PAULO CÉSAR MALDONADO

Z80
APLICAÇÕES

ÉRICA

Z80

APLICAÇÕES

PRÁTICAS

Dados de Catalogação na Publicação (CIP) Internacional
(Câmara Brasileira do Livro, SP, Brasil)

Maldonado, Paulo Cesar, 1951-
Z80 : aplicações práticas / Paulo Cesar Maldonado. -- São Paulo : Érica, 1990.

Bibliografia.

1. APL-80 (Computador) 2. Z-80 (Microprocessador) I. Título.

CDD-001.64
90-0187 -001.6404

Índices para catálogo sistemático:
1. APL-80 : Computadores : Processamento de dados 001.64
2. Microprocessadores Z-80 : Processamento de dados 001.6404
3. Z-80 : Microprocessadores : Processamento de dados 001.6404

PAULO CESAR
MALDONADO

Z80

APLICAÇÕES

PRÁTICAS

1a. EDIÇÃO - 1990

LIVROS ÉRICA EDITORA LTDA.

Dedicatória

Dedico esta obra,
a minha esposa
Aline

Informações sobre o Autor

Paulo Cesar Maldonado há 18 anos vem se especializando na área de Eletrônica Digital, Microcomputadores e Eletrônica Industrial.

Através desses anos, além do setor profissional, vem se dedicando à área didática, desenvolvendo um trabalho digno, a respeito de:

* Publicações técnicas em revistas ligadas à área de atuação.

* Treinamento especializado onde ministra cursos em empresas e indústrias ligadas à área.

* De 1980 a 1983, como Professor Diretor da CED - Cursos de Eletrônica Digital S/C Ltda. formou e profissionalizou mais de 2000 alunos, numa época carente desse tipo de mercado.

* Na área de consultoria, projetos, manutenção, automação industrial, etc; vem desenvolvendo trabalhos significativos em empresas de renome, tanto em São Paulo como em todo o Brasil.

* Seu aprimoramento didático e profissional deu-se através de Cursos especiali zados, realizados no Exterior.

* Adquiriu know-how para nacionalização de placas, equipamentos e projetos.

Seus objetivos principais são:

Elevar o nível técnico no Brasil através do fornecimento de conhecimentos tanto fundamentais quanto práticos, objetivando perspectivas de realização didática, profissional e financeira.

Baseado nesses propósitos, o autor desenvolve um trabalho sério, cujos padrões de qualidade são sempre o seu maior motivo de preocupação.

Uma característica importante da personalidade do autor é colocar Deus sobre todas as coisas, bem como utilizar a orientação cristã na sua prática onde prevalece a honestidade e o amor ao próximo.

Importante:
Informações, solicitação de serviços teóricos e/ou práticos, orientação técnica, projetos, cursos profissionalizantes, todo e qualquer assunto inerente à área, são tratados através do tefefone: (011) 274.9789.

SUMÁRIO

1. Bases Numéricas

2. Microprocessador Z-80

3. Microcomputador APL-80 (Hardware)

4. Microcomputador APL-80 (Software)

5. Interface de Impressora

6. Interface de Teclado

Apêndices

INTRODUÇÃO

Generalidades

Tenho observado diversas obras sobre o microprocessador Z80, entretanto, todas a respeito dele como "pastinha", ou seja, detalham-no sempre como um microprocessador e nunca como o poderoso microcomputador que é.
Preenchendo essa lacuna didática é que me proponho, ao longo dessa obra, mostrar como funciona um Z80 "aplicado", ou ainda, como pode ser interconectado a outros circuitos mais complexos e realizar uma série de funções exatas de um microcomputador.
O objetivo não é desenvolver algo tão complexo quanto um microcomputador voltado para a área de processamento de dados, mais sim um sistema modular com possibilidade de interconecção, gerando interessantes combinações.
O próposito básico desta obra é fazer com que o leitor "sinta" de maneira prática e simples o funcionamento do seu Z80.

A quem se Destina a Obra

Didaticamente, esta obra se destina ao estudante de eletrônica, entretanto por se tratar de uma literatura de conteúdo técnico-prático, pode acrescer conhecimentos, aos já existentes, a técnicos, projetistas e engenheiros relacionados à área.
Isto porque a obra apresenta módulos exaustivamente testados e acompanhados do seu respectivo sofware (programação).
Os módulos podem ser usados diretamente e ainda servem como base para pesquisa e estudo.
As aplicações selecionadas para esta obra são inicialmente simples e, no decorrer dela, alcançam estágios de complexidade

Como Estão Divididos os Capítulos

Os capítulos desta obra estão divididos de forma didática e racional, o que possibilita uma rápida localização dos ítens de seu interesse.
A forma de divisão escolhida permite ao leitor consultar a obra como um manual, bem como, estudar capítulos isolados sem que isto comprometa o entendimento dos outros restantes.

Como Estudar Corretamente

Nada mais lógico do que conhecer a obra na sua totalidade, entretanto para os leitores que já reúnem conhecimentos a cerca do Z80 como pastilha, torna-se possível eliminar o 1º e o 2º capítulos.
Mesmo assim, reúnem-se as condições necessárias para selecionar os capítulos práticos e partir para a montagem dos circuitos aplicativos.
Os leitores que desejarem estudar apenas o microcomputador básico deverão selecionar o capítulo 3.

Estudo dos Circuítos Aplicativos (Montagens)

O conteúdo dos circuítos aplicativos é bastante detalhado.
Como a descrição dos módulos é dirigida a estudantes, seu estudo seria altamente complemantado, se a montagem física dos mesmos pudesse passar da teória à prática de bancada.

Sequência de Montagem dos Módulos

Optando pela montagem dos circuítos, o leitor deverá iniciar seus estudos pelo kit "microcomputador básico".
Trata-se do módulo onde está incluído o chip Z80, o responsável pelo gerencialmente dos demais módulos, ou seja sem ele, torna-se impossível o funcionamento dos módulos subsequentes.
Após a montagem do microcomputador básico, aí sim, o leitor poderá selecionar, aleatoricamente, qualquer outro módulo de seu interesse e interligá-lo ao primeiro e principal (Z80).

Módulos/Componentes e Disponibilidade

As montagens são simples e seus componentes facilmente encontrados no mercado nacional especializado.
As placas de circuíto impresso e a memória com o programa monitor já gravado são possíveis de confecção através dos detalhes fornecidos no decorrer dos capítulos, entretanto, para maiores esclarecimentos e ou aquisição direta dos mesmos, esta Editora fornece informações de como proceder.
O propósito básico da obra não é a comercialização de placas ou componentes e sim desburocratizar barreiras para a execução dos módulos, visando concluir nossos objetivos essencialmente didáticos.

Descrição dos Módulos

Os módulos ou kits são detalhados em etapas:

a- Primeiramente, faz-se uma narrativa do circuíto tal como tipo de módulo, para que serve e onde usá-lo.

b- Posteriormente apresenta-se o esquema eletrônico acompanhado da descrição de suas funções e subfunções.

c- A última etapa trata da programação do módulo selecionado. Também é inter-relacionado o funcionamento do programa com a parte eletrônica, bem como, divulgada, com clareza de detalhes, a listagem do programa.

Nota:
Os circuítos apresentados são didáticos e servem para aplicações imediatas. Foram montados e testados exaustivamente, razão pela qual não nos responsabilizamos por componentes defeituosos ou erros de operação que venham comprometer o funcionamento dos circuítos.

Padrões Técnicos e Qualidade

Esta obra foi escrita baseada em critérios rígidos e dentro de padrões técnicos comparados aos das obras estrangeiras, mundialmente reputadas como de melhor qualidade.Toda literatura descrita está adaptada às necessidades e modalidades brasileiras, tornando-a fácil acessível e proveitosa.

Termos Técnicos

Os circuítos integrados utilizados nesta obra são de fácil aquisição, entretanto, por serem de origem americana, trazem siglas impressas no idioma inglês.
Por esta razão serão mantidos os termos na sua expressão natural.

Nota do autor:
Baseado nessa modalidade de trabalho, espero contribuir, através desta obra, para o engradecimento do técnico brasileiro e só assim terei alcançado os objetivos propostos.

CAPÍTULO 1

BASES NUMÉRICAS

ÉRICA

1.1 Introdução

A título de revisão, estudaremos neste capítulo, noções fundamentais sobre bases numéricas ou sistemas de numeração.
Existem variados tipos de bases de numeração, entretanto, apenas as bases (2), 10 e 16 serão estudadas por serem essenciais na operação dos microprocessadores.
Entendemos como base numérica uma faixa de números utilizados na forma de "base" de uma determinada família numérica.
Por exemplo: a base (2) (binária) compreende os números 0 e 1. A base 16 compreende os números de 0 a 15, ao passo que a base 32 corresponde aos números de 0 a 31.
A notação de uma base numérica é formada por um número seguido por um valor entre parênteses, correspondendo à base a que o número anterior pertence.

Ex.: $2_{(10)}$ $477_{(8)}$ $2AF_{(16)}$ etc.

Convém notar que o maior algarismo numérico conhecido é o 9. Como as bases maiores que 10 não têm algarismos para representá-las, costuma-se fazê-lo através de letras do alfabeto.
Outro aspecto importante é que nem sempre a faixa de número corresponde a uma base é a mesma para todos os casos. Por exemplo, inexistem os algarismos numéricos 8 e 9 na base octal (base 8), portanto é incorreto escrever 189(8) uma vez que os números que correspondem à base 8 são 0, 1, 2, 3, 4, 5, 6 e 7.
É preciso atenção quanto aos indicadores das bases numéricas. Como estamos acostumados a lidar com base 10 é natural que vejamos essa tendência tradicional. Nossa formação básica insistiu na utilização da base 10 (decimal) para representar grandezas básicas ou quanticas. Por isso, até parece estranho nos familiarizarmos com outras bases numéricas como por exemplo 398_{16}, 398_{12}.

Convém destacarmos sempre a base com a qual estamos trabalhando, pois uma omitida, poderá ser interpretada como base 10 e consequentemente não corresponde à realidade da questão.
Normalmente, quando escrevemos programas para micros de 8 bits, como por exemplo o Z80, o fazemos sem referenciar a base numérica, isto porque, dificilmente tais programas são escritos em outras bases a não ser a hexadecimal (16).
Acostumados a lidar com base decimal, mesmo porque temos 10 dedos e contamos até 10, precisamos saber que quando lidamos com máquinas estamos desperdiçando circuítos, uma vez que "circuítos decimais" jogam fora "bits".
Para entender como funciona uma base numérica decimal tomemos como exemplo o número 1988 (10).
O número possui 4 casas de algarismos e como a base é decimal podemos dizer que são 4 casas decimais.
A cada uma delas corresponde uma grandeza, ou seja, uma potência (de 10) que multiplicada pelas respectivas potências, permite-nos obter o valor final de cada casa.
Vejam que cada dígito ou casa decimal é submetido a uma potência consecutiva de 10.

$1988 = 8x10^0 + 8x10^1 + 9x10^2 + 1x10^3$
$1988 = 8x1 + 8x10 + 9x100 + 1x1000$
$1988 = 8 + 80 + 900 + 1000$

Notem que o número 1988 foi decomposto na sua forma mais simples e posteriormente reagrupado para voltar a sua forma originária.
O procedimento é o mesmo para outras bases de numeração.

1.2 Números Binários

Os circuitos digitais utilizam dois tipos de níveis de operação, denominados "níveis lógicos" e definidos por dois estados enterpretados de várias formas conhecidas como:

- ligado ou desligado
- aceso ou apagado
- ativado ou desativado
- verdadeiro ou falso
- alto ou baixo
- energizado ou desenergizado, etc

Como os circuítos digitais utilizam apenas 2 estados distintos, nada mais lógico do que associá-los aos números binários 0 e 1.
Tais números binários não só se comportam bem em relação aos circuítos lógicos eletrônicos, como também exprimem uma série de fenôminos munidos de sentido lógico.
Nos circuítos eletrônicos digitais costumamos utilizar uma determinada variação distinta de tensão para expressar níveis lógicos, ou seja:

de 0 a 1,4 volts = nível lógico 0
de 2,4 a 5 volts = nível lógico 1

Um circuíto eletrônico tipo lógico TTL (Transistor Transistor Logic) responde a apenas estes 2 níveis distintos, ignorando os demais níveis intermediários.
Existem outras famílias lógicas tais como CMOS, MOS, ECL etc, cujas faixas de lensão são outras, entretanto, conservam os aspectos básicos ou seja, níveis lógicos 0 e 1.
Em numeração decimal, cada elemento é chamado de casa, posição ou dígito decimal. Os dois elementos binários são os dígitos 0 e 1.
Convencionou-se ainda que os 2 elementos binários são chamados de BIT (binary unit) "unidade binária".
Um bit pouco representa numericamente, todavia uma combinação de bits pode representar uma grandeza decimal significativa.
Para formamos um número decimal precisamos de vários circuítos produtores de níveis lógicos 0 e 1, ou seja, circuítos capazes de controlar vários bits simultaneamente.

1.3 Conversão binária/decimal

Recordando o exemplo da composição de um número decimal, temos:

1(mil)=	$1x10^0$=	1000
9(novecentos)=	$9x10^1$=	900
8(oitenta)=	$8x10^2$=	80
8(oito)=	$8x10^3$=	8

		$1988_{(10)}$

Notem que o fator de potência dos números decimais é 10 (10^0, 10^1, 10^2, 10^3, etc)
Em se tratando de números binários o processo é análogo, porém, o fator de potência passa a ser 2.

Exemplo de números binários:
1010001, 10, 1001, 1111, 0000, etc

Não podemos ler um número em binário como escrevemos, devemos lê-lo tal como as máquinas ou processadores, bem como, relacioná-los com a nossa velha base decimal, ou seja:
Para cada casa de 1 bit ou dígito binário corresponde uma potência.
As potências normalmente aumentam da direita para a esquerda, mas isto não quer dizer que esta ordem não possa ser alterada.
O total da soma das respectivas potências para os bits = 1(ligados) corresponde ao valor relacionado em decimal.

Ex.: o número binário 1001101 equivale ao número decimal 77: $1001101_{(2)} = 77_{(10)}$.
Para decompor o número 1001101 e chegar ao 77 em decimal, costumamos proceder

da seguinte forma:

a) fornecer um valor de potência de 2 para cada dígito binário da direita para a esquerda

2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	**0**	**0**	**1**	**1**	**0**	**1**

b) efetuar as potências de 2 e tornar relacioná-las com o número binário:

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$

64	32	16	8	4	2	1
1	**0**	**0**	**1**	**1**	**0**	**1**

c) Somar os valores das potências equivalêntes aos dígitos binários ligados, ou seja, os bits = 1

$$1+4+8+64 = 77$$

Assim, podemos dizer que $1001101_{(2)} = 77_{(10)}$

Vejamos outro exemplo:

Converter o número binário 1010100 em decimal.

a) fornecer um valor de potência de 2 para cada dígito binário da direita para a esquerda.

2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	**0**	**1**	**0**	**1**	**0**	**0**

b) efetuar as potências de 2 e tornar a relacioná-los com o número binário.

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$

64	32	16	8	4	2	1
1	**0**	**1**	**0**	**1**	**0**	**0**

c) Somar os valores das potências equivalentes aos dígitos binários ligados, ou seja, os bits = 1

$$4+16+64 = 84$$

Assim, podemos dizer que $1010100_{(2)} = 84_{(10)}$

A todo esse processo exemplificado até agora chamamos de *conversão binária decimal.*

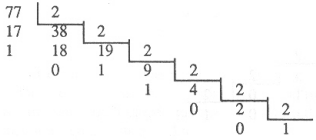
Tentem efetuar as seguintes conversões binárias decimais e chegar nos resultados corretos, abaixo fornecidos:

- $101_{(2)} = 5_{(10)}$
- $1000101_{(2)} = 69_{(10)}$
- $1001_{(2)} = 9_{(10)}$
- $100000001_{(2)} = 257_{(10)}$

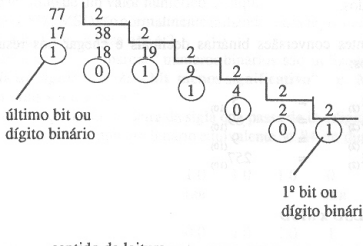
1.4 Conversão Decimal/Binária

A conversão decimal binária utiliza um processo diferente para passar um número da base 10 para a base 2.
Vejamos novamente o número $77_{(10)}$ e como fazer para chegar no binário 1001101.

a) Dividir o número decimal por tantas vezes quantas forem necessárias, até que se torne menor que 2.



b) Anotar os números da direita para a esquerda a partir do último quociente, seguido de todos os restos, bem como, escrevê-los da esquerda para a direita.



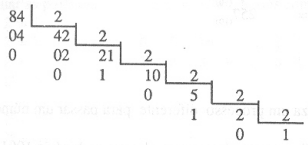
sentido da leitura ←

1º bit (1) 0 0 1 1 0 (1) último bit

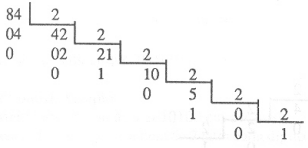
Portanto 77 = 1001101

Vejamos outro exemplo:
Converter o número decimal 84 em binário

a) Dividir o número decimal por 2 tantas vezes quantas forem necessárias, até que se torne menor que 2.



b) Anotar os números da direita para a esquerda a partir do último quociente, seguido de todos os restos, bem como, escrevê-los da esquerda para a direita.



Portanto $84_{(10)} = 1010100_{(2)}$

Notem que os exemplos 77 e 84 foram convertidos no item 1.3. Para clareza de seus estudos confira os resultados.

Tentem efetuar as seguintes conversões binárias decimais e chegar nos resultados corretos, abaixo fornecidos:

- $101_{(2)} = 5_{(10)}$
- $1000101_{(2)} = 69_{(10)}$
- $1001_{(2)} = 9_{(10)}$
- $100000001_{(2)} = 257_{(10)}$

1.5 Siglas LSB x MSB e LSD x MSD

Convém saber o significado das siglas LSB, MSB, LSD e MSD, bem como utilizá-las. Podemos escrever um número binário da direita para a esquerda (maneira mais usual) como da esquerda para a direita.

Este procedimento é válido para todas as bases numéricas, entretanto, é preciso identificar corretamente o início e o fim da leitura dos dígitos.
Vejam como pode ser escrito em decimal o número um mil novecentos e oitenta e oito.

1988 (maneira usual)
8891 (maneira de identificação correta)

As siglas MSD e LSD têm a finalidade única e exclusiva de "marcar" o dígito mais e o menos significativo de um determinado número independentemente do tipo de base.
A sigla **MSD** significa ***"most significant digit* - dígito menos significativo"** e posiciona o lado mais significativo do número, ou seja, onde se localiza a potência maior.
A sigla **LSD** significa ***"least significant digit* - dígito menos significativo"** e posiciona o lado menos significativo do número, ou seja, onde se localiza a potência menor.
Conforme determinam as siglas, notem como deve ser escrito o número.

1988
MSD LSD

8891
LSD MSD

Pode parecer estranho codificar números através das siglas, pensando que poderíamos simplesmente escrevê-los de maneira usual, entretanto, quando manipulamos circuitos digitais, concluímos que nem sempre o dígito mais significativo é o da esquerda, como acontece habitualmente.
Nos desenhos de esquemas, nos números dos displays, nas placas de circuítos impresso, etc isto acontece frequentemente.
Por esta razão é preciso posicionar corretamente as siglas LSD e MSD, determinando início e término de um valor numérico qualquer.
As siglas LSD e MSD são normalmente utilizadas para representar números não binários, ou seja, aqueles cuja base não é 2.
Por outro lado, para "marcar" números binários são utilizados duas siglas especiais: **LSB "*least significant bit* - bit menos significativo"** e **MSB "*most significant bit* - bit mais significativo".**
Note a diferença da última letra da sigla que passa de D (digit) para B (bit).
Podemos escrever o número binário equivalente ao 84 de duas o73 diferentes formas:

1 0 1 0 1 0 0
LSB MSB

0 0 1 0 1 0 1
LSB MSB

Quando são omitidas as siglas, significa que o dígito mais significativo está posicionado à esquerda e o menos significativo à direita.

1.6 Números BCD

Embora representar números decimais disperdice bits, muitos equipamentos geram e utilizam circuítos decimais para facilitar seu manuseio.
Como a maioria dos computadores e circuítos periféricos funcionam em base númerica binária, fica difícil imaginar como operam em números decimais.
Entretanto isso é possível, e para que circuítos binários funcionem adequadamente em decimal, é necessário utilizar um artifício que "empacota" bits de 4 em 4 e cada pacote de 4 bits passa a representar um dígito binário.
Fica fácil empacotar bits com o auxílio da tabela a seguir, que nada mais é do que os números decimais de 0 a 9 convertidos para binário, sempre utilizando 4 bits.

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

A sigla **BCD "*binàry coded decimal* - binário codificado em decimal"** é feita em grupos de 4 em 4, podendo comportar "n" números ou dígitos binários codificado em decimal.
Assim conclui-se que números BCD variam suas casas numéricas de acordo com as potências de 10 e não de 2.
Deve-se analisar separadamente a combinação dos bits em BCD para cada dígito.
Vejam o número decimal 1988 escrito em BCD.

1	9	8	8
0001	1001	1000	1000

Para expressar cada dígito binário foi utilizada a tabela decimal/binária.
Uma vez anotado ou codificado o número decimal em BCD, podemos unir o código binário de uma só vez, desde que anotemos a sua base (BCD).

$$0001100110001000_{BCD}$$

Para analisar o número BCD separa-se os dígitos de 4 em 4 tal como faz uma máquina que opera em BCD.

0001	1001	1000	1000

Um dígito BCD é representado por 4 bits e estes, por sua vez, podem representar 10 dígitos decimais.
A combinação de 4 bits pode gerar até 16 dígitos, conforme segue:

0000	0	-------- NÚMEROS BCD
0001	1	
0010	2	
0011	3	
0100	4	
0101	5	
0110	6	
0111	7	
1000	8	
1001	9	
1010	10	-------- COMBINAÇÕES NÃO UTILIZADAS
1011	11	
1100	12	
1101	13	
1110	14	
1111	15	

Assim conclui-se que "sobram" 6 combinações que poderiam ser utilizadas para outras finalidades.
Essa "sobra" nada mais é do que o desperdício de bits, mencionado anteriormente. Por isso analise a representação do número 1988 em binário puro e posteriormente em BCD.

a) Convertendo 1988 de decimal para binário:

1988	2														
18	994	2													
08	19	497	2												
0	14	09	248	2											
	0	17	04	124	2										
		1	0	04	62	2									
				0	02	31	2								
					0	11	15	2							
						1	1	7	2						
								1	3	2					
									1	1					

Logo 1988 = 11111000100

b) Tirando a prova (passando de binário para decimal)

2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**
1024	512	256	128	64	32	16	8	4	2	1
1	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**

1024+512+256+128+64+4 = 1988

c) Convertendo 1988 de decimal para BCD (tabela)

1	9	8	8
0001	1001	1000	1000

d) Comparando binário puro com BCD.

Binário puro = 11111000100
BCD = 0001100110001000

Observe que o número 1988 representado em BCD necessita de 16 bits, ao passo que quando representado em binário puro o faz com apenas 11 bits 5 bits parece não significar uma grande diferença para representar 4 dígitos decimais, todavia convém lembrar que quanto maior for o número de casas utilizadas em BCD maior será o desperdício quando comparado em binário puro.
Automaticamente, quanto maior o número de bits, maior será o número de circuítos utilizados.
Existe um número máximo de circuitos possíveis de desperdício e de acordo com o projeto em questão, custo dos circuítos e espaço disponível na placa. Esse número deve ser ponderado de acordo com as necessidades de cada projeto, e por esta razão devemos prever possibilidades de utilização de circuítos BCD.
A título de curiosidade, o número máximo que pode ser escrito em BCD com 16 bits é:

1001 1001 1001 1001

ou seja:

$9999_{(10)}$

Com essa disponibilidade de bits, se utilizados em circuítos binários puros, poderíamos representar o número binário máximo de:

1111 1111 1111 $1111_{(2)}$

ou seja:

$65535_{(10)}$

1.7 Conversão BCD/Decimal

A conversão BCD/Decimal é relativamente simples. Basta utilizar a tabela BCD decimal, completamente inversa à descrita anteriormente:

BCD	Decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9

Exemplo de conversão BCD/decimal:
Tomemos como exemplo a conversão do número 0001 1001 1000 1000_{BCD} em decimal.

a) isolar os bits em pacotes de 4 em 4

0001	1001	1000	1000

b) associar cada bloco de 4 bits com o número decimal equivalente, de acordo com a tabela, para obter o número decimal final.

0001	1001	1000	1000
1	9	8	8

Portanto:

0001	1001	1000	$1000_{(2)}$ =	$1988_{(10)}$

1.8. Números Hexadecimais

Uma vez familiarizados com os números BCD e binários, ficou extremamente fácil entender os números hexadecimais, largamente
utilizados em microputadores, minicomputadores e circuítos digitais em geral.
Números hexadecimais so números cuja base de trabalho é 16 e são formados da mesma maneira que os números BCD, todavia utilizam a base 16 ao invés da base 10.
Números hexadecimais ou hexa não disperdiçam bits e esta é a sua grande vantagem.
Basicamente, são formados pelos algarismos de 0 a 9 seguidos das letras A,B,C,D e F.
Um número em hexadecimal preenche o disperdício de bits ocorrido no código BCD.
A tabela a seguir, mostra o código binário puro de 4 bits utilizado para representar os números hexadecimais.

Binário	Hexa	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Tabela Hexadecimal/Decimal

Os números hexadecimais podem ser representados tanto no formato binário como no hexadecimal.
Por exemplo o número 1AF3:

1	A	F	3
	ou		
0001	1010	1111	0011

Apesar dos números hexadecimais serem representados por algarismos decimais de 0 a 9, nem sempre correspondem aos mesmo números decimais, veja $1988_{(16)}$ não correspode a $1988_{(10)}$

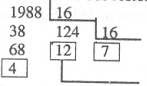
1.9 Conversão decimal/Hexadecimal

Para converter números decimais em hexadecimais utiliza-se a técnica da divisão sucessiva, tal como aconteceu na conversão decimal binária.
A diferença entre as técnicas é que a anterior dividia por 2, enquanto que a decimal/hexadecimal divide po 16.
Veja como fica o número decimal 1988 convertido hexadecimal:

a) dividir sucessivas vezes o número decimal por 16 até obter um quociente menor ou igual a 15.

1988 | 16			
38	124 | 16		
68	12	7	-> quociente menor que 15
4			

b) converter os valores dos restos de 2 dígitos para um só algarismo hexa.

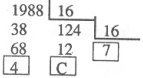


Valor do resto formado por 2 dígitos, que deve ser transformado num algarismo hexa de acordo com a tabela.

Logo:

$$12_{(10)} = C$$

c) anotar o valor do último quociente ao lado do primeiro resto da direita (MSD) para a esquerda (LSD) de maneira que o dígito MSD se posicione do lado esquerdo. Quando o quociente ou resto for composto por 2 dígitos é preciso transformá-los num só algarismo hexa.



LSB MSB

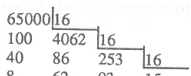
<- SENTIDO DA LEITURA

PORTANTO

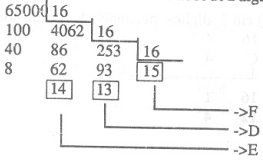
$$1988_{10} = 7C4_{16}$$

Outro exemplo:
Converter o número $65000_{(10)}$ em hexadecimal.

a) dividir o número sucessivas vezes por 16 até obter um quociente menor ou igual a 15.



b)converter os valores dos restos encontrados de 2 dígitos para um só algarismo hexa.



c) anotar os valores ao último quociente ao primeiro resto da direita (MSD) para a esquerda (LSD), de forma que o dígito MSD se posicione do lado esquerdo.

```
65000|16
100  4062|16
40   86   253|16
[8]  62   93  [15]
     [14] [13]
LSD  8    E   D   F   MSD
```

Sentido da Leitura

Portanto $<$ ——————————

$65000_{(10)} = FDE8_{16}$

O número hexadecimal FDE8 pode ser representado binariamente da seguinte forma:

1111	1101	1110	1000
F	D	E	8

1.10 Conversão Hexadecimal/Decimal

Uma das formas mais simples de converter números hexadecimais em decimais é submetê-los aos fatores de potências de 16.

A fim de que possamos testar os resultados, tomemos como exemplo os mesmos números utilizados nas conversões anteriores.

Converter o número hexadecimal 7C4 para decimal.

a) associar uma potência para cada base 16, quantas forem o número de casas hexadecimais.

16^2	16^1	16^0
7	C	4

b) efetuar o cálculo das potências das bases 16 e tornar a associá-las aos números hexadecimais.

$16^0 = 1$

$16^1 = 16$

$16^2 = 256$

logo	256	16	1
	7	**C**	**4**

c) converter as letras (se houver) em 2 dígitos decimais de acordo com a tabela hexadecimal/decimal.

	256	16	1
	7	C	4
		└——————→C = 12	
portanto	256	16	1
	7	**12**	**4**

d) Multiplicar o valor hexa convertido pela potência de 16, pré-calculada para cada casa.

```
 4  x    1  =    4
12  x   16  =  192
 7  x  256  = 1792
```

e)Somar os valores dos produtos efetuados

```
    4
  192
+ 1792
 1988
```

portanto

$1988_{10} = 7C4_{16}$

Outro exemplo
Converter o número hexadecimal FDE8 em decimal.

a)Associar uma potência para cada base 16, quantas forem o número de casas hexadecimais.

16^3	16^2	16^1	16^0
F	D	E	8

b)Efetuar o cálculo das potências das bases 16 e tornar a associá-las aos números hexadecimais.

$16^0 = 1$
$16^1 = 16$
$16^2 = 2564$
$16^3 = 4096$

4096	256	16	1
F	D	E	8

c)Converter as letras (se houver) em 2 dígitos decimais de acordo com a tabela hexadecimal/decimal.

4096	256	16	1
F	D	E	8
		>14	
	>13		
>15			

d)Multiplicar o valor hexa convertido pela potência de 16, pré-calculada para cada casa.

```
 8  x    1  =    8
14  x   16  =  224
13  x  256  = 3328
15  x 4096  = 61440
```

e)Somar os valores dos produtos efetuados

$$\begin{array}{r} 8 \\ 224 \\ 3328 \\ +\ 61440 \\ \hline 65000 \end{array}$$

logo

$$FDE8_{(16)} = 65000_{(10)}$$

Para exercitar seu aprendizado, converta de decimal para hexadecimal e posteriormente de hexadecimal para decimal, os seguintes números.

$154_{(10)} = 9A_{(16)}$
$31217_{(10)} = 79F1_{(16)}$
$65535_{(10)} = FFFF_{(16)}$
$1048576_{(10)} = 100000_{(16)}$

Para facilitar seus estudos, consulte o apêndice A e B onde se encontra uma tabela das potências de 2 e 16 mais usuais.

CAPÍTULO 2

MICROPROCESSADOR

Z-80

ÉRICA

2.1 Z80/Arquitetura Interna

Nosso principal objetivo não é esmiuçar o conteúdo do microcomputador Z80, mesmo porque existe uma profusão de literaturas a esse respeito.
A título de recordação, vejamos apenas as partes mais importantes do microprocessador em si, para posteriormente detalharmos suas aplicações.
A figura 2.1 mostra o diagrama em bloco da pastilha do Z-80.

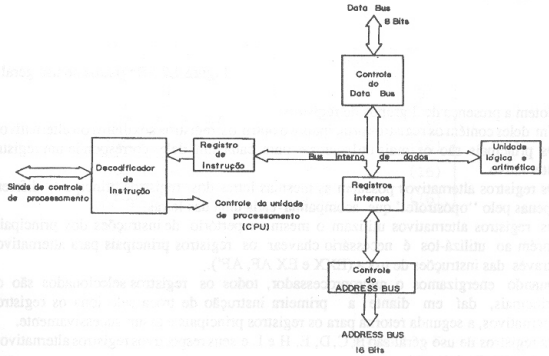


Figura 2.1 - Diagrama Geral do Z80

Nesta arquitetura interna, pode-se observar todas as funções es principais do Z-80.
O conjunto de registros, contadores, temporizadores, etc tem como finalidade prática obter dados de uma memória, bem como, interpretá-los para posteriormente executar uma operação de entrada ou saída.
Um microprocessador é uma máquina síncrona sequencial, capaz de executar milhares de instruções a cada segundo, numa certa, determinada e lógica sequência.

2.2 Registros de uso Geral

O Z80 contém 208 bits de registros na configuração de 8 a 16 bits e todos eles podem ser acessados pelo programador.
Convém salientar que os 208 bits são de memória RAM estática e nessas condições as informações contidas neles não necessitam de recirculação de dados.
A figura 2.2 mostra como estão configurados os registros de uso geral.

REGISTROS PRINCIPAIS		REGISTROS ALTERNATIVOS	
Acumulador A (8)	Flags F (8)	Acumulador' A' (8)	Flags' F' (8)
B (8)	C (8)	B' (8)	C' (8)
D (8)	E (8)	D' (8)	E' (8)
H (8)	L (8)	H' (8)	L' (8)

Figura 2.2 - Registros de uso geral

Notem a presença de 2 grupos de registros.
Um deles contém os registros principais e o outro os registros auxiliares ou alternativos.
Os principais são os mais utilizados e para cada um deles corresponde um registro alternativo.
Os registros alternativos possuem as mesmas letras dos registros principais, diferem apenas pelo "opóstrofe" que acompanha cada uma das letras.
Os registros alternativos utilizam o mesmo repertório de instruções dos principais, porém ao utilizá-los é necessário chavear os registros principais para alternativos através das instruções de troca (EXX e EX AF, AF').
Quando energizamos o microprocessador, todos os registros selecionados são os principais, daí em diante a primeira instrução de troca seleciona os registros alternativos, a segunda retorna para os registros principais e assim sucessivamente.
Os registros de uso geral são B, C, D, E, H e L e seus respectivos registros alternativos: B', C', D', E', H', e L'.

Os registros podem ser utilizados individualmente como registros de 8 bits ou aos pares como registros de 16 bits, ou seja, poderá ser utilizado o par HL ou apenas o L ou H. Quando utilizados aos pares, os registros menos significativos são chamados de byte de baixa ordem (C, E, e L) e os mais significativos de byte de alta ordem (B, D e H). O mesmo ocorre para os registros alternativos, os de baixa ordem (C', E', e L') e os de alta ordem (B, D e H).

2.3 Registros Especiais

Os registros especiais encontram-se na figura 2.3.
Observem na figura 2.2, os dois registros de 8 bits cujas funções são especiais. Trata-se do registro A (acumulador), F (flag), A' e F' .
O propósito do aumulador é assegurar o resultado das operações lógicas ou aritméticas de 8 bits.
O acumulador também serve como porta de entrada e saída para estabelecer comunicação com periféricos e memórias.
Os flags guardam informações a respeito dos resultados das operações aritméticas ou lógicas, indicando ao programa qual a decisão a ser tomada em relação à próxima instrução.
Os registros A e F são chaveados para os registros alternativos A' e F' com a instrução EX AF,AF' .
Os registros A e F são chaveados independentemente dos registros gerais B, C, D, H e L.
A figura 2.3 mostra os outros registros especiais bem como suas funções básicas.

VECTOR INTERRUPT I (8)	REFRESH REGISTER R (8)
Index register	I X (16)
Index register	I Y (16)
Stack pointer	S P (16)
Program counter	P C (16)

Figura 2.3 - Registros Especiais

a) Program Counter

Trata-se de um dos registradores mais importantes PC (program counter), ou seja contador de programa.

Basicamente, controla o fluxo do programa, indicando ao próprio microcomputador o endereço de memória onde se encontra a próxima instrução.

O PC é registro de 16 bits, todavia pode ser modificado pelo programa se necessário. Por se tratar de um registro de suma importância, é preciso muita cautela ao alterar seu conteúdo, pois apenas uma informação imprecisa pode comprometer todo o fluxo do programa.

O contador de programa é automaticamente incrementado a cada instrução concluída.

b) Stack Pointer

O Stack Pointer também chamado de apontador da pilha nada mais é do que um registro de 16 bits, cuja função específica é indicar ao processador em que endereço estão localizados os dados a serem utilizados como rascunho pelo próprio processador.

Muitas vezes estes dados guardam endereços de interrupção do programa antes que este entre em sub-rotinas.

Terminada a sub-rotina, o processador retorna ao endereço no Stack Pointer.

É como se o Stack Pointer fosse uma agenda cujos lembretes nela contidos fossem ininterruptamente consultados, mesmo após o retorno de um certo trabalho interrompido.

c) Index register IX e Index register IY

Tais registros, também chamados de registros indexados, são independentes apesar de possuírem a mesma finalidade.

Sua tarefa básica é realizar endereçamentos de memória indexada e como operam diretamente com a memória são registros de 16 bits.

No modo de endereçamento indexado, um registrador é usado como base para apontar de que região da memória o dado é retirado ou armazenado.

Um byte adicional é incluso nas instruções indexadas para, justamente, especificar o valor da base. Saibam que este valor especificado está na forma de complemento de 2.

O modo de endereçamento indexado é muito utilizado em bancos de dados.

d) Interrupt Register I.

Também chamado de registro de interrupção, é aquele utilizado quando uma chamada indireta para qualquer localização de memória pode ser obtida em resposta a uma interrupção.

O registro I contém 8 bits de alta ordem (mais significativos) no barramentos de endereços do microprocessador, enquanto que o periférico interruptor do programa fornece os 8 bits de baixa ordem (menos significativos).

Com esta técnica torna-se possível alocar dinamicamente rotinas em qualquer localização da memória.

e) Refresh Register (R)
Trata-se de um registro para "refrescar" a memória. Este contador está sempre sendo incrementado automaticamente. É um registro de 7 bits cuja finalidade é reciclar informações em memórias dinâmicas.
Saibam que este tipo de memória perde informação se dentro de um determinado espaço de tempo, seu conteúdo não for reciclado sucessivas vezes.
A informação contida dentro do registro R é enviada ao barramento de endereço, juntamente com um sinal de controle chamado refresh.
Este sinal torna-se ativo durante o intervalo de tempo em que o microprocessador decodifica ou executa uma instrução.

2.4 Unidade Lógica e Aritmética: ALU

Também chamada de ALU (Aritmetic Logic Unit) realiza operações aritméticas e lógicas tais como:

Soma
Subtração
Incremento
Comparação
Decremento
Rotação para a direita
Rotação para a esquerda
Lógica OR
Lógica OR exclusiva
Lógica And
Complemento

Internamente, o ALU se comunica com os demais registros, o barramento de dados interno e externo.
Embora o Z80 não execute diretamente operações de multiplicação e divisão, pode realizá-las através de circuítos auxiliares chamados processadores aritméticos (conforme foi visto na figura 2.1).

2.5 Registro de Instrução (IR)

O registro de instrução também chamado de "instruction register" armazena um determinado valor correspondente ao código de uma instrução. Esta por sua vez, vai para um circuíto chamado decodificador de instrução (instruction decoder), capaz de dividir o tempo do microcomputador para executar uma referência de atividades tais como: movimentar o conteúdo de um registgro para outro, carregar a memória com certos valores, ler, acionar dispositivos periféricos, etc.

O IR (instruction register) recebe valores provenientes da memória e armazena, não dados, mas apenas instruções, processadas posteriormente conforme visto na figura 2.1.

2.6 Barramento de dados e endereço

Existem 2 grandes barramentos, um de dados e outro de endereços, chamados respectivamente data bus e adress bus . São fundamentais para o perfeito funcionamento do Z80, pois são os meios de comunicação entre o micro e a memória ou dispositivo periféricos.
As linhas de endereço (address bus) são direcionais (16 linhas) ou seja, fluem sinais do micro em direção aos dispositivos periféricos.
Tais linhas contêm um valor hexadecimal referente ao endereço da memória ou periférico relacionado.
As linhas de dados (data bus) são linhas bidirecionais, ou seja, nelas transitam sinais tanto do micro para os periféricos, quanto dos periféricos para o micro.
Os sinais contidos no data bus (8 linhas) podem conter tanto informações de dados como instruções, tudo depende do tipo de sequência em que se apresentam tais informações.
A figura 2.1 mostra o data bus e o address bus.

2.7 Barramento de controle - "control bus"

O barramento de controle é formado por 13 linhas de controle divididas da seguinte forma:
a) Controle do Sistema (6 linhas)
Através das linhas de controle é possível controlar e temporizar os periféricos e a memória externa.
As duas principais linhas de controle do sistema são RD (read) e WR (write), ou seja, leitura e gravação, respectivamente.
Estes sinais de controle mostram à memória ou periférico, qual a direção e qual o momento em que os dados devem ser posicionados no barramento (data bus).
Quando o micro "lê" uma informaço na memória ou periférico, a linha RD é acionada com um pulso negativo.
Quando o micro "grava" uma informação na memória ou periférico, a linha WR é acionada com um pulso negativo.
Dessa forma, notem que os sinais WR e RD nunca são acionados simultaneamente, mesmo porque geram conflito de informações.
Duas outras linhas importantes são o MREQ (memory request) e o IORQ (input/output request).
A primeira delas (MREQ) quando ativa (pulso negativo), habilita a memória a realizar um ciclo de leitura ou gravação.
A linha IORQ quando ativa (pulso negativo), habilita os dispositivos periféricos a realizarem um ciclo de leitura ou gravação.

Finalmente, as outras 2 linhas de controle do sistema são M1 (machine cicle 1) e RFSH (refresh).
A linha M1 quando ativa (pulso negativo), informa aos dispositivos auxiliares que o Z80 está no seu 1º ciclo de máquina.
Este sinal é de extrema valia quando o Z80 está associado a outros dispositivos da mesma família.
O sinal RFSH é utilizado para "refrescar" ou realimentar as memórias dinâmicas, tal como mostra a figura 2.3.

b) controle do CPU (5 linhas)
O barramento de controle da CPU (unidade central de processamento) é formado por linhas que controlam diretamente o seu próprio funcionamento. São elas : Halt, Wait, Int, NMI e Reset.
A linha Halt é ativada a nível 0 sempre que o Z80 decodificar a instrução Halt, cujo código hexa é 76.
O micro em estado de Halt (parada) nada executa, a menos que seja ativada uma das linhas: Reset, Int ou NMI.
A segunda linha (Wait) é utilizada com periféricos lentos que "pedem" tempo ao Z80, enquanto enviam um nível lógico 0 para a linha Wait (espera), até que o dispositivo seja liberado e como conseqüência imediata, também o CPU.
Outras duas linhas dotadas de funções similares são a Int (interrupt) e a NMI (Mon Maskable Interrupt).
A função Int, como o próprio nome indica, é utilizada para interromper a CPU sempre que se encontra em rotina habitual, canalizando todo o fluxo do programa para atender a uma requisição de interrupção.
A linha NMI tem a mesma finalidade básica, entretanto, não pode ser "mascarada" ou programada, o que significa que, em qualquer hipótese, o programa pode ser interrompido para atender o dispositivo externo, gerador da interrupção, através da linha NMI.
Sintetizando, pode-se dizer que a linha Int é utilizada pela CPU condicionalmente, enquanto que a NMI incondicionalmente.

c) Controle de barramento da CPU (2 linhas)
Trata-se das linhas BUSRQ (Bus Request) e BUSAK (Bus Acknow Ledge), controladoras do barramento de dados e endereço, para que dispositivos externos possam ter acesso às mesmas linhas de dados e endereços do Z80.
Sempre que a linha BUSRQ é externamente ativada (nível 0), o Z80 executa a última instrução e imediatamente transfere o controle para o dispositivo externo, colocando em alta impedância (tri state) as linhas de dados, endereço e demais linhas de saída.
Posteriormente, a saída BUSAK vai a nível 0 e sinaliza os despositivos requisitantes do barramento, para imediatamente assumirem o controle.

Tal processo é muito utilizado em sistemas DMA "diret memory acess - acesso direto à memória", a fim de permitir que os dados provenientes da memória se dirijam para os

periféricos de alta velocidade, independentemente da interferência do Z80, resultando numa considerável econômia de tempo.
A figura 2.4 mostra a configuração dos barramentos.

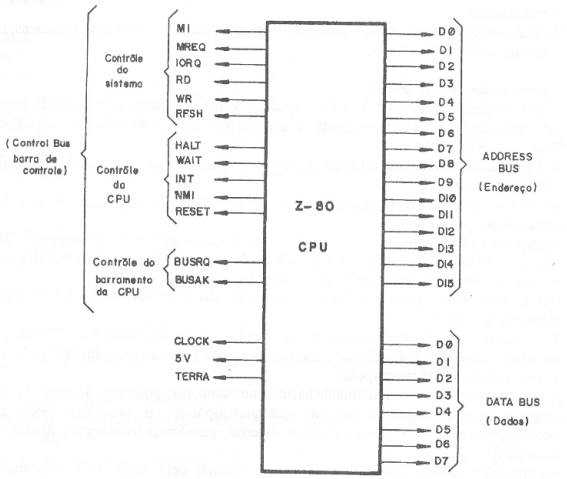


Figura 2.4 - Barramentos do Z80

2.8 Z80/PINAGEM:

O Z80 é um circuito integrado dotado de 40 pinos. A seguir, descreveremos a função básica de cada pino ou grupo de pinos. Antes porém, veja a figura 2.5.

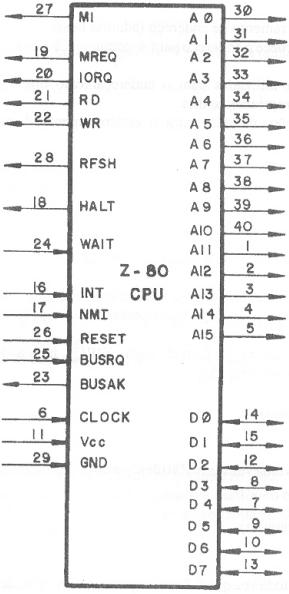


Figura 2.5 Pinagem de Z80 CPU

- D0 a D7: (data 0 a 7)
Entrada e saída em tri-state
Ativa em nível 1
Formam o barramento de 8 bits de dados (data bus).
São linhas bidirecionais com função de entrada e saída, destinadas à comunicação da memória e dispositivos periféricos.

- A0 a A15: (address 0 a 15)
Saída em tri-state
Ativa em nível 1
A0 a A15 formam o barramento de endereço (address bus).
O barramento (bus) fornece o endereço para a comunicação da memória e dispositivos de entrada e saída.
As linhas A0 a A7 são utilizadas para o endereçamento dos dispositivos de entrada e saída ou à memória propriamente dita.
As linhas A8 a A15 são utilizadas para o endereçamento da memória.
A0 é o LSB
A15 é o MSB

- M1: (machine cicle 1)
Saída
Ativa em nível 0
M1 é o indicador de que o ciclo de máquina corrente é o de busca (ciclo fetch) das instruções na memória .
Isto significa que cada vez que M1 for gerado, uma instrução será "buscada" na memória.
M1 poderá ocasionalmente ser gerado duas vezes quando o código da instrução também for de 2 bytes.
M1 também poderá aparecer juntamente com IORQ, indicando aos periféricos o reconhecimento de uma interrupção.

- MREQ: (memory request)
Saída em tri-state
Ativa em nível 0
O sinal MREQ é gerado toda vez que o Z80 desejar fazer um ciclo de leitura, ou gravação na memória externa.

- IORQ: (input-output request)
Saída em tri-state
Ativa em nível 0
O sinal IORQ é gerado toda vez que o Z80 desejar fazer um ciclo de leitura, ou gravação num dispositivo periférico.

- RD : (Read)
Saída em tri-state
Ativa em nível 0
O sinal RD é gerado no instante que o Z80 estiver lendo um dado da memória ou periférico.

-WR : (write)
Saída em tri state
Ativa em nível 0
O sinal WR é gerado no instante em que o Z80 estiver gravando um dado na memória ou periférico.

- RFSH: (refresh)
Saída
Atíva em nível 0
A saída RFSH quando ativa, indica à memória que os 7 bits menos significativos do address bus (A0 a A6), contêm o novo endereço de refrescamento para as memórias dinâmicas.
OBS: A0 a A6 contêm o valor do registro R, ao passo que A7 contém o nível lógico 0.

- Halt:
Saída
Ativa em nível 0
Halt indica que o Z80 executou uma instrução de Halt (código 76). Quando o Z80 permanece neste estado, fica aguardando os sinais de Reset, Int ou NMI.
Enquanto o Z80 permanecer em Halt, automaticamente fica refrescando as memórias dinâmicas.

- Wait:
Entrada
Ativa a nível 0
Este pino de entrada quando verdadeiro (nível 0), retarda o Z80, e o faz aguardar o término de um ciclo de leitura ou gravação num dispositivo periférico ou memória.
Este pino tem a finalidade de sincronizar a velocidade de qualquer memória ou periférico com a velocidade do Z80.

- Int: (interrupt)
Entrada
Ativa em nível 0
Este sinal deve ser gerado pelo dispositivo periférico sempre que necessitar do Z80 servi-lo com uma sub-rotina de interrupção.
Uma interrupção é atendida no fim da instrução em que ocorreu a interrupção. Quando

o Z80 aceita a interrupção, é gerado o sinal de reconhecimento IORQ + M1.

-NMI: (nom maskable interrupt)
Entrada
Sensível no flanco negativo do nível 1 para 0.
O sinal NMI quando gerado por um periférico, obriga Z80 terminar sua última instrução e salta para o endereço 0066 (hexa).
O valor do program counter (pc), é salvo, na memória, no endereço apontado pelo stack pointer (sp).
A entrada NMI tem prioridade sobre a entrada Int, e não é desabilitada pelo programa.

