CHAPTER 5-1

The Processor

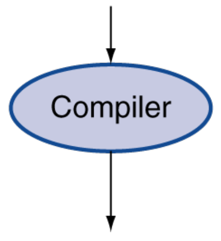
By Pattama Longani
Collage of arts, media and Technology

A BASIC MIPS IMPLEMENTATION

- The memory-reference instructions
  - lw, sw
- The arithmetic-logical instructions
  - add, sub, AND, OR, and slt
- The conditional instructions
  - bne, beq, j

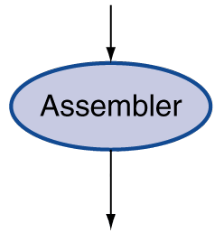
High-level language program (in C)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```



Assembly language program (for MIPS)

```
swap:
      muli $2, $5,4
      add  $2, $4,$2
      lw   $15, 0($2)
      lw   $16, 4($2)
      sw   $16, 0($2)
      sw   $15, 4($2)
      jr   $31
```



Binary machine language program (for MIPS)

```
00000000101000010000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

MIPS **R**-format Instructions

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

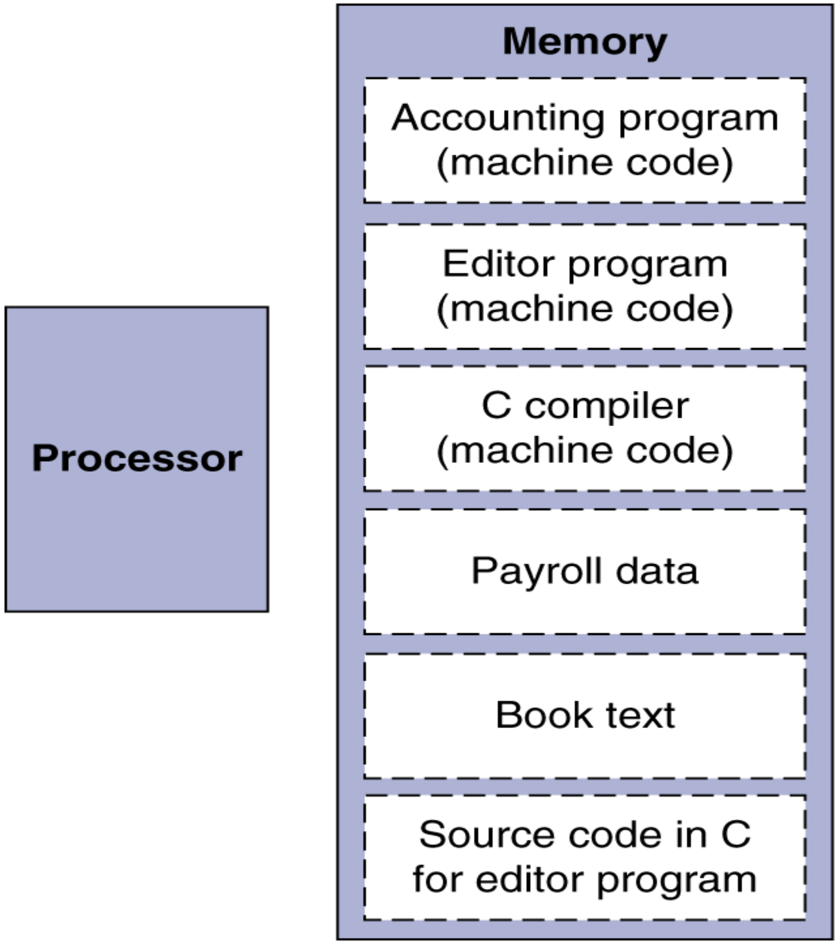
MIPS **I**-format Instructions

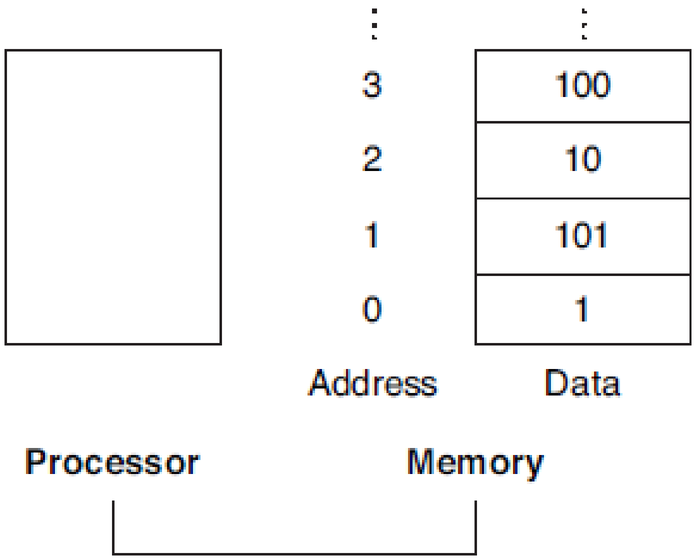
op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

MIPS **J**-format instruction

op	address
6 bits	26 bits

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


Memory
Accounting program (machine code)
Editor program (machine code)
C compiler (machine code)
Payroll data
Book text
Source code in C for editor program
Processor



Hardware

For every instruction, the first two steps are identical:

1. Check program counter (PC):
   - Points to the memory that contains the code and fetch the instruction from that memory.
   - MIPS kept instructions by using 32 bits.
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.

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

- if we pulled the power plug on the computer, we could restart it by loading the state elements with the values they contained before we pulled out the plug.
