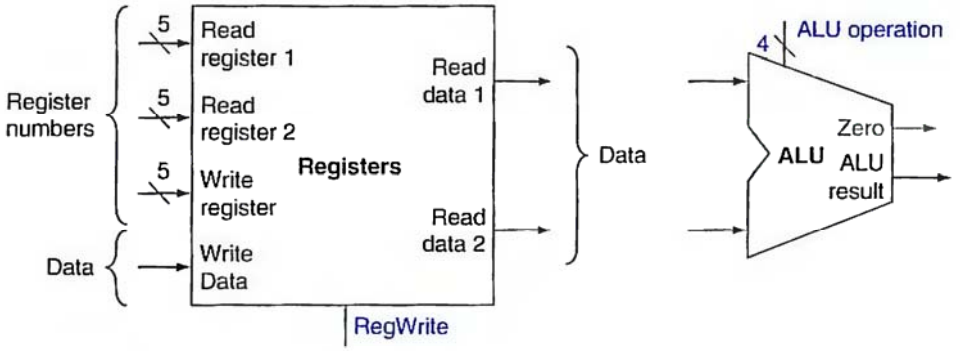


FIGURE 4.7 The two elements needed to implement R-format ALU operations are the register file and the ALU. The register file contains all the registers and has two read ports and one write port. The design of multiported register files is discussed in Section C.8 of Appendix C. The register file always outputs the contents of the registers corresponding to the Read register inputs on the outputs; no other control inputs are needed. In contrast, a register write must be explicitly indicated by asserting the write control signal. Remember that writes are edge-triggered, so that all the write inputs (i.e., the value to be written, the register number, and the write control signal) must be valid at the clock edge. Since writes to the register file are edge-triggered, our design can legally read and write the same register within a clock cycle: the read will get the value written in an earlier clock cycle, while the value written will be available to a read in a subsequent clock cycle. The inputs carrying the register number to the register file are all 5 bits wide, whereas the lines carrying data values are 32 bits wide. The operation to be performed by the ALU is controlled with the ALU operation signal, which will be 4 bits wide, using the ALU designed in Appendix C. We will use the Zero detection output of the ALU shortly to implement branches. The overflow output will not be needed until Section 4.9, when we discuss exceptions; we omit it until then.

具有双读端口的寄存器堆：双读端口的实现原理

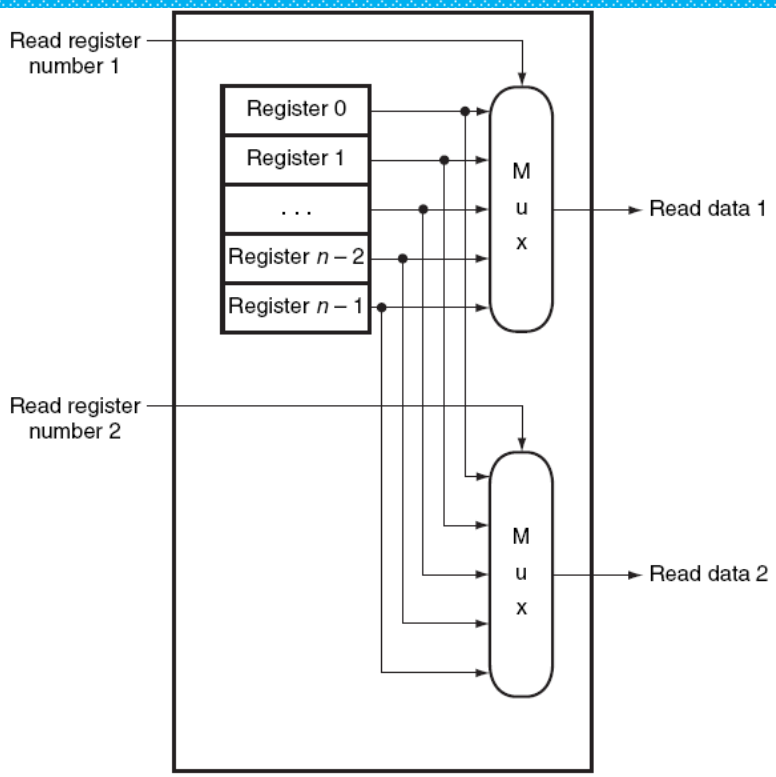


FIGURE C.8.8 The implementation of two read ports for a register file with *n* registers can be done with a pair of *n*-to-1 multiplexors, each 32 bits wide. The register read number signal is used as the multiplexor selector signal. Figure C.8.9 shows how the write port is implemented.