- Reset:
Entrada
Ativa a nível 1
O sinal externo de Reset habita o contador de programa (pc=zero) posiciona o Z80 no endereço 0000 da memória.
O Reset também zera os registros I e R, além de setar o modo 0 de interrupção.
Durante o Reset, o Address e Data Bus se posicionam no estado de alta impedância.

- Busrq: (bus request)
Entrada
Ativa a nível 0
Este sinal, deve ser gerado pelo periférico, sempre que este desejar obter acesso ao Bus do Z80.
Quando isto acontece, o Bus de endereço. dados e sinais de saída também vão para o estado de alta impedância.

-Busak (bus acknowledge)
Saída
Ativa nível 0
O sinal Busak é gerado pelo Z80 em resposta a uma requisição do Bus (Busrq).
Quando o Busak é gerado, indica que o data Bus, Address Bus e sinais de saída do Z80 estão em alta impedância, e fornece o controle ao periférico que gerou o Busak.

- Clock
Entrada de Clock fase única

CAPÍTULO 3

MICROCOMPUTADOR APL-80

(HARDWARE)

ÉRICA

3.1 - Apresentação

O microcomputador APL-80 é o módulo central destinado ao controle dos futuros circuitos aplicativos. É conveniente lembrar que este módulo sózinho, já torna possível "rodar" diversos programas, bem como pode ser interconectado a outros circuitos projetados pelo próprio leitor mais experiente.

Como o próprio nome diz (APL) trata-se de um microcomputador aplicativo destinado ao treinamento, controle, simulação, teste de equipamentos digitais, programação etc. As aplicações do APL-80 são ilimitadas, bastando para isto adaptá-lo para a condição desejada.

Neste capítulo nos deteremos apenas na descrição da parte eletrônica (hardware) do módulo, deixando a parte de programação, para o capítulo posterior, onde analisaremos com detalhes o programa monitor, que nada mais é do que o coração do sistema. Porém, para melhor entendimento da parte eletrônica, detalharemos microprogramas, relacionados diretamente com alguns circuitos integrados.

Embora o microcomputador APL-80 seja um circuito simples (cerca de 14 circuitos integrados), é um módulo bastante eficaz, podendo ser empregado em aplicações mais complexas, pois possui o que chamamos de "sistema mínimo de configuração". Como esta configuração mínima é padrão em microcomputadores, torna-se fácil adaptá-lo para outras configurações mais complexas.

3.2 CONFIGURAÇÃO DO APL-80

Nosso microcomputador APL-80 é composto por um microcomputador Z80, circuitos de reforço de níveis lógicos "buffers", gerador de clock a cristal, latches de entrada e saída para controle de teclado e display, circuitos de endereçamento, memórias, teclado e display.

Também possui um conector de 40 pinos que possibilita interligá-lo a outros dispositivos, tais como, os módulos que descreveremos em outros capítulos.

Todo o cojunto de circuitos descritos até agora, inclusive 6 displays e teclado de 24 teclas, são montados numa única placa.

Para montagem do conjunto sugerimos a utilização de placas de circuitos impresso padronizado tipo veroboard.

Ainda com referencia à montagem, existem algumas partes opcionais, que podem ser suprimidas, ou modificadas de acordo com a necessidade. Uma delas é o teclado que quando conectado pelo lado externo da placa, permite que diversos outros tipos de teclados possam ser ligados ou daptados. Outro exemplo de modificação, são os circuitos "buffers" que podem ser eliminados, quando o número de circuitos periféricos ligado ao APL for reduzido.

Através da figura 3.1 você pode ter uma noção da montagem da placa de circuito impresso, porém poderá fazê-la da forma que melhor lhe convier.

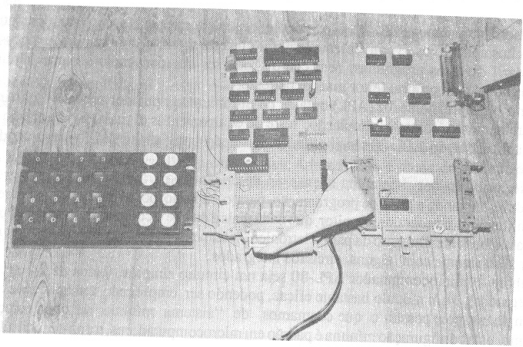


Figura 3.1 - Placa Completa do Microcomputador

3.3 Diagrama em Bloco:

Antes de entrarmos nos detalhes do funcionamento eletrônico, procuremos entender as funções básicas, em bloco, das partes mais importantes do circuito.
O diagrama em bloco geral do circuito está configurado na figura 3.2. Nela podemos observar 3 grandes barramentos principais:

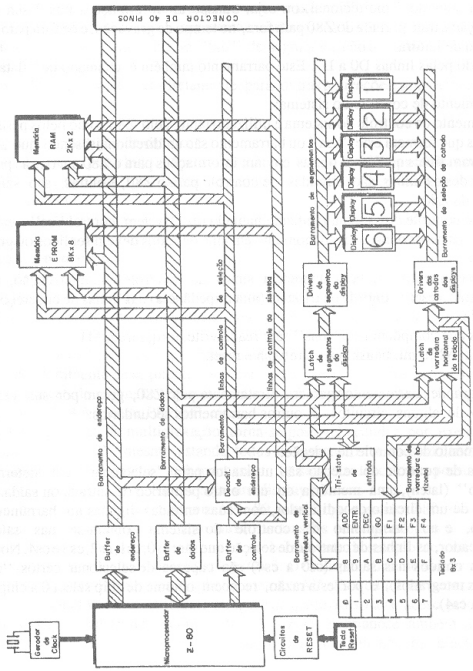


Figura 3.2 - Diagrama em Bloco Geral do Microcomputador APL-80

a) Barramento de endereço:
Este tem a finalidade de selecionar, ou endereçar memórias ou circuitos periféricos. É unidirecional, ou seja, somente transitam sinais em um sentido, (do micro para fora). É formado pelas linhas A0 a A15 e também são chamadas de "address bus"

b) Barramento de dados:
São os responsáveis pela transição de informações, entre o Z80 e as memórias ou periféricos.
Este barramento é bidirecional, ou seja, transporta informações nos 2 sentidos. A informação é transportada do Z80 para fora, no tempo de gravação, e de fora para dentro, no tempo de leitura.
É formado pelas linhas D0 a D7. Este barramento também é chamado de "data bus".

c) Barramento de controle do sistema:
O barramento de controle do sistema é utilizado tanto pelo Z80 como pelos periféricos. As linhas que formam este "bus" ou barramento são unidirecionais; sendo que algumas delas geram sinais no Z80 e outras enviam informações para o Z80 em outras palavras, algumas destas linhas, são entradas de controle para o Z80, e outras, são saídas do controle do Z80.
O bus de controle do sistema também chamado de "system control bus" serve para "avisar" os periféricos, o exato momento em que os dados devem ser lidos ou gravados pelo Z80.
Neste barramento também aparecem sinais de controle de interrupção, espera, sincronismo, reset e controle de tri state (alta impedância) das linhas de endereço, dados e controle.
Os sinais que compõem este "bus" são: read, write, iorq, mreq, M1
Halt, wait, NMI, int, buask busrq, refresh e reset.

Os três grandes barramentos gerados diretamente pelo Z80, geram por sua vez, com o auxílio de alguns circuitos, 5 outros barramentos secundários:

a) Barramento de controle de seleção:
Os sinais de controle de seleção são utilizados para "selecionar" um determinado "registro" (latch), uma memória ou um outro periférico de entrada ou saída.
Através de um circuito decodificador, com suas entradas ligadas ao barramento de endereço, e ao barramento de controle do sistema, obtêm-se nas saídas do decodificador, as linhas de controle de selação que são: cs0, cs1, cs2, cs3 e cs4. Note que, os sinais obtidos nas saídas (cs0 a cs4) são capazes de selecionar certos "chips" (circuitos integrados), e por esta razão, recebem o nome de chip select 0 a chip select 4 (cs0 a cs4).

b) Barramento de Segmentos:
É composto por 7 linhas (a, b, c, d, e, f, g) que polarizam os anôdos dos displays de catodo comum.
Estas linhas, "ligam" os 7 segmentos dos displays com tensão positiva, desde que haja tensão negativa nos respectivos catodos dos displays.
Existem dois fatos dignos de comentar neste tipo de barramento. Um deles, é que todos os segmentos, com a mesma denominação, estão em paralelo em todos os displays. Ou seja, o segmento "A" do 1º display está ligado ao "A" do 2º display, que por sua vez está ligado ao "A" do 3º, e assim sucessivamente até o último display (6º display).
Como nossos displays possuem 7 segmentos, então temos um barramento de 7 linhas.
O segundo fato a comentar, é que no "bus" de segmento, não transitam níveis lógicos, e sim tensões analógicas.
Portanto, não tente utilizá-las diretamente para outras aplicações, sem ser para excitar display.

c) Barramento de seleção de catodo:
Como você deve ter percebido, o barramento de segmento, está em paralelo com todos os displays, isto significa que, todos os anodos dos displays de mesma denominação, estão em paralelo.
Note que, se todas as linhas de segmentos estiverem positivas, e todos os catodos dos displays estiverem negativos, então se acenderão todos os segmentos do displays.
Para que o fenômeno descrito no último parágrafo não ocorra, empregamos a técnica de multiplexação, que nada mais é, do que "colocar" nas linhas de segmento, uma determinada combinação de tensões, (negativas para apagar e positivas para acender) formando, um algarismo numérico, alfanumérico ou um caractere qualquer de 7 segmentos.
Você já deve ter observado que mesmo configurando eletricamente um caractere qualquer no barramento de segmento, este caractere aparecerá em todos os displays que tiverem com o catodo na tensão negativa. Como temos seis displays, teremos algo como: 111111, 444444, AAAAAA, PPPPPP, etc.
Empregando a técnica da multiplexação, torna-se possível acender, por exemplo, "a mensagem" "APL-80", mesmo estando todos os segmentos de mesma denominação curto-circuítados. Isto porque o barramento de seleção de catodo, possui 6 fios, selecionando um catodo por vez, ou seja, um display por vez.
Para acender por exemplo, a mensagem APL-80, o microprocessador Z80, deve enviar inicialmente, ao barramento de segmentos, as tensões correspondentes ao caractere "A". Simultaneamente, o Z80, deve enviar ao barramento de seleção de catodo, uma tensão negativa para o catodo do 6º display (OBS: O 6º display fica do lado esquerdo). Feito isto, o Z80 acende num dos displays, a letra "A". Esta condição permanece por alguns milissegundos. Terminado este tempo, é apagado o display e aceso pelo mesmo processo, o display subsequente com a letra "P". Este, permanece também aceso por alguns milissegundos. Terminado este tempo, o ciclo se repete para os caracteres L, -, 8, 0.

Note que o Z80 repete a mensagem APL-80 ciclicamente, mantendo uma velocidade de multiplexação superior ao da inércia visual (24 Hertz), dando assim, a impressão de que a mensagem está fixa, quando na realidade está piscando.

d) Barramento de varredura horizontal:
Este barramento, é formado por 8 linhas, com a finalidade de "varrer a matriz do teclado (apenas linhas horizontais).
Note que o mesmo latch utilizado para "varrer" o barramento de seleção do catodo, é utilizado para varrer as linhas horizontais do teclado e que o barramento de seleção de catodo, está "isolado" por um circuito driver.
Foi utilizado um latch em comum para os 2 barramentos apenas por motivo de economia de componentes.
Outro aspecto importante com referência a estes 2 barramentos, é que apesar de estarem intimamente conectados (eletricamente), funcionam em tempos diferentes, e não geram conflito de informações. Resumindo, toda vez que o teclado estiver sendo varrido, o display é desligado pelos "segmentos" e vice-versa.
Mais adiante, na explicação do circuito do teclado, ver-se-á melhor o funcionamento deste conjunto.

e) Barramento de varredura vertical.
Este barramento é composto por apenas 3 linhas, responsáveis pela "leitura" propriamente dita do teclado.
Toda vez que uma linha horizontal é varrida (saída do Z80), uma linha vertical é lida (entrada do Z80). Comparando a linha vertical com a horizontal, e constatando que a tecla está acionada, o programa é capaz de determinar qual o código da respectiva tecla.

Agora que já estudamos os barramentos principais e secundários do APL-80, estamos aptos a estudar os blocos principais deste microcomputador.

a) Gerador de clock:
É o circuito responsável pela sequência e temporização de todo o circuito.
A parte mais importantes do gerador de clock é o cristal de quartzo. Utilizamos este tipo de componente devido à sua alta estabilidade de frequência.
Para diminuir o custo final do APL-80 lançamos mão de um cristal "barato" e de fácil aquisição, utilizado em TV a cores com frequência de 3.57 mhz.

b) Microprocessador Z80:
O microprocessador é o coração do sistema, sem ele, nada seria possível. Neste circuito, ele age como supervisor de fluxo de entrada (teclado), e saída (display) de dados, podendo também, dependendo do programa, controlar circuitos externos, tais como uma impressora, um outro teclado maior, um conjunto de relés, lâmpadas etc. Logicamente tudo isto é realizado pelo conector de interface de 40 pinos.
A função primária e básica do Z80 no APL-80, é a de ler dados do teclado, interpretá-los, e posteriormente colocar um determinado resultado no display.

c) Circuítos de reset:
No bloco "circuitos de reset" existem 2 tipos de reset um deles é automático, pois é operável independentemente de intervenção manual. Toda vez que o circuito geral do APL-80 é energizado, o sinal reset do Z80 fica forçado a nível 0, ou seja, torna-se ativo. Este nível ativo persiste por alguns milissegundos, zerando o Z80 e posicionando seus registros internos.
A outra forma de resetar o Z80, é através de uma chave (incluída no APL-80. Este processo de resetar o Z80 "a qualquer momento " através de uma chave manual, é chamado de "reset manual".

d)Buffer de endereço:
O significado da palavra "buffer" em eletrônica, é "reforço". Portanto o buffer de endereço é formado por um conjunto de circuitos com a finalidade específica de reforçar as 16 linhas de endereço.
Uma observação importante a salientar, é que, todos os buffers do APL-80, podem ser omitidos sem comprometimento do funcionamento do sistema. Porém, problemas podem ocorrer quando "interfaceamos" muitos periféricos ao sistema.
Caso o APL-80 seja utilizado apenas para "rodar" programas isoladamente, (sem nada a ele conectado), pode-se perfeitamente omitir os buffers.

e) Buffer de dados:
Um outro grupo de buffers, está conectado diretamente às linhas de dados, reforçando assim, não só os dados que entram no Z80, como também os que saem, formando assim o buffer de dados. Este buffer é bidirecional, e por este motivo, difere-se dos demais circuitos destinados a este propósito. Veja mais adiante na descrição eletrônica do circuito APL-80.

f) Buffer de controle
Este tipo de buffer tem a mesma configuração do de endereço, apenas é utilizado menor número de circuitos integrados, para realizar a função correspondente.
A finalidade principal deste circuito é reforçar "apenas" as linhas de controle de saída do Z80. As linhas de controle de entrada, não são "reforçadas" por motivo de economia de circuitos integrados, porém, caso necessário, poderão ser reforçadas nos próprios circuitos periféricos. Podemos adiantar que este tipo de procedimento é pouco comum pois, os circuitos periféricos possuem potência suficiente para excitar o Z80.

g) Decodificador de endereço:
Existem vários registros (latch), espalhados ao largo do conjunto de componentes, que formam o APL-80, e por esta razão, torna-se necessário endereçá-los ou selecioná-los individualmente. Para possibilitar essa operação, utilizamos o "decodificador de endereço," capaz de selecionar circuitos com "endereços" pré-estabelecidos eletricamente.
O caso do APL-80, o decodificador de endereço seleciona os dois circuitos de memória, o latch de segmentos do display, o latch de varredura horizontal e o tri-state de entrada.

h) Tri-state de entrada:
A palavra "tri-state", significa três estados, é utilizada, com referência a circuitos, e se caracteriza pela capacidade de isolar um circuito lógico dos demais.
O "tri-state" de entrada, lê apenas o teclado no tempo programado pelo Z80. Esta leitura é realizada através do "data bus" no tempo fornecido pelo decodificador de endereço.
Embora o tri state de entrada, neste caso, seja utilizado para ler as linhas verticais do teclado, você o verá mais adiante, em outros capítulos, como entrada de circuitos periféricos.

i) Latch de segmentos do display:
Este latch, é formado por um registro, com a habilidade de "guardar" informações, que no devido tempo será utilizado pelos segmentos dos displays.
De acordo com o código binário contido neste registro, torna-se possível formar caracteres numéricos, alfabéticos ou mesmo símbolos no display lembre-se que o código guardado neste "latch, não é o mesmo código utilizado para representar números e letras hexadecimais. Mais adiante, à combinação binária necessária para acender um display, chamaremos de código de sete segmentos.

j) Latch de varredura horizontal do teclado:
Este segundo latch, tem função similar à descrita anteriormente, ou seja, "guarda" uma certa informação binária, para selecionar, um determinado "catodo" do display, ou uma determinada linha horizontal do teclado.
Observe no diagrama em bloco, que o latch de varredura horizontal do teclado, é utilizado, tanto para varrer o teclado horizontalmente, como para selecionar o catodo de um ou mais
displays.

k) Driver de segmentos do display:
A palavra "driver" tem a mesma tradução da palavra "buffer", ou seja, ambas servem para reforçar um certo sinal. Porém, ambas se diferem na "quantidade" de corrente reforçada.
O "buffer", é utilizado na maioria dos casos, para reforçar sinais digitais, onde uma certa linha digital de saída deve ser ligada a uma grande quantidade de circuitos. Um exemplo típico seria o data bus e o address bus.
O "driver", é utilizado na maioria dos casos, para reforçar sinais digitais, além de convertê-los para analógicos.
Um exemplo, seria um circuito driver com "força" suficiente para excitar um display. Circuítos drivers, são normalmente utilizados, junto com relés, lâmpadas, transistores de potência etc.
Pode ocorrer, a possibilidade do leitor encontar circuitos drivers, utilizados como buffers. Quando isto acontece, significa que o circuito buffer, não reuniu "potência" suficiente para alimentar ou reforçar o sinal digital, e por este motivo é que foi substituído por um "driver".

Resumindo:
Buffer - Baixa potência - digital para digital
Driver - Alta potência - digital para digital ou analógico.
No caso do nosso diagrama em bloco, o driver de segmentos do display, transforma o sinal proveniente do seu respectivo latch, reforçando-o e adaptando-o para alimentar os 7 segmentos dos displays.

l) Driver dos catodos/displays:
Este diagrama em bloco, tem a mesma função do "driver de segmento", diferindo apenas quanto à polaridade da tensão.
O driver de segmento excita os displays com tensão positiva, ao passo que o driver de catodo excita os displays com tensão negativa.
Note também, que o "driver dos catodos", é alimentado pelo "latch de varredura horizontal do teclado", porém, apesar do teclado utilizar todos os 8 bits (todas as linhas) do latch, o "driver dos catodos", utiliza apenas 6 linhas, mesmo porque só existem 6 displays disponíveis para a excitação.

m) Memória Eprom 8k x 8.
Embora, o coração do microcomputador APL-80, consista no microprocessador Z80, a memória Eprom é que dá a "vida" ao sistema todo.
Na memoria é que estão gravados os programas, que informam ao Z80 como ele deve funcionar. Se portanto, retirarmos a memória, ou mesmo, posicionarmos incorretamente um Byte por ocasião da Programação, o resultado do funcionamento do conjunto APL-80 será desastroso.
Para que o APL-80 possa funcionar adequadamente, a memória deverá estar corretamente gravada com o programa monitor. Este será detalhado mais adiante, no próximo capítulo, porém podemos adiantar que sua função básica é endereçar, ler e gravar dados na própria memória do APL-80, possibilitando então gerar programas, bem como executá-los.

n) Memória Ram 2k x 8.
Muitas vezes desejamos gravar dados, ou instruções. Isto não é possível fazer na memória descrita anteriormente, pois trata-se de uma Eprom, ou seja memória somente de leitura. Por esta razão, é necessário acrescentar ao conjunto, uma memória Ram, que tanto pode ser gravada ou lida.
Ram vem do inglês "Random Acess Memory", que significa, Memória de Acesso Aleatório, e não tem nenhuma relação com gravação ou leitura. A Eprom, é também uma memória de acesso aleatório, porém, como já dissemos, só pode ser gravada.
Embora a escolha do nome Ram, seja um tanto infeliz, convencionou-se "relacionar" o termo Ram, com memória de leitura e gravação.
Uma Eprom (erasable programmable read only memory), como o próprio nome diz, é uma memória apagável e programável (somente de leitura). Sua vantagem principal, e que mantém as informações, independente do fornecimento da tensão de alimentação.

Sua desvantagem principal esta no fato de que não pode ser gravada normalmente como a Ram; para gravá-la é necessário o uso de programadores especiais.
A vantagem principal da Ram, é poder ser lida ou gravada normalmente.
A desvantagem principal da Ram é não conseguir manter a informação após o corte da tensão de alimentação.
O microprocessador não sabe quando está "trabalhando" com uma memória Ram ou Eprom, cabe ao programador separar e classificar os tipos de memórias utilizadas.
Como não existe um circuito integrado capaz de ler, gravar e "não volatizar", informações com a queda de tensão, então é necessário individualizar as memórias quanto aos tipos utilizados.

o) Display:
Toda informação processada, tanto pelo programa monitor, como pelo programa contido na Ram, deve retornar as informações pelo display (visor). Pode acontecer em alguns casos, que a informação do display, seja proveniente de um circuito periférico. Quando isto ocorrer, significa que a informação do display foi processada pelo microprocessador.
Lembre-se que nunca um dado do display, pode vir de outro circuito, sem passar pelo Z80.
O Display do APL-80 é formado por 6 dígitos de sete segmentos, podendo formar, além dos códigos hexadecimais, algumas letras do alfabeto ou caracteres especiais.
A placa de circuito foi projetada para abrigar diretamente um determinado tipo de display, porém caso seja necessário, poderão ser utilizados outros displays em placas separadas, não comprometendo o funcionamento do circuito.
Para realizar esta operação, é importante providenciar um cabo com 13 fios, soldado diretamente às placas. Embora seja totalmente possível "puxar" para fora os displays, o fato compromete o manuseio da placa e dificulta sua respectiva manutenção.

p) Teclado:
Trata-se de um conjunto de 28 teclas congiguradas numa matriz 8x3, sendo 8 linhas horizontais e 3 verticais.
Neste arranjo, as teclas estão dimensionadas de forma que as hexadecimais (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F) estejam separadas das de função (ADD, ENTR, DEC, GO, F1, F2, F3, F4).
As teclas F1,F2,F3 e F4, são de reserva e podem ser programadas pelo leitor, caso necessário.
Tal como acontece com o display, existe também a possibilidade de se utilizar um teclado externo, bastando para isto, "puxar" 12 fios para fora da placa.
Toda vez que um programa é carregado na memória através do teclado, grande é a quantidade de botões acionados e por esta razão recomendamos o uso de teclas da melhor qualidade.

Ainda um fato digno de ser comentado é com referência ao teclado, onde uma sub-rotina especial é embutida no programa monitor, com a finalidade de mascarar as trepidações mecânicas, provenientes das teclas, principalmente as de baixa qualidade, melhorando enormemente a digitação.

q) Conector de 40 pinos:
Todos os pinos do Z80, podem ter acesso ao exterior da placa APL-80, através do conector de 40 pinos. Contudo torna-se possível expandir a memória, bem como interconectar outros equipamentos ao APL-80.
Por intermédio deste conector, serão interconectados os demais módulos descritos em outros capítulos.
Convém lembrar que os demais sinais internos e barramentos secundários, não estão presentes neste conector.
Apesar das linhas de alimentação estarem presentes no conector de 40 pinos (conector 2), aparecem também no conector de 2 pinos (conector 1).
O conector 1 é utilizado para receber a tensão da fonte de alimentação externa de 5 volts.

3.4 O CIRCUITO:

Para maior facilidade de manuseio e entendimento, o circuito do APL-80 está dividido em 6 páginas. Cada uma delas contém o maior número possível de partes inter-relacionadas.
A seguir, abordaremos o circuito ou grupo de circuitos referente a cada página ou figura:
A figura 3.3 é formada pelos seguintes circuitos:
- CPU ou microprocessador
O CPU (central processing unit) é a unidade central de processamento de todo sistema. É formado pelo circuito integrado IC-2 (Z80A).
Para tanto, note no seu lado direito 2 dos seus barramentos principais, o de endereço e o de dados.

O barramento de endereço é formado pelas linhas A0, A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14 e A15, cujos pinos são respectivamente : 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 1, 2, 3, 4 e 5.
O barramento de dados é formado pelas linhas D0, D1, D2, D3, D4, D5, D6 e D7, cujos pinos são respectivamente: 14, 15, 12, 8, 7, 9, 10 e 13.

Agora vejamos o lado esquerdo do Z80 (CI-2), com 6 entradas de sinais e 8 saídas de sinais.
Os principais sinais de entrada utilizados são: o reset (pino 26) e o clock (pinos 6). Note que, exceto o sinal clock, todos os sinais, possuem uma barra, validando-os apenas no nível zero.

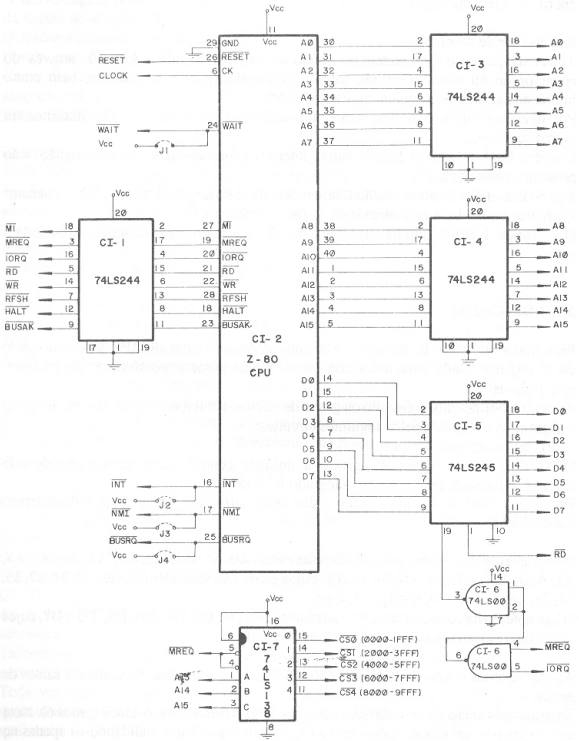


Figura 3.3 - Microcomputador APL80 CPU/BUFFERS/ADDRESS DECODER

Tanto o sinal clock como o reset são gerados na Fig. 3.5, portanto serão estudados mais adiante.
Os outros sinais de entrada "não utilizados" são o wait (pino 24), int (pino 16), NMI (pino 17) e o busrq (pino 25). Observe que estes pinos, estão interligados aos 5 volts (vcc) através dos "jumpers" J1, J2, J3 e J4. Caso o leitor queira utilizá-los, basta retirar o "jumper" correspondente e servir-se deste sinal, localizado no conector de 40 pinos. Portanto de acordo com a figura 3.4 temos a Associação dos jumpers com os sinais.

JUMPER	SINAL
J1	$\overline{\text{WAIT}}$
J2	$\overline{\text{INT}}$
J3	$\overline{\text{NMI}}$
J4	$\overline{\text{BUSRQ}}$

Figura 3.4 - Jumpers Associados aos Sinais

Os sinais de saída "utilizados" são mreq (pino 19), RD (pino 21) e WR (pino 22).
Os sinais de saída "nào utilizados" são M1 (pino 27), iorq (pino 20), RFSH (pino 28), Halt (pino 18) e busak (pino 23). Estes, apesar de não serem utilizados pelo APL-80, estão disponíveis no conector de 40 pinos.
Finalmente, o Z-80 possui seus pinos de alimentação VCC (pino 11) e GND (pino 29) ligados tanto no conector 1, como no conector 2. Nestes pinos deve ser conectada a tensão de 5 volts, conforme Figura 3.14.

- Buffer de endereço
O circuito integrado 74LS244 é um buffer direcional de 8 bits. São utilizados 2 destes integrados para "reforçar" os sinais do buffer de endereço.
O CI-3 reforça as linhas A0, A1, A2, A3, A4, A5, A6 e A7.
O CI-4 reforça as linhas A8, A9, A10, A11, A12, A13, A14 e A15.

- Buffer de dados:
O circuito integrado CI-5 (74LS245) é o responsável pelo reforço das linhas D0, D1, D2, D3, D4, D5, D6 e D7.
Note que o integrado 74LS245, é um buffer bidirecional e, por esta razão, é controlado pelas portas lógicas que compõem o CI-6:
O CI-5 possui 2 pinos de controle, o pino 19 e o pino 1.
O pino 19 controla "o momento" em que os sinais devem passar ou sair do próprio CI-5; isto significa que é uma espécie de chave geral. Em outras palavras, é o controle do tri-state, que isola ou não, o Z-80 do restante dos circuitos.
Quando o Z-80 desejar "trabalhar" com as memórias ou periféricos, gerará os sinais, mreq (memória) ou iorq (periféricos). Quando isto ocorrer, aparecerá um nível 0 no pino 4 ou no pino 5 do CI-6. Isto resultará num nível 1 no pino 6 de CI-6.

Como CI-6 pino 6 esta conectado a CI-6 pino 1 e 4, então, resultará num nível 0 na saída da porta (CI-6 - pino 3).
O nível 0 da saída da porta CI-6 pino 3 vai para a entrada de controle tri-state (pino 19) do CI-5, permitindo assim que o fluxo de dados possa sair ou entrar no Z-80.
Até agora, o pino 19 do CI-5 habilitou ou não o fluxo de dados do barramento de dados do Z80 para o barramento de dados do APL-80. Resta agora determinar a direção dos dados.
O pino 1 (CI-5) é responsável pelo controle da direção dos dados no "data bus". Assim, quando o Z80 "quiser" ler dados, a direção será de fora para dentro do Z80. Nesta condição, o microprocessador torna ativo o sinal RD (nível 0). Como o pino 1 de CI-5 está ligado ao sinal RD, Conclui-se que a direção do fluxo de dados será dos pinos 18, 17, 16, 15, 14, 13, 12 e 11 para 2, 3, 4, 5, 6, 7, 8 e 9, ou seja, de fora para dentro do Z80.
Quando o microprocessador executar uma gravação, o sinal RD permanecerá inativo (nível 1), orientando ao CI-5 que o fluxo de dados é do Z-80 para fora, ou seja, o CI-5 estará com sinais, transitando dos pinos 2, 3, 4, 5, 6, 7, 8 e 9 para 18, 17, 16, 15, 14, 13, 12, e 11 respectivamente.

- Circuíto decodificador de endereço

Normalmente, a seleção de memórias, e a selação dos periféricos, é feita separadamente, através de 2 circuitos decodificadores distintos. Um dos circuitos (o que endereça as memórias), recebe o sinal mreq, enquanto que, o outro circuito (o que endereça os periféricos), recebe o sinal iorq.
Desta maneira, podem ser gerados por exemplo, dois endereços com o mesmo número, sendo um da memória e outro do periférico.
A desvantagem deste processo está na utilização de circuitos separados. Contudo, existe uma técnica denominada "memory mapped I/O" que permite o endereçamento de periféricos e mémorias no mesmo circuito de decodificação de endereço.
Este processo é muito simples, além de econômico, pois utiliza apenas 1 circuito para executar 2 tarefas diferentes, e tem a desvantagem de sacrificar um endereço de memória para cada periférico utilizado.
Optamos por este processo devido à disponibilidade da memória utilizada.
Um exemplo típico de endereçamento misto de memória e periférico, é realizado no circuito do APL-80, através do circuito integrado CI-7 (74LS138). Trata-se de um decodificador capaz de decodificar 3 sinais binários (pinos 1, 2 e 3) em 8 saídas decimais, das quais utilizamos apenas 5 (pinos 15, 14, 13, 12 e 11).
Este decodificador só produz uma saída ativa (nível 0) quando sua entrada de controle estiver ativa, ou seja, pinos 4 e 5 e nível 0 e pino 6 a nível 1. Note que na entrada de controle invertida (pino 4 e 5) é injetado o sinal mreq. Este, habilita o decodificador e gera uma saída ativa, somente durante o tempo que o microprocessador estiver ligando com memórias.
O decodificador CI-7 gera 5 saídas utilizadas pelo APL-80. Duas delas são utilizadas na seleção das memórias Eprom (CS0) e Ram (CS1). As demais saídas (CS2, CS3 e CS4) são utilizadas para selecionar periféricos e impedem a presença de memórias nos endereços 4000, 5000 e 8000 hexadecimais.

Como a maior memória utilizada no APL-80 ocupa 8k de endereços (8096 Bytes), então configuramos o CI-7 de forma a selecionar uma saída diferente, em sequência, a cada 8k. Desta forma, temos então a saída CS0 que vai de 0000 a 1FFF, a saída CS1 que vai de 2000 a 3FFF, a saída CS2 que vai de 4000 a 5FFF, a saída CS3 que vai de 6000 a 7FFF e finalmente a saída CS4 que vai de 8000 a 9FFF.
A figura 3.5 mostra como está dividido o endereçamento dentro do APL-80.

ENDEREÇOS

BINÁRIO																HEXA
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFF
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFF
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000
0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	5FFF
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6000
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7FFF
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000
1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	9FFF

Figura 3.5 - Tabela de Endereçamento Utilizado no APL-80

Note que cada grupo de 4 bits forma um byte hexadecimal. Como temos 16 bits (A0 a A15), então o resultado será 4 dígitos hexadecimais.
Como o sistema de endereçamento todo, está "amarrado" à capacidade da maior memória (8k), então, para cada passo de endereçamento deve ser somado 2000 (8k). Portanto temos os seguintes endereços (início): 0000, 2000, 4000, 6000 e 8000.
Obtém-se o término de cada endereço, fazendo-se somas da seguinte forma:

1FFF (2000-1=1FFF) a cada início de endereço.

Logo temos:
0000+1FFF=1FFF
2000+1FFF=3FFF
4000+1FFF=5FFF
6000+1FFF=7FFF
8000+1FFF=9FFF

As linhas que endereçam a memória de 8k são A0 a A12, e formam o barramento das memórias. As linhas A13, A14 e A15 sobram, para decodificar o grupo de 8k de endereço (cada memória).
Para tanto, observe a tabela da figura 3.6 onde as linhas A13 a A15 só variam a cada 8k completos. Veja também, nesta mesma tabela, todos os pinos do CI-7 envolvidos nesta decodificação, com o próprosito de gerar os sinais CS0 a CS4.

Entradas de CI- 5			Saída Ativa do CI-7	Número do pino do CI-7	Sinal gerado	Faixa de endereço
Pino 3	Pino 2	Pino 1				
C	B	A				
A 15	A 14	A 13				
0	0	0	0	15	$\overline{CS0}$	0000
0	0	0	0	15	$\overline{CS0}$	1FFF
0	0	1	1	14	$\overline{CS1}$	2000
0	0	1	1	14	$\overline{CS1}$	3FFF
0	1	0	2	13	$\overline{CS2}$	4000
0	1	0	2	13	$\overline{CS2}$	5FFF
0	1	1	3	12	$\overline{CS3}$	6000
0	1	1	3	12	$\overline{CS3}$	7FFF
1	0	0	4	11	$\overline{CS4}$	8000
1	0	0	4	11	$\overline{CS4}$	9FFF

Figura 3.6 - Decodificador CI-7

Lembre-se que os sinais CS (chip select) só se tornarão ativos no tempo que for ativo o sinal mreq, habilitando então o endereçamento das memórias ou periféricos. Mais adiante, você verá para onde se dirigem os sinais CS 0 a CS4.

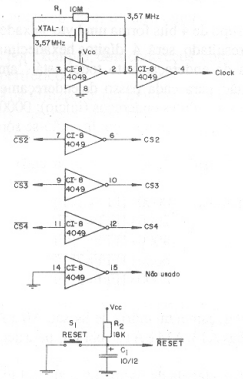


Figura 3.7 - Microcomputador APL-80 - Clock/Reset

Agora que você está familiarizado com o CPU, veja seus buffers e sistemas de endereçamento, todos contidos na Fig. 3.3.
A Figura 3.7 mostra alguns circuitos auxiliares complementares para o funcionamento do Z80:

- Gerador de clock

Para gerar uma onda quadrada necessária ao sincronismo do Z80, utilizamos o gerador de clock formado pelos 2 primeiros inversores do CI-8 (pinos 3, 2 e 5,4).
A oscilação do circuito é mantida pelo cristal (xtal-1), ligado entre a entrada e a saída do primeiro inversor (pinos 3,2). O resistor R1 mantém a polarização necessária para realimentar o circuito.
O segundo inversor (CI-8 pinos 5,4) amplifica e reforça o sinal oriundo do inversor anterior. Note que a saída clock (CI-8pino 4) contém a onda quadrada e frequência necessária para excitar o Z80, portanto o pino 4 de CI-8 vai diretamente para o CI-2 pino 6 da figura 3.3.
O cristal (xtal-1) é que determina a frequência do clock. Optamos pelo cristal de 3.57 mhz por ser mais facilmente obtido no comércio especializado, além de ter um custo muito mais acessível.
Outros cristais podem ser utilizados sem necessidade de modificar o circuito, contudo convém salientar, que existem várias versões do Z80, que determinam a frequência máxima de operação.
As versões de Z80 CPU mais encontradas no Brasil, são o Z80 cuja frequência máxima é 2.5 mhz, o Z80A para frequência máxima de 4.0 mhz e o Z80B para frequência máxima de 6.0 mhz.
Portanto, conclui-se que, a frequência de 3.57 mhz, está dentro da faixa de operação do Z80A, proposta no esquema da figura 3.3.

- Buffer inversores auxiliares

Logo abaixo do gerador de clock, pode ser notado um conjunto de 4 inversores, cuja finalidade é inverter e reforçar as saídas CS2, CS3, Cs4, provenientes do decodificador de endereço (CI-7) da figura 3.3.
Os três sinais CS2, Cs3 e Cs4 são invertidos a fim de que se compatibilizem com os latches, descritos pouco mais adiante.
O último inversor (CI-8 pinos 14, 15) não é utilizado. Trata-se de um circuito integrado CMOS cuja característica não admite entradas em aberto e por esta razão é que o pino de entrada CI-8 pino 14 está ligado a terra.

- Circuítos de reset

Logo abaixo dos inversores dos sinais CS2, CS3 e CS4, aparece o circuito gerador de reset.
O sinal reset é ativo quando a nível zero, com a finalidade de zerar registros do Z80, e posicioná-lo para um ciclo inicial. Note que o sinal vai para o pino 26 do Z80 (CI-2) da figura 3.3.

Uma outra finalidade do sinal reset é zerar o contador de programa (PC) no microprocessador e desta forma, o programa inicia no endereço 0000 sempre que o sinal reset e' gerado.

Existem 2 formas de gerar o sinal reset. A primeira delas seria acionando a tecla S1. Quando isto acontece,um nível 0 passa pela chave e gera diretamente o sinal reset. Observe que um dos pólos da chave S1 está conectado diretamente a terra (nível 0).

A segunda maneira de gerar o sinal reset, é automática e isto acontece quando energiza-se o APL-80 com os 5 volts de alimentação. Vejamos como isto ocorre.

Para melhor entendimento, suponha que o capacitor C1 esteja descarregado inicialmente. Ao aplicarmos uma tensão de alimentação no circuito, o capacitor C1, é carregado lentamente pelo resistor R2, que por vez está conectado ao VCC (pólo positivo).

Note que, durante o tempo em que o capacitor estiver carregado, o sinal reset ficará ativo (nível zero). Dependendo do valor de C1 e do resistor R2, o tempo de carga de C1 será maior ou menor. Com os valores indicados no circuito (18k e 10 μF), consegue-se um tempo aproximado de 300 milissegundos.

Outra finalidade do capacitor C1, é eliminar picos, provenientes da chave S1, durante o reset manual.

Como o sinal reset está conectado diretamente no conector de 40 pinos, então este pode ser "injetado" diretamente através de circuitos externos. Quando houver essa necessidade, o resistor R2 e o capacitor C1 devem ser removidos do circuito.

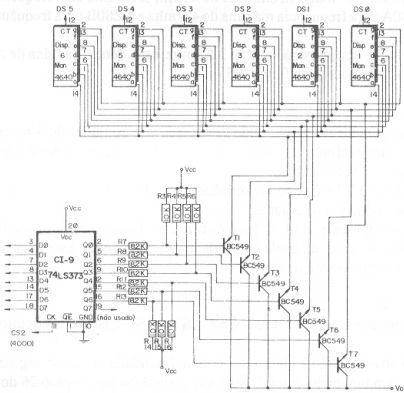


Figura 3.8 - Microcomputador APL - Circuítos do Display

O latch de 7 segmentos, os drivers de 7 segmentos e os displays formam o conjunto de circuitos contidos na figura 3.8.

- Latch de 7 segmentos

O circuito integrado CI-9 compõe o latch de 7 segmentos trata-se de um registro (latch) de 8 bits, dos quais são utilizados apenas 7, pois temos somente 7 segmentos para excitar.
O circuito integrado que realiza tal função é o 74LS373 (CI-9). Note, na lateral esquerda as linhas D0 a D7 (data bus) chegando nos pinos de entrada 3, 4, 7, 8, 13, 14, 17 e 18. Os sinais do data bus só "entram" e se "fixam" no 74LS373, quando o nível lógico no pino 11 (clock) variar de "zero" para "um" isto é realizado pelo sinal CS2, proveniente do circuito decodificador de endereço (figura 3.3 CI-7 pino 13). Veja que o sinal sai da figura 3.3 invertido e torna a inverter através do inversor CI-8 pinos 7 e 6 (figura 3.7), para definitivamente chegar em CI-9 pino 11.
O sinal CS2 só aparece quando o microprocessador executar endereços com o código 4000 no barramento de endereço.
Como já dissemos anteriormente, para o microprocessador, o endereçamento é transparente; ele não é capaz de distinguir se está endereçando memória, ou outro elemento lógico qualquer.
Portanto nossa posição de "memória" 4000, corresponde a um latch de 8 bits.
Uma instrução que grava um código de 8 bits no latch situado no endereço 4000 seria a seguinte:

LD (4000), A

Esta instrução, lê o conteúdo do acumulador, que já deve ter sido carregado anteriormente, com um código de 7 segmentos.
Em seguida, transfere o conteúdo do acumulador, para o latch CI-9.
Agora que já estudamos a entrada do latch, vejamos suas saídas localizadas no lado direito.
Os pinos 2, 5, 6, 9, 12, 15, 16 e 19 formam as saídas Q0 a Q7 do latch. Estas contêm a informação de 8 bits fornecida pelo microprocessador. É importante comentar, que as saídas de CI-9, são do tipo tri-state e por esta razão, o pino de controle de saída (OE) "output enable", deve ser mantido a nível zero, caso contrário as saídas ficarão em alta impedância ou flutuando.
Note que a saída Q7 (pino 19) não é utilizada, mesmo assim, o zero "manda" a informação referente a este bit.

- Driver de sete segmentos

Para que os segmentos possam acender corretamente, é necessário manter a corrente exata nos 7 anodos. Para tanto, utilizamos o circuito "driver", formado pelos transitores T1 aT7, e os resistores R3 a R16.
O circuito driver opera como um reforçador de corrente das saídas de CI-9 seu funcionamento é relativamente simples.
O sinal positivo (+5 volts) fornecido pelas saídas "Q" do latch CI-9, é limitado pela corrente nos resistores R7 a R13.

Estes sinais, passam pelos resistores de "pull up" R3, R4, R5, R6, R14, R15 e R16, reforçando a tensão positiva dos sinais, a fim de que possam ser entregues à base do transistor com a tensão correta.
Uma vez que o sinal "positivo" chegou do transistor, este entra em saturação (T1 a T7), conduzindo a tensão de coletor (VCC) para o emissor. A tensão positiva do emissor vai para os anodos dos displays de leds, contudo, só acendem se os catodos estiverem aterrados.

- Displays

Os 6 displays disp 1 a disp 6 estão conectados em paralelo, formando o barramento de 7 segmentos excitado pelos drivers a transistores, conforme discrição anterior observe que os catodos dos displays não estão em paralelo, pois cada display, é controlado individualmente pelo circuito "driver dos catodos", descrito a seguir.
Para maior flexibilidade de montagem, optou-se pelo uso de displays a led de 7 segmentos, do tipo man 4640. A vantagem desses componentes está no seu formato; encaixam-se perfeitamente num soquete de 14 pinos, dispensando o uso de solda.

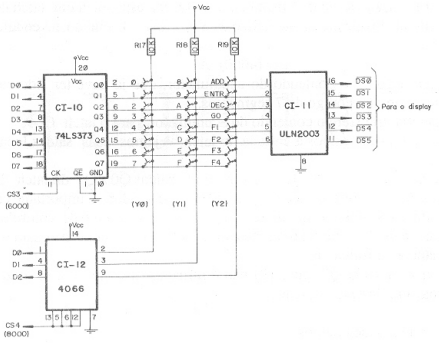


Figura 3.9 - Microcomputador APL-80 - Circuítos do Teclado e Parte do Display

A varredura horizontal e vertical do teclado, juntamente com o circuito driver do display formam o complexo da figura 3.9.

Apesar do circuito ser relativamente simples, requer um grande manuseio de software para funcionar adequadamente, e por esta razão, anteciparemos o estudo de algumas pequenas rotinas de programação, conjugando assim o "hardware ao sofware", para um melhor entendimento do circuito. Caso a compreensão da programação ainda esteja ligeiramente difícil, Não se preocupe, será revista no próximo capítulo com quantidade de detalhes.

- Latch de varredura horizontal do teclado

O circuito integrado 74LS373 já foi estudado anteriormente na figura 3.8 (CI-9) e o CI-10 da figura 3.9 tem funcionamento idêntico. A única diferença está no latch endereçado pelo código 6000 ao invés de 4000.

O CI-10 funciona também como um latch de 8 bits, e tem duas finalidades. Num determinado tempo, opera como latch de varredura de teclado, em outro, funciona como latch dos sinais que selecionam os catodos do display.

Vejamos primeiramente, como funciona na modalidade de latch de varrejadura horizontal do teclado.

Inicialmente, o microprocessador envia um dado de varredura de teclado, através do "data bus". Logo em seguida, envia o endereço 6000 no "address bus" e o sinal mreq (memory request). Neste exato momento, é gerado o sinal CS3, introduzido o dado de varredura de teclado no latch CI-10. Para tanto, veja as linhas de dados D0 a D7 ao lado esquerdo de CI-10, e bem como, o sinal CS3 entrando no pino 11 (clock) do mesmo circuito integrado.

Note que, neste latch, as saídas Q0 a Q7 estão sempre habitadas pelo pino 1 (output enable), mantendo constantemente as saídas com o conteúdo enviado pelo Z80.

Observe também que os pinos de saída 2, 5, 6, 9, 12, 15, 16 e 19, estão ligados nas linhas horizontais que varrem as teclas do teclado, na seguinte configuração:

Saída Q0 (pino 2)	varre teclas 0, 8, ADD
Saída Q1 (pino 5)	varre teclas 1,9 Entr
Saída Q2 (pino 6)	varre teclas 2, A, Dec
Saída Q3 (pino 9)	varre teclas 3, B, Go
Saída Q4 (pino 12)	varre teclas 4, C, F1
Saída Q5 (pino 15)	varre teclas 5, D, F2
Saída Q6 (pino 16)	varre teclas 6, E, F3
Saída Q7 (pino 19)	varre teclas 7, F, F4

Embora seja possível, eletronicamente, varrer todas as teclas no sentido horizontal, fica impossível determinar, qual delas foi acionada.

Por exemplo, se acionarmos a tecla 3, e varrermos todas as linhas horizontais simultaneamente, conseguiremos detectar que apenas uma das teclas foi acionada na primeira linha vertical, porém não se sabe em que linha horizontal isso ocorreu. Por este motivo, as linhas horizontais são varridas separadamente, ou seja, uma de cada vez.

A varredura horizontal é ativa quando a nível zero. Isto significa que, o microprocessador deve enviar para o latch de varredura horizontal, um código de 8 bits.

Todos os bits deste códigos, devem estar a nível um (não ativo) e, apenas o bit correspondente à linha horizontal a ser varrida, deve estar a nível zero, para tanto, veja a tabela da figura 3.10, que mostra quais são os códigos que devem ser enviados pelo programa para varrer cada linha horizontal, selecionando 3 entre 8 teclas.

Saídas do LATCH CI - 10	C Ó D I G O								TECLAS SELECIONADAS		
	FE	FD	FB	F7	EF	DF	BF	7F			
Q0	0	1	1	1	1	1	1	1	0	8	ADD
Q1	1	0	1	1	1	1	1	1	1	9	ENTR
Q2	1	1	0	1	1	1	1	1	2	A	DEC
Q3	1	1	1	0	1	1	1	1	3	B	GO
Q4	1	1	1	1	0	1	1	1	4	C	F1
Q5	1	1	1	1	1	0	1	1	5	D	F2
Q6	1	1	1	1	1	1	0	1	6	E	F3
Q7	1	1	1	1	1	1	1	0	7	F	F4

Figura 3.10 - Códigos Enviados pelo Microprocessador para selecionar 3 entre 8 teclas horizontais.

Observe que uma entre 3 teclas é posteriormente selecionada pelo circuito tri-state de varredura vertical, que será visto mais adiante.