- inputs come from a set of state elements and its outputs written into a set of state elements

- **Why we have to do the timing of reads and writes??**

a = a+b

c = a-d

- Because if a signal is written at the same time it is read, the value of the read could correspond to the old value, the newly written value, or even some mix of the two! Computer

A clocking methodology is designed to ensure predictability

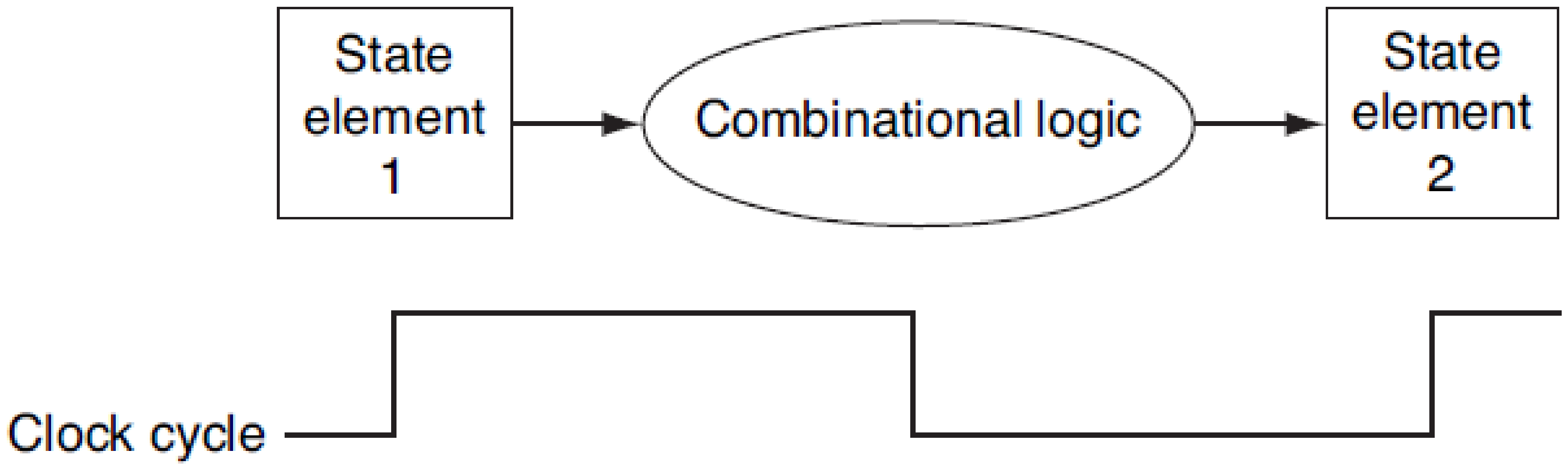
CLOCKING METHODOLOGY

- **Clocking methodology** defines when signals can be read and when they can be written
- **Edge-triggered** : A clocking scheme in which all **state changes occur on a clock edge**.

Read

Do something

Write



CONTROL SIGNAL

- To inform computer what to do.
- Both the clock signal and the write control signal are inputs, and the state element is changed only when the write control signal is asserted and a clock edge occurs.

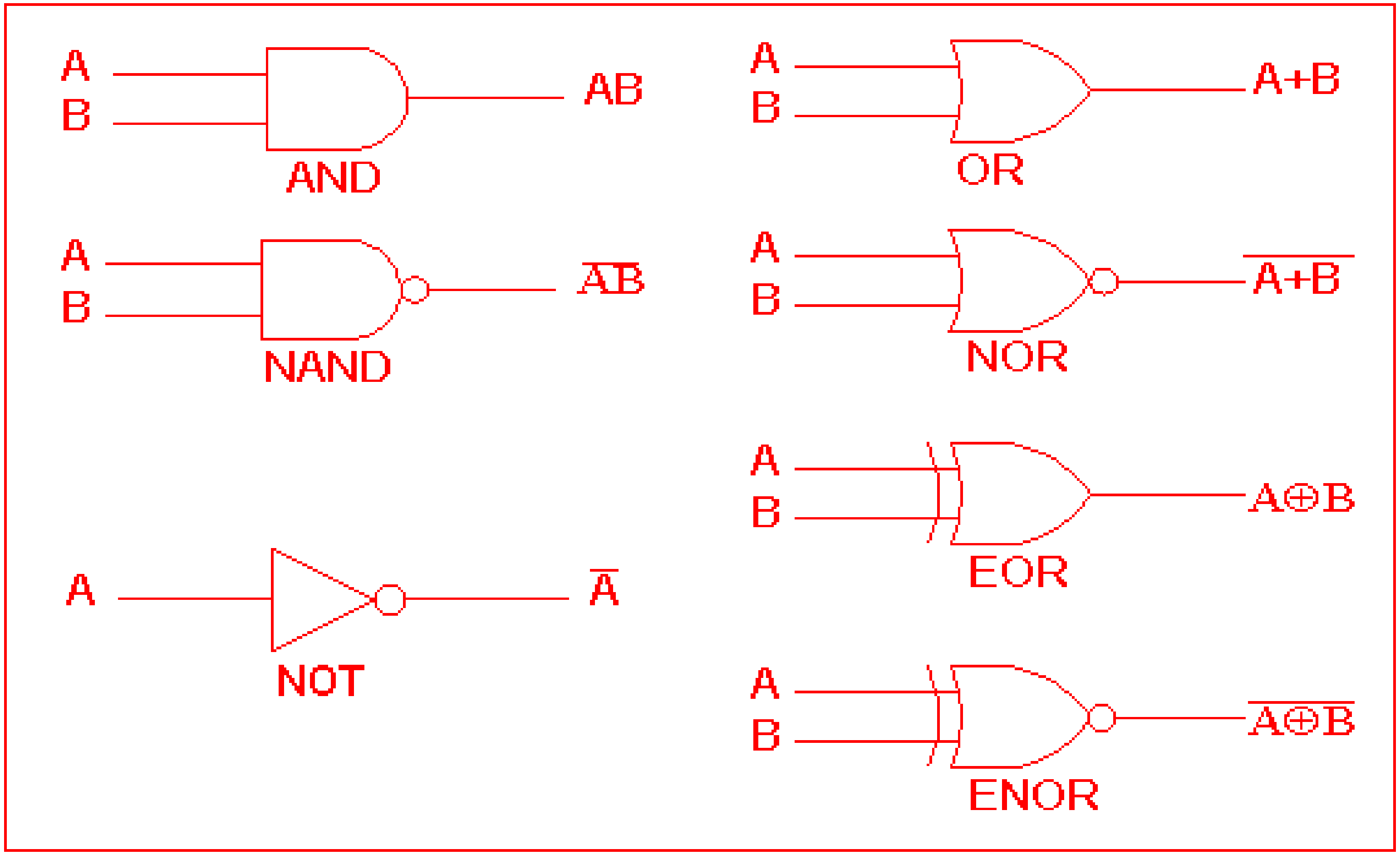
LOGIC ELEMENTS

datapath element : A components working with processor. In the MIPS implementation, includes

- The instruction and data memories
- the register file
- the ALU
- adders.

