알기쉬운

電子情報處理組織(EDPS) ◀Ⅶ▶

CHAPTER 6.

圖協 出版部

기억장치 (2)

6.5 Calculating

일단 데이터가 컴퓨터 시스템 내로 읽혀 들어가서 기억 장치의 해당 장소에 배치되면 연산이 시작될 수 있다. 각 컴퓨터는 Built-in operation이나 프로그램의 제어 아래 덧셈, 뺄셈, 곱셈, 나눗셈을 수행할 수 있다. 대부분 사무적인 적용을 위하여 이러한 조작은 충분하다. 많은 진보된 과학적인 처리 조차도 아주 복잡한 수식이 기본적인 연산의 단계로 변화될 수 있다. 그렇지만, 많은 특별한 조작은 수학적인 문제를 쉽게 해결하기 위하여 어떤 시스템에 의해 수행될 수도 있다.

간단한 연산의 모든 조작에 있어서 승수와 피승수, 제수와 피제수 같은 최소한 두 가지 요소가 포함된다. 이러한 요소들은 적(product)과 상(quotient) 같은 결과를 산출하기 위해서 기계의 연산장치로 조작된다. 그러므로, 모든 연산에 있어서 최소한 두 개의 기억 장소가 필요하게 된다. 하나의 양(one quantity)은 보통 주기억 장치에 있고, 다른 하나는 Register에 있게 된다. System/360에서는 양쪽 양은 Register에 있게 된다.

연산은 한 Factor를 Register에 위치시키는 것으로 시작할 수 있으며, 동시에 앞에서의 Factor나 그곳에 포함되어 있을지 모르는 결과를 깨끗이 한다. 명령어의 Address part는 첫째 Factor의 기억 장소를 지정한다. Register는 Operation에 의해 알게 된다. 어떤 컴퓨터에 있어서, 한 개 이상의 Register가 연산에 이용된다. 이러한 경우에, 번지는 사용될 Register를 지정하게 된다.

Factor 중의 하나가 Accumulator나 다른 알맞는 Register에 적당하게 위치하게 될 때, 사실상의 연산은 그 Operation part가 수행될 연산을 지정하고 Operand가 둘째 Factor의 장소를 지정하는 명령어에 의해 수행된다.

컴퓨터는 두 가지 Factor에 의해서 작동한다. 하나는 Register이고 다른 하나는 기억 장치이며, 이 양쪽 어느 곳에서던지 직접 결과를 산출한다.

결과는 어떤 Record의 Field로서 기억영역에 옮겨질 수 있다. Field는 Quantity, Amount, Name, Identify 등을 표시하기 위한 문자나 숫자에 관련된 배열이다.

어떤 실제적인 연산의 수는 단일한 명령어의 연속에서 많은 Factor 위에 일어날 수 있다. 즉, 한 Factor는 Register에 위치할 수 있고, 곱해질 수 있으며, 또한 몇 개의 다른 Factor는 그 결과에 더해지거나, 또는 그 결과로부터 빼어지기도 한다. 나눗셈도 후에 실행될 수 있으며, 덧셈이나 뺄셈의 다른 조작도 이 상(quotient)을 사용하여 계속될 수 있다. 중간 결과는 어떤 때에도 저장될 수 있다.

예를 들어, Employee hours worked를 갖고 있는 한 Field가 Register 내에 위치할 수 있으며, 급료를 산출하기 위하여 시간당 급료를 곱해서 급료를 구할 수 있다. 잡무와 보우너스 합계도 후에 Pay record에 내장되어 있는 정규의 전체 급료 합계를 밝히기 위하여 더해질 수 있다.

Total regular earning은 평균 시간율을 산출하기 위해서 시간으로 나누어질 수 있다. 이 비율은 시간외 수당을 산출하기 위하여 1.5 초과 시간으로 곱하여진다. Total gross pay는 뒤에 연산되어 내장된다. 세금은 연산된 Gross pay를 사용함으로써 계산된다. 다른 Payroll 데이터는 그들이 연산한 합계를 사용으로함써 모아진다. 세금의 합계와 공제액의 중간 결과와 Net pay는 결국 Pay record에 모두 저장된다.

Register 내용의 Shifting과 반올림 조작은 역시 결과를 조정하고, 늘이거나 줄이기 위해 주어진다. 이러한 조작으로서 10진수치는 처리되고, 소숫점의 배치를 위한 Direction은 컴퓨터에 주어지게 된다.

모든 계산은 기억장치나 관련된 Register에 있는 Factor의 대수적 부호(algebraic sign)를 계산에 넣어야 한다. 결과적으로 컴퓨터 시스템은 저장과 Factor의 부호를 식별하기 위한 어떤 설비를 갖추고 있다.

만약, Record가 Fixed word의 데이터로 구성되어 있다면, Word의 한 Position은 Sign position으로 지정되며, 자동적으로 그 Word에 동반된다. Register 역시 기억 장소의 특별한 부호 Position이나 프로그래머에게 이용될 수 있는 Sign indicator를 포함하게 된다, 이러한 방법으로 결과의 부호는 그 결과와 함께 연산 후에 결정된다. 컴퓨터는 모든 기본인 산술 연산에 있어서 대수학 법칙(rule of algebra)을 따른다.

Word의 크기, 양, 값은 각각 특정한 시스템의 설계에 따른다. Factor의 위치 선정, 결과의 크기 등을 지휘하는 엄격한 규칙은 얼마간 시스템에 따라 변한다.

결과가 Register의 용량을 초과할 것으로 예측되는 모든 경우에, 프로그래머는 그의 데이터가 부분적인 결과를 산출하도록 조정해야 하며, 후에 합계하기 위해서 이것들을 결합시켜야 한다. Scaling의 다른 조작은 아주 크거나 작은 수와 분수가 편리하게 처리될 수 있도록 하기 위해 실행될 수 있다. 처음에 수학적인 적용을 위해 설계된 컴퓨터가 보통 이 목적을 위해 일련의 특별한 조작을 포함한다(floating-point operation 참조).