Agora que já entendemos a primeira modalidade de funcionamento do latch CI-10 (varredura horizontal do teclado), vejamos a sua segunda possibilidade de funcionamento.

Quando é necessário acender o display, o fazemos "jogando" a terra o seu catodo. Para tanto, o micro deve "esquecer" que existe o teclado "temporariamente", e enviar para o latch CI-10, um código de seleção de display. Isto é realizado da mesma maneira como foi enviado o código de varredura de teclado.

Uma vez que o código de seleção de teclado esteja presente na saída do latch CI-10, este vai para a entrada de CI-11 para acender o display.

O circuito CI-11 (ULN2003), é um driver de potência, formado por 7 inversores, capazes de comutar até 500 MA de corrente, individuais, o que é mais do que suficiente para acender o display.

Veja na figura 3.9, que o CI-11 tem uma configuração bem simples, apenas entradas, saídas e o respectivo pino terra.

Apesar do CI-11 possuir 7 drives inversores, utilizamos apenas 6, isto porque só dispomos de 6 displays para a excitação. Pela mesma razão é que utilizamos apenas 6 linhas do latch CI-10.

As saídas do driver CI-11, são invertidas em relação à entrada, pois nossos displays, são de catodo comum, e este driver é gerador de alta corrente, apenas nas tensões próximas do nível zero. Como o catodo necessita de tensão negativa para acender, deve-se aplicar tensão positiva na entrada do driver (CI-11) para que o display acenda.

Veja a seguir, qual código o microprocessador deve carregar no latch CI-10 para acender um determinado display.

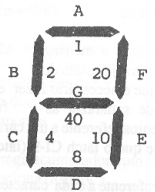
Código 01	acende	display 01
Código 02	acende	display 02
Código 04	acende	display 03
Código 08	acende	display 04
Código 10	acende	display 05
Código 20	acende	display 06
Código 3F	todos	displays

Combinando códigos, é possível acender mais de um display, porém, no programa residente (programa monitor), os displays não são utilizados desta forma.

Você deve estar lembrado que o latch CI-9 da figura 3.8, contém o código dos segmentos que devem ser acesos, a fim de tornar visível um certo número ou caractere de 7 segmentos. Combinando então o latch CI-9 (figura 3.8) com o latch CI-10 (figura 3.9), torna-se possível acender um determinado caractere (CI-9) num determinado display (CI-10).

Antes dos detalhes de funcionamento da varredura vertical do teclado, vejamos um pouco da programação dos circuitos referentes ao display.

Para que caracteres numéricos, alfabéticos ou simbólicos, possam ser apresentados no display, é necessário enviar, via microprocessador, um código ao latch de segmentos (CI-9 Figura 3.8). O código não é normal, e deve ser montado pelo programador de acordo com suas necessidades. Para gerar um código qualquer de sete segmentos, utilize a figura 3.11.



D7	D6	D5	D4	D3	D2	D1	D0	Nº da linha de dado
Não usado	G	F	E	D	C	B	A	Nome do segmento
80	40	20	10	8	4	2	1	Pêso hexa decimal

Figura 3.11- Peso Hexadecimal Associado ao Segmento do Display para Montagem do Código de 7 Segmentos

A figura 3.12 mostra 10 código de sete segmentos, para acender os caracteres APL-80. O procedimento é relativamente simples, basta configurar inicialmente os segmentos necessários para acender um determinado caractere (figura 3.11). Em seguida, basta relacionar os pesos hexadecimais relativos a cada um destes segmentos.
Finalmente, somam-se os pesos, gerando então 2 dígitos hexadecimais.

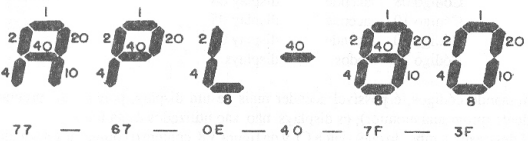


Figura 3.12 - Montagem dos Códigos de 7 Segmentos para Acender a Montagem APL-80 no Display

Note que os pesos 1, 2, 3, 4, 8, formam o dígito menos significativos (LSD), ao passo que os pesos 10 e 20 e 40 formam o mais sinificativo (MSD).
Quando a soma do dígito LSD, ultrapassar 9, deverá ser convertida para hexadecimal, ou seja, 10=A, 11=B, 12=C, 13=D, 14=E e 16=F.
Por exemplo, para formar a letra L, precisamos dos bits "ligados" 2, 4 e 8. Se adicionarmos os Bits, obteremos o valor 14, que em hexa é "E".
Como não existe nenhum bit ligado referente ao dígito MSD, então, este é considerado "zero".
O código resultante para a letra "L" é "0E" verifique a formação dos outros códigos mostrados na figura 3.12.
Para acender o display com a mensagem APL-80, é necessário um pouco de programação e alguns circuitos já estudados.
Vejamos inicialmente, o que é necessário fazer eletronicamente, para posteriormente, gerar um programa capaz de satisfazer as funções eletrônicas propostas.
Cada código de sete segmentos referente a um caractere, deve ser carregado no latch CI-9 da figura 3.8. Lembre-se que o latch CI-9 (latch de código de 7 segmentos) está no endereço 4000.
Na sequência, o display referente a cada caractere deve ser selecionado pelo latch CI-10 da figura 3.9. Lembre-se que o latch CI-10 (latch de seleção dos catodos dos displays) está no endereço 6000.
Logo que o display é selecionado, este deve ser mantido ligado (aceso) por alguns milissegundos, para posteriormente ser selecionado o próximo display.
Logo que todos os dígitos tiverem sido selecionados, acesos e temporizados, dá-se o início de outro ciclo, repetindo exatamente a mesma operação; com isto atinge-se limite da inérgia visual, dando a impressão de que todos os displays estão acesos ao mesmo tempo.

Como a mensagem APL-80 começa da esquerda para a direita, então o display 6, (display da esquerda), será o 1º display aceso com a letra "A", enquanto que, o display 1, será o último da sequência com o número 0. Confira conforme vem a seguir.

Display No.	Caractere	Código de 7 Segmentos
6	A	77
5	P	67
4	L	0E
3	-	40
2	8	7F
1	0	3F

Vejamos agora, como seria um programa simples, que pudesse acender o display na sequência proposta.

```
(início)   LD A,20          carrega acumulador com 20
           LD (6000),A      seleciona display 6
           LD A,77          carrega acumulador com 77
           LD (4000),A      liga display com "A"
           CALL "tempo"     temporiza

           LD A,10          carrega acumulador com 10
           LD (6000),A      seleciona display 5
           LD A,67          carrega acumulador com 67
           LD (4000),A      liga display com "P"
           CALL "tempo"     temporiza

           LD A,08          carrega acumulador com 08
           LD (6000),A      seleciona display 4
           LD A, 0E         carrega acumulador com OE
           LD (4000),A      liga display com "L"
           CALL"tempo"      temporiza

           LD A,04          carrega acumulador com 04
           LD (6000),A      seleciona display 3
           LD A,40          carrega acululadOr com 40
           LD (4000),A      liga display com "-"
           CALL"tempo"      temporiza

           LD A, 02         carrega acumulador com 02
           LD (6000),A      seleciona display 2
           LD A,7F          carrega acumulador com 7F
           LD (4000),A      liga display com "8"
           CALL "tempo"     temporiza
```

```
LD A,01          | carrega acumulador com 01
LD (6000),A      | seleciona display 1
LD A, 3F         > carrega acumulador com 3F
LD (4000),A      | liga display com "0"
CALL "tempo"     | temporiza
```

JP "início" **Retorna ao início e repete o ciclo**

O programa ligeiramente longo foi proposital para mostrar mais facilmente a sequência de seu funcionamento.
Mais adiante, no capítulo de programação, você perceberá que este mesmo programa será reduzido quase que pela metade, utilizando técnicas especiais de programação.
Observe as chaves que selecionam os grupos de 5 instruções, cada um deles tem a finalidade de acender uma letra num determinado display. Como existem 6 grupos, então acendemos 6 display.

Como os grupos são semelhantes entre si, examinemos apenas as primeiras instruções do 1º grupo.

A instrução "LD A, 20" carrega o acumulador com o código de seleção do display.

A segunda instrução "LD (6000),A" seleciona o display, com o código carregado no acumulador. No caso, o valor 20 (carregado no acumulador pela instrução anterior), seleciona o display 6. Lembre-se que o código 20 é carregado dentro do CI-10 da figura 3.9, gerando o sinal DS5 que habilita o catodo do 6º display.

A próxima instrução (LD A,77) tem a finalidade da primeira, isto é, "introduz" no acumulador o código 77.

A instrução "LD (4000),A" carrega o valor 77 dentro do CI-9 da figura 3.8, acionando, os segmentos A,B,C,E,F e G. Estes selecionados, geram a letra "A" no barramento de segmento do display, apesar dos displays estarem com seus segmentos de mesma designação em paralelo, apenas o display 6 acenderá com a letra "A", pois só ele foi selecionado através da 1ª e 2ª instrução.

Após acender a letra A no 6º display, só resta deixá-la acesa por alguns milissegundos. Isto é conseguido, através de um programa chamado "tempo", que aparece na 5ª instrução sob forma de sub-rotina, detalhadamente estudada no próximo capítulo.

Por enquanto, apenas considere que esta instrução mantém o display aceso por um pequeno período.

Analisando agora o próximo grupo de 5 instruções, observe que são semelhantes às já descritas, mudando somente o nº do display selecionado, bem como o caractere a ser exibido. Lembre-se apenas que o display 6 apaga, enquanto acende o display 5 com a letra P.

O ciclo se repete para os caracteres L, -, 8, 0, até encontrar a última instrução JP (início) que retorna ao início do programa, fazendo novo ciclo de atualização de display.

Como os ciclos de atualização de display são muito rápidos, a impressão que temos, é que permanecem acesos, fixando a mensagem APL-80.

- Varredura vertical do teclado.

O último circuito da figura 3.9 é o que controla a leitura ou varredura do teclado, também é chamado de circuito tri-state de entrada, formado pelo CI-12 e os resistores R17, R18 e R19.

O circuito integrado CI-12, é uma chave analógica bidirecional de silício, contendo 4 circuitos, dos quais são utilizados apenas 3.

Apesar do CI-12 ser uma chave bidirecional analógica, é utilizada como uma chave lógica direcional tri-state. Isto significa que os 3 sinais presentes nos pinos 2, 3 e 9 do CI-12 fluem em direção aos pinos 1, 4 e 8. observe que isto só ocorre quando o sinal CS4 está ativo (nível 1).

Portanto, podemos concluir que o CI-12 comporta como uma porta de entrada situada no endereço 8000.

Você deve estar lembrado que o CI-10, responsável pela varredura horizontal do teclado, seleciona 3 teclas a cada linha horizontal.

Suponha, por exemplo, que todas saídas de CI-10 estejam a nível 1, exceto o pino 2, a nível 0.

Sendo assim, este mesmo nível estará presente, numa das extremidades das teclas que compõe a 1ª linha horizontal, ou seja, teclas 0, 8 e ADD.

Caso nenhuma destas 3 teclas sejam acionadas, saiba que o microprocessador lê o conteúdo do endereço 8000, ou seja, é o mesmo que ler as 3 linhas "Y" do teclado (Y0, Y1 e Y2). Se nenhuma tecla for acionada durante a leitura das saídas do CI-12, então aparecerá um nível 1 nas 3 linhas Y0, Y1 e Y2. Isto ocorre porque os 3 reistores de "pull up" R17, R18 e R19 estão conectados no vcc (+5 volts).

Suponha agora, que a tecla ADD seja acionada e que o nível 0 esteja presente na 1ª linha horizontal.

Quando o microprocessador lê o endereço 8000, lê também automaticamente a saída de CI-12. Neste exato momento, é refletido no data bus, o nível lógico das linhas Y0, Y1 e Y2.

De acordo com a nossa suposição, a tecla ADD estava acionada e desta forma, o nível 0 presente num de seus terminais conectados na linha horizontal. Este passa agora para a linha vertical Y2 e consequentemente, o microprocessador consegue determinar, através de uma tabela interna "pré programada", o acionamento da tela ADD.

No programa que analisa as linhas Y0, Y1 e Y2, devem ser introduzidas as instruções capazes de isolar os demais bits do data bus, verificando uma linha de dados por vez, ou seja, D0 para Y0, D1 para Y1 e D2 para Y2. As demais linhas do data bus (D3 a D7) devem ser ignoradas pelo programa.
Resumindo o funcionamento do teclado, concluímos que para cada linha horizontal varrida por CI-10, são verificadas 3 linhas verticais "lidas" por CI-12.
Embora o funcionamento eletrônico do processo de varredura horizontal e vertical seja simples, o programa para "descobrir" qual tecla está acionada é relativamente complexo, porque envolve algumas sub-rotinas e tabelas de consulta. Programas que envolvem funções de teclado serão vistos no próximo capítulo. De qualquer forma, podemos prever de antemão, o programa "eco test ", que além de determinar a tecla acionada, também a exibe no display.

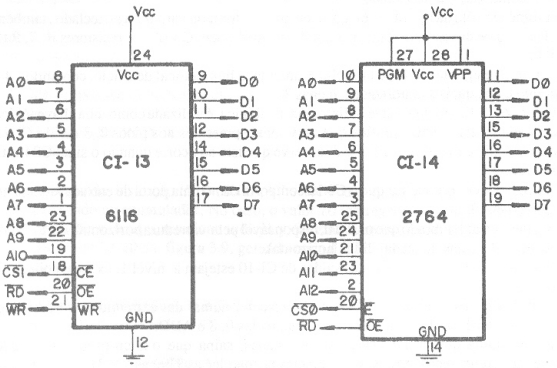


Figura 3.13 - Microcomputador APL-80 Memórias (8,3x12,5)

Os circuitos integrados CI-13 e CI-14 formam o banco de memória do microcomputador APL-80.
O CI-14 contém 8k bytes de memória (Eprom), ao passo que o CI-13 contém 2k bytes de memória (Ram), resultando uma soma de 10k, o que é mais do que suficiente para a maioria das aplicações.
Embora o microprocessador trate tanto da Eprom, como da Ram indiferentemente, vamos analisá-las separadamente.

- Memória Eprom (CI-14)
O CI-14, é composto por uma memória do tipo "somente de leitura", ocupando os endereços 0000 a 1FFF do microcomputador APL-80.

Os endereços de 1C4B a 1FFF são ocupados pelo programa monitor, estudado no próximo capítulo. Sem este programa, gravado na memória CI-14, o microcomputador não teria funções nem tampouco objetivos.
Para gravar este programa, recomendamos o uso de programadores de memória que gravem o tipo 2764. Caso você não disponha desse equipamento, obtenha informações através desta editora.
Através da figura 3.13, note que esta memória (CI-14), pode ser do tipo "somente de leitura", não recebe o sinal WR (write), ou seja "gravação".
Ao lado direito de CI-14, estão as linhas de dados (D0 a D7) em direção ao data bus do sistema.
Na parte superior de CI-14, você nota a presença dos pinos vcc (pino 28), vpp (pino 1) e pgm (pino 27).
O pino vcc é a alimentação de 5 volts normal da memória; já os pinos pgm e vpp são as entradas de programação, e devem estar sempre ligadas ao vcc (+5 volts) no modo de leitura.
Ao lado esquerdo, você nota as linhas A0 a A12 em direção ao barramento de endereço, ocupando 8096 (8k) bytes do sistema. Observe ainda, o pino 20 que nada mais é do que a entrada de habitação da memória. Nele está conectado o sinal CS0, gerado pelo circuito decodificador de endereço, sempre que o microprocessador desejar obter(ler) um dado desta memória.
Finalmente, o pino 22 é o controle tri-state das saídas (D0 a D7) da memória. Estas, só passam para baixa impedância, no momento em que o pino 2 for a nível zero. Isto é conseguido, conectando neste pino, o sinal RD (leitura).

- Memória Ram.
O CI-13 da figura 3.13 é uma memória tipo Ram, que tanto pode ser lida através do pino 20 (READ) "leitura", como também gravada através do pino 21 de controle de gravação (WR).
Esta memória, ocupa os endereços subsequentes aos da memória Eprom (CI-14), porém, sua capacidade está na casa dos 2k (2048 bytes) o que é suficiente para a maioria das aplicações. Este espaço de memória compreende os endereços de 2000 a 27FF, e são utilizados tanto pelo usuário do APL-80, quanto como rascunho pelo Z80.
Uma certa precaução, deve ser tomada durante a programacão, evitando misturar dados do programa, com dados do "rascunho" (stack) do Z80. Mais adiante no capítulo de programação, você verá que a primeira instrução enviada para o Z80, é a que determina o stack pointer (apontador da pilha de dados), localizando a memória stack no fim da Ram (CI-13).
O CI-13 possui também 8 linhas de dados (D0 a D7), que permitem, tanto gravar dados da memória para o Z80, como do Z80 para memória.
Além das linhas de dados interconectadas ao data bus, a memória CI-13 possui seus pinos de endereço que vão de A0 a A10, ocupando um espaço de 2k do sistema.
2k podem ser lidos ou gravados, mediante o pino de controle CE (chip enable) pino 17, receptador do sinal CS1, gerado nos intervalos de endereços de 2000 a 3FFF. Observe que como o decodificador de endereço varia de 8 em 8k, e a memória é de 2k, os endereços de 2800 a 3FFF são desprezados.

Finalmente, observe que os pinos 20 e 21 estão ligados respectivamente aos sinais RD (leitura) e WR (gravação).

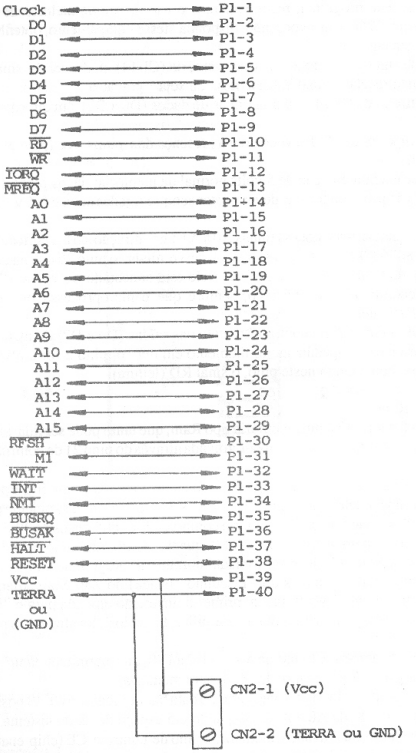


Figura 3.14 - Microcomputador APL-80 - Pinagem Plug.1 e Conector 2

Como dissemos anteriormente, o microcomputador APL-80 possui um conector para interligação com outros circuitos.
Trata-se do conector 1 (CN1) com 40 pinos. Optamos por um tipo de fácil manuseio, e alta confiabilidade.
O conector 1 é do tipo macho, por este motivo a designação de seus pinos vem como plug, cuja numeração vai de P1-1 (plug1 - pino 1) a P1-40 (plug 1 pino 40).
A figura 3.14 mostra no lado esquerdo, os sinais que vão para os circuitos do microcomputador APL-80, ao passo que do lado direito estão os números dos pinos que vão para o conector macho (plug).
O conector fêmea que encaixa neste plug, vai para os outros módulos discutidos nos futuros capítulos.
Logo mais abaixo na figura 3.14, você nota outro conector fêmea CN2, contendo 2 pinos. Através deles deve ser conctada a fonte de alimentação de 5 volts, com corrente suficiente, não só para alimentar o módulo do microcomputador APL-80, como também os demais.

CAPÍTULO 4

MICROCOMPUTADOR

APL-80

(SOFTWARE)

ÉRICA

4.1 Apresentação

No capítulo anterior, estudamos toda a parte que se refere à eletrônica do APL-80 (hardware). Vejamos neste capítulo, toda programação "básica" necessária para controlar a eletrônica descrita anteriormente.
Tudo que se refere à programação chamamos de "software", pois como o própio nome diz, é a parte "leve" da eletrônica.
Para não alongarmos muito este capítulo, consideramos que o leitor já tenha o mínimo conhecimento necessário sobre programação do Z80.
O Software que descreveremos a seguir, está em linguagem Assembly, é acompanhado do código de máquina necessário a ser carregado na memória, a fim de que o Z80 possa executar as instruções.
Neste capítulo, abordaremos com especial enfoque, o programa "monitor", seguido por alguns programas, contendo rotinas especiais para teste, ou melhor entendimento de algumas partes isoladas do sistema. Convém notar que qualquer programa ou rotina introduzida no APL-80, deve ser introduzido na memória com o auxílio do programa monitor.
Para maior facilidade de estudo, dividiremos o programa monitor em partes, sendo cada uma delas uma rotina ou sub-rotina, acompanhada do seu fluxograma.
Todas as partes do programa monitor estarão sob controle do programa principal.

4.2 Programa Monitor

O microcomputador APL-80 sozinho, sem uma memória programada, nada pode realizar.

Porém, quando a memória é provida de um programa monitor, torna o micro inteligente, capaz de ler dados do teclado, interpretá-los e posteriormente colocá-los no display de forma ordenada.
Nosso programa monitor deve estar gravado no CI-14 (memória EPROM 2764) e ocupa os endereços de 0000 a 0002 e 1C4B a 1FFF.
A função específica do programa monitor é gravar ou ler um valor num determinado endereço de memória.
Antes de entrar em detalhes de funcionamento das rotinas que compõem o programa monitor, vejamos como estas estão montadas. Veja para tanto a figura 4.1.

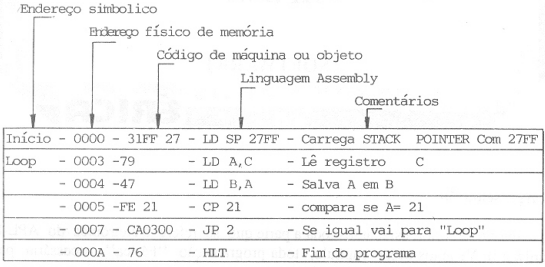


Figura 4.1

O primeiro campo que aparece nesta figura é o que chamamos de endereço simbólico. Através do endereço simbólico, torna-se possível determinar através de "marcas" no programa, para que trecho de memória o micro deve ir após a conclusão de uma determinada rotina.
A vantagem do endereço simbólico é de não necessitar de um determinado número de endereço, pois mudando o endereço físico do programa, o endereço simbólico permanece o mesmo. Por exemplo, veja no programa da figura 4.1, o endereço simbólico "Loop" que está no endereço "0003". Se no entanto mudarmos o símbolo "Loop" para a 5ª linha do programa, o endereço simbólico "Loop" passará a ser o mesmo ao passo que o endereço físico muda para "0007".
Outro campo que aparece logo a seguir, é o "endereço físico de memória". Estes números devem variar em hexadecimal, e são incrementados a cada byte (8 bits 2 dígitos) do código de máquina.
O endereço físico de memória é o lugar onde devem estar guardados dados ou instruções do programa.
Note que estes endereços mudam para cada início de instrução, e como existem instruções de 1, 2, 3 e 4 bytes, podem variar na mesma quantidade. Veja por exemplo a primeira instrução. Esta possui 3 bytes, ou seja, 3 pares de dígitos. Como esta instrução começa em "0000", então temos os bytes assim distribuídos na memória:

0000 - 31
0001 - FF
0002 - 27

Logo, conclui-se que o endereço da próxima instrução será 0003.
A segunda instrução é de apenas 1 byte, ocupando o endereço 0003.
O terceiro campo do formato do programa se refere ao "código de máquina". É através destes códigos, que o Z80 consegue entender, o que desejamos que ele faça. Por exemplo, quando o Z80 lê na memória a instrução 79, entende que é para transferir o conteúdo do registro "C" para o registro "A".
Para se escrever um programa em linguagem de máquina, também chamado de código de máquina ou linguagem objeto, é necessário lembrar todos os códigos do Z80. Isto é uma tarefa árdua e cansativa, por este motivo é que se utiliza o quarto campo no formato da programação.
Toda vez que escrevemos um programa, o fazemos em linguagem Assembly (quarto campo), logo a seguir, através de uma tabela, é que convertemos de linguagem assembly, para linguagem objeto. Esta tabela pode ser encontrada no apêndice D.
Finalmente no quinto campo, estão os comentários que facilitam o entendimento do programa.

4.3 Rotinas e Sub-rotinas do Programa Monitor

Diversas sub-rotinas e rotinas principais compõem o programa monitor. A seguir, estas serão estudadas individualmente acompanhadas de seu respectivo fluxograma.
Antes de iniciarmos o estudo das rotinas do programa monitor, vejamos inicialmente, algumas posições de memória (endereços), que são reservados na RAM (CI-13), com o objetivo de ajudar o programa monitor, a guardar dados e informações, a fim de que possa tomar decisões em cima destes valores.
Alguns destes endereços vagos, são utilizados como registros, armazenando valores que podem ser dados ou endereços indiretos. Outras das posições de memória são utilizadas como contadores de eventos.
Lembre-se que estas informações estão na RAM, e podem "atrapalhar" ou modificar o programa que nestes endereços possam ser gravados. Portanto, quando você for gravar um programa qualquer, não utilize os endereços 2700-270F.
Vejamos a seguir, quais são estes endereços e para que servem cada um deles.

- 2700 Registro de Dados (RD)
O registro de dados armazena o conteúdo do endereço mostrado nos 2* primeiros displays do lado direito, ou seja, o "RD" (registro de dados) apresenta o valor hexadecimal dos 2 primeiros dígitos do lado direito do display.
Por exemplo: Suponha que o display apresente a seguinte numeração: 2000-2D.
Conclui-se então, que o conteúdo do endereço 2000 é "2D". Portanto, 2D é o conteúdo do registro de dados (RD). Em outras palavras, neste caso, em 2700 temos "2D".

- 2701 Registro de Endereço LSD (RE)
- 2702 Registro de Endereço MSD (RE)

As posições 2701 e 2702 armazenam os endereços mostrados nos 4 displays da esquerda. Suponha que o display apresente a seguinte numeração: 1COF-39, então o registro 2701 conterá o valor menos significativo que é "OF". O registro 2702 conterá o valor mais significativo que é "1C". Lembre-se que o conteúdo de 1COF que é 39 vai para o registro 2700.

- 2703 Registro de Endereço de Tabela LSD (RT)
- 2704 Registro de Endereço de Tabela MSD (RT)

Estes dois endereços de memória, têm função semelhante às 2 anteriores (2701 e 2702). Porém, ao invés de apontar o endereço do display, aponta o início de qualquer tabela de 7 segmentos, cujos códigos serão apresentados posteriormente no display. Os conteúdos dos endereços apontados pelos registros 2703 e 2704, são "passados" para os endereços 2706, 2707, 2708, 2709, 270A e 270B, para posteriormente serem apresentados no display.

Suponha como exemplo, que o conteúdo dos endereços 2703 e 2704 sejam E8 e 1E respectivamente. Suponha também que os endereços 1EE8 a 1EEB contenham os seguintes valores:

1EE8 - 77
1EE9 - 7C
1EEA - 7C
1EEB - FF

Quando o programa executar uma rotina especial de display, inicialmente verificará qual é o "endereço de partida" da tabela. Isto é obtido, lendo o conteúdo dos endereços 2703 e 2704. Logo em seguida une valores lidos, formando o "endereço de partida" da memória (1EE8).

Uma vez apontado o 1º endereço da tabela, o programa lê os conteúdos da tabela sequencialmente e os transfere para os registros de display R6, R5, R4, R3, R2 e R1 (End: 2706, 2707, 2708, 2709, 270A, 270B). Numa próxima etapa de programação, os conteúdos de R6 a R1 são apresentados no display. Note que o código FF no endereço 1EEB determina o fim da tabela.

- 2705 Registro de Tecla Acionada (RK)

O endereço 2705 serve para armazenar o código da última tecla acionada. Os códigos de cada tecla aparecem a seguir:

TECLA	CÓDIGO	TECLA	CÓDIGO	TECLA	CÓDIGO
0	00	8	08	ADD	F1
1	01	9	09	ENTR	F2
2	02	a	0A	DEC	F3
3	03	b	0B	GO	F4
4	04	c	0C	F1	F5
5	05	d	0D	F2	F6
6	06	e	0E	F3	F7
7	07	f	0F	F4	F8

- 2706 Registro do 6º Display (R6)
- 2707 Registro do 5º Display (R5)
- 2708 Registro do 4º Display (R4)
- 2709 Registro do 3º Display (R3)
- 270A Registro do 2º Display (R2)
- 270B Registro do 1º Display (R1)

Os registros 2706 a 270B guardam informações em forma de 7 segmentos correspondentes a cada display. O display R6 é o mais significativo (lado esquerdo).

- 270C Registro Fim da Tabela (RF)

Os endereços 2706 a 270B podem ser utilizados diretamente como uma tabela para o display. Quando isto ocorrer, o endereço 270C deve conter o código FF, determinando fim da tabela de dados.

- 270D Contador de Dados (CD)

O endereço 270D é utilizado como contador de tecla de dados. Este acompanha os dados que vão sendo introduzidos nos conteúdos dos endereços.

- 270E Contador de Endereço (CE)

Este endereço de memória é utilizado como contador de endereço. É incrementado a cada tecla de endereço apresentada no display. Este contador pára a contagem após ser digitada a 4ª tecla de endereço.

- 270F Contador de Invalidade (CI)

Toda vez que uma tecla for acionada fora de sequência, o programa incrementará contador de invalidade e cancelará a operação.

ENDR.MEMÓRIA	FUNÇÃO	REGISTRO
2700	Registro de Dados	RD
2701	Registro de Endereço LSD	RE
2702	Registro de Endereço MSD	RE
2703	Registro de Endereço de Tabela LSD	RT
2704	Registro de Endereço de Tabela MSD	RT
2705	Registro de Tecla Acionada	RK
2706	Registro do 6º Display	R6
2707	Registro do 5º Display	R5
2708	Registro do 4º Display	R4
2709	Registro do 3º Dislay	R3
270A	Registro do 2º Display	R2
270B	Registro do 1º Display	R1
270C	Registro Fim de Tabela	RF
270D	Contador de Dados	CD
270E	Contador de Endereço	CE
270F	Contador de Invalidade	CI

Figura 4.2 - Endereços da memória RAM utilizados como registros e contadores.

Para facilitar e organizar o estudo das rotinas do programa monitor, resumimos na Figura 4.2, os endereços discutidos até agora.
Se você teve alguma dúvida com referência ao uso dos registros e contadores descritos até o momento, não se preocupe, pois a seguir, detalharemos toda s as rotinas, sub-rotinas e tabelas do programa monitor. Muitas delas, envolvem os registros citados, e serão melhor compreendidos ao longo da descrição do funcionamento de tais rotinas.
Vejamos a seguir, estas rotinas em sequência:

A) ROTINA PRINCIPAL (PRNC)

Endereços ocupados: 0000 a 0002 e 1C4B a 1C7C.
- Fluxograma:
Através da figura 4.3, pode ser notado o fluxograma da rotina principal. Está dividida basicamente em 2 estágios.

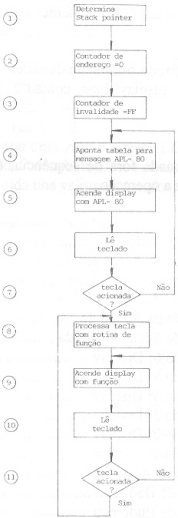


Figura 4.3 - Fluxograma Rotina Principal (PRNC)

No primeiro estágio desta rotina, o programa recicla os passos 4 a 7, acendendo o display com a mensagem APL-80. Este processo se repete até o instante que uma tecla seja acionada. Quando isto ocorrer, o programa será chaveado para um segundo estágio.
No segundo estágio da rotina, a função da tecla é processada e o display é aceso de acordo com a função da tecla acionada. Este processo recicla os passos 9 a 11 até que outra tecla seja acionada. Neste instante, o programa processa a função desta tecla, através do passo 8. Logo em seguida são reciclados os passos 9 a 11.
Observe que o segundo estágio da rotina, permanece indefinidamente, até que seja acionado o sistema de Reset do micro APL-80.

- Programa:

O trecho do programa monitor correspondente à rotina principal está na figura 4.4. Vejamos a seguir, seu funcionamento por instrução em sequência.

ENDEREÇO SIMBÓLICO	ENDER. FÍSICO	CÓDIGO OBJETO	ASSEMBLY	COMENTÁRIOS
	0000	C34B1C	JP (início)	Salta para área inícial do programa monitor
INÍCIO	1C4B	31FF27	LD SP,27FF	Determina stack pointer no fim da Ram (27FF)
	1C4E	3E00	LD A,00	Zera A
	1C50	320E27	LD (270E),A	Zera contador de endereços
	1C53	3EFF	LD A,FF	Carrega A com FF
	1C55	320F27	LD (270F),A	Carrega FF no contador de invalidade
APL	1C58	21001D	LD HL,1D00	Aponta ender. inicial da tabela da mensagem APL-80
	1C5B	220327	LD (2703),HL	Aponta ender. inicial da tabela da mensagem APL-80
	1C5E	CD4D1F	CALL (DISP)	Chama sub-rotina Display (DISP)
	1C61	CD6B1F	CALL (LEIT)	Chama sub-rotina leitura de teclado (Leit)
	1C64	3A0527	LD A,(2705)	Lê registro de tecla (2705)
	1C67	FEFF	CP FF	Verifica se a tecla está acionada
	1C69	28ED	JR Z(APL)	Caso negativo vai para APL
LOOP-2	1C6B	CD901C	CALL (FUNC)	Chama rotina de funcão (FUNC)
LOOP-1	1C6E	CD4D1F	CALL (DISP)	Chama sub-rotina display (DISP)
	1C71	CD6B1F	CALL (LEIT)	Chama sub-rotina leitura de teclado (LEIT)
	1C74	3A0527	LD A,(2705)	Lê registro de tecla (2705)
	1C77	FEFF	CP FF	Verifica se a tecla está acionada
	1C79	28F3Pd	JR Z (LOOP-1)	Caso negativo vai para LOOP-1
	1C7B	18EE	JR (LOOP-2)	Caso afirmativo vai para LOOP-2

Figura 4.4 Rotina Principal (PRNC)

. JP (início)

Os endereços de memória 0003 a 1C4A, estão disponíveis para outros programas que serão utilizados pelos periféricos relacionados, se necessário. Este espaço pode também ser utilizado com programas do próprio leitor, desde que você tenha um programador de memória para isso.
A instrução JP (início), apenas salta do endereço 0000 de memória para o endereço 1C4B, onde praticamente começa a rotina principal.

. LD SP, 27FF

Quando é iniciado o programa, após a alimentação do circuito, uma das primeiras instruções que deve ser enviada ao micro, é a que define a memória Stack.
Uma vez determinado o "stack pointer", todas as sub-rotinas utilizarão este endereço, para retornar ao programa principal.
A instrução LD SP, 27FF, carrega o stack pointer com o valor 27FF.
O endereço 27FF é a última posição de memória RAM disponível. Por esta razão, o Z80 guarda os endereços de retorno de sub-rotina, sempre decrementando o valor do stack pointer, ou seja, do endereço 27FF para baixo.
Lembre-se que o stack pointer é um registro de 16 bits dentro do Z80. Lembre-se também que, este registro (SP) não guarda os endereços de retorno das sub-rotinas, apenas aponta o endereço da memória RAM, onde estão guardados os endereços de retorno das sub-rotinas.
Se você ainda não entendeu como este processo é feito, aguarde a 9ª instrução, que completará o entendimento do manuseio do stack pointer.

. LD A, 00

O acumulador (registro A), é carregado com o valor 00, ou seja, a instrução LD A,00 zera o acumulador, preparando-o para a próxima etapa do programa. Note que as instruções do grupo Load (LD) são as mais utilizadas. Load significa carga. No caso LD A,00 diz para o micro "carregar" zero no registro A.
O movimento da carga das instruções do tipo Load, é da direita para a esquerda. Por exemplo a instrução "LD A,B" carrega o registro A com o valor contido em B.
OBS: Alguns autores substituem o termo carga por transferência.

. LD (270E), A

Este tipo de instrução é do tipo carga ou transferência de registro para memória.
Sempre que um valor, ou par de letras (HL, DE, BC) estiver entre parênteses, significa endereço de memória. Se por exemplo, o conteúdo do par de registradores HL for 2000, e se estiver entre parênteses, significa que estamos nos referindo ao endereço 2000 e não ao par de registradores em questão.
Portanto a instrução "LD (270E),A" carrega o endereço 270E com o conteúdo de "A". Como o registro A foi carregado com "00" na instrução anterior, então conclui-se que o endereço 270E passa a ter 00.
Como já descrevemos anteriormente, o endereço 270E é utilizado como contador de endereço, portanto é zerado neste estágio do programa.

. LD A,FF

Carrega o registro A com FF, preparando-o para a próxima instrução.

. LD (270F),A

Transfere o conteúdo do registro "A" (FF) para o endereço 270F.
Conforme já vimos anteriormente, o endereço 270F é utilizado como contador de invalidade.

Quando o programa está normal, e nenhuma tecla foi acionada fora de sequência, seu valor permanece com FF.
Mais adiante este contador será utilizado em outras rotinas. Por enquanto lembre-se que está posicionando com o conteúdo igual a FF.

. LD HL, 1D00.

Como pode ser observado através do fluxograma, o objetivo de primeiro estágio da rotina principal é posicionar alguns registros de sinalização e apresentar no display a mensagem APL-80.
A instrução LD HL, 1D00 prepara o programa para apresentar a mensagem APL-80. Note que esta instrução carrega o par de registros H e L com o valor 1D00.

. LD (2703), HL.

Na instrução anterior, o registro L foi carregado com "00" e o registro H com "1D". Agora através da instrução "LD (2703),HL", o conteúdo de L é carregado em 2703 e o conteúdo de H é carregado em 2704. Portanto temos:

L = 00 ——————> 2703
H = 1D ——————> 2704

Em outras palavras, a instrucão "LD (2703), HL" é do tipo LD (NN),HL onde NN representa um endereco de memória . Sua finalidade é carregar L no endereço NN, e H no endereço NN+1. Portanto NN=2703 e NN+1=2704.
Apenas recordando, você estudou que os endereços 2703 e 2704 são utilizados como "registros de endereço de tabela". Agora note que, 2703 contém "00" e 2704 contém 1D. Se agregarmos os conteúdos destes dois endereços de memória, teremos o valor 1D00. Este valor, é o endereço inicial da tabela que contém a mensagem APL-80.
A próxima instrução executa uma sub-rotina que utiliza a tabela situada a partir do endereço 1D00.
A tabela que exige a mensagem APL-80 faz parte da rotina principal, portanto pode ser observada através da tabela 1 (TB-1) na figura 4.5.

1D00	27	(A)
1D01	67	(P)
1D02	OE	(L)
1D03	40	(-)
1D04	7F	(8)
1D05	3F	(0)
1D06	FF	Fim dos caracteres da tabela

Figura 4.5 - Tabela para gerar mensagem APL-80 (TB-1)

Mais adiante quando estudarmos a rotina de display, você entenderá melhor o funcionamento desta tabela. Apenas note a sequência dos caracteres APL-80, sendo representados por códigos de 7 segmentos. Veja ainda, o último código FF, que determina o fim da sequência de caracteres.

. CALL (disp)

A instrucão acima é do tipo Call (NN). Através desta, é possível dirigir o programa para qualquer parte da memória. Esta instrução na realidade é do tipo "salto", como foi vista na primeira linha do programa (JP (início)).

A diferença é que a instrução do tipo "JP (NN)" salta para uma área do programa e lá continua a executar outras instruções.

Já a instrução do tipo CALL (NN), salta para uma área do programa, executando lá diversas instruções, até encontrar uma instrução do tipo RET (retorno). Quando isto ocorrer, o programa voltará para o endereço logo após onde estava a instrução CALL.

Com a utilização de um CALL, torna-se possível executar uma rotina principal, chamar (CALL) uma ou mais rotinas secundárias (sub-rotinas) para depois retroceder à rotina principal.

As instruções do tipo CALL, utilizam a memória apontada pelo stack pointer, com o objetivo de guardar o endereço de retorno das sub-rotinas.

Para melhor compreensão do funcionamento de uma sub-rotina veja a figura 4.6. Nela você pode observar um pequeno programa dividido em duas partes.

ENDEREÇO DO PROGRAMA	INSTRUÇÃO	STACK POINTER	MEMÓRIAS DO STACK POINTER 27FF	27FE	27FD
0000	LD SP,27FF	27FF	XX	XX	XX
0003	LD A,A	27FF	XX	XX	XX
0004	LD A,A	27FF	XX	XX	XX
0005	LD A,A	27FF	XX	XX	XX
0006	CALL(2000)	27FD*	XX	00	09
0009	HLT	27FF	XX	00	09
2000	LD B,B	27FD	XX	00	09
2001	LD B,B	27FD	XX	00	09
2002	LD B,B	27FD	XX	00	09
2003	RET	27FF*	XX	00	09

Figura 4.6 - Exemplo de Funcionamento de uma sub-rotina

A primeira parte é a rotina principal e ocupa os endereços 0000 a 0009.

A segunda parte é a sub-rotina propriamente dita, e ocupa os endereços 2000 a 2003.

Este programa utiliza também os endereços 27FF, 27FE e 27FD como área de trabalho do stack pointer.

Vejamos agora, o funcionamento do nosso pequeno programa.

A primeira instrução (LD SP, 27FF), surge no endereço 0000. Esta simplesmente carrega o stack pointer do Z80, como valor 27FF, determinando assim, o endereço do início da pilha de dados, onde serão guardados os "endereços" de retorno "das sub-rotinas.

Veja que esta instrução, apenas aponta o endereço 27FF, porém, não utiliza esta memória por enquanto. Por esta razão é que você deve ter notado os endereços 27FF, 27FE e 27FD, contendo qualquer valor (XX).

Como a primeira instrução ocupa 3 bytes (seu código objeto é 31FF27), então o próximo endereço do programa é 0003.
Do endereço 0003 a 0005 inserimos 3 instruções "LD A,A". Estas nao afetam o programa, apenas foram colocadas ai, para mostrar que inúmeras instruções poderiam ser inseridas neste trecho do programa.
A 5ª instrução (CALL (2000)) desvia o programa do endereço 0006 para 2000. Antes porém, o endereço da próxima instrução (6ª instrução), que é 0009 é guardado na memória apontada pelo stack pointer. Portanto o Z80 decrementa o stack pointer (27FF-1=2FFE), e o valor resultante (1FFE), é utilizado para apontar o endereço de memória, onde é guardado, o byte mais significativo de endereço de retorno da sub-rotina.
Portanto o valor 00, é armazenado na memória 1FFE.
Logo em seguida, o Z80 decrementa o stack pointer novamente, gerando então o valor 1FFD (1FFE-1=1FFD).
O valor resultante (1FFD), é utilizado para apontar o endereço de memória, onde é guardado, o byte menos significativo de endereço de retorno da sub-rotina.
Portanto o valor 09, é armazenado na memória 1FFD, veja 5ª linha do programa.
Agora que o Z80 armazenou o endereço de retorno da sub-rotina (0009), então vai para a sub-rotina que inicia no endereço 2000.
Uma vez na sub-rotina (2000 a 2003), o programa executa normalmente as instruções contidas em 2000 a 2002. Estas nada fazem, apenas são figurativas, representando diversas instruções que podem ser inseridas entre a primeira e última linhas do programa.
Uma vez o programa chegando no último endereço da sub-rotina (2003), é executada a instrucão de retorno (RET). Nesta ocasião, o Z80 consulta o conteúdo do endereço apontado pelo stack pointer e utiliza-o como endereço de retorno para a rotina principal.
Isto é realizado lendo os endereços 27FE e 27FD, obtendo o valor 0009. Em seguida, o Stack pointer volta ao valor original 27FF.
Finalmente, a instrução contida no endereço 0009, é executada. No caso, trata-se de uma instrução de parada (HLT).
Neste ponto, o programa é estacionado, porém outras instruções poderiam ser aí inseridas.
Resumindo o fluxo deste programa, temos o seguinte percurso através da memória. 0000, 0001, 0002, 003, 0004, 0005, 0006, 0007, 0008, 27FE, 27FD, 2000, 2001, 2002, 2003, 27FD, 27FE e 0009.
Ainda com referência à sub-rotinas, convém lembrar que pode existir a possibilidade de uma sub-rotina estar dentro da outra. Em outras palavras seria, a "sub-rotina da sub-rotina". Para tanto, o procedimento é normal ao descrito até o momento, porém lembre-se que o stack pointer é decrementado 2 unidades para cada instrução CALL, e incrementado 2 unidades para cada instrução RET.
Agora que você entendeu o mecanismo das sub-rotinas, vamos voltar a nossa instrução, da 9ª linha da rotina principal (CALL (DISP)).
Esta instrução desvia o programa para o endereço 1F4D onde é iniciada a sub-rotina de display (DISP).

A sub-rotina display, você estudará mais adiante por enquanto, apenas saiba que, esta gera no display a mensagem APL-80, pois foram preparados os endereços 2703 e 2704, cujos conteúdos apontam a tabela que gera a mensagem APL-80 no display.
Lembre-se que esta sub-rotina (CALL (DISP)) após ter sido executada, o programa retorna na rotina principal correspondente à 10^a linha.

. CALL (LEIT)
Esta sub-rotina, tem o mesmo mecanismo de funcionamento da sub-rotina anterior, ou seja, desvia-se da rotina principal, vai para a sub-rotina de leitura de teclado (LEIT), e ao terminá-la retorna à rotina principal.
A finalidade básica da sub-rotina LEIT, é ler teclado, gerando então um determinado código que é gravado no endereço 2705.
Observe o código objeto da instrução "CALL (LEIT)" que é CD6B1F. O primeiro byte CD evidencia uma instrução CALL. Os dois bytes remanescentes, apontam o endereço da sub-rotina 1F 6B.
Lembre-se que, os endereços do código objeto devem ser fornecidos ao Z80 invertidos, isto é, primeiro o byte menos significativo "6B" e em segundo lugar o byte mais significativo "1F".
Se não houver nenhuma tecla acionada durante a execução da sub-rotina de leitura de tecla, então, o registro de tecla acionada, é carregado com o valor FF (2705=FF), caso contrário o código da tecla é gravado em 2705.

. LD A, (2705)
Esta instrução apenas carrega o registro A, com o valor do código da tecla, que foi anteriormente gravado em 2705.
Lembre-se que. se nenhuma tecla foi acionada, o código em 2705 será FF.

. CP FF
O registro "A" é comparado com a constante "FF". Se o resultado da comparação for igual, então o flag zero, dentro do Z80 é ligado. Caso contrário (comparação desigual), o flag "não zero" é ligado.
Em resumo, a instrução "CP FF" prepara os flags do Z80 para a próxima instrução.

. JR Z (APL)
Se a tecla não foi acionada na sub-rotina de leitura de teclado, o endereço 2705 (reg.de tecla acionada), é carregado com o código FF.
Durante a instrução de comparação "CP FF", o flag zero é ligado.
A instrução "JR Z" quando executada, testa o flag zero. Se estiver ligado, então o programa salta para o endereço simbólico APL (1C58). Caso o flag zero não esteja ligado, então o programa simplesmente ignora a instrução JR Z.
O formato da instrução "JR Z,e" é onde "JR Z" significa "JUMP RELATIVE" ou seja, salto relativo. A letra "e" representa o endereço relativo do salto, podendo ser para trás ou para frente.
A vantagem de se utilizar saltos do tipo relativo, está na economia de bytes do programa.

Um salto normal do tipo JP, ocupa 3 bytes na memória, sendo um byte de código da instrução, e 2 bytes para determinar, o endereço do salto (veja a primeira instrução). Um salto do tipo JR, ocupa 2 bytes na memória, sendo um byte de código da instrução e outro byte de endereço relativo. Portanto cada salto relativo, economiza 1 byte na memória.

Outra vantagem do salto relativo, é que este indica ao programa, quantos bytes deve saltar para frente ou para trás, não sendo necessário especificar o endereço real, somente o endereço relativo.

A desvantagem do salto relativo, é não poder avançar além de uma página de memória (128 bytes).

Para se calcular o valor do salto relativo é utilizada a tabela do apêndice C. Vejamos como exemplo, a própria instrução "JR Z(APL)" que está na 13ª linha do programa, ou seja, no endereço 1C69. Esta instrução é formada por 2 bytes, sendo o primeiro byte, o código da instrução "JRZ" que é 28. O segundo byte "ED" representa o endereço de deslocamento com a finalidade de saltar o programa para o endereço APL (1C58).

Para calcular o 2º byte é necessário "contar" quantos bytes existem a partir do endereço IC6A, onde termina a instrução JR 2 (APL), até o endereço IC58, onde começa a instrução apontada pelo endereço simbólico APL.

Considere para tanto os seguintes bytes a partir de 1C58:

21, 00, 1D, 22, 03, 27, CD, 4D, 1F, CD, 6B, 1F, 3A, 05, 27, FF, FE, 28, ED.

Se você contar todos eles observará um total de 19 bytes.

Agora, através da tabela de saltos relativos "para trás" podemos observar que para 19 o valor é ED. Portanto, o código de JR Z (APL) será "28ED".

Mais adiante, nós detalharemos em outros programas, como é formado o código de salto para frente.

Note que os saltos podem ser condicionais ou não. Uma instrução JR, salta para trás ou para frente sob quaisquer condições. Já a instrução JR Z, salta apenas se o flag zero estiver ligado. Portanto trata-se de uma instrução condicional.

As instruções condicionais, é que decidem o que o programa deve fazer. Na realidade, são a inteligência do programa.