Next, we will examine the major components required to execute each class of MIPS instructions

Logic gate

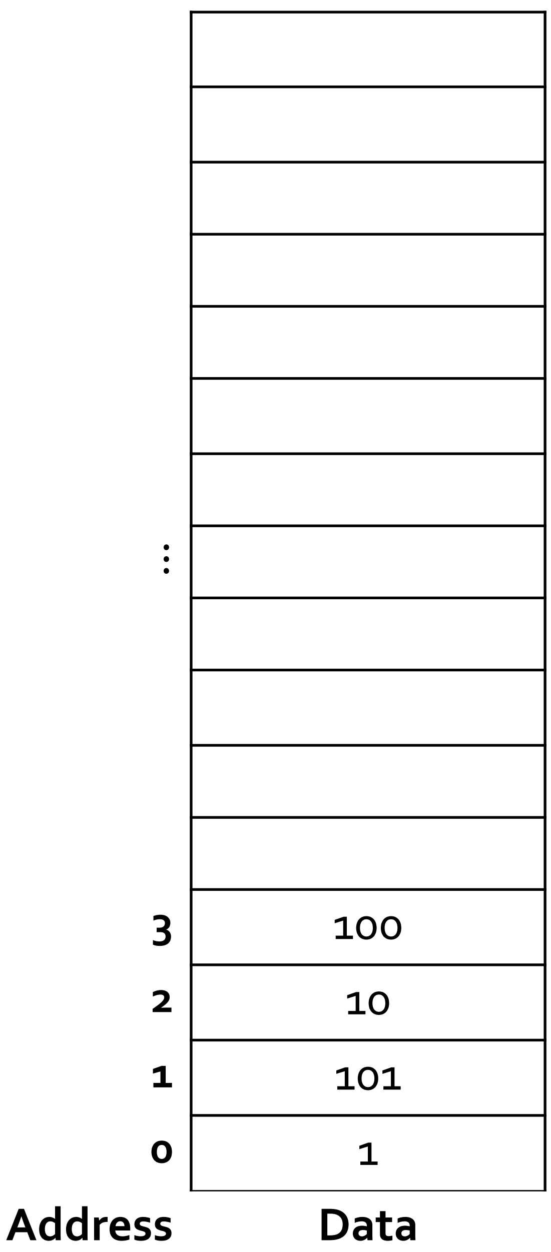

A
B
AND
AB
A
B
NAND
$\overline{AB}$
A
NOT
$\overline{A}$
A
B
OR
A+B
A
B
NOR
$\overline{A+B}$
A
B
EOR
$A \oplus B$
A
B
ENOR
$\overline{A \oplus B}$

To execute any instruction,

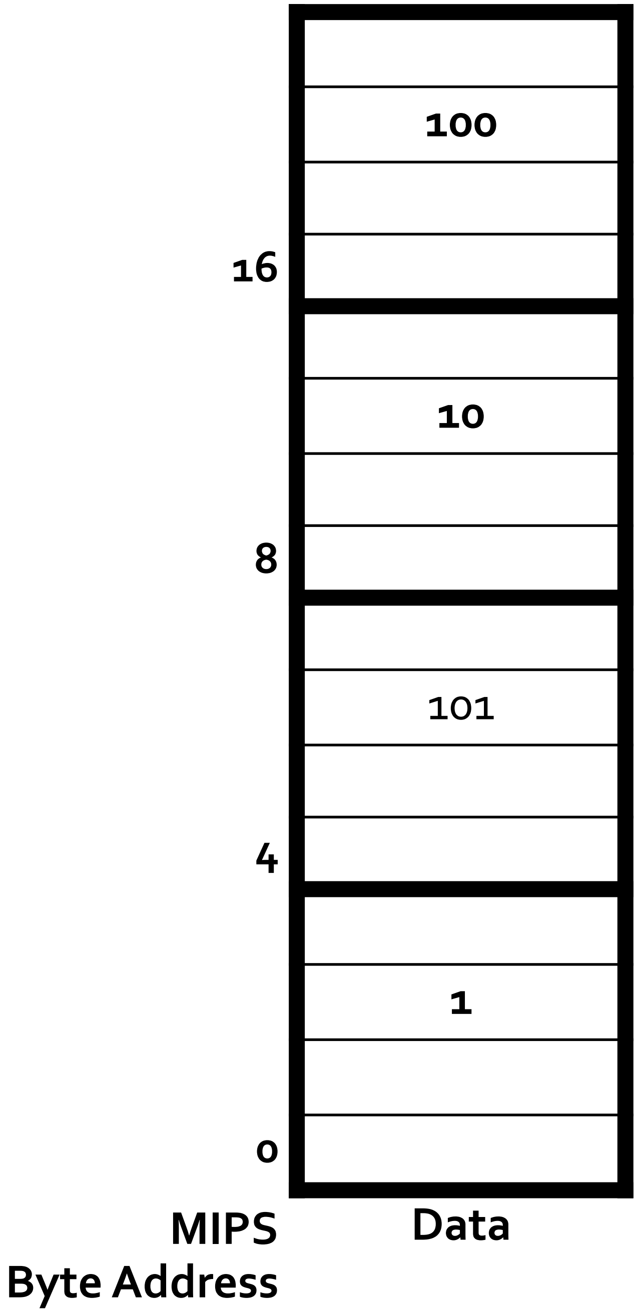
1 we must **fetch** the instruction from memory.

2.To prepare for executing the next instruction,

- we must also increment the program counter to points at the next instruction.
- Because MIPS uses 4 bytes word, so we point to the next instruction 4 bytes later.



