Chapter 04: Instruction Sets and the Processor organizations

Lesson 08:

Processor Instructions - Part 1

Objective

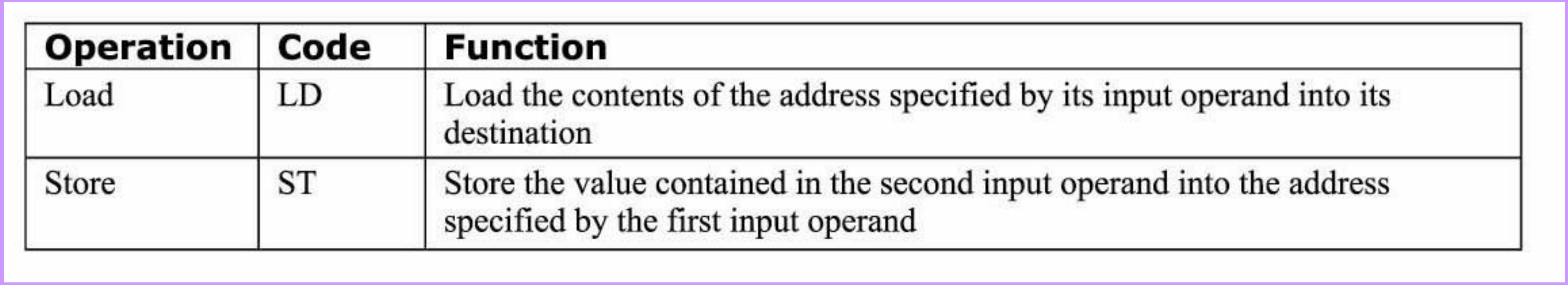
- Understand the load, store instructions and register transfer instructions
- Understand program flow and
- Learn branch Instructions and labels

Load and Store

Load LD and Store ST

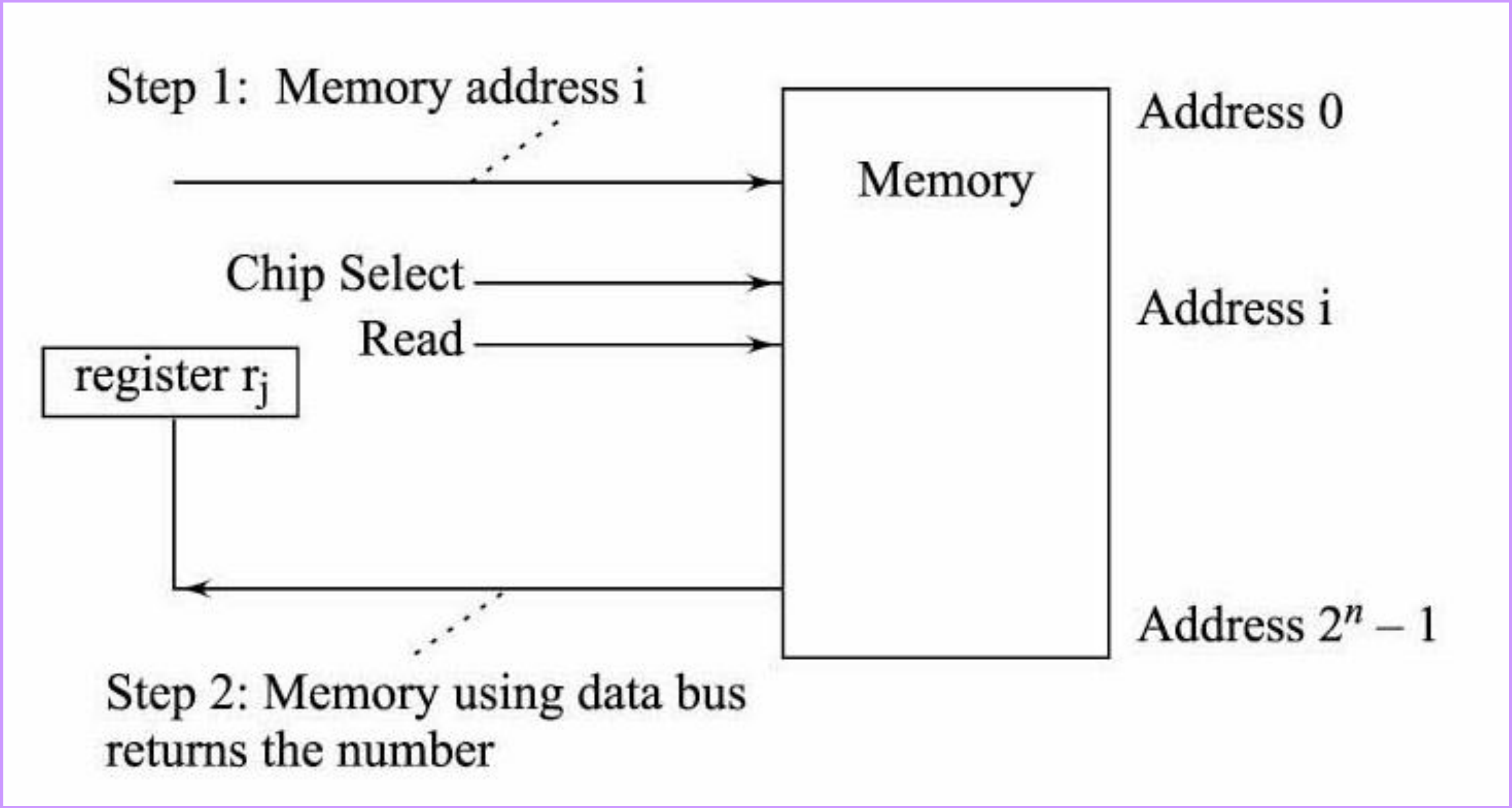
- Load the contents of the address specified by its input operand into its destination
- Store the value contained in the second input operand into the address specified by the first input operand
- All processors have CISC or RISC

