技術解說

마이크로 · 프로셋서應用 소프트웨어 開發硏究

安 秀 桔

서울大學校 工科大學 副敎授(工博)

1. 序 論

大量生産을 通하여 原價節減을 이루고 이로서 얻어진 低廉한 價格때문에 새로운 需要를 誘發하고 이러한 需要의 增加가 다시 大量生産을 通하여 原價의 減少를 가져오는 等 循環을 通하여 需要가 큰 製品만이 有利하고, 한편 特殊한 그리고 需要가 적은 製品은 高價하게 되는 大量生産의 時代가 되었다.

이와 함께 하나의 機械를 하나의 目的을 위해서 專用하는 專用機로부터 두개 以上의 目的을 위해서 簡單한 操作을 通하여 機能을 바꾸게 할 수 있는 汎用機로 옮겨가는 傾向을 볼 수 있다. microprocessor도 이러한 見地에서 여러 instruction에 該當되는 많은 機能을 發揮할 수 있는 汎用機의 範疇에 屬하는 것으로 볼 수 있다. 一般的인 뜻에 있어서의 汎用機는 그中 하나의 機能으로 바꿨을 때 그 機能으로 固定시켜 한 동안 使用을 하다가 必要에 따라 다시 다른 機能으로 바꾸는 것이 보통이지만 microprocessor(μP)는 한 instruction이 끝날 때 마다 새 instruction을 遂行하기 위해서 機能變換이 急速하게 일어나는 것을 要求하는 點이 다르다. 要求되는 機能을 二進法 數字로서 μP에 傳達하고 μP는 이와같이 外部에서 들어오는 二進數字의 各 bit가 „0"인가 „1"인가 하는 組合에 따라 機能을 바꿔간다. 勿論 하나의 具體的인 組合에 따라서 機能은 一意的으로 定해진다. 一例를 들어 8080A에서 "00000111"이라는 組合을 傳達하면 그때에만 8080A는 accumulator가 갖고 있는 數字를 2倍해 주는 機能을 갖게 되며, 어느 때에 있어서도 "00000111"이라는 instruction은 一意的으로 이 機能만을 주는 것이고 "00001111"에 依해서는 反對로 accuulator가 갖고있는 數字를 1/2로 해주는 機能을 갖는다. 結局 使用者가 어떠한 機能을 順次的으로 갖게 하고 싶은가에 따라 μP는 機能을 바꾸고 使用者는 이러한 自己의 뜻을 program이라는 "instruction의 모임을 通하여 μP에 傳達하는 것이다. μP로 부터 어떠한 結果를 끌어내는가 하는 것은 어떤 瞬間에 어떠한 機能을 要求하는가 하는 使用者의 窮理에 달려있다. 이 窮理에 따라 똑같은 μP로 부터 얻어내는 結果는 相當히 다를것이기 때문에 똑같은 hardware(機械)가 여러 目的에 쓰이는 汎用機가 되는 것이다. 이때 機能의 差는 오로지 使用者의 窮理에 달려있고 따라서 program이 달라지면 같은

hardware에서 全혀 다른 結果가 나오는 것이다. 이 program은 μP의 경우에는 Read Only Memory의 形態로서 같은 番地에 찾아가면 꼭 恒常 같은 二進法數字(instruction을 나타내는)을 發見할 수 있는 즉 „0"와 „1"이(마치 實線으로 連結되어 있는 것과 같이) 固定되어 있는 hardware(Prom)의 形態로 되어 있을 때가 많아서 이런 경우의 software(program)를 Hardened software라고도 한다. 따라서 機能이 서로 다른 機械들에게 實際 差를 나타내는 것은 이 ROM의 內容만이다. 이 ROM을 바꿔줌으로서 같은 機械로부터 여러 性能을 얻어내어서 μP가 汎用機械라고 말할 수 있는 것이며 따라서 需要가 集中되고 따라서 價格이 싸게 된다. 이 近來에 와서는 $ 10 以下의 경우가 許多하다. 이러한 μP에서 ROM內의 program의 要請에 따라 必要한 機能을 만들어내는 部分이 CPU인데 이 CPU에서만 機能變換이 觀察되는 것이 아니고 μP와 外部世界(peripherals) 사이를 結合시키는 interface에서도 역시 같은 汎用化 傾向이 보인다.

2. Universal Peripheral Interface

CPU에서 얻어 낸 結果를 外部世界에 聯關시켜 일을 하기 위해서 peripheral을 使用하는데 이 peripheral은 一例를 들어 teletype의 印字를 할 때의 경우와 같이 電磁石을 驅動해야 하는 等 큰 電力 또는 큰 誘導電壓을 取扱해야 할 때가 많고 종류도 廣범위 하기 때문에 이러한 peripheral과 μP사이에 interface를 두어야 했고 이 interface는 맞춤 양복과 같이 그 한 case를 위해서 μP를 잘 아는 高級技師가 interface 回路의 設計를 해야 하기 때문에 또 다시 專用機의 경우와 같은 原價上昇의 危機에 直面한 것이다. 여기에서 여러 경우에 쓸 수 있는 Universal peripheral interface가 登場하였다. Intel의 8255, 8251 Motorola의 6820, 6850, 6852, Zilog의 PIO, SIO, Fairchild의 3861 等이 이와같이 주어진 instruction에 따라 必要한 interface機能을 萬能的으로 提供하여 주는 programmable device로 우리는 우리가 必要로 하는 機能을 만들어주는 instruction만 찾아내면 配線하나 變更할 必要없이 願하는 機能을 發揮하여 준다. 卽 이들은 그곳에 入力하는 信號에 따라 機能이 달라지는 瞬時變換汎用機로서 機能變換用信號가 入力되면 이것을 register에 貯藏하여 놓고 그 內容이 바뀌기 前에는 항상 같은 機能을 提供하게 되어 있다. 機能決定을 위한 instruction이 아니고 이 入出力 interface device를 通하여 出入하는 眞正한 data를 考慮할 때 이 data는 8 bit로 構成되어 있는데 이미 列擧한 programmable interface 中에는 各各의 data가 時間間隙을 두고 順序대로 들어오는(따라서 入出力 端子가 한雙만 要求되는) serial interface USART, UART 等과 8個 端子가 있어서 한꺼번에 8 bits의 信號가 傳達되는 parallel interface PPI, PIO 等이 있다.

3. System과 信賴度

簡單한 部分回路가 모여서 좀더 複雜한 system을 構成하게 되는데 故障이 생겼을 때 그 故障場所의 把握의 容易를 위해서는, 또는 故障된 部分은 極히 一部일 때에도 그 當該 IC를 버려야 하고 收容素子의 數가 클수록 故障頻度 따라서 破棄頻度가 커지기 때문에 system 규모가 작은 것이 有利하다. 그러나 要求되는 機能의 複雜化에 따라 system 規模는 當然히 커지는 趨勢에 있다. 以前에도 system을 크게 만드는 것은 不可能한 것은 아니였지만 故障頻度가 너무 크면

제대로 使用할 수 없기 때문에 限定이 되지 않을 수가 없었다. system의 素子로서의 回路들(또는 機器들) 사이의 connection은 납땜 및 connector로서 이루어졌으나 信賴度가 높지 못하였다. IC의 發達에 따라 한 chip 속에 內藏할 수 있는 回路의 機能과 規模가 커졌고 따라서 connection의 많은 部分이 IC內에 吸收되어 버려서 信賴度가 높아졌고 信賴度가 높기 때문에 集積規模를 키울 수가 있다는 循環을 되풀이 하였다. 한편 IC의 內部에서는 모든 것이 작고 짧기때문에 高周波 및 高速動作에 適合하다는 利點과 더불어 system의 規模를 키울 수 있고 따라서 機能對 價格의 面에서 有利하여 需要도 刺戟하게 되었는데 이 低邊에는 한 bit의 誤差도 없이 動作을 한다는 信賴度의 向上에 그 功이 있다. μp의 경우에는 한 bit의 잘못이 한(結果의) error로 끝나는 專用機의 경우와 달리 한 bit의 잘못이 그 μP system의 機能의 設定의 잘못으로 되어서 全혀 그 以上 쓸모없게 되는 것이고 보면 μP system의 可能與否는 素子의 信賴度에 依存한 바가 크다고 하겠다. 專用機에서의 error는 計數의 誤差로 끝날때가 많지만 "programmable"이란 形容詞가 붙는 機器들은 error가 生기면 reset를 하는等 하여 機能設定을 다시 하여야 한다.

4. 端子의 共用과 Bus

專用機器로부터 汎用機器로 向한 傾向은 CPU를 포함한 μP用 LSI의 端子(pin)에서도 볼 수 있다. 例를 들어 8080A의 data bus들은 time multiplex로 되어 있어서 本來의 目的인 바 data의 傳達以外에 每 machine cycle 初期에 잠깐 그 machine cycle에서 行하게 될 일을 나타내는 一意的인 二進數字를 CPU 外部에 傳達하는데에도 쓰이고 있으며 data 傳達에 專用되어 있는것이 아니다. 反對로 address bus들은 address만을 위해서 專用되고 있다.

