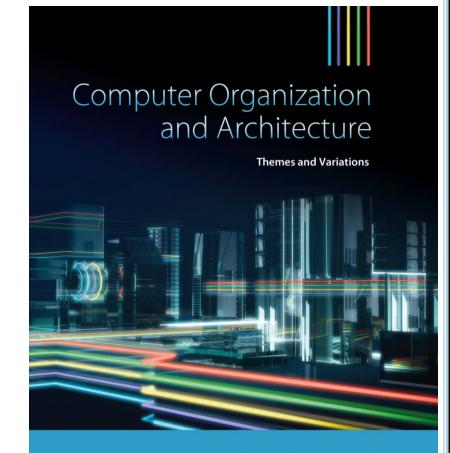
Part 1

CHAPTER 3

Architecture and Organization



Alan Clements

1

These slides are being provided with permission from the copyright for CS2208 use only. The slides must not be reproduced or provided to anyone outside of the class.

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



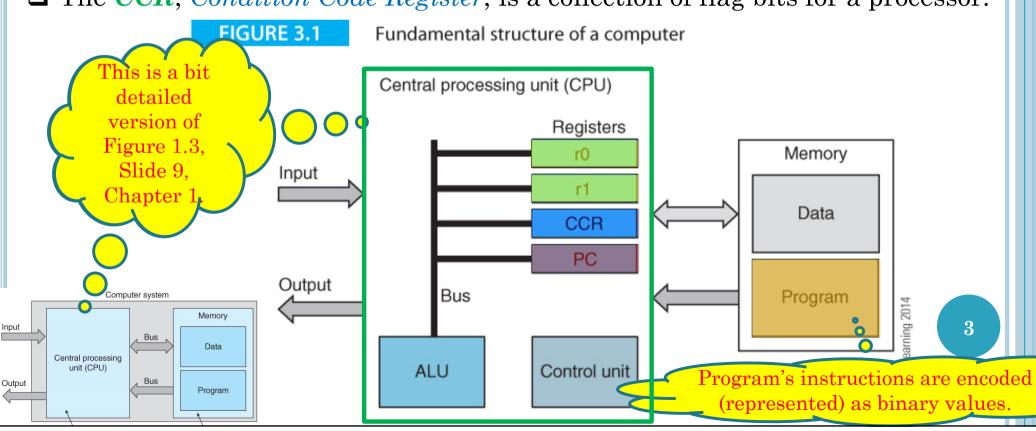
The Instruction Set Architecture

In this chapter, we will:

- Revisit the *stored program machine* and **show** how an instruction is executed
- ☐ Introduce instruction formats, including
 - o memory-to-register,
 - o register-to-memory, and
 - o 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.
- \square *Temporary data* is stored in *registers* such as r0 and r1.
- ☐ 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.



Instruction Formats

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

LDR registerDestination, memorySource

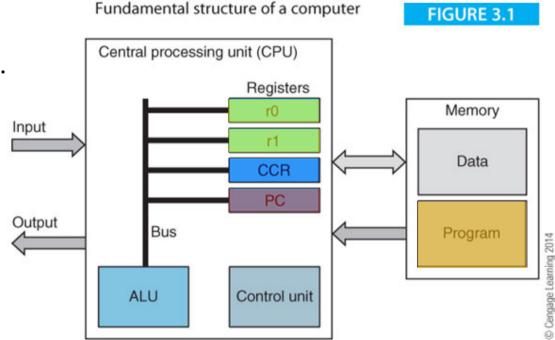
STR registerSource, memory Destination

Operation registerDestination,registerSource1,registerSource2

```
LDR r1, 1234
```

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.



- ☐ A stored program operates in a *fetch*/execute two-phase mode.
 - In the *fetch phase* the next instruction is read from memory and decoded.
 - o In the *execute phase* the instruction is interpreted and executed by the CPU's logic.

Review Slides 28 and 29 in Chapter 1.

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

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, ..., ri 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.
- 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.

