알기쉬운
電子情報處理組織(EDPS) ◀ Ⅵ ▶

CHAPTER 6.

國協出版部

기억장치(1)

6.1 개 요

데이터가 입력 매체에 의해 복사된 뒤에, 그 컴퓨터 시스템은 전체 처리와 결과의 작성을 대신할 수 있다. 그렇지만, Computer system 내에서 일어나는 진행 절차는 System이 수행할 수 있는 Operation의 형식으로 정확하게 정의되어야 한다. 각 Step은 그 Computer의 명령어로 기록되어야 한다.

전체의 처리에 적합한 일련의 명령어를 프로그램이라고 부른다. 현대의 자료처리 조직에 있어서, 프로그램은 Storage 내부에 저장되며, System은 전자적인 속도로 그 명령어를 Access하게 되었다. 이러한 프로그램을 내장 프로그램(stored program) 기법이라 부른다.

6.2 명령어(Instruction)

컴퓨터는 각각의 작업을 수행하기 위하여 주기억 장치에 위치한 특정한 정보의 단위인 명령어에 의하여 제어된다. 이 정보는 수행될 조작으로서 CPU에 의하여 해석된다. 데이터가 들어오면 명령어는 데이터를 위하여 컴퓨터를 제어한다. 만약 어떤 장치가 제어되었다고 하면(예를 들어 magnetic tape) 명령어는 장치에 필요한 조작을 명령한다.

Instruction은 Indicator의 상태를 변경시킬 수 있으며, 또한 기억장치 내의 한 위치로부터 다른 위치로 자료를 옮길 수 있다. 또 한 편 그들은 Tape unit를 Rewind 시키거나 혹은 Counter의 내용을 변경시킬 수 있다. 어떤 명령어 또는 어떤 기계

장치의 결과나 Data indication은 다음 명령어의 기억 장소를 지정할 수 있다. 이러한 방법으로 어떠한 명령어나 다음에 오는 명령어 Block의 순서를 바꾸는 것이 가능하다.

Instruction(Fig. 6-1)은 일반적으로 두 개의 부분으로 구성되어 있다.

Operation	Operand
Select	Tape Unit 200
Read	One Record into Storage Positions 1000-1050
Clear & Add	Quantity in Storage Location 1004 in Accumulator
Subtract	Quantity in Storage Location 1005 from Contents of Accumulator
Store	Result in Storage Location 1051
Branch	To Instruction in Storage Location 5004

Fig. 6-1. Instruction

1. Oeration part는 Read, Write, Add, Subtract, Compare, Move data 등과 같은 명령이고,
2. Operand part는 정보의 번지나 특정한 조작에 필요한 장치를 지정한다.

명령어를 수행하는 동안에 다음 명령어는 기억 장치로부터 선택되고, 중앙처리장치에 분석되며, Operation part는 수행되어질 조작을 지시한다. 이 정보는 Computer에게 특수한 의미를 주기 위하여 코우드화된다.

예를 들면, System/360에서 A는 'ADD'로, C는 'Computer'로, SIO는 'Start input/output', TR은 'Translate'로 성립된다. 다른 컴퓨터는 작업을 정리하기 위하여 다른 코우딩과 문자의 수 또는 Position을 사용한다.

Operand는 Operation의 기능을 더욱 더 한정하거나 증대시킨다. 예를 들면, 연산작업을 수행하기 위하여 포함된 여러 요소들 중 하나의 기억 장소가 지정된다. 입출력 장치에 있어서는 사용될 장치가 명시된다. 읽기와 쓰기에 있어서는 입력과 출력 레코오드를 위한 기억 장치의 영역이 지정되거나 또는 기계 설계에 의해 고정된다.

모든 명령어는 데이터로서 같은 기억 매체를 사용하기 때문에 같은 코우딩의 형태로 표시되어야 한다. 어떤 형태의 컴퓨터에서는 IBM 7090과 같은 명령어는 길이(1 word)가 고정되어 있다. IBM 1401이나 1410과 같은 컴퓨터는 문자의 길이가 변수(Variable number)로 될 수 있다. S/360에서 Instruction은 세 가지 길이 중에

서 어떤 것이라도 될 수 있다. 즉, Half word(two byte), Full word(4 byte), Word-and-a-half(six byte)는 특정한 Instruction operand의 필요에 달려 있다.