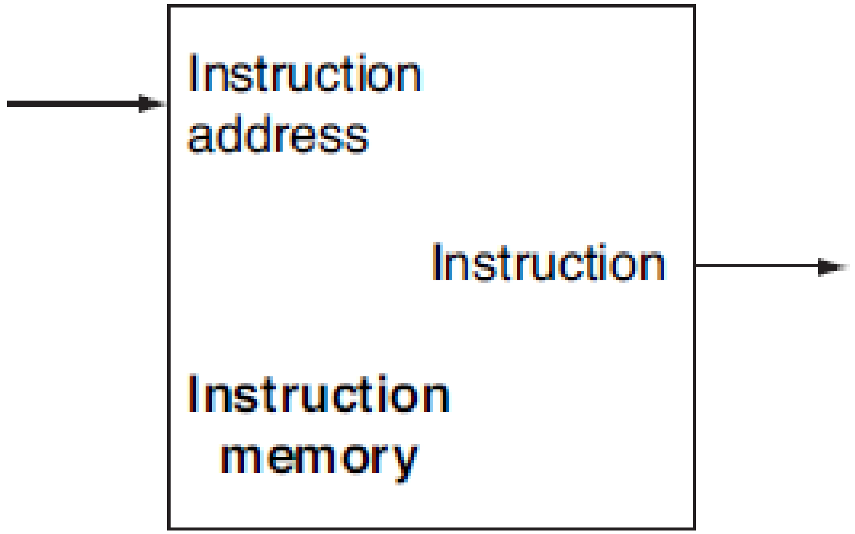
⋮
3
100
2
10
1
101
0
1
Address
Data



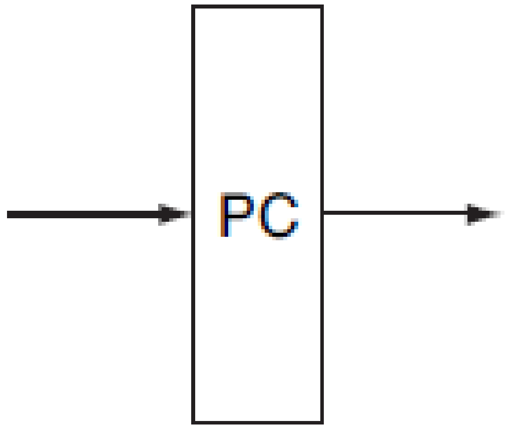
100
16
10
8
101
4
1
0
MIPS Byte Address
Data

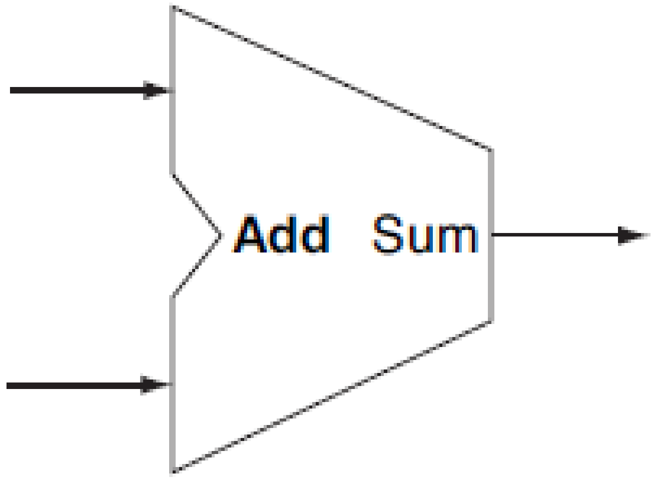
BUILDING DATAPATH

- **memory unit**
  - store the instructions of a program
  - supply instructions given an address



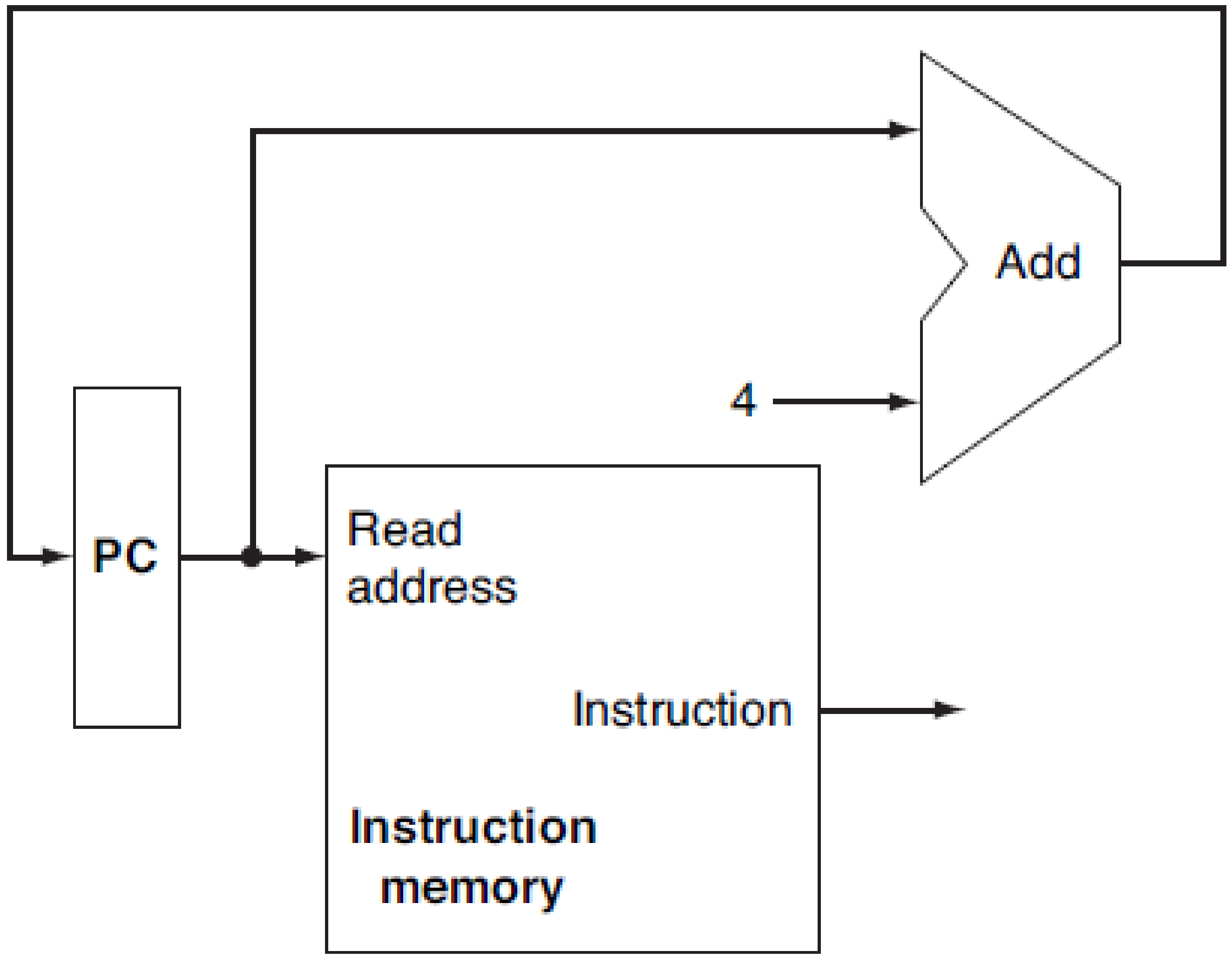
- **program counter (PC)**
  - a register that holds the address of the current instruction
- **adder**
  - to increment the PC to the address of the next instruction.