모든 컴퓨터 시스템에서 연산은 입출력에 비해 아주 높은 속도의 비율로 수행된다. 왜냐하면 읽기와 쓰기는 계산이 전자적으로 수행되는 동안 기적계인 장치와 Document의 이동을 필요로 하기 때문이다. 많은 상업적 적용에 있어서 계산은 비교적 간단하며, 시스템의 전체적인 도입은 보통 입출력 장치의 속도에 의해 좌우된다. 수학적인 적용에서 그 상황은 반대이다. 계산은 복잡하며 연루되어 있고, 높은 계산 속도가 필수적이다. 어떤 특별한 시스템의 설계는 계산과 Record-handling능력간의 사실적인 조화를 달성할 수 있도록 해야 한다.

6.6 Logical Operation

내장 프로그램 컴퓨터가 그 자체의 명령어에 따르는 순서는 두 가지 중의 한가지 방법으로 결정된다. 하나는 컴퓨터가 명령어를 연속적인 기억 장소 내에서 발견하는 것이거나 또는 Instruction operand가 역시 각각 다음에 오는 명령어를 지정하는 것이다. 만약, 명령어가 단지 Fixed pattern에서 순서대로 따라 나올 때, 프로그램은 그 Procedure에서 예외를 다룰 가능성을 갖지 않거나, 또는 처리되는 데이터에서 직면하는 특별한 상태의 기초 위에서 다른 방도를 선택할 어떤한 능력도 없이 조작의 단일한 통로만을 따라가는 것이다. 더우기, 주어진 일련의 명령어를 반복하기 위해 컴퓨터를 다시 놓을 방법이 없는 경우에 File에 있는 각 Record를 위하여 완전한 프로그램을 갖는 것은 필요하다.

[Fig. 6-9]에서 Block diagram으로 설명된 프로그램을 생각해 보자. 이러한 명령어는 단지 한 레코오드 T를 계산하기 위해 취해졌다. 그러나 처음 명령어로 되돌아 감으로써 Loop로서 같은 프로그램을 반복하여 어떤 수의 레코오드도 처리할 수 있다. 이러한 목적으로 다른 명령어가 처음 명령어로 되돌아 가기 위해 주어진다

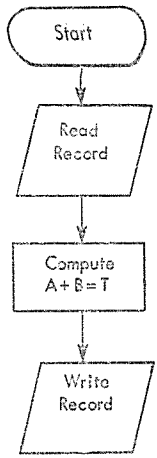

Fig. 6-9. Program flowchart A+B=T

(Fig. 6-10). 일단, 이 프로그램이 시작되면 더 이상 처리할 레코오드가 없을 때까지 작동이 계속된다. 프로그램 Loop는 일반적이며 여러가지서 방법으로 끝나게 할 수 있다.

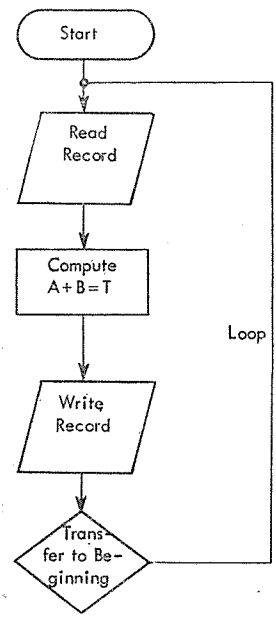

Fig. 6-10. Program loop

예를 들면, 컴퓨터는 계산되는 매 시간 마다. T를 검사하기 위해 지시될 수 있

으며, T의 값이 음수가 되면 어떤 루우틴으로 가게끔 지시될 수 있다(Fig. 6-11).

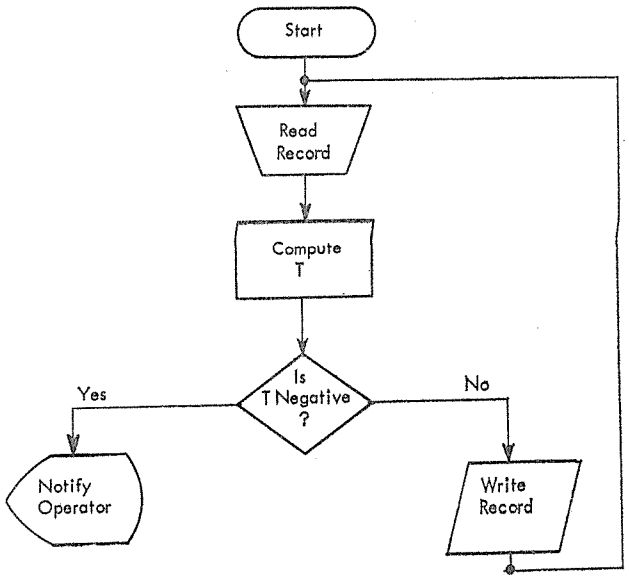

Fig. 6-11. Conditional transfer

이러한 경우에, 명령어는 조건부 Transfer가 된다. 프로그램은 만약, 미리 결정된 상태를 만족하면 반복된다. 컴퓨터는 역시 10 records 동안 프로그램을 실행하고 어떤 루우틴으로 가라고 지시될 수 있다(Fig. 6-12).

상수 10과 1이 컴퓨터 내에 있고, Loop가 이루어질 때 마다 10에서 1을 뺀다고 가정하자. 10회전 후에 10을 포함하고 있던 장소는 0으로 채워져 있을 것이다. Transfer나 Branch가 후에 그 Loop를 끝낸다.

조건부의 Transfer나 Branch operation은 프로그램의 정상적이거나 직선통로 밖에서 처리될 특별한 목적의 Subroutine을 발생시키기 위해서 사용된다. 이 Subroutine은 예정된 예외나 조건이 기계에 알려졌을 때에 실행된다.