LD and ST

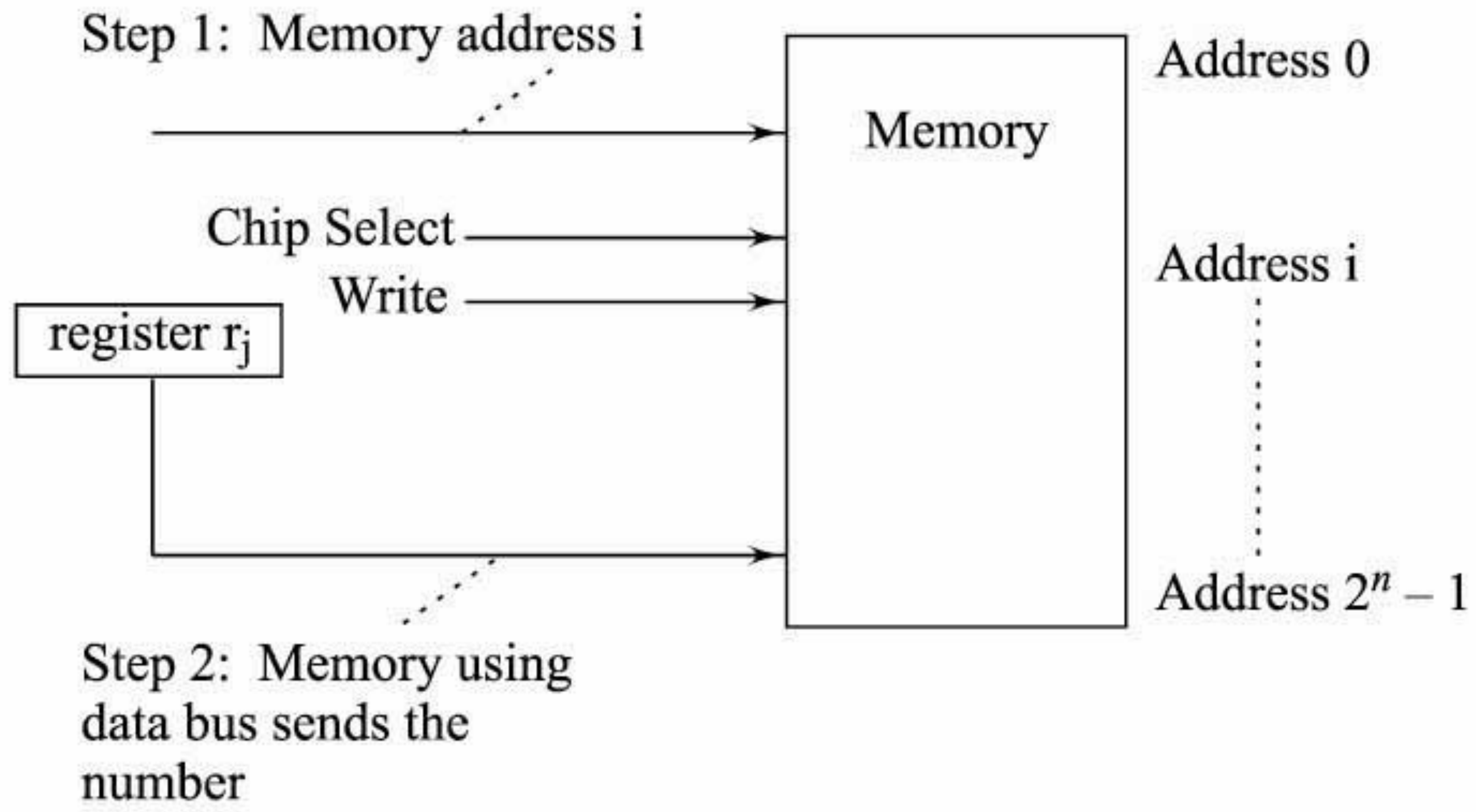


Operation	Code	Function
Load	LD	Load the contents of the address specified by its input operand into its destination
Store	ST	Store the value contained in the second input operand into the address specified by the first input operand

LD rj, M(i) Load operation



ST M(i), rj Store instruction



Register Transfer Instructions in a GPR based Processor

Instructions in a GPR organization

