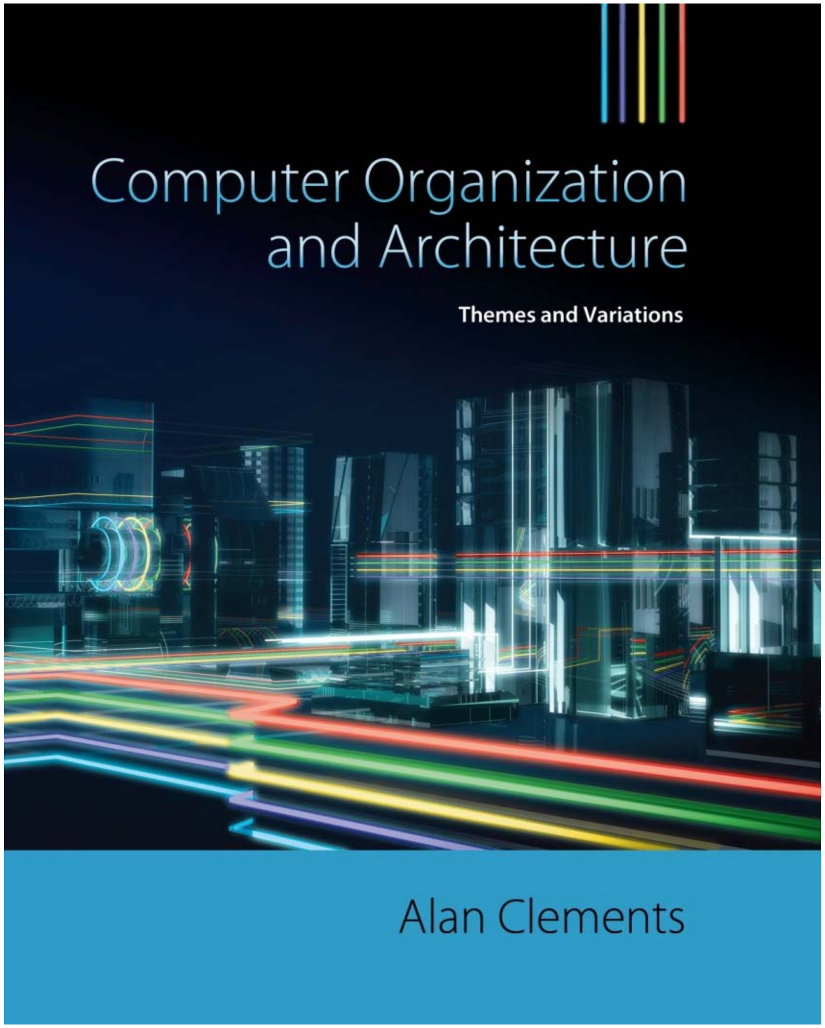
Part 0x1

Chapter 3

Architecture and Organization



These slides are being provided with permission from the copyright for in-class (CS2208B) use only. The slides must not be reproduced or provided to anyone outside of the class.

All download copies of the slides and/or lecture recordings are for personal use only. Students must destroy these copies within 30 days after receipt of final course evaluations.

Music: *"Corporate Success"* by *Scott Holmes*, used under *Attribution-NonCommercial License*

CENGAGE Learning

The Instruction Set Architecture

In this chapter, we will:

- ❑ **Revisit** the *stored program machine* and **show** how an instruction is executed
- ❑ **Introduce** instruction formats, including
  - ○ *memory-to-memory*,
  - ○ *register-to-memory*, and
  - ○ *register-to-register*
- ❑ **Demonstrate** how a processor implements *conditional behavior*
- ❑ **Describe** a set of computer assembly instructions (*instructions set*)
- ❑ **Show** how computers access data (*addressing modes*)
- ❑ **Introduce** an ARM's *Integrated Development Environment* (IDE) and
  show how ARM programs are written

The Instruction Set Architecture

- ❑ Figure 3.1 illustrate the structure of a **simple** *hypothetical* *stored program computer*.
- ❑ The ***CPU*** *reads* instructions from memory and *executes* them.
- ❑ *Temporary data* is stored in *registers* such as *r0* and *r1*.

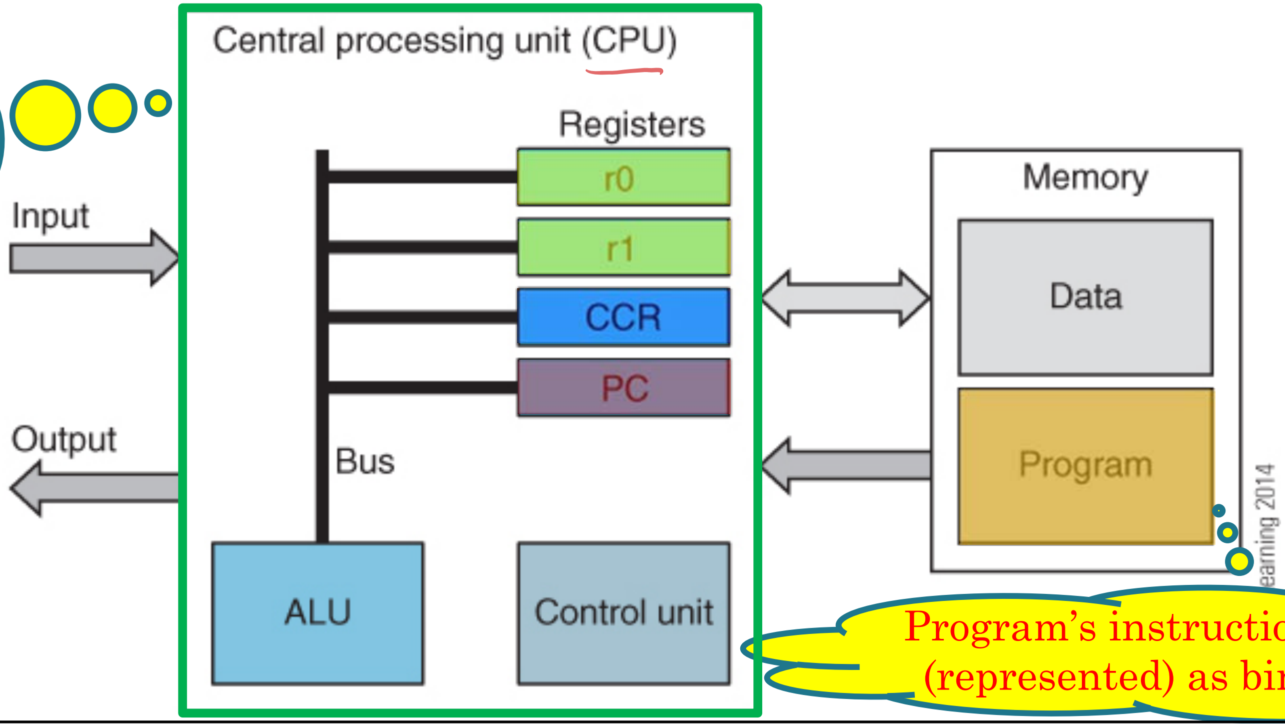
registers -

- ❑ The ***PC***, *program counter*, is the register that *points at (i.e., contains the address of) the next instruction to be executed*.
- ❑ The ***CCR***, *Condition Code Register*, is a collection of flag bits for a processor.

Intel calls it the "***Instruction Address Pointer***", a better name than the ***PC***

FIGURE 3.1 Fundamental structure of a computer

This is a bit detailed version of Figure 1.3, Slide 9, Chapter 1.



earning 2014

Program's instructions are encoded (represented) as binary numbers.

This slide is modified fro

Instruction Formats

- A computer executes instructions from 8 bits wide to multi-bytes wide.
- The instruction format defines the ***anatomy** of an instruction*
  - the number of operands, and
  - the number of bits devoted to defining each operation,
  - the format of each operand.
- Below are several *hypothetical* examples of assembly instructions:

LDR **registerDestination**,memorySource
STR registerSource,**memoryDestination**
Operation **registerDestination**,registerSource1,registerSource2

LDR	**r1**,1234	load the content on location 1234 to r1
STR	r3,**2000**	store the content in r3 to location 2000.
ADD	**r1**,r2,r3	add up r1, r2 and store the result in r3
SUB	**r3**,r3,r1	r3 = r3 - r1.