8085의 경우에는 address bus A_8~A_{15}는 address 專用 bus이지만 A_0~A_7은 data와 time multiplex 되어 있어서 address數字와 data를 時間的으로 分割하여 交代로 傳達하고 있다. 外部에서는 8bit latch 二組를 같은 bus에 連結하여두고 各各 알맞는 時間에 loading을 許容함으로서 address 數字와 data 數字를 分離할 수가 있다. 8155/8156, 8355, 8755 等 8085와 함께 使用하기 위해 開發한 memory IC들은 cpu bus에 連結만 해 놓으면(ALE단자도 連結) 이러한 節次가 自動的으로 行하여진다.

memory用 IC의 경우에 있어서도 必要한 端子의 數가 많기 때문에 28 pin 以上의 大型 package를 使用하게 될때가 많은데 address bus를 잇는 端子의 數는 4096 bit memory 의 경우는 4096 가지의 address 區分을 위해서 2^{12} 즉 address만을 위해서도 12個의 pin이 必要하다. Mostek 會社의 memory IC model 4096 또는 4116은 各各 4096 bit와 16.384 bit의 memory인데 두가지 모두다 16bit의 DIL package를 使用하고 있어서 單位面積의 프린트 基板上에 積載할 수 있는 package數 따라서 bit 數가 크다. 28 pin 以上의 pin을 갖는 memory package의 경우에는 길이와 함께 幅도 크기 때문에 프린트 基板單位面積當收容 memory bit數가 적은 데 16pin 짜리를 쓰면 越等히 有利할 것은 明白하다. 前記 model 4116等이 16pin package에 16kbit收容이 可能한 理由는 addres用 14bit가 專用으로 pin을 하나씩 使用하고 있는 것이 아니고 A_0로부터 A_6까지의 7個 pin을 時間的으로 두번에 나누어 使用하여 IC 內部에서 14bit의 latch에 收容하기

때문에 可能하게 된 것이다. 8708 memory는 하나의 address가 並列로 8個의 data bit를 한꺼번에 記憶하기 때문에 全體로 8192 bit의 收容能力을 가졌지만 1024個의 address만 갖고 있어서 1024×8와 같이 表示하는데 $1024=2^{10}$이니까 address는 A_0로부터 A_9까지 10個의 pin을 쓰고 있다. 容量을 더 크게 할려면 例를 들어 1024×8 bit인 8708로서 4096×8를 만들고 싶으면 4個의 8708(各各 chipA, chipB, chipC, chipD,로 呼稱) package를 20番 pin $\overline{CS}$/WE 以外의 pin을 모두 다 並列로 結線하여 使用하면 된다. 이中에서 使用하고저 하는 chip의 $\overline{CS}$/WE pin만을 ground 電位로 내려주면 그 chip만을 使用하게 된다. chipA에 0000 0000 0000로부터 0011 1111 1111(MSB先行)까지를 chipA에 수용하기로 한다. 이곳 address의 記法은 二進法으로 하였는데 4자리씩 모아서 16進法으로 記錄하면 chipA에는 $(000)_{16}$로 부터 $(3FF)_{16}$가 收容된다는 말이 된다. (1111=F, 0011=3) μp取扱者들은 16代身에 H字를 써서 0H로부터 3FFH(先行 zero는 16進法 數字가 alphabet로 始作 되었을때 만)까지라고 記한다. 다음 chipB, chipC, 및 chip$D$는 各各 400H로부터 7FFH까지, 800H로부터 0BFFH까지, 0COOH로부터 0FFFH까지(先行 zero는 16進法 數字가 Alphabet로 始作되었을 때만)를 收容하게 된다. 이들은 두번째 16進法 數字 以後는 똑같이 움직이고 처음 것만 달라서 chipA는 C, chipB는 D等으로 된다. 즉 下位 binary address 10 bit는 같이 움직이지만 11bi$_t$와 12bit는 chipA는 各各 00, chipB는 各各 01, chipC는 10, chipD는 11가 되는 것이니까 A_{10} bus와 A_{11} bus에 decoder를 連結하여 該當되는 信號에서만 各各의 memory chip의 $\overline{CS}$/WE pin이 交代로 ground 電位가 되게 해 주어야 한다. (그림 1) memory IC를 많이 使用할 경우 address bus를 共用하나 $\overline{CS}$(chip select)만은 모두 다 „1"로 있고 그中 하나의 選擇된 memory IC의 $\overline{CS}$만이 „0"으로 되어야 한다.

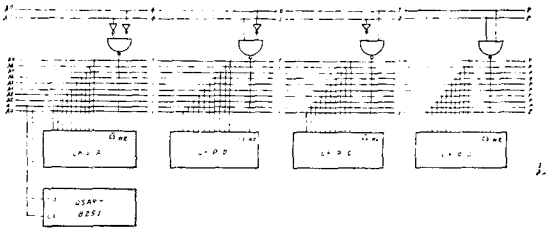


그림 1. Memory chip의 選擇回路

이와 같이 모든 memory IC가 bus線을 共用하고 있기 때문에 bus(合乘, 共用)라고 부른다. Bus가 아니였더라면 memory cell(4116에서는 16,384個)와 같은 個數의 配線이 必要할뻔 한 것이다.

5. μP의 使用上注意

여러 會社에서 一長一短이 있는 難兄難弟格인 製品이 導入되고 있으나 이 글에서는 資料入手와 筆者의 保有機器等의 사정으로 8080A系列을 中心으로 다루겠다. 이 系列은 Intel나 NEC의 8080A, AMD의 9080, zilog의 Z80, Intel의 8085等이 該當되고 其他 많은 會社에서 second sourcing을 하고 있기 때문에 이를 主軸으로 삼고 說明을 하겠다. 그러나 入門은 다른 册子들에 미루고 中間 入門으로서 좀더 具體的인 使用上의 注意와 program의 實例 그리고 그러한 program을 다른 結線을 가진 micro computer(μC라 略記)에 使用할 때 行하여져야 할 modification을

說明하겠다.

① Bus driving과 Tri-state

Address bus와 data bus에 連結된 CPU出力 部分은 그림 2와 같은 構造이기 때문에 飽和된

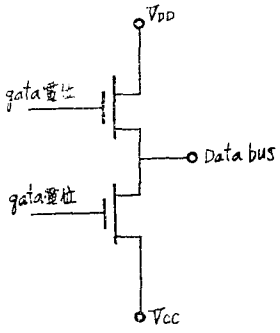


그림 2. MOS totempole出力 BUS 驅動回路

MOS transistor를 通하여 ground나 V_{DD}電壓線에 直結되어 있기 마련이어서 外部에서 이 bus의 電壓을 最終端 MOS transistor의 gate入力에 逆行하여 „1"인것을 „0"으로 또는 „0"인것을 „1"로 하려고 하면 飽和하고 있는 MOS transistor를 破壞시키게 될 다름이지 目的을 達成하지 못한다. 따라서 bus의 電位는 CPU에게 全的으로 一任하되 或時 bus의 電位를 支配하기를 願할때에는 CPU에 hold instruction을 보내고 이에 따라서 CPU가 하던일을 完了하고 bus에 連結되어 있는 바 CPU 內部 最終端 totem pole MOS transistor의 各各의 gate에 channels을 off해주는 信號를 보내어 어느 편도 飽和가 안되고 도리어 cut-off가 되게 하여줌으로 해서 bus가 „0", „1"間 어떠한 電位가 되어도 지장이 없게 되게 만든 다음 hold acknowledge를 해 올때까지 기다려서 bus電位를 支配하여야 한다. 이와 같이 totempole의 兩 transistor가 모두 off 된 狀態를 tri-state라고 하며 同一한 bus를 여러 回路가 支配하기를 願할때 各各의 回路는 自己

文 字	7Bit ASCII	文 字	7Bit ASCII	文 字	7Bit ASCII	文 字	7Bit ASCII
空白	20	3	33	E	45	X	58
!	21	4	34	F	46	Y	59
"	22	5	35	G	47	Z	5A
#	23	6	36	H	48	[	5B
$	24	7	37	I	49	/	5C
%	25	8	38	J	4A	]	5D
&	26	9	39	K	4B	↑	5E
▲	27	:	3A	L	4C	←	5F
(	28	;	3B	M	4D		
)	29	<	3C	N	4E		
*	2A	=	3D	O	4F	CR	0D
+	2B	>	3E	P	50	LF	0A
,	2C	?	3F	Q	51	BEL	07
—	2D	@	40	R	52	NUL	00
.	2E	A	41	S	53	RUB	7F
/	2F	B	42	T	54		
0	30	C	43	U	55		
1	31	D	44	V	56		
2	32			W	57		

그림 3. ASCII code 表(7bit)

가 支配 할 차례가 아닐때는 이 tri-state가 되어 있어야 하며 그렇지 못하고 出力들이 얕은 impedance를 나타내면 驅動하는 transistor에서 많은 電流를 供給해야 하고 上述한 바와 같이 transistor 그리고 따라서 그 transistor를 收容하고 있는 IC 全體를 破壞하는 수가 있다. CPU에 hold를 要求하고 CPU가 이 tri-state로 避身한다음 入出力機械와 memory間에 bus를 通한 data의 授受를 行하는 것을 DMA(direct memory access)라고 하는데 普通 data를 CPU의 accumulator를 通하여 일일이 보낸 경우에 比해서 越等히 速度가 빠르다, 이때 CPU는 全혀 介入하지 않고 bus가 나타내는 address가 順次的으로 增加하면서 入出力機器와 memory의 同一 address에 있는 data끼리가 移動을 한다.