具有双读端口的寄存器堆：写端口实现原理

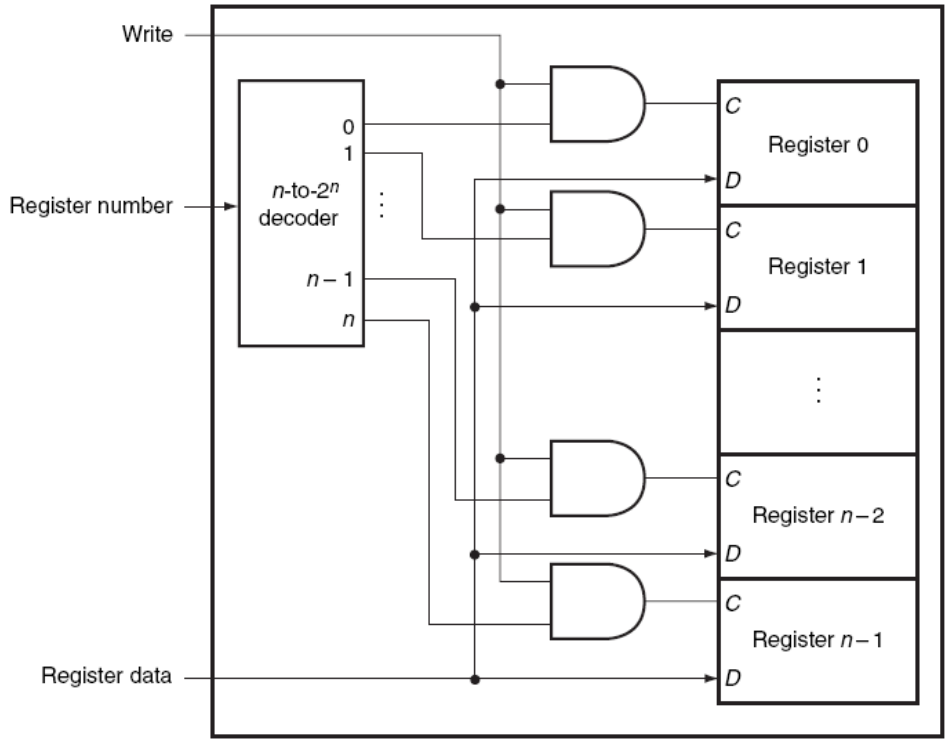


FIGURE C.8.9 The write port for a register file is implemented with a decoder that is used with the write signal to generate the *C* input to the registers. All three inputs (the register number, the data, and the write signal) will have setup and hold-time constraints that ensure that the correct data is written into the register file.

ALU的功能（内部实现原理略）

FIGURE C.5.13 The values of the three ALU control lines, Bnegate, and Operation, and the corresponding ALU operations.

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

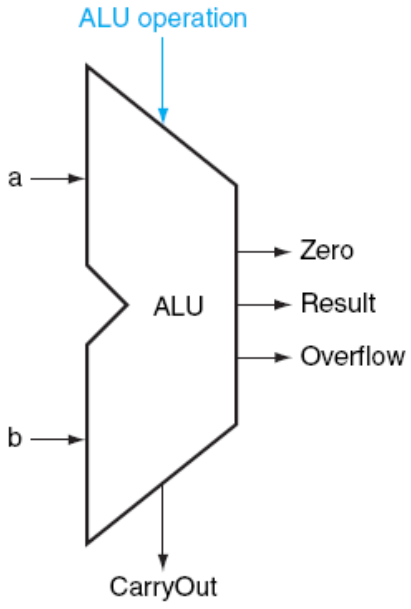


FIGURE C.5.14 The symbol commonly used to represent an ALU, as shown in Figure C.5.12. This symbol is also used to represent an adder, so it is normally labeled either with ALU or Adder.