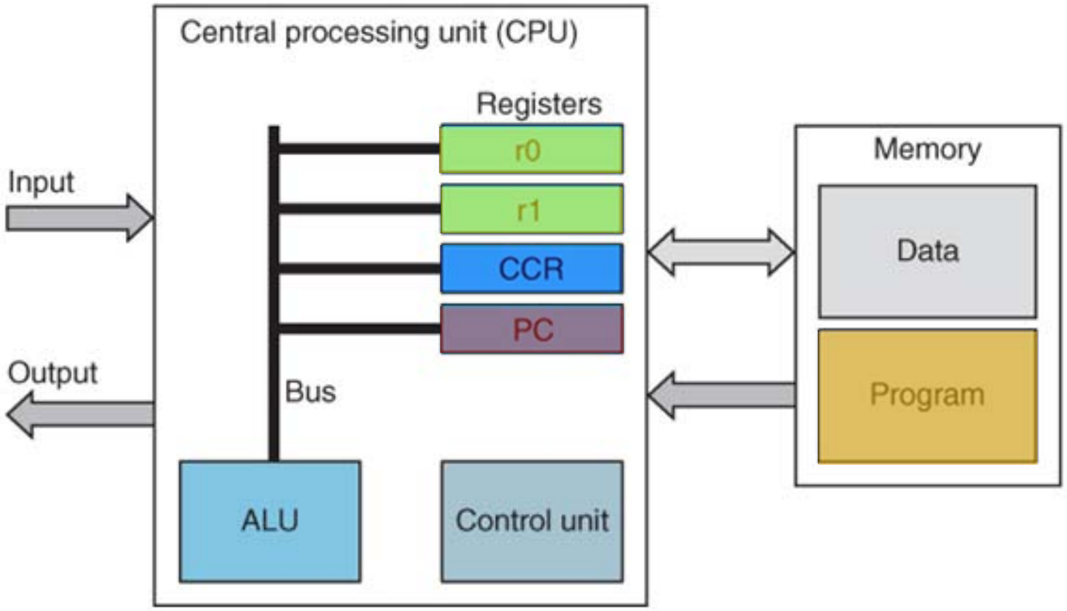
bold stands the destination memory.

Features

- ❑ A stored program machine is a computer that has a program in binary form in its main memory.
- ❑ The program and data are stored in the same memory.
- ❑ The program counter (PC) points to the next instruction to be executed and is incremented after executing each instruction.

Fundamental structure of a computer

FIGURE 3.1



© Cengage Learning 2014

- ❑ A stored program operates in a *fetch*/execute two-phase mode.
  - ○ In the *fetch phase* the next instruction is read from memory and decoded. memory —> CPU
  - ○ In the *execute phase* the instruction is interpreted and executed by the CPU's logic. CPU excute, meanwhile memory doing nothing.

Review Slides 27 and 28 in Chapter 1.

- ❑ Modern computers are ***pipelined***, where fetch and execute operations ***overlap***.

Fetch Excute
Fetch Excute
Fetch Excute.

Let both CPU and memory work

Features

A stored program computer has several registers.

r0 - *ri* The register file is a *set of general-purpose registers*, e.g., r0, r1, r2, …, r*i* that store temporary (working) data, for example, the intermediate results of calculations, where *i* is typically 8, 16, or 32.

A computer requires at *least one* general-purpose register.

PC The *program counter* contains the *address* of the next instruction to be executed. Thus, the PC *points* to the location in memory that holds the next instruction.

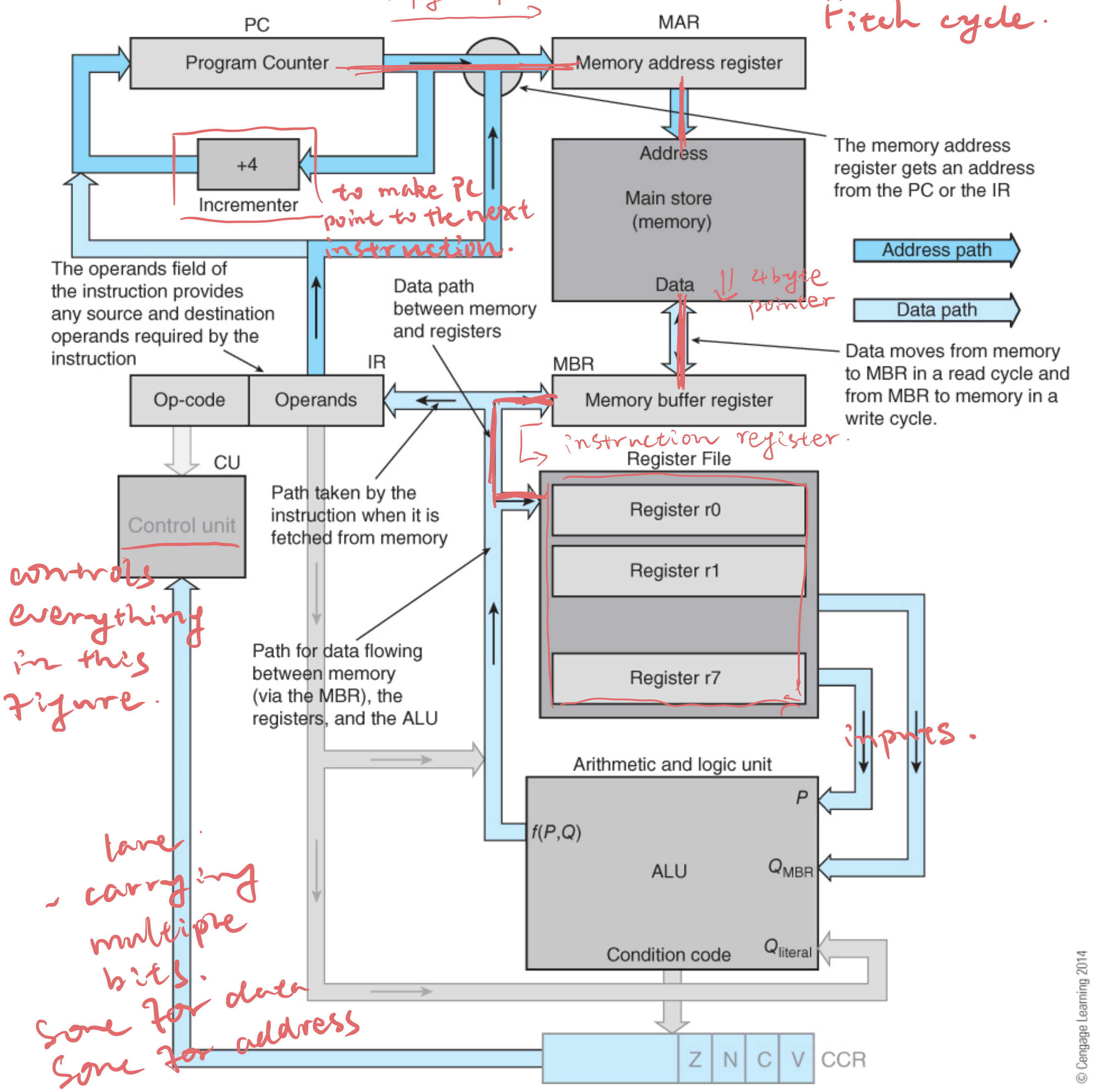
IR The *instruction register* stores the instruction most recently read from main memory. This is the instruction currently being executed.

MAR The *memory address register* stores the *address* of the location in main memory that is currently being accessed by a read or write operation.

MBR The *memory buffer register* stores data that has just been read from main memory, or data to be immediately *written* to main memory.