② Teletype와 8251

μP와 가장 많이 쓰이는 入出力機械로서 Teletype會社의 ASR-33 model teletype가 있는데 여기에서 나오는 binary 信號는 7bit로서 그림 3과 같다. 16進法으로 表現되어 있는데 이때 上位數字는 二進數字 3bit로서 構成되어 있어서 7을 넘지 않는 것을 볼 수 있다. 한편 거기에서 나온 信號는 „0"가 start bit의 役割로서 先行하고 LSB로 부터 始作하여 MSB로 끝난다음 parity bit가 다음에 오며 마지막으로 „1"이 stop bit로서 맨 마지막으로 오는데 그 길이는 1bit, 1bit¥ 또는 2bit로서 8251 USART(universal synchronous/asynchronous receiver/transmitter)에 보내는 command 命令에 依해 이中 하나를 擇해줄 수가 있다. 8251은 入力으로 使用하건 出力으로 使用하건 一旦 두 차례 出力 instruction 11010011(D3로 表示)을 通하여 mode型式과 Command內容을 通해서 우리가 願하는 8251의 機能이 무엇인가를 通告하고 나서 진짜 data의 入力이나 出力을 하게 된다. mode나 command를 보낼 때에는 data의 경우와 달리 C/$\bar{D}$端子(12番 pin)에 „1" 을 加하여야 한다. (data를 보낼때는 „0") 12番 pin은 普通 實線으로서 address bus中 LSB인 A_0에 連結되어 있기 때문에 出力 instruction(D3)다음 byte에 오는 數字가 반드시 奇數(마지막 bit „1")가 된다 (data 入出力때는 偶數) mode 設定에서 보내고 받는 data가 synchronous인가 asynchronous인가를 通告하여 주고 asynchronous의 경우에는 R×C(25番 pin)와 T×C(9番 pin)에 보내는 USART data 傳送用 clock이 teletype 發着信號의 baud rate(110 baud)의 1倍, 16倍, 64倍中 어떤 것인가, code의 길이가 5bit(체신부 使用 Murray 5unit code)인가 6bit인가 또는 ASR-33의 경우와 같이 7bit인가 parity를 check하는가 안하는가 한다면 奇數를 „1"로 하는가 偶數를 „1"로 하는가 그리고 stop bit의 數等을 通告하여 USART가 必要한 準備(性能調整)를 할 수 있게 해준다. command는 이것에 추가하여 送信을 할 것인가 受信을 할 것인가 等을 通告하기 위한 準備 過程인데 mode 設定이 된 다음 처음 C/$\bar{D}$=„1"은 command로 看做되고 data入出力은 C/$\bar{D}$=„0"로 使用하되 다시 C/$\bar{D}$=„1"이 들어오면 command變更으로 看做한다. 즉 命令만 가지고 機能을 번번히 바꿔 줄 수가 있다. mode를 變更하기를 願할 때에는 역시 C/$\bar{D}$=„1"로 出力을 要求하되(勿論出力 되지 않음) 이때 accumulator에 收容시켜 놓은 data의 D_6(7番째) bit가 „1"이어야 한다. D_6 bit가 internal reset를 맡고 있어서 이곳에 „1"이 들어오면 設定된 mode를 클러 버리기 때문에 mode를 새로운 命令으로 바꿔줄 수 있게 된다. 이것은 CPU reset 때에도 마찬가지이기 때문에 CP∩

reset 다음에는 반드시 다시 mode와 command 設定이 뒤따라야 한다.

8251 動作過程에서 error가 있거나 하면 USART內部 status register에 記錄이 되기 때문에 이러한 status를 읽어 낼려면 入力 instructron (DB)를 使用하되(入力機器에서 진짜 data가 들어오기를 期待하지 말고) 역시 C/$\bar{D}$="1"로 하여야 한다(C/$\bar{D}$="0"은 data를 읽어들일 때).

③ stack address의 指定과 stack poituer

電算處理란 subroutine을 빼놓고 생각할 수가 없는데 이와같이 어떤 特定의 處理를 위해서 만들어 놓은 subroutine들을 活用하기 위해서는 그 subroutine으로 떠나기 直前의 出發地點의 address의 다음 address(卽 subroutine에서 돌아오게 될 address)를 어디엔가 남겨두며 또한 이러한 subroutine으로 가있는 中에 喪失되게 할 수 없는 data들도 貯藏하여 두어야 하겠는데 이 目的으로 使用되는 것이 stack이다. stack을 CPU內에 갖고 있는 경우도 있지만 一般的으로 memory의 一部를 使用할 경우가 많다. 이 경우에 다음 stack으로 使用될 memory의 address를 貯藏하고 있는 16 bit register가 있는데 이는 길 案內板의 役割을 하고 있기 때문에 stack pointer라고 불리우고 stack가 많이 占有되어 있는가 아닌가에 따라서 表示하고 있는 數字가 적어지기도 하고 커지기도 한다. μC를 start시킬 때에는 subroutine의 도움을 받는 범위에서는 반드시 stack pointer의 內容을 우리가 指定하는 番地로서 채워 놓아야 한다. 이것은 LXI SP XXXXH等으로 直接 넣어 줄 수도 있다.

6. μC program의 實例

microcomputer의 software는 system의 目的 및 요구되는 software의 機能과 性能에 따라서 設計된다. microcomputer 응용 전자제품의 개발은 software의 개발이 불가피하므로 software의 作成費와 hardware의 가격의 적정선에서 software와 hardware의 분담이 결정된다. software의 비용은 생산되는 제품수가 많을수록 내려가며, hardware의 부분이 간소화 될수록 system 전체의 cost는 감소하고 부품수의 감소에 의해 信賴性은 向上된다. 그러므로 software의 개발은 앞으로의 전자제품 생산에 매우 중요한 부분이 되었으며, software를 개발하기 위해서는 debug 및 test를 행할 수 있는 support system이 필수요건이 되겠다.

本 program 部分에서는, 8080 CPU의 PANT OS model 10 microcomputer와 TTY(ASR33)와 EPROM WRITER(MP7-03)를 interface 시킨 system과 JAPAN MICROCOMPUTER의 Intel board debugger S-80 等을 써서 紙 tape上의 program을 load시키는 loader program과 PROM의 programming, memory 내용의 listing 等 관련 program과 일반적인 수치연산 program들을 보였다.

① system initalize와 I/O device의 入出力 program, microcomputer system에서는 power-on時나, reset, 또는 정상 동작사이에서도 必要에 따라 행할 수 있는 initialize가 있어야 한다, 이 部分은 system을 정상으로 동작시키기 위해 system 設計에 따라 완벽하게 검토되어 있어야 한다. I/O device의 제어, register類의 reset 및 초기설정, RAM 내용의 소거等 필요에 따라 여러가지의 program을 포함하게 되는데, 이 부분에 error가 생기면 system이 이상동작을 하게 되어 언제까지 기다려도 정상동작으로 돌아오지 않을 수도 있다. 다음에 보이는 program에서는 stack pointer를 초기 설정한 후, TTY와

interface시키는 USART(8251)의 제어와 정상 동작임을 사용자에게 알리는 부호의 出力, 그리고 panel switch로 부터 실행 address 入力을 읽어 program counter에 보내며, 이때 panel control 기능에 의해 wait 상태에 들어가면 panel上의 start switch의 入力을 넣음으로써 main program을 실행하도록 되어 있다. 그러나 microcomputer system이 어떤 目的만을 위해 사용되면 initialize에서 바로 main program으로 jump할 수도 있다. 以下 그림 4와 그림 5에 flow chart와 program을 보인다.