注意参考学习随书光盘附录C的相关内容

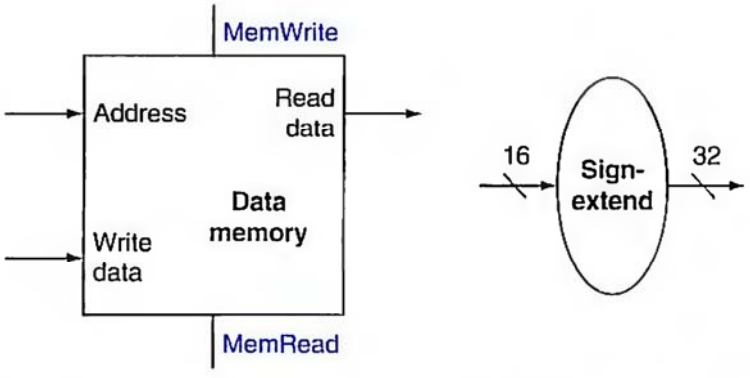
MIPS ALU的一种 Verilog 行为建模实现方法

```
module MIPSALU (ALUctl, A, B, ALUOut, Zero);
   input [3:0] ALUctl;
   input [31:0] A,B;
   output reg [31:0] ALUOut;
   output Zero;
   assign Zero = (ALUOut==0); //Zero is true if ALUOut is 0
   always @(ALUctl, A, B) begin //reevaluate if these change
      case (ALUctl)
         0: ALUOut <= A & B;
         1: ALUOut <= A | B;
         2: ALUOut <= A + B;
         6: ALUOut <= A - B;
         7: ALUOut <= A < B ? 1 : 0;
         12: ALUOut <= ~(A | B); // result is nor
         default: ALUOut <= 0;
      endcase
    end
endmodule
```

FIGURE C.5.15 A Verilog behavioral definition of a MIPS ALU.

主要功能单元实现技术

存取操作数所需要的数据存储器和符号扩展单元



a. Data memory unit b. Sign extension unit

FIGURE 4.8 The two units needed to implement loads and stores, in addition to the register file and ALU of Figure 4.7, are the data memory unit and the sign extension unit. The memory unit is a state element with inputs for the address and the write data, and a single output for the read result. There are separate read and write controls, although only one of these may be asserted on any given clock. The memory unit needs a read signal, since, unlike the register file, reading the value of an invalid address can cause problems, as we will see in Chapter 5. The sign extension unit has a 16-bit input that is sign-extended into a 32-bit result appearing on the output (see Chapter 2). We assume the data memory is edge-triggered for writes. Standard memory chips actually have a write enable signal that is used for writes. Although the write enable is not edge-triggered, our edge-triggered design could easily be adapted to work with real memory chips. See Section C.8 of Appendix C for further discussion of how real memory chips work.