일반적으로는 명령어 자체를 위한 특정한 기억 장치의 영역은 준비되지 않는다. 대부분의 경우에 명령어들은 함께 묶여지며, 상향적 순서의 장소에는 컴퓨터로 처리하기 위하여 정상적인 순서로 자리를 잡게 된다. 그렇지만, 처리의 순서는 특별한 명령이나 시스템 내에 있는 데이터나, 장치에 앞서 결정한 데이터의 조건이나, 외부 시스템(teleprocessing input)으로부터의 예기치 않았던 Interruption, 특별한 세트의 프로그램들로부터 서어비스를 요구하는 Hardware의 조건, Unusual 우선권을 요구하는 다른 프로그램에 의해 바뀌어질 수 있다.

완전한 프로그램에 있어서 컴퓨터 조작의 정상적인 순서는 다음과 같다. 즉, 컴퓨터는 이러한 목적으로 지정된 앞서 결정한 기억 장치의 장소를 찾거나 수동적인 재배치에 의해 첫번째 명령어는 처리된다.

컴퓨터는 다음에 다음 명령어를 배치하며 그것을 처리한다. 이 과정을 명령어 하나씩 차례대로 그 프로그램이 완료되거나 컴퓨터가 정지되도록 명령을 받을 때까지 자동적으로 계속한다.

Two-Address Instruction

System/360과 같은 보통 컴퓨터에 있어서, 명령어는 두 개의 Address position을 갖고 있다. 명령어의 기능에 의하면, 두 개의 Address는 한개 장치의 사용을 지정하고, 처리한 데이터나 혹은 처리될 두 요소의 데이터를 지정한다. 사용될 출력 장치는 하나의 번지에 의해 지정되고, 인쇄될 정보의 기억 장소는 다른 번지에 의해 지정될 수 있다.

Arithmetic operation에 있어서 두 개의 Address는 Multiplier와 Multiplicand, Divisor와 Dividend, Addend와 Augend 등의 자료에 두 개의 관련된 요소를 지정할 수 있다.

Single-address 명령어 보다 소수의 Double-address 명령어가 작업을 수행해 나가는 데 필요하게 된다. 이것은 프로그래밍 절차를 간단하게 하여 주었으며, 기억 장치 내의 space를 절약하는 결과를 가져왔다.

Instruction과 Data

주기억 장치 내에서 명령어와 자료를 구별할 수 있는 유일한 방법은 그것이 CPU

로 들어오는 시간을 보는 것이다. 만일, 한 정보가 명령 사이클 동안에 들어오면 그것은 명령어로 간주되며, 다른 형태의 사이클 동안에 들어오면 그것은 데이터로 간주한다.

만약, 이러한 명령어가 자료로서 공급된다면, Computer는 그 자체의 명령으로 일할 수 있다. Computer는 과정을 취급하는 동안에 발생하는 조건에 의해서 자신의 명령의 변경에 따라 Program이 될 수 있다. 거의 무한정한 융통성과 소위 말하는 내장 프로그램의 Logical ability를 갖춘 명령어를 처리하는 것이 이것의 능력이다.

6.3 프로그램의 개발

프로그램을 개발하기 위하여 프로그래머가 반드시 알아 두어야 할 사항은

1. 컴퓨터 시스템으로 작업하는데 이용되는 다른 Operation(그리고 그들의 기능)의 수,
2. 절차 그 자체를 단계적으로 컴퓨터 명령어로 변형시켜야 하고,
3. 처리 결과의 충족 요구 등이다.

Program의 준비에 있어 첫째 단계는 Program 되는 적용 업무의 특수한 조건과 계획의 완전한 분석이다. 절차를 포함해 이 분석은 개발된 순서도와 Block diagram에 의해 정상적으로 성취될 수 있는데, 왜냐하면 대부분의 자료처리 적용업무는 수많은 선택, 종류, 예외를 가지고 있기 때문이다.

이러한 가능성을 언어로만 표현하기는 곤란하다. 이리하여 시스템 분석자는 Layout형, Control panel diagram, Manpower planning chart 등을 포함하여 그림으로 표시하는 많은 Type의 사용 방법을 찾아 낸다. 여기에서 논의하는 두 가지 대표적인 것은 System flowchart와 Program flowchart이다.

Flowchart의 중요한 가치는, 그것만으로 대략적인 개요를 대충 훑어 볼 수 있다는 것이다. Flowchart는 논리적인 과정과 데이터의 흐름을 그림으로 나타내기 때문에 그들의 상호 관계에 따라서 광범위한 주요점과 많은 상세한 것들을 쉽게 알 수 있다.

이러한 순서와의 상호 관계는 Text의 상세한 조항으로부터 뽑아내는 것이 어렵고, Program에 대해서는 보조 자료 없이 결정하는 것은 거의 불가능한 일이다.