Veja no caso, a própria instrução "JR Z (APL)" que salta para o endereço APL só se nenhuma tecla for acionada, exibindo assim ciclicamente no display a mensagem APL-80. "Quando" uma tecla for acionada, a instrução "JR Z (APL)" desviará o programa para executar a próxima instrução, abortando então a mensagem APL-80 no display.

. CALL (FUNC)

Toda vez que uma tecla for acionada, a primeira sub-rotina executada é a de função (FUNC). Esta separa qual tecla de dado ou função foi acionada , e encaminha-a para o programa final adequado.

Note que quando termina a sub-rotina FUNC, o programa retorna para rotina principal, executando a próxima instrução a seguir.

Um fato importante a comentar, é que dentro da sub-rotina "FUNC" existem sub-rotinas. Porém, todas retornam à rotina principal.

. CALL (DISP)

Anteriormente, já vimos a sub-rotina display sendo utilizada para exibir no display a mensagem APL-80. Nesta instrução (15ª linha do programa), a sub-rotina de display exibe outro tipo de mensagem no display.

Dependendo da tecla acionada, são apresentados no display, valores ou mensagens diferentes, em outras palavras, a instrução "CALL (DISP)" da 15ª linha do programa, depende da rotina "FUNC", que por sua vez, depende da tecla acionada.

. CALL (LEIT)

Uma nova leitura de teclado ocorre com a finalidade de testar se há alguma tecla acionada.

Lembre-se que se nenhuma tecla for acionada, o conteúdo do endereço 2705 será FF, caso contrário, conterá o código da tecla.

. LD A, (2705)

Transfere o código da tecla contido em 2705 para o registro A.

. CP FF

Compara se o registro A contém FF. Caso afirmativo liga o flag "zero", preparando-o para ser testado pela próxima instrução.

. JR Z(LOOP1)

Esta é uma instrução de decisao do programa, se o flag zero foi ligado pela instrução anterior, significa que nenhuma tecla foi acionada, desta forma o programa vai para o endereço simbólico LOOP1 (1C6B), reciclando então, a mensagem referente à última tecla acionada, através da rotina "CALL (DISP)".

Em seguida, é lido novamente o teclado, através da instrução "CALL (LEIT)", referente à 16ª linha do programa.

Finalmente, através das instruções da 17ªA 19ª linha, o programa testa novamente se nenhuma tecla foi acionada.

Se nenhuma tecla foi acionada, o ciclo se repete entre a 15ª linha e 19ª linha do programa.

Veja que, entre o endereço 1C6E, até o endereço 1C7A (fim da instrução JR Z(LOOP1)), existem 13 bytes.

Se você consultar a tabela de saltos relativos para trás, verá que para 13 bytes, o valor do salto relativo é F3. Portanto o código objeto da instrução JR Z (LOOP1) é "28F3".

Quando uma tecla for acionada, o programa passará a não satisfazer a instrução "JR Z (LOOP1)", pois o conteúdo do endereço 2705, passará a não ter FF, e sim o código da tecla acionada.

Quando isto ocorrer, a instrução de comparação "CP FF" não ligará o flag "zero", sendo assim a instrução JR Z(LOOP1) será saltada, direcionando o programa, para a última linha. Nesta, aparece um salto incondicional para o endereço simbólico LOOP2 (1C6E). Uma vez o programa no endereço 1C6E, chama a sub-rotina de função, classifica e executa a função da tecla.

Logo a seguir, através da instrução "CALL (DISP)", apresenta a mensagem no display referente à nova tecla (15ª linha do programa).
O ciclo entre as linhas 16 a 19 do programa, é executado com o objetivo de pesquisar, se a tecla foi ou não acionada.
Caso a tecla tenha sido acionada, o programa vai para o LOOP2, caso contrário vai para o LOOP1.
Este ciclo se repete até que seja desligada a alimentação do circuito, ou seja acionada a tecla G0.
Para finalizar a rotina principal,observe através da figura 4.7, os códigos de máquina (código objeto) que compõem a rotina principal (PRNC).

```
0000-   C3 4B 1C

1C4B-   31 FF 27 3E 00 32 0E 27 3E FF 32 OF 27 21 00 1D
1C5B-   22 03 27 CD 4D 1F CD 6B 1F 3A 05 27 FE FF 28 ED
1C6B-   CD 90 1C CD 4D 1F CD 6B 1F 3A 05 27 FE FF 28 F3
1C7B-   18 EE
```

Figura 4.7 - Código Objeto da Rotina Principal (PRNC)

Os primeiros 4 dígitos da esquerda representam o endereço inicial de cada linha.
Os bytes (par de dígitos), após o hífen, são os conteúdos dos endereços em sequência.
Por exemplo, se você deseja saber qual o endereço do 7º byte, da 2ª linha da figura 4.7, basta "contar" em hexadecimal, a partir do endereço inicial à direita anterior ao hífen:

```
1C4B  =   31 (1º byte)
1C4C  =   FF (2º byte)
1C4D  =   27 (3º byte)
1C4E  =   3E (4º byte)
1C4F  =   00 (5º byte)
1C50  =   32 (6º byte)
1C51  =   0E (7º byte)
```

Compare o conteúdo do endereço 1C51 da Figura 4.7 com a Figura 4.4

B) SUB-ROTINA DISPLAY (DISP)
Endereços ocupados: 1F4D a 1F64
- Fluxograma:
Para que um ou mais caracteres possam ser exibidos no display, utiliza-se a sub-rotina iniciada no endereço 1FD4.
Antes que a sub-rotina "DISP" seja iniciada, é necessário que outras rotinas, "carreguem" os endereços 2703 e 2704, com o valor inicial do endereço onde estarão os caracteres, que serão apresentados no display.
Esta sub-rotina a grosso modo, "pega" valores em 6 endereços de memória, e os coloca sob forma de caracteres no display.

Vejamos agora, o funcinamento geral desta sub-rotina através do fluxograma da Figura 4.8.

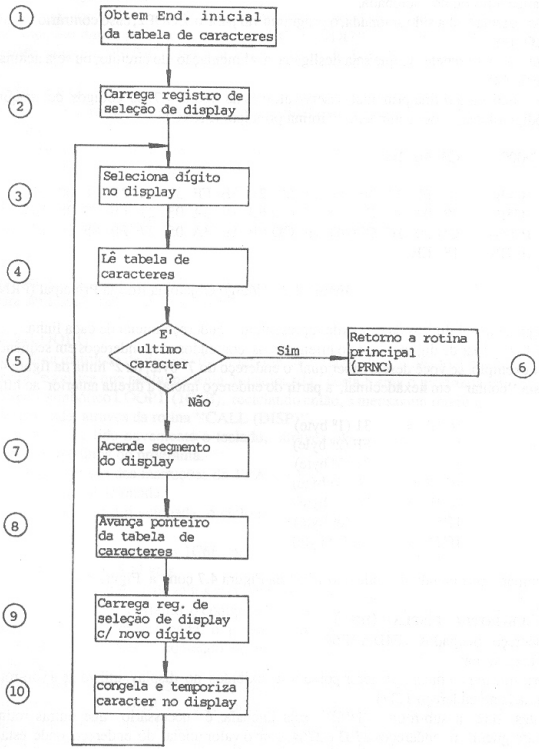


Figura 4.8 - Fluxograma da Sub-rotina Display (DISP)

Inicialmente, o endereço inicial da tabela de caracteres que irão ser exibidos no display, são lidos dos registros de endereços da tabela (2703 e 2704).
Como segundo e terceiro passo do fluxograma, é selecionado o display que conterá o 1º caractere.
No 4º passo, o 1º caractere a ser apresentado é lido da tabela de caracteres, cujo endereço inicial, foi apontado no 1º paso do fluxograma.
O 5º passo do fluxograma, apenas verifica se o último caractere já foi exibido no display, caso afirmativo o programa vai para o Posso 6, retornando ao programa principal. Caso contrário, vai para o 7º passo, acendendo o display, com o valor lido na tabela de caracteres.
O passo 8 avança para o próximo endereço da tabela com o objetivo de obter o 2º caractere a ser exibido no display.
O passo 9, prepara para selecionar o 2º display, porém, os circuitos ainda estão selecionando o 1º display.
O 10º e último passo do fluxograma, mantém o display aceso por alguns milissegundos com o abjetivo de fixar o caractere por algum tempo.
Após o 10º passo, o fluxograma vai para o passo 3 onde novo display, e novo caractere são exibidos.
Este ciclo se repete até que o último caractere seja exibido.
Um comentário importante a fazer, é que a sub-rotina de display, quando executada, acende 6 displays, e depois os apaga. Para que os mesmos fiquem sempre acesos, é necessário reciclar a sub-rotina "display" constantemente. Esta reciclagem, como você deve estar lembrado, é realizada pela rotina principal.

- Programa:
O trecho do programa monitor corresponde à sub-rotina de display está na figura 4.9.

ENDEREÇO SIMBÓLICO	ENDER. FÍSICO	CÓDIGO OBJETO	ASSEMBLY	COMENTÁRIOS
	1F4D	2A0327	LD HL,(2703)	Obtém ender. inicial da tabela de caracteres
	1F50	0620	LD B,20	Carrega registro de seleção do display
INIC	1F52	78	LD A,B	Prepara para ligar catodo do display
	1F53	320060	LD (6000),A	Liga catodo do display
	1F56	7E	LD A,(HL)	Lê caractere da tabela
	1F57	FEFF	CP FF	Verifica se é fim da tabela
	1F59	C8	RET Z	Caso afirmativo retorna à rotina principal
	1F5A	320040	LD(4000),A	Liga segmentos do display
	1F5D	23	INC HL	Incrementa ponteiro da tabela de dados
	1F5E	CB08	RRC BP	Prepara para selecionar próximo display
RECI	1F60	OC	INC C	Incrementa registro timer
	1F61	20FD	JR NZ (RECI)	Recicla se não for zero
	1F63	18ED	JR (INIC)	Volta ao início para acender outro display

Figura 4.9 - Sub-rotina Display (DISP)

Vejamos a seguir, seu funcionamento na sequência.

. LD HL, (2703)

Esta instrução carrega o conteúdo do endereço 2703 no registro "L", e o conteúdo do endereço 2704 no registro "H". Desta forma, obtém-se assim, dentro do par "HL", o endereço inicial, onde estão os dados que serão apresentados no display.

Note que as posições de memória 2703 e 2704, são utilizadas como endereço indireto. Já o conteúdo do par "HL", é utilizado como endereço direto.

Procure recordar que na rotina principal, os endereços 2703 e 2704, foram carregados como valor "1D00", que indicava o endereço inicial da tabela-1. Lembre-se também que esta tabela contém os caracteres para acender o display com a mensagem APL-80.

Considere portanto como exemplo, que a sub-rotina display deve exibir a mensagem APL-80. Portanto, neste instante, HL contém "1D00" que é o endereço inicial da tabela 1 apresentada na Figura 4.5.

. LD B,20

Durante todo o período de execução da sub-rotina "DISP", o registro "B" é utilizado como "Registro de seleção de display".

Note que a instrução LD B,20 carrega o registro B com o valor 20, preparando-o para a próxima instrução.

. LD A,B

Como o registro "B" não pode ser destruído, então o seu conteúdo, é transferido para o registro A. A instrução LD A,B prepara o registro "A" para próxima instrução:

. LD (6000),A

A instrução LD (6000),A transfere o valor do acumulador (registro A), para o endereço 6000

Como o acumulador contém 20, então este valor é transferido para 6000.

No endereço 6000, você deve lembrar que existe um registro fisíco, que seleciona os displays de acordo com o código nele existente.

Apenas para recordar veja a seguir:

Código	Display
20	6
10	5
08	4
04	3
02	2
01	1

Como o código transferido para o endereço 6000 foi "20", então foi selecionado o 6º display (1º display do lado esquerdo).

Lembre-se que o display só acende quando selecionamos os seus segmentos. Portanto, a instrução LD A, (6000) apenas seleciona o display, não o acendendo ainda.

. LD A, (HL)

Na primeira linha do programa, o par HL foi carregado com os conteúdos dos endereços 2703 e 2704. Apenas a título de exemplo, consideramos o par HL com 1D00, indicando então, o endereço inicial da tabela 1.

Portanto a instrução LD A, (HL), lê a memória apontada por HL (1D00) e carrega o registro A com o valor lido.

Note na figura 4.5, que o endereço 1D00 contém 27. Este valor corresponde ao código de 7 segmentos, correspondente à letra A.

Ao término da instrução LD A, (HL) o registro A passa a ter então o valor 27.

. CP FF

Nosso programa não sabe quantos displays possuímos, por esta razão é necessário finalizar a sub-rotina após o último caractere ser aceso no display.

Uma forma simples de indicar ao programa quando terminou a sequência de caracteres, é colocar uma marca na memória, logo após o último caractere que deve ser apresentado no display.

Esta marca pode ser qualquer caractere, desde que este não seja um código de 7 segmentos.

Como o valor FF, não é utilizado em nenhum código de 7 segmentos, então padronizamos este valor, para fim de sequência de caracteres em todas as tabelas do programa monitor.

Veja na figura 4.5, o código FF no último endereço de memória, representando assim, o fim de caracteres da tabela.

Portanto a instrução CP FF, compara se o dado contido no acumulador, é igual a FF. Caso o resultado da comparação seja igual, então o flag zero é ligado. Preparando o programa para próxima instrução.

. RET Z

Trata-se de uma instrução de retorno condicional, isto é o programa só retorna à rotina principal (PRNC), se o flag zero estiver ligado, caso contrário, a instrução RET Z é ignorada e o programa passa a executar a próxima linha.

Convém observar que a instrução RET Z, só é executada, quando o resultado da comparação realizada pela instrução anterior, for verdadeiro, ou seja, o caractere lido na memória pela 5ª instrução, for "FF", evidenciando fim da sequência de caracteres, a serem exibidos no display.

A instrução RET Z é a saída da sub-rotina de display. e logicamente o programa só sai por este ponto, após o último caractere ser apresentado no display.

. LD (4000), A

Na 5ª instrução, o programa leu a memória da tabela de dados, e obteve o código do 1º caractere que deve aparecer no display (27). Este código por sua vez, fica no acumulador para ser utilizado pela 8º instrução do programa.

Como já dissemos no 3º capítulo, o endereço 4000 é ocupado fisicamente pelo registro de segmentos do display, portanto, quando é executada a instrução "LD (4000),A" o valor de A que contém o código 27, passa para o registro de segmentos do display.

Neste instante, no 6º display aparece a letra "A".

. INC HL

Esta instrução incrementa o conteúdo do par de registradores "HL" com o objetivo de endereçar o próximo caractere da tabela (letra P).
Como o "HL" anteriormente continha 1D00, passa a ter agora 1D01, preparando assim o programa para apresentar o próximo caractere no display.
Não esqueça que esta instrução não "mexe" no display, apenas prepara o par HL para apontar um novo endereço de memória.

. RRC B

Já sabemos que o registro B, é utilizado para guardar o código que selecionará o próximo display.
Também sabemos que o registro B, foi carregado pela 2º instrução do programa com o valor 2o.
A instrucão "RRC B" tem a finalidade de deslocar todos os Bits do registro B, gerando assim, um novo código de seleção de display.
Este movimento de rotação, desloca 1 bit para a direita, toda vez que o programa executar a instrução "RRC B", preparando então o registro B, para selecionar um novo display.
Portanto, se o valor inicial de B era 20, passa agora a ser 10, preparando assim para selecionar o 5º display.
Veja na figura 4.10, todos os movimentos de rotação que a instrução "RRC B" executa, bem como todos os códigos que são gerados até o último display ser selecionado.

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	Potência Binária
80	40	20	10	8	4	2	1	Valor Hexa
0	0	1	0	0	0	0	0	1º Valor do Registro B
0	0	0	1	0	0	0	0	Valor do Registro B após 1º Deslocamento
0	0	0	0	1	0	0	0	Valor do Registro B após 2º Deslocamento
0	0	0	0	0	1	0	0	Valor do Registro B após 3º Deslocamento
0	0	0	0	0	0	1	0	Valor do Registro B após 4º Deslocamento
0	0	0	0	0	0	0	1	Valor do Registro B após 5º Deslocamento

Figura 4.10 - Movimentos de Rotação de Registro B para Gerar Códigos de Seleção de Display

. INC C

Devido à inércia, é necessário que deixemos cada display aceso por alguns milissegundos, desta forma é necessária introduzir no programa um certo atraso.
O registro C é utilizado como timer, é incrementado até atingir o valor zero.
A instrução INC C incrementa C uma vez apenas.

. JR NZ (RECI)

Esta instrução, trabalha em conjunto com a anterior.

Quando o registro "C" é incrementado, o flag de "não zero", é ligado toda vez que o incremento gerar um valor diferente de zero.

A instrução JR NZ (RECI) é do tipo salto relativo condicional, ou seja, é executada somente se o flag de não zero" estiver ligado, caso contrário, a instrução JR NZ (RECI) é ignorada e o programa continua na instrução seguinte.

Veja que o registro C, é incrementado pela instrução anterior (INC C), logo a seguir, a segunda instrução "JR NZ (RECI)", testa o flag de "não zero", e se este estiver ligado, o programa salta para o endereço simbólico RECI (1F60).

Esta reciclagem (RECI) continua, até que o registro "C" atinja o valor zero.

Quando "C" for zero, o flag "não zero é desligado e finalmente o programa vai para a última instrução.

Você deve ter percebido que, a sub-rotina de display, fica reciclando entre a 11ª e 12ª instrução até atingir a contagem zero no registro C.

Como o registro "C" possui 8 bits, então a capacidade máxima de incrementos é 256 (28=256). Portanto, note que de acordo com a frequência do cristal, o microprocessador gasta 256 incrementos e 256 saltos relativos, gerando assim um determinado tempo para manter aceso o display.

Sabendo que a frequência do cristal é 3,57 MHZ, torna-se possível calcular este tempo da seguinte forma.

1- Cálculo do state ou período do clock

$$\text{Período} = \frac{1}{\text{frequência}} = \frac{1}{3.57 \times 10^6} = \frac{1}{3.57} \times 10^{-6} = 0{,}280\mu S$$

2- Cálculo do tempo gasto para 256 instruções de incremento.

1 instrução "INC" gasta 4 states ou 4 clocks.
logo, 1 instrução INC gasta 4X0,280µs = 1,120µs
então, 256 instruções INC gastam 1,120µs x 256 = 286,72µs

3- Cálculo do tempo gasto para 256 instruções de salto relativo.

1 instrução "JR NZ" gasta 12 states ou 12 clocks
logo, 1 instrução "JR NZ" gasta 12x0,280µs = 3,36µs
então, 256 instruções "JR NZ" gastam 3,36µs x 256 = 869,16µs

4- Cálculo do tempo total:

256 instruções "JR NZ" = 860,16µs
256 instruções "INC" = 286,72µs

Total = 860,16 + 286,72 = 1,146µs.

Portanto, nosso timer formado pelas instruções da 11ª e 12ª linha atrasa nosso programa em 1,146 milissegundos.

. JR (INIC)

Esta última instrução, salta incondicionalmente para o endereço simbólico INIC (início), iniciando então novo ciclo de display. Desta forma apaga o 6º display, acende o 5º display com a letra P e aguarda o timer.

Outros ciclos idênticos ocorrerão para acender os caracteres L, -, 80 nos displays 4, 3, 2 e 1 respectivamente

Na figura 4.11, aparece o código objeto da sub-rotina display (DISP).

```
1F4D-   2A 03 27 06 20 78 32 00 60 7E FE FF C8 32 00 40
1F5D-   23 CB 08 0C 20 FD 18 ED
```

Figura 4.11 - Código Objeto da Sub-rotina Display (DISP)

c) SUB-ROTINA LEITURA DE TECLAS (LEIT)

Endereços ocupados: 1F6B a 1FAA

- Fluxograma:

A finalidade básica da sub-rotina de leitura de teclas, é fazer uma varredura completa do teclado.

Se nenhuma tecla foi acionada, esta sub-rotina apenas carrega o endereço 2705 com o valor "FF", orientando assim, outras rotinas sobre a situação do teclado.

Se alguma tecla foi acionada, então a sub-rotina de leitura de tecla, com o auxílio da tabela 2, procura descobrir o código da tecla acionada. O código encontrado, é carregado em 2705 para ser utilizado por outras rotinas.

Convém comentar também, que esta sub-rotina se utiliza de uma outra sub-rotina com a finalidade de aguardar as trepidações mecânicas das teclas.O fluxograma da sub-rotina de leitura de teclas (leit) pode ser notado através da figura 4.12.

Antes de entrarmos em detalhes de funcionamento desta sub-rotina, vejamos a função de alguns registros do Z80 e endereços de memória utilizados.

1- Registro B.

Para selecionarmos o gerador de varredura horizontal descrito no capítulo 3, é necessário utilizar um registro qualquer, de forma que o programa possa ler ou gravar o código de seleção das linhas "X" do teclado. Para esta finalidade utilizamos o registro B do Z80.

2- Registro C.

Para selecionarmos as linhas Y do teclado, utilizamos o registro C, de forma semelhante ao registro B.

3- Par de Registradores HL.

Existe uma tabela que o programa consulta constantemente esta é utilizada para descobrir o código da tecla acionada.Para endereçar esta tabela é utilizado o par HL.

4- Endereço 2705.

Você deve estar lembrado que o código da tecla acionada, deve ser armazenado no registro de tecla acionada.

O registro utilizado para esta finalidade é o endereço de memória 2705. Veja a figura 4.2.

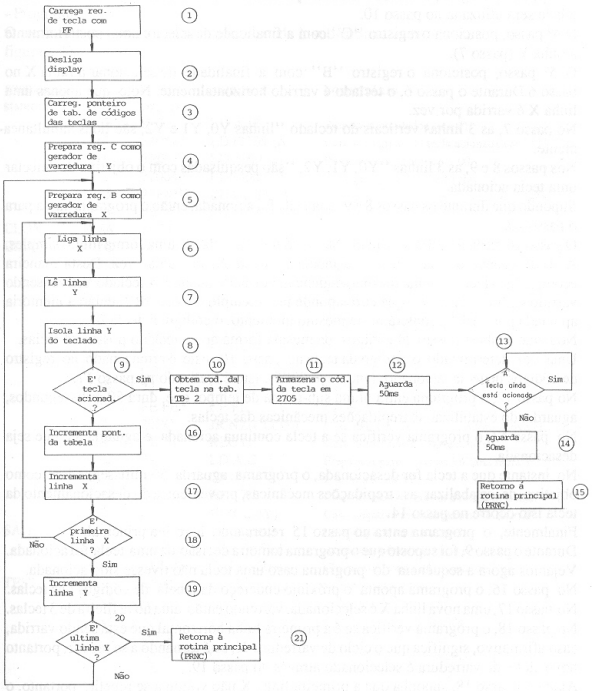


Figura 4.12- Fluxograma da Sub-rotina Leitura de Teclado (LEIT)

Iniciando o fluxograma da sub-rotina de leitura de teclas, você pode observar que o primeiro passo do programa, é carregar o registro de teclas com o código "FF". Caso não haja nenhum acionamento de tecla, então o código FF permanecerá guardado neste registro, indicando às demais rotinas, que não houve acionamento do teclado.
O 2º passo do programa, apaga todos os displays, com o objetivo de desligar as linhas de varredura X.
O 3º passo, carrega o par HL com o endereço inicial da tabela de códigos das teclas. Esta tabela será utilizada no passo 10.
O 4º passo, posiciona o registro "C" com a finalidade de selecionarnar posteriormente a linha Y (passo 7).
O 5º passo, posiciona o registro "B" com a finalidade de selecionar a linha X no passo 6.Durante o passo 6, o teclado é varrido horizontalmente. Note que apenas uma linha X é varrida por vez.
No passo 7, as 3 linhas verticais do teclado "linhas Y0, Y1 e Y2, são lidas simultaneamente.
Nos passos 8 e 9, as 3 linhas "Y0, Y1, Y2, "são pesquisadas com o objetivo de detectar uma tecla acionada.
Supondo que durante os passos 8 e 9 uma tecla foi acionada, então o programa passa para o passo 10.
O passo 10, utiliza a tabela - 2 para obter o código da tecla, de uma forma muito simples. A cada avanço de varredura X, a tabela é incrementada uma vez. Desta maneira acompanha o teclado numa mesma sequência. Portanto, quando o teclado estiver sendo varrido na linha X e Y, que corresponde por exemplo à tecla "B", então a memória apontada por "HL", conterá neste mesmo momento, o código da tecla B.
Note que a tabela possui 24 códigos, da mesma forma que o teclado possui 24 teclas.
Uma vez determinado o código da tecla no passo 10, este é armazenado no registro de códigos de tecla acionado (endereço 2705). Isto é realizado no passo 11.
No passo 12, o programa entra numa sub-rotina de tempo, que dura 50 milissegundos, aguardando estabilizar as trepidações mecânicas das teclas.
No passo 13, o programa verifica se a tecla continua acionada e aguarda até que seja desacionada.
No instante que a tecla for desacionada, o programa aguarda 50 milissegundos como objetivo de estabalizar as trepidações mecânicas, provenientes do desacionamento da tecla isto ocorre no passo 14.
Finalmente, o programa entra no passo 15 retornando à rotina principal.
Durante o passo 9, foi suposto que o programa tomou a decisão de uma tecla foi acionada. Vejamos agora a sequência do programa caso uma tecla não tivesse sido acionada.
No passo 16, o programa aponta o próximo endereço da tabela de códigos das teclas.
No passo 17, uma nova linha X é selecionada, varrendo então uma nova fileira de 3 teclas.
No passo 18, o programa verifica se é a primeira linha horizontal que está sendo varrida, caso afirmativo, significa que o ciclo de varredura X, está tornando a se repetir, portanto novo ciclo de varredura é selecionado através do passo 19.
Ainda no passo 18, suponha que a primeira linha X não voltou a se repetir, portanto, o programa retrocede para o passo 6, e fica reciclando entre os passos 6 e 18, até que uma tecla seja acionada, ou a primeira linha X, volte a ser varrida.

No passo 20, o programa verifica se a última linha Y esta sendo selecionada, caso afirmativo, significa que a última tecla foi varrida e não estava acionada. Quando isto ocorre, a sub-rotina termina seu ciclo e vai para o passo 21, retornando à rotina principal.
Ainda durante o passo 20, se não for detectada a última linha Y, então o programa retorna para o passo 5, fazendo assim uma nova varredura de teclado.

- Programa:
O trecho do programa monitor corresponde à sub-rotina de leitura de teclado está na figura 4.13.

ENDEREÇO SIMBÓLICO	ENDER. FÍSICO	CÓDIGO OBJETO	ASSEMBLY	COMENTÁRIO
	1F6B3	EFF	LD A,FF	Prepara acumulador com FF
	1F6D	320527	LD (2705),A	Carrega registro de tecla acionada com "FF"
	1F70	3E00	LD A,00	Prepara acumulador com 00
	1F72	320040	LD (4000),A	Desliga segmentos do display
	1F75	21E81F	LD HL,1FE8	Carrega HL com endereço inicial da tabela-2
	1F78	0E 01	LD C,01	Carrega "C" com linha Y=0
CLOY	1F7A	06 FE	LD B,FE	Carrega "B" com linha X=0
CLOX	1F7C	78	LD A,B	Transfere B para A
	1F7D	320060	LD (6000),A	Liga linha X do teclado com valor de "B"
	1F80	3A0080	LD A,(8000)	Lê linha Y do teclado
	1F83	A1	AND C	Isola linha Y do teclado
	1F84	FE 00	CP 00	Está a tecla acionada?
	1F86	2810	JR Z(SAI1)	Se acionada vai para saída 1
	1F88	23	INC HL	Incrementa ponteiro da tabela-2
	1F89	CB00	RLC B	Gera nova linha de varredura X
	1F8B	78	LD A,B	Prepara registro A para comparação
	1F8C	FE FE	CP FE	Compara se é 1ª linha linha X
	1F8E	20EC	JR NZ(CLOX)	Caso não seja última linha retorna novo ciclo x
	1F90	CB 01	RLC C	Gera nova linha de varredura Y
	1F92	79	LD A,C	Prepara A para testar se é última linha Y
	1F93	FE 08	CP 08	É última linha Y?
	1F95	C8	RET Z	Caso afirmativo retorna à rotina principal
	1F96	18 E2	JR (CLOY)	Caso negativo retorna para novo ciclo Y
SAI1	1F98	7E	LD A,(HL)	Lê código da tecla na tabela
	1F99	320527	LD (2705),A	Armazena código da tecla em 2705
	1F9C	CDB01F	CALL (50ms)	Chama sub-rotina de retardo de 50ms
TEST	1F9F	3A0080	LD A,(8000)	Lê novamente a tecla
	1FA2	A1	AND C	Isola linha Y do teclado
	1FA3	FE 00	CP 00	Está a tecla ainda acionada
	1FA5	28F8	JR Z(TEST)	Testa novamente a tecla
	1FA7	CDB01F	CALL (50ms)	Chama sub-rotina de retardo de 50 ms
	1FAA	C9	RET	Retorna a rotina principal

Figura 4.13 Sub-rotina de Leitura de Teclado (LEIT)

Vejamos a seguir, seu funcionamento em sequência.

. LD A,FF
Carrega o acumulador com FF preparando-o para próxima instrução.

. LD (2705),A
Transfere conteúdo do acumulador (FF) para o endereço 2705.
O endereço 2705 é o registro de código de tecla acionada, portanto, quando a sub-rotina de leitura de teclado é iniciada, este registro é posicionado com o valor FF, indicando a outras rotinas, que nenhuma tecla foi acionada. No entanto, se alguma tecla for acionada ao longo desta sub-rotina, o conteúdo do endereço 2705, é alterado com o código da tecla acionada.

. LD A,00
Carrega o acumulador com o valor "00", preparando-o para a próxima instrução.

. LD (4000),A
Como você deve estar lembrado, no endereço 4000 existe fisicamente o registro de segmentos do display.Para que os displays não acendam durante a leitura do teclado, este registro é zerado através da instrução LD (4000),A.

. LD HL,1FE8
Como já foi dito no fluxograma, a tabela 2 da figura 4.14, é utilizada pela sub-rotina de leitura de teclado.

ENDEREÇO	CODIGO	TECLA
1FE8	00	0
1FE9	01	1
1FEA	02	2
1FEB	03	3
1FEC	04	4
1FED	05	5
1FEE	06	6
1FEF	07	7
1FF0	08	8
1FF1	09	9
1FF2	0A	A
1FF3	0B	B

ENDEREÇO	CODIGO	TECLA
1FF4	0	CC
1FF5	0	DD
1FF6	0	EE
1FF7	0	FF
1FF8	F1	ADD
1FF9	F2	ENTR
1FFA	F3	DEC
1FFB	F4	GO
1FFC	F5	F1 (Função 1)
1FFD	F6	F2 (Função 2)
1FFE	F7	F3 (Função 3)
1FFF	F8	F4 (Função 4)

Figura 4.14 Tabela de Códigos das Teclas - TB-2

Cada vez que uma tecla é "varrida", o endereço da tabela é incrementado. Portanto, o código da tecla está constantemente sendo apontado por esta tabela, e também sempre em fase com a tecla varrida. Isto significa que quando a tecla "B", por exemplo, for varrida, a tabela estará refletindo o código "OB" no endereço 1FF3.

Convém notar que se nenhuma tecla for acionada durante toda a varredura do teclado, nenhuma informação será retirada da tabela.
A instrução LD HL, 1FE8 carrega o par de registradores HL com o endereço inicial da tabela (1FE8), preparando assim o programa para "seguir" a tabela junto com cada tecla varrida.

. LD C,01
Esta instrução prepara o registro "C", para ser utilizada como selecionador de varredura vertical (linhas "Y").
Como existem 3 linhas Y(Y0, Y1 e Y2), conclui-se então, que durante a execução da sob-rotina de leitura de teclado, o registro C pode assumir 3 valores diferentes. São eles: 01, 02 e 04.
Mais adiante na 11ª instrução, o registro será novamente utilizado.

. LD B,FE
Da mesma forma que utilizamos o registro "C" para selecionar a varredura vertical, utilizamos o registro "B", como gerador de varredura horizontal do teclado (linhas "X").
Lembre-se que, tanto o registro "C" como "B", não são utilizados diretamente, pois seus conteúdos sempre passam antes pelo acumulador.
A instrução "LD B, FE", carrega o registro "B" com um determinado valor (FE), permitindo que a primeira linha X (X0) seja varrida.
Como existem 8 linhas horizontais (X0aX7), é necessário varrer apenas uma por vez. Para que sejam satisfeitos os circuitos digitais, é necessário manter a nível zero, somente uma linha X, as demais devem permanecer a nível 1.
Portanto nosso registro B gera inicialmente um nível zero, somente na linha X0, mais tarde este registro, é deslocado para a esquerda, deslocando concomitantemente, todos seus bits.
O deslocamento dos bits do registro "B", permite então, gerar códigos distintos, varrendo assim uma linha horizontal por vez.
Mais adiante na 15ª instrução (RLC B), veja os códigos geradores no registro B.

. LD A,B
Lê o registro B e passa para o acumulado, preparando assim o programa para próxima instrução.
Lembre-se que na primeira execução do programa, "B" contém o valor "FE".

. LD (6000),A
O valor "FE" é transferido para o endereço 6000, onde fisicamente está conectado o registro de varredura horizontal do teclado. Sendo assim, a linha "X0" é mantida a nível zero, e as linhas X1, X2, X3, X4, X5 e X6 são mantidas a nível 1.
Com a linha X0 a nível zero, as teclas "0", "8" e "ADD" são varridas, ou seja, são mantidas a nível zero em um dos seus pólos. Cabe portanto ao circuito de varredura Y, determinar qual das 3 foi acionada.
Veja na próxima instrução, como isto é realizado.

. LD A,8000

No endereço 8000, está presente fisicamente o circuito vertical de leitura do teclado. Quando a instrução "LD A,(8000)" é executada, são lidas as linhas Y0, Y1 e Y2 para dentro do acumulador.

Normalmente, quando nenhuma tecla foi acionada, então o resultado da leitura destas 3 linhas, é um nível "1" em cada uma. Se no entanto, uma das 3 teclas "0", "8" e "ADD" for acionada, o resultado da leitura no acumulador será um nível no bit correspondente à linha Y da tecla acionada.

Lembre-se que Y0 seleciona as teclas 0, 1, 2, 3, 4, 5, 6 e 7, Y1 seleciona as teclas 8, 9, A, B, C, D, E e F, Y2 seleciona as teclas ADD, ENTER, DEC, G0, F1, F2, F3 e F4.

. AND C

Você deve ter notado que a instrução anterior carregou no acumulador o resultado do "estado lógico" das linhas Y0, Y1 e Y2.

Como a linha "X" varrida é a "X0", então falta determinar se existe ou não alguma tecla acionada ao longo da linha X. Isto é realizado isolando o bit correspondente a cada linha Y, e verificando se o resultado é 1 ou 0.

A instrução AND C tem exatamente a finalidade de mascarar ou isolar um determinado bit no acumulador.

O registro C é utilizado para selecionar ou isolar uma determinada linha Y. Este, durante o programa, pode assumir o valor 01, para seleção da linha Y0, 02 para seleção da linha Y1, e 04 para seleção da linha Y2. Como a instrução da 6ª linha (LD C,01) carregou "C" com "01", então neste momento, o registro C isolará a linha Y0 do teclado.

A instrução AND C, executa uma lógica "AND" entre o acumulador e o registro "C", sendo que o resultado é depositado no próprio acumulador.

Suponha que o registro C contenha o valor "01" e que a tecla "0" foi acionada.

A tecla "0" corresponde à linha Y0, que por sua vez está conectada ao "Data Bus" "D0".

Sabemos também que, quando uma tecla é acionada, e esta correspondente a uma linha X ativa (sendo varrida), o resultado é que a linha Y referente à tecla acionada, vai a nível zero.

Portanto, conclui-se que o bit correspondente à linha Y0, é refletido no Data Bus (linha D0), e finalmente está a nível zero.

A instrução AND C, faz lógica "AND" entre o acumulador, e o Registro "C", com os seguintes valores.

Registro C	0000 0001
Acumulador	xxxx xxx0
Resultado Acumulador	0000 0000

Observe que não interessa os demais bits do acumulador, a instrução "AND", se preocupa em isolar, apenas o 1º bit do acumulador.

Portanto o resultado da operação é zero, zerando então o acumulador.

Observe que se a tecla "0" não fosse acionada, então, o resultado da operação lógica seria 1, conforme segue.

Registro C	0000 0001
Acumulador	xxxx xxx1
Resultado Acumulador	0000 0001

Recorde a função "AND"

*Duas entradas a nível "0",	a saída é "0"
*Duas entradas diferentes,	a saída é "0"
*Duas entradas a nível "1",	a saída é "1"

. CP00

Na instrução anterior, a tecla "zero" foi acionada, portanto o resultado da lógica "AND" zerou o acumulador.
A instrução CP00 simplesmente compara o acumulador como valor "00".
Se o resultado da comparação for igualdade, então o flag zero é ligado, preparando o programa para próxima instrução.

. JR Z (SAI 1)

Trata-se de uma instrução de salto relativo condicional.
Se o flag zero estiver ligado, esta instrução desvia o programa para o endereço SAI 1 (saída 1), caso contrário (tecla não acionada), continua o programa na próxima instrução.
Observe que no endereço da saída 1 (1F98), existe um pequeno programa que determina o código da tecla, e depois retorna à rotina principal.
Este trecho do programa estudaremos mais adiante, por enquanto, lembre-se que a saída 1, ocorre sempre que uma tecla acionada for detectada.

. INC HL

Se o programa na instrução anterior não detectou nenhuma tecla acionada na linha Y0 com X0, então o programa continua no endereço 1F88.
A instrução INC HL, incrementa o par de registros HL, orientando assim o programa, a apontar novo endereço da tabela, gerando assim o código da tecla "1". Este conteúdo da tabela será utilizado caso a tecla 1 esteja acionada.

. RLC B

Como já vimos anteriormente, o registro B é utilizado como gerador de varredura X. Assim sendo, é necessário que seu conteúdo, seja modificado a cada vez que uma nova linha X for selecionada.
Sabemos que a linha X selecionada, tem que ficar a nível zero, enquanto as demais permanecem a nível 1.
O conteúdo inicial do registro "B" era FE. Este valor foi assim obtido por ocasião da execução da 7ª instrução.
O valor "FE" zera a linha X0 e mantém a nível 1 as demais, selecionando as três telas correspondentes à linha X ("0", "8", "ADD").

A instrucão RLC B, desloca todos os bits do registro B para a esquerda, gerando então o valor "FD", ou seja: 1111 1101.
Note que "FD" possui o 2º bit zerado, sendo assim a linha X1 é então selecionada, varrendo as teclas "1", "9" e "ENTER".

. LD A,B
Como o registro B não pode ser comparado diretamente, então o seu valor é depositado no acumulador.
A instrucão LD A,B executa exatamente esta tarefa, preparando o registro B para ser comparado na próxima instrução.

. CP FE
O registro B partiu inicialmente com o valor FE, endereçando assim a linha X0. Logo, na 15ª instrução, seu valor foi modificado para "FD", selecionando a linha X1.
O movimento de rotação do registro B, é executado 8 vezes pela 15ª instrução. Na 8ª vez que for executado o movimento de rotação para a esquerda (RL C B), "B" volta a assumir o valor FE. Quando isto ocorre, significa que todas as linhas X foram varridas, e tem-se iniciado novo ciclo de varredura horizontal, pois "B" novamente posicionando a selecão da 1ª linha X (X0).
A instrução CP FE compara a constante FE com o valor do acumulador. Como o acumulador foi carregado anteriormente com o conteúdo do registro A, então na prática, a instrução "CP FE compara o valor FE, com o registro B.
Como o registro B contém "FE", então o resultado da comparação é igualdade, ligando assim o flag zero.
Toda vez que o resultado da instrução da 17ª linha do programa der igualdade, significa que um ciclo de varredura X foi completado, e está iniciando novo ciclo.
Este processo é utilizado para determinar quando nova linha Y deve ser selecionada pelo registro C.

. JR NZ (CLOX)
Para que uma nova linha Y seja selecionada, é necessário antes varrermos todas 8 linhas X.
A instrução JR NZ (CLOX), salta para o endereço CLOX (CICLO X) quando o resultado da comparação da instrução anterior não for igual. Isto significa que, nem todas as linhas X foram varridas.
Na prática, a instrução JR NZ (CLOX), salta para 1F7C, somente se o flag zero "não" estiver ligado. Este ciclo se repete até a linha X7 ser varrida.
Quando a linha X0 for novamente selecionada, então a instrução de "salto condicional relativo" não é satisfeita, consequêntemente, é abortada, e o programa continua na próxima instrução.

. RLC C
Você deve estar lembrado que por ocasião da execução da 6ª instrução do programa, o registro C foi carregado com o valor 01, selecionando a linha Y0.

Quando o programa atinge a 19ª instrução, é sinal que todas as linhas X foram varridas, e que durante todas as varreduras horizontais, uma linha Y se manteve selecionada. Portanto nesse momento, as teclas correspondentes à linha Y0 foram varridas, ou seja, 0, 1, 2, 3, 4, 5, 6 e 7.
Neste momento, outra linha Y deve ser selecionada. Isto é realizado, deslocando todos os bits do registro C para a esquerda através da instrução "RLC C".
Como o valor anterior do registro "C" era 01, então, após ser executada a instrução "RLC C", o valor de "C" passa a ser 02.
Como o registro C seleciona as linhas "Y", então o código 02 selecionará a linha Y1, habitando a varredura das teclas 8, 9, A, B, C, D, E e F.

. LD A,C

Como o registro C não pode ser comparado diretamente com uma constante, então é necessário antes passar seu conteúdo para o acumulador.
Isto é executado pela instrução LD A,C.

. CP 08

Através da 19ª instrução, o registro C é deslocado uma vez para esquerda, gerando os valores 01, 02, 04 e 08 a cada execução da instrução da "RLC C".
Os valores 01, 02 e 04 selecionam as linhas Y0, Y1 e Y2 respectivamente. Porém, quando "C" atingir o valor 08, significa que a última tecla "F4" foi varrida.
Neste momento, se nenhuma tecla foi encontrada acionada, o programa se prepara para seu término.
A instrução CP 08 compara se o registro C atingiu o último valor, isto é, compara se a última tecla já foi varrida. Caso afirmativo, a comparação "CP 08" detecta uma igualdade, ligando assim o flag zero.

. RET Z

Caso a instrução anterior detecte uma igualdade, então o programa retorna à rotina principal, caso contrário, continua na instrução seguinte.
Note que RET Z é uma instrucão de retorno condicional.

. JR (CLOY)_

Trata-se de um salto relativo incondicional, ou seja, quando o programa executa a instrução JR (CLOY), faz um salto direto para o endereço 1F7A (CICLO Y).
Este salto faz o programa testar se está acionando um novo grupo de teclas correspondente à outra linha Y.
Daí a razão do salto se referir ao endereço simbólico CLOY (CICLO Y).
Para que você possa entender melhor a sequência de varredura das teclas, bem como os valores dos registros "C" e "B", veja através da figura 4.15 como o programa controla as linhas X e Y.

LINHA Y	CÓDIGO DO REG. C	LINHA X	CÓDIGO HEXA REG. B	CÓDIGO BINÁRIO REG. B	TECLA SELECIONADA
Y0	01	X0	FE	1111 1110	0
Y0	01	X1	FD	1111 1101	1
Y0	01	X2	FB	1111 1011	2
Y0	01	X3	F7	1111 0111	3
Y0	01	X4	EF	1110 1111	4
Y0	01	X5	DF	1101 1111	5
Y0	01	X6	BF	1011 1111	6
Y0	01	X7	7F	0111 1111	7
Y1	02	X0	FE	1111 1110	8
Y1	02	X1	FD	1111 1101	9
Y1	02	X2	FB	1111 1011	A
Y1	02	X3	F7	1111 0111	B
Y1	02	X4	EF	1110 1111	C
Y1	02	X5	DF	1101 1111	D
Y1	02	X6	BF	1011 1111	E
Y1	02	X7	7F	0111 1111	F
Y2	04	X0	FE	1111 1110	ADD
Y2	04	X1	FD	1111 1101	ENTER
Y2	04	X2	FB	1111 1011	DEC
Y2	04	X3	F7	1111 0111	GO
Y2	04	X4	EF	1110 1111	F1
Y2	04	X5	DF	1101 1111	F2
Y2	04	X6	BF	1011 1111	F3
Y2	04	X7	7F	0111 1111	F4

Figura 4.15 - Sequência de Controle das Linhas X, Y e Leitura das Teclas pela Sub-rotina de Leitura de Teclado.

. LD A,(HL)

Quando a sub-rotina atinge a 24ª instrução, é sinal que uma tecla foi acionada, e cabe ao programa detectar qual o código desta tecla.

Como já dissemos anteriormente, o par de registradores HL apontam a cada passo do programa, o código da tecla que está sendo varrida.

Quando a tecla que está sendo varrida pela linha X, e lida pela linha Y, for acionada, então o programa salta para o endereço simbólico SAI1 (saída 1).

A instrução "LD A, (HL)" consulta a tabela da figura 4.14, e lê o conteúdo do endereço apontado por HL. O resultado da leitura é carregado no acumulador.

Por exemplo, suponha que foi executada a varredura "X7" e foi lida a linha "Y1". Também suponha que a tecla "F", estava acionada neste instante. Quando isto ocorrer, o par HL conterá o valor 1FF7.

A instrução LD A, (HL) neste caso, será o mesmo que dizer: carregue "A" com o conteúdo do endereço 1FF7.

Como 1FF7 contém o código "OF" correspondente à tecla "F", então conclui-se que o acumulador é carregado com "OF", gerando assim no acumulador o código da tecla acionada.

. LD (2705),A

Você deve estar lembrado que reservamos o endereço de memória 2705 como registro de tecla acionada.
Portanto, a instrução LD (2705),A transfere o valor do acumulador para o endereço 2705. Desta forma, o código da tecla acionada fica guardado no endereço 2705, para ser utilizado mais tarde por outras rotinas.

. CALL (50ms)

A sub-rotina de 50 ms, faz o programa gastar 50 milissegundos, com o objetivo de aguardar a estabilização mecânica da tecla.
Isto é muito importante no programa, pois caso não houvesse esta sub-rotina, durante o acionamento de uma tecla, ocorreria várias leituras da mesma, dando a informação ao programa, que a tecla foi acionada mais de uma vez.
Mais adiante na próxima sub-rotina, você entenderá melhor como é feito para retardar o programa com 50 ms.

. LD A,(8000)

Esta instrução faz exatamente o mesmo que foi descrito na 10ª linha do programa, com a finalidade de selecionar a linha X para ler a tecla.

. AND C

A mesma linha Y do teclado é isolada novamente, tal como foi executado na 11ª linha do programa.
A instrução "AND C", pesquisa se o bit referente à linha Y está a nível zero ou um, zerando ou não o acumulador.

. CP00

Tem a mesma função da instrução da 12ª linha do programa. Tem a finalidade de ligar ou desligar o "flag zero". Se a tecla foi acionada, o resultado desta comparação será igualdade, ligando então o flag zero.
O flag zero é desligado caso a tecla não tenha sido mais acionada.

. JR Z (TEST)

Trata-se de um salto relativo condicional. Esta instrução, desvia o programa para o endereço TEST (1F9F), caso o flag zero tenha sido ligado por ocasião de uma tecla acionada, caso contrário (tecla desacionada), a instrução "JR Z (TEST)" é abortada, e o programa continua na próxima linha.
Você deve ter notado, que se a tecla permanecer acionada, o programa permanecerá num ciclo contínuo, executando assim as instruções, "LD A, (8000)", "AND C", "CP00" e "JR Z(TEST)", repetidamente. Este ciclo se mantém até o momento que a tecla for solta.

Quando isto ocorrer, a instrução JR Z (TEST) será ignorada pelo programa e a linha seguinte da sub-rotina será executada.

. CALL (50ms)

Após ser desacionada a tecla, muitos ruídos elétricos serão gerados devido às trepidações mecânicas dos contactos da tecla. Para "esperar" passar estes ruídos, utilizamos novamente nossa sub-rotina de retardo de 50 milissegundos.

. RET

Uma vez terminado o tempo de 50ms, a sub-rotina de leitura de teclado é terminada e volta à rotina principal através da execucão da última instrução de retorno incondicional "RET".

Para finalizar veja através da figura 4.16, a listagem do código objeto da sub-rotina de leitura de teclado.

```
1F6B-   3E FF 32 05 27 3E 00 32 00 40 21 E8 1F 0E 01 06
1F7B-   FE 78 32 00 60 3A 00 B0 A1 FE 00 28 10 23 CB 00
1F8B-   78 FE FE 20 EC CB 01 79 FE 08 C8 18 E2 7E 32 05
1F9B-   27 CD B0 1F 3A 00 B0 A1 FE 00 28 F8 CD B0 1F C9
```

Figura 4.16 - Código Objeto da Sub-rotina de Leitura de Teclado

D) SUB-ROTINA 50 MILISSEGUNDOS (50 MS)

Endereços ocupados: 1FB0 a 1FB9

- Fluxograma:

A sub-rotina de 50 milissegundos, tem a finalidade de retardar o programa, 50 milésimos de segundo, toda vez que for executada.

Os registros "L" e "H" são utilizados como contadores, sendo que o contador H, é usado como base de tempo, enquanto que o contador L, é usado como multiplicador da base de tempo.O fluxograma da sub-rotina de 50 ms pode ser observado através da figura 4.17.

A sub-rotina inteira está "presa" aos registros "H" e "L". Observe que basicamente os registros L e H, utilizados como contadores, estão dispostos de tal forma que um controla o tempo do outro.