Subroutine의 하나의 일상적인 예는 그것이 마그네틱 테이프로부터 읽거나 기록한 레코오드의 정확성을 검사하는 것이다. 각 레코오드가 중앙 처리 장치로 들어 가거나 빠져 나오는 것으로 입출력 과오 탐지기는 검사된다. 만약, Indicator가 켜지면 컴퓨터는 Error를 고치기 위한 시도로서 명령어의 Subroutine이 들어 가도록 지시된다. 이러한 조작(the reading only)을 위한 프로그램 Logic은 [Fig. 6-13]에서 보여준다. 비슷한 Loop도 역시 Writing을 위해 포함된다.

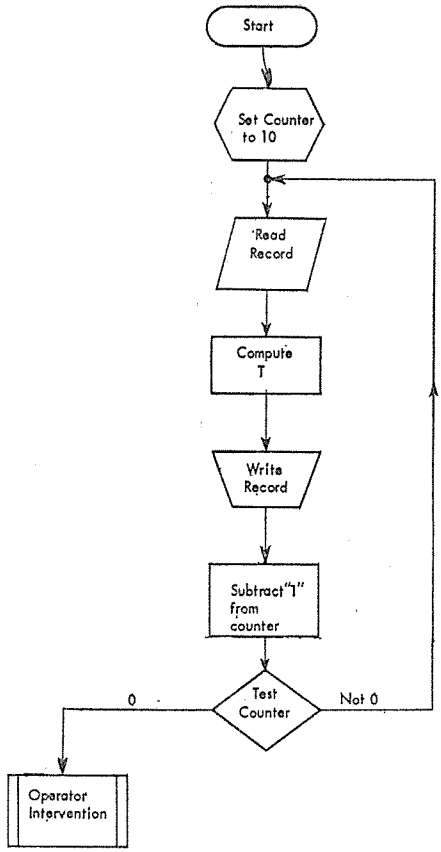

Fig. 6-12. Record count conditional transfer

Reading error가 발견되었을 때 Branch가 Error subroutine을 실행한다. Counter는 다시 읽기가 시도되는 횟수를 세기 위하여 10으로 다시 채워진다. 테이프는 Error 위로 Backspace하며, 두번째 읽기 명령어가 주어진다. 다른 검사는 이 조작이 정확한가를 결정하기 위해 시작된다. 만약 그렇다면 Transfer는 Main program에 다시 돌아오며, 거기에서 계산이 계속된다.

만약, Error가 지속되면, Counter로부터 1이 빼지며, Counter는 0으로 조사된다. Error loop가 다시 들어오면, 두번째의 다시 읽기와 조사가 실행된다. 기계는 10번 반복하여 읽을 수 있다. 만약, Error가 정정되지 않으면 프로그램은 다른 Routine로 이동되며, 거기서 Error record를 잘라내기 위한 분류 절차를 실행하며, 다시 그것을 읽어들이고 다음 레코오드를 처리한다. 그렇지만, Hypertape와 2400 series nine-track 테이프에서 주기적인 여분 문자 형태는 자동적으로 Single-track

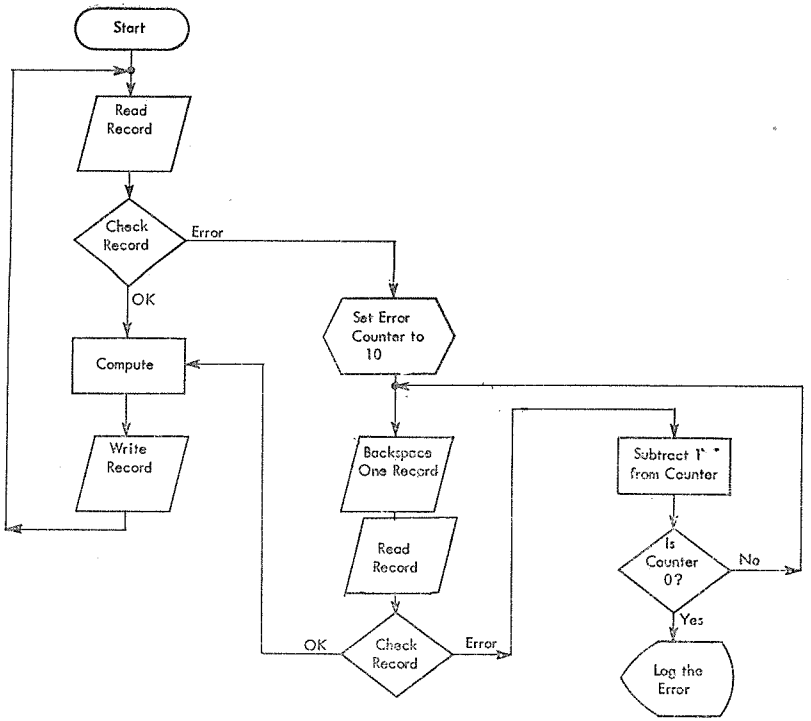

Fig. 6-13. Read error loop

error를 정정하며, 이와 같이 하여 많은 경우에 이러한 형태의 프로그램의 필요성을 배제하였다. 역시, 입출력 제어 시스템과 Operating system(IBM이 공급하는 programming system)은 형태의 조사를 포함하기 위하여 Problem program의 필요성을 배제하므로, 이러한 모든 자료 처리 조직에 있어서 이용될 수 있다.

프로그램은 역시 기계가 알아볼 수 있도록 하기 위해서 하나의 파일로부터 처리되는 하나나 여러가지 형태의 레코오드로 배열될 수 있다. 연산의 방법은 스토러지 내의 레코오드의 형태에 따라 변화될 수 있다. 이 절차는 하나의 Master file(예를 들면, 파일 유지의 업무)에 대한 많은 형태나 종류의 업무가 처리될 때 일반적이다.

Master stock 상황 레코오드의 파일(data set)이 생산 계획에 이용될 수 있는 많은 부분에 영향을 미치는 양을 포함하고 있다고 가정하자. 단지 설명을 위한 목적으로, 이 레코오드가 역시 재고품의 현황에 관계 있는 다른 중요한 정보를 갖고 있을 때, 이 실례는 단지 이용도를 나타내 주는데 사용되는 영역에서만 관계가 있다. 이러한 영역은 ① 재고품의 양, ② 주문의 양, ③ 이용할 수 있는 양이다.