Flowcharting에 있어서, Symbol과 Word는 서로 관련되며, Text에 있어서 분명하게 나타나는 Identification과 Description은 도해된 순서에 놓이게 될 때 더욱 중요한 의미를 갖게 된다. 의미 있는 기호의 사용이 일치함으로써, 더욱 더 의사의

소통을 향상시키는 것이다. [Fig. 6-2]는 IBM Template 봉투를 복사한 것이다.

이 Template는 System flowchart와 Program flowchart의 두 가지 Flowchart 기호를 보여 주고 있다.

이러한 구분은 새로운 것은 아니다. System flowchart는 자료 처리 조직의 모든 부분에 걸친 데이터의 흐름을 나타내며, Program flowchart는 자료 처리 조직에 있는 특별한 *Program* 안에서 일어나는 자료의 흐름을 나타낸다.

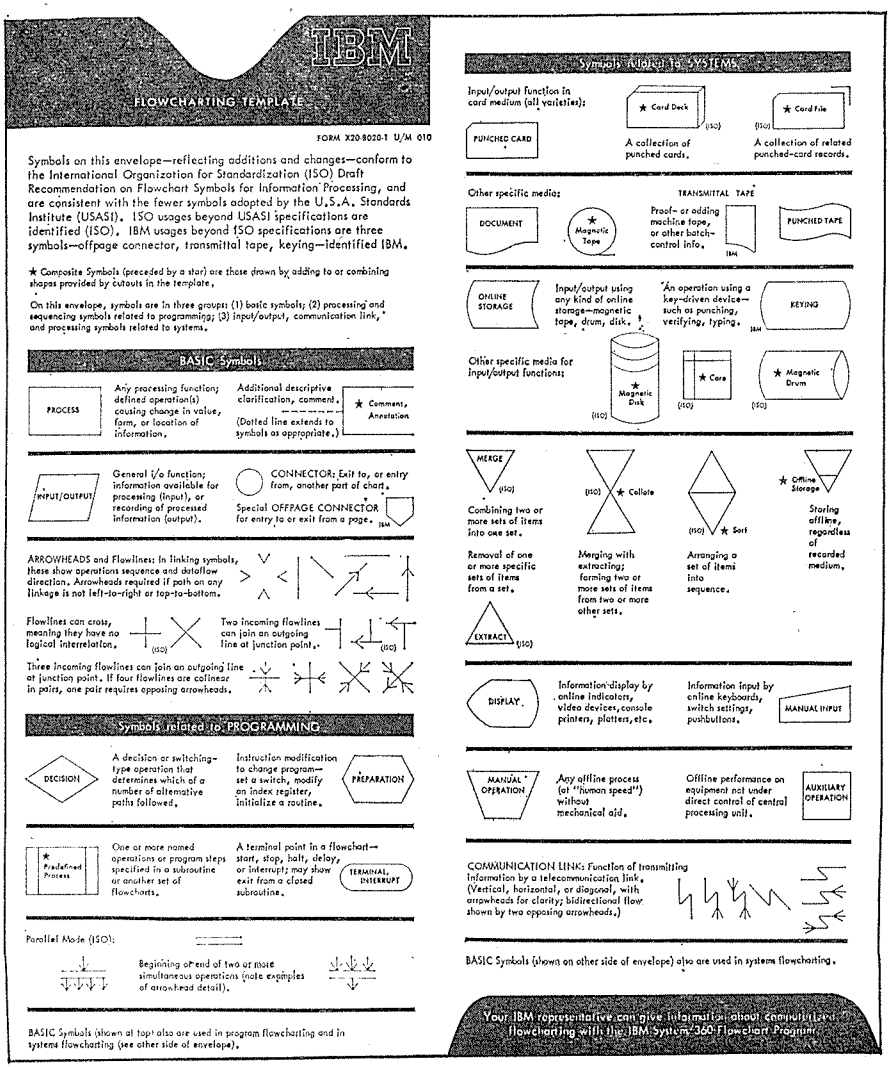

Fig. 6-2. Program and system flowchart symbols

System Flowchart

System flowchart는 원시 매체에 의해 제공된 데이터가 최종적인 매체에 의해 바뀌어지는 Application을 나타낸다. 중요한 점은 동반하는 매체와 그들이 통과되는 Work station이다. 바꾸어 말하면 Program flowchart에서의 중요한 점은 컴퓨터의 결정과 처리이다. Chart는 문제 해결의 상황, Coding에 사용된 Program logic과 처리 순서를 나타낸다.

기본적인 Program flowchart는 System flowchart로부터 발전되어 왔기 때문에 Program flowchart는 본질적으로 System flowchart보다 세밀하다. 그 밖에도 프로그램 순서도는 빈번히 일련의 보조순서도를 아주 세밀한 수준으로 'Exploding'하여 넣으므로 아주 명확하다.