No passo 1 da sub-rotina, o registro L é carregado com o valor hexadecimal 32. Este valor em decimal equivale a 50. A finalidade básica deste contador é variar de 32 (hexa) a "00", gerando assim um determinado tempo.

No passo 2, o registro H é carregado com o valor máximo "FF". Este número em decimal equivale a 128. A finalidade básica deste contador é variar de "FF" à "00".

No passo 3, o registro H é decrementado apenas 1 vez, preparando para o próximo passo.

Note que, para decrementar 1 vez o registro H, o microprocessador gasta 1,12 microssegundos. Este valor é obtido a partir da frequência do cristal, e também pelo número de "states" ou períodos de clock, necessários para executar uma instrução de decrementação de registro.

O cristal utilizado no projeto oscila na frequência de 3.57 mhz. Como o período é o inverso da frequência, então temos:

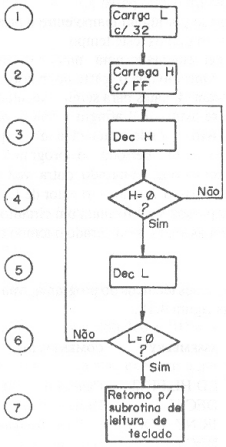


Figura 4.17 - Fluxograma da Sub-rotina de Retardo do 50 Milissegundos

$$T = \frac{1}{3.57} \times 106 = 0{,}280 \times 106 = 280 \text{ nanossegundos}$$

Portanto 1 state demora 280 ns.

A instruçao DEC H, gasta 4 states, logo, o seu tempo de execução será:

$4 \times 280 \times 10^{-9} = 1.120 \times 10^{-9} = 1{,}12 \times 10^{-6} = 1{,}12\ \mu s$

Portanto, a instrução DEC H demora 1,12 microssegundos para ser executada.

Logo após o registro "H" ter sido decrementado, o programa verifica no passo 4, se o valor do conteúdo de H, é igual a zero. Caso não haja igualdade, então o passo 3 é executado novamente, e o registro H é decrementado mais uma unidade. Em seguida, é verificado novamente se o valor do registro "H" é zero.
O circuito entre o passo 3 e 4 se mantém até que "H" seja igual a zero.

Quando isto ocorrer, o programa irá para o passo 5.
Observe que o programa fica preso por algum tempo entre o passo 3 e 4. Mais adiante no estudo do programa, determinaremos este tempo.
No passo 5, o registro L é decrementado uma unidade, gastando para isto 1,12 microssegundos. Este tempo é o mesmo que o gasto no passo 3, porque as instruções "DEC L" e "DEC H" demoram 4 states para serem executadas.
No passo 6, é verificado se o registro "L" atingiu o valor zero, caso negativo, o programa é retornado para o passo 3, onde reciclará entre os passos 3 e 4 durante um determinado tempo. Finalizado este período, o programa entra no passo 5, decrementando novamente o registro L, atingindo outra vez o passo 6.
O programsma então só termina no passo 7, quando o valor de L for igual a zero.
Você deve ter percebido que os passos 3 e 4 formam um circuito de tempo, ao paso que os passos 5 e 6 determinam quantas vezes será gerado o tempo obtido nos passos 3 e 4.

- Programa:
Para compreender melhor os detalhes técnicos do programa, vamos estudar a sub-rotina de retardo completa que está na figura 4.18.

ENDEREÇO SIMBÓLICO	ENDER. FÍSICO	CÓDIGO OBJETO	ASSEMBLEY'	COMENTÁRIOS
	1FB0	2132FF	LD HL,FF32	Carrega o par HL com o valor FF32
LOOP	1FB3	25	DEC H	Decrementa H
	1FB4	20 FD	JR NZ (LOOP)	Se H não for igual a zero vai para loop
	1FB6	2D	DEC L	Decrementa L
	1FB7	20 FA	JR NZ (LOOP)	Se L não for igual a zero vai para loop
	1FB9	C9	RET	Retorna à sub-rotina de LEIT de teclado

Figura 4.18 Sub-rotina de Retardo de 50 milissegundos (50ms)

. LD HL,FF32
Esta instrução, carrega o par HL com o valor FF32.
O valor FF é introduzido no registro H, e o registro L recebe o valor 32.
Como os registros H e L são utilizados como contadores, então, os valores neles carregados, determinam a constante de tempo a ser gerada.

. DEC H
O registro H é decrementado 1 unidade, gastando para isto 1,12 microssegundos. (valor já determinado no passo 3 do fluxograma).

. JR NZ (LOOP)
Quando o registro H é decrementado, seu valor vai diminuindo até atingir zero, ligando assim o flag zero, caso contrário, o flag zero permanece desligado.
A instrução "JR NZ (LOOP)", testa se o flag zero esta desligado. Caso afirmativo, o programa volta para o momento simbólico "LOOP" (1FB3), executando novamente a instrução DEC H.

Este círculo vicioso entre as instruções "DEC H"e "JR NZ (LOOP)", permanece até "H" atingir zero. Quando isto ocorrer, o programa ignora a instrução JR NZ (LOOP), e executa a próxima instrução.
Como o registro "H" foi inicialmente carregado com o valor FF, então podemos concluir que "H" é decrementado 256 vezes, até atingir o valor zero. Pois FF16 = 25610. Note também que a instrução "JR NZ (LOOP), esta no mesmo círculo vicioso que a instrução "DEC H", sendo assim também é executada 256 vezes.
Para determinar o tempo gasto para executar as instruções "DEC H" e "JR NZ (LOOP)", é necessário separá-las para fazer o cálculo. Vejamos inicialmente quanto tempo gasta a instrução DEC H para ser executada 256 vezes:

Dados:

1 state = 280 ns (frequência do cristal = 3.57 mhz)
DEC H = 4 states (dado do fabricante do Z80)
número de ciclos executados pela instrução DEC H = 256

Logo, o tempo gasto para executar 256 ciclos será:

$256 \times 280 \times 10^{-9} \times 4 = 286.720 \times 10^{-9} = 0{,}286$ ns

Vejamos agora, qual o tempo gasto para executar a instrução "JR NZ (LOOP)" 256 vezes:

Dados:

1 states = 280 ms
JR NZ (LOOP) = 10 states (dado do fabricante do Z80)
número de ciclos executados pela instrução JR NZ (LOOP) = 256

Logo, o tempo gasto para executar 256 ciclos será:

$256 \times 280 \times 10^{-9} \times 10 = 716.800 \times 10^{-9} = 0{,}716$ ms

Finalmente, podemos calcular o tempo total gasto pelas 2 instruções:

0,286ms + 0,716ms = 1,002ms

portanto,

o tempo total é de aproximadamente 1 milissegundo.

. DEC L

Esta instrução decrementa o registro L uma vez. Seu tempo de execução não é crítico, pois o registro L é utilizado como contador de ciclos, da 2ª e 3ª instruções do programa.

. JR NZ (LOOP)

Esta instrução tem o mesmo efeito que a instrução da 3ª linha do programa, ou seja, testa o registro "L", e se este não for igual a zero, então o programa vai para o endereço simbólico LOOP.

É importante notar que o tempo de execução desta instrução, bem como a anterior (DEC L), não é crítico, pois estas controlam a quantidade de vezes que a 2ª e 3ª instruções do programa recircularão entre si.
Observe que cada vez que uma instrução "DEC L" for executada, a "DEC H" ; é executada 256 vezes.
Como o registro L foi inicialmente carregado com o valor hexadecimal 32, que equivale a 50 decimal, conclui-se então, que tanto a instrução da 4ª como da 5ª linhas do programa, são executadas 50 vezes, pois o registro L é decrementado de 3216 a 00.
Note também, que cada vez que a instrução de 5ª linha for executada, serão executadas 256 vezes as linhas 2 e 3 do programa. Isto dá para concluir que, as instruções "DEC L" e "JR NZ (LOOP)" (5ª linha), "contam" quantas vezes ciclará o tempo de 1 milissegundo, referente à 2ª e 3ª linhas do programa.
Portanto, o tempo total da sub-rotina de retardo de 50 ms será: 1,002x50 = 50,1 ms
Caso você queira levar em consideração o tempo de execução das instruções da 4ª e 5ª linhas, então terá:

$$\text{DEC L} = 50 \times 280 \times 10^{-9} \times 4(\text{states}) = 56\mu s$$

$$\text{JR NZ(LOP)} = 50 \times 280 \times 10^{-9} \times 10(\text{states}) = 140\mu s$$

portanto,

o tempo da 4ª e 5ª instruções será: 56μs + 140μs = 196μs

Se agregarmos este valor, ao tempo total da sub-rotina, teremos o valor exato de:

```
  50,100 ms
+ 0,196 ms
----------
  50,296 ms
```

. RET
Esta instrução é executada no término da sub-rotina, fazendo o programa voltar à sub-rotina de "leitura de teclado", ou outra rotina qualquer que possa utilizar a sub-rotina de 50 ms.

Finalmente através da figura 4.19, você pode observar a listagem do código objeto da sub-rotina de retardo de 50 milissegundos.

```
1FB0-    21  32  FF  25  20  FD 2D 20  FA C9
```

Figura 4.19 - Código Objeto da Sub-rotina de Retardo de 50 milissegundos (50ms)

E) SUB-ROTINA DE SEPARAÇÃO DE FUNÇÃO (FUNC)

Endereços ocupados: 1C90 a 1CB8

- Fluxograma:

Esta sub-rotina faz parte da rotina principal, e é utilizada logo após uma leitura de teclado, quando uma tecla for acionada.

Seu funcionamento é relativamente simples. Tão logo uma tecla seja acionada, a rotina principal, lê o código da tecla, através da sub-rotina de leitura de teclado. No final desta operação, o código da tecla é guardado no endereço 2705.

Na rotina principal da figura 4.4, na 11ª linha do programa, o valor do código da tecla é lido do endereço 2705, e introduzido no acumulador.

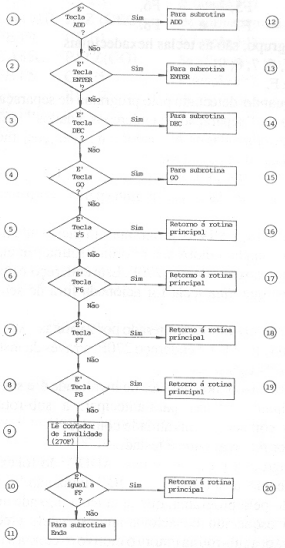


Figura 4.20 - Fluxograma da Sub-rotina de Separação de Função (FUNC)

Mais adiante ainda na figura 4.4, a rotina principal chama a sub-rotina "FUNC" na 14ª linha do programa.
Portanto, é fácil concluir que, quando a sub-rotina de separação de função é iniciada, o código da tecla acionada, já está no acumulador, e pronto para ser comparado por diversas constantes ao longo do programa.

Existem 3 grupos de teclas no nosso teclado.
O primeiro grupo corresponde às funções válidas tais como;

"ADD" (endereçamento),
"ENTR" (entrada),
"DEC" (decremento) e
"GO" (partida).

O segundo grupo são as teclas de função não programadas, ou seja, as de reserva.

São elas: F5, F6,
F7 e F8.

Finalmente, o terceiro grupo, são as teclas hexadecimais

0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
A, B, C, D, E e F.

Este grupo de teclas, quando detectado pelo programa de separação de função (FUNC), é enviado ao programa de "separação de endereços e dados" (ENDA), pois as teclas hexadecimais podem ser utilizadas como dados ou endereços, tudo depende da função selecionada com as teclas de funções válidas.
Nos pasos 1, 12, 2, 13, 3, 14, 4 e 15 do fluxograma, a sub-rotina "FUNC" verifica qual tecla de função válida foi acionada, e a encaminha para a sub-rotina específica da função correspondente.
Nos pasos 5, 16, 6, 17, 7, 18, 8 e 19, o programa simplesmente detecta se uma tecla de função não válida foi acionada, ignora-a, e retorna à rotina principal.
No passo 9, é lido o conteúdo do endereço 270F. Este endereço é utilizado como contador de invalidade. Toda vez que uma tecla for acionada fora de sequência, este contador mudará de valor.
O conteúdo do contador de invalidade é alterado por 2 rotinas. A primeira delas é a rotina principal, esta carrega o valor FF no endereço 270F, através da instrução correspondente à 6ª linha do programa da figura 4.4.
A outra rotina que altera o valor do contador de invalidade é a sub-rotina "ADD", que será estudada mais adiante. Apenas para antecipar, a sub-rotina "ADD" quando executada, carrega o contador de invalidade com "00".
No passo 10, o contador de invalidade é testado.
Se o seu valor for FF, significa que a sub-rotina "ADD" não foi executada, sendo assim, o programa subentende que, a sub-rotina "ADD" ainda não foi executada.
Portanto, é considerado pelo programa, que uma tecla hexadecimal foi acionada, ou acidentalmente, ou foi esquecido de acionar anteriormente a tecla "ADD".
Qualquer que seja a razão, a sub-rotina ignora o motivo e retorna à rotina principal através do passo 20.
Quando o programa atingir o passo 11, significa que o contador de invalidade foi carregado com 00 pela rotina ADD.

Uma vez o programa estando no passo 11, é deslocado para a sub-rotina de separação de endereços e dados (ENDA).

- Programa

A sub-rotina de separação de função (FUNC) pode ser observada através da figura 4.21. Vejamos suas instruções de sequência.

ENDEREÇO SIMBÓLICO	ENDER. FÍSICO	CÓDIGO OBJETO	ASSEMBLY	COMENTÁRIOS
	1C90	FE F1	CP F1	É função F1?
	1C92	CAD51E	JP Z(ADD)	Se afirmativo vai para sub-rotina ADD
	1C95	FE F2	CP F2	É função F2?
	1C97	CA351D	JP Z (ENTR)	Se afirmativo vai para sub-rotina ENTR
	1C9A	FE F3	CP F3	É função F3?
	1C9C	CA101D	JP Z (DEC)	Se afirmativo vai para sub-rotina DEC
	1C9F	FE F4	CP F4	É função F4?
	1CA1	CA85C	PJP Z(GO)	Se afirmativo vai para sub-rotina GO
	1CA4	FE F5	CP F5	É função F5?
	1CA6	C8	RET Z	Se afirmativo retorna para rotina principal
	1CA7	FE F6	CP F6	É fução F6?
	1CA9	C8	RET Z	Se afirmativo retorna para rotina principal
	1CAA	FE F7	CP F7	É função F7?
	1CAC	C8	RET Z	Se afirmativo retorna para rotina principal
	1CAD	FE F8	CP F8	É função F8?
	1CAF	C8	RET Z	Se afirmativo retorna para rotina principal
	1CB0	3A0F27	LD A,(270F)	Lê contador de invalidade
	1CB3	FEFE	CP FF	É igual a "FF"?
	1CB5	C8	RET Z	Se afirmativo retorna para rotina principal
	1CB6	C3C01C	JP (ENDA)	Se negativo vai para sub-rotina ENDA

Figura 4.21 - Sub-rotina de Separação de Função (FUNC)

. CP F1

Como já vimos no fluxograma, o acumulador foi carregado com o código da tecla acionada.

Esta primeira instrução compara o acumulador com o código F1 correspondente à tecla ADD. Se houver igualdade na comparação, então o flag zero é ligado, preparando o programa para a próxima instrução.

. JR Z (ADD)

Caso o resultado da comparação da instrução anterior tenha sido igualdade, significa que a tecla acionada foi a "ADD", sendo assim o programa salta para a sub-rotina "ADD" localizada no endereço "1ED5". Caso a igualdade não tenha sido satisfeita, a instrução "JR Z(ADD)" é abortada, e o programa continua na próxima instrução.

. CP F2
Esta instrução compara se a tecla acionada (valor no acumulador), é F2, cujo código correspondente à tecla "ENTER". Caso afirmativo, liga o flag zero.

. JR Z (ENTR)
Se a tecla acionada foi "ENTER", então o programa continua no endereço da sub-rotina ENTER (1D35), caso contrário, o programa executa a instrução seguinte.

. CP F3
F3 é o código da tecla "DEC" (decremento). A comparação CP F3 liga o "flag zero" se a tecla acionada foi "DEC".

. JR Z (DEC)
Se houver igualdade na comparação da instrução anterior, então o programa vai para o endereço "1D10" executar a rotina DEC, caso contrário, executa a próxima instrução

. CP F4
F4 é o código da tecla "GO" '(partida). Se a tecla acionada foi "GO", então o "flag zero" é ligado.

. JR Z (GO)
Se na comparação anterior houve igualdade, então o programa salta para 1C85, executando a sub-rotina GO, caso contrário, a próxima instrução é executada.

. CP F5
F5 é o código da tecla "F5". Se esta tecla foi acionada, o flag zero é ligado.

. RET Z
F5 é uma tecla não programada, portanto se o resultado da comparação anterior for igualdade, então o programa retorna à rotina principal.

. CP F6
F6 é o código da tecla "F6" se esta tecla foi acionada, o flag zero é ligado.

. RET Z
F6 também é uma tecla não programada, portanto se o resultado da comparação anterior for igualdade, então o programa à rotina principal

. CP F7
F7 é o código da tecla "F7". Se esta foi acionada, o flag zero é ligado.

. RET Z
F7 também é uma tecla não programada, portanto se o resultado da comparação anterior for igualdade, então o programa retorna à rotina principal.

. CP F8
F8 é o código da tecla "F8". Se esta foi acionada, o flag zero é ligado.

. RET Z
F8 é a última tecla não programada, portanto se o resultado da comparação anterior for igualdade, então o programa retorna à rotina principal.

. LD A (270F)
Esta instrução lê o contador de invalidade, carregando seu conteúdo no acumulador.

. CP FF
Compara se o acumulador é igual a "FF", caso afirmativo liga o "flag zero". Lembre-se que o conteúdo do acumulador é o conteúdo do contador de invalidade.

. RET Z
Se o contador de invalidade for igual a "FF" então a sub-rotina é terminada, encaminhando o programa para a rotina principal.
Caso o contador de invalidade for diferente de "FF", então é sinal que uma tecla hexadecimal foi executada corretamente, sendo assim, a instrução seguinte é executada.

. JR (ENDA)
Esta última instrução é executada, somente se a comparação da 18ª linha do programa não for uma igualdade.
Quando a comparação for desigual, o programa então salta para o endereço "1CC0", com o objetivo de executar a sub-rotina de separação de endereços e dados (ENDA). Na figura 4.22, pode ser observada a sequência dos códigos, objeto da sub-rotina de separação de função.

```
1C90-   FE F1 CA D2 1E FE F2 CA 35 1D FE F3 CA 10 1D FE
1CA0-   F4 CA 85 1C FE F5 C8 FE F6 C8 FE F7 C8 FE F8 C8
1CB0-   3A 0F 27 FE FF C8 C3 C0 1C
```

Figura 4.22 - Código Objeto da Sub-rotina de Separação de Função (FUNC)

F) SUB-ROTINA "ADD" (ADDRESS):
Endereços ocupados: 1ED2 a 1EE5

- Fluxograma:
Tão logo o microcomputador é ligado, após a tecla de RESET ser acionada, a primeira função que deve ser selecionada é a "ADD". Isto é realizado acionando-se a tecla ADD. A função ADD quando ativada, posiciona alguns registros, e prepara o fluxo do programa para receber 4 dígitos de endereço, ou seja, é uma função de preparação de endereçamento.

Através da Figura 4.23, pode ser observado o fluxograma da sub-rotina "ADD" (endereçamento).

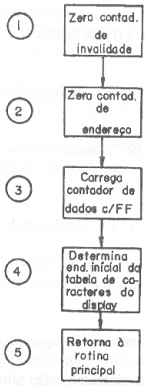


Figura 4.23 - Fluxograma da Sub-rotina "ADD"

Trata-se de uma sub-rotina bem simples, pois não contém blocos de decisão lógica.
No passo 1, o contador de invalidade é carregado com o valor "00", preparando-o para a sub-rotina "ENDA".
No passo 2, o contador de endereço também é zerado, preparando-o para ser utilizado na sub-rotina "1 END".
No passo 3, o contador de dados é carregado com "FF", preparando-o para ser utilizado na sub-rotina "1 DAD".
No passo 4, o registro de endereço da tabela de caracteres é carregado com o endereço inicial da tabela, que mostrará a mensagem "ADD", tão logo a tecla ADD seja acionada.
Note que a mensagem ADD só é apresentada no display, no momento que for terminada a sub-rotina ADD, e iniciada a sub-rotina "DISP" (DISPLAY) dentro da rotina principal.
No passo 5, a sub-rotina ADD é terminada e o programa volta para o programa principal, aguardando que a primeira tecla hexadecimal de endereço seja acionada.

- Programa

O programa completo pode ser observado através da Figura 4.24

ENDEREÇO SIMBÓLICO	ENDER. FÍSICO	CÓDIGO OBJETO	ASSEMBLY	COMENTÁRIOS
	1ED2	3E00	LD A,00	Zera acumulador
	1ED4	320F27	LD (270F),A	Zera contador de invalidade
	1ED7	320E27	LD (270E),A	Zera contador de endereços
	1EDA	3EFF	LD A,FF	Carrega FF no acumulador
	1EDC	320D27	LD (270D),A	Carrega FF no contador de dados
	1EDF	21E81E	LD HL,1EE8	Carrega HL com 1EE8 (end.inicial - tabela 3)
	1EE2	220327	LD (2703),HL	Carrega registro de end. de tabela com 1EE8
	1EE5	C9	RET	Retorna à rotina principal

Figura 4.24 - Sub-rotina ADD

Vejamos a seguir, suas instrucões em sequência.

. LD A,00
Esta instrução zera o acumulador, com o objetivo de zerar endereços de memória nas 2 próximas instruções.

. LD (270F), A
Transfere o conteúdo do acumulador (00) para o endereço 270F, zerando o contador de invalidade.

. LD (270E), A
Transfere o conteúdo do acumulador (00) para o endereço 270E, zerando o contador de endereço.

. LD A,FF
Carrega o acumulador com o valor FF, preparando-o para transferir seu conteúdo para o endereço determinado pela próxima instrução.

. LD (270D), A
Transfere o conteúdo do acumulador (FF) para o endereço 270D, preparando então o registro de endereço para ser utilizado na sub-rotina "1 END".

. LD HL, 1EE8
Os registros de endereços de tabela "2703" e 2704", não podem ser carregados diretamente por uma determinada constante, pois não existe este tipo de instrução no repertório de instruções do Z80. Por este motivo, é necessário antes, carregar o valor desejado nos registros HL, para depois transferir para os registros de endereço de tabela. A instrução LD HL, 1EE8 carrega "HL" com 1EE8, preparando para a próxima instrução.

. LD (2703),HL

A instrução anterior carregou o registro "L" com o valor "E8", enquanto que o registro "H" foi carregado com o valor "1E".

A instrução "LD (2703),HL" transfere o conteúdo de "L" (E8), para o endereço 2703, e o conteúdo de "H" (1E) para o endereço 2704.

Portanto, os registros de endereço de tabela, conterão o valor 1EE8, tão logo a instrução "LD (2703),HL" seja executada.

O valor 1EE8, aponta o endereço inicial da tabela, que apresentará no display a mensagem "ADD".

Veja na figura 4.25, os caracteres que compõem a tabela-3 (TB-3).

```
1EE8-   77  A
1EE9-   7C  D
1EEA-   7C  D
1EEB-   FF  FIM
```

Figura 4.25 - Tabela para Gerar a Mensagem ADD (TB-3)

Esta última instrução retorna à rotina principal, apresentando no display a mensagem "ADD", tão logo a sub-rotina "display" seja executada como parte do programa principal.

Veja através da figura 4.26, os códigos objeto da sub-rotina "ADD".

```
1ED2-   3E 00 32 0F 27 32 0E 27 3E FF 32 0D 27 21 E8 1E
1EE2-   22 03 27 C9
```

Figura 4.26 - Código Objeto da Sub-rotina ADD

G) SUB-ROTINA DE SEPARAÇÃO DE ENDEREÇOS/DADOS (ENDA)

Endereços ocupados: 1CC0 a 1CF0

- Fluxograma:

A finalidade básica da sub-rotina de separação de endereço e dados (ENDA), é encaminhar o valor das 4 primeiras teclas, aos displays de endereço, ao passo que, a 5ª e 6ª teclas, são encaminhadas aos 2 displays de dados.

É conveniente notar que, a sub-rotina "ENDA", não processa o código da tecla, apenas, encaminha para o programa que trata de cada tecla, na devida sequência.

Trata-se de uma sub-rotina importante dentro do programa monitor, pois encaminha cada tecla acionada, à sua sub-rotina de tratamento específico.

Apenas para melhor compreensão, suponha que após acionada a tecla "ADD", são acionadas as 4 seguintes teclas na sequência: 2, 7, 0, 1.

No término do acionamento da 4ª tecla, o display conterá o valor 2701, sendo que os últimos dígitos da direita, estarão apagados. Isto significa que o valor 2701 está sendo endereçado na memória.

Em seguida, acionamos a tecla "Enter" imediatamente após o seu acionamento, o display mostrará o valor 2701XX, onde XX poderá ser qualquer valor, dependendo que informação continha o endereço 2701.
Para terminar a sequência, suponha que sejam digitadas as teclas "4" e "8". Tão logo seja acionada a última tecla, o valor 48 é armazenado no endereço "2701" e mantém apagados os dígitos de dados, aguardando para armazenar o dado do próximo endereço. Mais adiante no fim deste capítulo, você entenderá melhor a sequência de manuseio do teclado no item referente à operação do APL-80.
Vejamos agora, o funcionamento dos 18 passos do fluxograma da sub-rotina ENDA (figura 4.27).

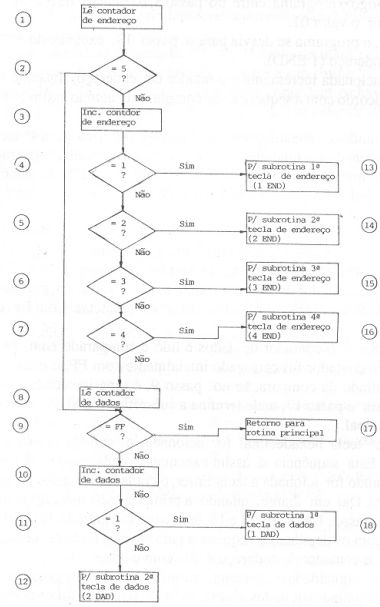


Figura 4.27 - Fluxograma da sub-rotina de separação ender./dados(ENDA)

Inicialmente na rotina principal, figura 4.4 4ª linha do programa, o contador de endereço é zerado.
A primeira providência da sub-rotina "Enda", é ler o contador de endereço (passo 1).
A finalidade do contador de endereço, é de encaminhar o programa para as sub-rotinas de 1ª a 4ª teclas de endereço, ou 1ª e 2ª teclas de dados.
No passo 2 do fluxograma, o contador de endereço é comparado com o valor 5. Caso seu conteúdo seja 5, então o programa salta para o trecho do programa, onde são tratadas as teclas de dados, caso o conteúdo não seja 5, então o programa continua no passo 3, onde serão tratadas as teclas de endereço 1 a 4.
Você deve estar lembrado que ao iniciar a rotina principal o contador de endereço foi zerado. Tão logo o programa entre no passo 3, o contador de endereço é incrementado, passando a ter o valor 01.
No passo 4, o programa se desvia para o passo 13, executando então a sub-rotina da 1ª tecla de endereço (1 END).
Cada tecla acionada incrementa o contador de endereço, fazendo assim o programa servi-las de acordo com a sequência de entrada, executando assim os passos 5, 14, 6, 15, 7 e 16.
Quando o contador de endereço atingir o valor 04, significa que a 4ª tecla de endereço foi acionada. Daí em diante qualquer tecla hexadecimal digitada, é ignorada pela sub-rotina "ENDA", a menos que a tecla "Enter" seja acionada. Quando isto ocorrer, o contador de dados no passo 8 terá o valor 00, liberando o programa para gravar dados na memória.
Suponha agora que a tecla Enter não foi acionada, e que a 5ª tecla está sendo digitada. Quando isto ocorrer, observe no passo 2, que a comparação não será 5, pois a penúltima tecla digitada (4ª tecla), posicionou o contador de endereço com "04".
No passo 3, o contador de endereço passa a ter o valor 05, portanto as comparações dos passos 4, 5, 6 e 7 serão desiguais, não sendo satisfetas. Com isto o programa atinge o passo 8.
Nos passos 8 e 9, o contador de dados é lido e comparado com FF. Como na rotina principal este contador foi carregado inicialmente com FF, e nada ainda o modificou, então o resultado da comparação no passo 9, é uma igualdade, conduzindo assim o programa para o passo 17, onde termina a sub-rotina "ENDA" e o programa vai para a rotina principal.
Quando a 6ª tecla hexadecimal for acionada, a sub-rotina executará os passos 1, 2, 8, 9 e 17. Esta sequência é assim executada a cada nova tecla "Hexa" digitada.
Somente quando for acionada a tecla Enter, o contador de dados do passo 8, passará a ter o valor 00. Daí em diante, quando a primeira tecla hexa for acionada, o programa executará os passos 1, 8, 9, 10, 11 e 18, dando início à entrada da primeira tecla de dados.
Vejamos agora os detalhes da sequência para entrada da tecla de dados.
No passo 1, o contador de endereço é lido com o valor "05".
No passo 10, a igualdade é satisfeita, encaminhando o programa para o passo 8.
No passo 8, o contador de dados é lido com "00" pois este foi zerado quando foi digitada a tecla Enter.
No passo 9, a comparação não é satisfeita, direcionando o programa para o passo 10.

No passo 10, o contador de dados passa a ter o valor 01.
No passo 11, a igualdade é satisfeita, encaminhando a sub-rotina para o passo 18, onde é atendida a 1ª tecla de dados.
Vejamos agora, qual a sequência de funcionamento para a 2ª tecla de dados.
Quando a próxima tecla hexa for digitada, o programa incrementará o contador de dados no passo 10, não satisfará a condição no passo 11 e continuará no passo 12, onde finalmente será atendida a 2ª tecla de dados.
Portanto, o circuito completo para atender à 2ª tecla de dados será: 1, 2, 8, 9, 10, 11 e 12.
É importante comentar que, a sequência ímpar de acionamento do teclado (3ª, 5ª, 7ª etc) executará a mesma função que a 1ª tecla de dados, ou seja, serão encaminhadas para a sub-rotina "1 DAD".
A sequência par do acionamento do teclado (2ª, 4ª, 6ª etc) executará a mesma função que a 2ª tecla de dados, ou seja, serão encaminhadas para a sub-rotina "2 DAD".
As sub-rotinas "1 DAD" e "2 DAD" automaticamente controlam o chaveamento das entradas de sequência par ou ímpar, modificando o valor do contador de dados.

- Programa

O programa completo da sub-rotina de separação de endereço e dados pode ser notado através da figura 4.28.

ENDEREÇO SIMBÓLICO	ENDER. FÍSICO	CÓDIGO OBJETO	ASSEMBLY	COMENTÁRIOS
	1CC0	3A0E27	LD A,(270E)	Lê contador de endereço
	1CC3	EE05	CP 05	E' = 5 ?
	1CC5	2818	JR Z(DADO)	Se = 5 vai para dado
	1CC7	3C	INC A	Incrementa acumulador
	1CC8	320E27	LD (270E),A	Deposita valor incrementado no cont. de End.
	1CCB	FE01	CP 01	E' = 1 ?
	1CCD	CAA71E	JP Z(1End)	Se = 1 vai para sub. 1ª tecla End (1End)
	1CD0	FE 02	CP 02	E' = 2 ?
	1CD2	CA771E	JP Z(2End)	Se = 2 vai para sub. 2ª tecla End (2End)
	1CD5	FE 03	CP 03	E' = 3 ?
	1CD7	CA571E	JP Z(3End)	Se = 3 vai para sub. 3ª tecla End (3End)
	1CDA	FE 04	CP 04	E' = 4 ?
	1CDC	CA251E	JP Z(4End)	Se = 4 vai para sub. 4ª tecla End (4End)
DADO	1CDF	3A0D27	LD A,(270D)	Lê contador de dados
	1CE2	FEFF	CP FF	E' = FF ?
	1CE4	C8	RET Z	Se = FF retorna à rotina principal
	1CE5	3C	INC A	Incrementa acumulador
	1CE6	320D27	LD (270D),A	Deposita valor increm. no contador de dados
	1CE9	FE 01	CP 01	E' = 1 ?
	1CEB	CA001E	JP Z(1DAD)	Se = 1 vai para 1ª tecla de dados (1DAD)
	1CEE	CE901D	JP (2DAD)	Se = 1 vai para 2ª tecla de dados (2DAD)

Figura 4.28 - Sub-rotina de separação de endereço e dados (ENDA).

. LD A,(270E)
Carrega o acumulador com o conteúdo do endereço 27OE, ou seja, lê o contador de endereço.
. CP 05
Compara o conteúdo do endereço 27OE que está no acumulador, com a constante 05. Se a comparaçao for uma igualdade, então o flag zero é ligado.

. JR Z(DADO)
Se o flag zero estiver ligado, o programa salta para o endereço simbólico "Dado", caso contrário, o programa continua na linha seguinte.
Observe que esta instrução, separa os códigos das teclas que correspondem a dados, ou endereços.

. INC A
Incrementa o conteúdo do acumulador, preparando para retornar este valor incrementado ao contador de dados.

. LD (27OE),A
Carrega o conteúdo do acumulador incrementado no endereço (27OE), incrementando assim o registro de endereço.

. CP 01
Compara se o acumulador quando foi incrementado, atingiu o valor 01. Caso afirmativo, liga o flag zero.

. JP Z(1END)
Se na instrução anterior houve uma igualdade, é sinal que a primeira tecla de endereço foi acionada sendo assim, o programa é deslocado para a sub-rotina "1END" que atenderá e processará o código da 1ª tecla de endereço.
Caso o flag zero não esteja ligado, o programa continua na linha seguinte.

. CP 02
Compara se o acumulador é igual a 02. Caso afirmativo, liga o flag zero.

. JP z(2END)
Se na instrução anterior houve igualdade, então o programa é deslocado para executar a sub-rotina referente à 2ª tecla de endereço (2END).

. CP 03
Compara se o acumulador contém o valor 03, caso afirmativo, liga o flag zero.

. JP z(3END)
Se na comparação anterior houve igualdade, então o programa é deslocado para executar a sub-rotina referente à 3ª tecla de endereço (3END).

. CP 04
Compara se o acumulador contém o valor 04, caso afirmativo, liga o flag zero.

. JP Z(4END)
Se na comparação anterior houve igualdade, então o programa é deslocado para executar a sub-rotina referente à 4ª e última tecla de endereço (4END).

. LD A,(270D)
Carrega o acumulador com o conteúdo do endereço 270D, ou seja, lê o contador de dados.
Note que este contador, é utilizado para selecionar a sequência ímpar ou par de teclas de dados acionadas, direcionando assim o programa para as sub-rotinas 1DAD e 2DAD.

. CP FF
Compara se o conteúdo do acumulador referente ao contador de dados é igual a FF. Caso afirmativo, liga o flag zero.

. RET Z
Se a instrução anterior ligou o flag zero, é sinal que a tecla enter ainda não foi digitada, então o programa retorna à rotina principal.
Note que a instrução "Ret z" é um retorno condicional, e só é executada quando o flag zero estiver ligado.

. INC A
Incrementa o conteúdo do acumulador. Como na 14ª linha do programa, o acumulador foi carregado com o valor do contador de dados, então a instrução INC A da 17ª linha do programa prepara para incrementar o contador de dados.

. LD (270D),A
Completa a instrução anterior, incrementando o registro de dados.
Note que quando o programa atinge pela primeira vez a 18ª linha, significa que a primeira tecla de dados foi acionada.

. CP 01
Compara o acumulador com o valor 01. Se houver igualdade, então o flag zero é ligado.

. JP Z(1DAD)
Se o flag zero foi ligado na instrução anterior, é sinal que a 1ª tecla de dados foi acionada, sendo assim, o programa é desviado para executar a sub-rotina "1DAD".

. JP (2DAD)
Caso o flag zero não tenha sido ligado, a instrução anterior é ignorada pela sub-rotina ENDA, e o programa é deslocado para executar a sub-rotina "2DAD".

Para finalizar, observe os códigos objeto, referentes à sub-rotina "ENDA" na figura 4.29.

```
1CC0-   3A 0E 27 FE 05 28 18 3C 32 0E 27 FE 01 CA A7 1E
1CD0-   FE 02 CA 77 1E FE 03 CA 57 1E FE 04 CA 25 1E 3A
1CE0-   0D 27 FE FF C8 3C 32 0D 27 FE 01 CA 00 1E C3 90
1CF0-   1D
```

Figura 4.29 - Códigos objeto da sub-rotina de separação de endereço e dados (ENDA).

H) SUB-ROTINA 1ª TECLA DE ENDEREÇO (1END)
Endereços ocupados: 1EA7 a 1ECF

- Fluxograma:
A sub-rotina "1END" tem a finalidade de carregar o registro de endereço MSD com o código hexadecimal referente ao 1º dígito MSD.
Uma outra finalidade desta sub-rotina, é preparar determinados registros para a sub-rotina de display. Acendendo o 6º display com o valor da primeira tecla de dados, e apagando todos outros displays.
Vejamos agora através da figura 4.30, o funcionamento do fluxograma da sub-rotina "1END".
No passo 1, é lido o código referente à primeira tecla de endereço.
No passo 2, o código adquirido no passo anterior é armazenado no registro de endereço mais significativo.
Veja através da figura 4.2, que são utilizados 2 endereços de memória com a finalidade de guardar 4 dígitos de informação de endereço. Estes dígitos correspondem às 4 teclas de endereço digitadas.
A sub-rotina "1END" e "2END" guardam os valores MSD de endereço.
A sub-rotina "3END" e "4END" guardam os valores LSD de endereço.
No passo 3, é utilizada uma sub-rotina que transforma o código da tecla digitada, num outro código de sete segmentos, capaz de acender o display com o mesmo número digitado.
Esta sub-rotina será estudada mais adiante. Por enquanto, apenas tenha em mente que ela torna o código do teclado, compatível com o display.
Para que os displays possam ser acesos de forma correta, utilizamos os registros de códigos do display. Estes contêm códigos de sete segmentos prontos para serem enviados para o display.
os endereços 2706, 2707, 2708, 2709, 270A e 270B, correspondem aos displays 6, 5, 4, 3, 2 e 1 respectivamente, conforme figura 4.2.
Portanto, cada display possui seu próprio registro, por este motivo é que no passo 4, o código de 7 segmentos obtido no passo anterior, vai para o endereço 2706. Note que o registro de código do 6º display é 2706.
No passo 5, todos os registros de códigos de display são zerados, exceto o registro 2706, que contém neste instante, o código da 1ª tecla de endereço, já transformado em 7 segmentos.

A razão de zerar os registros de códigos do display, é manter apagados os displays 5 a 1 durante a execução da sub-rotina display, dentro da rotina principal.

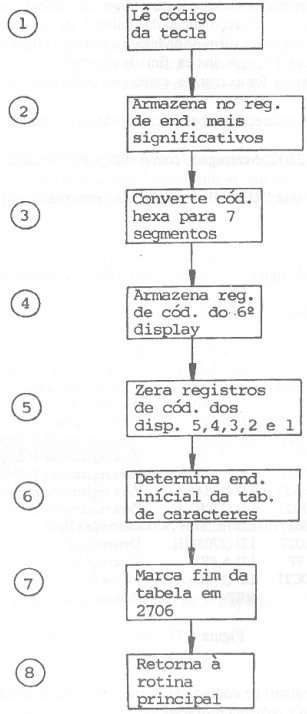


Figura 4.30 - Fluxograma sa sub-rotina 1ª tecla de endereço (1END)

Um detalhe importante, é que toda a informação que desejamos apresentar no display,

deve estar presente numa área da memória e em sequência, tal como você deve ter percebido, nas tabelas da mensagem "APL-80", figura 4.5, e "ADD", figura 4.25.
Para apresentar no display, os valores digitados, utilizamos um processo semelhante. Inicialmente, são carregados os endereços de código do display, com códigos de 7 segmentos, em seguida, o endereço inicial da "tabela" de caracteres a serem "displayados" é fornecido para os registros de endereço de tabela. Finalmente, no endereço 270C, é gravado o código FF, que indica fim de caracteres.
Quando a sub-rotina display for executada, então esta exibirá os dados dos endereços 2706 e 270B.
No passo 6, o registro de endereço da tabela, é carregado com o endereço inicial da tabela de caracteres.
No passo 7, o endereço 270C, é carregado com o código FF, fazendo uma marca de fim dos caracteres.
No passo 8, a sub-rotina "1END" é terminada, retornando o programa à rotina principal.

- Programa
Vejamos agora através da figura 4.31, a sub-rotina 1END completa e em sequência.

ENDEREÇO SIMBÓLICO	ENDER. FÍSICO	CÓDIGO OBJETO	ASSEMBLY	COMENTÁRIOS
	1EA7	3A0527	LD A,(2705)	Lê código da tecla
	1EAA	320227	LD (2702),A	Armazena no registro de END. MSD
	1EAD	CD201F	CALL (HEX7)	Cconverte hexa para 7 segmentos
	1EB0	320627	LD (2706),A	Armazena no registro do 6º display
	1EB3	3E00	LD A,00	Zera acumulador
	1EB5	320727	LD (2707),A	Zera registro do 5º display
	1EB8	320827	LD (2708),A	Zera registro do 4º display
	1EBB	320927	LD (2709),A	Zera registro do 3º display
	1EBE	320A27	LD (270A),A	Zera registro do 2º display
	1EC1	320B27	LD (270B),A	Zera registro do 1º display
	1EC4	210627	LD HL,2706	Determina Endr. inicial tab. de caracteres
	1EC7	220327	LD (2703),HL	Determina Endr. inicial tab. de caracteres
	1ECA	3E FF	LD A,FF	Ccarrega FE no acumulador
	1ECC	320C27	LD (270C),A	Marca fim de tabela
	1ECF	C9	RET	Retorna à rotina princial

Figura 4.31 - Sub-rotina 1ª tecla de endereço (1END).

. LD A,(2705)
Esta instrução lê o registro de código da tecla acionada e deixa-o disponível no acumulador, aguardando a próxima instrução.
Note que o código em 2705, correspondente à primeira tecla de endereço digitada e ao longo do programa, aparecerá no 6º display do lado esquerdo.

. LD (2702),A
De acordo com a figura 4.2, o endereço 2702 é utilizado como registro de endereço MSD.
Neste registro, são guardados os 2 valores de endereço correspondentes à 1ª e 2ª teclas de endereço.
A primeira tecla de endereço é auxiliada pela sub-rotina 1END, enquanto que a 2ª, é auxiliada pela sub-rotina "2END" que será estudada mais adiante.
A instrução "LD (2702),A", carrega no registro de endereço mais significativo (MSD), o código da 1ª tecla de endereço.

. CALL (Hex7)
O código da 1ª tecla de endereço, obtido na 1ª instrução desta sub-rotina, ainda não foi destruído. Portanto, a sub-rotina Hex7 transforma-o em um código de 7 segmentos, tornando-o compatível para excitar o display na forma correta.

. LD (2706),A
Uma vez que o código da primeira tecla foi transformado em 7 segmentos através da última instrução, então resta guardá-lo no seu registro apropriado.
Isto é realizado pela instrução LD (2706),A. Portanto, quando a sub-rotina "Display" for executada, o código armazenado em 2706 será apresentado no 6º display (1º display da esquerda).

. LD A,00
Zera o acumulador, preparando-o para as próximas instruções.

. LD (2707),A
Como a última instrução zerou o acumulador, então "LD (2707),A" zera o registro de código do 5º display.

. LD (2708),A
Da mesma forma que a instrução anterior, zera o registro de código, referente ao 4º display.

. LD (2709),A
Zera também o registro de código, referente ao 3º display.

. LD (270A),A
Zera o registro de código, referente ao 2º display.

. LD (270B), A
Zera o registro de código, referente ao 1º display.

. LD HL,2706
Carrega o valor 2706, no par de registros HL, determinando assim para a próxima instrução, o endereço inicial da tabela de caracteres, a serem apresentados no display.

. LD (2703),HL
Carrega o valor de "L" (06) no endereço 2703, em seguida carrega o valor de "H"(27) no endereço 2704.
Como os endereços 2703 e 2704, são os registros de endereço de tabela de caracteres a serem exibidos no display, então estes conterão o endereço inicial da tabela "2706", onde estão guardados em sequência, os códigos de 7 segmentos.
Note que nesta altura do programa, quando a rotina principal mandar executar a sub-rotina display, somente o 6º display, acenderá com o valor digitado referente à 1ª tecla de endereço.

. LD A,FF
Prepara o acumulador para gerar marca de fim na próxima instrução.

. LD (270C),A
Grava o código "FF" no endereço 270C, determinando o fim da tabela de caracteres.

. RET
Retorna à rotina principal.
Para finalizar observe através da figura 4.32, os códigos objeto referente à sub-rotina 1ª tecla de endereço "1END".

```
1EA7-   3A 05 27 32 02 27 CD 20 1F 32 06 27 3E 00 32 07
1EB7-   27 32 08 27 32 09 27 32 0A 27 32 0B 27 21 06 27
1EC7-   22 03 27 3E FF 32 0C 27 C9
```

Figura 4.32 - Códigos objeto da sub-rotina 1ª tecla de endereço (1END).

1) SUB-ROTINA: 2ª TECLA DE ENDEREÇO (2END)
Endereços ocupados:

- Fluxograma:
A finalidade básica da sub-rotina "2END", é exibir no 5º display, o valor da 2ª tecla de endereço acionada.
Esta sub-rotina também mostra no 6º display, o valor da 1ª tecla de endereço acionada, por este motivo, o conteúdo do registro de endereço MSD é deslocado e somado com o novo valor, gerando 2 dígitos hexadecimais.
Vejamos para maiores esclarecimentos, o fluxograma apresentado na figura 4.33.
No passo 1, o valor do registro de endereço MSD é lido a fim de obter o valor da 1ª tecla de endereço.
No passo 2, o conteúdo do registro de endereço MSD é deslocado 4 bits para esquerda, passando o dígito do lado direito para o lado esquerdo, gerando então uma lacuna para receber posteriormente o código da 2ª tecla acionada.
Para melhor compreensão suponha que a 1ª tecla de endereço acionada foi "F", e a 2ª tecla foi "8".

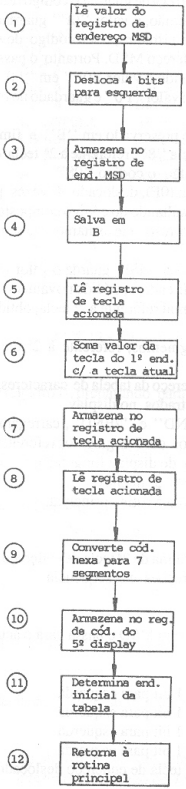


Figura 4.33 - Fluxograma da sub-rotina 2ª tecla de endereço (2END)

A sub-rotina "1END", se encarrega de guardar o código referente à tecla "F", no registro de endereço MSD, resultando então, o valor "0F" guardado no endereço 2702.
Como a tecla "F" foi digitada primeiro, o seu código deve ser carregado no lado mais significativo do registro de endereço MSD. Portanto o passo 2 do fluxograma, desloca 4 bits para esquerda, transformando o valor "0F" em "F0".
Veja agora, que o nosso suposto valor "F0" é guardado no registro de endereço MSD. Isto é realizado pelo passo 3.
No passo 4, o mesmo valor é preservado em "B" a fim de ser utilizado mais tarde.
No passo 5, a nossa suposta tecla "8", referente à 2ª tecla de endereço é lida no registro de tecla acionada, obtendo então o código "08".
No passo 6, o valor da 1ª tecla (0F), deslocado 4 vezes para esquerda (F0), é somado com o valor da 2ª tecla (08), resultando no código composto "F8", ou seja, F0 + 08 = F8.
Observe que o código "F8" corresponde exatamente à sequência de teclas acionadas da direita para esquerda.
No passo 7, o registro de endereço MSD, guarda o valor gerado no passo anterior (F8).
No passo 8, o valor da 2ª tecla de endereço é lido novamente do registro de tecla acionada.
No passo 9, o código hexadecimal referente à 2ª tecla, obtido no passo 8, é transformado em código de 7 segmentos.
No passo 10, o valor de 7 segmentos, referente à 2ª tecla acionada, é armazenado no registro do 5º display (2707).
No passo 11, o registro de endereço da tabela de caracteres, é carregado com o endereço inicial dos dados a serem mostrados no display.
Note que as sub-rotinas "1END" e "2END", carregam os registros de código dos displays 6 e 5, com os códigos de 7 segmentos referentes à 1ª e 2ª teclas de endereço. Portanto, quando a sub-rotina de display for gerada, os valores correspondentes à 1ª e 2ª teclas apresentar-se-ão no 6º e 5º displays.
No passo 12, o programa retorna à rotina principal.

- Programa

O programa completo da sub-rotina de 2ª tecla de endereço (2END) está na figura 4.34. Vejamos portanto, suas instruções em sequência.

. LD A,(2702)

Lê o valor do registro de endereço MSD e passa para o acumulador, ou seja, lê o código da 1ª tecla de endereço.

. RLC A desloca acumulador 1 bit para esquerda
. RLC A desloca acumulador 1 bit para esquerda
. RLC A desloca acumulador 1 bit para esquerda
. RLC A desloca acumulador 1 bit para esquerda
obtendo então o código da 1ª tecla de endereço deslocado 4 bits para esquerda.