부분적인 이용도의 상황에 영향을 미치는 업무는 매일 발생한다. 이러한 업무는

활동의 각 형태에 적합한 동일한 숫자 코우드로서 카아드에 펀치된다.

코우드는 다음과 같다.

코우드 1 영수(receipt)

코우드 2 주문(order)

코우드 3 회수(withdrawal)

코우드 4 정산 플러스(adjustment plus)

코우드 5 정산 마이너스(adjustment minus)

각 업무는 스토러지 내에서 배열되며, 코우드는 이 업무가 어느 부류에 속해 있는가를 결정하기 위하여 분석한다(Fig. 6-14). Branch 명령어는 후에 Availability를 계산하고 일치하는 Master record를 조정하고자 특정한 프로그램 서브루우틴을 옮기게 된다. 조정된 Master record의 읽기와 쓰기는 순서도에서 나타나지 않는다.

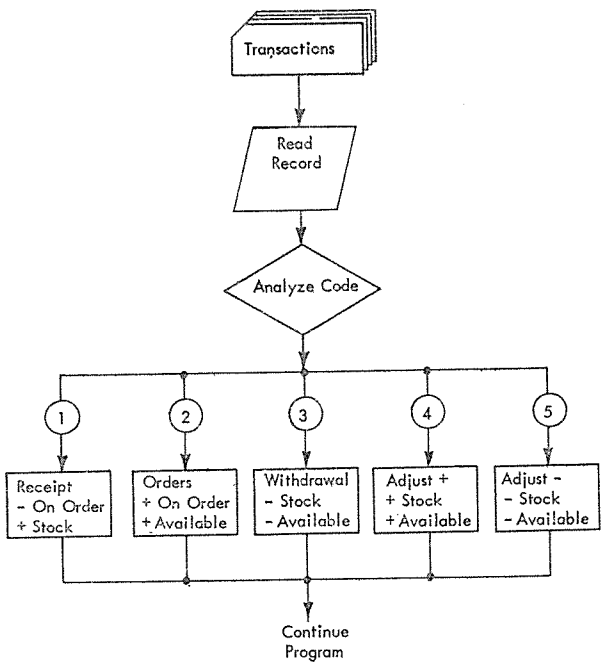

Fig. 6-14. Branching by code

6.7 Comparing

컴퓨터가 Program logic의 기초 위에서 제한된 결정을 내릴 수 있는 능력은 사

실상 비교 작업을 통해서 확대되었다. 이러한 작업은 컴퓨터가 스토러지에 있는 두 개의 Data field가 조화하는지, 또는 어느 하나가 다른 것보다 크던가 작은가를 결정할 수 있게 해준다. 비교작업의 근본은 회로 내에 설정된 미리 결정한 순서에 의한 세트이다.

순서는 모든 형태의 레코오드의 정상적인 서류 정리 순서로 생각될 수 있다. 예를 들면, 숫자 0에서 9까지의 일상적인 상승적 순서에서 숫자 9가 그 열에서 가장 큰 것으로 생각된다. 같은 방식으로, 문자 2가 알파벳에서 가장 큰 문자로 생각된다. 그러므로, 컴퓨터에서는 어떤 파일로서든지 순서에 있어서 숫자 162는 159보다 크며, Jones라는 이름이 Smith라는 이름보다 하위이다. /, @, *, −와 같은 특수 문자도 역시 포함된다. 왜냐하면 모든 컴퓨터 데이터는 다른 가치(value)와 비교할 수 있는 Value값을 갖기 때문이다. 이것은 Collating sequence로 알려져 있다.

비교 작업은 파일의 순서 점검, 프로그램 작성, 절차 분류 또는 필요한 명령에서의 레코오드 재조정에 사용된다. 만약 모든 파일이 이 공통의 영역과 같은 순서이면, 한 레코오드에 있어서 동일한 영역과 다른 레코오드의 동일한 영역과의 비교는

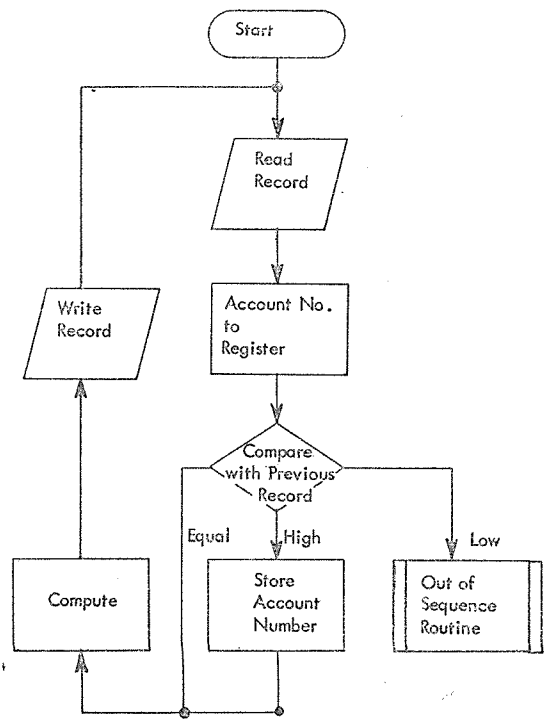

Fig. 6-15. Sequence checking

하나의 처리 절차에서 컴퓨터가 많은 관련된 파일을 다룰 수 있게 해준다.

한 개나 양쪽 Field는 스토러지 레지스터들 내에 위치하게 된다. 비교명령어는 후에 둘째 Field에 대하여 첫째 Field와 비교를 실행한다(시스템 /360에서 둘째 field는 둘째 레지스터에 있을 것이고, 다른 시스템에서는 주기억 장치 내에 있을 것이다). 비교의 결과는 그 상태를 결정하고자 후에 조회되는 Indicator 또는 Trigger에 의해 높고 낮거나, 또는 동일 상태로서 기록된다.