System flowchart(시스템 순서도)는 Program flowchart(프로그램 순서도)보다 덜 형식적이고 그리기 쉽기 때문에 간편하다. 그것은 또한 훨씬 융통성이 있다.

부호의 많은 선택 뿐만 아니라 사용에 있어서도 넓은 범위가 적용된다. 전형적인 System flowchart에 사용되는 부호, 협정, 기법의 예는 해설로서 〔Fig. 6-3〕에서 보여 준다.

Program Flowchart

프로그램 순서도는 어떤 관점에서 볼 때 그 자체가 중요시될 가치를 가지고 있다. 이것은 아주 특수하며, 완전한 Routine과 프로그래밍의 독창적인 단계를 통합하기 때문이다. 프로그래머에게 프로그램 순서도는 일종의 다목적용 도구이다. 이것은 프로그램의 청사진이다. 프로그램을 개발할 때 프로그래머는 순서도를 업무의 모든 부분에 속속들이 사용한다. 즉, 그의 업무를 종이 위에 배열하고 연결하며, 문제와 논리 해법의 윤곽을 잡으며, 문제를 전체로서 조직적으로 다룬다. 프로그래머는 순서도를 그 자신의 참고서류 기타 기억하여 둘 수 있도록 계획을 단계적으로 세우기 위해 사용한다.

프로그램의 개발 단계에 있어서 순서도는 Logic을 설계하는 것 같은 여러 가지의 접근 방법을 가진 실험적인 수단으로서 사용된다. 프로그래머는 계획된 프로그램의 주기능을 나타내는 부호로서 시작한다. 프로그래머는 입출력 기능을 설명하기 위해 Block을 결합하고, Record의 확인과 선택을 위한 단계, 판단 기능으로 전체 Logic을 전개한다.