다음으로 TTY의 入出力 program을 보자. TTY를 USART를 통해 interface시켰을 때 CPU는 USART의 Status를 읽을 필요가 있다. 즉, TTY로 文字를 出力했을 경우는 status中에서 T×RDY bit를 조사하여 data가 모두 전송되었는가를 보고나서 전송이 완료되었을 때 다음 step으로 넘어가야 한다. 또 TTY로 부터 data를 받을 경우에도 R×RDY bit를 조사하여 入力이 완료되었을 때 data를 읽어야 한다.

```
TTY의 tape reader구동 relay접속번지        : ϕ4H
TTY로의 data出力번지                        :  ϕH
USIRT(8251)의 control번지                   : ϕ1H
   〃          status入力번지                : ϕ3H
Panel control上의 下位 address入力번지      :ϕ2FH
   〃              上位   〃      〃         :ϕ2EH
〈Initialize program〉
INITI : LXI   SP, 2FFFH : SP를 RAM번지 2FFFH로 설정
        MVI   A, ϕH     : RDR disable
        OUT   ϕ4H
        MVI   A,ϕFAH    : Mode instruction
        OUT   ϕ1H
        MVI   A,ϕ32H    : Command instruction
        OUT   ϕ1H
        MVI   B,ϕ2AH    : ※부호를 出力
        CALL  TYOUT,
        MVI   B,ϕDH     : CR을 出力
        CALL  TYOUT
        MVI   B,ϕAH     : LF를 出力
        CALL  TYOUT
        IN    ϕ2FH      : 下位 address入力
        MOV   L,A
        IN    ϕ2EH      : 上位 address入力
        MOV   H,A
        PCHL
```

그림 5. Initialization program

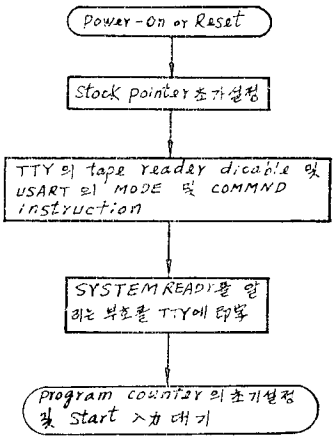
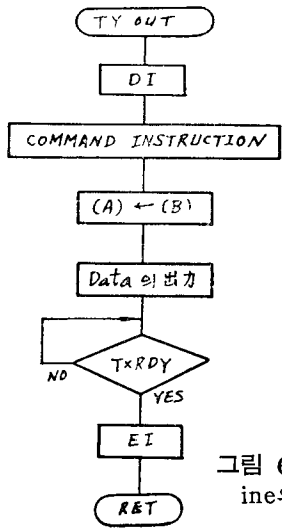
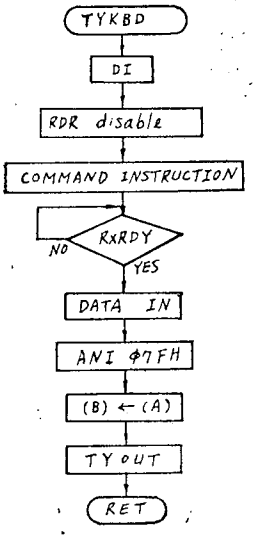


그림 4. TTY를 I/O device로 한 initialze routine flow chart



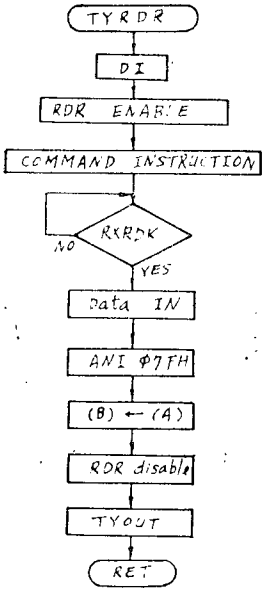

```
TYOUT : DI
        MVI A,ϕ33H
        OUT ϕ1H
        MOV A,B
        OUT ϕH
WAIIT1: IN  ϕ3H   } TxRDY?
        ANI ϕ1H   }
        cpi ϕ1H   }
        JNZ WAIT1 }
        EI
        RET
```

그림 6. TTY로의 出力 routine의 flow chart와 program




```
TYKBD : DI
        MVI A,φH
        OUT φ4H
        MVI A,37H
        OUT φ1H
WAIT2 : IN  φ3H
        ANI φ2H
        CPI φ2H
        JNZ WAIT2
        IN  φH
        ANI φ7FH : BIT 7 ±H
        MOV B,A
        CALL TYOUT
        RET
```

그림 7. TTY의 keyboard로 부터의 入力




```
TYRDR : DI
        MVI A,φ1H
        OUT φ4H
        MVI A,φ37H
        OUT φ1H
WAIT3 : IN  φ3H
        ANI φ2H
        CPI φ2H
        JNZ WAIT3
        IN  φH
        ANI φ7FH : BIT 7 ±H
        MOV B,A
        MOV A,φH
        OUT φ4H
        CALL TYOUT
        RET
```

그림 8. TTY의 tape reader로 부터의 入力 routine

그림 6,7,8에 TTY와 USART(8251)를 interface시켰을 때의 data의 入出力 subroutine들을 보인다. 이때의 data들은 7bit의 ASCII code 제 3도로 이루어져 있어야 한다. 그리고 이 subroutine들은 B register를 入出力 data의 기억 장소로 쓰고 있다.

② Loader

Loader란 미리 정해진 format에 따라 紙tape 等에 기록된 program을 memory에 load시키는 機能을 가진 program이다. 여기서는 memory에 load되는 data를 나타내는 方式에 따라 구분된, BNPF型, hexadecinal型, binary型의 3가지의 一般的인 loader에 대해서 말하겠다.

BNPF型은, 1 byte의 data를 B文字(ASCII code로 42H)로 시작하여 data의 각 bit가 „1"이면 P文字(5φH), "0"이면 N文字(4EH)로 나타내며 F文字(46H)로 끝맺어, 총 10 byte의 data로 변환시켜 object中의 data로 쓰고 있다.

Hexadecimal型은, 1 byte의 data를 2개의 16진수로 하여, 그것을 ASCII code化하여 object中의 data로 사용하고 있다.

Binary型은, 1 byte의 data를, 그대로 object中의 data로 사용하고 있다.

以上의 3가지 型의 data表示의 例를 그림 9에 보인다. 이러한 object言語의 data型 中에서 어느 것을 택할 것인가 하는 점에서 보면, 우선 data error check의 精度가 問題가 되며, 그 다음으로는 object 紙tape의 소모길이와 비교적 간편하면서도 어느 정도 많은 정보를 실을 수 있는지가 문제가 되겠다. error check의 문제에 있어서는 BNPF型, hexadecimal型, binary型의 순서로 精度가 낮아지고, 두 세번째의 문제에서는 역의 순서로 효율이 낮아짐은 당연하겠다. 그러나 이것은 data 入出力裝置 및 情報

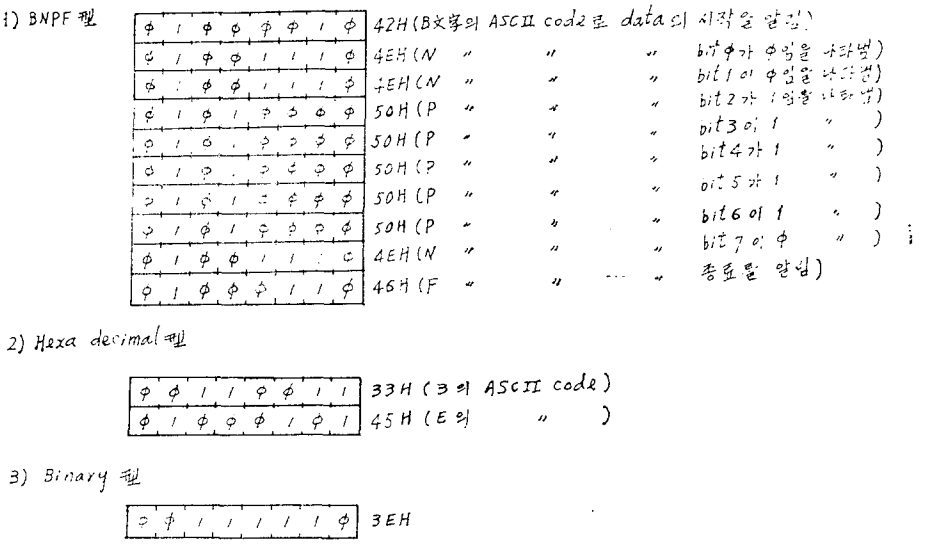


그림 9. data 3EH의 3가지 型으로의 表示

媒體의 信賴性도 감안해야 한다.

Ⓐ BNPF型 loader.