FIGURE 3.2 Partial structure of a hypothetical stored program machine Program Counter Memory address register The memory address Address register gets an address from the PC or the IR Main store Incrementer (memory) Address path The operands field of Data Data path the instruction provides between memory Data path any source and destination and registers operands required by the Data moves from memory instruction **MBR** to MBR in a read cycle and from MBR to memory in a Op-code Operands Memory buffer register write cycle. Register File CU Path taken by the Register r0 instruction when it is fetched from memory Register r1 Path for data flowing between memory Register r7 (via the MBR), the registers, and the ALU Arithmetic and logic unit f(P,Q) Q_{MBR} ALU Q_{literal} Condition code CV ZINI

Structure of a Computer

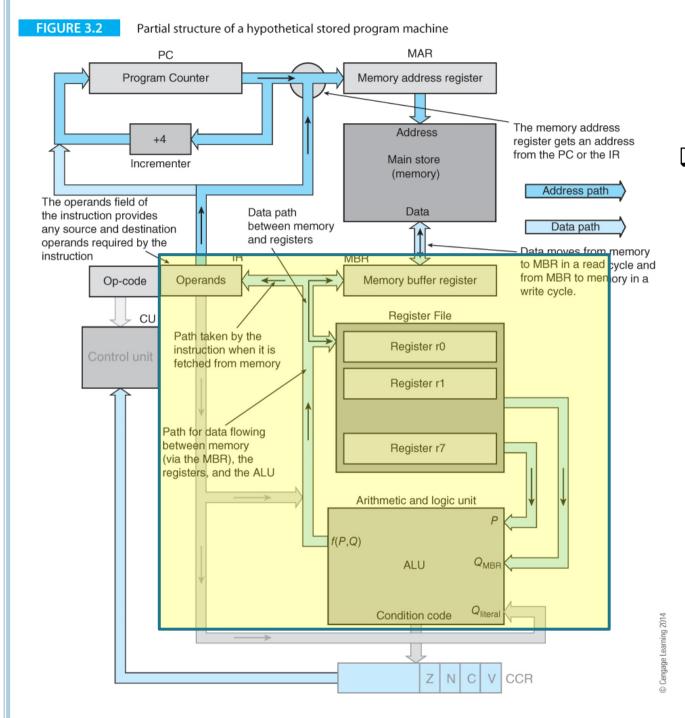
- We are going to use an ARM processor to introduce assembly language and a modern ISA.
- However, it would be better to begin with the description of a very simple
 hypothetical computer to keep things simple.

FIGURE 3.2 Partial structure of a hypothetical stored program machine PC **Program Counter** Memory address register The memory address Address register gets an address from the PC or the IR Main store Incrementer (memory) Address path The operands field of Data Data path the instruction provides between memory Data path any source and destination and registers operands required by the Data moves from memory instruction **MBR** to MBR in a read cycle and from MBR to memory in a Op-code Operands Memory buffer register write cycle. CU Register File Path taken by the Register r0 instruction when it is fetched from memory Register r1 Path for data flowing between memory Register r7 (via the MBR), the registers, and the ALU Arithmetic and logic unit f(P,Q) Q_{MBP} ALU Q_{literal} Condition code ZNCV

Structure of a Computer

- □ 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, <u>and</u> then copied to the Instruction Register, 8 IR, where the

This slide is modified from the original slide by the author A. Clements and used with permission. New content added and coding the by the author A. Clements and used with permission. New content added and coding the by the author A. Clements and used with permission.

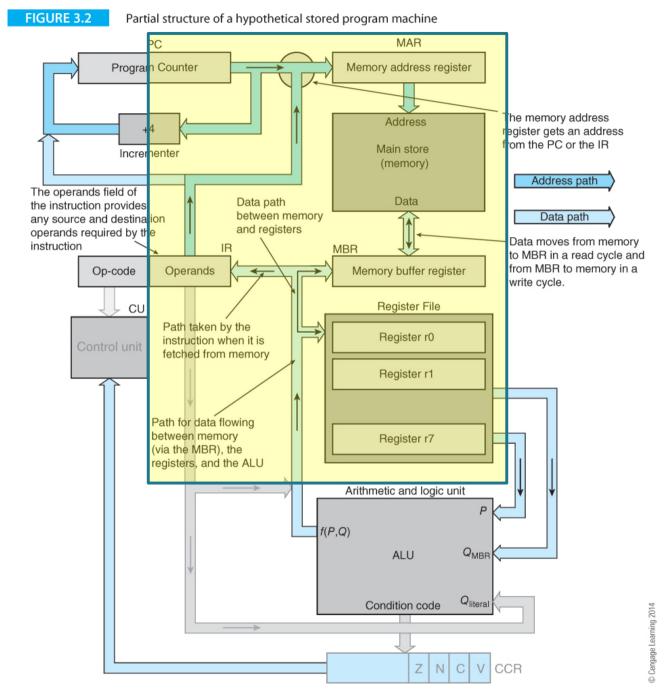


☐ In the *execute phase*, the operands may be read from the *register file*, transferred to the ALU (*arithmetic and logic unit*) where they are operated on and then the result passed to the *destination register*.