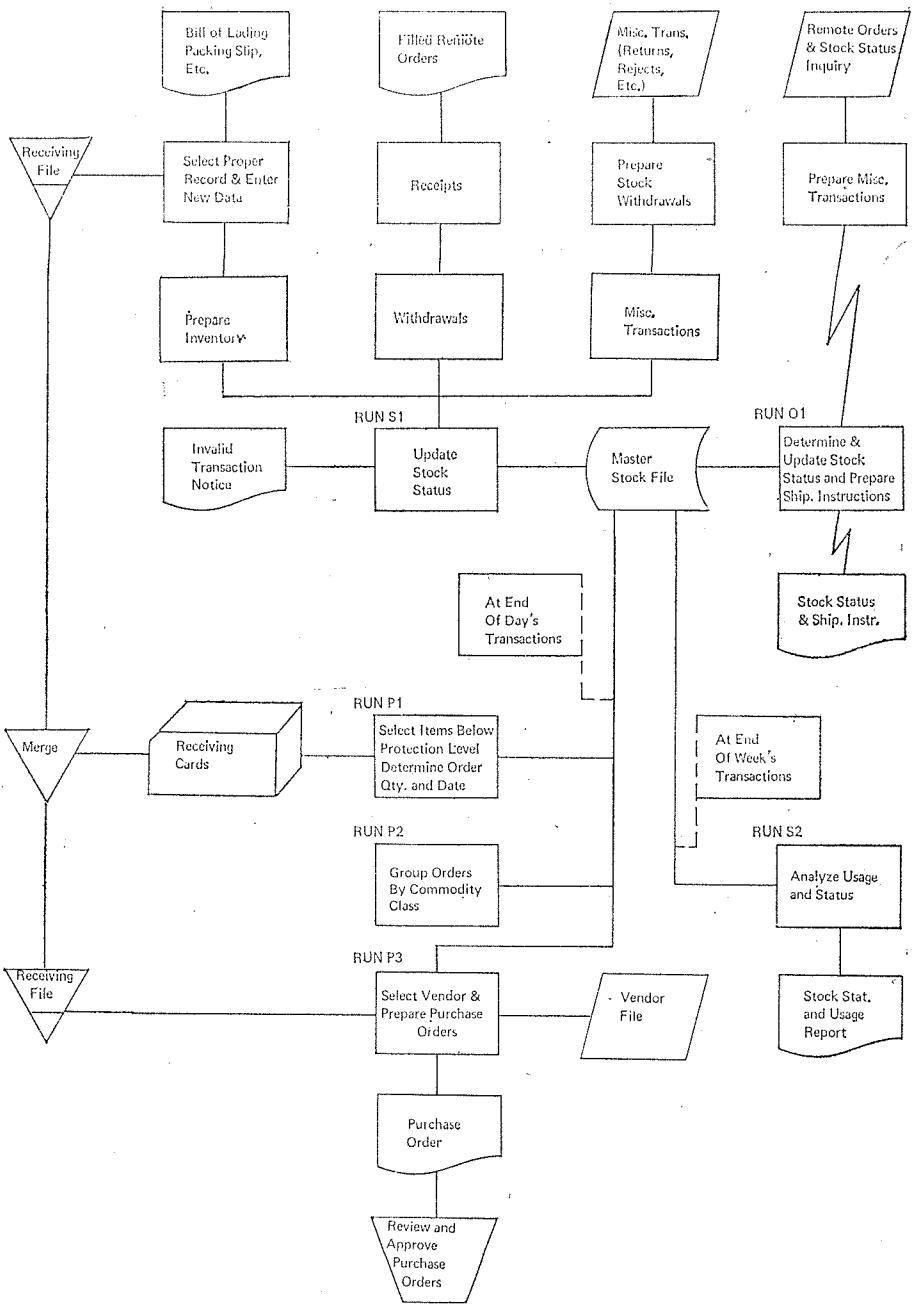

Fig. 6-3. Flowchart for an inventory control system

일단 프로그래머가 임시로 Mainline logic을 정하면, 그는 보통 큰 부분을 뽑아내며, 보조 Chart에 더욱 상세하게 큰 부분에 대해서 설명하게 된다. 이것은 일반적이고 모든 것을 포함하는 Map(전체도 : all-inclusive map)으로 시작하는 더욱 정교한 Map(상세도 : detailed map)의 세트의 그림과 같은 것이며, 더욱 더 정교해진 것을 보여주는 각 Map의 계속적인 Map(연속도 : succeeding map) 위에 그 그림의 부분을 다음에 설명한다.

이 기법은 Modular program flowcharting이라 불려진다. 이러한 기초 위에 완전한 Documentation을 위해 전형적인 File-maintenance routine은 아마 80개 이상의 순서도를 필요로 하게 될 것이다.

프로그래머가 그 과정(procedure)이 완전하다고 생각하였을 때, 그 순서도를 코우딩의 지침으로 사용한다. 이러한 단계에서 종종 Program logic은 Machine logic에 맞추도록 하기 위해 수정될 수 있으며, Chart는 재작성되거나 재조사될 수 있다. Testing, 설치, 미래 조작을 하는 동안에도 변경할 수 있다.

Flowcharting Worksheet

대부분의 프로그램 순서도가 아주 상세하기 때문에, 그것이 일관성 있고 합리적으로 조직된 형식으로 그려진다는 것은 큰 잇점이 있다. 이러한 '형식화(formalization)'는 이 목적을 위해 고안된 표준 형태로 가장 좋게 이루어졌다.

IBM Flowcharting worksheet form이 그 위에 전형적인 Chart를 첨부하여서 [Fig. 6-4]에 표시하였다. $11\times16\frac{1}{2}$ inch의 Worksheet는 모든 종류의 Flowchart에 사용될 수 있으나 이것은 특히 Program flowcharting에 유용하다.

본래 Worksheet는 알파벳과 숫자의 좌표를 가진 50개 Block의 배열이다. 10개의 가로줄은 꼭대기(A)부터 밑바닥(K)까지 문자로 나타난다. 5개의 세로줄은 좌측에서 우측으로 1부터 5까지의 숫자로 표시했다. 이러한 좌표는 기록과 전후 참조에 도움을 준다.

Worksheet는 50개 Position의 Guideline과 각 Position의 가로와 세로의 중간을 표시하는 Crossline을 가지고 있다. 이것은 Flowline과 직각을 이루며 부호를 각 Position의 중앙에 넣으며, 부호들 사이에 동일한 간격을 유지하는데 단지 도움·주고 있다. 그 결과로 깨끗하게 배열되고, 그렇게 혼합하지 않은 조밀한 Chart를 얻을 수 있다, Worksheet 자체는 사진 복사에서 Guideline이 나타나지 않도록 하기 위해 엷은 잉크로 인쇄되어 있다.