分支指令的数据通路

分支指令的数据通路通过ALU计算分支条件是否成立；而另外的加法器将增值后的PC值与符号扩展后并左移2位的指令低16位（分支偏移量）相加，得到分支的目标地址

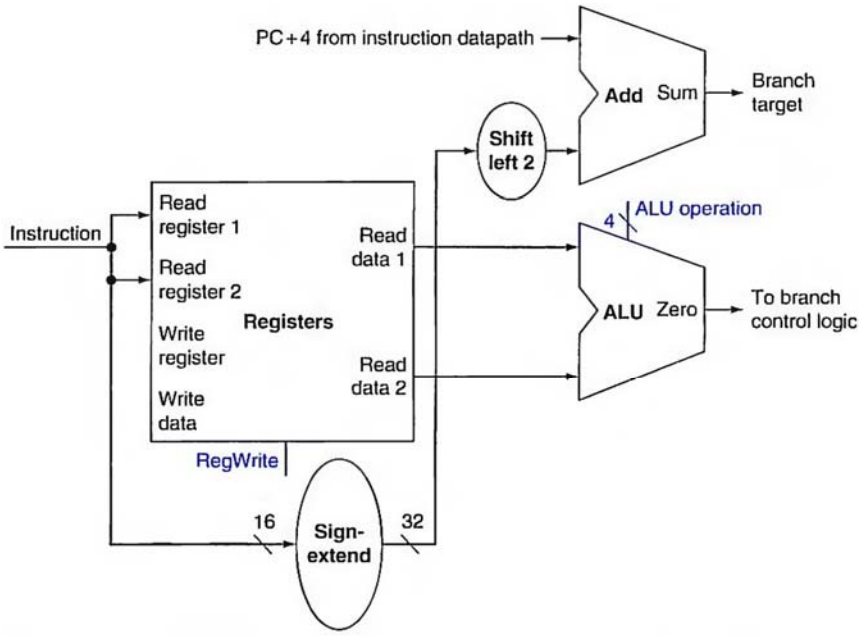


FIGURE 4.9 The datapath for a branch uses the ALU to evaluate the branch condition and a separate adder to compute the branch target as the sum of the incremented PC and the sign-extended, lower 16 bits of the instruction (the branch displacement), shifted left 2 bits. The unit labeled *Shift left 2* is simply a routing of the signals between input and output that

多路复用器

存取指令与R类指令的数据通路

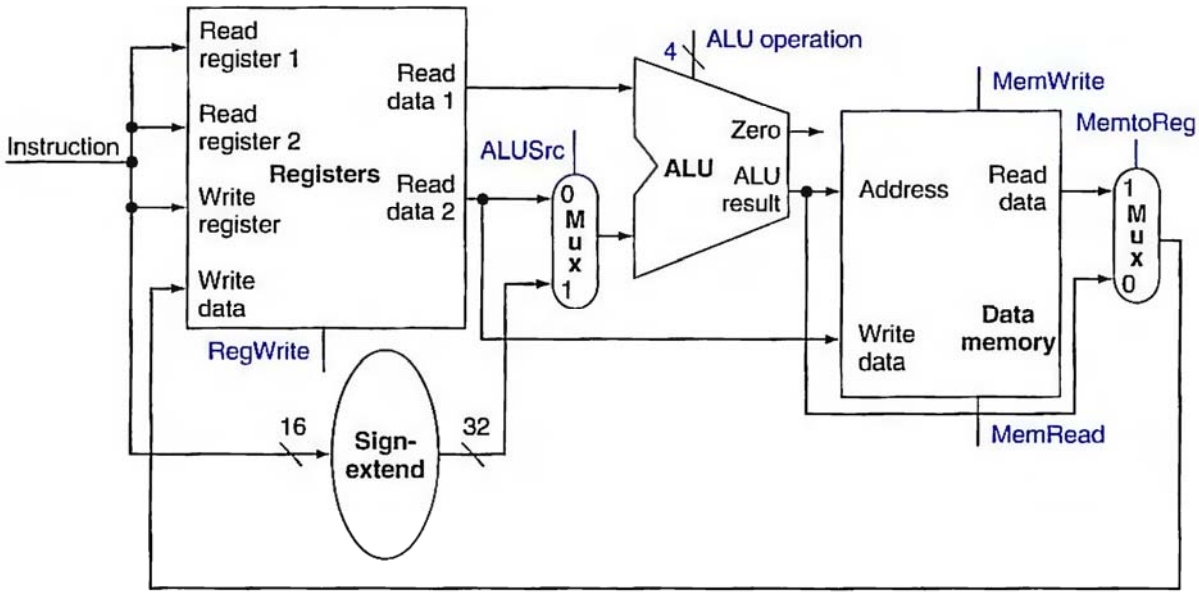


FIGURE 4.10 The datapath for the memory instructions and the R-type instructions. This example shows how a single datapath can be assembled from the pieces in Figures 4.7 and 4.8 by adding multiplexors. Two multiplexors are needed, as described in the example.