BUS

- For the 32-bit MIPS architecture, We use *buses* to carry the 32-bit signal
  - We may want to combine several buses to form a wider bus; for example, we may want to obtain a 32-bit bus by combining two 16-bit buses.


Add
4
PC
Read
address
Instruction
Instruction
memory

R-FORMAT INSTRUCTIONS

- They all read two registers, perform an ALU operation on the contents of the registers, and write the result to a register.
- This instruction class includes add, sub, AND, OR, and slt, which were introduced in Chapter 2.

```
add $t1,$t2,$t3
```

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

R-FORMAT INSTRUCTIONS

- R-format instructions have three register operands
  - read two data words from the register file
  - write one data word into the register file for each instruction.
- **register file** : a state element that consists of a set of registers that can be read and written
  - supplying a register number to be accessed.

R-FORMAT INSTRUCTIONS

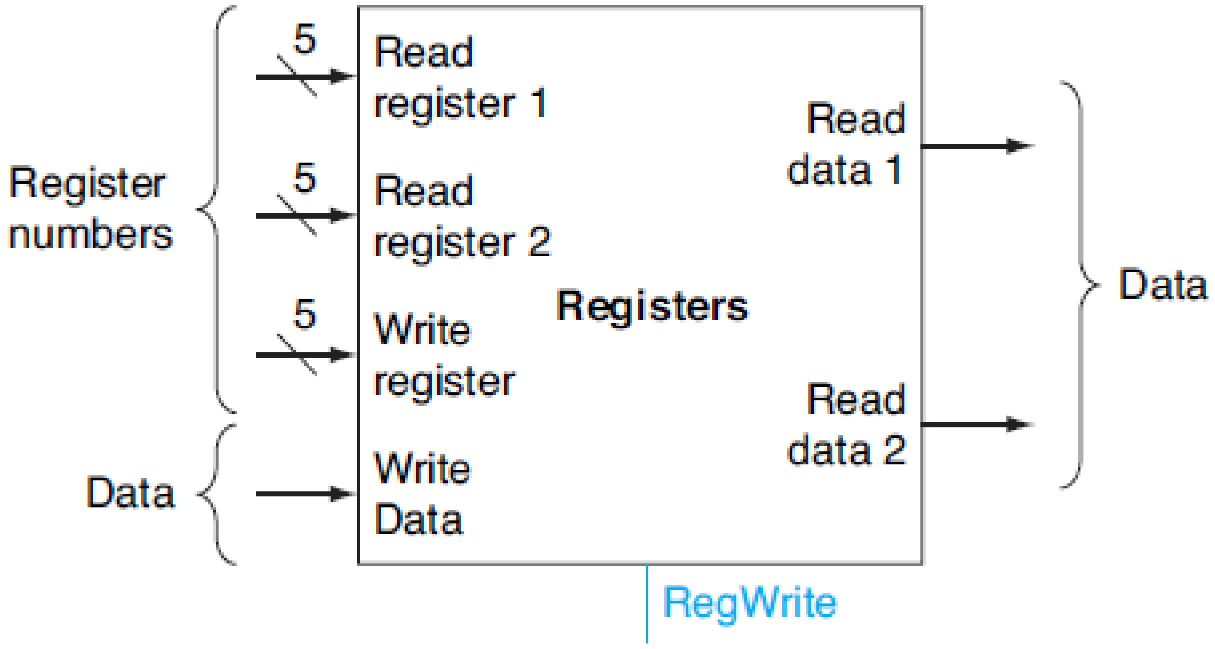
- **To read** we need
  - the <u>register number</u> to be read
  - an output from the register file that will carry the value that has been read from the registers.
- **To write** we need
  - the *register number* to be written
  - the *data* to be written into the register

R-FORMAT INSTRUCTIONS

- The register file always **outputs** the contents of whatever **register numbers** are on the Read register inputs.
- Writes are controlled by the **write control signal**, which must be asserted for a write to **occur at the clock edge.**

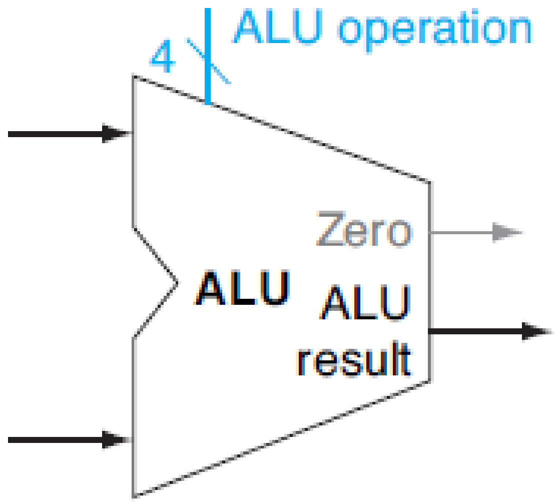
R-FORMAT INSTRUCTIONS

- The register number inputs are 5 bits wide to specify one of 32 registers
- The data input and two data output buses are each 32 bits wide.



BUILDING DATAPATH

- **ALU**
  - which takes two 32-bit inputs
  - produces a 32-bit result
  - 1-bit signal if the result is 0.