만약, Indicator가 켰졌을 때 Branch (transfer) 명령은 기계를 비교의 결과에 따라 처리가 계속될 Subroutine으로 옮기게 된다. [Fig. 6-15]는 레코오드들 중의 하나의 파일에서 순서 점검을 위한 전형적인 프로그램 조정을 보여준다. 파일에 있는 모든 레코오드는 계산 숫자에 의한 상향적인 순서로 생각된다.

입력 영역은 동시에 입력 장치로부터 레코오드가 받아 들여지게 되는 스토러지 내의 한쪽에 놓여 있다. 둘째 영역도 역시 다음 레코오드로부터 숫자를 저장하기 위하여 스토러지 내에 지정된다. 이 영역의 목적은 들어오는 레코오드의 숫자와 앞의 레코오드의 동일한 영역과 비교할 수 있도록 하는 것이다.

만약 파일이 상향 순서라면, 들어오는 레코오드는 항상 그것에 앞서 간 레코오드보다는 크다. 한쌍의 레코오드가 들어왔을 때 들어오는 레코오드는 앞서 들어온 레코오드와 같다. 만약, 어떤 들어오는 레코오드가 앞서의 레코오드 보다 적을 때, 이것은 Out-of-sequence condition으로 인식되며, 그 프로그램은 정정 작업을 받기 위해 Subroutine으로 옮겨진다. 각기의 High comparison 후에 숫자 영역은 다음 레코오드와 비교되어야 하는 스토러지 내에 위치하게 된다.

6.8 Instruction Modification

앞서의 몇가지 예는 어떻게 Branch나 Transfer 명령어가 컴퓨터로 하여금 프로그램을 따라 잡다한 진로를 진행하였는가를 보여 주었다. 처리될 루우틴은 Zero balance, Error condition 등에 의해 이미 세트된 앞서의 비교나 또는 Indicator의 테스트 결과에 의존한다.

프로그램을 변경하는 다른 방법은 명령어 자체의 명령 부분(operation part)을 바꾸는 것이나 또는 수정하는 것이다. 예를 들어 명령어 수정은 선택할 수 있는 두 개의 길 중의 하나를 기계가 선택하도록 해주는 프로그램 스위치를 세우는데 사용할 수 있다. 스위치는 명령어에 의해 켜지거나 꺼진다. 스위치의 사용 예로서는

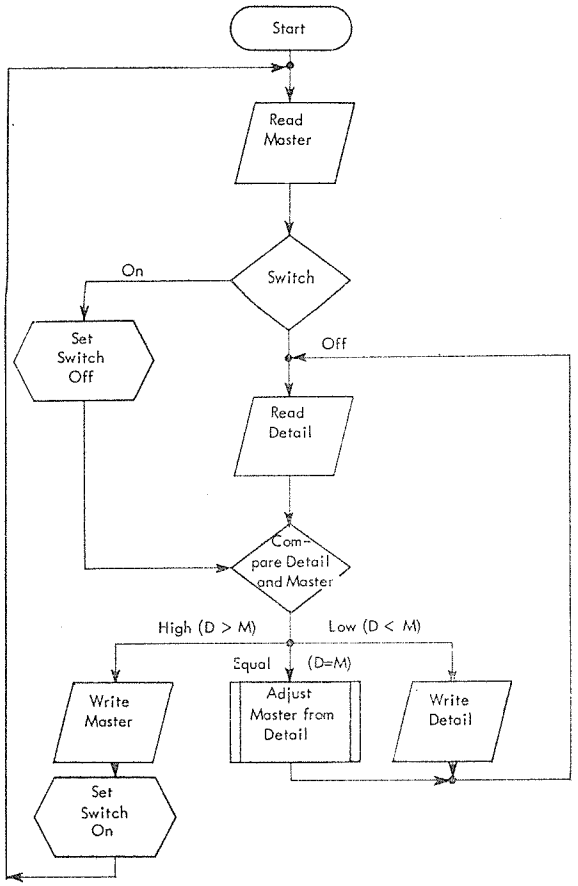

Fig. 6-16. Program switch

[Fig. 6-16]에 표시하였다.

두 개의 파일(data set)이 읽혀지고 있다고 가정하자. 이 파일은 Part number, Account number, Employee number와 같은 Common identifying field의 순서이다. 한 파일이 Master file이고, 다른 파일은 Master file의 조정을 나타내는 업무 파일이다. 업무에 동일한 Master file(data set)을 적용했을 때 세 가지의 상태가 일어나게 될 것이다.

1. 한 개나 그 이상의 업무가 하나의 Master record에 어울릴 것이다.
2. Master record를 위한 업무가 없다.
3. Master에 조화되지 않는 업무가 생길 것이다. 이러한 것은 과오나 새로운 첨가이다.

한 단계에서 두 개의 파일(data set)을 처리하는 것은 필요하다. 즉, 만약 한 개가 있으면 각 업무 레코오드는 동일한 Master record에 대해 비교해야 할 것이다. 만약, 여러 개의 업무가 같은 Mas ter record에 적용되면, 업무 파일(data set)은 새로운 Master record를 읽지 않고 읽기를 계속해야 한다. 반대로, 만약 Master record가 동일한 업무 없이 읽혀지면 이 레코오드는 변함 없이 인쇄되며, 다음의 Master는 읽혀들어가게 된다. Master record의 읽기와 쓰기는 조화된 업무가 발견될 때까지 계속된다.