多路复用器的实现原理

左图为多路复用器，右图为其门电路实现原理

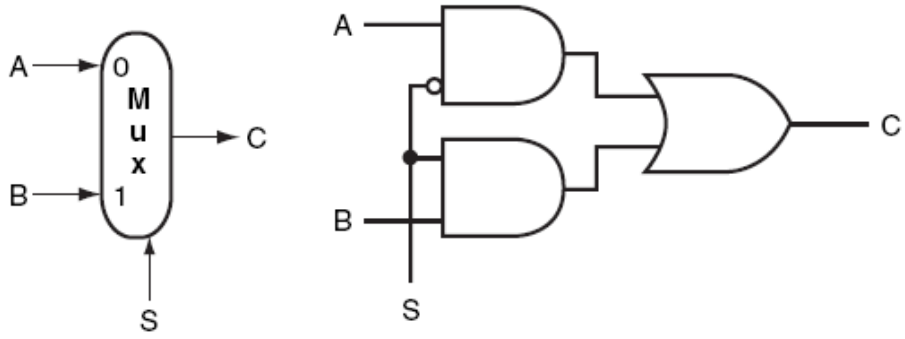


FIGURE C.3.2 A two-input multiplexor on the left and its implementation with gates on the right. The multiplexor has two data inputs (*A* and *B*), which are labeled *0* and *1*, and one selector input (*S*), as well as an output *C*. Implementing multiplexors in Verilog requires a little more work, especially when they are wider than two inputs. We show how to do this beginning on page C-23.

4.4 A Simple Implementation Scheme

不同指令所需要的功能单元所组成的MIPS简单数据通路

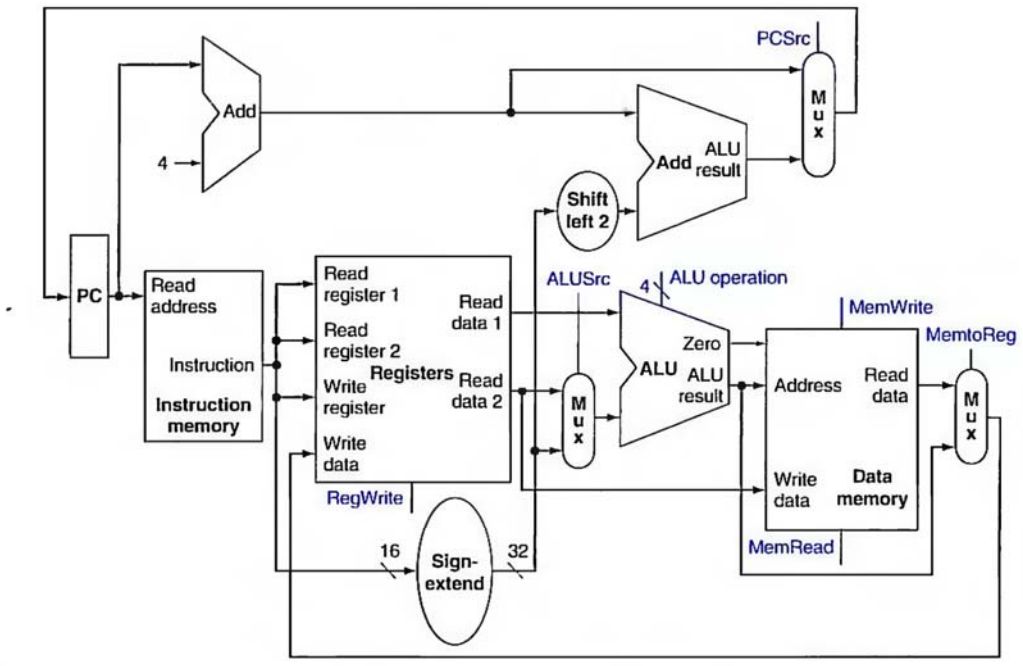


FIGURE 4.11 The simple datapath for the MIPS architecture combines the elements required by different instruction classes. The components come from Figures 4.6, 4.9, and 4.10. This datapath can execute the basic instructions (load-store word, ALU operations, and branches) in a single clock cycle. An additional multiplexor is needed to integrate branches. The support for jumps will be added later.