Structure of a Computer

FIGURE 3.2 Partial structure of a hypothetical stored program machine

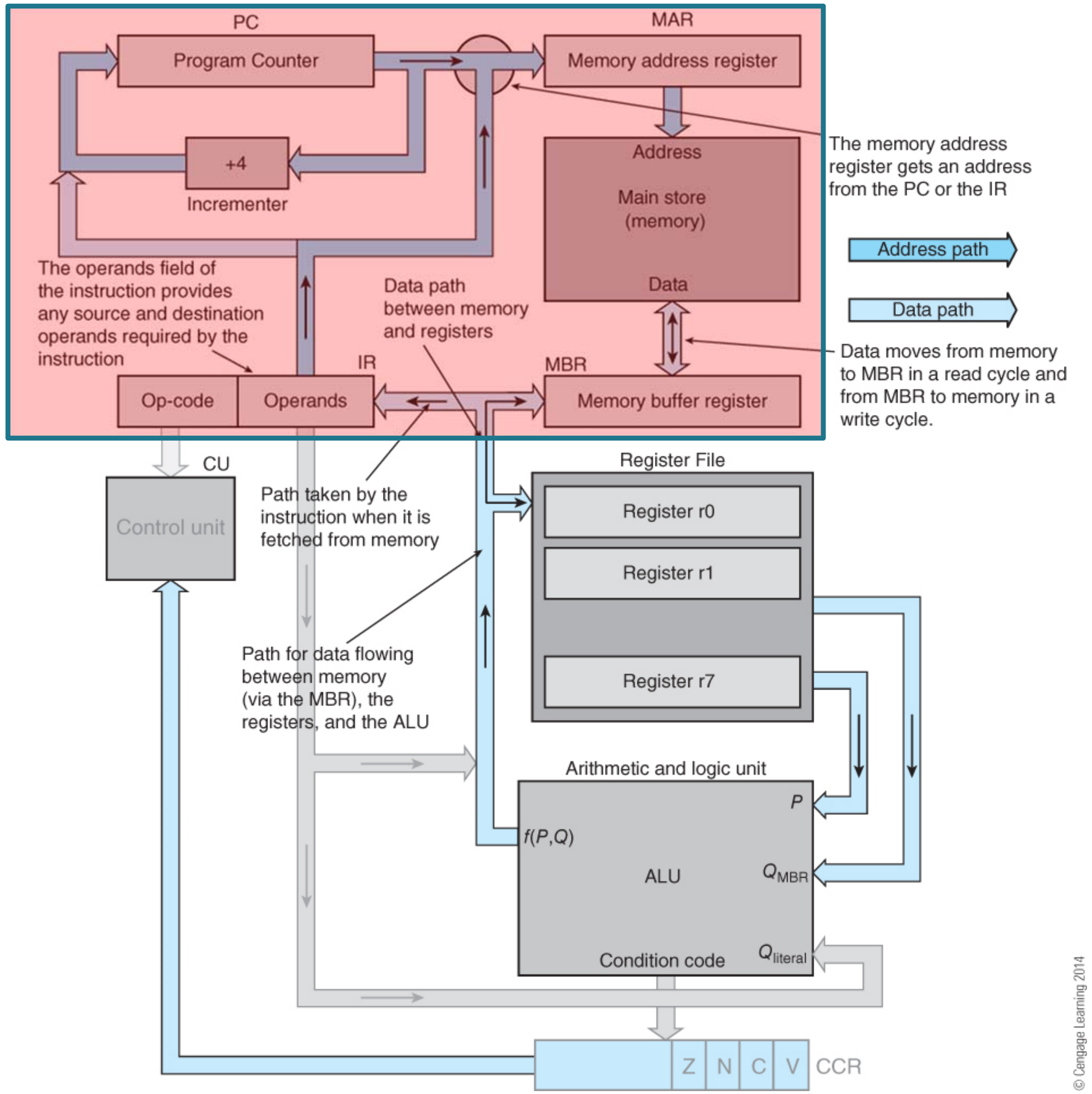

copy the pointer.
Fitch cycle.
PC
Program Counter
MAR
Memory address register
+4
Incrementer
to make PC point to the next instruction.
Address
Main store
(memory)
Data
4byte pointer
The memory address register gets an address from the PC or the IR
Address path
Data path
The operands field of the instruction provides any source and destination operands required by the instruction
Data path between memory and registers
IR
MBR
Op-code
Operands
Memory buffer register
Data moves from memory to MBR in a read cycle and from MBR to memory in a write cycle.
instruction register.
CU
Control unit
Register File
Register r0
Register r1
Register r7
Path taken by the instruction when it is fetched from memory
controls everything in this figure.
Path for data flowing between memory (via the MBR), the registers, and the ALU
inputs.
Arithmetic and logic unit
P
f(P,Q)
ALU
Q_{MBR}
$Q_{literal}$
Condition code
lane
- carrying multiple bits.
Some for data
Some for address
Z
N
C
V
CCR
© Cengage Learning 2014

- ❑ We are going to use an ARM processor to introduce assembly language and a modern ISA.
- ❑ *However*, it would be better to begin with the description of a very simple =D *hypothetical* computer to keep things simple.

This slide is modified from the original slide by the author A. Clements and used with permission. New content added and copyrighted by © Mahmoud R. El-Sakka.

Structure of a Computer

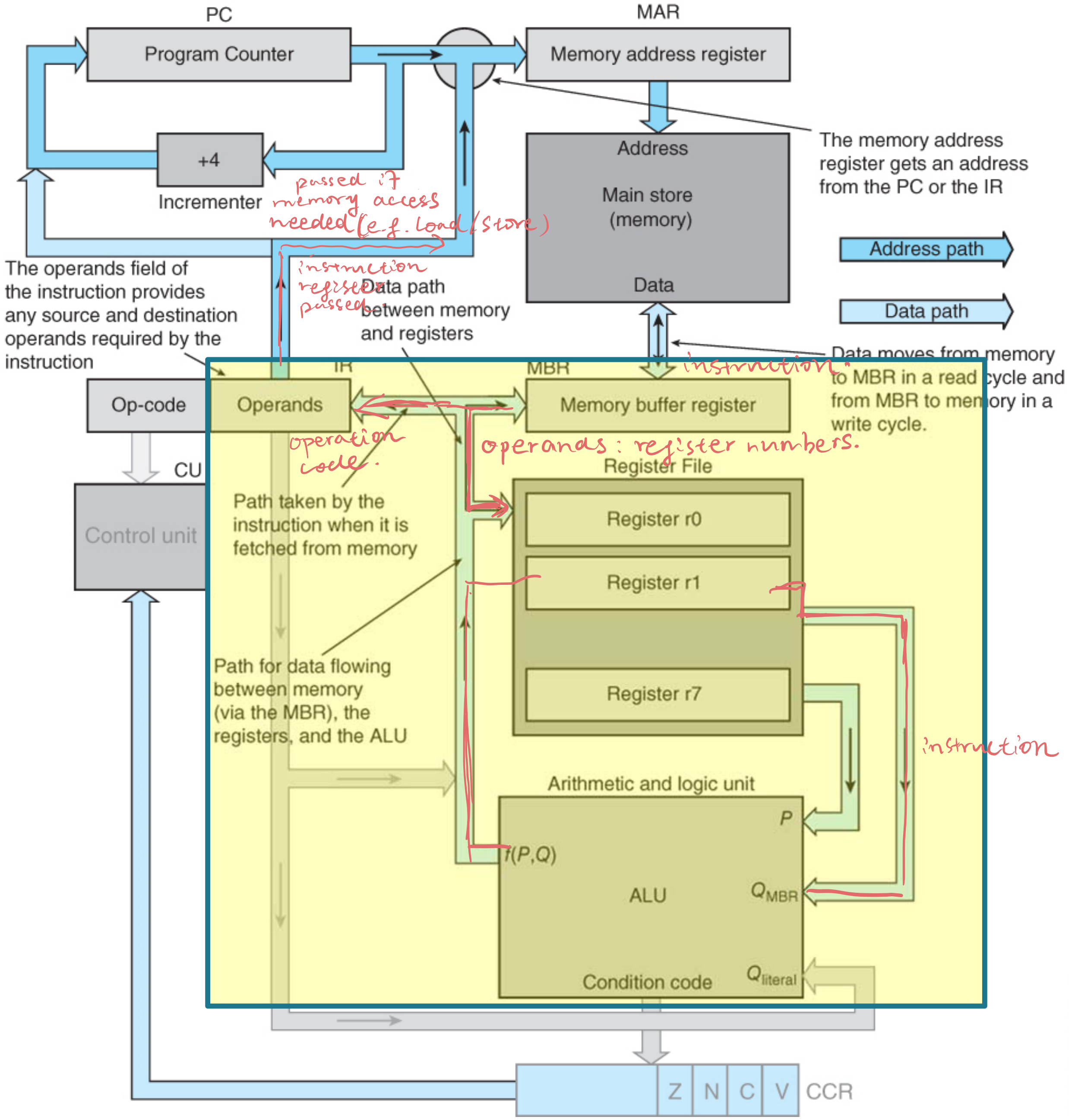
FIGURE 3.2 Partial structure of a hypothetical stored program machine


PC
Program Counter
MAR
Memory address register
+4
Incrementer
Address
Main store
(memory)
Data
The operands field of the instruction provides any source and destination operands required by the instruction
Data path between memory and registers
IR
MBR
Op-code
Operands
Memory buffer register
The memory address register gets an address from the PC or the IR
Address path
Data path
Data moves from memory to MBR in a read cycle and from MBR to memory in a write cycle.
CU
Control unit
Path taken by the instruction when it is fetched from memory
Register File
Register r0
Register r1
Register r7
Path for data flowing between memory (via the MBR), the registers, and the ALU
Arithmetic and logic unit
P
f(P,Q)
ALU
Q_MBR
Condition code
Q_literal
Z
N
C
V
CCR
© Cengage Learning 2014

- ❑ In the ***fetch phase***, the Program Counter, PC, supplies the address of the next instruction to be executed to the MAR to read this instruction ***and*** the PC is *incremented by the size of an instruction*.

- ❑ The instruction is read and loaded into the Memory Buffer Register, MBR, ***and*** then copied to the Instruction Register, IR, where the op-code is decoded.