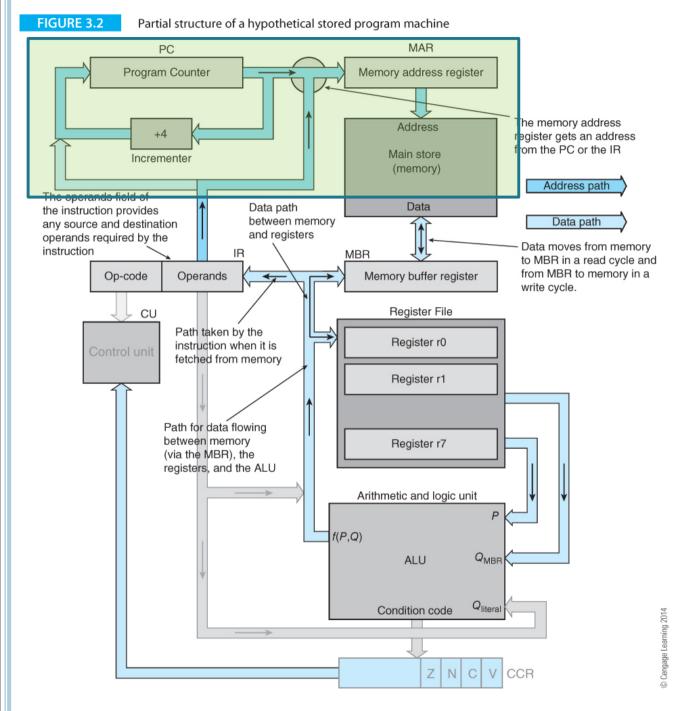
This is what we call,

register-to-register

operation.



☐ 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.



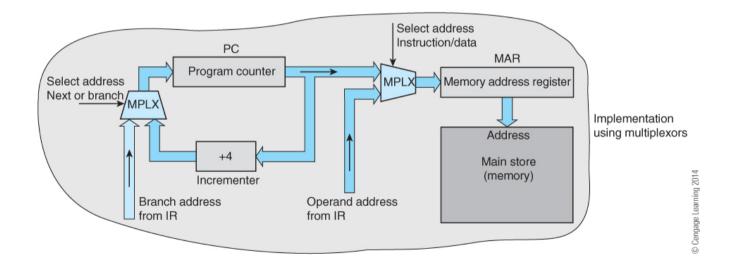
☐ But how can we combine two input data lines together?

FIGURE 3.2 Partial structure of a hypothetical stored program machine MAR PC Memory address register **Program Counter** he memory address Address egister gets an address om the PC or the IR Main store Incrementer (memory) Address path Data Data path the instruction provides between memory 1 Data path any source and destination and registers

anarande required by the

Structure of a Computer

But how can we combine two input data lines together?



□ Fetch/execute cycle in RTL.

Review Slide 26 in Chapter 1.

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

```
[MBR] ← [[MAR]]; Step 2: read instruction pointed at by MAR
```

 $[IR] \leftarrow [MBR]$; Step 3: copy instruction in MBR to IR

```
EXECUTE
```

```
LDR [MAR] \leftarrow [IR(address)]
```

; Step 4: copy operand address from IR to MAR

```
[MBR] \leftarrow [[MAR]]; Step 5: read operand value from memory
```