带有控制单元的简单数据通路

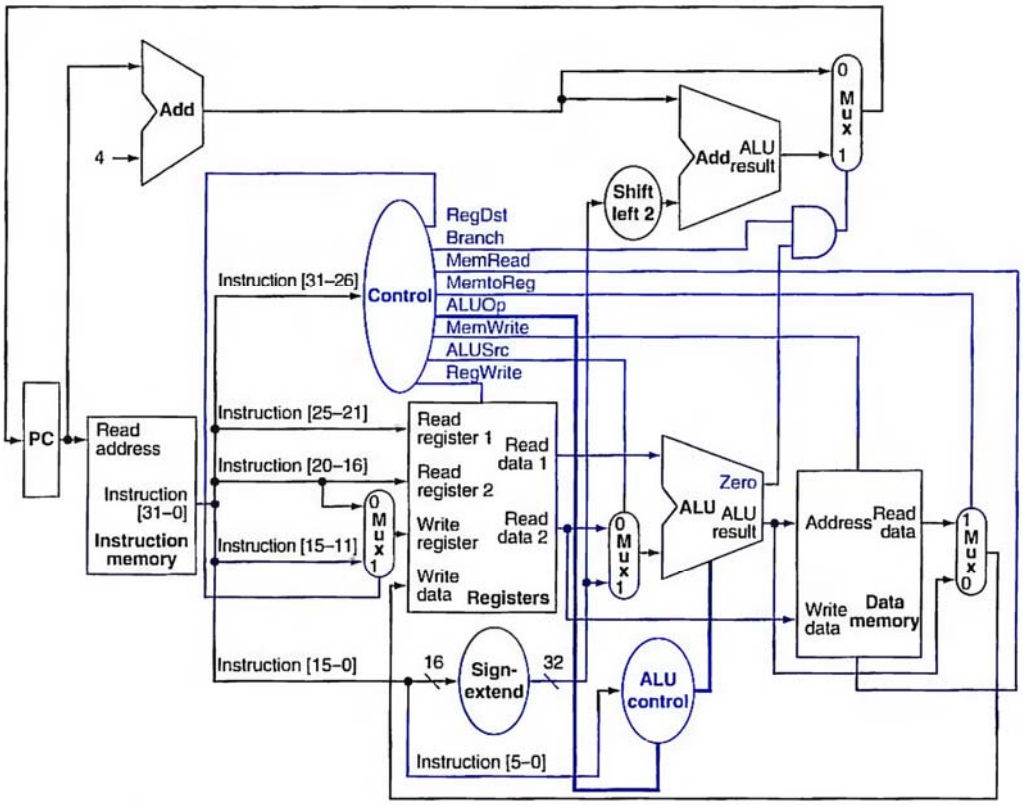