이 型에서는 1 byte의 data가 10 byte씩으로 변환되기 때문에 load의 先頭番地나 data의 총 byte수 等을 포함시키기 어려우므로 紙tape에는 dapa만을 싣고 실행번지는 TTY의 keyboard를 통하여 入力시키는 bootstrap型式을 취하였다. 1 byte의 data를 나타내는 10개의 文字를 word field라고 하며, 각각의 word field사이에는 TTY의 printer로 monitor 할 수 있도록 Carriage Return, Line Feed 및 번지수를 넣을 수 있게 되어있다. word field中에는 규정된 문자 이외에는 허용하지 않으며, tape 작성中 오자가 생겼을 경우, 해당되는 word field의 B문자부터 다시 punch하면 된다. 만약 error가 발견되기 전에 종료문자가 들어 갔을 때는 해당 word field 전체를 RUB out시켜야 한다. 以上과 같은 紙 tape가 만들어 졌을 경우, 第11圖에 보이는 program을 실행시켰을 때의 과정을 알아 보자. 그림 10에 TTY로 monitor된 一例를 보였다. 本 program은 command search로 시작되는데, command로써는, T,R,C,E의 4가지가 있다. 그리고 入力번지는 bank 숫자와 bank內의 下位번지를 3자리의 십진수로 나타낸 선두번지와 종료번지를 필요로 하고 있다. 각 bank는 000_{10}~255_{10}까지의 256 byte를 단위로 하여 구성되어 있으며, loader는 1 bank分씩의 data를 처리하게 되어 있다. 각 command의 의미는 다음과 같다.

a) T-command : Keyboard로 부터 入力되는 번지의 RAM에 tape reader로 부터 data를 읽어 기억시킨다.

b) R-command : T-command실행中 data에서 format error가 생겼을 경우 다시 새로운 data를 읽

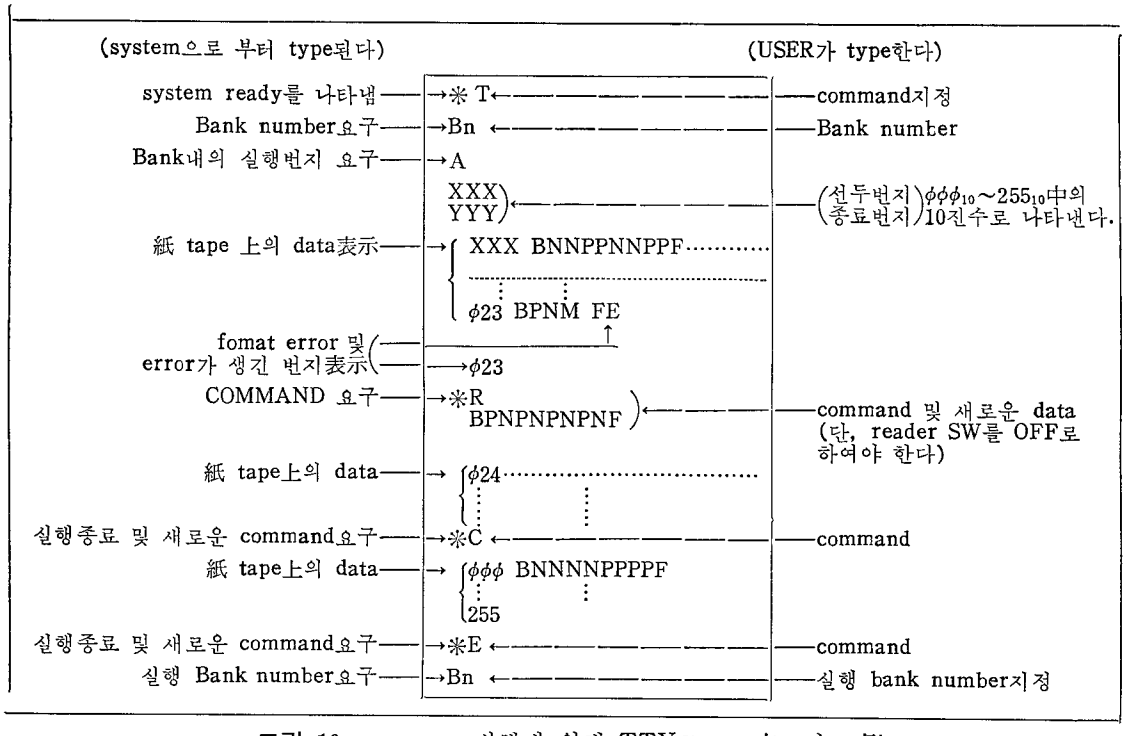


그림 10. program실행에 의해 TTY로 monitor된 一例

어 들인다. (이때 주의할 것은 R문자를 type하기에 앞서 tape reader의 SW를 OFF로 하여야 하며 keyboard로 R 문자에 계속하여 새로운 data를 넣고 SW를 ON으로 하여야 한다.)

c) C-command : T-command에 의해 지정된 bank分을 실행하였을 때 다음 bank에 계속해서 紙 tape의 data를 읽어 들인다.

d) E-command : RAM에 program이 load된 후 program을 실행시키는 명령이다. 이때는 bank숫자만을 받아 들이고 下位번지는 00H로 하여 program을 실행한다.

以上 command로써 program을 수행하게 되며 本 program에서는 RAM의 上位번지를 10H와 bank 숫자의 合으로 하고 있다. 이것은 system上에서의 RAM의 번지와 용량에 의해 변화 시킬 수 있으나, 本 program에서는 BNPF型의 loader가 PROM WRITER를 제어하기 위한 system과 같이 精度가 크게 요구될 때 주로 사용되므로 RAM의 번지와 용량을 1000H~13FFH로 한정하여 小容量에 적합하도록 되어 있다. 그리고 13F0H~13FFH까지는 loader program이 점유하고 있다. 이것을 다음에 보인다.

13F0H: 入力된 bank숫자가 ASCII code로 기억된다.

13F1H
13F2H
13F3H
: 入力된 선두번지의 각 자릿수가 ASCII code로 차례로 기억된다.

13F4H, 13F5H, 13F6H : 入力된 종료번지가 ASCII code로 기억된다.

13FAH: 入力되는 data를 合成하는 장소이며, 完了되면 실행번지로 合成된 data를 보내고 새로운 data를 合成한다.

13FCH: 10H와 bank숫자의 合을 기억한다.

13FDH: 선두번지가 1byte의 binary로 변환되어 기억되며, 入力되어 생성된 data가 RAM에 load될 때마다 1씩 증가한다.

13FEH: 종료번지가 binary로 변환되어 기억된다.