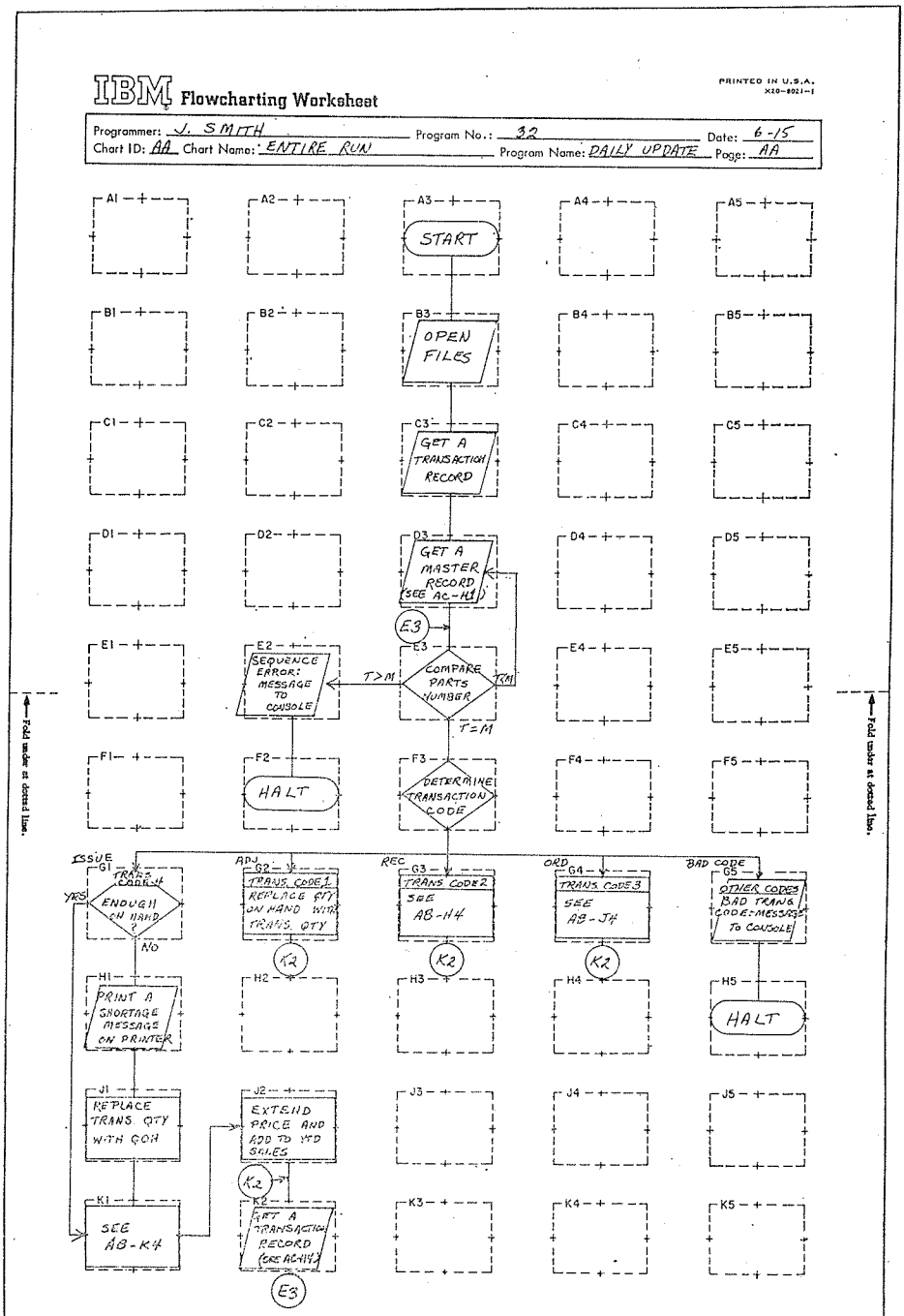

Fig. 6-4. First sheet of a sample program flowchart for an updating run of inventory records.

기타의 기법(Miscellaneous Technique)

프로그램 순서도의 작성에 있어서 좀 더 상세한 기법이 간략히 묘사되었고 설명되었다.

Cross-referencing

Cross-referencing은 프로그램 순서도와 Source language program(뒤에 설명한다)을 연결시킨다. 하나의 방법은 명령어를 그것의 Label이나 또는 그 명령어가 나타나는 Coding sheet의 페이지나 Line number에 의해 배치하는 것이다. Cross-reference는 〔Fig. 6-5〕에서 보이는 것과 같이 Symbol의 Upper left corner의 위에 배치 될 수 있다.

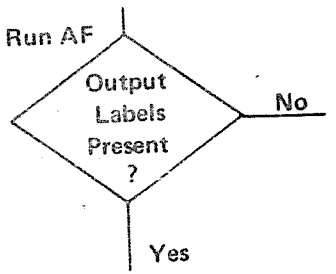

Fig. 6-5. Example of cross-referencing

Composite symbol

합성 기호(composite symbol)는 정의된 하나의 그룹이다. 비록 30여개의 기호가 봉투(Fig. 6-2) 위에 소개되었지만, 1/3정도 가량의 기호가 Template에 삭제되어 표시되었다. 첨가적인 기호는 Template에 추가되거나 삭제하여 표현되는 결합 형태로서 그려진다. 이러한 Composite symbol은 Cutout 아래에서 동일시 되며, 그들의 이름을 '*' 앞에 두고, 기호 그 자체는 봉투 위에 나타내진다.