FIGURE 3.2 Partial structure of a hypothetical stored program machine


PC
Program Counter
MAR
Memory address register
+4
Incrementer
Address
Main store
(memory)
Data
The memory address register gets an address from the PC or the IR
Address path
Data path
The operands field of the instruction provides any source and destination operands required by the instruction
Data path between memory and registers
Data moves from memory to MBR in a read cycle and from MBR to memory in a write cycle.
IR
Op-code
Operands
MBR
Memory buffer register
CU
Control unit
Path taken by the instruction when it is fetched from memory
Register File
Register r0
Register r1
Register r7
Path for data flowing between memory (via the MBR), the registers, and the ALU
Arithmetic and logic unit
P
f(P,Q)
ALU
Q_{MBR}
Condition code
$Q_{literal}$
Z N C V CCR
passed if memory access needed (e.g. load/store)
instruction register passed
instruction.
operation code.
operands: register numbers.
instruction

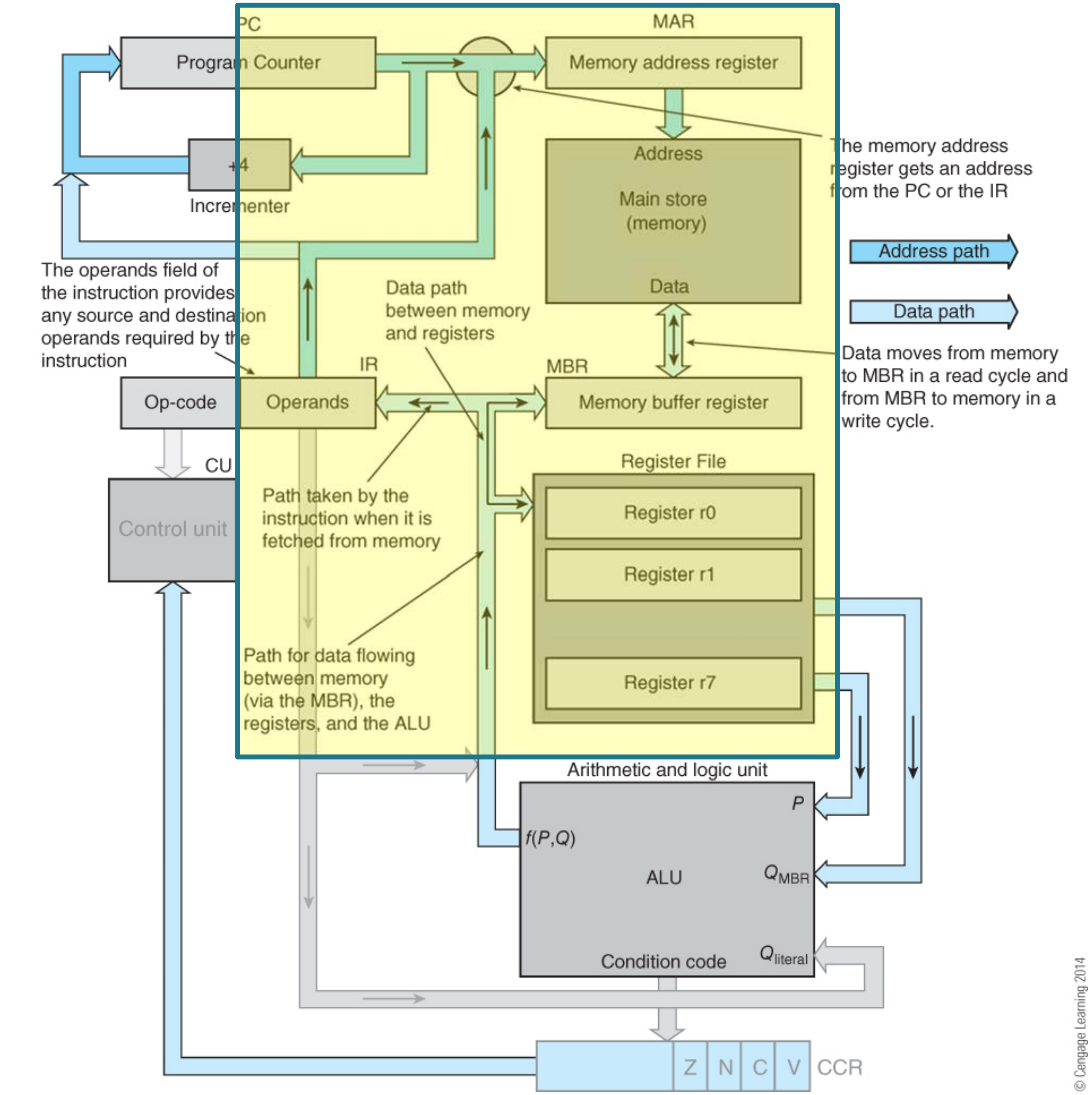
© Cengage Learning 2014

Structure of a Computer

- In the ***execute phase***, the operands may be ①read from the ***register file***, ②transferred to the ALU (*arithmetic and logic unit*) where they are operated on and ③then the result passed to the ***destination register***.
This is what we call, *register-to-register* operation

This slide is modified from the original slide by the author A. Clements and used with permission. New content added and copyrighted by © Mahmoud R. El-Sakka.

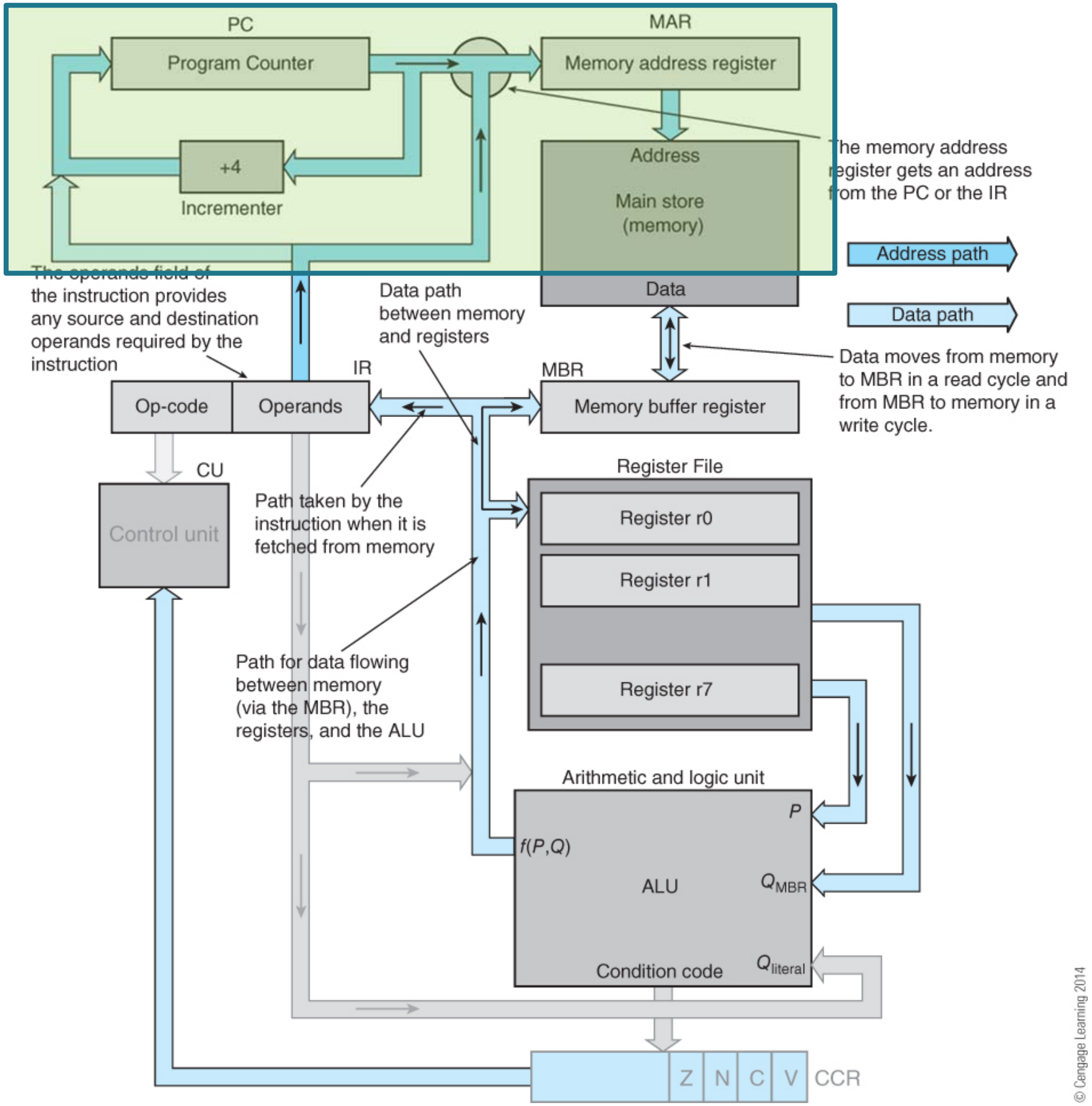
FIGURE 3.2 Partial structure of a hypothetical stored program machine


PC
Program Counter
MAR
Memory address register
+4
Incrementer
Address
Main store (memory)
Data
The memory address register gets an address from the PC or the IR
Address path
Data path
The operands field of the instruction provides any source and destination operands required by the instruction
Data path between memory and registers
IR
MBR
Op-code
Operands
Memory buffer register
Data moves from memory to MBR in a read cycle and from MBR to memory in a write cycle.
CU
Control unit
Register File
Register r0
Register r1
Register r7
Path taken by the instruction when it is fetched from memory
Path for data flowing between memory (via the MBR), the registers, and the ALU
Arithmetic and logic unit
P
f(P,Q)
ALU
Q_MBR
Condition code
Q_literal
Z
N
C
V
CCR
© Cengage Learning 2014

Structure of a Computer

- If the operation requires a memory access (e.g., a load or store), the memory address in the instruction register is sent to the MAR and a read or write operation performed.