〔Fig. 6-16〕의 순서도는 처음 Master record가 읽혀지는 것을 보여준다. 스위치 명령어는 Master와 업무의 읽기 사이에 삽입된다. 작업이 시작될 때, 하나의 업무가 읽혀 들여가도록 하기 위해서, 이 스위치는 꺼진다, 업무의 동일한 영역은 Master에 대하여 비교된다. 만약 같으면, Master는 정리되고 두번째의 업무가 읽혀진다. 만약 이 업무가 Master에 대하여 적용되지 않으면(master는 storage 내에 있다) 비교했을 때 크게 되어야만 한다. 앞서 조정된 Master는 후에 인쇄되며, 스위치가 켜지게 된다. 새로운 Master는 후에 Storage 내에 위치하게 되며, 스위치가 켜지기 때문에 업무는 읽혀지지 않는다. 대신, 기계는 직접 비교 명령을 전달하게 된다. 스위치는 이것이 일어날 때마다 꺼지게 된다. 작업은 스토러지 내에 있는 각각의 새로운 레코오드와 비교하며 계속된다. 만약 업무가 느리면 독립된 출력 장치를 통해 작업을 인쇄해 내며, 새로운 업무를 그 다음에 읽어 들이게 된다.

스위치가 켜졌을 때에는 무조건 이동의 지정된 Operation part를 가지고 있다. Address part는 비교 명령어(compare instruction)의 Location이다. 스위치를 끄면 Operation part는 No operation으로 변한다. 이 경우에 기계는 명령어를 무시하며 아래의 명령어를 실행한다. 업무를 읽어라.

6.9 Address Modification

명령어의 Address position은 역시 데이터로 취급된다. 명령어 번지는 연산에 의해 수정될 수 있으며, 다른 번지나 요소들에 대해 비교될 수 있으며, 임의로 스토러지 내에서 옮길 수 있다. 번지 수정은 두 가지 목적에 이용된다.

1. 프로그램 내의 명령어 전체 수를 데이터나 다른 요소의 스토러지 용량을 보호하기 위하여 변경시킬 수 있다. 한 개의 명령어나 또는 하나의 명령어들의 연속은 스토러지 내에서 변화하기 쉬운 Location에 번지를 정할 수 있도록 해준다.

2. 프로그램에 의해 제어되는 작업의 기본적인 흐름은 입력 데이터, 연산의 결과 여러 가지의 Error condition, End-of-file의 감지 등에 의해 요구되는 것으로서 바꿀 수 있는 절차의 형태로서 공급할 수 있다.

예를 들면, Table로부터 정보를 받는 명령어의 Address portion은 Register에 있는 문자의 Value(문자 번역 테이블을 다루는 유용한 절차)나 또는 Terminal line number의 Value(주어진 line을 향한 입력의 정당한 상태를 구성하는데 필수적인) 에 의하여 수정된다.

6.10 Indexing

많은 컴퓨터에 있어서 명령어의 Address portion은 한 개나 여러 개의 특수한 목적의 Counter에 포함되어 있는 변하기 쉬운 양의 덧셈이나 뺄셈에 의해 수정될 수 있다. 이 Counter는 특별히 이러한 목적으로 따로 설치하였을 때에 Index register라 불릴 수 있으며, 또는 이것은 Index word라 부르는 코어 스토러지 내부에 미리 결정된 Location일 수도 있다. 컴퓨터는 몇개의 Index register나 또는 Index word를 위한 많은 Storage location을 가질 수 있다. Index register와 Index word는 같은 기능을 수행한다. 그렇지만, Index word가 보통 더욱 빠르게 프로그램에 접근할 수 있으며, 결과적으로 그의 사용에 많은 융통성을 제공한다.

Indexing 형태를 가진 컴퓨터는 Instruction operand의 부분으로서 표시하는데 특정한 Register나 Word가 필요한 Expanded instruction format을 사용한다.

50개의 양이 Location 1001에서 1050까지 스토러지의 Ascending word position에 위치하고 이 양을 레지스터의 내용에 더한다고 생각해 보자. Indexing이나 Address modification 없이는, ADD 1001, ADD 1002, ADD 1003 등과 같이 1씩 증가시켜 각 명령어의 번지를 50번이나 더하는 명령을 반복하는 것이 필요하다.

Indexing을 사용할 때, 덧셈 명령어는 50이라는 양을 가진 Index register에 의해 감소되는 번지를 가진 ADD 1051로서 쓰여지게 된다. 번지수는 1051로 남아 있지만, 컴퓨터는 실제적인 번지수 1051−50 또는 1001번지로 계산한다. 덧셈 명령이 처리되었을 때, Index register의 내용은 역시 1씩 감소하여 나머지로 49를 남겨둔다. 같은 덧셈 명령이 다시 처리되었을 때 다시 같은 Index register의 내용은 다시 감소하며, 실제의 번지는 1051−49 또는 1002가 된다. 만약, 한 프로그램 Loop가 이러한 과정을 반복하는 형태를 취하면, 덧셈 명령어의 실제 번지는 처리된 각 시

점마다 1씩 증가한다(index register가 1씩 감소하는 것으로서). Index register가 0이 되고 50개의 양이 모두 끝마쳐지게 되면 Loop는 끝난다. 결과적으로 컴퓨터는 같은 명령어를 사용하여 50개의 작업을 수행하였다.

〔Fig. 6-17〕은 Index loop의 Flow diagram이다. 처음 명령어는 Index register 4에 50개의 양을 두고 있다. 1051번지를 가진 덧셈 명령어 역시 주어진 번지가 Index register 4에 포함되어 있는 양에 의해 수정되는 지정을 그 Operand의 부분으로서 부여 받게 된다.

다음의 명령어는 Branch on index이며 이것은 Index register의 내용을 1씩 변화시키는 것을 의미한다. 만약, Register의 내용이 0보다 클 때, 덧셈 명령을 반복하기 위하여 Branch한다. 만약, Index register의 내용이 0과 같을 때, 프로그램에서 다음 명령을 계속한다. Indexing 형태는 반복적인 연산의 프로그래밍과 또는 다른 작업을 아주 간단하게 해 주었으며, 또한 요구되는 명령어의 수를 줄여 주었다.