- Specify the registers that hold their input operands and the register where their result will be written
- Instructions encoding varies from processor to processor
- Some processors use the rightmost operand as their destination register, with other operands being the inputs to the instruction

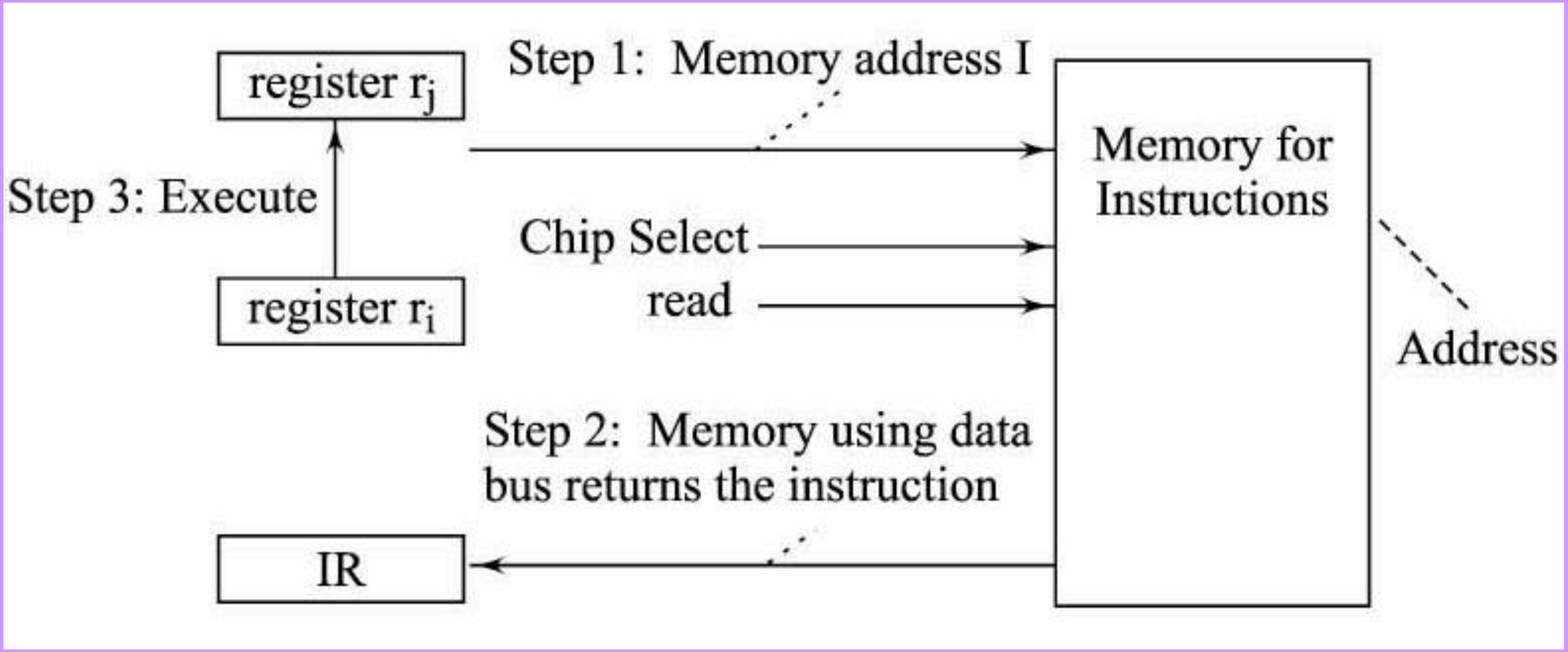
Instructions in a GPR organization

- Some processors use the encoding with one other 2 or 3 operands in the instructions though requiring 3 operands in the arithmetic or logic instruction