  - The 4-bit control signal, define operations



LOAD-STORE INSTRUCTIONS

- If the instruction is a load, the value read from memory must be written into the register file in the specified register
- Thus, we will need both the register file and the ALU

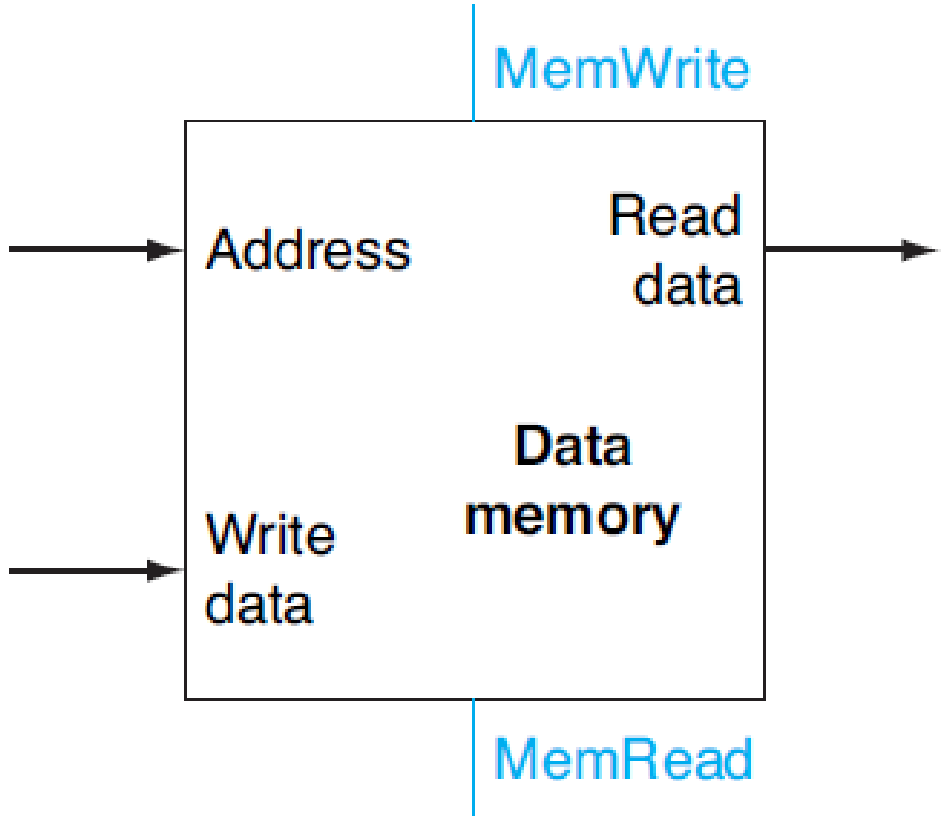
op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

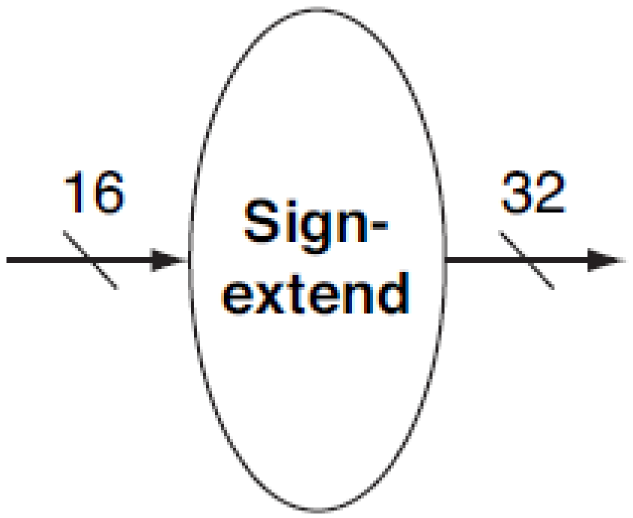
```
lw  $t1,offset_value($t2)
```

LOAD-STORE INSTRUCTIONS

- We need
- **sign-extend** the 16-bit offset field in the instruction to a 32-bit signed value
- **memory unit** to read from or write to.
  - The data memory must be written on store instructions
  - has read and write control signals
  - an address input
  - the data to be written into memory.

LOAD-STORE INSTRUCTIONS





BRANCH INSTRUCTIONS

- has three operands
  - **two registers** that are compared for equality
  - **branch target address**: 16-bit offset