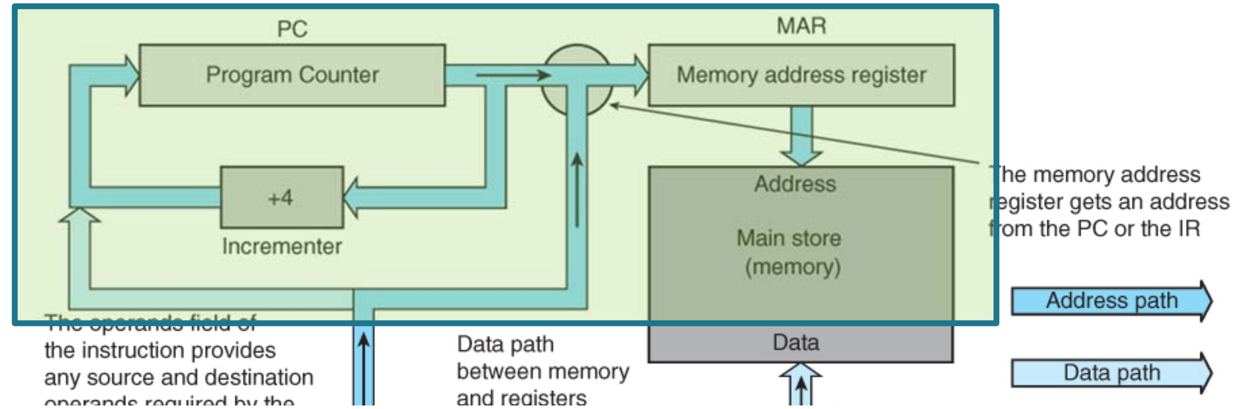
FIGURE 3.2 Partial structure of a hypothetical stored program machine



Structure of a Computer

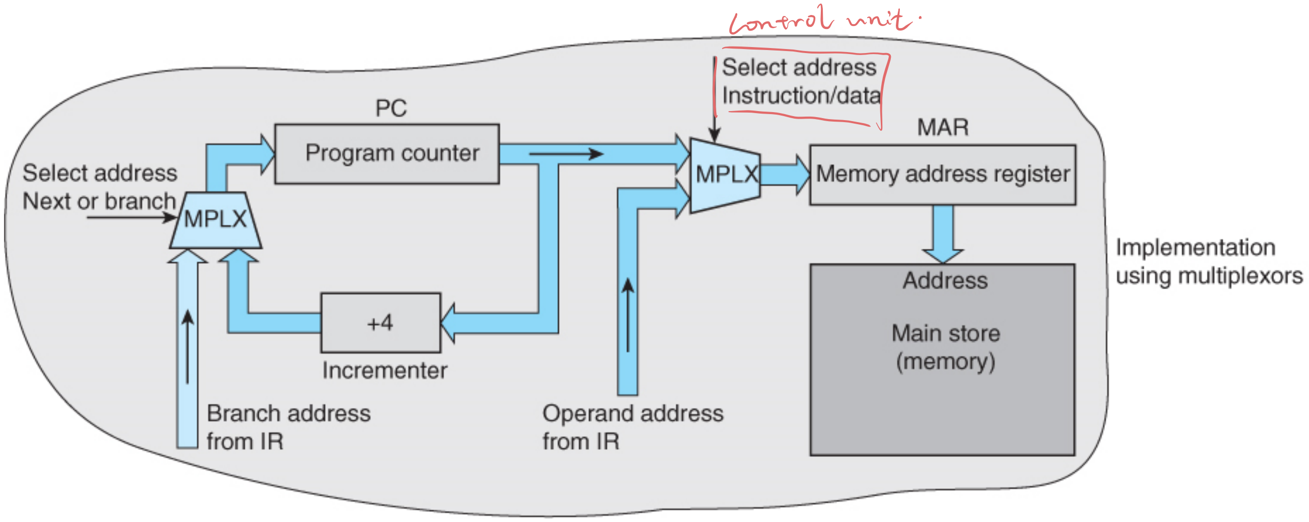
- ❑ But, how can we combine two input data lines together?

FIGURE 3.2 Partial structure of a hypothetical stored program machine


PC
Program Counter
MAR
Memory address register
+4
Incrementer
Address
Main store
(memory)
Data
The memory address register gets an address from the PC or the IR
Address path
Data path
The operands field of the instruction provides any source and destination operands required by the
Data path between memory and registers

Structure of a Computer

- ❑ But, how can we combine two input data lines together?



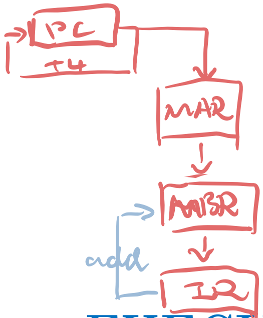
© Cengage Learning 2014

Structure of a Computer

Register Translation Language.

- ❑ Fetch/execute cycle in RTL.

Review Slide 26 in Chapter 1.

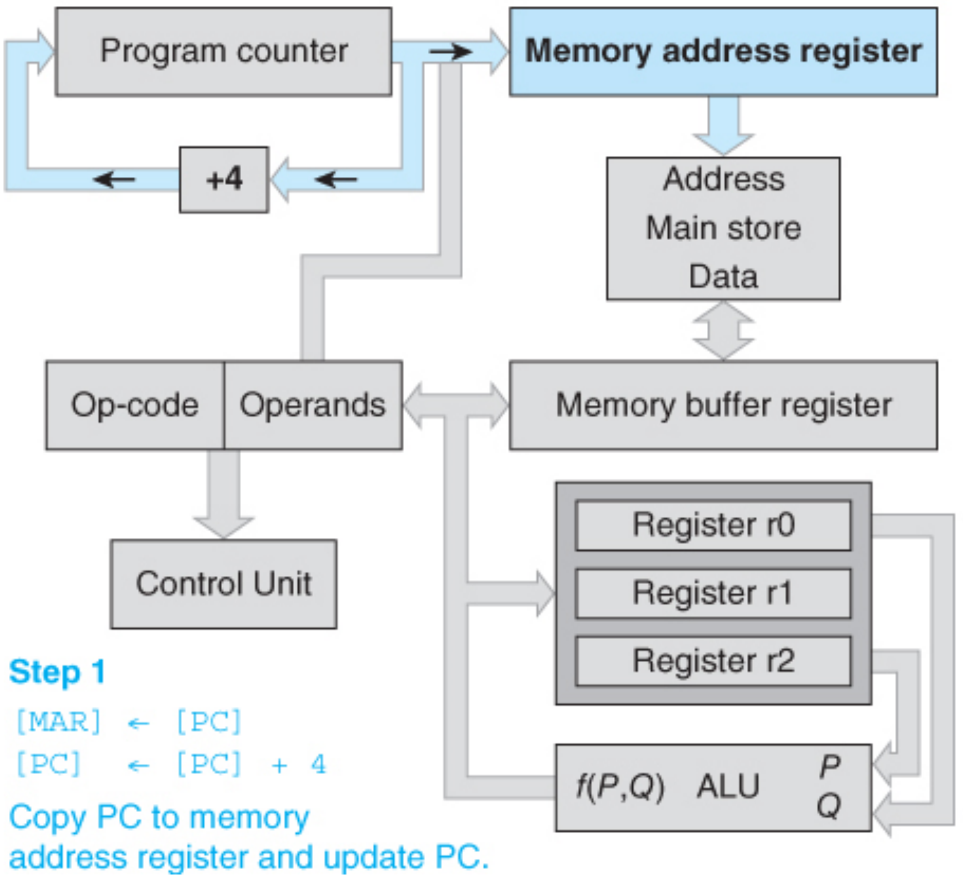


```
FETCH [MAR] ← [PC]           ; Step 1: copy PC to MAR
      [PC] ← [PC] + 4        ; Step 1: increment PC

      [MBR]  ← [[MAR]]       ; Step 2: read instruction pointed at by MAR
      [IR] ← [MBR]           ; Step 3: copy instruction in  MBR to  IR

EXECUTE
LDR   [MAR] ← [IR(address)]
                             ; Step 4: copy operand address from IR to MAR
      [MBR] ← [[MAR]]        ; Step 5: read operand value from memory
      [r1] ← [MBR]           ; Step 6: copy the operand to a register, e.g.,  r1
```

Review Slides 27 and 28 in Chapter 1.