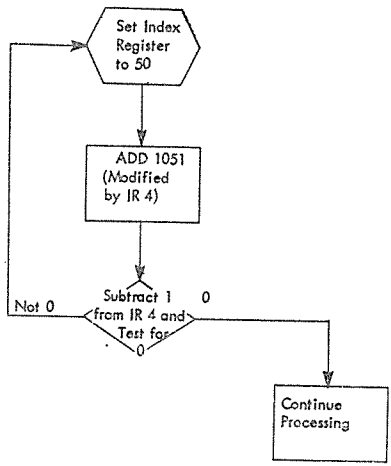

Fig 6-17. Index loop

6.11 Indirect Address

앞서의 설명에서 토의된 모든 명령어의 번지는 직접적으로 분류되었다. 즉 그들은 직접 데이터의 Location이나 스토러지 내에 있는 다른 명령어의 Location을 지정하며, 기계 장치를 선택하거나 사용될 제어의 형태를 열거한다.

번지는 역시 간접적으로 정해줄 수 있다. 이러한 번지는 단지 다른 번지를 가진

Storage location을 지정할 수 있다. 두번째의 번지는 차례대로 데이터의 Location, 기계 장치, 또는 제어 기능을 지정한다.

Indirect addressing은 번지 수정을 수행하는데 아주 쓸모가 있다. 예를들면, 프로그램에 있어서 그 프로그램이 진행되고 있는 동안 그 값을 바꾸어야 하는 많은 명령어를 지정하는 경우에 필요하게 될 것이다. Indirect Addressing이 없다면 많은 수정 명령(modification instruction)이 필요하게 될 것이다.

그렇지만 만약 명령어가 한 코어 스토리지 Location에 간접적으로 번지를 갖게 되었을 때, 그 Location은 하나의 번지를 가질 수 있게 된다. 그 번지의 Value는 프로그램에서 사용된다. 그러므로, 모든 명령어 번지를 바꾸거나 수정하기 위하여 단지 그 명령어들이 지정하는 하나의 사실상 번지만 수정하는 것이 필요하게 된다 (Fig. 6-18). 프로그램을 통하여 나타나는 어떠한 많은 수의 Indirect address도 하나의 사실상의 번지를 지정할 수 있다. [Fig. 6-18]에서 각각의 간접적으로 번지를 잡은 명령어(SEL 4069)는 Location 4069 대신에 Core location 200의 내용을 가지고 오게 될 것이다.

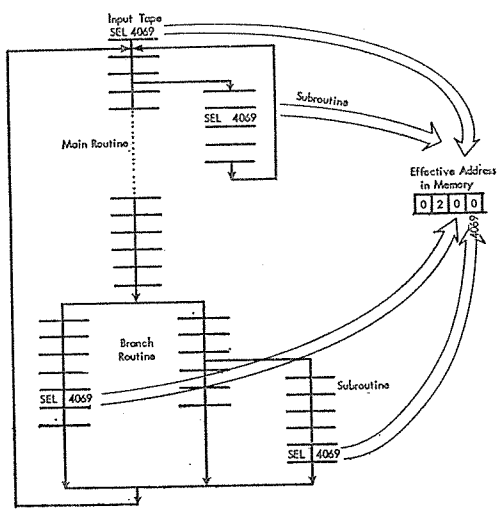

Fig. 6-18. Indirect Addrses

6.12 Linking

지금까지 우리는 조건부의 Branch(transfer)에 대해 논의하였으며, 방금 Indirect addressing의 정의에 대하여 소개하였다. 최근의 컴퓨터에 있어서 한 명령어를 그

프로그램이 서브루우틴이 끝난 후 Store, Unload, Reload 하지 않고, 다른 House keeping job을 수행하는 주프로그램으로부터 떨어져 나온 그 점으로 되돌아 갈 수 있도록 하기 위하여 서브루우틴에 'Link'시키는 것은 가능하다. Linking procedure에서 Indirect addressing을 사용함으로써, 프로그래머는 독립적으로 많은 서브루우틴을 쓸 수 있다. Link 명령어는 다음에 컴퓨터가 서브루우틴이 들어오는 매 시간마다 서브루우틴의 적당한 명령어에 있어서 요구하는 사실상의 되돌아올 번지를 삽입하도록 해 주었다. 우리는 이러한 각각의 Subroutine을 Data set(회전하는 bookcase로서)에 작용하는 디스크 스토러지의 Library에 있는 Book과 맞먹는 것(Fig. 6-19)으로 생각해야만 한다. 엄밀하게 Linking 방법은 컴퓨터 대 컴퓨터의 방법과는 다르다. 간단한 Procedure는 아래에서 개설하였다.

많은 여러 가지 형태의 보고서가 IBM 1050 원격 터미널에서 인쇄되고, 또한 컴퓨터는 역시 다른 원격 터미널로부터 입력을 등록한다고 가정하자. 역시 출력 리포오트의 각기 다른 형태의 정보로 내부처리가 완료되자마자 그 정보는 Standard BCD format으로 Disk file 내에 저장된다고 가정하자. 메시지가 Disk file로부터 빼내어지는 매 시간마다 이 메시지는 약간 다른 1050 BCD code format으로 번역되어야만 한다.