```
bne  $s0,$s1,Exit
```

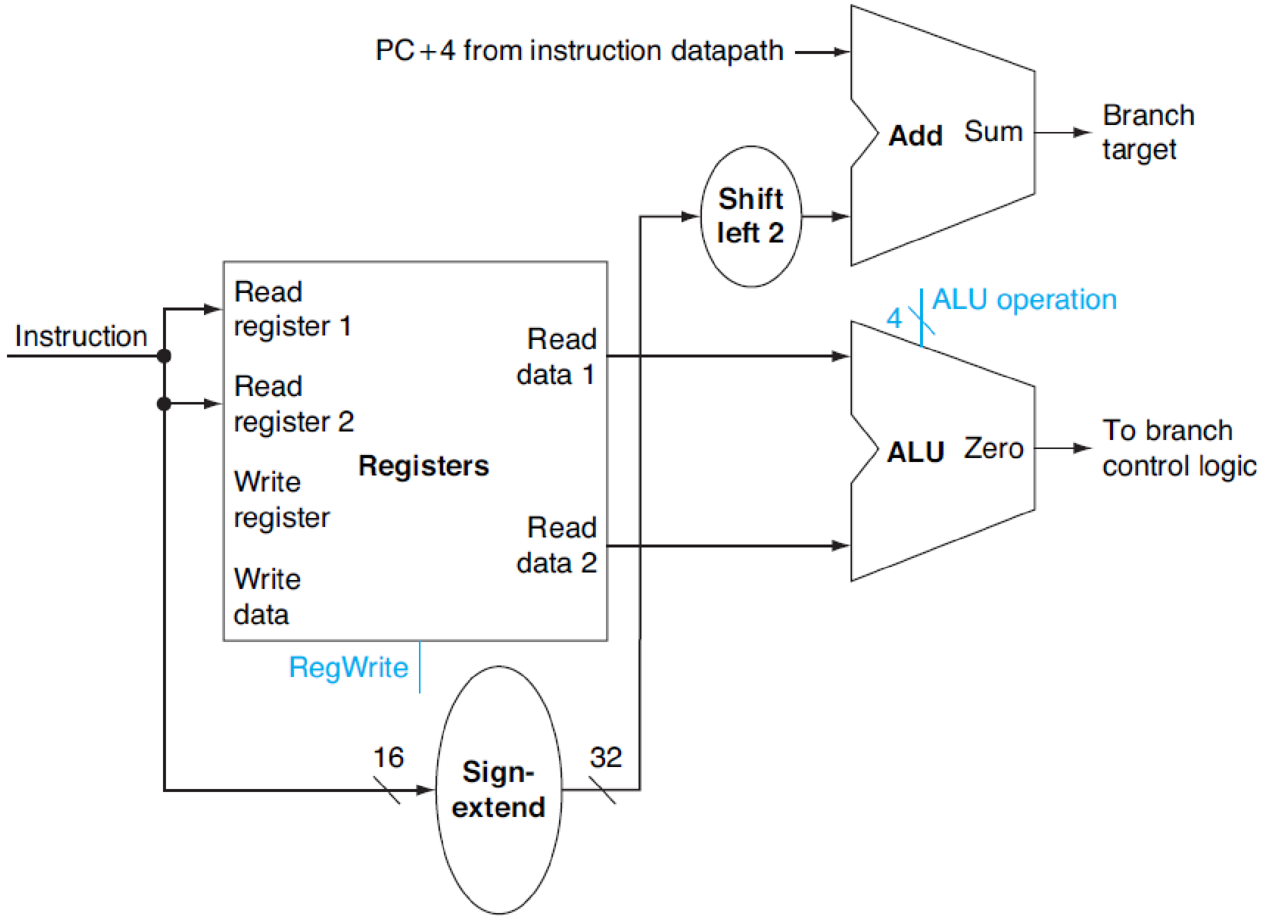
5	16	17	Exit
6 bits	5 bits	5 bits	16 bits

BRANCH INSTRUCTIONS

- we must determine whether the next instruction address
  - **branch is taken**, the branch target address becomes the new PC (the incremented PC should replace the current PC)
  - **branch** is **not taken**, just as for any other normal instruction
    - PC + 4
    - shifted left 2 bits

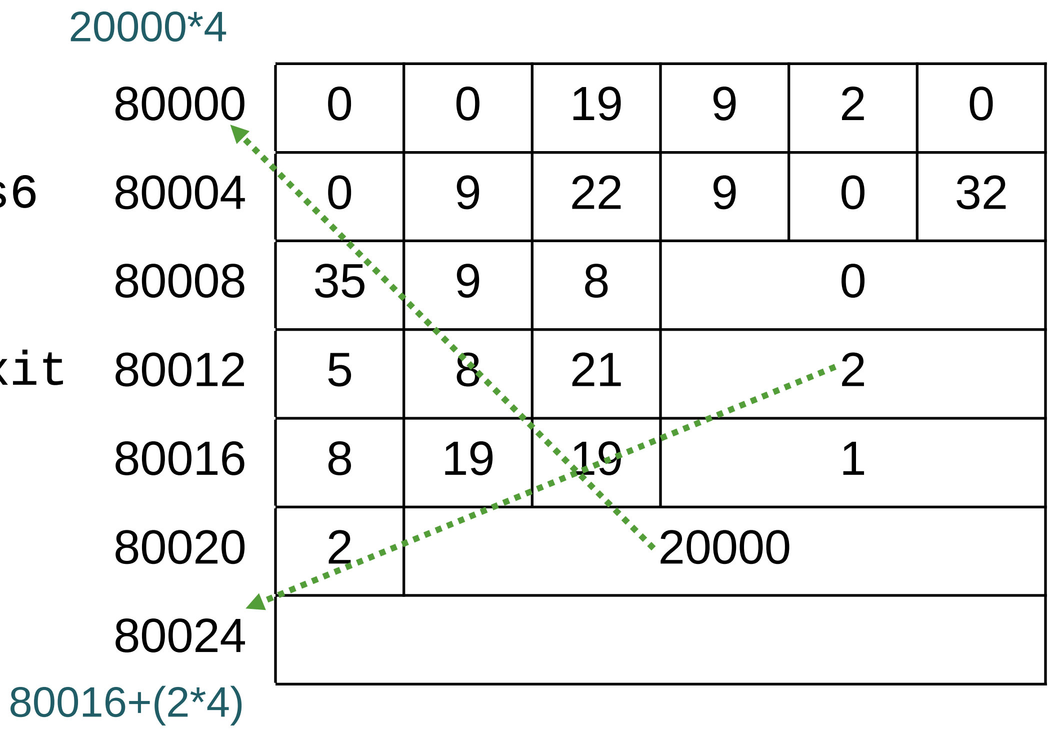
BRANCH INSTRUCTIONS

- The branch datapath must do two operations:
  - compute the branch target address
  - compare the register contents



Target Addressing Example

- Loop code from earlier example
  - Assume Loop at location 80000



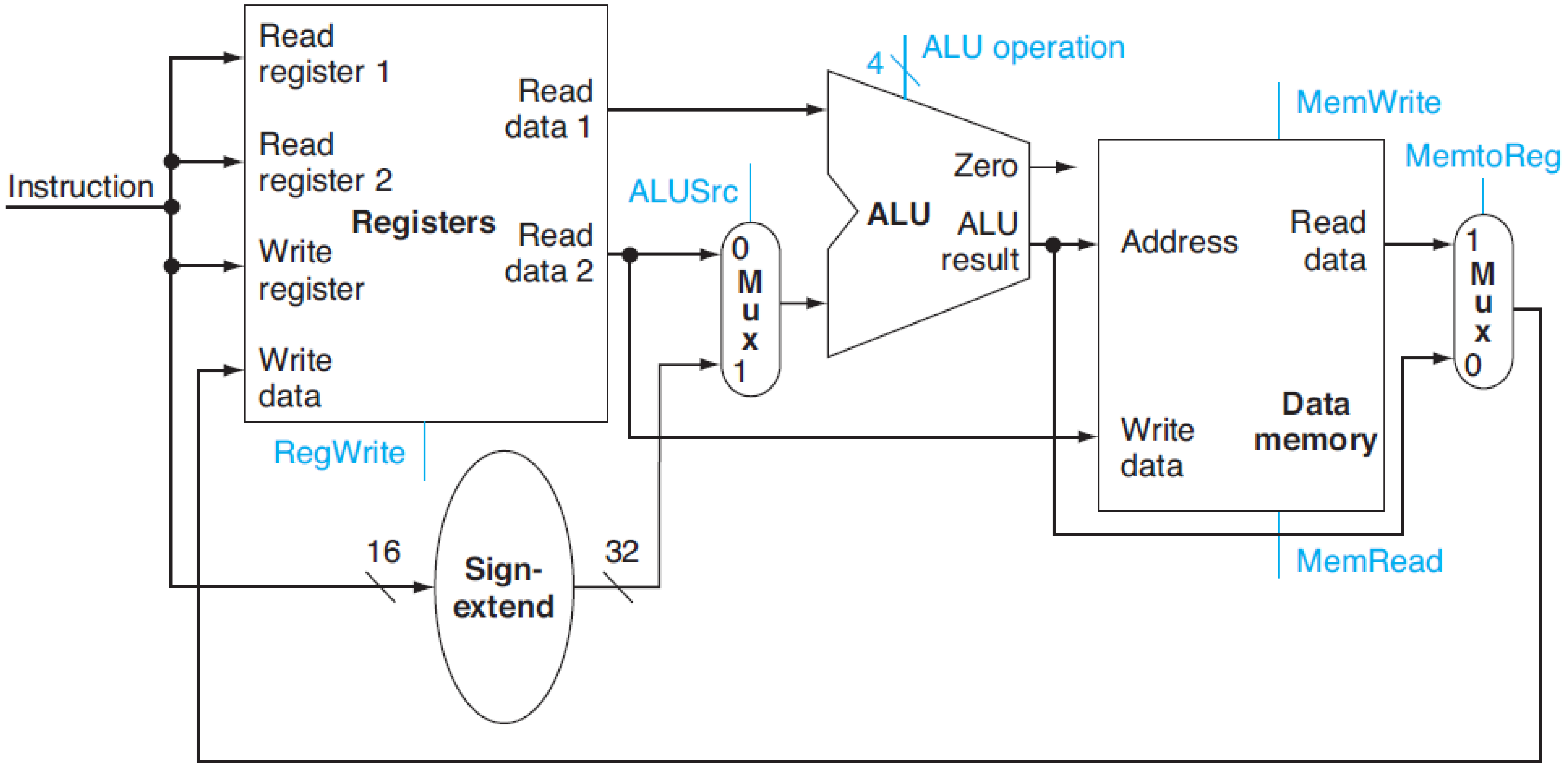
```
Loop: sll  $t1, $s3, 2       80000
      add  $t1, $t1, $s6     80004
      lw   $t0, 0($t1)       80008
      bne  $t0, $s5, Exit    80012