The coming 3 slides show the above 6 steps graphically.

Fetching and Executing an Instruction



Program counter

Memory address register

+4

Address

Main store

Data

Op-code

Operands

Memory buffer register

Register r0

Register r1

Register r2

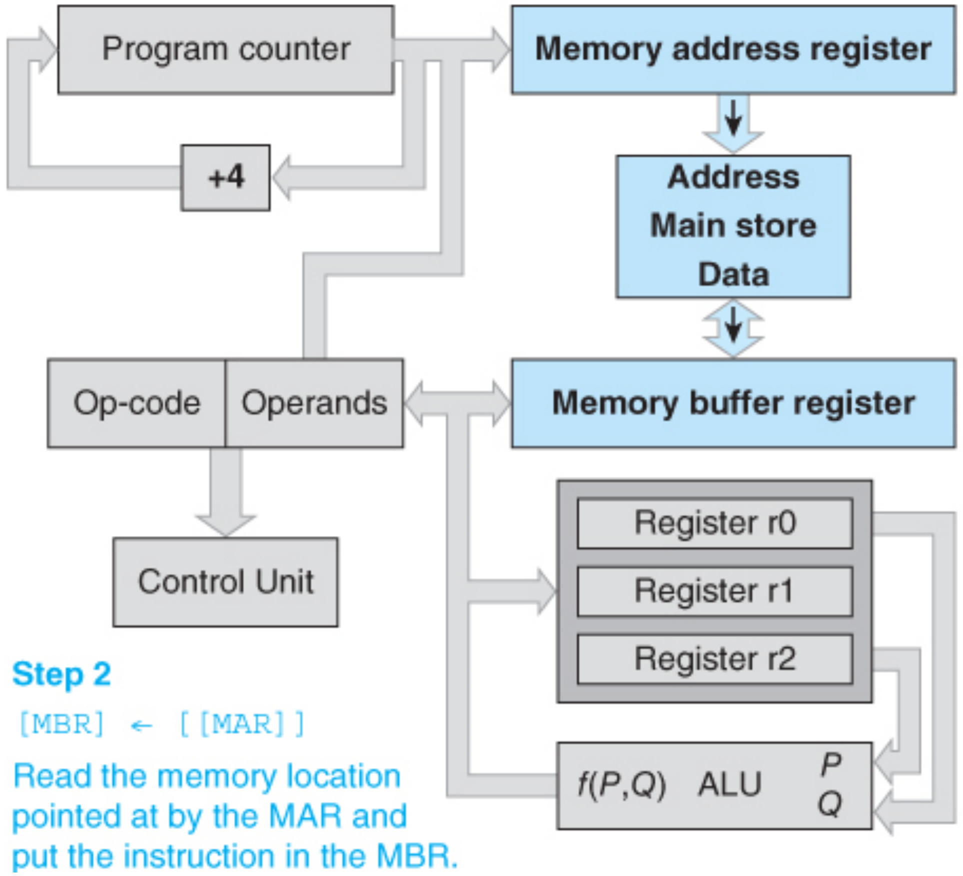
Control Unit

$f(P,Q)$ ALU P Q

Step 1

```
[MAR] ← [PC]
[PC]  ← [PC] + 4
```

Copy PC to memory address register and update PC.



Program counter

Memory address register

+4

Address

Main store

Data

Op-code

Operands

Memory buffer register

Register r0

Register r1

Register r2

Control Unit

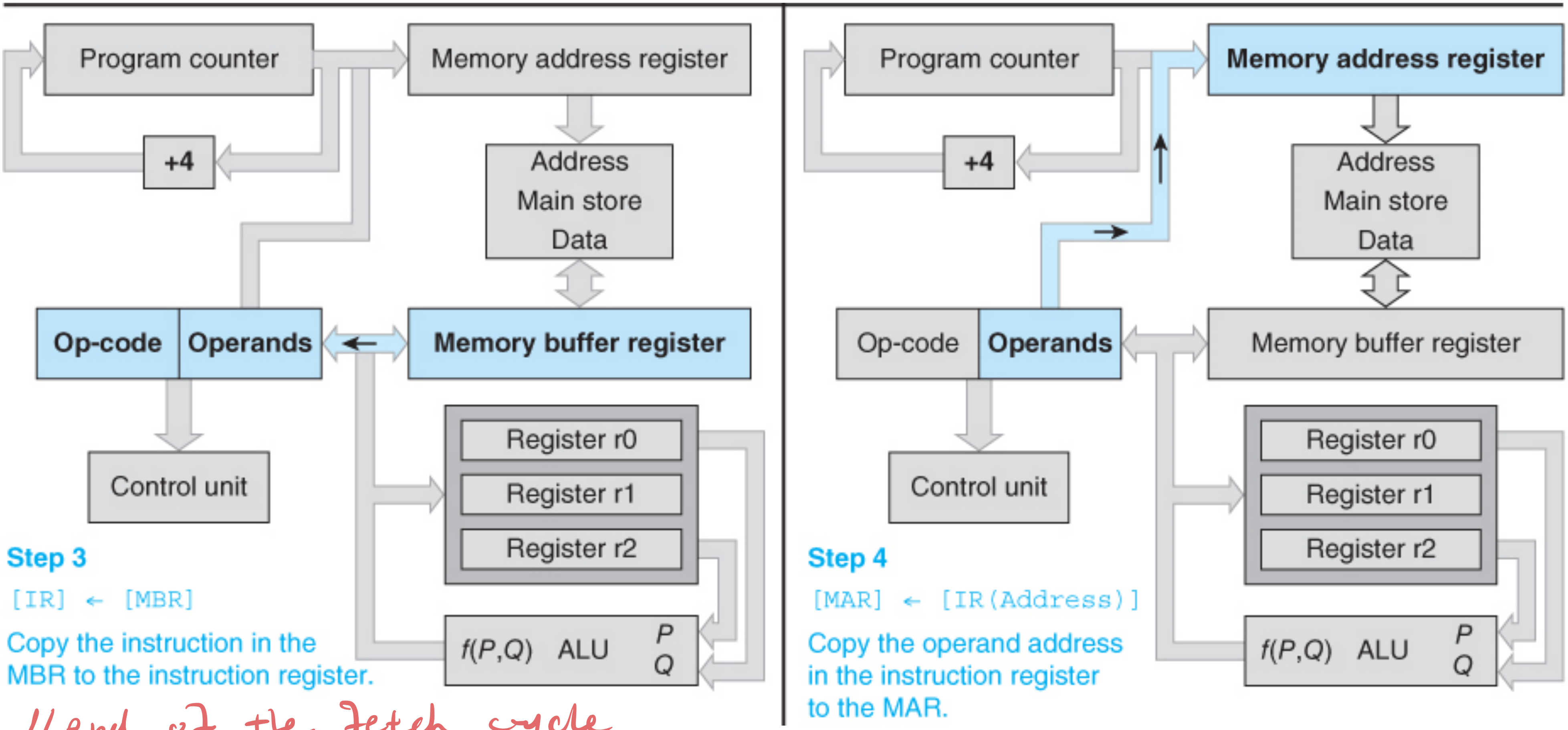
$f(P,Q)$ ALU P Q

Step 2

```
[MBR] ← [[MAR]]
```

Read the memory location pointed at by the MAR and put the instruction in the MBR.

Fetching and Executing an Instruction



Step 3

`[IR] ← [MBR]`

Copy the instruction in the MBR to the instruction register.