. LD (2702),A

Transfere o código da 1ª tecla deslocado para o registro de endereço MSD.

ENDEREÇO SIMBÓLICO	ENDR. FÍSICO	CÓDIGO OBJETO	ASSEMBLY	COMENTÁRIOS
	1E77	3A0227	LD A,(2702)	Lê valor MSD do reg. de endereço
	1E7A	07	RLC A	Desloca acumulador para esquerda
	1E7B	07	RLC A	Desloca acumulador para esquerda
	1E7C	07	RLC A	Desloca acumulador para esquerda
	1E7D	07	RLC A	Desloca acumulador para esquerda
	1E7E	320227	LD (2702),A	Armazena no reg. de endr. MSD
	1E81	47	LD B,A	Salva em B
	1E82	3A0527	LD A,(2705)	Lê registro de tecla acionada
	1E85	80	ADD A,B	Soma 1a. com 2a. tecla de endereço
	1E86	320227	LD (2702),A	Armazena no reg. de endr. MSD
	1E89	3A0527	LD A,(2705)	Lê registro de tecla acionada
	1E8C	CD201F	CALL (HEX7)	Transforma em código 7 segmentos
	1E8F	320727	LD (2707),A	Armazena no reg. de código do 5o. display
	1E92	210627	LD HL,2706	Determina endr. inicial da tabela
	1E95	220327	LD (2703),HL	Carrega reg. endr. tab. c/ endr. inicial
	1E96	C9	RET	Retrona à rotina principal

Figura 4.34 - Sub-rotina 2a. tecla de endereço (2END)

. LD B,A
Salva valor deslocado para o registro B, preparando para somar com o valor da 2ª tecla.

. LD A,(2705)
Lê o registro de tecla acionada, obtendo assim o código referente à 2ª tecla de endereço.

. ADD A,B
Soma o código deslocado da 1ª tecla de endereço com o código da segunda tecla, obtendo como resultado no acumulador, o código de 2 dígitos referente à 1ª e 2ª teclas.

. LD (2702),A
Retorna código composto para o registro de endereço MSD.

. LD A(2705)
Lê código referente à 2ª tecla de endereço acionada, carregando-o no acumulador.

. CALL (Hex7)
Transforma valor lido da 2ª tecla em código de sete segmentos.

. LD (2707),A
Deposita no registro de código do 5º display, o valor referente à 2ª tecla.
Não esqueça que este valor se apresenta em código de 7 segmentos.

. LD HL,2706
Prepara HL, com endereço inicial da tabela de caracteres, a serem exibidos no display.

. LD (2703),HL
Carrega os registros de endereço de tabela 2703 e 2704 com o valor de "L" e "H" respectivamente, determinando endereço inicial da fonte de caracteres a serem mostrados no display.

. RET
Retorna à rotina principal.

Para finalizar a sub-rotina 2ª tecla de dados (2END), veja seus códigos objeto na figura 4.35.

```
1E77 -   3A 02 27 07 07 07 07 32 02 27 47 3A 05 27 80 32
1E87 -   02 27 3A 05 27 CD 20 1F 32 07 27 21 06 27 22 03
1E97 -   27 C9
```

Figura 4.35 - Códigos objeto da sub-rotina 2ª tecla de endereço.

J) SUB-ROTINA 3ª TECLA DE ENDEREÇO (3END)
Endereços ocupados: 1E57 a 1E69

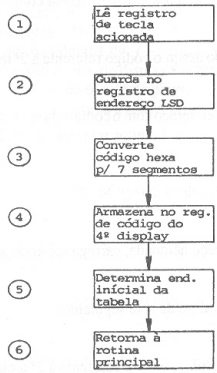


Figura 4.36 - Fluxograma da sub-rotina de 3ª tecla de endereço (3END)

- Fluxograma:
A rotina "3END" é similar à "1END". Sua finalidade básica, é ler o código correspondente à 3ª tecla de endereço, guardá-lo no registro de endereço LSD, convertê-lo para sete segmentos e armazená-lo no registro de código do 4º display.
Vejamos o fluxograma da figura 4.36.
Inicialmente no passo 1, o valor correspondente à 3ª tecla de endereço, é lido e carregado no acumulador.
No passo 2, o mesmo valor lido anteriormente no passo 1, é armazenado no registro de endereço LSD.
Note que o registro de endereço MSD, guarda valores referentes à 6ª e 5ª teclas de endereço, ao passo que o registro de endereço LSD, guarda valores referentes à 4ª e 3ª teclas de endereço.
No passo 3, o código referente à 3ª tecla de endereço é convertido para 7 segmentos e posteriormente ser utilizado no passo seguinte.
No passo 4, o código de 7 segmentos é guardado no registro referente ao 4º display.
No passo 5, o endereço inicial da tabela de caracteres é carregado no registro de endereço de tabela.
No 6º e último passo, o programa retorna à rotina principal.

- Programa
Através da figura 4.37, podemos observar a sub-rotina de 3ª tecla de endereço completo.

ENDEREÇO SIMBÓLICO	ENDER. FISÍCO	CÓDIGO OBJETO	ASSEMBLY	COMENTÁRIOS
	1E57	3A0527	LD A,(2705)	Lê código da tecla acionada
	1E5A	320127	LD (2701),A	Guarda no registro endereço LSD
	1E5D	CD201F	CALL (HEX7)	Transforma em código de 7 segmentos
	1E60	320827	LD (2708),A	Armazena no registro do 4º display
	1E63	210627	LD HL,2706	Determina endereço inicial da tabela
	1E66	220327	LD (2703),HL	Carrega reg. end. da tabela com 2706
	1E69	C9	RET	Retorna à rotina principal

Figura 4.37 - Sub-rotina 3ª Tecla de Endereço (3END)

Vejamos a seguir, suas instruções em sequência.

. LD A(2705)
Transfere para o acumulador, o código da 3ª tecla de endereço.

. LD (2701),A
Guarda código da 3ª tecla, no registro de endereço LSD.

. CALL (Hex7)
Converte código hexa, referente à 3ª tecla de endereço, em código de 7 segmentos.

. LD (2708),A
Armazena código de 7 segmentos, no registro de código do 4º display.

. LD HL,2706
Carrega HL, com o endereço inicial da tabela de caracteres a serem exibidos no display.

. LD (2703),HL
Carrega os registros de endereço de tabela 2703 e 2704, com os conteúdos de L e H.

. RET
Retorna à rotina principal.

Para finalizar a sub-rotina "3END", veja na figura 4.38 seus códigos objeto.

```
1E57-   3A 05 27 32 01 27 CD 20 1F 32 08 27 21 06 27 22
1E67-   03 27 C9
```

Figura 4.38 - Códigos objeto da sub-rotina de 3ª tecla de endereço (3END).

K) SUB-ROTINA 4ª TECLA DE ENDEREÇO (4END)
Endereços ocupados: 1E25 a 1E48
-Fluxograma:
Esta sub-rotina assemelha-se com a "2END", já descrita anteriormente.
Sua função principal é "montar" um código composto com os valores da 3ª e 4ª teclas.
Posteriormente, este resultado é gravado no registro de endereço LSD.
Note que esta sub-rotina também transforma o valor hexa da 4ª tecla, em código de 7 segmentos, armazenando-o no registro de código do 3º display.
Vejamos agora em sequência, o fluxograma da figura 4.39.
No passo 1, o valor da tecla anterior (3ª tecla) é lido no registro LSD.
No passo 2, o valor lido no passo anterior, é isolado apenas no dígito da direita, obtendo então somente o valor dos 4 bits da direita.
No passo 3, o código referente à 3ª tecla obtido nos passos 1 e 2, é deslocado 4 bits para esquerda, trazendo assim o dígito do lado direito para o lado esquerdo, bem como zerando o lado direito.
No passo 4, o valor obtido no passo anterior é armazenado no registro de endereço LSD.
No passo 5, o valor 4 é salvo no registro B.
No passo 6, o registro de tecla acionada é lido, carregando então, no acumulador, o valor da 4ª tecla de endereço.
No passo 7, a 3ª tecla é adicionada à 4ª tecla, gerando assim um código composto de 2 dígitos.
No passo 8, o valor obtido no passo anterior, é carregado no registro de endereço LSD.
No passo 9, o registro de tecla acionada é lido, obtendo o código da 4ª tecla de endereço.
No passo 10, o valor da 4ª tecla é convertido para sete segmentos.
No passo 11, o código de 7 segmentos é armazenado no registro do 3º display.

No passo 12, o início da tabela de caracteres é determinado, para gerar dados para o display.
No passo 13, o programa retorna à rotina principal.

Figura 4.39 - Fluxograma da sub-rotina de 4a. tecla de endereço (4END)

-Programa

Através da figura 4.40, pode ser notada a sub-rotina completa da 4ª tecla de endereço. Vejamos seu funcionamento em sequência:

ENDEREÇO SIMBÓLICO	ENDR. FÍSICO	CÓDIGO OBJETO	ASSEMBLY	COMENTÁRIO
	1E25	3A0127	LD A,(2701)	Lê registro de end. LSD
	1E28	E60F	AND 0F	Isola dígito da direita
	1E2A	07	RLC A	Desloca 1 bit para esquerda
	1E2B	07	RLC A	Desloca 1 bit para esquerda
	1E2C	07	RLC A	Desloca 1 bit para esquerda
	1E2D	07	RLC A	Desloca 1 bit para esquerda
	1E2E	320127	LD (2701),A	Armazena no reg. de end. LSD
	1E31	47	LD B,A	Salva em B
	1E32	3A527	LD A,(2705)	Lê registro de tecla acionada
	1E35	80	ADD A,B	Soma 3ª com 4ª tecla de endereço
	1E36	320127	LD (2701),A	Armazena no registro de end. LSD
	1E39	3A052	LD A,(2705)	Lê registro de tecla acionada
	1E3C	CD201F	CALL (HEX7)	Transforma em código de 7 segmentos
	1E3F	320927	LD (2709),A	Armazena no reg. de código do 3º display
	1E42	210627	LD HL, 2706	Determina end. inicial da tabela
	1E45	220327	LD (2703),HL	Carrega reg. end. de tabela com 2706
	1E48	C9	RET	Retorna à rotina principal

Figura 4.40 - Sub-rotina 4ª Tecla de Endereço - (4END)

. LD A, (2701)

Lê o código da 3ª tecla de endereços contido no registro de endereços LSD.

. AND 0F

Isola os 4 bits do lado direito do acumulador, filtrando assim, a qualquer valor indesejável nos 4 bits da esquerda, ou seja, zera o dígito mais significativo do acumulador.

. RCL A Deslocam o conteúdo do acumulador 1 bit para esquerda

. RLC A Idem

. RLC A Idem

. RLC A

Desloca o conteúdo do acumulador para esquerda pela 4ª vez, trazendo então o dígito da direita para o lado esquerdo.

. LD (2701), A

O valor obtido pela instrução anterior é depositado no registro de endereço LSD.

. LD B,A
O valor do acumulador, correspondente à 3ª tecla de endereço (deslocada 4 bits para esquerda), é salvo no registro B.

. LD A, (2705)
O acumulador, é carregado com o código referente ao registro de tecla acionada, que corresponde à 4ª tecla de endereço.

. ADD A,B
O código do registro B corresponde à 3ª tecla de endereço deslocada para esquerda, ao passo que, o acumulador contém o código referente à 4ª tecla de endereço.
Quando a instrução "ADD A,B" é executada, o acumulador soma o seu próprio conteúdo com o registro B, devolvendo o resultado da soma ao acumulador, obtendo o código final da 3ª e 4ª teclas.

. LD (2701),A
O resultado da edição anterior é guardado no registro de endereço MSD.

. LD A, (2705)
Carrega o acumulador com o código da 4ª tecla de endereço (última tecla acionada).

. CALL (HEX 7)
Transforma o código hexadecimal da 4ª tecla para 7 segmentos.

. LD (2709),A
Armazena no registro de código do 3º display.

. LD HL, 2706
Carrega "HL" com o endereço inicial da tabela de caracteres a serem exibidos no display.

. LD (2703),HL
Carrega os registros de endereço das tabelas 2703 e 2704, com os conteúdos de "L" e "H", respectivamente.

. RET
Retorna à rotina principal

Observe que após a execução da sub-rotina 4END, os registros de endereços LSD e MSD, estão armazenando os valores digitados, correspondentes às 4 teclas de endereço, desta forma, o microcomputador APL-80, está pronto para endereçar qualquer área da memória, com o objetivo de ler ou gravá-la.
Exceto as sub-rotinas auxiliares e tabelas, os demais programas a seguir, têm a finalidade de ler ou gravar ordenadamente na memória, os dados digitados no teclado.

Veja através da figura 4.41, como são gravados nos registros de endereço, os valores correspondentes às 4 teclas de endereço digitadas.

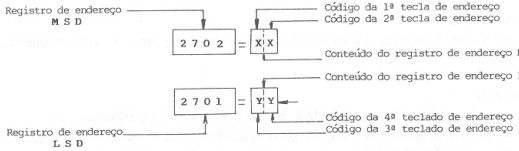


Figura 4.41 - Sequência de gravação na memória das teclas de endereço

Veja para finalizar através da figura 4.42, os códigos objeto referentes à sub-rotina de 4ª tecla de endereço.

```
1E25-   3A 01 27 E6 0F 07 07 07 07 32 01 27 47 3A 05 27
1E35-   80 32 01 27 3A 05 27 CD 20 1F 32 09 27 21 06 27
1E45-   22 03 27 C9
```

Figura 4.42- Códigos objeto da sub-rotina de 4ª tecla de endereço (4END)

L) SUB-ROTINA DE TRANSFORMAÇÃO HEXADECIMAL PARA 7 SEGMENTOS:
Endereços ocupados: 1F20 a 1F2A

- Fluxograma
Esta sub-rotina, tem a finalidade de transformar, qualquer valor de "0" a "F", contido no acumulador, em um código de 7 segmentos.
Observe que é utilizada, sempre que um valor hexadecimal lido da memória, tiver que ser apresentado no display.
Como o código hexadecimal, não é compatível diretamente com o código usado para utilizar o display, então é empregada a tabela de consulta da figura 4.43.

1F00 ..... 3F	1F08 ..... 7F
1F01 30	1F09 73
1F02 ..... 6D	1F0A 77
1F03 79	1F0B 5E
1F04 72	1F0C 0F
1F05 ..... 5B	1F0D 7C
1F06 ..... 5F	1F0E 4F
1F07 31	1F0E 47

Figura 4.43 - Tabela de transformação hexadecimal para 7 segmentos (TB-4).

Note na tabela-4 (fig.4.43), que são utilizados endereços de memória, cujos dois últimos dígitos (LSD) são os códigos hexadecimais "0" a "F" que devem ser convertidos.
Note também na tabela-4, que para cada valor hexadecimal, obtém-se no próprio conteúdo do endereço, o código de 7 segmentos. Por exemplo, no endereço, cujos 2 dígitos finais são "09", o código de 7 segmentos é 73. Portanto podemos dizer que para acender o display com o número "nove", é necessário enviar o código "73" aos circuitos do display.
O fluxograma é bastate simples e pode ser observado através da figura 4.44. Vejamos portanto, seu funcionamento em sequência.

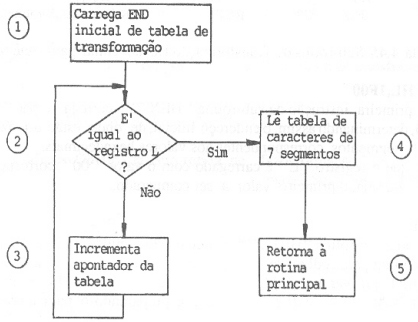


Figura 4.44 Fluxograma da Sub-rotina de Transformação Hexadecimal para 7 Segmentos (HEX 7)

No passo 1, o endereço inicial da tabela de transformação hexadecimal para 7 segmentos, é determinado para iniciar a transformação.
No passo 2, o registro L que contém os valores hexadecimal em sequência, é comparado com o código do acumulador, que deve conter o valor hexá a ser transformado.
No passo 3, o programa só entra se não houve igualdade na comparação do passo 2. Portanto, uma unidade é incrementada no registro "L", apontando novo código hexadecimal a ser pesquisado.
O programa só entra no passo 4 quando ouve igualdade na comparação do passo 2, ou seja, se o valor hexa do acumulador, era igual ao registro "L". Caso afirmativo, o conteúdo do endereço apontado na tabela é lido e depositado no acumulador.
No passo 5, o programa retorna à rotina que gerou esta sub-rotina (HEX 7).

- Programa

Para melhor entendimento da sub-rotina "HEX 7", vejamos em detalhes as instruções da figura 4.45.

ENDEREÇO SIMBÓLICO	ENDER. FÍSICO	CÓDIGO OBJETO	ASSEMBLY	COMENTÁRIO
1F20	21001F	LD HL,1F00		Carrega HL com endr. tab. de transformação
CICL	1F23	BD	CP L	Compara código hexa com reg. L
	1F24	2803	JR Z(SAID)	Se igual vai para (SAID)
	1F27	18FA	JR (CICL)	Recicla para obter nova comparação
SAID	1F29	7E	LD A,(HL)	Lê tabela
	1F2A	C9	RET	Retorna à sub-rotina original

Tabela 4.45 Sub-rotina de Transformação Hexadecimal para Sete Segmentos (HEX 7)

. LD HL,1F00

Esta primeira instrução da sub-rotina "HEX 7", carrega o par "HL", com o valor 1F00, determinando assim, o endereço inicial, de onde estão armazenados os códigos de 7 segmentos, correspondentes aos valores hexadecimais.

Note que o registro "L" é carregado com o valor "00", correspondendo ao HEXA "00", ou seja, o primeiro valor a ser comparado.

. CPL

Quando a sub-rotina "HEX 7" é executada, o acumulador contém o valor que se deseja transformar para código de 7 segmentos. Portanto, a instrução "CP L", compara se o registro L que varia de "00" a "0F". é igual ao acumulador. Se houver igualdade na comparação, então o flag zero é ligado, preparando-o para a próxima instrução.

. JR Z (SAID)

Se na instrução anterior houve igualdade, então o programa vai para o endereço simbólico "SAID" (saída), executando instruções que serão a memória da tabela.

Se não houve igualdade na comparação anterior, significa que o "flag zero" foi ligado, portanto o programa continua na próxima linha.

. INC L

Como não houve igualdade na instrução "CP L", então o programa atinge a 4ª linha. Neste ponto, o registro "L" é incrementado, apontando novo endereço de memória de tabela.

Apesar de só ser incrementado o registro "L", note que o endereço da tabela é formado pelo conteúdo do par "H" e "L".

. JR (CICL)

Através do salto incondicional "JR (CICL)", o programa retorna para o endereço simbólico "CICL" (ciclo), repetindo então nova comparação.

. LD A, (HL)
Quando o programa atinge a 6ª linha, significa que houve igualdade, entre a comparação do registro L e o código do acumulador. Portanto a instrução "LD A (HL)" lê a memória endereçada por "HL", depositando o valor lido no acumulador. Como "HL" aponta o endereço da tabela TB-4, então o valor resultante no acumulador é o código de 7 segmentos do número hexadecimal procurado.

. RET
A última instrução de retorno, faz o programa continuar onde foi interrompido para executar a sub-rotina HEX 7".
Resumindo,a sub-rotina "HEX 7" recebe um valor "hexadecimal de "0" a "F" no acumulador, transforma-o em código de 7 segmentos, e o devolve transformado no acumulador.

Na figura 4.46, você pode observar os códigos objeto da sub-rotina de transformação hexadecimal para 7 segmentos.

1F20- 21 00 1F BD 28 03 2C 18 FA 7E C9

Figura 4.46 Códigos Objeto da Sub-rotina de Transformação Hexadecimal para 7 Segmentos (HEX 7).

M) SUB-ROTINA TECLA "ENTER" (ENTR)
Endereços ocupados 1D35 a 1D62

- Fluxograma:
A sub-rotina "ENTR" é gerada pela sub-rotina "FUNC". Após o seu término, o programa retorna à rotina principal.
A sub-rotina "ENTR", entra em funcionamento toda vez que a tecla "ENTER" é acionada. Esta possui 2 funções básicas.
A primeira função da tecla "ENTER", é desligar o programa das sub-rotinas de endereçamento, ligando as sub-rotinas de "entrada" de dados.
A segunda função da tecla ou sub-rotina "ENTER", é ler os dados do endereço mostrados no display. Cada vez que a tecla ENTER for acionada, o endereço do display, será incrementado, e o respectivo conteúdo aparecerá no display.
Para entender melhor a utilização da tecla "ENTER", suponha que inicialmente é acionada a tecla "ADD", preparando o programa para receber 4 valores de endereço. Em seguida, suponha que são digitadas em sequência as teclas 2, F, 7 e 0. Neste momento, o display apresentará o valor 2F70, e mesmo que novos códigos sejam digitados nada acontecerá.
Nesta altura, o programa exige que seja acionada a tecla "ENTER".
Se acionarmos apenas 1 vez a tecla ENTER, então o display mostrará 2F70XX, onde XX será qualquer valor gravado no endereço 2F70.
Se continuarmos digitando valores hexadecimais, então, estes serão introduzidos aos pares na memória, a partir do endereço 2F70.

Note que a cada 2 teclas acionadas, um novo endereço aparece no display, a fim de receber um novo conteúdo. Por este processo torna-se possível "carregar" valores em toda memória. Após a digitação das teclas ADD, 2, F, 7 e 0, se acionarmos repetidamente a tecla "ENTR", então o display mostrará em sequência, o endereço e seu respectivo conteúdo.

Veja o exemplo a seguir:
1F70 - 34
1F71 - F3
1F72 - 98
1F73 - A4, ETC.

Esta última sequência analisada é a utilização da tecla "ENTR" como examinadora de memória.

Veja agora, o fluxograma da sub-rotina tecla ENTER na figura 4.47

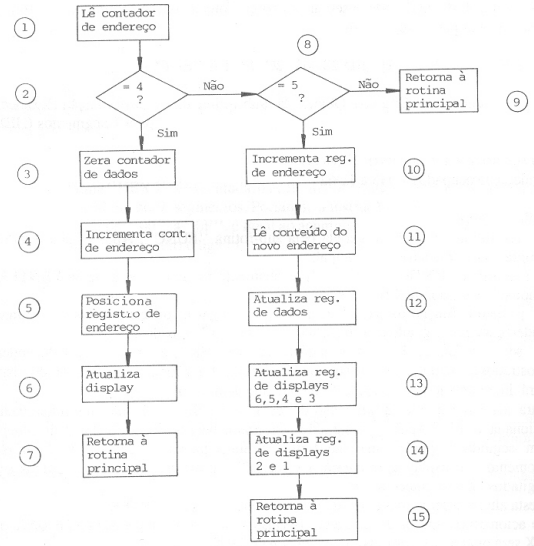


Figura 4.47 - Fluxograma da Sub-rotina Tecla ENTER (ENTR)

Note que está dividido em 2 partes a primeira delas é formada pelos passos 1 a 7. Esta parte da sub-rotina "ENTR", é utilizada para posicionar registros, e preparar o programa para receber dados, para serem gravados na memória em sequência.
A segunda parte é formada pelos passos 8 a 15, e tem a função de ler ou examinar um endereço de memória a cada vez que é acionada.
Antes de estudar o fluxograma, veja que são utilizados alguns endereços de memória RAM como registros e contadores.
O contador de dados, conta quando o dados é digitado para ser gravado na memória, é o digito MSD ou LSD. Você o entenderá melhor quando estudar as rotinas "1DAD" e "2DAD".
O contador de endereço, nas contagens 1 a 4 seleciona as sub-rotinas já estudadas "1END", "2END", "3END" e "4END".
Se o contador de endereço estiver na contagem 4, e a tecla ENTER for acionada, então a sub-rotina "ENTR", preparará o programa, para receber códigos do teclado, que serão utilizados como dados a serem gravados na memória.
Quando o contador de endereço atingir a contagem 5, é sinal que a tecla ENTER foi acionada pela 2ª vez consecutivamente. Portanto, é desencadeado o processo de leitura de endereço de memória.
Os registros de display 6, 5, 4, 3, 2 e 1, armazenam o código de 7 segmentos, correspondente ao número hexadecimal, que o display deve mostrar.
Vejamos agora o funcionamento da sub-rotina "ENTR" em sequência.
No passo 1, o contador de endereço é lido para determinar se a tecla ENTER será utilizada como examinadora de memória, ou como aviso de entrada de dados.
No passo 2, o contador de dados é comparado com 04. Caso afirmativo, significa que a tecla "ENTER" foi acionada a primeira vez após os 4 dígitos de endereço, portanto o programa continua no passo 3, preparando alguns registros, para executar as sub-rotinas "1DAD" e "2DAD", que carregarão na memória os dados digitados no teclado.
No passo 3, o contador de dados é zerado para ser utilizado pelas sub-rotinas "1DAD" e "2DAD".
No passo 4, o contador de endereço é incrementado para 05, preparando o programa para executar uma leitura de memória, caso a próxima tecla acionada seja "ENTR".
Caso contrário, a sub-rotina "ENDA" desvia o programa para executar as sub-rotinas "1DAD" e "2DAD".
No passo 5, o registro de endereço é lido, posicionando para mostrar no display, o endereço selecionado pelas 4 teclas iniciais de endereço, bem como o respectivo conteúdo.
No passo 6, todos os displays são atualizados e acesos
No passo 7, o programa volta para rotina principal
No passo 8, o contador de endereço é comparado com o valor 05. Caso afirmativo, o programa vai para o passo 10, caso contrário, vai para o passo 9, retornando à rotina principal.
Estando o programa no passo 10, significa que a tecla ENTER foi acionada pela 2ª vez consecutivamente, por isso, o registro de endereço é incrementado, tendo o objetivo de exibir no display o próximo endereço de memória a ser lido.

No passo 11, o conteúdo do endereço incrementado é lido para ser apresentado no display.
No passo 12, os dados referentes ao conteúdo da memória exibidos no display, são copiados no registro de dados, a fim de serem utilizados em outros programas.
No passo 13, os registros que contêm os códigos de 7 segmentos dos displays que mostram o endereço, são atualizados.
No passo 14, os registros que contêm os códigos de 7 segmentos dos displays, que mostram o conteúdo do endereço, são também atualizados.
No passo 15, o programa volta à rotina principal

- Programa:
Através da figura 4.48, pode ser observada a sub-rotina tecla ENTER (ENTR) completa. Vejamos portanto seu funcionamento em sequência de instrução.

ENDEREÇO SIMBÓLICO	ENDR. FÍSICO	CÓDIGO OBJETO	ASSEMBLY	COMENTÁRIOS
	1D35	3A0327	LD A,(270E)	Lê contador de endereço
	1D38	F304	CP 04	É igual a 4
	1D3A	2804	JR Z(1ENT)	Se igual vai para "1ENT"
	1D3C	FE05	CP 05	É igual a 5
	1D3E	2811	JR Z(2ENT)	Se igual vai para "2ENT"
1ENT	1D40	3E00	LD A,00	Zera acumulador
	1D42	320D27	LD (270D),A	Zera contador de dados
	1D45	3A0E27	LD A,(270E)	Lê contador de endereço
	1D48	3C	INC A	Incrementa acumulador
	1D49	320E27	LD (270E),A	Incrementa contador de endereço
	1D4C	2A0127	LD HL,(2701)	Posiciona registro de endereço
	1D4F	1804	JR (ATUA)	Vai para (ATUA) atualização de display
2ENT	1D51	2A0127	LD HL,(2701)	Lê registro de endereço
	1D54	23	INC HL	Inc. registro de endereço
ATUA	1D55	220127	LD (2701),HL	Lê registro de endereço
	1D58	7E	LD A,(HL)	Lê memória apontada pelo display
	1D59	320027	LD (2700),A	Atualiza registro de dados
	1D5C	CDB91D	CALL (A6-3)	Atualiza registro dos displays 6,5,4,3
	1D5F	CD701D	CALL (A2-1)	Atualiza registros dos displays 2 e 1
	1D62	C9	RET	Retorna à rotina principal

Figura 4.48 - Sub-rotina Tecla ENTER (ENTR)

. LD A, (270E)
O acumulador é carregado com o conteúdo do contador de endereço (270E)

. CP 04
O conteúdo do contador de endereço contido no acumulador é comparado com 04. Caso haja igualdade, o flag zero é ligado.

. JR Z (1ENT)
Se a comparação anterior foi igual a 04, então o programa "percebe" que a tecla "ENTER" foi acionada pela 1ª vez, sendo assim, o programa é desviado para a 1ª entrada (1ENT).
Caso a comparação não seja igual a 04, o programa continua na próxima linha.

. CP 05
O conteúdo do contador de endereço contido no acumulador é comparado com 05. Caso haja igualdade, o flag zero é ligado.

. JR Z (2ENT)
Se a comparação anterior foi igual a 05, então o programa "percebe" que a tecla "ENTER" foi acionada pela 2ª vez consecutivamente, sendo assim, o programa, é desviado para a 2ª entrada (2ENT).

. LD A,00
O conteúdo do acumulador é zerado, preparando-o para a próxima instrução.

. LD (270D),A
O conteúdo do acumulador (00) é transferido para o contador de dados, zerando-o

. LD A,(270E)
O conteúdo do contador de endereço é lido com a finalidade de ser incrementado.

. INC A
O acumulador é incrementado uma unidade.

. LD (270E), A
O acumulador incrementado é devolvido ao contador de endereço, incrementando o mesmo.

. LD HL, (2701)
Carrega o conteúdo do registro de endereço LSD (2701) em L. e o conteúdo do registro de endereço MSD (2702) em H, lendo então o registro de endereço.

JR (ATUA)
Este salto incondicional (JR), desvia o programa para o endereço simbólico (ATUA) atualização de display.
A finalidade deste trecho do programa é acender os displays de 1 a 6 com seus respectivos valores. Veja mais adiante da 15ª linha até o fim da sub-rotina "ENTR".

. LD HL, (2701)
Carrega o conteúdo do registro de endereço LSD (2701) em L, e o conteúdo do registro de endereço MSD (2702) em H, lendo assim o registro de endereço.

. INC HL
Como o par HL contém o valor do registro de endereço, então a instrução INC HL, incrementa o conteúdo do registro de endereço apontando novo valor para ser visualizado.

. LD (2701), HL
Trasfere o conteúdo de HL (incrementado) para o registro de endereço.

. LD A, (HL)
Carrega o acumulador com o conteúdo do endereço apontado por HL.
Como HL contém o valor do endereço mostrado no display, então o acumulador, passa a ter o conteúdo do endereço apontado no display.

. LD (2700),A
O endereço da memória RAM "2700" é utilizado como registro de dados, portanto o conteúdo do endereço de memória, apontado no display, deve ser sempre carregado no registro de dados. Isto é realizado pela instrução "LD (2700),A".

. CALL (A6-3)
A sub-rotina A6-3, atualiza os registros de códigos de endereço de display, com seus novos códigos de 7 segmentos. Você entenderá melhor este mecanismo quando estudarmos esta sub-rotina. Lembre-se apenas que ela atualiza os displays que mostram o endereço, isto é, 6, 5, 4 e 3.

. CALL (A2-1)
Esta sub-rotina quando executada, tem a mesma finalidade da descrita anteriormente, difere-se no entanto, porque atualiza os displays de dados 2 e 1.

. RET
Como esta sub-rotina é parte do programa principal, então a última instrução de retorno "RET", retorna à rotina principal.

Na figura 4.49, você pode observar os códigos objeto da sub-rotina "ENTR".

```
1D35-   3A 0E 27 FE 04 28 04 FE 05 28 11 3E 00 32 0D 27
1D45-   3A 0E 27 3C 32 0E 27 2A 01 27 18 04 2A 01 27 23
1D55-   22 01 27 7E 32 00 27 CD B9 1D CD 70 1D C9
```

Figura 4.49 - Códigos da Sub-rotina Tecla ENTER (ENTR)

N) SUB-ROTINA DE ATUALIZAÇÃO DOS REGISTROS DOS DISPLAYS 6 A 3 (A6-3)
Endereços ocupados: 1DB9 a 1DED

- Fluxograma:

A função básica da sub-rotina de atualização dos registros dos displays de 6 a 3 (A6-3), é transformar o conteúdo dos registros de endereço MSD e LSD, para posteriormente, depositar o código transformado em sete segmentos, nos registros de código do display.
Veja através da figura 4.50, como os valores são transferidos dentro da memória, quando a sub-rotina "A6-3" é executada.

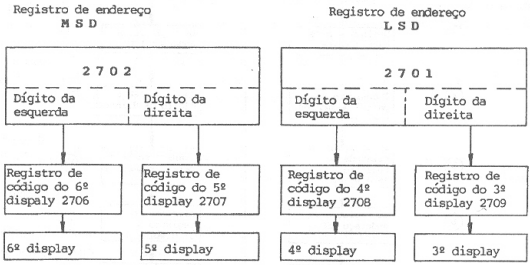


Figura 4.50 - Fluxo da Transferência de Dados na Memória desencadeado pela sub-rotina (A6-3) (5,6x11,2)

A sub-rotina de atualização dos registros dos displays de 6 a 3, apesar de ser um pouco extensa, é relativamente simples.
Vejamos o seu funcionamento através do seu fluxograma que aparece na figura 4.51.
No passo 1, o registro de endereço MSD é lido, obtendo como resultado um "byte" (conjunto de 8 bits), contendo 2 dígitos, que chamaremos de dígito da esquerda e dígito da direita.
No passo 2, o dígito da esquerda é isolado, com este processo, o valor do dígito da direita é zerado, portanto obtém-se um resultado do tipo "X0", onde "x" representa o dígito da esquerda preservado e o zero representa o dígito da direita zerado.
Apenas como exemplo, suponha que o valor lido no passo 1 seja "4A". Após ser executado o passo 2 o valor pasa a ser 40, ou seja, preservamos o dígito da esquerda e zeramos o da direita.
No passo 3, o valor obtido no passo anterior, é deslocado 4 bits para direita. Isto permite que, nosso valor "40" obtido no passo anterior, seja transformado em 04.
Veja agora, que o valor 04 representa o número 4, referente ao dígito da direita contido no registro de endereço MSD. Lembre-se também que, este suposto número 4, corresponde à 1ª tecla de endereço acionada, e por esta razão, é que este valor é transformado em código de 7 segmentos no passo 4, e carregado no registro de código do 6º display no passo 5.

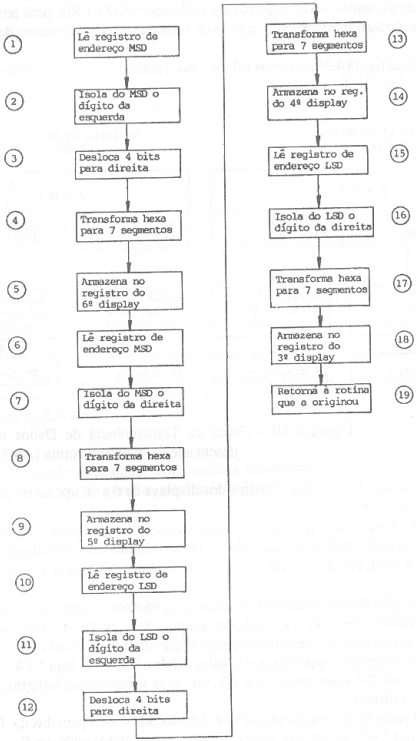


Figura 4.51 - Fluxograma da Sub-rotina de Atualização dos Registros dos Displays 6 a 3 (A6-3)

No passo 6,o registro de endereço MSD é lido novamente com o objetivo de obter o dígito da direita.
No passo 7, o dígito da direita é isolado para ser utilizado no passo seguinte.
No passo 8, o código obtido no passo 7, é transformado para 7 segmentos diretamente, não sendo necessário deslocá-lo para direita, pois já está do lado direito.
No passo 9, este código de 7 segmentos, obtido no passo anterior, é carregado no registro de código do 5º display.
Os passos 10, 11, 12, 13 e 14, transformam para 7 segmentos, o dígito, da esquerda do registro de endereço LSD, depositando o resultado obtido no registro de código do 4º display.
Os passos 15, 16, 17 e 18, transformam para sete segmentos, o dígito da direita do registro de endereço LSD, depositando o resultado obtdido no registro de código do 3º display.No passo 19, o programa retorna para rotina que gerou a sub-rotina A6-3.

- Programa:
Através da figura 4.52, você pode notar com detalhes a sub-rotina de atualização dos registros dos displays 6 a 3 (A6-3), portanto vejamos seu funcionamento em sequência de instruções.

ENDEREÇO SIMBÓLICO	ENDR. FÍSICO	CÓDIGO OBJETO	ASSEMBLY	COMENTÁRIOS
	1DB9	3A0227	LD A,(2702)	Lê registro de endereço MSD
	1DBC	E6 F0	AND F0	Isola do MSD o dígito da esquerda
	1DBE	1F	RRA	Desloca 1 bit para direita
	1DBF	1F	RRA	Idem
	1DC0	1F	RRA	Idem
	1DC1	1F	RRA	Idem
	1DC2	CD201F	CALL (HEX 7)	Transforma hexa para 7 segmentos
	1DC5	320627	LD (2706),A	Armazena no registro do 6º display
	1DC8	3A0227	LD A, (2702)	Lê registro de endereço MSD
	1DCB	E6 0F	AND 0F	Isola do MSD o dígito da direita
	1DCD	CD201F	CALL (HEX 7)	Transforma hexa para 7 segmentos
	1DD0	320727	LD (2707),A	Armazena no registro do 5º display
	1DD3	3A0127	LD A, (2701)	Lê registro de endereço LSD
	1DD6	E6 F0	AND F0	Isola do LSD o dígito da esquerda
	1DD8	1F	RRA	Desloca 1 bit para direita
	1DD9	1F	RRA	Idem
	1DDA	1F	RRA	Idem
	1DDB	1F	RRA	Idem
	1DDC	CD201F	CALL (HEX 7)	Transforma hexa para 7 segmentos
	1DDF	320827	LD (2708),A	Armazena no registro do 4º display
	1DE2	3A0127	LD A, (2701)	Lê registro de endereço LSD
	1DE5	E6 0F	AND 0F	Isola do LSD o dígito da direita
	1DE7	CD201F	CALL (HEX 7)	Transforma hexa para 7 segmentos
	1DEA	320927	LD (2709),A	Armazena no registro do 3º display
	1DED	C9	RET	Retorna à rotina que gerou a sub. "A6-3"

Figura 4.52 Sub-rotina de Atualização dos Registros dos Displays 6 a 3 (A6-3)

. LD A(2702)
Esta instrução carrega o acumulador com o conteúdo do registro de endereço MSD (2702)

. AND F0
Isola o dígito da esquerda do registro de endereço MSD.

. RRA - Desloca o conteúdo do acumulador 1 bit para a direita.
. RRA - Idem
. RRA - Idem
. RRA Idem, obtendo no lado direito do acumulador o dígito que estava à esquerda.

. CALL HEX 7
Transforma o código hexadecimal do acumulador, em 7 segmentos. Note que o resultado em 7 segmentos, fica no acumulador.

. LD (2706),A
Armazena o código de 7 segmentos no registro de códigos do 6º display. Preparando o display para apresentar o dígito da esquerda de endereço MSD.
Lembre-se que o display só acenderá com este valor, durante a sub-rotina de display.

. LD A, (2702)
Lê novamente o registro de endereço MSD e carrega-o no acumulador.

. AND 0F
Isola o dígito da direita do registro de endereço MSD.

. CALL (HEX 7)
Transforma o conteúdo do acumulador em código de 7 segmentos

. LD (2707),A
Armazena o código de 7 segmentos, obtido pela instrução anterior, no registro de código de 5º display, preparando o display para apresentar o 2º dígito mais significativo de endereço.

. LD A, (2701)
Carrega o acumulador com o conteúdo do registro de endereço LSD (2701)

. AND F0
Isola no acumulador, o dígito da direita

. RRA - Desloca o conteúdo do acumulador 1 bit para direita
. RRA - Idem
. RRA - Idem.
.RRA - Idem, obtendo no lado direito do acumulador o dígito que estava à esquerda.

. CALL (HEX 7)
Transforma o código hexadecimal do acumulador, em 7 segmentos.

. LD (2708),A
Armazena o conteúdo do acumulador no registro de código do 4º display (2708).

. LD A, (2701)
Lê novamente o registro de endereço LSD, carregando o valor obtido no acumulador.

. AND OF
Isola o dígito da direita do registro de endereço LSD.

. CALL (HEXA 7)
Transforma o código hexadecimal do acumulador, em 7 segmentos.

. LD (2709),A
Armazena o conteúdo do acumulador no registro de código do 3º display.

. RET
Retorna ao programa que gerou esta sub-rotina.

Veja na figura 4.53, os códigos objeto da Sub-rotina "A6-3".

```
1DB9-   3A 02 27 E6 F0 1F 1F 1F 1F CD 20 1F 32 06 27 3A
1DC9-   02 27 E6 0F CD 20 1F 32 07 27 3A 01 27 E6 F0 1F
1DD9-   1F 1F 1F CD 20 1F 32 08 27 3A 01 27 E6 0F CD 20
1DE9-   1F 32 09 27 C9
```

Figura 4.53 - Códigos Objeto da Sub-rotina de Atualização dos Registros dos Displays 6 a 3 (A6-3)

o) SUB-ROTINA DE ATUALIZAÇÃO DOS REGISTROS DOS DISPLAYS 2 E 1.
Endereços ocupados: 1D70 a 1D8A.

- Fluxograma
Esta rotina é semelhante à sub-rotina "A6-3", exceto que ao invés de atualizar os displays 6, 5, 4 e 3, atualiza os displays 2 e 1, com o código de 7 segmentos, referente ao valor hexadecimal do registro de dados (2700).

- Fluxograma:
Na figura 4.54, voce pode observar o fluxograma geral da sub-rotina de atualização dos registros dos displays 2 e 1 (A2-1).

Inicialmente no passo 1, o registro de dados é lido, obtendo assim os 2 dígitos que serão convertidos em códigos de 7 segmentos.

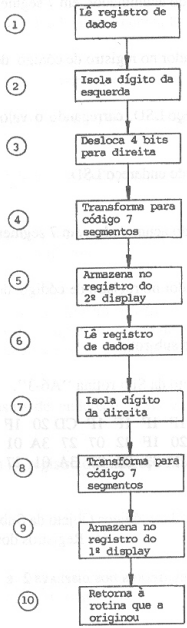


Figura 4.54 - Fluxograma da Sub-rotina de Atualização dos Registros dos Displays 2 e 1 (A2-1)

No passo 2, o dígito da esquerda é isolado do acumulador, enquanto que o dígito da direita é zerado.
no passo 3, o dígito da esquerda é transferido com seus 4 bits para direita.
No passo 4, o código obtido no passo anterior é convertido para 7 segmentos.
No passo 5, o valor de 7 segmentos é introduzido no registro do 2º display.
No passo 6, o registro de dados é lido novamente a fim de aproveitar o dígito da direita.

No passo 7, o conteúdo do registro de dados é isolado no seu dígito da direita e zerado no seu dígito da esquerda.
No passo 8, o valor obtido no passo anterior é convertido em código de 7 segmentos.
No passo 9, o código obtido no passo 8 é armazenado no registro de código do 1º display.
No 10º e último passo, o programa retorna à rotina que o originou.

- Programa:

ENDEREÇO SIMBÓLICO	ENDR. FÍSICO	CÓDIGO OBJETO	ASSEMBLY	COMENTÁRIOS
	1D70	3A0027	LD A,(2700)	Lê registro de dados
	1D73	E6 F0	AND F0	Isola o dígito da esquerda
	1D75	1F	RRA	Desloca bit para direita
	1D76	1F	RRA	Desloca bit para direita
	1D77	1F	RRA	Desloca bit para direita
	1D78	1F	RRA	Desloca bit para direita
	1D79	CD201F	CALL (HEX 7)	Transforma código hexa para 7 segmentos
	1D7C	320A27	LD (270A),A	Armazena no reg. do 2º display
	1D7F	3A0027	LD A,(2700)	Lê registro de dados
	1D82	E6 0F	AND 0F	Isola o dígito da direita
	1D84	CD201F	CALL (HEX 7)	Transforma código hexa p/ 7 segmentos
	1D87	320B27	LD (270B),A	Armazena no reg. do 1º display
	1D8A	C9	RET	Lê registro de dados

Figura 4.55 Sub-rotina de Atualização dos Registros dos Displays (A2-1).

. LD A,(2700)
Transfere para o acumulador o conteúdo do registro de dados.

. AND F0
Isola o dígito da esquerda do acumulador, além de zerar o dígito da direita.

. RRA Desloca o conteúdo do acumulador 1 bit para direita
. RRA Idem
. RRA Idem
. RRA Idem, obtendo no lado direito do acumulador, o dígito presente no lado esquerdo.

. CALL (HEX 7)
Transforma o código hexadecimal obtido no acumulador, para código de 7 segmentos.

. LD (270A),A
Carrega o registro de código do 2º display com o conteúdo do acumulador

. LD A,(2700)
Lê novamente registro de dados, transferindo assim o seu conteúdo para o acumulador.

. AND 0F
Isola o dígito da direita do acumulador, além de zerar o dígito da esquerda.

. CALL (HEX 7)
Transforma o código hexadecimal; obtido no acumulador, para código de 7 segmentos

. LD (270B),A
Carrega o registro de código do 2º display com o conteúdo do acumulador.

. RET
Retorna à rotina que originou esta sub-rotina (A2-1).

Para finalizar esta sub-rotina, atente para os códigos objeto que aparecem na figura 4.56.

```
1D70-   3A 00 27 E6 F0 1F 1F 1F 1F CD 20 1F 32 0A 27 3A
1D80-   00 27 E6 0F CD 20 1F 32 0B 27 C9
```

Figura 4.56 - Códigos Objeto da Sub-rotina de Atualização dos Registros dos Displays 2 e 1 (A2-1)

P) SUB-ROTINA 1ª TECLA DE DADOS (1DAD)
Endereços Ocupados: 1E00 a 1E19

- Fluxograma
A sub-rotina 1ª tecla de dados (1DAD) tem 3 finalidades.

I- Quando terminamos de digitar as 4 teclas de endereço, o display, bem como os registros de endereço, apontam um determinado endereço de memória.
Suponha agora que o valor do endereço de memória seja "2015". neste instante, o conteúdo dos registros de endereço "MSD" e "LSD" são respectivamente "20" e "15". Também o display exibirá o valor 2015.Tudo isto significa que o microcomputador APL-80 está orientado para "gravar" um dado (par de dígitos) no endereço 2015.
Ao acionarmos a "1ª tecla de dados" a sub-rotina 1DAD entra em execução, depositando o valor da tecla acionada no conteúdo do endereço 2015.
Lembre-se que o conteúdo de um endereço qualquer de memória, é formado por um byte de 8 bits, o que corresponde a 2 dígitos hexadecimais.
Portanto, a sub-rotina "1ª tecla de dados", carrega o dígito mais significativo (lado esquerdo), do dado referente ao conteúdo do endereço de memória visualizado no display.
Mais adiante, você perceberá que, o segundo dado referente ao dígito menos significativo (lado direito) do lado referente ao conteúdo do endereço de memória apresentado no display, é carregado com o auxílio da sub-rotina 2ª tecla de endereço 2DAD.Convém observar que, cada vez que uma tecla de dados é acionada, as sub-rotinas 1DAD e 2DAD entram em funcionamento alternadamente, carregando um byte na memória.

Tecla Acionada	Endereço Apresentado no Display	Conteúdo do Endereço de Memória Gravado	Sub-rotina Envolvida
2	2	-	1 END
0	20	-	2 END
1	201	-	3 END
5	2015	-	4 END
ENTER	2015	-	ENTR
3	2015.3	30	1 DAD
A	2015.3A	3A	2 DAD
4	2016.4	40	1 DAD
9	2016.49	49	2 DAD
1	2017.1	10	1 DAD
0	2017.1	01	2 DAD
8	2018.8	80	1 DAD
F	2018.8F	8F	2 DAD

Figura 4.57 - Gravação Sequência de Dados na Memória a partir do Endereço 2015

Ao término da sub-rotina "2END", o endereço de memória apresentado no display, é incrementado, dando início a uma nova sequência de depósito de memória.
Para melhor entendimento, suponha que desejamos "gravar" os valores 3A, 49, 10 e 8F, a partir do endereço 2015 da memória.
Veja como isto ocorre na sequência da Figura 4.58, bem como qual sub-rotina é envolvida para cada tecla acionada.

II. Além de gravar valores na memória, a sub-rotina "1DAD", também atualiza o conteúdo do registro de dados, que contém sempre o valor apresentado nos 2 displays de dados (2 últimos do lado direito).