      addi $s3, $s3, 1       80016
      j    Loop              80020
Exit: …                      80024
```

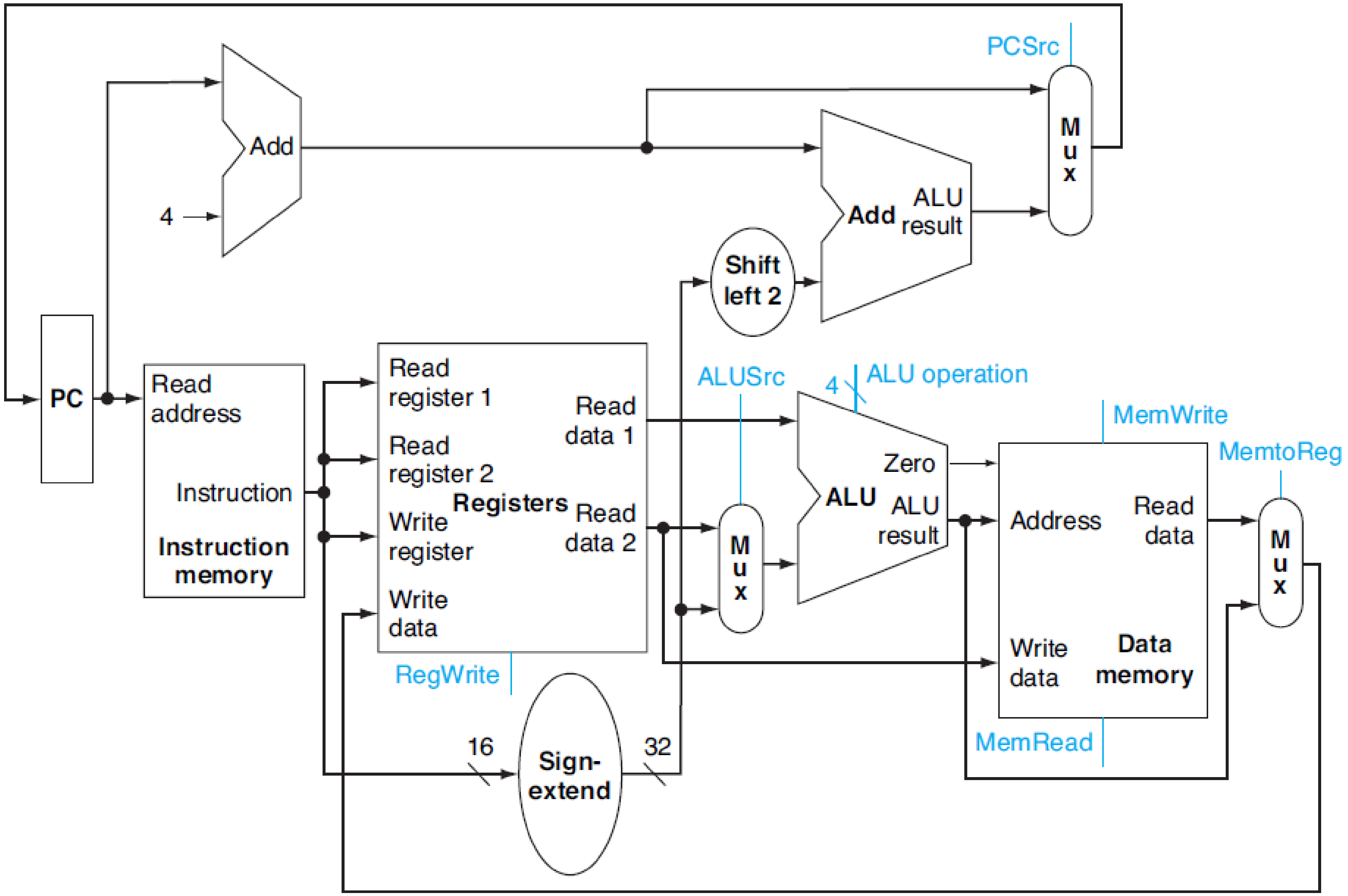
80000	0	0	19	9	2	0
80004	0	9	22	9	0	32
80008	35	9	8	0		
80012	5	8	21	2		
80016	8	19	19	1		
80020	2	20000				
80024						

COMBINE DATAPATHS

- We can combine them into a single datapath and **add a control**
- Execute all instructions in one clock cycle.
    - no datapath resource can be used more than once per instruction, so any element needed more than once must be duplicated.
- memory for instructions is separated from memory for data.

COMBINE DATAPATHS




PCSrc
Add
4
M u x
Add
ALU result
Shift left 2
PC
Read address
Instruction
Instruction memory
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
RegWrite
ALUSrc
4
ALU operation
M u x
ALU
Zero
ALU result
MemWrite
MemtoReg
Address
Read data
Write data
Data memory
MemRead
M u x
16
Sign-extend
32