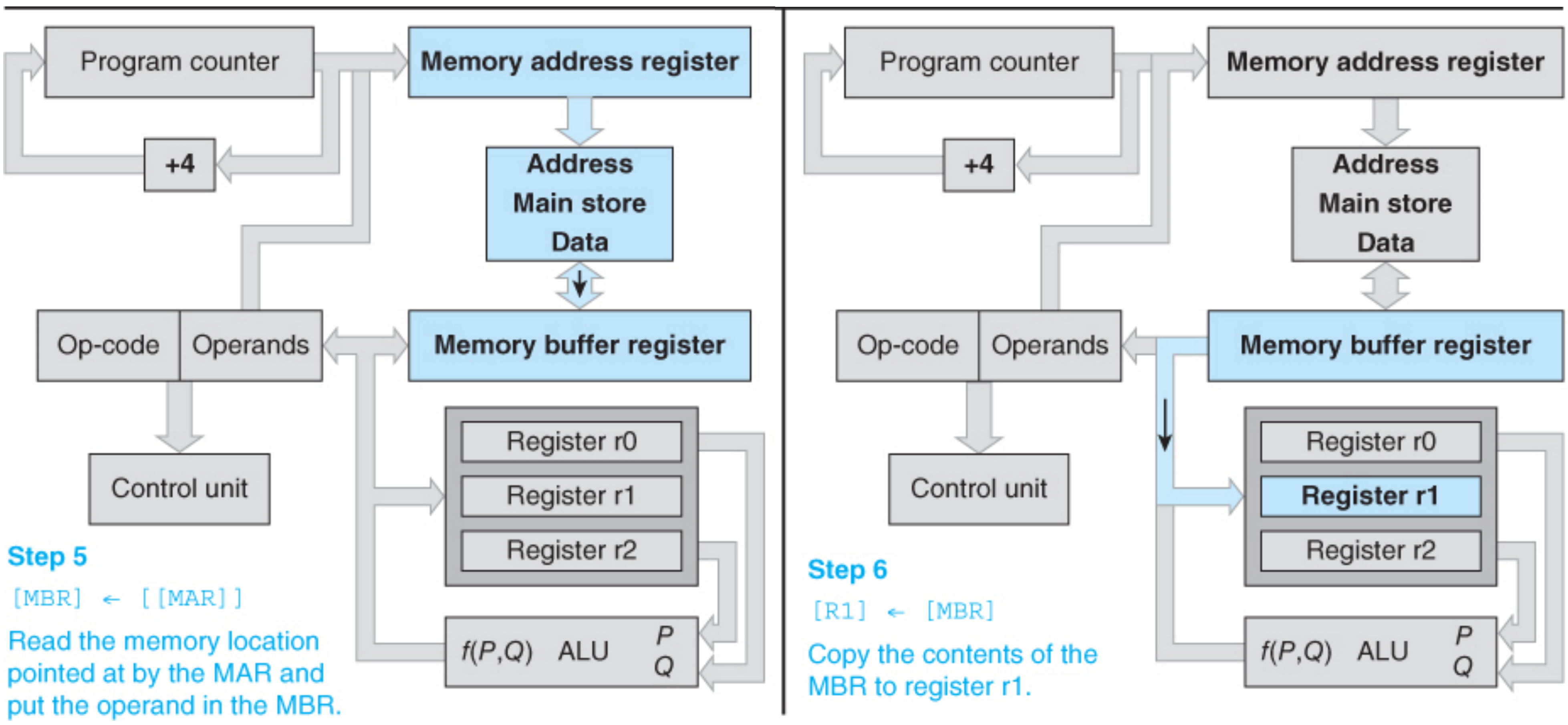
// end of the fetch cycle

Step 4

`[MAR] ← [IR(Address)]`

Copy the operand address in the instruction register to the MAR.

Fetching and Executing an Instruction



Step 5

[MBR] ← [[MAR]]

Read the memory location pointed at by the MAR and put the operand in the MBR.

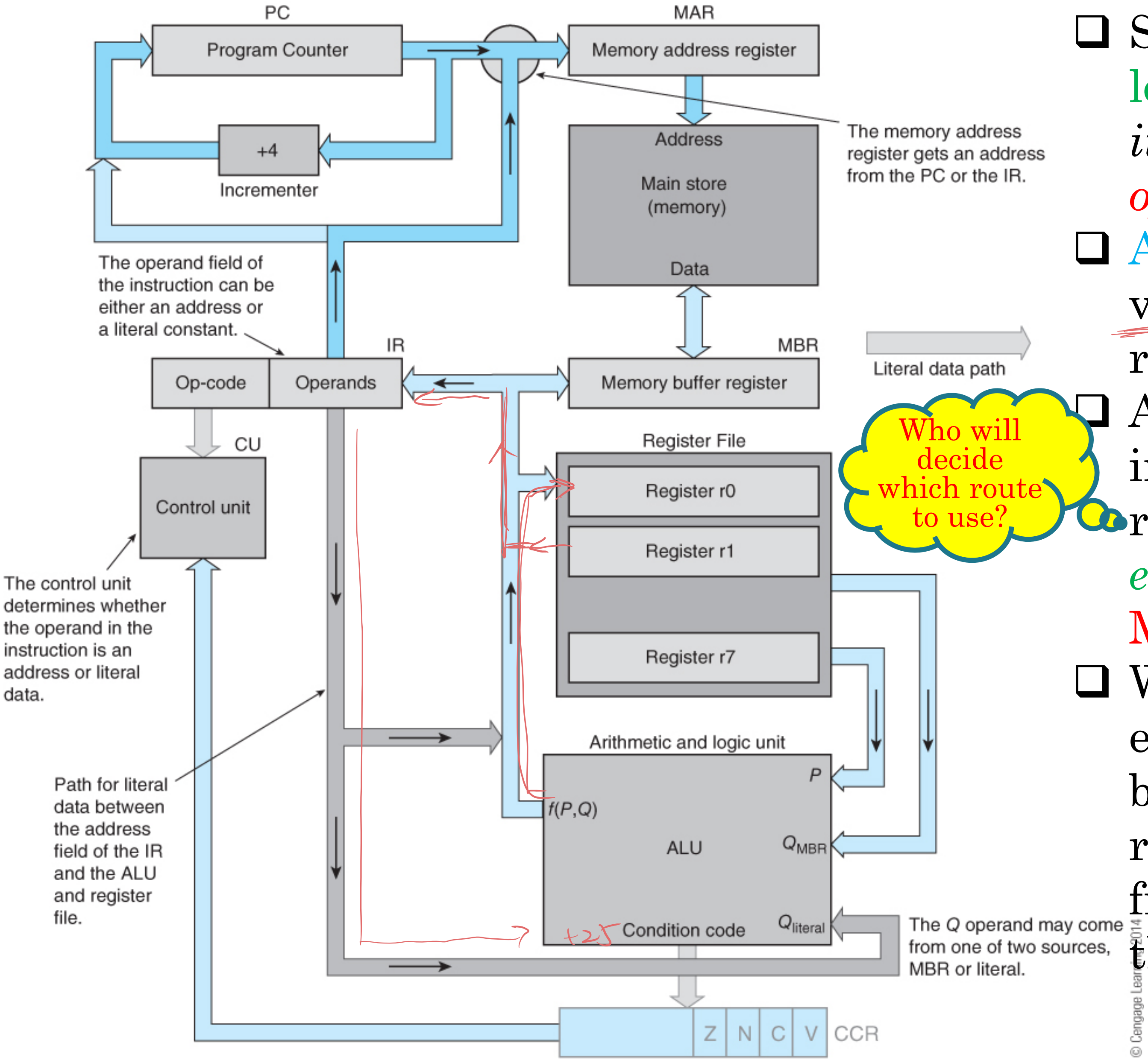
Step 6

[R1] ← [MBR]

Copy the contents of the MBR to register r1.

Dealing with Constants

FIGURE 3.4 Information paths for literal operands



The memory address register gets an address from the PC or the IR.

The operand field of the instruction can be either an address or a literal constant.

The control unit determines whether the operand in the instruction is an address or literal data.

Path for literal data between the address field of the IR and the ALU and register file.