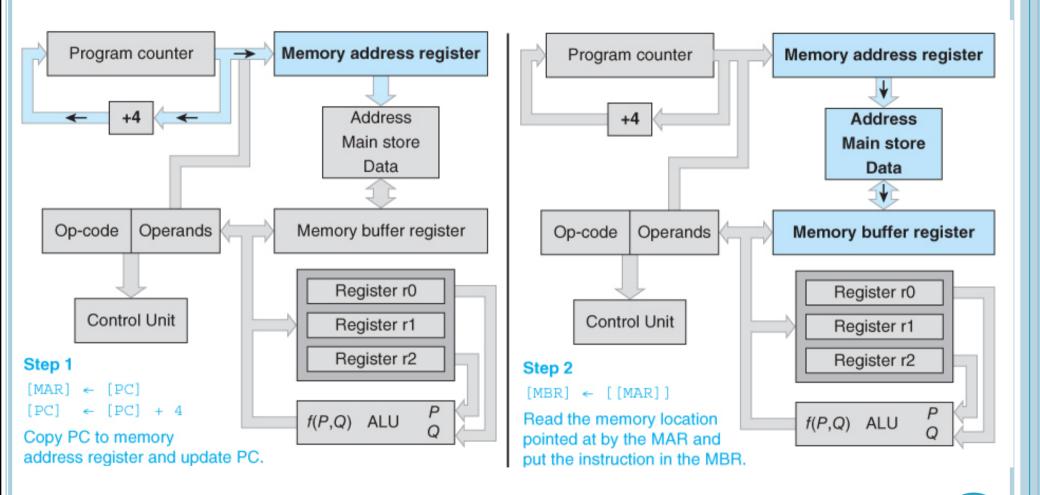
 $[r1] \leftarrow [MBR]$; Step 6: copy the operand to a register, e.g., r1

Review Slides 28 and 29 in Chapter 1.