예를 들어, Merge는 꺼꾸로 된 삼각형으로 표시된다. 가로 줄을 더하여 줌으로써 그 Composite symbol은 'Offline storage'를 의미하게 된다.

Sort는 국제 표준 협회의 표준에 맞추기 위해 그것의 중간 지점에 가로줄을 더할 필요가 있다. 결과적으로 이것은 별표가 붙은 Composite symbol로서 Template 위에 나타나 있다.

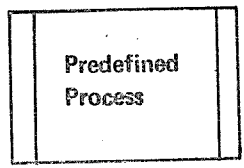

Fig. 6-6. Predefined process symbol

미리 결정된 처리 기호에 표현되는 Program unit의 더욱 상세한 순서도는 이것을 지정하기 위하여 사용된다(Fig. 6-6).

Decision technique

Decision technique는 몇 가지 방법으로 나타낼 수 있다. [Fig. 6-7]에서 보아라. 이러한 판단은 프로그램에 의해 다음에 수행될 작업을 결정한다(branch operation을 보라).

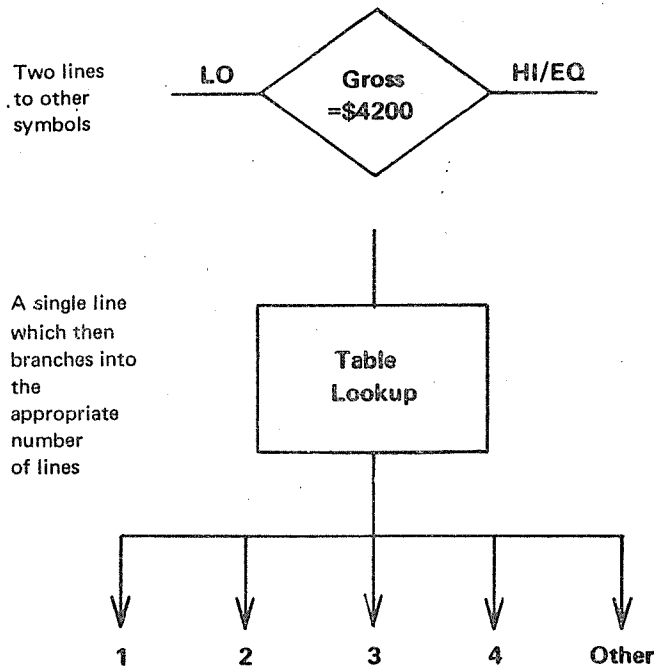

Fig. 6-7. Example of decision techniques

[Fig. 6-4](먼저 제시한 Flowcharting worksheet 위에 덧붙여 그린)에서 전형적인 프로그램 순서도의 실례를 보았다. 종합 재고 레코오드의 매일매일의 Updating

run을 위한 프로그램 순서도는 System flowchart에 기초를 두고 있다. System flowchart(Fig. 6-8)는 그 Updating이 그날의 Adjustment, Receipt, Order, Issue로부터 있음을 표시한다. 더하여, Shortage fnd-recorder listing이 작성된다. System은 Master file에 포함된 Addition이나 Deletion을 요구하지는 않는다.

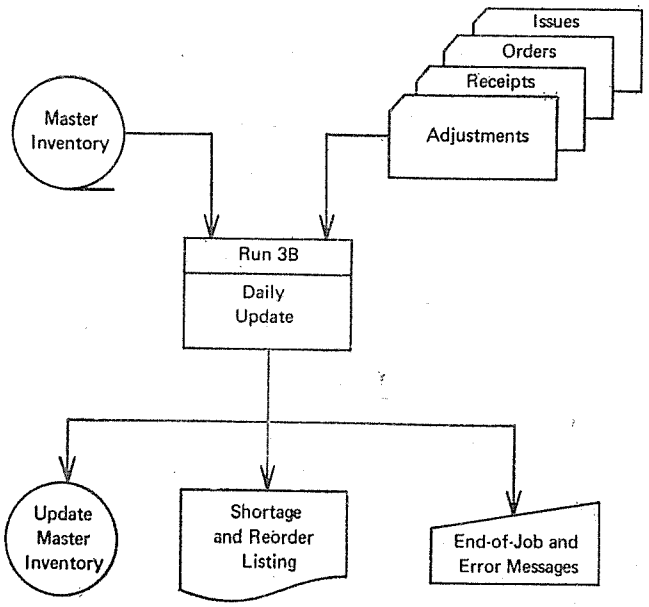

Fig. 6-8. System flowchart

Flowcharting by Computer

앞에서는 모두 손으로 Chart를 작성하는 Manual flowcharting에 대해서 다루었다. 그렇지만 컴퓨터에서 Flowchart에 대한 많은 진보가 이루어졌다. IBM System/360을 위한 IBM flowchart program은 컴퓨터가 세밀한 명령어로부터 보통 손으로 작성할 때와 같은 기호를 기계가 번역할 수 있도록 하는 완전한 Flowchart를 인쇄해 낼 수 있다. 이 프로그램을 S/360 Flowchart라 부른다.