13FFH: Data의 bit수를 count하는 장소이다.

```
  BNPF型 LOADER
START : CALL CRLF
        MVI B,φ2AH    :※부호를 TTY에 出力
        CALL TYOUT
        CALL TYKBD    :COMMAND 入力
        MVI A,φ54H    :T?
        CMP B
        JZ TAPIN
        MVI A,φ52H    :R?
        CMP B
        JZ REDIN
        MVI A,φ43H    :C?
        CMP B
        JZ CONTI
        MVI A,φ45H    :E?
        CMP B
        JZ EXCUT
        CALL ERROR    :COMMAND error
        JMP START
————★————★————★————
TAPIN: CALL ENTRA     :실행 대상번지 入力
REDIN: CALL DATIN     :data入力
        RAR
        JC START      :carry가 있으면 start로 분기
        MVI L,φFAH    :data를 RAM의 지정번지에 load
        MOV C,M
        CALL SETMA
        MOV M,C
        CALL CHECK    :END?
        JZ START      :Zero면 END
        JMP REDIN     :다음 data 入力
CONTI:  LXI H,13FCH   :Bank숫자 INR
        INR M
        INX H
        MVI M,φH      :선두 下位번지로 φφH로 set
        INX H
        MVI M,φFFH    :종료번지를 FFH로 set
        JMP REDIN     :Data入力
EXCUT:  LXI H,13FφH   :Bank入力
        CALL ADRSH
        CALL CRLF
        MOV A,M
        SUI φ3φH
        CPI φ4H
        JNC START     :Bank숫자가 4이상이면 START로 분기
        ADI φ1φH      :  〃  3이하면 program실행
        XRA A
        PCHL
————★————★————★————
        SUBROUTINE
————★————★————★————
CRLF:   MVI B,φDH     :CR,LF를 TTY에 出力
        CALL TYOUT
        MVI B,φAH
        CALL TKOUT
        RET
ERROR:  MVI B,φ3H     :?를 TTY에 出力
        CALL TYOUT
        RET
ENTRA:  LXI H,13FφH   :Bank숫자 入力
ENTRH:  CALL ADRSH
```

```
ENTRL: CALL ADI      :선두 및 종료번지 入力
       CALL CRLF
       MVI L,φE6H    :ASCII code로 된 data
                      를 binary로 변환하여
                      기억시킨다.
       CALL BCDBI
       MOV C,B
       DCX H
       CALL BCDBI
       DCX H
       MOV A,M
       SUI φ3φH
       ADI φ1φH
       MVI L,φFCH
       MOV M,A
       INX H
       MOV M,B
       INX H
       MOV M,C
       RET
ADRSH: CALL CRLF     :CR,LF,B문자를 TTY
                      ⋮ 出力한 후 bank숫
                      자를 받아들임.
       MVI B,φ42H
       CALL TYOUT
       CALL TYKBD
       MOV M,B
       RET
ADRSL: CALL CRLF     :CR,LF,A문자, CR,LF
                      를 出力한 후 선두 및
                      종료번지를 받아들임.
       MVI B,φ41H
       CALL TYOUT
AD1:   CALL CRLF
       MVI C,φFDH
AD2:   CALL TYKBD
       INX H
       MOV M,B
       INR C
       JNZ AD2
       RET
BCDBI: MOV A,M       :ASCII code로 된 10진
                      수를 binary숫자로 변
                      환시킨다.
       SUI φ3φH
       MOV B,A
       DCX H
       MOV A,M
       SUI φ3φH
       MOV B,A
BB1:   JZ BB2
       MVI A,φAH
       ADD B
       MOV B,A
       DCR E
       JMP BB1
BB2:   DCX H
       MOV A,M
       SUI φ3φH
       MOV E,A
BB3:   RZ
       MVI A,φ64H
       ADD B
       MOV B,A
       JMP BB3
DATIN: CALL TYRDR    :TTY의 tape reader로
                      부터 B문자가 익힐때
                      까지 tape를 읽는다.
       MVI A,φ42H
       CMP B
       JNZ DATIN
DATA1: LXI M,13FFH   :data bit수를 set
       MVI M,φF8H
DATA2: CALL TYRDR    :1 byte의 data를 읽어
                      P,N,B 또는 RUB인가
                      를 찾아 분기하고 다
                      른 入力일 경우 Format
                      error로 처리한다.
       MVI L,φFAH
       MVI A,φ5φH
       CMP B
       JZ PDATA
       MVI A, φ4EH
       CMP B
       JZ NDATA
       MVI A,φ42H
       CMP B
       JZ DATA1
       MVI A,φ7FH
       CMP B
       JZ DATA2
```

```
        JMP FORHT
PDATA:  MVI A,ϕ1H       :P文字가 入力되었으면 BF H번지의 RAM에 bitϕ에 1을 넣는다.
        RAR
        MOV A,M
        RAR
        MOV M,A
        JMP DATA3
NDATA:  XRA A           :N이면 bitϕ에 ϕ를 넣는다.
        MOV A,M
        RAL
        MOV M,A
DATA3:  MVI L,ϕFFH      :data bit count를 하여 8bit가 차지 않으면 계속해서 data를 읽고, 8bit가 채워지면 FPATA로 넘어간다.
        MOV B,M
        INR B
        MOV M,B
        JNZ DATA2
FDATA:  CALL TYRDR      :F,B,RUB인가 조사하여 아니면 format error로 처리하고, B이면 data bit counter에 (−8)을 set 후 data를 다시 받아들이고, RUB는 무시하며, F이면 다음 byte의 data를 읽는다.
        MVI A,ϕ46H
        CMP B
        JZ DATA4
        MVI A,ϕ42H
        CMP B
        JZ DATA1
        MVI A,ϕ7FH
        CMP B
        JZ DATA2
        JNZ FORMT
DATA4:  XRA A
        RET
FORMT:  MVI B,ϕ46H      :Format error처리로 F,E자를 出力한 후 error번지를 出力하고 Acc에 ϕ1H를 넣고 RET.
        CALL TYOUT
        MVI B,ϕ45H
        CALL TY
LISTA:  CALL CRLF
PRTAD;  MVI L,ϕFDH
        MOV B,M
        CALL BIBCD
        MVI E,ϕFDH
        DCX H
        DCX H
FM1:    MOV B,M
        CALL TYOUT
        INX H
        INR E
        JNZ FM1
        MVI A,ϕ1H
        RET
BIBCD;  LXI H,13F1H     :Binary숫자를 ASCII code의 10진수로 변환
BNBD:   MVI C,ϕH
        MOV A,B
BD4:    SUI ϕ24H
        JC BD2
        INR C
        JMP BD1
BD2:    ADI ϕ94H
        MOV B,A
        MOV A,C
        ADI ϕ3ϕH
        MOV M,A
        MVI C,ϕH
        MOV A,B
BD3:    SUI ϕAH
        JC BD4
        INR C
        JMP BD3
BD4:    ADI ϕAH
        MOD B,A
        MOV A,C
        ADI ϕ3ϕH
        INX H
        MOV M,A
        MOV A,B
        ADI ϕ3ϕH
        INX H
```

```
        MOV M,A
        RET
SETMA:  LXI H,13FCH     ;현재 실행번지를 HL에
                         set
        MOV D,M
        INX H
        MOV A,M
        MOV L,A
        MOV H,D
        RET
CHECK:  LXI H,FEH       :종료번지와 현재실행
                         번지를 비교하여 같으
                         면 그대로 RET 다르
                         면 현재 실행번지를 1
                         증가시키고 RET.
        MOV A,M
        DCX H
        CMP M
        RZ
        INR M
        RET.
```

그림 11. BNPF型의 program list.

Ⓑ Hexadecimal型 loader.

이 型은 TTY를 I/O device로 사용할 때 가장 편리하게 쓸 수 있다. error check는 data가 hexadecimal數로 되어 있는가의 조사와 data의 총 byte수와 실제 入力된 data의 수가 일치하는가의 조사, 2가지만 하면 충분하다. hexadecimal型에서의 object는 ASCII code의 16진수에 해당하는 숫자와 종료부호, 그리고 data수정을 위한 RUB out과 monitor를 위한 comment인 Carriage Return과 Line Feed로 구성되어 있다. 紙 tape의 format이 그림 12에 있으며 loader의 실행과정의 list가 그림 13에 나타나 있다. Hexadecimal型에서의 1byte data는 object tape 中에서 2 frame을 차지하게 되는 데 이때 2 frame씩의 사이에 CR,LF를 넣을 수 있다. 그리고 BUS OUT은 紙 tape 작성 도중 error가 생긴 경우 유효하게 쓸 수가 있는데, 1byte의 data를 나타내는 2frame의 부호중 1번째 frame의 data가 error일 경우 계속하여 RUB out을 punch하면 다시 data를 넣을 수 있으며, 2번째 frame에 error가 있으면 계속해서 RUB OUT을 punch함에 의해 error가 생긴 부분의 첫 번째 frame부터 다시 넣을 수 있는데 이때는 RUB OUT을 punch한 수 만큼 data를 거슬러 올라가 다시 넣을 수도 있다. data의 入力도중 16진수를 벗어나는 error가 생기면 "?"부호가 type되고 keyboard로부터 올바른 入力을 요구하게 된다. 또, data의 총 byte수만큼 data를 읽은 다음, 다음 文字가 종료부호인 "$"자가 아니면 "space"와 "?"부호를 type한 후 program의 START부분으로 되돌아 온다. 이때 TTY에 의해 monitor된 list를 보아 data가 바르게 入力되었으면 무시할 수 있으며 data에 error가 있으면 tape上의 error를 나타낸다. program이 load되면 E-command에 의해 program을 실행할 수 있다. 이 hexadecimal型 loader를 그림 14에 보인다.

© Binary Loader

Binary loader는 1 byte의 data를 그대로 object中의 data로 쓰고 있으므로 紙 tape의 소요길이가 가장 짧으며, 임의의 data를 1 byte로 나타낼 수 있는고로 자유스럽게 여러가지의 機能을 부여할 수 있다. 반면 data의 error check는 data의 총byte수를 紙 tape上에 싣고 실제 入力된 data의 수를 비교하는 것과 loader 내의

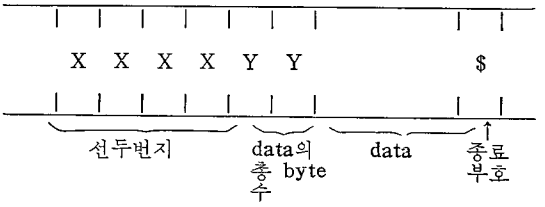


그림 12. 紙 tape format.