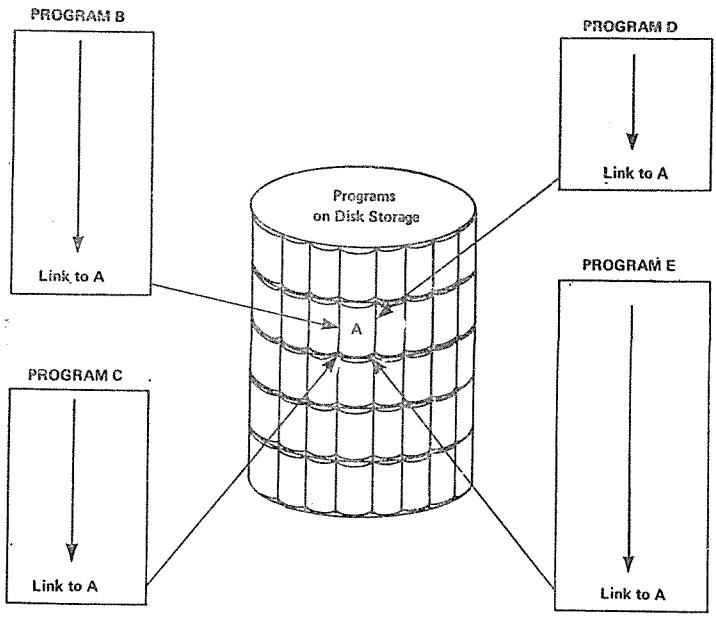

Fig. 6-19. Modular programs with linking

그 프로그램 순서는

INSTRUCTION NUMBER

1000 Link to : 디스크로부터 message 1을 읽어라.

1001 Link to : 1050 번역 루우틴.

1002 Link to : 만약 있으면 line 1로부터 입력을 처리하라.

1003 Link to : line 2 위의 1050에 메시지 1을 써라.

1004 Link to : 만약 있으면 line 3으로부터 입력을 처리하라.

1005 Link to : 디스크로부터 메시지 2를 읽어라.

1006 Link to : 1050 번역 루우틴.

1007 Link to : 만약 있으면, line 4로부터 입력을 처리하라.

1008 Link to : line 2 위의 1051에 메시지를 써라.

이러한 경우에 프로그램이 1050 번역 루우틴에 연결되는 첫번째는 프로그래머가 Return address로서 1002번지를 넣거나 또는 컴퓨터가 자동적으로 하게 된다. 두번째 Return address는 1007이다. Step 1002에 있어서, 프로그램은 어떠한 상태를 테스트하기 위하여 루우틴에 연결된다. 이것은 그 Subroutine이 다른 Subroutine으로 끝나게 되는 것을 뜻한다. 만약, Line 1으로부터 입력이 없으면, 즉 어떤 Indicator의 테스트 결과로서, 그 프로그램은 Indirect address(프로그래머가 작성한 프로그램이나 퓨컴터에 의해 미리 저장된)를 발견하기 위해 Branch 한다. 만약, Line 1에 입력이 있다면 Subroutine의 마지막 명령어는 1003으로 되돌아가는 길을 지시하는 Indirect addres를 가져야 할 것이다.

이러한 형태의 프로그램을 필수불가결하게 만드는 오늘날의 컴퓨터 요소는 어떤 형태의 입출력 활동이 시작될 때까지 처리를 계속할 수 있도록 하거나, 또는 Error condition이 수정될 때까지 첫번째 프로그램이 기다리거나, 또는 I/O 작업이 시작되는 동안 두번째 프로그램을 처리하도록 하는 Internal interrupt system이며, 이것은 원격 터미널로부터 Teleprocessing interrupt를 가능하게 한다.

위에서 설명한 간단한 프로그래밍의 예나 Interrupt를 지시하는 프로그램은 Linking의 체계를 갖춘 짧은 Subroutine으로 구성되어야 하며, 만약 컴퓨터 자체가 'Linked to' subroutine에 Return address를 삽입시키지 못할 때는 그 프로그램은 Indirect address의 Table을 세워야 한다는 것을 [Fig. 6-19]로부터 쉽게 알 수 있다.

컴퓨터의 설계에 따라 다소간의 이러한 Linking절차는 자동적으로 이루어질 수 있다. 프로그래머는 Subroutine을 저장하여야 하며, 컴퓨터가 자동으로 그 일을 할

수 없을 때 그 Subroutine을 불러내게 된다.

Interrupt는 간격 계시 기구의 시간 재기나 입출력 조작의 완료와 같은 전형적인 예상된 컴퓨터 기능의 결과로서 발생할 수 있으며, 주기억 장치 내에서 고정된 Location에 있는 번지에 자동적인 연결을 시킬 수 있다.

기계 설계에서 미리 예상하지 못하였던 다른 형태의 Link들은 프로그램 자체에서 Linking과 Indirect address의 유지를 해줄 것을 요구한다.

6.13 Chaining

Chaining은 현재 IBM 컴퓨터에 있어서 몇 가지 가능한 함축성 있는 의미를 내포한다. System/360과 같은 범용 컴퓨터에 있어서 Chaining은 'Command chaining 또는 'Data chaining'으로서 지령 또는 Data address의 Program-linking을 I/O채널에게 위탁한다. Linking은 Program subroutine의 끝에서 Return address를 지키기 위한 시스템을 묘사하였다. 반면, Chaining은 CPU의 활동과 독립적으로 수행될 입출력 채널 지령의 "Subroutine"을 위한 프로그래밍 기법과 비슷하다. 이것은 지령의 부분으로서 채널이 실행되는 입출력 조작을 끝냈을 때 채널이 처리하고자 하는 다음 지령의 번지를 프로그래머가 갖도록 해주는 컴퓨터 시스템의 내에 자리를 잡게 된다. 다음의 지령은 다른 입출력 조작을 시작하거나 또는 주기억 장치 내의 다른 Location으로 데이터를 보내거나 받게 한다.

Chaining의 다른 개념은 Communication 제어를 위해 특별히 설계한 컴퓨터 내에 있다. 여기서 Chaining은 그것이 가능한 어느 곳에던지 앞서 지정한 Block의 마지막 두 개의 문자에서 메시지에 지정한 각각의 새로운 Block의 Address를 자동적으로 넣는 컴퓨터로서, 각 Line으로부터 들어온 데이터는 이용될 수 있는 주기억 장치 Space의 어느 곳이던지 자동적으로 저장되는 것에 의해 자동적으로 Block을 할당하는 시스템이다. 출력 메시지에 있어서 Chaining은 그렇게 아주 자동적인 것은 아니다. 프로그램은 앞서의 마지막 두 개의 문자에서 각각의 새로운 Block(32 문자의) 번지를 넣는다. 이 형태의 Chaining은 각 Line을 위한 정적인 Storage space의 양을 그 회수만큼 앞에 할당할 필요를 없애 주었으며, 공백은 그의 비활용성 때문에 비효과적이며 낭비이다. 이러한 견지에서 Chaining은 앞서 입출력장치의 아래에서 설명한 Data buffering의 유효한 형태이다.