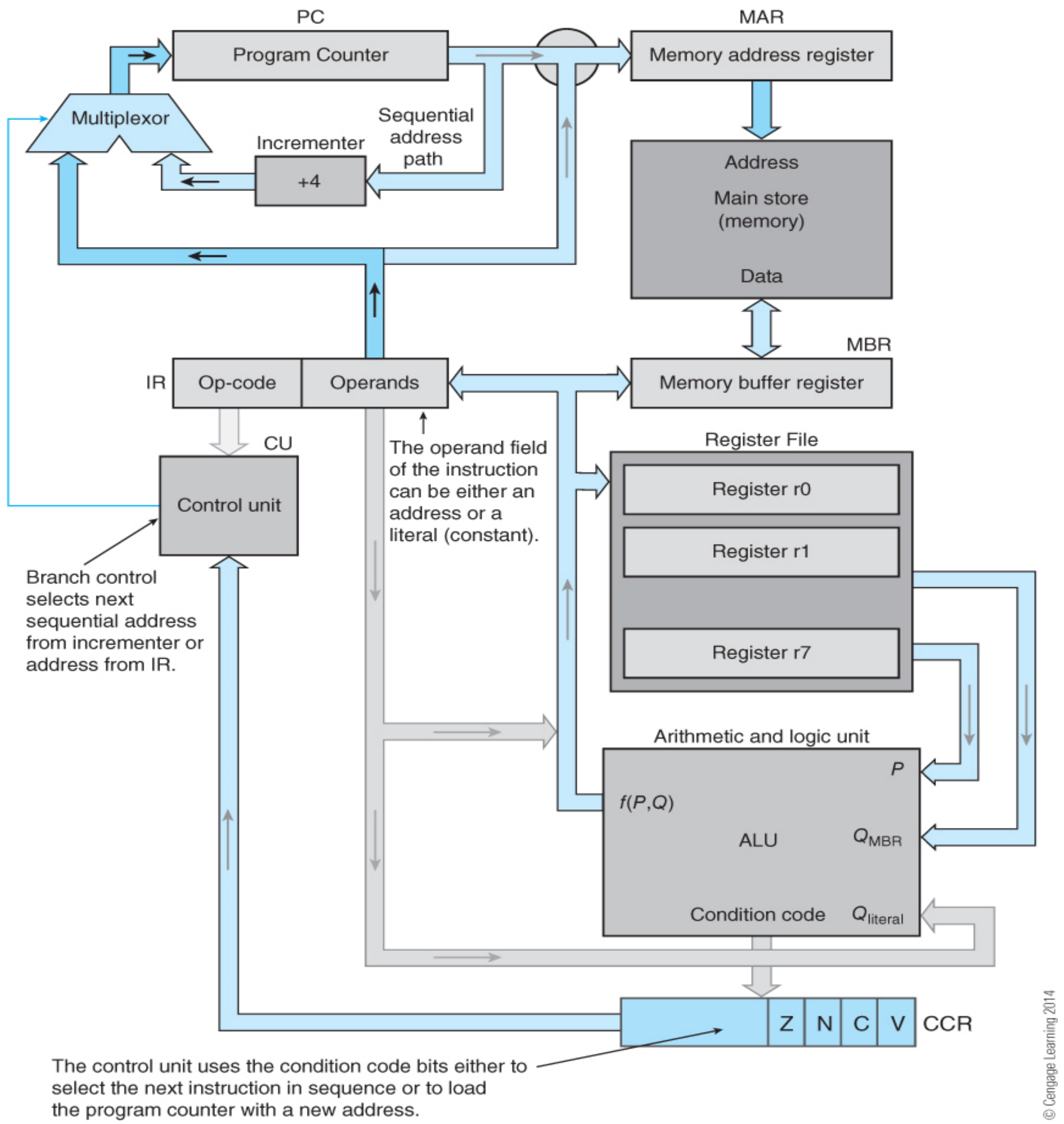
The Q operand may come from one of two sources, MBR or literal.

Who will decide which route to use?

- ❑ Suppose we want to load the *number 1234 itself* (*a.k.a. literal operand*) into register r1.
  whatever comes after # is literal.
- ❑ ADD **r0**,r1,#25 adds the value 25 to the content of r1 and puts the sum in r0
  constant data, not address.
- ❑ A path from the instruction register, IR, routes a literal operand to *either* the register file, MBR, and ALU
- ❑ When ADD **r0**,r1,#25 is executed, the operand to be added to r1, i.e., #25, is routed from the operand field of the IR, rather than from registers.

Flow Control

FIGURE 3.5 Implementing conditional behavior at the machine level



The operand field of the instruction can be either an address or a literal (constant).

Branch control selects next sequential address from incrementer or address from IR.

The control unit uses the condition code bits either to select the next instruction in sequence or to load the program counter with a new address.

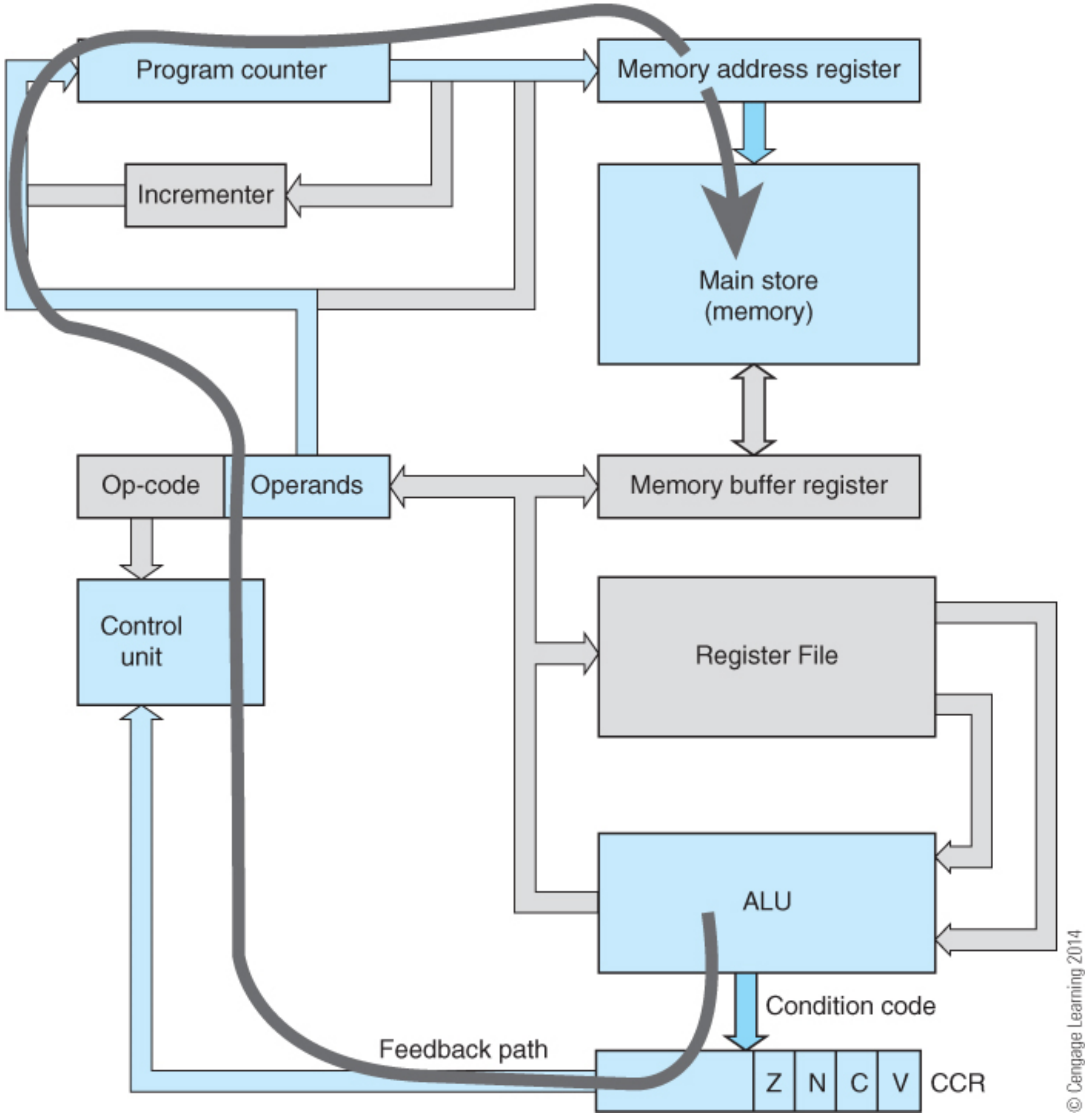
© Cengage Learning 2014

- ❑ *Flow control* refers to any action that *modifies* the normal instruction sequence.
- ❑ *Conditional behavior* allows a processor (*based on the values in the CCR register*) to select one of two possible courses of actions:
  - ○ Continuing executing the *next instruction* in sequence, or
  - ○ Loading the Program Counter with a new value and executing a *branch to another region* of code.

Flow Control

Figure 3.6 illustrate how the result from the ALU can be used to modify the sequence of instructions.

FIGURE 3.6 Feedback from ALU to instruction



Status Bits (Flags)

- When a computer performs an operation, it stores the *status* or *condition* information in the *Condition Code Register* (***CCR***).
- The processor records whether the result is (flag.)
  - **Z**ero (Z), result =0 Z=1; ≠1 Z=0
  - **N**egative in two's complement terms (N), result <0 N=1
  - generated a **C**arry (C), or (has carry out C=1)
  - generated an arithmetic o**V**erflow (V).

V=1 iff overflow occur.

This is the carry-out

Status Bits (Flags)

- ❑ Example (*assume that we are dealing with an 8-bit processor*):

00110011	11111111	01011100	11011100
+01000010	+00000001	+01000001	+11000001
01110101	100000000	10011101	C N 110011101
Z = 0 , N = 0	Z = 1 , N = 0	Z = 0 , N = 1	Z = 0 , N = 1
C = 0 , V = 0	C = 1 , V = 0	C = 0 , V = 1	C = 1 , V = 0
51	-1	92	-36
+66	+1	+65	-63
---	---	---	---
117	0	-99	-99

Carry out

CISC means COMPLEX Instruction Set Computer

- ❑ CISC processors, like the *Intel IA32*,
  - o automatically update status flags after each operation.

RISC means REDUCED Instruction Set Computer

- ❑ RISC processors, like the *ARM*,
  - o require the programmer to request updating the status flags.
- ❑ In *ARM* processors, *programmers need to request updating* the *status flags by appending an S to the instruction*;
  - ❑ for example, SUBS (instead of SUB) or ADDS (instead of ADD).