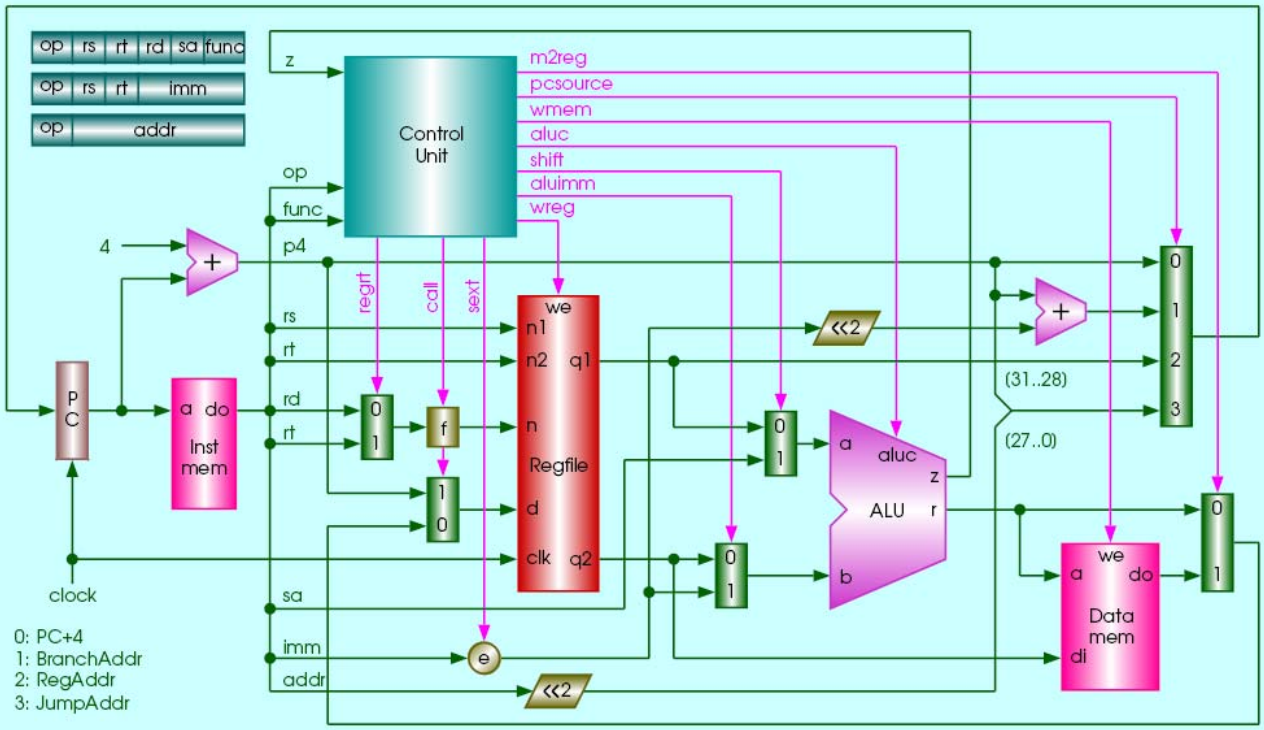