기계화된 Flowcharting은 프로그램의 수정에서 Chart를 재작성 을 필요가 있을 때 종종 시간의 낭비가 뒤따르므로, 특히 Programming에서 유익하다. S/360 Flowchart로서 앞에서 설명한 방법으로 손으로 작성하고(코우드하는), 순서도가 자동적으로 작성될 수 있게 되었다. 일단 순서도가 작성되면 이것은 최소의 시간과 노력으로 수정되거나 재가동 될 수 있다.

[Fig. 6-4]에서의 Flowcharting worksheet는 이러한 실행에 훌륭하게 연결된다. 따라서 Worksheet가 제공하는 50개의 Grid은 그 자체가 기계화에 쓰여질 준비가 되어 있다.

6.4 Reading Data

컴퓨터 시스템에 들어가는 모든 데이터는 입력장치가 처음 읽어야 하며, 주기억장치로 전송된다, 각 입력장치는 각 기억 장소가 역시 Location address를 지정 받는 것과 같은 방법으로 그 번지로서 할당되기 위해 숫자를 지정 받는다.

데이터의 처리 절차는 정상적으로 한 개나 그 이상의 입력 매체에서 레코오드의 Entire file(operating system/360에서는 'data set'이라 부른다)과 관련이 있다. 이러한 File은 컴퓨터가 그들에 Access하므로 입력 장치로 공급되거나 보조 기억장치로부터 직접 읽혀 들어 간다. File로 부터 Record를 읽기 위해서, 프로그램에 있는 한 개가 그 이상의 명령어가 입력 장치에 작용하며, 이 Record는 주기억 장치에 배치된다.

이러한 입장에서 볼 때 들어오는 레코오드가 기억 장치의 어느 곳에 있는가는 명백하게 결정되어야 하며, 명령어는 이 레코오드를 미리 결정한 장소에 보내기 위해 기계를 감독해야만 한다. 또한 조종의 계획에 있어서, 연속적인 실행 단계에서 필요한 정보를 어디에서 찾을까를 언제나 알고 있는 것은 필요하다.

이러한 고려는 논리적이고 편리한 방식으로 특정한 목적을 위한 기억 공간의 배치를 포함한다.

예를 들면 특정한 Field나 양은 연산을 위해 사용 될 수 있다. 후에 사용될 명령어는 각 Record로부터 온 이 정보가 발견될 수 있는 기억 장치 안의 장소를 명기해야만 한다.

자료 처리 조직이 들어오는 레코오드의 배치와 필요에 따라 내장된 레코오드의 Fetching을 처리하기 위해서 프로그램의 Operating system을 가지는 때에는 Prolemb programmer는 그 Location에 세밀히 관계할 필요가 있다.

Reading operation은 아래의 독특한 기능을 수행한다.

1. 입력 장치는 사실상 읽기가 시작되기 전에 선택되어 준비를 갖는다. 장치선택은 프로그래머에 의해 결정된 특정한 File record에 Access하는 것이다. 이 장치는 정해진 코우드 Number나 Address의 지정으로 선택된다.

2. 읽기 명령어는 앞서 선택된 장치가 레코오드를 컴퓨터 기억 장치로 옮기는 작업을 수행하도록 해준다.

   이 레코오드는 이러한 목적을 위해 할당된 특정한 기억 영역에 배치되며, 다음의 처리에 이용된다. 많은 입력 영역은 몇개의 관련된 레코오드를 동시에 처리하기 위하여 지정될 수 있다. 예를 들면, Master record와 그것과 관련된 세부적인 업무 등이다.

3. 프로그램에 있어서, 읽기와 명령어의 차례는 File이 읽혀지는 순서를 결정하게 된다.

   다른 명령어는 Detail 대 Master, Detail 대 Detail등의 관계를 결정하기 위하여 뒤에 독립적인 File로부터 Record를 비교한다.

4. 동시에 기억 장치에 배치될 수 있는 Record의 수는 File의 구조와 처리될 Record의 형태 그리고 길이를 이용할 수 있는 기억 용량에 따르게 된다.