Register Transfer Instructions MOV

- Move the input operand at a register to the second by a word copy

MOV rj, ri


Step 1: Memory address I
register r_j
Step 3: Execute
register r_i
Chip Select
read
Memory for Instructions
Address
Step 2: Memory using data bus returns the instruction
IR

Example— a GPR set program to compute the function 2 + (7 × 3)

- Assume— The architecture has 16 registers, r0-r15
- Assume— All registers contain 0 at the start of the program
- The result of the computation may be left in any register

Solution

```
MOV rl, #7
MOV r2, #3
MUL r3, rl, r2
MOV r4, #2
ADD r4, r3, r4
```

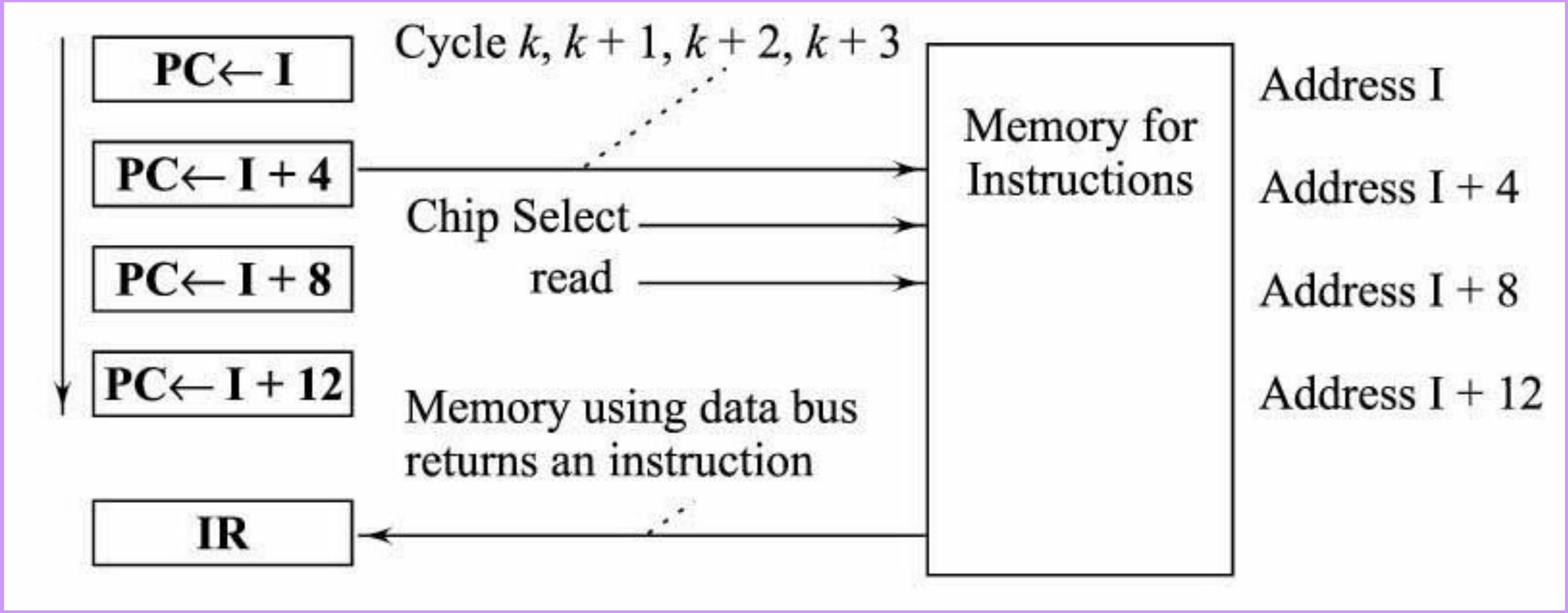
Instruction Execution and straight-line Instruction sequencing

Example

Consider the following sequence of instructions at StepK of a program:

stepK:
(instruction k) at address I
(instruction k + 1) at address I + 4
(instruction k + 2) at address I + 8