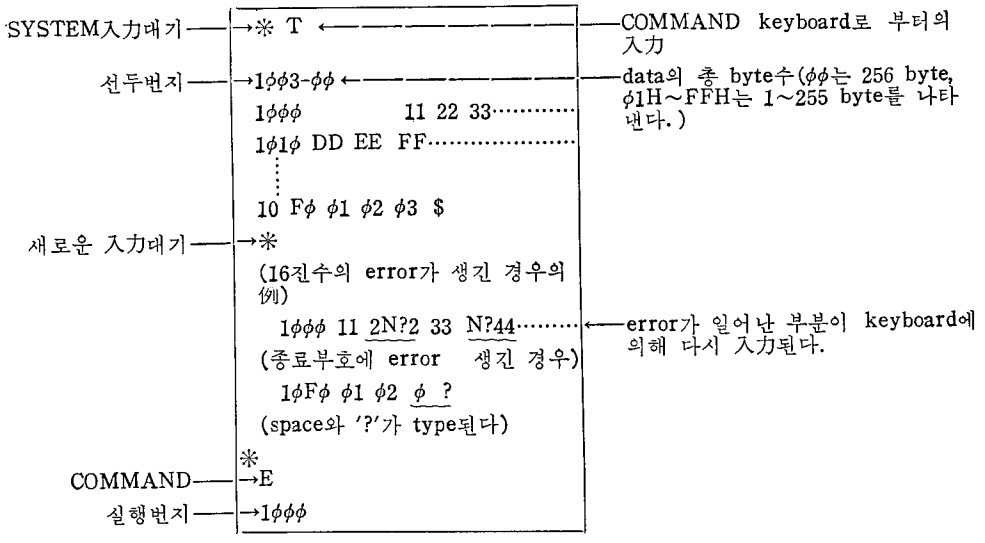


그림 13. Hexadecimal型 loader 실행과정의 list.

```
START: CALL  CRLF    :CR,LF,※부호를 出力
       MVI   B, φ2AH
       CALL  TYOUT
       CALL  TYKBD   :Keyboard로 부터
                      command 入力
       MVI   A, φ54H :T?
       CMP   B
       JZ    TAPIN
       MVI   A,φ45H  :E?
       CMP   B
       JZ    RXCUT
       CALL  ERROR   :COMMAND error.
       JNP   START
-------★-------★-------★-------
TAPIN: CALL  ADPRO   :실행번지 및 총 dapa
                      byte수를 읽고 이에따
                      른 TTY에의 list처리
                      를 한후 data를 읽는다.
DATRD: CALL  DBRDR
       MOV   H,A
       CALL  SPADP
       JNZ   DATRD
       CALL  TYRDR   :종료부호 $를 확인
       MVI   Aφ24H
       CMP   B
       JZ    START
       CALL  SPACE   :종료부호 error
       CALL  ERROR
FXCUT: CALL  ADSRT   :실행번지 入力
       PCHL          :실행개시
-------★-------★-------★-------
              SUBROUTINE
-------★-------★-------★-------
SPACE: MVI   B, φ2φH :"Space"를 TTY에
                      出力
       CALL  TYOUT
       RET
DBRDB: INR   C       :1번째 frame이 RNB
                      OUT 있을때 수행번
                      지 및 counter를 DCR
       DCX   H
DBRDR: CALL  TYRDR   :data를 읽어 16진수인
                      가 확인 후 1byte로
                      合成한다.
       MVI   A,φTFH
       CMP   B
       JZ    DBRDB
       CALL  ASCBI
       RLC
       RLC
       RLC
       RLC
       MOV   D,A
       CALL  TYRDR
       MVI   A,φ7FH
```

```
        CMP   B
        JZ    DBRDR
        CALL  ASCBI
        ORA   D
        RET
ASCBI:  MOV   A,B       :ASCII code의 data를 binary로 변환하면서 error가 있으면 keyboard로 부터 다시 入力을 받는다.
        SUI   ϕ4H
        JC    ASCBD
        SUI   ϕ6H
        JNC   DATER
        ADI   ϕ1ϕH
        RET
ASCBD:  ADI   ϕ11H
        jump if minas   DATER
        SUI   ϕAH
        JNC   DATER
        ADI   ϕAH
        RET
DATER:  CALL  ERROR
        CALL  TYKBD
        JMP   ASCBI
DBIAS:  MOV   A,D       :d regster 있는 binary data를 그 문자의 ASCII code로 변환하여 TTY에 出力한다.
        ANI   ϕFϕH
        RRC
        RRC
        RRC
        RRC
        CALL  BIASC
        MOV   A,D
        ANI   ϕFH
        CALL  BIASC
        RET
BICSC:  SUI   ϕAH       :D register의 1문자 分의 data를 ASCII code로 변환
        JC    BIASD
        ADI   ϕ7H
BIASD:  ADI   ϕ3H
        MOV   B,A
        CALL  TYOUT
        RET
ADSET:  CALL  CRLF      :address 入力을 읽어 H.L에 set시키고, 총 data byte수를 읽어 E에 set시킨다.
        CALL  DBRDR
        MOV   H,A
        CALL  DBRDR
        MOV   L,A
        CALL  SPACE
        CALL  DBRDR
        MOV   E,A
        CALL  CRLF
ADPRT:  CALL  CRLF      :HL에 set된 RAM번지를 TTY에 出力한다.
        MOV   D,H
        CALL  DBIAS
        MOV   D,L
        CALL  DBIAS
        RET
ADPRO:  CALL  ADSCT     :선두 번지를 읽어들여 HL에 Set시킨 후 해당 address의 최하위 숫자가 ϕ인 address를 出力하고 해당 address 부분앞까지 space를 出力한 뒤 해당번지 부터 읽어들인 data를 type해 나간다. address가 최하위가 F까지 진행되면 다음행에 번지를 한 후 data를 같은 과정으로 읽어 들인다.
        MOV   D,H
        CALL  DBIAS
        MOV   A,L
        ANI   ϕFϕH
        MOV   D,A
        CALL  DBIAS
        CALL  SPACE
        MOV   A,L
        ANI   ϕFϕH
        MOV   D,A
        MOV   C,A
        JZ    NOSPA
ADDSP:  CALL  SPACE
        CALL  SPACE
        CALL  SPACE
```

```
        DCR   D
        JNZ   ADDSP
        MVI   A,1φH
        SUB   C
        MOV   C,A
        RET
NOSPA:  MVI   C,1φH
        RET
SPADP:  INX   H        :1 byte의 data를 수행한 후 HL을 1증가시키고 data byte 실행수를 1 감소시켜 16개 완료되었으면 다시 C register에 1φH를 Set한 후 번지를 type하고 space를 出力한 뒤 총 data byte수를 1감소시켜 RET, 16개가 차지 않으면 space를 出力하고 총 data byte수를 1감소하고 RET.
        DCR   C
        JNZ   FOLLO
        MVI   C,φ1φH
        CALL  ADPRT
FOLLO:  CALL  SPACE
        DCR   E
        RET.
```

그림 14. Hexadecimal型 loader program listing.

수식부호의 error 검출만을 행하고 있으므로 入出力情報媒體의 信賴度가 높아야 한다. 紙 tape의 format는 그림 15와 같이 되어 있다. 그中 tape 수식부호는 tape가 data를 갖거나, 실행번지를 주어 program을 실행시킴을 나타낸다. 그리고 여기에서의 binary loader는 absolute object와 relocatable object의 2가지가 가능하며 영역 확보도 가능하다. 그림 16에 tape 수식부호와 그 識別 code를 보인다. 다음으로 data部에 있어서 relocatable을 실현시키기 위해 모든 data는 그 data에 대한 정보를 갖는 head라는 1byte의 data가 필요하다. 즉, data部에서는 head가 먼저 위치하고 그 情報에 따라서 지정한 수 만큼 data를 받아들이며, 또 다시 head가 있고 이에따른 data가 있게 된다. Head는 그림 17에 보여준다. 먼저 data型과 address型, 영역 확보형의 3가지로 나누어 data型은 head이후 최대 65 byte까지 data를 실을 수 있다. Address型은 USER에 의해 설정된 ROM선두

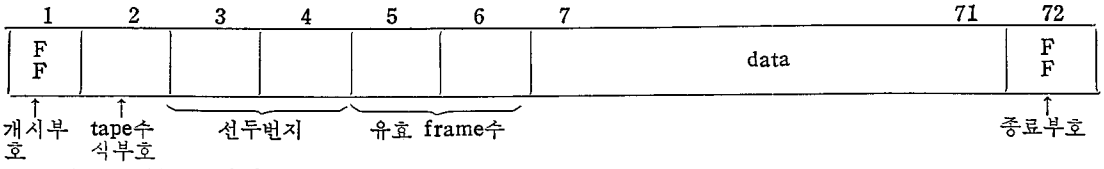


그림 15. 紙 tape의 format.