FIGURE 4.17 The simple datapath with the control unit. The input to the control unit is the 6-bit opcode field from the instruction.

单周期CPU+指令存储器+数据存储器



单周期指令执行方式

一个核心问题：前图中的控制器该如何设计？

通用的原理方法：

1. 将指令集中所有指令在执行时所需要的数据通路分别明确；
2. 确定每种指令在执行时的（含各种情况下，大部分指令只有一种情况）所有控制信号；
3. 针对所有指令及其各种情况，建立针对所有控制信号的真值表；
4. 根据以上信息，可以把以操作码作为输入，以控制单元包括所有的输出为输出的逻辑，综合描述为一张大的真值表；
5. 上述真值表所表达的逻辑经化简实现后，即为所需要的控制器。

因该控制器为组合逻辑电路，对于该示例中的单周期指令系统，其控制器属一种硬连线控制器。

前图中的控制器真值表

控制信号线的设置完全取决于指令的操作码字段（含func字段）

Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

FIGURE 4.18 The setting of the control lines is completely determined by the opcode fields of the instruction. The first row of the table corresponds to the R-format instructions (add, sub, AND, OR, and slt). For all these instructions, the source register fields are rs and rt, and the destination register field is rd; this defines how the signals ALUSrc and RegDst are set. Furthermore, an R-type instruction writes a register (RegWrite = 1), but neither reads nor writes data memory. When the Branch control signal is 0, the PC is unconditionally replaced with PC + 4; otherwise, the PC is replaced by the branch target if the Zero output of the ALU is also high. The ALUOp field for R-type instructions is set to 10 to indicate that the ALU control should be generated from the funct field. The second and third rows of this table give the control signal settings for lw and sw. These ALUSrc and ALUOp fields are set to perform the address calculation. The MemRead and MemWrite are set to perform the memory access. Finally, RegDst and RegWrite are set for a load to cause the result to be stored into the rt register. The branch instruction is similar to an R-format operation, since it sends the rs and rt registers to the ALU. The ALUOp field for branch is set for a subtract (ALU control = 01), which is used to test for equality. Notice that the MemtoReg field is irrelevant when the RegWrite signal is 0: since the register is not being written, the value of the data on the register data write port is not used. Thus, the entry MemtoReg in the last two rows of the table is replaced with X for don't care. Don't cares can also be added to RegDst when RegWrite is 0. This type of don't care must be added by the designer, since it depends on knowledge of how the datapath works.

控制器设计真值表示例

指令	指令格式	op	rs	rt	rd	sa	func	z	pcsource [1..0]	aluc [3..0]	shift	aluimm	sext	wmem	wreg	m2reg	regrt	call jal
add	add rd, rs, rt	000000	rs	rt	rd	00000	100000	x	0 0	0 0 0 0	0	0	x	0	1	0	0	0
sub	sub rd, rs, rt	000000	rs	rt	rd	00000	100010	x	0 0	x 1 0 0	0	0	x	0	1	0	0	0
and	and rd, rs, rt	000000	rs	rt	rd	00000	100100											
or	or rd, rs, rt	000000	rs	rt	rd	00000	100101						已删减，由同学自填					
xor	xor rd, rs, rt	000000	rs	rt	rd	00000	100110											
sll	sll rd, rt, sa	000000	00000	rt	rd	sa	000000	x	0 0	0 0 1 1	1	0	x	0	1	0	0	0
srl	srl rd, rt, sa	000000	00000	rt	rd	sa	000010											
sra	sra rd, rt, sa	000000	00000	rt	rd	sa	000011											
jr	jr rs	000000	rs	00000	00000	00000	001000	x	1 0	x x x x	x	x	x	0	0	x	x	x

指令	指令格式	op	rs	rt	rd	sa	func		pcsource [1..0]	aluc [3..0]	shift	aluimm	sext	wmem	wreg	m2reg	regrt	call jal
addi	addi rt, rs, imm	001000	rs	rt	imm				0 0	0 0 0 0	0	1	1	0	1	0	1	0
andi	andi rt, rs, imm	001100	rs	rt	imm				0 0	x 0 0 1	0	1	0	0	1	0	1	0
ori	ori rt, rs, imm	001101	rs	rt	imm													
xori	xori rt, rs, imm	001110	rs	rt	imm													
lw	lw rt, imm(rs)	100011	rs	rt	imm				0 0	x x x x	0	1	1	0	1	1	1	0
sw	sw rt, imm(rs)	101011	rs	rt	imm													
beq	beq rs, rt, imm	000100	rs	rt	imm			0 1	0 0 0 1	x x x x	0	0	1	0	0	0	0	0
bne	bne rs, rt, imm	000101	rs	rt	imm													
lui	lui rt, imm	001111	00000	rt	imm													
j	j addr	000010	addr						1 1	x x x x	x	x	x	0	0	x	x	x
jal	jal addr	000011	addr						1 1	x x x x	x	x	x	0	1	x	x	1

指令	指令格式	op	rs	rt	rd	sa	func		pcsource [1..0]	aluc [3..0]	shift	aluimm	sext	wmem	wreg	m2reg	regrt	call jal

Acknowledgements

- These slides contain material developed and copyright by:
  - —Arvind (MIT)
  - —Krste Asanovic (MIT/UCB)
  - —John Kubiatowicz (UCB)
  - —David Patterson (UCB)
  - —Geng Wang (SJTU)
  - —Yanmin Zhu (SJTU)
  - —Li Yamin(Hosei Univ.)