III. Finalmente, uma das últimas finalidades da sub-rotina "1DAD" é atualizar o conteúdo do registro de código do display 2, com seu respectivo valor de 7 segmentos.
O fluxograma da sub-rotina 1ª tecla de dados pode ser observada através da figura 4.58.
Portanto vejamos seu funcionamento em sequência.
No passo 1, a tecla digitada é lida no registro de tecla acionada, obtendo assim o código da 1ª tecla de dados.
No passo 2, o dígito do lado direito referente ao código da tecla acionada, é deslocado para o lado esquerdo.
No passo 3, o valor obtido anteriormente através do passo 2, é armazenado no registro de dados.
No passo 4, o código da tecla acionada é lido novamente com a finalidade de ser transformado no passo seguinte.
No passo 5, o código do acumulador referente à tecla acionada, é transformado para 7 segmentos

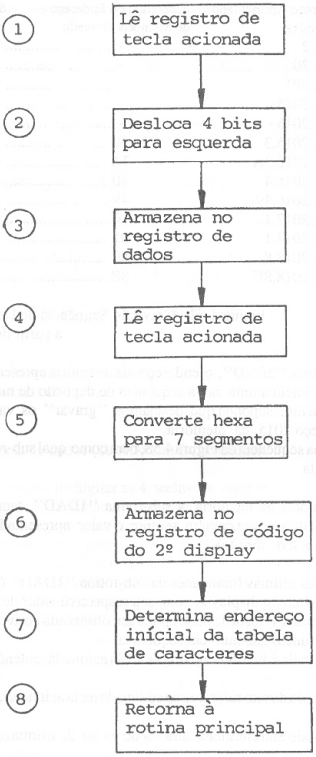


Figura 4.58 - Fluxograma de Sub-rotina 1a. Tecla de Dados (1DAD)

No passo 6, o valor obtido no passo anterior, é armazenado no registro de códigos do display 2.
No passo 7, o registro de endereço da tabela de caracteres é carregado com o endereço inicial dos dados a serem apresentados no display.
No passo 8, o programa retorna à rotina principal

- Programa
Através da figura 4.59, você pode observar com detalhes a sub-rotina de dados. Veja a seguir, seu funcionamento em sequência.

ENDEREÇO SIMBÓLICO	ENDR. FÍSICO	CÓDIGO OBJETO	ASSEMBLY	COMENTÁRIOS
	1E00	3A0527	LD A,(2705)	Lê código da tecla acionada
	1E03	07	RLCA	Desloca 1 bit para a esquerda
	1E04	07	RLCA	Desloca 1 bit para a esquerda
	1E05	07	RLCA	Desloca 1 bit para a esquerda
	1E06	07	RLCA	Desloca 1 bit para a esquerda
	1E07	320027	LD (2700),A	Aarmazena no registro de dados
	1E0A	3A0527	LD A,(2705)	Lê código da tecla acionada
	1E0D	CD201F	CALL (HEX7)	Converte código Hexa para 7 segmentos
	1E10	320A27	LD (270A),)	Armazena no registro do 2º. display
	1E13	210627	LD HL,2706	Determina endereço inicial da tabela
	1E16	220327	LD (2703),HL	Carrega no reg. de endr. de tabela
	1E19	C9	RET	Retorna à rotina principal

Figura 4.59 Sub-rotina 1ª Tecla de Dados (1DAD)

. LD A, (2705)
O acumulador é carregado com o conteúdo do registro de tecla acionada (2705), obtendo então o código da 1ª tecla de dados.

. RLCA
Desloca o conteúdo do acumulador 1 bit para esquerda, zerando o espaço ocupado pelo bit anterior.

. RLCA Idem
. RLCA Idem
. RLCA Idem,
trazendo finalmente o dígito que estava do lado direito para o lado esquerdo do acumulador.

. LD (2700),A
Carrega o valor obtido através da última instrução, no registro de dados (2700)

. LD A, (2705)
Lê novamente o código da tecla acionada, colocando-o no acumulador

. CALL (HEX 7)
Transforma o valor do acumulador, num código de 7 segmentos.

. LD (270A),A
Carrega o registro de código do 2º display com o código de 7 segmentos obtido na instrução anterior.

. LD HL, 2706
Carrega o registro L com o conteúdo do endereço 2706, e carrega o registro H com o conteúdo do endereço 2707.

. LD (2703), HL
Carrega o registro de endereço de tabela 2703 e 2704, com os conteúdos de "L" e "H" respectivamente, determinando assim o endereço inicial da tabela dos caracteres a serem apresentados no display.

. RET
Retorna à rotina principal

Para finalizar a sub-rotina "1DAD", veja seus códigos objeto na figura 4.60.

```
1E00-   3A 05 27 07 07 07 07 32 00 27 3A 05 27 CD 20 1F
1E10-   32 0A 27 21 06 27 22 03 27 C9
```

Figura 4.60 Códigos Objeto da Sub-rotina 1ª Tecla de Dados - (1DAD)

Q) SUB-ROTINA 2ª TECLA DE DADOS (2 DAD)
Endereços ocupados: 1D90 a 1DB7

- Fluxograma:
A finalidade de sub-rotina 2ª tecla de dados, é gravar o valor do 2º dígito (LSB) na memória, cujo endereço está presente no display.
A função da sub-rotina 2DAD é semelhante à 1DAD, exceto que trata do 2º dígito digitado, ao invés do 1º.
Vejamos para melhores esclarecimentos, o fluxograma da figura 4.61.
No passo 1, a 2ª tecla de dados digitada é lida e armazenada no acumulador.
No passo 2, o valor do acumulador é salvo no registo B.
No passo 3, o código da primeira tecla de dados, corresponde ao dígito do lado esquerdo, é somada ao dígito do lado direito, correspondente à 2ª tecla acionada, desta forma, obtém-se código composto de 2 dígitos.
No passo 4, o valor obtido no passo anterior, é depositado no registro de dados.
No passo 5, o código composto obtido no passo 3, é armazenado na memória, cujo endereço está presente no display, e contido no registro de endereço.
Neste passo do programa, os códigos digitados pelas 2 teclas de dados são armazenadas num endereço da memòria.

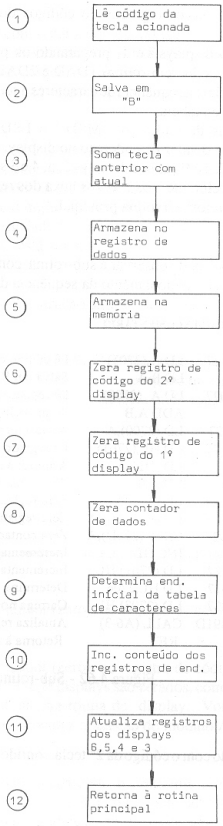


Figura 4.61 - Fluxograma da Sub-rotina 2a. Tecla de Dados (2DAD)

No passo 6, 7 e 8, são zeradas os registros de códigos dos displays "2" e "1", bem como o contador de dados.
Esta providência apaga os displays 2 e 1, preparando-os para receber os códigos das 2 próximas teclas referentes às sub-rotinas 1DAD e 2DAD.
No passo 9, o endereço inicial da sequência de caracteres a serem apresentados no display, é determinado no registro de tabela.
No passo 10, os registros de endereço "MSD" e LSD", são incrementados, com o obetivo de apresentar novo valor de endereço no display.
No passo 11, os registros de códigos dos displays 6, 5, 4 e 3 são atualizados com o novo código de 7 segmentos referente aos códigos hexa dos registros de endereço.
No passo 12, o programa retorna à rotina principal.

- Programa:
Através da figura 4.62, você pode observar a sub-rotina completa da 2ª tecla de dados 2 DAD. Vejamos portanto seu funcionamento da sequência de suas instruções.

ENDEREÇO ENDR. SIMBÓLICO	CÓDIGO FÍSICO OBJETO	ASSEMBLY	COMENTÁRIO
1D90	3A0527	LD A,(2705)	Lê código da Tecla
1D93	47	LD B,A	Salva em B
1D94	3A0027	LD A, (2700)	Lê registro de dados
1D97	80	ADD A,B	Soma tecla anterior com atual
1D98	320027	LD (2700),A	Armazena no registro de dados
1D9B	2A0127	LD HL,(2701)	Lê registro de endereço
1D9E	77	LD (HL),A	Armazena no ender. apontado p/ reg. end.
1D9F	3E00	LD A,00	Zera acumulador
1DA1	320A27	LD (270A),A	Zera registro de código do 2º dipslay
1DA4	320B27	LD4P (270B),A	Zera registro de código do 1ª display
1DA7	320D27	LD4P (270D),A	Zera contador de dados
1DAA	23	INC HL	Incrementa HL
1DAB	220127	LD (2701),HL	Incrementa reg. de endereços
1DAE	210627	LD HL,2706	Determina end. inicial da tabela
1DB1	220327	LD (2703),HL	Carrega no reg. de endereço de tabela
1DB4	CDB91D	CALL (A6-3)	Atualiza reg. dos displays 6, 5, 4 e 3
1DB7	C9	RET	Retorna à rotina principal

Figura 4.62 - Sub-rotina 2ª Tecla de Dados (2DAD)

. LD A,(2705)
O acumulador é carregado com o código da 2º tecla contido no registro de tecla acionada (2705).

. LD B,A
O código da tecla acionada é preservado no registro B, com a finalidade de ser utilizado na 4ª instrução do programa.

. LD A, (2700)
O acumulador é carregado com o conteúdo do registro de dados (2700).
Note que o registro de dados foi carregado anteriormente com o código da 1ª tecla de dados. Portanto o dígito do lado esquerdo contém o código da 1ª tecla de dados, ao passo que o dígito do lado direito está zerado, aguardando o código da 2ª tecla de dados.

. ADD A,B
O código da primeira tecla de dados guardado em B pela 2ª instrução do programa, é adicionado ao acumulador, que contém o código da 2ª tecla de dados.
Note que o registro A, contém o dígito do lado direito equivalente ao código da 2ª tecla de dados, enquanto que o dígito do lado esquerdo esta zerado.
O registro B, contém o dígito do lado esquerdo equivalente à 1ª tecla de dados, enquanto que o dígito do lado direito está zerado.

. LD (2700),A
Armazena o código obtido na instrução anterior no registro de dados.

. LD HL,(2701)
O conteúdo do registro de endereço "LSD" (2701), é carregado no registro L, enquanto que, o conteúdo do registro "MSD" (2702), é carregado no registro H, preparando desta forma, o par HL, com o endereço de memória onde receberá o código composto, digitado pela 1ª e 2ª teclas de dados.

. LD (HL),A
O código composto referente à combinação da 1ª e 2ª teclas de dados é carregado no endereço de memória deteminado por HL.

. LD A,00
O acumulador é zerado, preparando-se para a próxima instrução

. LD (270A),A
Carrega o conteúdo do acumulador (zero), no registro de código do 2º display (270A).

. LD (270B),A
Carrega o conteúdo do acumulador (zero), no registro de código do 1º display (270B)
Lembre-se que os registros do 1º e 2º displays são zerados, com a finalidade de mentê-los apagados durante a execução da sub-rotina de display. Você entenderá melhor este detalhe quando estudar mais adiante a operação do módulo APL-80.

. LD (270D),A
O contador de dados é zerado, preparando o programa monitor para receber nova sequência de 1ª e 2ª teclas de dados, com o objetivo de gravar tais valores no próximo endereço de memória apontado, tanto pelos registros de endereço, como pelos displays.

. INC HL
O conteúdo do par de registros HL é incrementado, preparando assim para adicionar uma unidade aos registros de endereços na próxima instrução.

. LD (2701), HL
O conteúdo de "L" é carregado no endereço 2701, ao passo que o conteúdo de "H" é carregado no endereço 270 2.
Esta instrução incrementa os registros de endereço.

. LD HL, 2706
Carrega HL com o valor 2706, determinando assim o início da tabela de dados a serem monstrados no display.

. LD (2703), HL
Carrega o conteúdo de L no endereço 2703, e o conteúdo de H em 2704, carregando assim o registro de endereço de tabela com 2706.

. CALL (A6-3)
Prepara registros de códigos dos displays 6 a 3, com seus novos valores para serem apresentados no display.

. RET
Retorna à rotina principal

Para finalizar, observe na figura 4.63, os códigos objeto referentes à sub-rotina 2ª tecla de dados (2DAD)

```
1D90-   3A 05 27 47 3A 00 27 80 32 00 27 2A 01 27 77 3E
1DA0-   00 32 0A 27 32 0B 27 32 0D 27 23 22 01 27 21 06
1DB0-   27 22 03 27 CD B9 1D C9
```

Figura 4.63 Códigos Objeto da Sub-rotina 2ª Tecla de Dados (2DAD)

R) SUB-ROTINA TECLA DE DECREMANTAÇÃO (DEC)
Endereços ocupados: 1D10 a 1D32

- Fluxograma:
A sub-rotina "DEC" é utilizada para corrigir os dados de um endereço "anterior" de memória, ou mesmo para ler valores de memórias, a partir de um endereço maior, em direção a um menor, ou seja para "decrementar", e ler a memória.

Para melhor entendê-la, vejamos a sequência do fluxograma apresentado na figura 4.64.

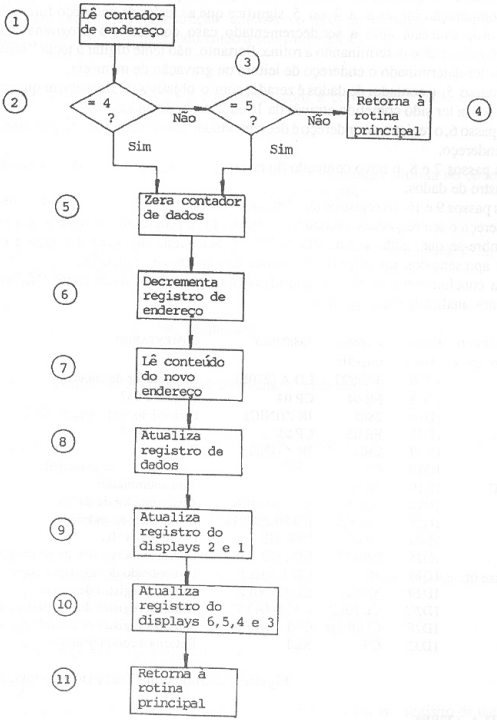


Figura 4.64 - Fluxograma da Sub-rotina Tecla de Decrementação (DEC)

No passo 1 do fluxograma, o contador de endereço é lido a fim de ser comparado posteriormente. Se o seu valor for menor que 4, significa que as 4 teclas de endereço ainda não foram digitadas.

No passo 2 e 3, o valor do contador de endereço é comparado com 04 e 05. Se o resultado da comparação for igual a 4 ou 5, significa que as teclas de endereço foram digitadas e o programa está apto a ser decrementado, caso contrário, o programa é abortado e vai para o passo 4, terminando a rotina. Portanto, não tente digitar a tecla "DEC", sem antes ter determinado o endereço de leitura ou gravação de memória.
No passo 5, o contador de dados é zerado, com o objetivo de sincronizar qualquer erro, que possa ter sido cometido através da 1ª ou 2ª tecla de dados.
No passo 6, o registro de endereço é decrementado, apresentando no display o novo valor de endereço.
Nos passos 7 e 8, o novo conteúdo do endereço decrementado é lido e armazenado no registro de dados.
Nos passos 9 e 10, os registros de códigos de todos displays são atualizados com o novo endereço e seu respectivo conteúdo.No passo 11, o programa retorna à rotina principal.
Lembre-se que, cada vez a tecla "DEC" é acionada, um novo endereço e conteúdo são apresentados no display, no modo decrescente e sequencial.
Para concluir o entendimento, veja através da figura 4.65, a sub-rotina DEC completa. Vamos analisá-la em sequência.

ENDEREÇO SIMBÓLICO	ENDER. FÍSICO	CÓDIGO OBJETO	ASSEMBLY	COMENTÁRIOS
	1D10	3A0E27	LD A (270E)	Lê contador de endereço
	1D13	FE 04	CP 04	É igual a 04?
	1D15	2805	JR Z(INIC)	Para início do programa (INIC)
	1D17	FE 05	CP 05	É igual a 05?
	1D19	2801	JR Z (INIC)	Para início do programa (INIC)
	1D1B	C9	RET	Retorna à rotina principal
INIC	1D1C	3E00	LD A,00	Zera acumulador
	1D1E	320D27	LD (270D),A	Zera contador de dados
	1D21	2A0127	LD HL,(2701)	Lê registro de endereço
	1D24	2B84	DEC HL	Decrementa HL
	1D25	220127	LD (2701),HL	Decrementa registro de endereço
	1D28	7E	LD A, (HL)	Lê conteúdo do novo endereço
	1D29	320027	LD (2700),A	Atualiza registro de dados
	1D2C	CD701D	CALL (A2-1)	Atualiza registro dos displays 2 e 1
	1D2F	CDB91D	CALL (A6-3)	Atualiza registro dos displays 6 a 3
	1D32	C9	RET	Retorna à rotina principal

Figura 4.65 Sub-rotina Tecla Decrementação (DEC)

. LD A, (270E)
O acumulador é carregado com o conteúdo do contador de endereço.

. CP 04
O contador de endereço (270E) é comparado com a constante "04" caso haja igualdade, o flag zero é ligado.

. JR Z (INIC)
Se o flag zero foi ligado na instrução anterior, o programa continua no endereço simbólico "INIC" (início), iniciando então a sub-rotina DEC.

. CP 05
O contador de endereço (270E) é comparado com a constante "05". Caso haja igualdade, o flag zero é ligado.

. JR Z (INIC)
Se o flag zero foi ligado na instrução anterior, o programa continua no endereço simbólico "INIC" (início), indicando então A sub-rotina DEC.

. RET
Esta é a primeira saída da sub-rotina "DEC". Só ocorre quando a comparação da 2ª E 4ª linhas resultar numa igualdade.

. LD A,00
Zera o conteúdo do acumulador, preparando-o para a próxima instrução.

. LD (270D), A
Transfere o conteúdo do acumulador (zero) para o contador de dados.

. LD HL,(2701)
Transfere o conteúdo dos endereços 2701 e 2702 para os registros "L" e "H" respectivamente.

. DEC HL
Decrementa o conteúdo do par de registros "HL".

. LD (2701), HL
Transfere o conteúdo de HL para os endereços 2701 e 2702, decrementando assim, os registros de endereços.

. LD A, (HL)
Lê o conteúdo do endereço determinado pelo par "HL"

. LD (2700),A
Transfere o conteúdo do endereço determinado por HL, para o registro de dados (2700)

. CALL (A2-1)
Atualiza com códigos de sete segmentos, os registros de código dos displays 2 e 1.

. CALL (A6-3)
Atualiza com código de sete segmentos os registros de código dos displays 6, 5, 4 e 3.

. RET
É a 2ª saída desta sub-rotina sempre ocorre em condições normais, retornando o programa para a rotina principal.

Para finalizar a sub-rotina "DEC", veja seus códigos objeto na figura 4.66.

```
1D10-   3A 0E 27 FE 04 28 05 FE 05 28 01 C9 3E 00 32 0D
1D20-   27 2A 01 27 2B 22 01 27 7E 32 00 27 CD 70 1D CD
1D30-   B9 1D C9
```

Figura 4.66 Códigos Objeto da Sub-rotina Tecla de Decrementação (DEC)

S) ROTINA TECLA GO (G0)
Endereços ocupados: 1C85 a 1C88

- Fluxograma:
Esta rotina não tem retorno para o programa principal, por esta razão não é tratada como sub-rotina.

Apesar de ser uma rotina muito simples, é de grande importância no programa monitor. Sua finalidade é de desligar o programa monitor e dar o controle ao programa do usuário gravado na memória. Em outras palavras, esta rotina desvia o programa monitor para qualquer área de memória, executando o programa que lá estiver.
O nome "G0" foi utilizado por já existir em diversos microcomputadores deste tipo, significa partida. Portanto toda vez que a tecla "G0" for acionada, o programa monitor "pára", partindo o programa cujo endereço inicial está apresentado no display. Esta rotina será melhor entendida na parte de operação do microcomputador APL-80, que será estudada mais adiante.
Por se tratar de uma sub-rotina extremamente simples, dispensaremos o fluxograma propriamente dito e passaremos a analisar o programa completo, que pode ser observado na figura 4.67.

ENDEREÇO SIMBÓLICO	ENDR. FÍSICO	CÓDIGO OBJETO	ASSEMBLY	COMENTÁRIOS
	1C85	2A0127	LD HL,(2701)	Lê registro de endereço
	1C88	E9	JP (HL)	Salta para endereço em HL

Figura 4.67 Rotina Tecla "G0" (G0)

. LD HL,(2701)
Esta instrução transfere os conteúdos dos registros de endereço 2701 e 2702 para os registros "L" e "H" respectivamente. Isto significa que após a execução desta instrução, o par HL conterá o mesmo endereço mostrado no display.

. JP (HL)

A instrução JP (HL) salta para o endereço contido no par "HL". Como "HL" foi carregado com o valor apresentado no display, então conclui-se que o programa prossegue a partir do endereço "displayado".

Por se tratar de uma rotina de apenas 4 bytes, não expomos seus códigos objeto como de costume.

4.4. Códigos Objeto do Programa Monitor:

Para que programa monitor possa ser "carregado" numa memória tipo Eprom, é necessária a listagem em linguagem de máquina (código objeto). Para facilitar este procedimento, utilize a listagem da figura 4.68.

```
0000- C3 4B 1C

1C4B- 31 FF 27 3E 00 32 0E 27 3E FF 32 0F 27 21 00 1D
1C5B- 22 03 27 CD 4D 1F CD 6B 1F 3A 05 27 FE FF 28 ED
1C6B- CD 90 1C CD 4D 1F CD 6B 1F 3A 05 27 FE FF 28 F3
1C7B- 18 EE FF FF FF FF FF FF FF FF 2A 01 27 E9 FF FF
1C8B- FF FF FF FF FF FE F1 CA D2 1E FE F2 CA 35 1D FE
1C9B- F3 CA 10 1D FE F4 CA 85 1C FE F5 C8 FE F6 C8 FE

1CAB- F7 C8 FE F8 C8 3A 0F 27 FE FF C8 C3 C0 1C FF FF
1CBB- FF FF FF FF FF 3A 0E 27 FE 05 28 18 3C 32 0E 27
1CCB- FE 01 CA A7 1E FE 02 CA 77 1E FE 03 CA 57 1E FE
1CDB- 04 CA 25 1E 3A 0D 27 FE FF C8 3C 32 0D 27 FE 01
1CEB- CA 00 1E C3 90 1D FF FF FF FF FF FF FF FF FF FF
1CFB- FF FF FF FF FF 77 67 0E 40 7F 3F FF FF FF FF FF

1D0B- FF FF FF FF FF 3A 0E 27 FE 04 28 05 FE 05 28 01
1D1B- C9 3E 00 32 0D 27 2A 01 27 2B 22 01 27 7E 32 00
1D2B- 27 CD 70 1D CD B9 1D C9 FF FF 3A 0E 27 FE 04 28
1D3B- 04 FE 05 28 11 3E 00 32 0D 27 3A 0E 27 3C 32 0E
1D4B- 27 2A 01 27 18 04 2A 01 27 23 22 01 27 7E 32 00

1D5B- 27 CD B9 1D CD 70 1D C9 32 00 27 CD B9 1D CD 70
1D6B- 1D C9 FF FF FF 3A 00 27 E6 F0 1F 1F 1F 1F CD 20
1D7B- 1F 32 0A 27 3A 00 27 E6 0F CD 20 1F 32 0B 27 C9
1D8B- FF FF FF FF FF 3A 05 27 47 3A 00 27 80 32 00 27
1D9B- 2A 02 27 77 3E 00 32 0A 27 32 0B 27 32 0D 27 23

1DAB- 22 01 27 21 06 27 22 03 27 CD B9 1D C9 FF 3A 02
1DBB- 27 E6 F0 1F 1F 1F 1F CD 20 1F 32 06 27 3A 02 27
1DCB- E6 0F CD 20 1F 32 07 27 3A 01 27 E6 F0 1F 1F 1F
```

```
1DDB- 1F CD 20 1F 32 08 27 3A 01 27 E6 0F CD 20 1F 32
1DEB- 09 27 C9 FF FF FF FF FF FF FF FF FF FF FF FF FF
1DFB- FF FF FF FF FF 3A 05 27 07 07 07 07 32 00 27 3A

1E0B- 05 27 CD 20 1F 32 0A 27 21 06 27 22 03 27 C9 FF
1E1B- FF FF FF FF FF FF FF FF FF FF 3A 01 27 E6 0F 07
1E2B- 07 07 07 32 01 27 47 3A 05 27 80 32 01 27 3A 05
1E3B- 27 CD 20 1F 32 09 27 21 06 27 22 03 27 C9 FF FF
1E4B- FF FF FF FF FF FF FF FF FF FF FF FF 3A 05 27 32

1E5B- 01 27 CD 20 1F 32 08 27 21 06 27 22 03 27 C9 FF
1E6B- FF FF FF FF FF FF FF FF FF 32 0E 27 3A 02 27 07
1E7B- 07 07 07 32 02 27 47 3A 05 27 80 32 02 27 3A 05
1E8B- 27 CD 20 1F 32 07 27 21 06 27 22 03 27 C9 FF FF
1E9B- FF FF FF FF FF FF FF FF FF FF FF FF 3A 05 27 32

1EAB- 02 27 CD 20 1F 32 06 27 3E 00 32 07 27 32 08 27
1EBB- 32 09 27 32 0A 27 32 0B 27 21 06 27 22 03 27 3E
1ECB- FF 32 0C 27 C9 FF FF 3E 00 32 0F 27 32 0E 27 3E
1EDB- FF 32 0D 27 21 E8 1E 22 03 27 C9 FF FF 77 7C 7C
1EEB- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
1EFB- FF FF FF FF FF 3F 30 6D 79 72 5B 5F 31 7F 73 77

1F0B- 5E 0F 7C 4F 47 FF FF FF FF FF FF FF FF FF FF FF
1F1B- FF FF FF FF FF 21 00 1F BD 28 03 2C 18 FA 7E C9
1F2B- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
1F3B- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
1F4B- FF FF 2A 03 27 06 20 78 32 00 60 7E FE FF C8 32

1F5B- 00 40 23 CB 08 0C 20 FD 18 ED 61 1F C3 52 1F FF
1F6B- 3E FF 32 05 27 3E 00 32 00 40 21 E8 1F 0E 01 06
1F7B- FE 78 32 00 60 3A 00 80 A1 FE 00 28 10 23 CB 00
1F8B- 78 FE FE 20 EC CB 01 79 FE 08 C8 18 E2 7E 32 05
1F9B- 27 CD B0 1F 3A 00 80 A1 FE 00 28 F8 CD B0 1F C9

1FAB- FF FF FF FF FF 21 32 FF 25 20 FD 2D 20 FA C9 FF
1FBB- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
1FCB- FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
1FDB- FF FF FF FF FF FF FF FF FF FF FF FF FF 00 01 02
1FEB- 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F F1 F2 F3
1FFB- F4 F5 F6 F7 F8
```

Figura 4.68 Listagem do Programa Monitor

4.5 Operação do Módulo - Microcomputador APL-80

A finalidade básica do programa monitor é controlar o teclado e o display de forma correta, com o objetivo de endereçar, gravar, ler e corrigir dados na memória, além de executar programas.
Para que você possa executar algumas destas tarefas, é necessário digitar o teclado na sequência correta. Portanto, a seguir, daremos exemplos para cada caso. Veja para tanto a figura 4.69.

Figura 4.69 - Distribuição do Display e Teclado do Microcomputador - APL-80

a) Inicialização do Sistema:
Para dar início à execução do programa monitor é necessário ligar a alimentação de 5 volts e em seguida acionar a tecla "RESET" no próprio microcomputador APL-80.
É conveniente comentar que o "RESET" é automático, não sendo necessário, na maioria dos casos, acionar a tecla RESET. Porém, se algo diferente ocorrer no acionamento automático, pode ser utilizado o RESET manual.
Tão logo o RESET (automático ou manual) seja acionado, o display exibirá a mensagem "APL-80". Quando isto ocorrer, significa que o sistema está apto a aceitar uma operação do teclado.

b) Endereçamento da Memória:
Para gravar, ler ou mesmo executar programas na memória, é necessário inicialmente determinar o endereço inicial da memória, onde serão, gravados, lidos ou executados os dados.

O endereçamento da memória é iniciado, acionando-se a tecla "ADD" (endereçamento). Tão logo a tecla "ADD" seja acionada, o display apaga a mensagem APL-80, e acende nos 3 primeiros dígitos da esquerda, com a mensagem "ADD", mantendo apagados os demais displays.
Neste estágio, o programa está aguardando os 4 dígitos de endereço. Para tanto basta digitar 4 teclas hexadecimais quaisquer. Apenas como exemplo, vamos utilizar o endereço "2000" que tanto pode ser lido como gravado, pois trata-se da memoria RAM.
Ao acionarmos a tecla 2, a mensagem "ADD" apaga e acende o odisplay nº 6 como o valor 2. Os demais ficam apagados.
Ao acionarmos a tecla "0" 3 vezes, os displays 5, 4 e 3 acendem em sequência com o valor "0", completando os 4 dígitos do display com o número "2000".
Neste instante, o endereçamento está completo e os 2 displays de dados estão apagados, aguardando serem gravados ou lidos. Portanto a sequência de endereçamento está terminada.

c) Gravação na memória:
Uma vez que o endereçamento foi completado e o endereço 2000 foi selecionado, podemos gravar um dado neste endereço.
Para tanto digitamos a tecla "ENTR" (entrada). Quando isto é realizado, os displays 1 e 2 acendem com um valor qualquer, correspondente ao conteúdo do endereço 2000.
Neste instante, o programa está aguardando as teclas de dados serem digitadas. Apenas como exemplos vamos gravar o valor 73 no endereço 2000.
Isto é realizado, digitando-se a tecla "7" seguida da "3".
Quando acionamos a tecla 7, o display nº 2 acende com o valor 7, enquanto que o display nº 1, apaga.
Quando acionamos a tecla 3, os displays nº 1 e nº 2 apagam, e o valor do endereço 2000 nos 4 primeiros displays de endereço, passa para 2001. Isto significa que a memória 2000 foi gravada com o valor 73, e que o endereço 2001 já está disponível para ser lido ou gravado.
Suponha agora, que desejamos gravar "5F" no endereço 2001.O processamento é idêntico, ou seja, acionamos em sequência a tecla "5" seguida da "F".
Ao acionar a tecla "5", o display nº 2 acende com o valor "5", enquanto que o display nº1 apaga.
Ao acionar a tecla "F", os 2 displays nº 1 e nº 2 apagam, enquanto que os displays nº 6, 5, 4 e 3 acendem com o valor 2002.
Neste instante, o código "F5" é carregado na memória 2001, enquanto que o display aponta o novo endereço (2002) disponível para receber novo dado.
Este processo se repete, e enquanto estivermos digitando valores hexadecimais, o programa monitor armazenará byte a byte na memória, sequencialmente.
Lembre-se que a única área de memória disponível para gravação compreende os endereços 2000 a 27FF.
Um outro fato importante, é que não devem ser utilizados os endereços 2700 a 27FF, pois estes são manipulados pelo programa munitor.

Para interromper uma sequência de gravação e continuar em outro endereço, basta acionar a tecla "ADD", os 4 dígitos de endereço, a tecla "ENTR" e finalmente, os novos dados a serem introduzidos na memória.

d) Leitura de Memória:
Para realizarmos a leitura da memória, é necessário endereçá-la conforme item "b", em seguida acionar a tecla enter cada vez que desejarmos executar uma nova leitura.
O endereço de memória é automaticamente incrementado . Por exemplo, suponha que desejamos ler a última linha do programa monitor. Veja a figura 4.68.
O endereço inicial é 1FFB, portanto digita-se "ADD" seguido das teclas 1, F, F e B.
Neste instante, cada vez que for acionada a tecla "ENTR", você terá um "byte" na memória, conforme segue:

```
1FFB  -F4
1FFC  -F5
1FFD  -F6
1FFE  -F7
1FFF  -F8
```

e) Leitura Decrementativa:
Também é possível ler a memória para trás. Para isto, basta endereçar a memória, acionar a tecla ENTR e finalmente digitar uma vez a tecla "DEC" para cada endereço lido.
O decremento no display de endereço é automático. Note que esta tecla tem função inversa da tecla "ENTR".
A tecla "DEC" pode ser utilizada para corrigir uma gravação, ou seja, ao acioná-la no meio de um processo de gravação, o endereço volta uma unidade, permitindo regravar novo dado.

f) Execução de Programas:
Trata-se da parte mais importante da operação, pois pelo processo descrito a seguir, torna-se possível executar qualquer programa de memória.
Para executar ou "rodar" um programa na memória, é necessário antes gravá-lo de acordo com o item "C".
Uma vez gravado o programa na memória, devemos posicionar o APL-80 com o endereço inicial do programa a ser executado, e finalmente acionar a tecla "G0". Neste instante, o programa monitor é desligado, e o microprocessador Z-80 passa a executar o programa, cujo endereço inicial estava presente no display.

Por exemplo, grave na memória "RAM" o seguinte programa.

```
2000  -C3    ou   2000-C30000 - JP (0000)
2001  -00
2002  -00
```

Após carregar a memória com os dados descritos anteriormente, enderece a memória em 2000 e digite a tecla "G0". Neste instante, aparecerá a mensagem "APL-80" no display.
Este programa é bem simples, pois salta do endereço 2000 para o endereço 0000, onde dá execução no início do programa monitor. Por esta razão acende a mensagem "APL-80" no display.
Logicamente, programas mais complexos podem ser executados, basta carregá-los na memória conforme já estudamos.

4.6 Programa "Eco-teste"

Para você ficar mais familiarizado com a operação do módulo microcomputador APL-80, tente carregar o programa "eco-teste" na memória e executá-lo.A função deste programa é mostrar no display o nome ou número da tecla acionada.Para executar este programa, é necessário antes carregá-lo nos endereços 2000 a 2067 (Figura 4.71).
A tabela de caracteres da figura 4.70também deve ser carregada nos endereços 2200 a 225C .

2200	FE	3F	00	00 0
2204	FD	30	00	00 1
2208	FB	6D	00	00 2
220C	F7	79	00	00 3
2210	EF	72	00	00 4
2214	DF	5B	00	00 5
2218	BF	5F	00	00 6
221C	7F	31	00	00 7
2220	FE	7F	00	00 8
2224	FD	73	00	00 9
2228	FB	77	00	00 ..... A
222C	F7	5E	00	00 ..... B
2230	EF	0F	00	00 C
2234	DF	7C	00	00 ..... D
2238	BF	4F	00	00 ..... E
223C	7F	47	00	00 ..... F
2240	FE	77	7C	7C ..... ADD
2244	FD	4F	54	78 ..... ENT
2248	FB	7C	4F	0F ..... DEC
224C	F7	1F	3F	00 ..... G0
2250	EF	47	30	00 ..... F1
2254	DF	47	6D	00 ..... F2
2258	BF	47	79	00 ..... F3
225C	7F	47	72	00 ..... F4

Figura 4.70 Tabela de Caracteres da Rotina "Eco-teste".

ENDEREÇO SIMBÓLICO	ENDER. FÍSICO	CÓDIGO OBJETO	ASSEMBLY	COMENTÁRIOS
(INIC)	2000	31FF27	LD SP(27FF)	Seta stack pointer 27FF
	2003	3E00	LD A,00	Zera acumulador
	2005	320040	LD (4000),A	Zera linhas de segmentos do display
	2008	210022	LD HL,2200	Aponta início da tabela de dados
(CIC 1)	200B	CD3520	CALL (LEIT)	Chama sub-rotina leitura de teclado (LEIT)
	200E	E601	AND 01	Isola linha Y0 do teclado
	2010	CD3D20	CALL (COMP)	Chama sub-rotina comparação (COMP)
	2013	FE20	CP 20	É última linha X?
	2015	C20B20	JP NZ (CIC 1)	Se não for zero vai para ciclo 1 (CIC 1)
(CIC 2)	2018	CD3520	CALL (LEIT)	Chama sub-rotina leitura de teclado (LEIT)
	201B	E602	AND 02	Isola linha Y1 do teclado
	201D	CD3D20	CALL (COMP)	Chama sub-rotina comparação (COMP)
	2020	FE40	CP 40	É última linha X?
	2022	C21820	JP NZ (CIC 2)	Se não for zero vai para ciclo 2 (CIC 2)
(CIC 3)	2025	CD3520	CALL (LEIT)	Chama sub-rotina leitura de teclado (LEIT)
	2028	E604	AND 04	Isola linha Y2 do teclado
	202A	CD3D20	CALL (COMP)	Chama sub-rotina comparação (COMP)
	202D	FE60	CP 60	É última linha X?
	202F	C22520	JP NZ (CIC 3)	Se não for zero vai para ciclo 3 (CIC 3)
	2032	C30020	JP (INIC)	Vai para início (INIC)
(LEIT)	2035	7E	LD A,(HL)	Lê tabela (1º byte)
	2036	320060	LD (6000), A	Liga latch de fileira
	2039	3A0080	LD A,(8000)	Lê porta de teclado (linha Y)
	203C	C9	RET	Retorno à rotina originária
(COMP)	203D	FE00	CP 00	É tecla acionada
	203F	CC4720	CALL Z (DISP)	para sub-rotina de display (DISP)
	2042	7D	LD A,L	|
	2043	C604	ADD 04	> Soma 4 em L
	2045	6F	LD L,A	|
	2046	C9	RET	Retorna à rotina originária
(DISP)	2047	3E04	LD A,04	Prepara 3º display
	2049	CD5B20	CALL (ACIO)	Chama sub-rotina aciona[mento] display (ACIO)
	204C	3E02	LD A,02	Prepara 2º display
	204E	CD5B20	CALL (ACIO)	Chama sub-rotina aciona[mento] display (ACIO)
	2051	3E01	LD A,01	Prepara 1º display
	2053	CD5B20	CALL (ACIO)	Chama sub-rotina aciona[mento] display (ACIO)
	2056	7D	LD A,L	|
	2057	D603	SUB 03	> Subtrai 3 de L
	2059	6F	LD L,A	|
	205A	C984	RET	Retorna à rotina originária
(ACIO)	205B	320060	LD (6000),A	Liga catodo do display
	205E	23	INC HL	Avança 1 byte na tabela
	205F	7E	LD A,(HL)	Lê tabela
	2060	320040	LD (4000),A	Liga segmentos do display
LOOP	2063	0C	INC C	Temporiza
	2064	C26320	JP NZ (LOOP)	
	2067	C9	RET	Retorna à rotina originária

Figura 4.71 Programa Eco-teste

Uma vez carregado o programa "Eco-teste" com a sua respectiva tabela, deve-se digitar "ADD", "2000" seguido da tecla "G0". Neste instante, o display apaga, e permanece apagado até o instante que uma tecla for acionada.
Quando acionarmos uma tecla qualquer, o display acenderá com o nome da mesma, permanecendo acesa até o instante que for desacionada. Nesta ocasião, o display apaga. Este ciclo se repete sempre que uma tecla for acionada.
Para melhor entendimento do programa, vamos estudá-lo na sequência de suas instruções, de acordo com a figura 4.71.

. LD SP (27FF)
O endereço 27FF é o último disponível na memória RAM, portanto a instrução LD SP (27FF) determina o stack pointer com este valor, permitindo que os dados manipulados pelo Z-80, sejam gravados da última posição da memória RAM, para baixo.

. LD A,00
Zera o acumulador, preparando-o para próxima instrução

. LD (4000),A
Zera todas as linhas de segmentos do display para que não acendam durante uma leitura de teclado. Lembre-se que o endereço 4000 corresponde fisicamente ao latch de segmentos do display.

. LD HL,2200
A tabela de dados que é utilizada para apresentar um caractere no display começa em 2000, portanto o par HL é carregado com 2200 para ser utilizado posteriormente.

. CALL (LEIT)
A sub-rotina de leitura (LEIT) inicia no endereço 2035. Tem a finalidade de ler a tabela de caracteres, ligar o latch de fileira, selecionando apenas 1 linha, e finalmente ler as linhas verticais, ou colunas do teclado.
Observe na tabela, que o primeiro byte de cada linha, contém um código que se repete a cada 9 endereços.
A finalidade deste código, é selecionar com nível zero, apenas uma linha "x" (fileira) do teclado por vez. Vejam que os códigos do primeiro byte de cada linha da tabela são FE, FD, FB, F7, EF, DF, BF e 7F. Portanto, estes varrem as oito linhas de fileira do teclado.
Como o teclado possui 24 teclas, então torna-se necessário repetir estes códigos, 3 vezes (3x8=24).

Para melhor compreensão da sub-rotina de leitura, vejamos as suas 4 instruções:

a) **LD A,(HL)**
Esta instrução localizada em 2035, carrega o acumulador com o primeiro byte da tabela de caracteres, gerando o código que determinará qual linha x será selecionada com nível "0".

Portanto, os códigos lidos FE, FD, FB, F7, EF, DF, BF e 7F geram nível zero nas linhas X0, X1, X2, X3, X4, X5, X6 e X7 respectivamente.

b) **LD (6000),A**
Transfere o código gerado pela instrução anterior para o latch de fileira do teclado, selecionando assim uma linha de fileira com nível 0 mantendo as demais com nível 1.

c) **LD A,(8000)**
Esta instrução lê o estado das 3 linhas verticais do teclado e deposita o código lido no acumulador.

d) **RET**
Esta última instrução da sub-rotina de leitura, retorna o programa, para executar a próxima instrução, logo após a sub-rotina de leitura ter sido "chamada".

. AND 01
Quando a sub-rotina de leitura é executada pela instrução anterior, as 3 linhas "y" do teclado são lidas, porém apenas uma deve ser utilizada por vez.
A instrução AND 01 isola exatamente a linha Y0, com a finalidade de determinar se alguma tecla correspondente a esta coluna foi acionada.

. CALL (COMP)
A sub-rotina de comparação tem duas finalidades. A primeira delas é determinar se houve uma tecla acionada durante à última leitura do teclado. Caso afirmativo, a sub-rotina de display é iniciada, dando sequência à exibição do nome da tecla acionada, no display. A segunda finalidade da sub-rotina "COMP", é incrementar o registro "L" 4 unidades, apontando assim o endereço do próximo byte da tabela de caracteres, referente à próxima linha "X" a ser varrida.

. CP 20
A tabela de caracteres está dividida em 3 partes.
Os endereços de 2200 a 221F correspondem às teclas 0 a 7.
Os endereços de 2220 a 223F correspondem às teclas 8 a F.
Os endereços de 2240 a 225F correspondem às teclas ADD, ENT, DEC, G0, F1, F2, F3 e F4.
Também é conveniente lembrar que cada grupo de tecla mencionando é lido por uma linha Y e varrido pelas linhas X0 a X7.
Portanto as instruções CP 20, CP 40 e CP 60 correspondentes à 8ª, 13ª e 18ª linhas do programa, comparam quando o registro "L", atingiu a última linha X de varredura para cada grupo de tecla.
Quando o resultado da comparação for igualdade, então o flag zero será ligado.

. JP NZ (CIC 1)
Se o programa não atingiu a última linha X, referente ao primeiro grupo de teclas, então é reciclado novamente para o endereço "CIC 1" (ciclo 1).

. CALL (LEIT)
Chama novamente a leitura de teclado, selecionando nova linha X.

. AND 02
Isola a linha Y1 do teclado, verificando se existe alguma tecla acionanda na linha "Y 1".

. CALL (COMP)
Vai para sub-rotina de comparação, verificando se a tecla foi acionada ou não, é incrementando o registro "L" 4 unidades.

. CP 40
Verifica se o registro "L" é igual a 40, caso afirmativo, significa que a última tecla, referente às linhas Y1 e X7, foi varrida.
Quando a última tecla da 2ª coluna for varrida, a instrução CP 40 ligará, o flag zero.

. JP NZ (CIC 2)
A instrução JP NZ (CIC 2) salta para o endereço "CIC 2" (ciclo 2) só quando o flag zero estiver desligado, caso contrário, o programa continua na linha seguinte.

. CALL (LEIT)
Quando o programa atinge esta instrução pela 1ª vez, significa que a 1ª linha "X" (X0), referente à coluna Y2, está sendo varrida. Portanto uma nova leitura de teclado tem início.

. AND 04
A linha Y2 do teclado é isolada, com o objetivo de verificar, se existe alguma tecla acionada.

. CALL (COMP)
Uma nova comparação é realizada, verificando se a tecla foi acionada ou não, é incrementado o registro "L" 4 unidades.

. CP 60
Verifica se o registro L contém o valor 60, caso afirmativo significa que a última tecla referente às linhas Y2 e X7, foi varrida. Se isto ocorrer, o flag zero é ligado.

. JP NZ (CIC 3)
Se o flag da instrução anterior não estiver ligado, o programa vai para o endereço "CIC 3" (ciclo 3), caso contrário, a próxima instrução é executada.

. JP (INIC)
Quando o programa atingir este ponto, significa que todas teclas já foram varridas, portanto, volta para o início (INIC) para executar novamente todo o programa.

. **LD A,(HL)**
Esta instrução já estudada, faz parte da sub-rotina de leitura

. **LD (6000),** A Idem à anterior

. **LD A (8000)** Idem à anterior

. **RET** Idem à anterior

. **CP 00**
Esta instrução faz parte da sub-rotina de comparação. Tem a finalidade de ligar o flag zero, caso a comparação entre o acumulador e a constante "00" resulte numa igualdade.

. **CALL Z (DISP)**
Se o flag zero estiver ligado, esta instrução chama a sub-rotina de display, com a finalidade de apresentar no display o nome da tecla acionada.
A sub-rotina de display será examinada mais adiante, a partir do endereço 2047.

. **LD A,L**
Transfere o conteúdo do registro "L" para o acumulador.

. **ADD 04**
Incrementa 4 unidades ao conteúdo do acumulador.

. **LD L,A**
Devolve o valor do acumulador incrementado para o registro "L".

. **RET**
Retorna para a instrução seguinte ao "CALL" que originou a sub-rotina de comparação (COMP).

. **LD A,04**
Esta é a primeira instrução da sub-rotina de display. Sua função é posicionar o acumulador para endereçar, ou acionar o 3º display.

. **CALL (ACIO)**
Chama a sub-rotina de acionamento do display (ACIO), com a finalidade de acender o 3º display com o primeiro caractere da tecla acionada.
Existem, no máximo, 3 caracteres na tabela para representar o nome da tecla acionada. O primeiro, segundo e terceiro caracteres apresentados no display, correspondem respectivamente ao 3º, 2º e 1º displays.
Suponha por exemplo, que a tecla "ADD" foi acionada. Portanto os displays 3, 2, 1 acenderam respectivamente com os caracteres "A", "D" e "D".

Se o nome da tecla for composto por apenas 1 caractere, o display 3 acenderá com o caractere referente ao nome da tecla, enquanto que os displays 2 e 1 receberão o código "00" de acionamento, mantendo-os apagados. Observe como exemplo, a primeira linha da tabela da figura 4.70. (endereço 2200).
O primeiro byte da tabela, cujo código é FE, seleciona a linha X de varredura do teclado, e não é utilizado no processo de acendimento do display.
O segundo byte da tabela, cujo código é "3F", representa o caractere que forma o dígito zero. Este caractere é apresentado no 3º display.
O terceiro e quarto byte da tabela, cujo código é "00", apagam os displays 2 e 1.

. LD A,02
Posiciona o acumulador para selecionar o 2º display durante a sub-rotina de acio – namento do display (ACIO).

. CALL (ACIO)
Chama a sub-rotina de acionamento do display, com a finalidade de acender o 2º display com o 3º byte da tabela. Lembre-se que o 1º byte da tabela não é utilizado pelo display e sim pelo teclado.

. LD A,01
Posiciona o acumulador para selecionar o 1º display durante a sub-rotina de acionamento do display.

. CALL (ACIO)
Aciona o 1º display com o código referente ao 4º byte da tabela, gerando assim, o último caractere no display, com o nome da tabela acionada.

.LD A,L Lê o registro L, carregando seu conteúdo no acumulador.

. SUB 03
Decrementa 3 unidades do acumulador, preparando-o para posicionar o registro L numa condição para retornar ao programa principal no endereço 2042.
Para melhor compreensão desta instrução (SUB 03), da anterior (LD A,L) e da próxima (LD L,A), suponha que o registro L, contenha o valor 47, referente ao último endereço (2247) da 18ª linha da tabela da figura 4.70.
Quando o programa ajunta o conteúdo do registro H e L, forma o endereço 2247.
Se você observar, o endereço 2247 contém o valor 78, que corresponde ao código do caractere "T", que está sendo apresentado no display. Suponha para tanto, que a tecla "ENTER" esteja acionada.
Quando o programa executar as instruções "LD A,L", "SUB 03" e LD L,A, o registro "L" será subtraído 3 unidades, voltando ao primeiro endereço da 18ª linha da tabela, ou seja 2247 - 3 = 2244.
Uma vez que o registro L estiver posicionado no início de qualquer linha da tabela, o resto do programa terá facilidade em sincronizar o endereçamento dos bytes na tabela.

. LD L,A
Esta instrução já estudada na instrução anterior, transfere o conteúdo do acumulador decrementado 3 unidades para o registro L.

. RET
Retorna para o programa que originou a sub-rotina de display.

. LD (6000),A
Esta é a primeira instrução da sub-rotina de acionamento de display. Sua função é de "acionar" o catodo do display, correspondente ao caracter apontado pela tabela.

. INC HL
Incrementa uma posição na tabela para ler o próximo caractere a ser exibido no display.

. LD A,(HL)
Carrega o acumulador com o código lido na tabela.

. LD (4000),A
Transfere o conteúdo do acumulador para o latch de segmentos, acendendo o display com o caractere da tabela.

. INC C
Incrementa o registro com uma unidade com o objetivo de atrasar o programa.

. JP NZ (LOOP)
Verifica se a instrução anterior zerou o acumulador, caso contrário, retorna para o endereço (LOOP).
Observe que esta instrucão, bem como a anterior, formam um pequeno temporizador com a finalidade de estabilizar o código no display.

. RET
Retorna para a rotina que gerou o acionamento do display.

CAPÍTULO 5

INTERFACE

DE

IMPRESSORA

ÉRICA

5.1 - Introdução:

Agora que você já entendeu o funcionamento da placa do microcomputador APL-80, já está apto a iniciar o estudo de seus "interfaces", ou seja, a manusear um dispositivo de entrada ou saída, conectando para isto, o micro APL-80 numa outra placa que iremos estudar a seguir.