Sequences of PC during 4 instruction cycles with no branch or call



Branching Instructions

Branching Instructions

- Control operations, which are often called branches, divided into two categories:

1. unconditional and
2. conditional

Control operations

- Many processors provide different *addressing modes* for branch instructions that cause the branch instruction to perform a computation and then set the PC to result of the computation instead of just setting the PC to the value of its input

Unconditional branch — jump, NOP

- Jump— Set the PC equal to their input operand
- NOP— processor skips the instruction and simply jumps to next address for the instruction

Using labels instead of numeric addresses

Destination addresses Specification

- Instead, instructions are labeled with text values, and branch instructions refer to these values

Destination addresses Specification for branching

- loop_start1: (instruction)

```
loop_start1: (instruction)
             (instruction)
             (instruction)
.
.
             BR loop_start1;
```

-

Destination addresses Specification

count = *n*

- loop_start2: (instruction)

  (instruction— decrease count)

  (instruction — check count *n* =0)

- BNE loop_start2;

Example

- Consider the following sequence of instructions at StepK of a program:
-
- stepK:
- (instruction k)
- (instruction k + 1)
- (instruction k + 2)

Example

BR StepL;

- .
- .
- StepL: (instruction l)
- .
-
- PC gets the address of StepL after the branch instruction.

Destination addresses Specification by labels

- When programmers write assembly-language programs or compilers generate them from high-level language programs, they *generally* do not specify the, destination addresses (the values of its input operand of branch) as the address of the destination instruction in memory

Programmer/ Compiler Labels Use

- Labels specify the destination of each branch, and the assembler calculates the address of each label when the program is assembled

Using labels instead of numeric addresses First advantage

- Label is much easier for the programmer to understand
- Looking at the code label Loop_Start, it is clear where the target of the branch instruction is, even if you a programmer does not know anything about the architecture of the processor

Numeric addresses Disadvantage

- If the target were specified by numeric address, programmer would have to know how much space each instruction tab up to be able to figure out the destination of each branch, and even that would takes some work

Using labels instead of numeric addresses Second Advantage

- The address corresponding to a label can change if the instructions before the label change. If numeric addresses were used instead of labels, the destination of each branch would have to be changed every time the number of instructions before the branch change

Conditional Branching Instructions

Conditional branches

- Branch on Equality BEQ ─ Test its two integer operands whether they are equal by a hypothetical subtraction and if yes, then set the PC (IP) equal to the value of its input operand

Conditional branches

- Branch on No Equality BNE— Test its two integer operands whether they are not equal by a hypothetical subtraction and if not, then set the PC (IP) equal to the value of its input operand

Example

- instruction I – 1: c = count + 1
- loop_startK:
- instruction I: decrease c
- .
- .instruction I + n – 1:
- instruction I + n: compare instruction not equal c and 0)
- BNE loop_startK; [Branch to loop_startK if not equal.]

EXAMPLE of a combined instruction DJNZ

- instruction I – 1: c = count;
- loop_startY: instruction I:
- .
- .
- instruction I + n – 1:
- instruction I + n: DJNZ c, loop_startY; [decrease c, compare with 0, if not zero then branch (jump) to loop_startY]

Conditional branches

- Branch on Greater BGT— Test its two integer operands or whether first is greater by a hypothetical subtraction and if yes, then set the PC (IP) equal to the value of its input operand

Conditional branches

- Branch on Lesser BLT─ Test its two integer operands or whether first is lesser by a hypothetical subtraction then and if yes, then Set the PC (IP) equal to the value of its input operand

Conditional branches

- Branch on Greater or Equality BGE— Test its two integer operands whether first is greater or they are equal by a hypothetical subtraction and if yes, then set the PC (IP) equal to the value of its input operand

Conditional branches

- Branch on Lesser or Equality BLE— Test its two integer operands whether first is lesser or they are equal by a hypothetical subtraction and if yes, then set the PC (IP) equal to the value of its input operand

Conditional branches

- Branch on Test BTST— Test its two operands whether they are equal by a hypothetical logical AND and if yes, then set the PC (IP) equal to the value of its input operand

Summary

We learnt

- Instructions for Load, Store, MOV, Linear programmed flow in sequence without branch
- Unconditional Branch
- Advantages of using the labels in the program before using the machine codes
- Conditional Branch

End of Lesson 08 on

Processor Instructions - Part 1