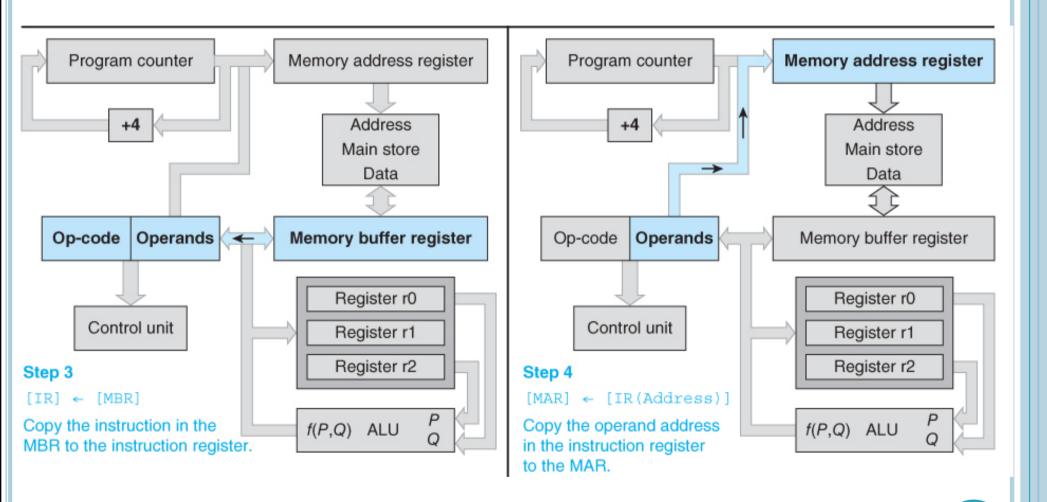
13

The coming 3 slides show the above steps graphically. —

Fetching and Executing an Instruction



Fetching and Executing an Instruction



Fetching and Executing an Instruction

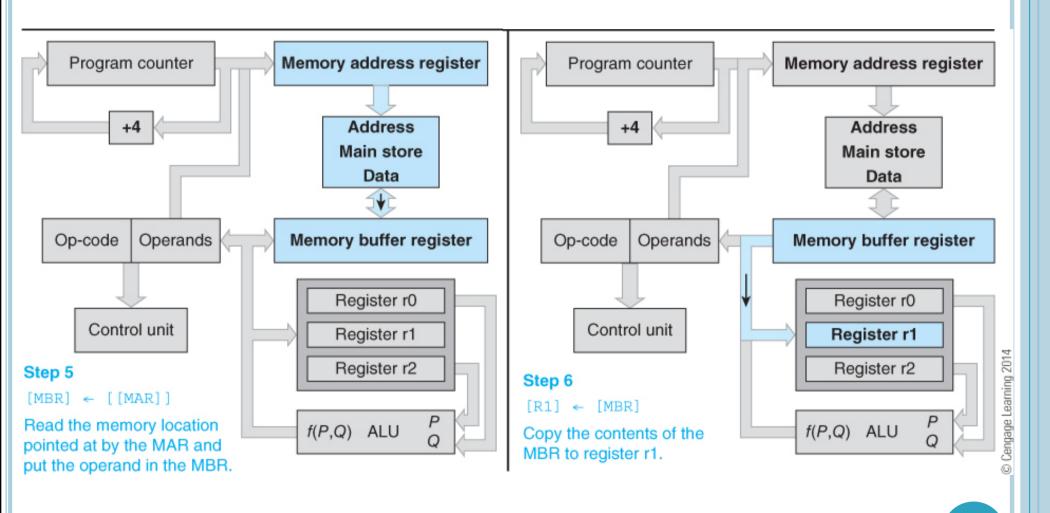


FIGURE 3.4 Information paths for literal operands PC MAR Program Counter Memory address register The memory address Address register gets an address from the PC or the IR. Main store Incrementer (memory) The operand field of Data the instruction can be either an address or a literal constant. MBR Literal data path Op-code Operands Memory buffer register Who will Register File CU decide which route Register r0 Control unit to use? Register r1 The control unit determines whether the operand in the instruction is an Register r7 address or literal data. Arithmetic and logic unit Path for literal data between f(P,Q)the address Q_{MBB} ALU field of the IR and the ALU and register file. Q_{literal} The Q operand may come Condition code from one of two sources, MBR or literal.

Dealing with Constants

- □ Suppose we want to load the *number 1234* itself (a.k.a. literal operand) into register r1.
- ADD r0,r1,#25 adds the value 25 to the content of r1 and puts the sum in r0 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.

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 3.5 Implementing conditional behavior at the machine level PC MAR **Program Counter** Memory address register Sequential Multiplexor address Incrementer Address Main store (memory) Data Op-code Operands Memory buffer register Register File CU The operand field of the instruction Register r0 can be either an Control unit address or a literal (constant). Register r1 Branch control selects next sequential address from incrementer or Register r7 address from IR. Arithmetic and logic unit f(P,Q) Q_{MBR} ALU Condition code Q_{literal} ZNC CCR 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.

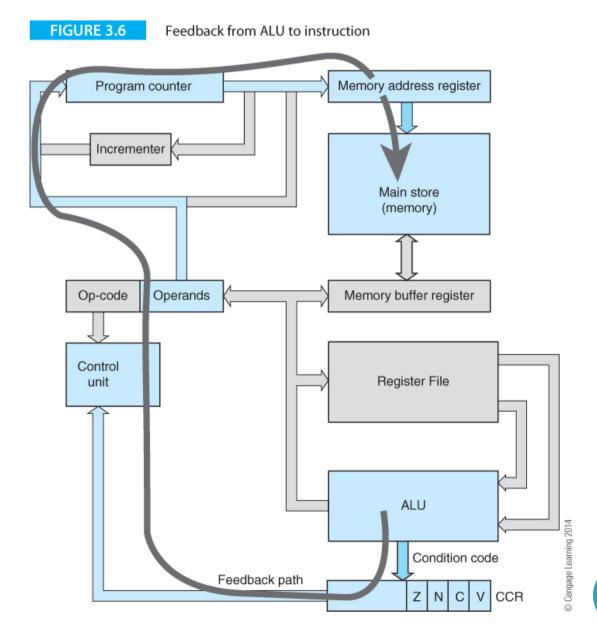
Flow Control

- ☐ *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
 - Counter with a new value and executing a branch to anothe region of code.

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.

Flow Control

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



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
 - o Zero (Z),
 - Negative in two's complement terms (N),
 - o generated a Carry (C), or • •
 - o generated an arithmetic oVerflow (V).

This is the carry-out

Status Bits (Flags)

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

00110011	11111111	01011100	11011100
+01000010	+0000001	+01000001	+11000001
01110101	10000000	10011101	$\overline{110011101}$
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

CISC means COMPLEX Instruction Set Computer

-99

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

RISC means REDUCED Instruction Set Computer

 \square RISC processors, like the *ARM*,

117

- 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).

-99