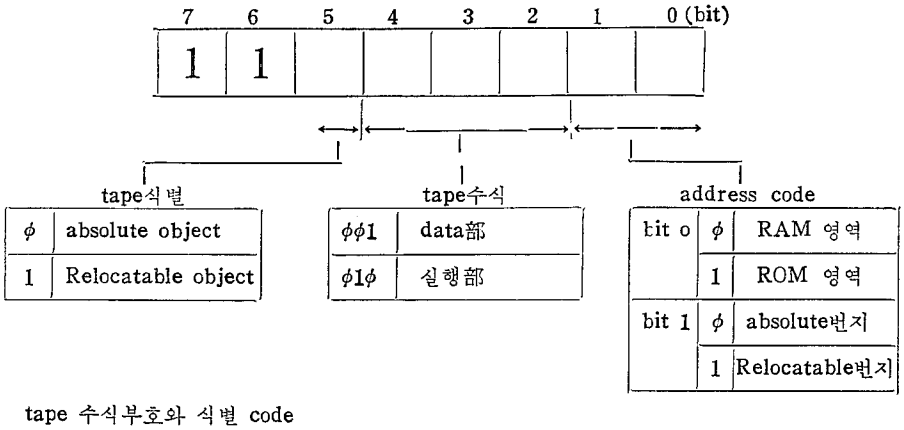


그림 16. tape 수식부호와 식별 code

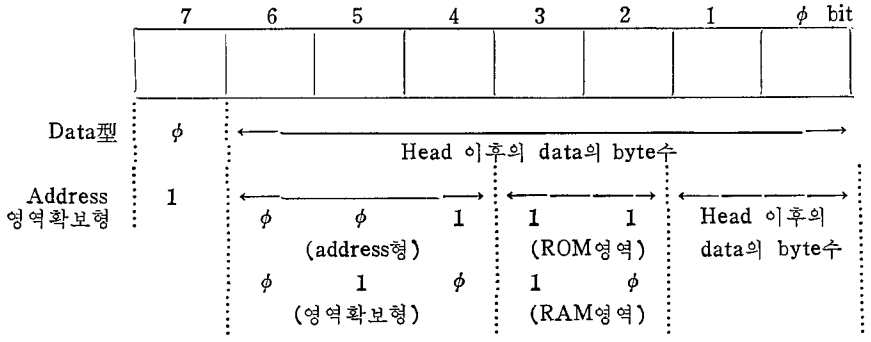


그림 17. Head의 code.

번지와 RAM 선두번지를 참조로 하여 Head 이후에 나온 2 byte의 data를 더한 것이 load되는 data가 된다. 영역 확보型은 head 이후 지정한 byte만큼 영역을 확보해 두게 된다. 그러므로 USER가 ROM 선두번지와 RAM 선두번지를 설정함에 의해서 tape上의 address를 나타내는 data는 relocatable object로 가능하게 되는 것이다. 따라서 여기서의 binary loader는 그림 18에 보이는 만큼의 RAM 영역을 확보해 있어야 한다. 그 中에서 ROMAD와 RAMAD번지에는 사용자가 loader 를 실행하기에 앞서 data를 기억시켜야 한다. 그리고 load 실행번지에 있어서도 tape 수식부호의 bit 5와 bitφ에 의해서, absolute면 실행 번지 그대로, relocatable이면 bitφ에 지정한 RAM 또는 ROM선두번지를 더한 번지에 program이 load 되도록 되어 있다. program 中에서 紙 tape reader의 status번지는 3BH로 bit3은 전원 OFF flag이고 bitφ는 busy flag를 나타낸다. data의 入力은 7BH번지로 들어온다. 개시 및 종료부호는 FFH를 사용하였고 tape의 길이는 반드시 72 frame으로 이루어져 있어야 한다. program을 그림 19에 보인다.

SYMBOL	필요 byte	용 도
ROMAD	2	ROM 선두번지
RAMAD	2	RAM 선두번지
ERP	1	Error를 나타낸다
STADD	2	load 실행번지
PTRIDT	2	PTRT의 현재 참조번지
PTRT	65	tape의 data를 일단 읽는다.
ST	2	실행부의 program 실행번지

그림 18. Binary loader가 점유하는 RAM영역

```
STAR CT:ALL  PTRL
        CPI  FFH     :개시부호검출
        JNZ  START.
        MVI  B,FFH
        CALL PTRL    :Tape 수식부호入力
        ANI  DCH
        CPI  C4H
        JZ   DATRD   :DATA부호
        MOV  A,C
        ANI  DCH
        CPI  C8H
        JZ   EXCUT   :실행부호
        JMP  ERROR
DATRD:  MOV  P,C     :DATA로 읽어들인다
        CALL PTRL    :실행번지入力
        MOV  L,A
        CALL PTRL
        MOV  H,A
        MOV  A,D
        ANI  2φH
```

```
        JZ      ABSOL   :Absolute object처리로
        MOV     A,D
        ANI     φ1H
        JZ      RAM     :RAM번지참조
        XCHG
        LHLD    ROMAD   :ROM번지참조
        JMP     ADPRO
RAM:    XCHG
        LHLD    RAMAD   :RAM번지참조
ADPRO:  DAD     D
        SHLD    STADD
        CALL    PTRL    :유효 frame수를 읽어냄
        CALL    PTRL
        MVI     D,65D   :Data부분을 읽어들임
RDSUC:  LXI     H,PTRT
        CALL    PTRL
        MOV     M,A
        INX     H
        DCR     D
        JNZ     RDSUE
        CALL    PTRL    :종료부호검출
        CPI     FFH
        JNZ     ERROR
        MOV     M,A
        LXI     H,PTRT
        SHLD    PTRIDT
TDSUC:  LHLD    PTRIDT
        MOV     A,M
        ANA     A
        JZ      START
        JP      PATA    :Data형
        ANI     FBH
        CPI     PAH
        JZ      ADDFO   :ADDRESS형
        MOV     A,M
        CPI     A2H
        JZ      BSSF    :영역확보형
        JMP     ERROR   :Head의 format error
DATA:   MOV     C,A     :DATA型 처리
        INX     H
        XCHG
        LHLD    RTADD
        XCHG
DASUC:  MOV     A,M
        XCHG
        MOV     M,A
        INX     H
        SHLD    STADD
        XCHG
        INX     H
        SHLD    PTRIDT
        DCR     C
        JNZ     DASUC
        JMP     TDSUC
ADDFO:  MOV     A,M     :Address형
        ANI     φ4H
        JZ      RAMD
        LHLD    ROMAD   :ROM address 참조
        JMP     ADSUD
RAMD:   LHLD    RAMAD   :RAM address 참조
ADAUC:  PUSH    H
        LHLD    PTRIDT
        INX     H
        MOV     E,M
        INX     H
        MOV     D,M
        INX     H
        SHLD    PTRIDT
        POP     H
        DAD     D
        XCHG
        LHLD    STADD
        MOV     M,E
        INX     H
        MOV     M,P
        INX     H
        SHLD    STADD
        JMP     TESUC
BSSF:   INX     H       :영역확보
        MOV     E,M
        INX     H
        MOV     D,M
        INX     H
        SHLD    PTRIDT
        LHLD    STADD
```

```
        DAD   D
        SHLD  STADD
        JMP   TDSUC
"EXCUT: CALL  PTRL    :실행부의 실행번지 入力
        MOV   L,A
        CALL  PTRL
        MOV   H,A
        SHLD  ST
        MVI   D, 68D  :tape의 나머지 부분을
                       읽어넘김
EXSUC:  CALL  PTRL
        DCR   D
        JNZ   EXSUC.
END:    HLT.
ERROR:  MVI   A,1     :error처리
        STA   ERP
        JMP   END
```

★ ★ ★

SUBROUTINE

★ ★ ★

```
PTRL:   CALL  PTR     :data로 읽어 C에 넣고
                       B와 exllusiv의 OR를
                       행한다.
        MOV   C,A
        XRA   B
        MOV   B,A
        MOV   A,C
        RET
PTRL:   IN    3BH     :STATUS를 읽고 준비
                       가 되면 Data를 읽어
                       들인다.
        ANI   ϕ9H
        JNZ   PTR
        IN    7BH
        RET
```

그림 19. Binary loader program list.

7. 맺는 말

μC는 産業界에 具體的인 硏究開發의 對象이 되고 있는데 急激하게 發達하고 있는 分野이기 때문에 硏究結果가 서로 서로 빨리 交換이 되어야 하겠는데 産業界 內部끼리는 이러한 것이 期待되기 힘들다. 이 글을 쓰기 위한 硏究는 産學協同財團의 1977年度 硏究費에 依해서 行하여졌고 그 結果가 産業界에 넓게 알려져서 實務者들에게 도움이 되기를 바라는 바이다. 便宜를 提供하여 주신 諸賢께 感謝하고 充分치 못한 點에 對한 指摘과 疑問에 對한 質疑와 相談을 서슴없이 하여 주시기를 부탁드리는 바이다.

參考文獻

1. Intel 8080 Microcomputer Systems User's Manual
2. Pantos計測器 取扱說明書(日本 電子科學株式會社)
3. Intel 8080 Assembly Language Programming Manual.
4. The μCOM-8 software manual, NEC.
5. Interface 1977年 4月號, CQ出版社.

◇