É conveniente salientar que, o microcomputador desempenha grande parte do controle de um periférico.

Por esta razão circuitos de interface a ele conectados, tornam-se na maioria das vezes extremamente simples, possibilitando até o leitor menos experiente, a desenvolver seus próprios interfaces.

Uma das interfaces mais interessantes que encontramos para aplicação prática, foi módulo de controle para impressora. Com este interface, é possível por exemplo utilizar a maioria das impressoras existentes no mercado nacional.

Uma aplicação típica, seria a impressão de etiquetas, relatórios, cartas etc.

Convém lembrar que nosso microcomputador é didático, portanto, deve ser utilizado dentro de uma certa simplicidade, a fim de tornar mais simples a programação.

Embora nosso microcomputador, quando bem programado, e bem assessorado por circuitos de interface, possa executar tarefas complexas como os computadores comerciais, não é nosso objetivo atingir esta meta, porém, mostrá-lo funcionando na sua forma mais simples e didática.

Existem tarefas que aparentemente são complexas, porém, como são repetitivas, tornam-se simples para nosso micro. Podemos citar por exemplo, a impressão de vários rótulos, com a mesma especificação de um único produto.

Outra aplicação, seria a impressão dos números de série, de um determinado artigo. E finalmente, a aplicação mais atrativa, seria utilizar nosso micro com seu respectivo módulo de interface, para imprimir diversas cartas ou circulares com o mesmo conteúdo, isto é utilizá-lo como uma máquina copiadora.
O procedimento para realizar a tarefa de impressão repetitiva é realtivamente simples, bastando para isto gravar na memória o conteúdo da impressão. A seguir. deve-se vincular este conteúdo a um programa de impressão. Finalmente, deve-se executar o programa de impressão, dando-se início à impressão repetitiva.

5.2 Interligação entre Módulo Microcomputador/ Interface/ Impressora

Para que o sistema possa operar adequadamente, são necessárias 2 interligações, ou seja uma entre a placa do microcomputador APL-80 e a placa de interface, a outra entre a placa de interface e a impressora.
A ligação entre o microcomputador APL-80 e a placa de interface foi realizada, utilizando-se um "flat cable" de 40 vias juntamente com 2 conectores de 40 pinos. Através da figura 5.1, pode se observar na placa do microcomputador APL-80, os plugs P1 e P2 que contêm os 40 sinais referentes aos 40 pinos do microprocessador Z-80. Com estes 40 sinais, o Z-80 pode controlar qualquer placa de interface, bastando para isto, apenas interligá-los adequadamente aos circuitos do módulo interface.

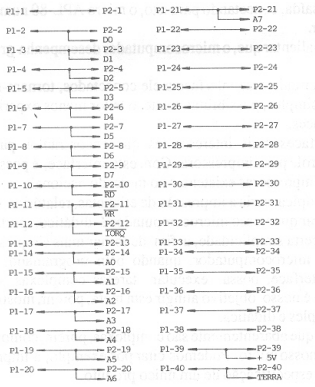


Figura 5.1 - Ligações dos Plugs (macho) P1 e P2

Convém lembrar que a placa APL-80, contém 2 plugs (P1 e P2), contendo sinais idênticos, ou seja, o pino 1 do plugs 1 contém o mesmo sinal do pino 1 do plug 2 assim em diante. Esta providência foi tomada com o intuito de permitir as ligação em cadeia ou cascata de vários periféricos, ou até mesmo 2 interfaces diretamente no microcomputador APL-80.

Outra interligação importante, está na placa de interface da impressora. Trata-se de um conctor de 36 pinos.

Nos 36 pinos, são ligados os sinais data 0 a data 7 que enviam o código do caractere a ser impresso. Na impressora os sinais "STROBE", "ACK", e "BUSY" estão também disponíveis para o controle do fluxo dos sinais de dados (data 0 a data 7). Veja através da figura 5.2, o conector C1 e os seus respectivos sinais que serão detalhados mais adiante.

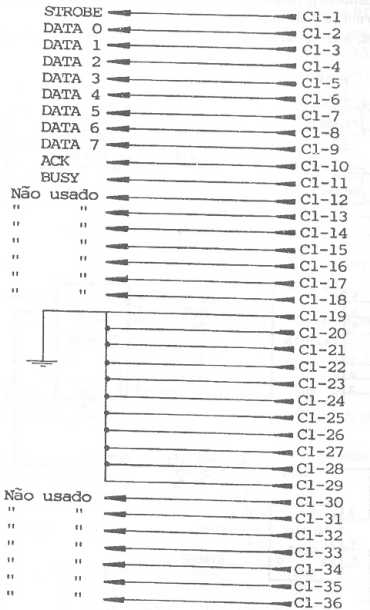


Figura 5.2 Conector (fêmea) C1 - (impressora)

Um detalhe importante a comentar, está no cabo que interliga do interface à impressora. Este deve ser constituído por um flat cabe de 36 vias, contendo numa das extremidades um plug de 36 pinos e do outro lado um conector padrão "centronics", próprio para conectar impressoras.

5.3 Decodificador de Endereço e Gerador de Strobe

Este circuito faz parte da placa de interface de impressora, e tem a finalidade de permitir que o microcomputador APL-80 consiga selecionar a placa de interface de impressora no endereço 00 de entrada e 00 de saída.
Este circuito, também gera um pulso chmado strobe, com o objetivo de ativar a impressora, iniciando um ciclo de impressão.
Vejamos agora, através da figura 5.3, como funcionam estes circuitos.

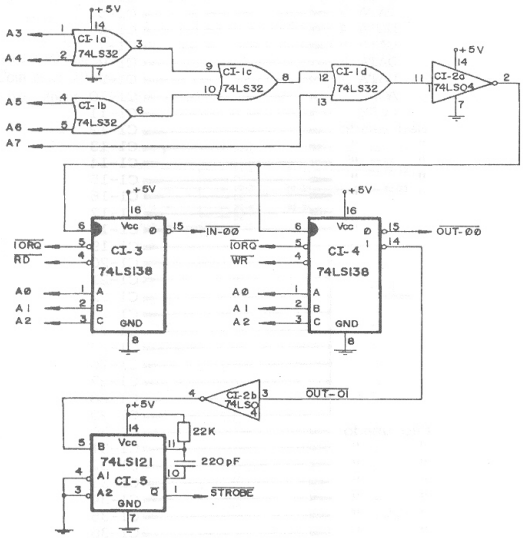


Figura 5.3 Decodificador de Endereço e Gerador de Strobe

Suponha que através de uma determinada programação (será vista mais adiante), o Z-80 envie para a placa de interface de impressora, o endereço 00, ou seja, gerando nível zero nas linhas A0, A1, A2, A3, A4, A5, A6 e A7. Quando isto acontece, o sinal IORQ também aparecerá, informando ao circuito periférico (interface), que uma operação de entrada ou saída teve início. Finalmente, se for uma operação de entrada, o Z-80 gerará o sinal RD, e se for uma operação de saída, o Z-80 gerará o sinal WR.
Veja agora ainda na figura 5.3, como é gerado o sinal de entrada "IN00".
Como dissemos anteriormente, o endereço zero é gerado pelo Z-80, nivelando todas as linhas de endereço (A0 a A7).
As linhas A0, A1 e A2 estão a nível zero e consequentemente estão ligadas às entradas A, B e C de CI-3, então a saída "zero" pino 15 é selecionada, porém, ainda não ativada, permanecendo a nível 1.
Ainda no CI-3, os pinos 5 e 4 são selecionados com os sinais IORQ e RD a nível zero. Neste momento, o CI-3 está aguardando apenas o pino 6 ficar a nível um, para habilitar e gerar o sinal IN00. Portanto, analisemos para isto os sinais A3, A4, A5, A6 e A7.
Como A3 e A4 estão a nível zero e os mesmos conectados à CI-1a pinos 1 e 2, então um nível zero aparece na saída 3 de CI-1a.
Caso semelhante ocorre com a porta CI-1-b. Os sinais A5 e A6 a nível zero, estão conectados às entradas 4 e 5 CI-1-b, consequentemente a saída 6 de CI-1-b vai a nível zero.
Continuando o mesmo circuito, os pinos 9 e 10 de CI-1-C estão a nível zero, gerando assim nível zero em CI-1c pino 8.
A entrada de CI-1-d pino 13 está a nível zero porque A7 está a nível zero. Como CI-1-d pino 12 está a nível zero (ligado a CI-1-c pino 8), a saída da porta CI-1-d pino 11 vai a zero.Em seguida, o inversor CI-2-a gera um nível 1 em CI-2-a pino 2.
Como CI-2-a pino 2 está conectado à CI-3 pino 6, então é gerado um nível zero na saída pino 15 de CI-3.
Mais adiante você verá como este sinal (IN00) é utilizado.
Ainda na mesma figura 5.3, note do lado direito, outro decodificador representado pelo CI-4. A diferença deste circuito, é que gera sinais de saída (out), e não de entrada (IN). Portanto se difere do anterior no pino 4, onde está conectado o comando WR (gravação) e não o RD (leitura), gerando assim as saídas out 00 e out 01.
Mais adiante você entenderá para que serve o sinal out 00.
O sinal out 01 que é gerado em CI-4 pino 14 é invertido por CI-2-b, atingindo desta maneira o CI-5.
Uma vez um pulso em CI-5 pino 5, este e limitado em 50 Os gerando assim o pulso strobe que será estudado mais adiante.

5.4 Input/Output Latch

Os latches de entrada e saída (input/outpt) aparecem na figura 5.4.
A função do latch de entrada (input), é de armazenar o estado do sinal "busy", vindo da impressora, e enviá-lo ao microprocessador no tempo que for selecionada a entrada IN00 pelo programa.

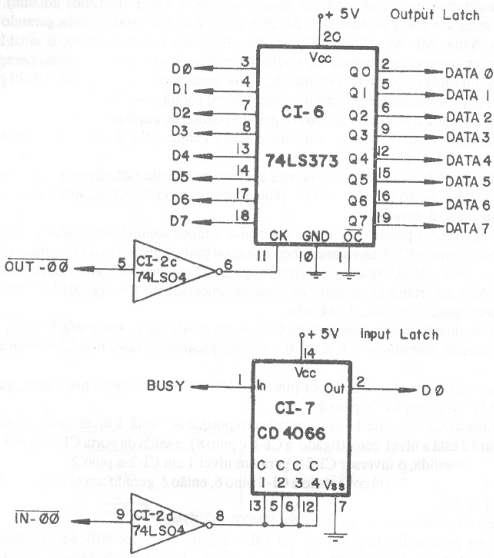


Figura 5.4 Input/Output - Latch.

O sinal busy é gerado pela impressora sempre que esta estiver "ocupada", realizando uma impressão. Este sinal só atinge nível zero no término do ciclo de impressão. Portanto, o circuito CI-7 funciona como isolador entre o microprocessador e a impressora.
O circuito integrado CI-2-d, tem a finalidade única e específica, de inverter o sinal IN00, e adaptá-lo ao circuito integrado CI-7.
O sinal OUT00 é invertido por CI-2-c e vai para a entrada clock (pino 11) de CI-6. Toda vez que este sinal é gerado, os sinais D0 a D7 são carregados do microprocessador para dentro do latch CI-6.
Uma vez os sinais dentro de CI-6, são refletidos para seu exterior nas saídas Q0 a Q7, gerando assim os sinais data 0 a data 7 que serão utilizados pela impressora.
Notem que data 0 a data 7 só serão introduzidos na impressora com o auxílio do sinal strobe.

5.5 Software

Para que o circuito de interface de impressora, possa operar adequadamente, é necessário que o microprocessador APL-80 seja programado de forma a acionar o canal zero de entrada e saída, e o canal 01 de saída.

Para enviar um caractere à impressora, é necessário que um código hexadecimal, seja transmitido pelo interface para impressora. Por exemplo, o código 41 quando enviado à impressora, faz imprimir a letra "A".

Para que uma letra ou caractere qualquer seja impresso, é necessário inicialmente, carregar o latch do interface (circuito integrado CI-6) para posteriormente enviar o sinal strobe para a impressora. Na figura 5.5, você pode observar os códigos que devem ser enviados à impressora para imprimir a letra desejada.

Code Hex	Char.	Code Hex	Char.	Code Hex	Char.	Code Hex	Char.	Code Hex	Char.	Code Hex	Char.	Code Hex	Char.	Code Hex	Char.
20	(Space)	36	6	4C	L	62	b	78	x	AF	¯	C5	Å	DB	Û	F1	ñ
21	!	37	7	4D	M	63	c	79	y	B0	°	C6	Æ	DC	Ü	F2	ò
22	"	38	8	4E	N	64	d	7A	z	B1	±	C7	Ç	DD	Ý	F3	ó
23	#	39	9	4F	O	65	e	7B	{	B2	²	C8	È	DE	Þ	F4	ô
24	$	3A	:	50	P	66	f	7C	|	B3	³	C9	É	DF	ß	F5	õ
25	%	3B	;	51	Q	67	g	7D	}	B4	´	CA	Ê	E0	à	F6	ö
26	&	3C	<	52	R	68	h	7E	~	B5	µ	CB	Ë	E1	á	F7	÷
27	'	3D	=	53	S	69	i	80	(Blank)	B6	¶	CC	Ì	E2	â	F8	ø
28	(	3E	>	54	T	6A	j	A1	¡	B7	·	CD	Í	E3	ã	F9	ù
29	)	3F	?	55	U	6B	k	A2	¢	B8	¸	CE	Î	E4	ä	FA	ú
2A	*	40	@	56	V	6C	l	A3	£	B9	¹	CF	Ï	E5	å	FB	û
2B	+	41	A	57	W	6D	m	A4	¤	BA	º	D0	Ð	E6	æ	FC	ü
2C	,	42	B	58	X	6E	n	A5	¥	BB	»	D1	Ñ	E7	ç	FD	ý
2D	-	43	C	59	Y	6F	o	A6	¦	BC	¼	D2	Ò	E8	è	FE	þ
2E	.	44	D	5A	Z	70	p	A7	§	BD	½	D3	Ó	E9	é		
2F	/	45	E	5B	[	71	q	A8	¨	BE	¾	D4	Ô	EA	ê		
30	0	46	F	5C	\	72	r	A9	©	BF	¿	D5	Õ	EB	ë		
31	1	47	G	5D	]	73	s	AA	ª	C0	À	D6	Ö	EC	ì		
32	2	48	H	5E	^	74	t	AB	«	C1	Á	D7	×	ED	í		
33	3	49	I	5F	_	75	u	AC	¬	C2	Â	D8	Ø	EE	î		
34	4	4A	J	60	`	76	v	AD	-	C3	Ã	D9	Ù	EF	ï		
35	5	4B	K	61	a	77	w	AE	®	C4	Ä	DA	Ú	F0	ð		

Figura 5.5 Tabela ASC II

No microcomputador APL-80, a sequência de programação seria:

a)LD A,41 carrega o acumulador com o código da letra "A"
b)Out 00 carrega o latch da impressora (CI-6) com o código da letra A.
c)Out 01 transmite o sinal strobe iniciando um ciclo de impressão.

Logicamente, para transferir maior quantidade de dados à impressora, o programa tem que ser um pouco mais complexo.

Inicialmente, os dados (códigos) a serem impressos devem estar na memória, em seguida, o programa deve a cada ciclo de impressão, verificar se a impressora já está desocupada para realizar nova impressão.

O programa de impressão consecutiva para vários caracteres pode ser observado através da figura 5.6. Para que este programa possa operar adequadamente, é necessário que os códigos referentes às letras ou símbolos impressos estejam ordenados em sequência a partir do endereço 2100.

ENDEREÇO SIMBÓLICO	ENDER. FÍSICO	CÓDIGO OBJETO	ASSEMBLY	COMENTÁRIOS
Partida	1000	210021	LD HL,2100	Carrega HL com 2100 (tabela de dados)
Novo Ciclo	1003	7E	LD A,(HL)	Carrega "A" com o conteúdo do END. em HL
	1004	D300	Out (00),A	Transfere "A" para canal de saída 00
	1006	D301	Out (01),A	Transfere "A" para canal de saída 01
Teste	1008	DB00	IN A,(00)	Transfere para "A" o conteúdo do canal 00 de entr.
	100A	E601	AND A,01	Isola a linha D0
	100C	C20810	JP NZ,(teste)	Vai para teste caso D0 = 0
	100F	7E	LD A,(HL)	Carrega "A" com o conteúdo do END. em HL
	1010	23	INC HL	Avança tabela de dados
	1011	FEFF	CP A,FF	Compara se é último byte
	1013	C20310	JP NZ,(novo ciclo)	Se diferente de FF vai para novo ciclo
	1016	76	HLT	Para o programa

Figura 5.6 Programa de Impressão

Outro detalhe importante é que no término da tabela de caracteres a serem impressos, deve obrigatoriamente conter o código FF, a fim de determinar o fim da tabela.

Veja através da tabela 5.1, como estes dados estão dispostos para imprimir as siglas APL-80.

2100	41	A
2101	50	P
2102	4C	L
2103	2D	-
2104	38	8
2105	30	0
2106	FF	Fim

Tabela 5.1 Tabela para Impressão das Siglas APL-80

Para melhor entendimento do funcionamento do programa da figura 5.6, vejamos em seguida, o estudo em sequência de instrução.

. LD HL, 2100

A primeira instrução deste programa, determina o endereço inicial da tabela de dados, a serem impressos.

Note que este endereço, pode estar situado em qualquer lugar da memória. Apenas utilizamos o endereço 2100 para maior facilidade de manuseio dos dados.

. LD A,(HL)

Carrega o acumulador com o conteúdo do endereço contido em 2100.

Como em 2100 contém o código 41, referente ao primeiro caractere a ser impresso, então após a execução desta instrução, o acumulador (registro A) passa a ter o valor 41.

. OUT (00),A

O conteúdo do acumulador (valor 41), é transferido para o latch de dados na placa de interface da impressora, deixando disponíveis os dados para a impressora.

. OUT (01),A

Embora o conteúdo do acumulador seja transferido para o canal 01, este não é aproveitado pelo interface da impressora, porém é aproveitado o sinal OUT 01 para gerar o pulso strobe, com o objetivo de carregar na impressora, os dados contidos em CI-6.

. IN A,(00)

Esta instrução tem a finalidade de verificar se a impressora está ocupada, executando seu ciclo de impressão.

Quando a instrução IN A,(00) é executada, as linhas D0 a D7 são lidas no canal 00 de entrada.

Como no canal 00, apenas a linha "D0" está conectada ao interface da impressora, então o sinal "busy" (ocupada) da impressora, é monitorado pela respectiva linha D0.

Portanto, veja a figura 5.4 CI-7 pinos 1 e 2.

Quando a impressora estiver ocupada, então a linha D0 vai a nível 1.

Quando a impressora estiver livre, então a linha D0 vai a nível 0.

. AND A,01

Esta instrução, tem a finalidade de isolar apenas a linha "D0" do acumulador, com o objetivo de ser testada mais tarde pelo programa.

Se a linha D0 estiver a nível "0", então o resultado será 00, não importando o estado dos demais bits. Trata-se de uma operação lógica AND.

. JP NZ(teste)

A instrução JP NZ (teste) salta para o endereço simbólico "teste" (endereço) físico 1008), apenas se o resultado da operação realizada pela instrução anterior não for nulo. Ou seja, caso a impressora esteja ocupada.

. INC HL

Uma vez que a impressora desocupe, a instrução "INC HL" é executada, apontando então para o endereço do próximo caractere a ser impresso.

. LD A(HL)

Carrega o acumulador com o próximo dado a ser impresso. Neste caso, seria o 2º caractere (letra P).

. CP A,FF

Compara se é o último caractere da tabela. Caso afirmativo, liga o flag zero, preparando para a próxima instrução.

. JP NZ, (novo ciclo)

Caso a comparação anterior não for igual a FF, então o programa vai para o endereço simbólico "novo ciclo", iniciando a impressão do 2º caractere da tarefa.

. HLT

Quando a instrução anterior encontrar igualdade, ou seja, o código da tabela for igual a FF, então, terá terminada a impressão do último caractere e consequentemente, o programa finalizará.

CAPÍTULO 6

INTERFACE

DE

TECLADO

ÉRICA

6.1 - Instrução

O interface de teclado é um módulo, que tem a finalidade de encaminhar os dados de um teclado alfanumérico, ao circuito do microcomputador APL-80.
Logicamente, o circuito de interface de teclado possui 2 conectores, sendo interconectados na impressora e no microcomputador APL-80.
Escolhemos este tipo de "entrada" para ser ligado ao APL-80, exatamente para diferenciar do dispositivo de saída, descrito no capítulo anterior (interface de impressora).
Dispositivos de entrada, são aqueles que introduzem sinais ou dados no microcomputador, portanto um teclado chave, fotocélulas, sensores diversos, desde que devidamente adaptados podem ser ligados no microcomputador.
A adaptação dos diversos circuitos de entrada ao microcomputador, é chamado de interface de entrada. No nosso caso, o interface de teclado é o interface de entrada.
O teclado é um conjunto de chaves ligadas a circuitos eletrônicos, que quando acionadas geram códigos binários, que representam a tecla acionada no teclado. Convém notar que, o código de uma letra de uma tecla, é o mesmo necessário para imprimir esta mesma letra na impressora. Portanto, se interligarmos os dois programas de impressão e leitura de teclado, e possível imprimir o caractere da tecla acionada, diretamente na impressora.
Como nosso objetivo é extremamente didático, apenas descrevemos o princípio básico de funcionamento do interface, cabendo ao leitor, adaptá-lo para os diversos tipos comerciais existentes.

6.2 Circuito de Interface

Existem diversos tipos de teclado, porém entre eles estão divididos em 2 grupos: os com circuitos lógicos/eletrônicos e os puramente de contatos elétricos na primeira categoria, as teclas são pesquisadas por circuitos e já "soltam" na saída, o sinal codificado de cada tecla, para ser diretamente aproveitado pelo microcomputador.
Na segunda categoria, os teclados necessitam de circuitos de interface mais sofisticados, com a finalidade de gerarem os sinais apropriados para o microcomputador.
A fim de simplificar o circuitos de interface de teclado para utilizar com o microcomputador APL-80, lançamos mão de um teclado completo, com toda mecânica e eletrônica necessária.
O teclado escolhido foi o PD-52 fabricado pela digiponto, porém outros equivalentes poderão ser utilizados para tal propósito.
O circuito de interface utilizado para controlar o teclado, pode ser observado através da figura 6.1.

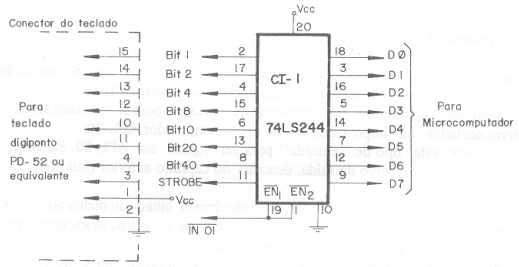


Figura 6.1 Circuito de Interface do Teclado
Obs: O Sinal IN01 é Retirado do Interface da Impressora

Para maior facilidade de estudo, considere o teclado apenas pelo seu conector na parte esquerda da figura. Isto é, as linhas bit 1 a bit 40, o sinal strobe e finalmente a alimentação VCC e terra.
Uma vez alimentado o circuito do teclado através dos pinos 1 e 2 do conector, dá-se início à varredura das teclas. Isto provoca a oscilação das linhas bit 1 a bit 40.
Quando uma tecla é acionada, cessam as oscilações das linhas bit 1 a bit 40, aparecendo nestas respectivas linhas, o código da tecla acionada. Simultaneamente, o sinal strobe, vai a nível 1 com o objetivo de indicar ao circuito de interface que uma tecla foi acionada, e o seu respectivo código encontra-se disponível.
Quando a tecla é solta, o sinal strobe vai a nível zero e as oscilações nas linhas bit 1 a bit 40, iniciam-se novamente.

Vejamos agora, como funciona o interface do teclado propriamente dito.
Para evitar a utilização de outra placa de interface, utilizamos a mesma placa de interface da impressora, e nela introduzimos apenas mais um circuito integrado (IC-1), portanto aproveitamos o decodificador de endereço do circuito de interface da impressora, e retiramos o sinal IN01, economizando assim a repetição de circuitos.
O circuito integrado IC1, é composto por um conjunto de tri-states, capazes de isolar o teclado do microcomputador APL-80.
Somente no tempo que for gerado pelo microcomputador o sinal IN-01, o circuito integrado IC1 transfere as informações do teclado para o microcomputador. Este procedimento é executado sob controle do programa milhares de vezes por segundo, com o objetivo de pesquisar a validade do sinal strobe.

6.3 Software:

Da mesma forma que o circuito de interface do teclado, o software é extremamente simples.
O princípio básico do funcionamento do software do teclado, é pesquisar continuamente o sinal strobe (linha D7), até o instante que for a nível 1. Quando isto ocorrer, o programa entra numa segunda fase, com o intuito de ler o código da tecla acionada, e armazená-lo no endereço 2100.
Uma vez o código da tecla acionada armazenado no endereço 2100, é possível examiná-lo, apenas lendo esse respectivo endereço, com o auxílio do programa monitor e a tecla "EX".

Na figura 6.2, você pode observar, um programa capaz de ler o teclado somente quando uma tecla for acionada, e em seguida, guardá-lo no endereço 2100.

ENDEREÇO SIMBÓLICO	ENDER. FÍSICO	CÓDIGO OBJETO	ASSEMBLY	COMENTÁRIOS
teste	2000	DB01	IN A,(01)	Lê canal do teclado
	2002	E680	AND A,80	Isola linha strobe
	2004	FE80	CP 80	Compara se strobe é = 1
	2006	C20020	JP NZ (teste)	Se strobe = 0 vai para teste
	2009	DB01	IN A, (01)	Lê canal do teclado
	200B	320021	LD (2100),A	Armazena dados no End. 2100
	200E	76	HLT	Fim

Tabela 6.2 Programa Destinado a Ler o Código do Teclado e Guardá-lo no Endereço de Memória 2100

Este programa utiliza os endereços 2000 a 200E, porém pode ser gravado em qualquer área de memória.
Vejamos agora, seu funcionamento na sequência das instruções.

. IN A,(01)
Esta instrução, tem a finalidade de ler o teclado através do circuito de interface. Ela é responsável pela geração do sinal IN01.
Quando executada, lê as linhas bit 1 a bit 40 e o sinal strobe, colocando-os no acumulador.

. AND A,80
Isola apenas a linha D7 dentro do acumulador, referente ao sinal strobe.
No final da execucão desta instrução, se a linha strobe estiver a nível 1, o resultado será 80, caso contrário, o resultado será 00.

. CP 80
O conteúdo do acumulador é comparado com 80. Se for igual a 80, então ligará o flag zero, caso contrário, o flag "não zero" "NZ" será ligado.

. JP NZ (teste)
A instrução "JP NZ (teste)" salta para o endereço simbólico teste se o flag "não zero" estiver ligado, caso contrário, o programa continua na próxima instrução.
Em resumo: Se a linha strobe estiver ligada, o programa continua na próxima instrução
para ler o código, caso contrário, o programa volta para a primeira linha para testar
novamente a linha strobe.

. IN A,(01)
Esta instrução lê todas as linhas de dados, formatando no acumulador, o código da tecla acionada.
Convém lembrar que a linha strobe, também está incluída no código e deve ser subtraída para compor o código correto.

. LD (2100),A
Armazena no endereço 2100, o código da tecla acionada.

. HLT
Encerra o programa.

APÊNDICES

A, B, C, D

ÉRICA

APÊNDICE - A

POTÊNCIAS DE 16

16^0	1
16^1	16
16^2	256
16^3	4.096
16^4	65.536
16^5	1.048.576
16^6	16.777.216
16^7	268.435.456
16^8	4.294.967.296

APÊNDICE - B

POTÊNCIAS DE 2

2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1.024
2^{11}	2.048
2^{12}	4.096
2^{13}	8.192
2^{14}	16.384
2^{15}	32.768
2^{16}	65.536
2^{17}	131.072
2^{18}	262.144
2^{19}	524.288
2^{20}	1.048.576
2^{21}	2.097.152
2^{22}	4.194.304
2^{23}	8.388.608
2^{24}	16.777.216
2^{25}	33.554.432
2^{26}	67.108.864
2^{27}	134.217.728
2^{28}	268.435.456
2^{29}	536.870.912
2^{30}	1.073.741.824
2^{31}	2.147.483.648

APÊNDICE - C

TABELA DE SALTOS RELATIVOS PARA FRENTE

MSD \ LSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

TABELA DE SALTOS RELATIVOS PARA TRÁS

MSD \ LSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	128	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113
9	112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97
A	96	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81
B	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65
C	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49
D	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
E	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
F	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

SOURCE STATEMENT	OBJ CODE	SOURCE STATEMENT	OBJ CODE	SOURCE STATEMENT	OBJ CODE	SOURCE STATEMENT	OBJ CODE	SOURCE STATEMENT	OBJ CODE
ADC A,(HL)	8E	BIT 2,(HL)	CB56	CP (HL)	BE	IND	EDAA	LD B,(IY+d)	DD4605
ADC A,(IX+d)	DD8E05	BIT 2,(IY+d)	FDCB0556	CP (IX+d)	DDBE05	INDR	EDBA	LD B,(IY+d)	FD4605
ADC A,(IY+d)	FD8E05	BIT 2,A	CB57	CP (IY+d)	FDBE05	INI	EDA2	LD B,A	47
ADC A,A	8F	BIT 2,B	CB50	CP A	BF	INIR	EDB2	LD B,B	40
ADC A,B	88	BIT 2,C	CB51	CP B	B8	JP aa	C3DA05	LD B,C	41
ADC A,C	89	BIT 2,D	CB52	CP C	B9	JP (HL)	E9	LD B,D	42
ADC A,D	8A	BIT 2,E	CB53	CP D	BA	JP (IX)	DDE9	LD B,E	43
ADC A,E	8B	BIT 2,H	CB54	CP E	BB	JP (IY)	FDE9	LD B,H	44
ADC A,H	8C	BIT 2,L	CB55	CP H	BC	JP C,aa	DADA05	LD B,L	45
ADC A,L	8D	BIT 3,(HL)	CB5E	CP L	BD	JP M,aa	FADA05	LD B,n	0620
ADC A,n	CE20	BIT 3,(IX+d)	DDCB055E	CP n	FE20	JP NC,aa	D2DA05	LD BC,(aa)	ED4BDA05
ADC HL,BC	ED4A	BIT 3,(IY+d)	FDCB055E	CPD	ED A9	JP NZ,aa	C2DA05	LD BC,nn	01DA05
ADC HL,DE	ED5A	BIT 3,A	CB5F	CPDR	ED B9	JP P,aa	F2DA05	LD C,(HL)	4E
ADC HL,HL	ED6A	BIT 3,B	CB58	CPI	ED A1	JP PE,aa	EADA05	LD C,(IX+d)	DD4E05
ADC HL,SP	ED7A	BIT 3,C	CB59	CPIR	ED B1	JP PO,aa	E2DA05	LD C,(IY+d)	FD4E05
ADD A,(HL)	86	BIT 3,D	CB5A	CPL	2F	JP Z,aa	CADA05	LD C,A	4F
ADD A,(IX+d)	DD8605	BIT 3,E	CB5B	DAA	27	JR C,e	3826	LD C,B	48
ADD A,(IY+d)	FD8605	BIT 3,H	CB5C	DEC (HL)	35	JR NC,e	3021E	LD C,C	49
ADD A,A	87	BIT 3,L	CB5D	DEC (IX+d)	DD3505	JR NZ,e	2021E	LD C,D	4A
ADD A,B	80	BIT 4,(HL)	CB66	DEC (IY+d)	FD3505	JR Z,e	2821E	LD C,E	4B
ADD A,C	81	BIT 4,(IX+d)	DDCB0566	DEC A	3D	JR e	1826	LD C,H	4C
ADD A,D	82	BIT 4,(IY+d)	FDCB0566	DEC B	05	LD (aa),A	32	LD C,L	4D
ADD A,E	83	BIT 4,A	CB67	DEC BC	0B	LD (aa),HL	2200	LD C,n	0E20
ADD A,H	84	BIT 4,B	CB60	DEC C	0D	LD (aa),IX	DD2200	LD D,(HL)	56
ADD A,L	85	BIT 4,C	CB61	DEC D	15	LD (aa),IY	FD2200	LD D,(IX+d)	DD5605
ADD A,n	C620	BIT 4,D	CB62	DEC DE	1B	LD (aa),SP	ED7300	LD D,(IY+d)	FD5605
ADD HL,BC	09	BIT 4,E	CB63	DEC E	1D	LD (BC),A	72	LD D,A	57
ADD HL,DE	19	BIT 4,H	CB64	DEC H	25	LD (DE),A	73	LD D,B	50
ADD HL,HL	29	BIT 4,L	CB65	DEC HL	2B	LD (HL),A	74	LD D,C	51
ADD HL,SP	39	BIT 5,(HL)	CB6E	DEC IX	DD2B	LD (HL),B	75	LD D,D	52
ADD IX,BC	DD09	BIT 5,(IX+d)	DDCB056E	DEC IY	FD2B	LD (HL),C	3620	LD D,E	53
ADD IX,DE	DD19	BIT 5,(IY+d)	FDCB056E	DEC L	2D	LD (HL),D	0070D5	LD D,H	54
ADD IX,IX	DD29	BIT 5,A	CB6F	DEC SP	3B	LD (HL),E	0070D5	LD D,L	55
ADD IX,SP	DD39	BIT 5,B	CB68	DI	F3	LD (HL),H	0070D5	LD D,n	1620
ADD IY,BC	FD09	BIT 5,C	CB69	DJNZ e	10[E	LD (HL),L	0070D5	LD DE,(aa)	ED5BDA05
ADD IY,DE	FD19	BIT 5,D	CB6A	EI	FB	LD (IX+d),A	DD7705	LD DE,nn	1100D5
ADD IY,IY	FD29	BIT 5,E	CB6B	EX (SP),HL	E3	LD (IX+d),B	DD7005	LD E,(HL)	5E
ADD IY,SP	FD39	BIT 5,H	CB6C	EX (SP),IX	DDE3	LD (IX+d),C	DD7105	LD E,(IX+d)	DD5E05
AND (HL)	A6	BIT 5,L	CB6D	EX (SP),IY	FDE3	LD (IX+d),n	DD360520	LD E,(IY+d)	FD5E05
AND (IX+d)	DDA605	BIT 6,(HL)	CB76	EX AF,AF'	08	LD (IY+d),A	FD7705	LD E,A	5F
AND (IY+d)	FDA605	BIT 6,(IX+d)	DDCB0576	EX DE,HL	EB	LD (IY+d),B	FD7005	LD E,B	58
AND A	A7	BIT 6,(IY+d)	FDCB0576	EXX	D9	LD (IY+d),C	FD7105	LD E,C	59
AND B	A0	BIT 6,A	CB77	HALT	76	LD (IY+d),D	FD7205	LD E,D	5A
AND C	A1	BIT 6,B	CB70	IM 0	ED46	LD (IY+d),E	FD7305	LD E,E	5B
AND D	A2	BIT 6,C	CB71	IM 1	ED56	LD (IY+d),H	FD7405	LD E,H	5C
AND E	A3	BIT 6,D	CB72	IM 2	ED5E	LD (IY+d),L	FD7505	LD E,L	5D
AND H	A4	BIT 6,E	CB73	IN A,(C)	ED78	LD (IY+d),n	FD360520	LD E,n	1E20
AND L	A5	BIT 6,H	CB74	IN B,(C)	ED40	LD (nn),A	32DA05	LD H,(HL)	66
AND n	E620	BIT 6,L	CB75	IN C,(C)	ED48	LD (nn),BC	ED43DA05	LD H,(IX+d)	DD6605
BIT 0,(HL)	CB46	BIT 7,(HL)	CB7E	IN D,(C)	ED50	LD (nn),DE	ED53DA05	LD H,(IY+d)	FD6605
BIT 0,(IX+d)	DDCB054E	BIT 7,(IX+d)	DDCB057E	IN E,(C)	ED58	LD (nn),HL	2200D5	LD H,A	67
BIT 0,(IY+d)	FDCB054E	BIT 7,(IY+d)	FDCB057E	IN H,(C)	ED60	LD (nn),IX	DD2200D5	LD H,B	60
BIT 0,A	CB47	BIT 7,A	CB7F	IN L,(C)	ED68	LD (nn),IY	FD2200D5	LD H,C	61
BIT 0,B	CB40	BIT 7,B	CB78	INC (HL)	34	LD (nn),SP	ED73DA05	LD H,D	62
BIT 0,C	CB41	BIT 7,C	CB79	INC (IX+d)	DD3405	LD A,(BC)	0A	LD H,E	63
BIT 0,D	CB42	BIT 7,D	CB7A	INC (IY+d)	FD3405	LD A,(DE)	1A	LD H,H	64
BIT 0,E	CB43	BIT 7,E	CB7B	INC A	3C	LD A,(HL)	7E	LD H,L	65
BIT 0,H	CB44	BIT 7,H	CB7C	INC B	04	LD A,(IX+d)	DD7E05	LD H,n	2620
BIT 0,L	CB45	BIT 7,L	CB7D	INC BC	03	LD A,(IY+d)	FD7E05	LD HL,(aa)	2ADA05
BIT 1,(HL)	CB4E	BIT [IX+d)	DDCB057E	INC C	0C	LD A,(aa)	3ADA05	LD HL,nn	2100D5
BIT 1,(IX+d)	DDCB054E	CALL C,nn	DCDA05	INC D	14	LD A,I	ED57	LD I,A	ED47
BIT 1,(IY+d)	FDCB054E	CALL M,nn	FCDA05	INC DE	13	LD A,n	7E	LD IX,(aa)	DD2ADA05
BIT 1,A	CB4F	CALL NC,nn	D4DA05	INC E	1C	LD A,R	3E20	LD IX,nn	DD2100D5
BIT 1,B	CB48	CALL NZ,nn	C4DA05	INC H	24	LD A,(BC)	0A	LD IY,(aa)	FD2ADA05
BIT 1,C	CB49	CALL P,nn	F4DA05	INC HL	23	LD A,(DE)	1A	LD IY,nn	FD2100D5
BIT 1,D	CB4A	CALL PE,nn	ECDA05	INC IX	DD23	LD A,(HL)	7E	LD (HL),E	5E
BIT 1,E	CB4B	CALL PO,nn	E4DA05	INC IY	FD23	LD A,I	ED57	LD (IX+d),E	DD5E05
BIT 1,H	CB4C	CALL Z,nn	CCDA05	INC L	2C	LD A,R	3E20	LD (IY+d),E	FD5E05
BIT 1,L	CB4D	CALL nn	CDDA05	INC SP	33	LD B,(HL)	46	LD L,A	6F
				IN A,(n)	DB20			LD L,B	68
								LD L,C	69

SOURCE STATEMENT	OBJ CODE	SOURCE STATEMENT	OBJ CODE	SOURCE STATEMENT	OBJ CODE	SOURCE STATEMENT	OBJ CODE	SOURCE STATEMENT	OBJ CODE
LD L,D	6A	RES 1,A	CB8D	RETI	ED4D	SET 0,(HL)	CBC6	SET 0,IX+d	DDCB
LD L,E	6B	RES 2,(HL)	CB96	RETN	ED45	SET 0,IX+d	DDCB	SET 0,IY+d	FDCB
LD L,H	6C	RES 2,IX+d	DDCB0596	RL (HL)	CB16	SET 0,IY+d	FDCB0506	SET 7,A	CBF7
LD L,L	6D	RES 2,IY+d	FDCB0596	RL IX+d	DDCB0516	SET 0,A	CBC7	SET 7,B	CBF8
LD L,n	2E20	RES 2,A	CB97	RL IY+d	FDCB0516	SET 0,B	CBC0	SET 7,C	CBF9
LD R,A	ED4F	RES 2,B	CB90	RL A	CB17	SET 0,C	CBC1	SET 7,D	CBFA
LD SP,(nn)	ED7B6A0B	RES 2,C	CB91	RL B	CB10	SET 0,D	CBC2	SET 7,E	CBFB
LD SP,HL	F9	RES 2,D	CB92	RL C	CB11	SET 0,E	CBC3	SET 7,H	CBFC
LD SP,IX	DDF9	RES 2,E	CB93	RL D	CB12	SET 0,H	CBC4	SET 7,L	CBFD
LD SP,IY	FDF9	RES 2,H	CB94	RL E	CB13	SET 0,L	CBC5	SLA (HL)	CB26
LD SP,nn	318A05	RES 2,L	CB95	RL H	CB14	SET 1,(HL)	CBCE	SLA IX+d	DDCB
LDD	EDA8	RES 3,(HL)	CB9E	RL L	CB15	SET 1,IX+d	DDCB0596	SLA IY+d	FDCB
LDDR	EDB8	RES 3,IX+d	DDCB059E	RLA	17	SET 1,IY+d	FDCB0506	SLA A	CB27
LDI	EDA0	RES 3,IY+d	FDCB059E	RLC (HL)	CB06	SET 1,A	CBCF	SLA B	CB20
LDIR	EDB0	RES 3,A	CB9F	RLC IX+d	DDCB0506	SET 1,B	CBC8	SLA C	CB21
NEG	ED44	RES 3,B	CB98	RLC IY+d	FDCB0506	SET 1,C	CBC9	SLA D	CB22
NOP	00	RES 3,C	CB99	RLC A	CB07	SET 1,D	CBCA	SLA E	CB23
OR (HL)	B6	RES 3,D	CB9A	RLC B	CB00	SET 1,E	CBCB	SLA H	CB24
OR IX+d	DDB605	RES 3,E	CB9B	RLC C	CB01	SET 1,H	CBCC	SLA L	CB25
OR IY+d	FDB605	RES 3,H	CB9C	RLC D	CB02	SET 1,L	CBCD	SRA (HL)	CB2E
OR A	B7	RES 3,L	CB9D	RLC E	CB03	SET 2,(HL)	CBD6	SRA IX+d	DDCB
OR B	B0	RES 4,(HL)	CBA6	RLC H	CB04	SET 2,IX+d	DDCB0596	SRA IY+d	FDCB
OR C	B1	RES 4,IX+d	DDCB05A6	RLC L	CB05	SET 2,IY+d	FDCB0506	SRA A	CB2F
OR D	B2	RES 4,IY+d	FDCB05A6	RLCA	07	SET 2,A	CBD7	SRA B	CB28
OR E	B3	RES 4,A	CBA7	RLD	ED6F	SET 2,B	CBD0	SRA C	CB29
OR H	B4	RES 4,B	CBA0	RR (HL)	CB1E	SET 2,C	CBD1	SRA D	CB2A
OR L	B5	RES 4,C	CBA1	RR IX+d	DDCB051E	SET 2,D	CBD2	SRA E	CB2B
OR n	F620	RES 4,D	CBA2	RR IY+d	FDCB051E	SET 2,E	CBD3	SRA H	CB2C
OTDR	EDBB	RES 4,E	CBA3	RR A	CB1F	SET 2,H	CBD4	SRA L	CB2D
OTIR	EDB3	RES 4,H	CBA4	RR B	CB18	SET 2,L	CBD5	SRL (HL)	CB3E
OUT (C),A	ED79	RES 4,L	CBA5	RR C	CB19	SET 3,(HL)	CBDE	SRL IX+d	DDCB
OUT (C),B	ED41	RES 5,(HL)	CBAE	RR D	CB1A	SET 3,IX+d	DDCB059E	SRL IY+d	FDCB
OUT (C),C	ED49	RES 5,IX+d	DDCB05AE	RR E	CB1B	SET 3,IY+d	FDCB059E	SRL A	CB3F
OUT (C),E	ED59	RES 5,IY+d	FDCB05AE	RR H	CB1C	SET 3,A	CBDF	SRL B	CB38
OUT (C),H	ED61	RES 5,A	CBAF	RR L	CB1D	SET 3,B	CBD8	SRL C	CB39
OUT (C),L	ED69	RES 5,B	CBA8	RRA	1F	SET 3,C	CBD9	SRL D	CB3A
OUT (n),A	D320	RES 5,C	CBA9	RRC (HL)	CB0E	SET 3,D	CBDA	SRL E	CB3B
OUTD	EDAB	RES 5,D	CBAA	RRC IX+d	DDCB050E	SET 3,E	CBDB	SRL H	CB3C
OUTI	EDA3	RES 5,E	CBAB	RRC IY+d	FDCB050E	SET 3,H	CBDC	SRL L	CB3D
POP AF	F1	RES 5,H	CBAC	RRC A	CB0F	SET 3,L	CBDD	SUB (HL)	96
POP BC	C1	RES 5,L	CBAD	RRC B	CB08	SET 4,(HL)	CBE6	SUB IX+d	DD9605
POP DE	D1	RES 6,(HL)	CBB6	RRC C	CB09	SET 4,IX+d	DDCB0596	SUB IY+d	FD9605
POP HL	E1	RES 6,IX+d	DDCB05B6	RRC D	CB0A	SET 4,IY+d	FDCB0596	SUB A	97
POP IX	DDE1	RES 6,IY+d	FDCB05B6	RRC E	CB0B	SET 4,A	CBE7	SUB B	90
POP IY	FDE1	RES 6,A	CBB7	RRC H	CB0C	SET 4,B	CBE0	SUB C	91
PUSH AF	F5	RES 6,B	CBB0	RRC L	CB0D	SET 4,C	CBE1	SUB D	92
PUSH BC	C5	RES 6,C	CBB1	RRCA	0F	SET 4,D	CBE2	SUB E	93
PUSH DE	D5	RES 6,D	CBB2	RRD	ED67	SET 4,E	CBE3	SUB H	94
PUSH HL	E5	RES 6,E	CBB3	RST 00H	C7	SET 4,H	CBE4	SUB L	95
PUSH IX	DDE5	RES 6,H	CBB4	RST 08H	CF	SET 4,L	CBE5	SUB n	D620
PUSH IY	FDE5	RES 6,L	CBB5	RST 10H	D7	SET 5,(HL)	CBEE	XOR (HL)	AE
RES 0,(HL)	CB86	RES 7,(HL)	CBBE	RST 18H	DF	SET 5,IX+d	DDCB05EE	XOR IX+d	DDAE05
RES 0,IX+d	DDCB0586	RES 7,IX+d	DDCB05BE	RST 20H	E7	SET 5,IY+d	FDCB05EE	XOR IY+d	FDAE05
RES 0,IY+d	FDCB0586	RES 7,IY+d	FDCB05BE	RST 28H	EF	SET 5,A	CBEF	XOR A	AF
RES 0,A	CB87	RES 7,A	CBBF	RST 30H	F7	SET 5,B	CBE8	XOR B	A8
RES 0,B	CB80	RES 7,B	CBB8	RST 38H	FF	SET 5,C	CBE9	XOR C	A9
RES 0,C	CB81	RES 7,C	CBB9	SBC A,(HL)	DE20	SET 5,D	CBEA	XOR D	AA
RES 0,D	CB82	RES 7,D	CBBA	SBC A,IX+d	DD9E0E	SET 5,E	CBEB	XOR E	AB
RES 0,E	CB83	RES 7,E	CBBB	SBC A,IY+d	FD9E05	SET 5,H	CBEC	XOR H	AC
RES 0,H	CB84	RES 7,H	CBBC	SBC A,A	9F	SET 5,L	CBED	XOR L	AD
RES 0,L	CB85	RES 7,L	CBBD	SBC A,B	98	SET 6,(HL)	CBF6	XOR n	EE20
RES 1,(HL)	CB8E	RET	C9	SBC A,C	99	SET 6,IX+d	DDCB05F6		
RES 1,IX+d	DDCB058E	RET C	D8	SBC A,D	9A	SET 6,IY+d	FDCB05F6		
RES 1,IY+d	FDCB058E	RET M	F8	SBC A,E	9B	SET 6,A	CBF7		
RES 1,A	CB8F	RET NC	D0	SBC A,H	9C	SET 6,B	CBF0		
RES 1,B	CB88	RET NZ	C0	SBC A,L	9D	SET 6,C	CBF1		
RES 1,C	CB89	RET P	F0	SBC HL,BC	ED42	SET 6,D	CBF2		
RES 1,D	CB8A	RET PE	E8	SBC HL,DE	ED52	SET 6,E	CBF3		
RES 1,E	CB8B	RET PO	E0	SBC HL,HL	ED62	SET 6,H	CBF4		
RES 1,H	CB8C	RET Z	C8	SBC HL,SP	ED72	SET 6,L	CBF5		
				SCF	37	SET 7,(HL)	CBFE		

BIBLIOGRAFIA

1. How to program the Z80
   RODNAY ZAKS

2. Microcomputer Data Book
   MOSTEK

3. RACIMEC S.A. Impressora ITA-2

Impressão e Acabamento
GRÁFICA E EDITORA FCA
com filmes fornecidos pelo editor
AV. HUMBERTO DE ALENCAR CASTELO BRANCO, 3972 - TEL.: 419-0200
SÃO BERNARDO DO CAMPO - CEP 09700 - SP

OUTRAS PUBLICAÇÕES

- INFORMÁTICA
- ELETRÔNICA
- TELECOMUNICAÇÕES
- PROCESSAMENTO DE DADOS
- MICROPROCESSADORES
- ELETROTÉCNICA
- MECÂNICA
- CIÊNCIAS EXATAS

ÉRICA

A MARCA DO CONTEÚDO COM QUALIDADE
RUA JARINÚ, 594 - TATUAPÉ - CEP 03306
CX. P. 15617 - TEL.: (011) 294-8686 - S.P.