CIS-350
INFRASTRUCTURE TECHNOLOGIES

Supplement to Chapter 7 (The CPU and Memory)

Translation of Relative Addresses to Physical Addresses

Because programs are built of small modules (functions) using local variables, they execute within a relatively small area of memory – this makes small address field (operand) in the instruction word adequate. Then how shall one modify the instruction's operand to reach a large memory address?

Example

The LMC program below loads the contents of memory location 90 to the accumulator. As a result, the accumulator contains 10. Then the program adds the contents of memory location 91 to the current contents of the accumulator. As a result, the accumulator contains 25. Finally, the program stores the contents of the accumulator in memory location 92. As a result, memory location 92 contains 25.

```
00      LDA 90
01      ADD 91                  program area
02      STO 92
--------------------
--------------------
90      10
91      15                      data area
92      25
```

The addresses for the LMC program instructions above are chosen by you arbitrarily; they start at address 00 and are incremented by 1. Also, the program instructions use memory addresses 90 through 92 for the operands, chosen by you arbitrarily, for memory locations in which the data are stored. (Any other memory locations could have been used.) In other words, program instructions that one writes in assembly language or which appear after program compilation use relative (logical or virtual) addresses which start at 0. They also use relative addresses for the operands.

Let's assume that this program (the program area and the data area) is stored on disk and is going to be executed. Thus, it needs to be loaded to memory. The operating system (OS) loaded this program to memory starting at address, say, 50000. (Thus the program counter – PC – contains 50000.) The program is ready to execute. Before any instruction of the program is actually executed, the program's operands relative addresses have to be translated to physical (real or absolute) addresses. The operands represent relative (logical or virtual) addresses - memory locations 90-92. As the program area and data area are loaded starting at 50000, the references to memory locations 00-02 and 90-92 become irrelevant because now the program is stored in memory locations 50000-50002 and data is stored at memory locations 50090-50092.

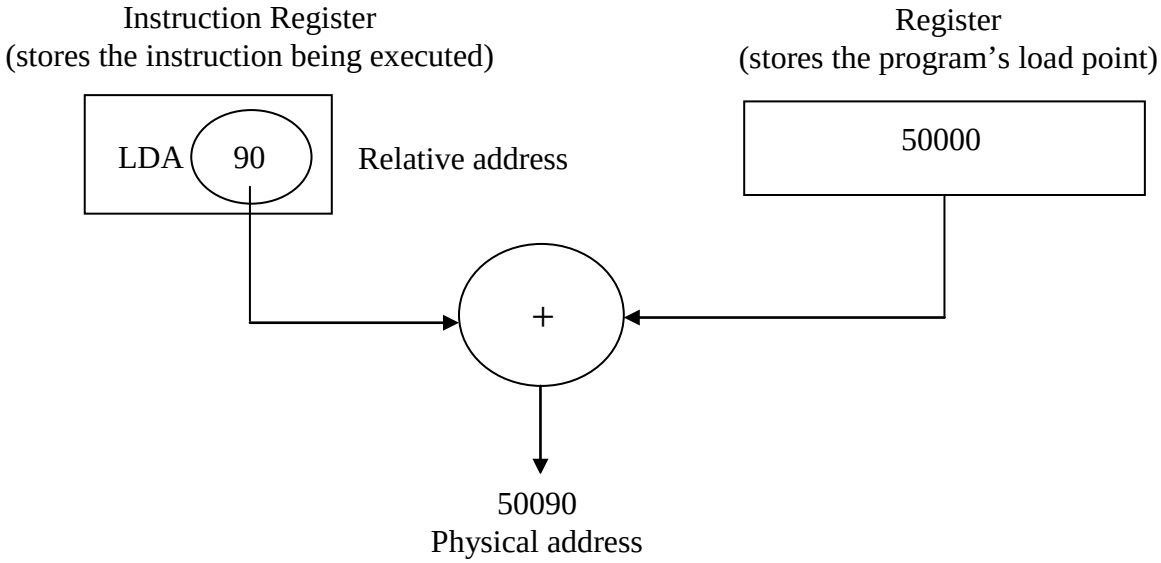
Memory

Address	Instruction or data	
00	?	
01	?	
02	?	
---	---	
---	---	
90	?	
91	?	
92	?	
---	---	
---	---	
50000	LDA 90	
50001	ADD 91	program area
50002	STO 92	
---	---	
---	---	
50090	10	
50091	15	data area
50092	25	

How are the operands' relative addresses translated to the physical addresses?

It is done this way!

The program load point, which is 50000, is loaded into one of the base registers in the CPU and the instructions' operands have to be modified by the program's load point. Simply, the operand (the address field of the instruction) has to be added to the contents of the base register.



This address translation is done after the fetch phase of the machine cycle and before the execute phase. It is done at step 3 of the machine cycle (IR [addr] →MAR) as the MAR needs a physical address. Address translation is very fast as it is implemented in hardware.

The example above illustrates a simple case of address translation. It assumes that the entire program is loaded into memory and executed. The process of address translation is somewhat more complicated because programs are divided into segments, pages, and segments and pages. We will discuss it soon.

The program load point can be changed. The program above or any other program can be swapped out of memory and swapped back in to memory at a different memory location, say, 250000, determined by the OS. The register which contains the program's load point will now contain 250000, and all program's operands will be modified by 250000.

Programs can be relocated in memory. This happens all the time. It is a very important concept in multiprogramming. During execution time, the program can be moved to a different memory location without changing any instruction. Only the contents of the base register needs to be changed. After a program is compiled, the program's relative (logical or virtual) addresses never change. However, physical (real or absolute) addresses change and depend on the program's load point.

Program instructions do not use physical addresses because we simply do not know where the OS will load the program in memory. Also, it would be very inconvenient to use physical addresses because instruction words will be very long and occupy much memory space.